Marylou, Heather, Jenn, moim rodzicom i moim nauczycielom

Jim

Debbie, mojemu ojcu, pamięci mojej matki i moim dzieciom Elisa, Lori i Karin

Andy

Jenni, moim rodzicom i moim nauczycielom

Steve

Mojej rodzinie, mojemu nauczycielowi Robowi Kirby i pamięci mojego ojca

John

Oraz wszystkim naszym studentom.

Thomasowi Carlyle, który po dowiedzeniu się, że służąca przez nieuwagę spaliła właśnie ukończony rękopis jego pracy *The French Revolution*, zaczął pisanie od początku

Dick

Z angielskiego przełożył
prof. dr hab. inż. Jan Zabrodzki

Instytut Informatyki
Politechniki Warszawskiej

J. D. Foley
A. van Dam
S. K. Feiner
J. F. Hughes
R. L. Phillips

WPROWADZENIE DO GRAFIKI KOMPUTEROWEJ

Wydawnictwa Naukowo-Techniczne
Warszawa

Dane o oryginale

James D. Foley
Georgia Institute of Technology

Andries van Dam
Brown University

Steven K. Feiner
Columbia University

John F. Hughes
Brown University

Richard L. Phillips
Los Alamos National Laboratory and The University of Michigan

INTRODUCTION TO COMPUTER GRAPHICS

Copyright © 1994, 1990 by Addison-Wesley Publishing Company, Inc.

Tłumaczenie, wydanie i rozpowszechnianie za zgodą
Addison-Wesley Publishing Company, Inc.
Reading Massachusetts, USA

Redaktor
Zofia Dackiewicz

Przygotowanie do druku
Ewa Eckhardt, Anna Szeląg

Opracowanie techniczne
Marta Jeczeń

Okładkę i strony tytułowe projektowała
Ewa Bohusz-Rubaszewska

Copyright © for the Polish edition by Wydawnictwa Naukowo-Techniczne
Warszawa 1995

Utwór w całości ani we fragmentach nie może być powielany ani rozpowszechniany za pomocą urządzeń elektronicznych, mechanicznych, kopiujących, nagrywających i innych bez pisemnej zgody posiadacza praw autorskich.

All rights reserved
Printed in Poland

ISBN 83-204-1840-2

Wstęp

Książka ta jest adaptacją drugiego wydania książki *Computer Graphics: Principles and Practice* (*CGPP*), napisanej przez Foleya, van Dama, Feinera i Hughesa. Książka *Wprowadzenie do grafiki komputerowej* powstała w wyniku skrócenia i zmodyfikowania tej obszernej pracy o charakterze podręcznika i pracy źródłowej, w celu przystosowania do potrzeb różnych kursów i różnych wymagań profesjonalnych. Chociaż ma ona połowę objętości swojego pierwowzoru, nie jest to po prostu jej skrócona wersja. Książka zawiera wiele nowego materiału, a w niektórych miejscach materiał jest przedstawiony w inny sposób – wszystkie te zmiany wprowadzono z myślą o potrzebach odbiorców.

Książka jest tak opracowana, żeby mogła być wykorzystywana na jedno- albo dwusemestralnych wykładach z grafiki komputerowej w dowolnej wyższej uczelni przy założeniu tylko niewielkiego przygotowania z zakresu matematyki na jednosemestralnym wykładzie w pomaturalnych szkołach dwuletnich. *Wprowadzenie do grafiki komputerowej* jest również idealną książką dla profesjonalistów, którzy chcą poznać podstawy tej dynamicznej i fascynującej dziedziny po to, żeby zacząć ją uprawiać, albo po prostu po to, żeby zrozumieć różnorodne zastosowania grafiki komputerowej.

Istotne zalety książki *Wprowadzenie do grafiki komputerowej* są następujące:

- Językiem programowania używanym w książce zarówno we fragmentach pseudoprogramów, jak i w kompletnych działających programach jest nowoczesny język ANSI C. Korzystanie z języka C jest

w zgodzie z obecną praktyką zarówno w nauczaniu, jak i w pracy zawodowej, zwłaszcza w grafice komputerowej.

- Bezpośrednią zaletą korzystania w książce z języka C jest pełna zgodność między typami danych i funkcjami kodu używanymi w książce z odpowiednikami w pakietach programowych SRGP i SPHIGS.
- Wspomniany pakiet SPHIGS został wzbogacony o nowe cechy, takie jak wielokrotne źródła światła, ulepszony rendering i ulepszona korelacja wskazywania dla lepszej współpracy interakcyjnej.
- Książka zawiera sporo rozwiązanych przykładów. Przykłady te są zamieszczone w tych rozdziałach, w których najlepiej pomagają wyjaśniać trudne pojęcia. Jeden z takich przykładów to kompletny działający program dla interakcyjnego definiowania parametrycznych krzywych Béziera trzeciego stopnia.
- Pokazano rolę grafiki komputerowej w rozwijającej się dziedzinie multimediów – podano kilka przykładów wraz z rysunkami oraz odwołania do literatury.
- Rozdział o przekształceniach geometrycznych zawiera punkt o niezbędnych wiadomościach z matematyki, które są potrzebne czytelnikowi do zrozumienia i wykorzystania wszystkich późniejszych partii materiału w książce.

Możliwe programy wykładów

Czytelnik może korzystać z tej książki na wiele różnych sposobów. Niżej proponujemy kilka możliwych wariantów; istnieje oczywiście pełna dowolność w wyborze odpowiedniego wariantu odpowiadającego potrzebom czytelnika. Kolejność studiowania również może być zmieniana. Na przykład materiał związany ze sprzętem może być umieszczony w programie wcześniej albo później, niż by to wynikało z zamieszczenia materiału w rozdz. 4.

Minimalny jednosemestralny wykład poświęcony grafice 2D. Jeżeli celem jest przekazanie studentom dobrego, choć niekoniecznie pełnego, przeglądu elementów grafiki 2D, to taki wykład może być odpowiedni dla studentów w dwuletniej szkole pomaturalnej albo w wyższej uczelni.

Rozdział	Punkty
1	Wszystkie
2	2.1 – 2.2
3	3.1 – 3.3
4	4.1 – 4.3 i 4.5

Rozdział	Punkty
5	5.1 (jeżeli jest potrzebny), 5.2–5.3, 5.4
6	6.1–6.3, 6.4.1, 6.4.2
8	Wszystkie
9	9.1, 9.2.1–9.2.3
11	11.1, 11.2
12	Wybrane fragmenty ilustrujące zaawansowane możliwości

Jednosemestralny wykład poświęcony przeglądowi grafiki 2D i 3D. Ten program obejmuje solidne podstawy z zakresu grafiki i jest przeznaczony dla czytelników o dobrym przygotowaniu matematycznym.

Rozdział	Punkty
1	Wszystkie
2	Wszystkie
3	3.1–3.5, 3.10, 3.12, 3.15
4	4.1, 4.2, 4.3 i 4.5
5	5.1 (jeżeli jest potrzebny), 5.2–5.5, 5.7, 5.8
6	6.1–6.5, 6.6 (oprócz 6.6.4), 6.7
7	7.1–7.4, 7.10
8	Wszystkie
9	9.1, 9.2.1–9.2.3, 9.3.1, 9.3.2
11	Wszystkie
12	Wszystkie
13	13.1, 13.2
14	14.1, 14.2

Dwusemestralny wykład obejmujący grafikę 2D i 3D, modelowanie i rendering. Wszystkie rozdziały (być może z pominięciem wybranych elementów z rozdz. 9 i 10) plus wybrane elementy z *CGPP*.

Podziękowania

Należy stwierdzić, że wszyscy autorzy *CGPP* brali w pewnym stopniu udział w przygotowaniu tej książki, lecz tylko ja ponoszę wszelką odpowiedzialność za nowe błędy, które pojawiły się w procesie adaptacji.

David Sklar był autorem *CGPP* i wiele z napisanego przez niego materiału w tamtej książce pozostało tutaj w rozdz. 2 i 7. Pomógł mi on również w przeniesieniu elektronicznej wersji kodu komputerowego i rysunków z pierwotnej książki.

Mój redaktor Peter Gordon w czasie trwania tego projektu zawsze udzielał mądrych, wyważonych i na czasie rad, a mój kierownik produkcji Jim Rigney spędził wiele godzin ucząc mnie „chwytów zawodowych".

Wiele osób pomogło mi w różnych sprawach związanych z napisaniem tej książki. Wśród nich są: Yvonne Martinez, Chuck Hansen, Tom Rokicki, David Cortesi, Janick (J.) Bergeron, Henry McGilton, Greg Cockroft, Mike Hawley, Ed Catmull, Loren Carpenter, Harold Borkin, Alan Paeth i Jim White.

Specjalne podziękowania należą się Edowi Angelowi z The University of New Mexico oraz jego dzielnym studentom za beta – a właściwie alfa – testowanie pierwszej wersji książki jesienią 1992 r.

Wreszcie bez D.C. do niczego by nie doszło.

Santa Fe, N.M. *R.L.P.*

Spis treści

1. Wprowadzenie: grafika komputerowa 21

1.1. Przykłady zastosowań grafiki komputerowej 21

1.2. Krótka historia grafiki komputerowej 27

1.2.1. Wyprowadzanie informacji 29

1.2.2. Wprowadzanie informacji 33

1.2.3. Przenośność oprogramowania i standardy graficzne . . 33

1.3. Zalety grafiki interakcyjnej 36

1.4. Ogólny schemat grafiki interakcyjnej 37

1.4.1. Model zastosowania 38

1.4.2. Wyświetlanie modelu 39

1.4.3. Obsługa interakcji 41

Podsumowanie 42

Zadania 43

2. Pakiet SRGP 45

2.1. Rysowanie za pomocą pakietu SRGP 46

2.1.1. Określenie prymitywów graficznych 46

2.1.2. Atrybuty 52

2.1.3. Wypełniane prymitywy i ich atrybuty 55

2.1.4. Zapamiętywanie i odtwarzanie atrybutów 59

2.1.5 Tekst 59

2.2. Podstawy obsługi interakcji 62

2.2.1. Czynniki ludzkie 62

2.2.2. Logiczne urządzenia wejściowe 63
2.2.3. Próbkowanie a przetwarzanie sterowane zdarzeniami . 65
2.2.4. Tryb próbkowania 68
2.2.5. Tryb zdarzeń 69
2.2.6. Korelacja wskazywania przy obsłudze interakcji 74
2.2.7. Ustawianie stanu urządzenia i atrybutów 76
2.3. Cechy grafiki rastrowej 79
2.3.1. Kanwy 79
2.3.2. Prostokąty obcinające 82
2.3.3. Operacja SRGP_copyPixel 83
2.3.4. Tryb Write albo RasterOp 85
2.4. Ograniczenia SRGP 90
2.4.1. Układ współrzędnych zastosowania 90
2.4.2. Pamiętanie prymitywów dla celów ponownej specyfikacji 91
Podsumowanie 93
Zadania 95
Projekty programowe 97

3. Podstawowe algorytmy rysowania prymitywów 2D w grafice rastrowej 99

3.1. Przegląd 100
3.1.1. Wpływ architektury systemu wyświetlania 100
3.1.2. Programowy potok wyjściowy 104
3.2. Konwersja odcinków 105
3.2.1. Podstawowy algorytm przyrostowy 106
3.2.2. Rysowanie odcinka – algorytm z punktem środkowym 108
3.2.3. Dodatkowe problemy 115
3.3. Konwersja okręgów 118
3.3.1. Ośmiokrotna symetria 118
3.3.2. Rysowanie okręgu – algorytm z punktem środkowym . 119
3.4. Wypełnianie prostokątów 124
3.5. Wypełnianie wielokątów 126
3.5.1. Krawędzie poziome 129
3.5.2. Drzazgi 130
3.5.3. Spójność krawędziowa a algorytm konwersji 131
3.6. Wypełnianie wzorami 135
3.6.1. Wypełnianie wzorami przy konwersji wierszowej . . . 135

3.6.2. Wypełnianie wzorami bez wielokrotnej konwersji wierszowej 137
3.7. Pogrubione prymitywy 140
3.7.1. Powielanie pikseli 141
3.7.2. Ruchome pióro 142
3.8. Obcinanie w technice rastrowej 143
3.9. Obcinanie odcinków 145
3.9.1. Obcinanie punktów 145
3.9.2. Obcinanie odcinków na zasadzie rozwiązywania układu równań 146
3.9.3. Algorytm Cohena-Sutherlanda obcinania odcinków . . 147
3.9.4. Parametryczny algorytm obcinania odcinków 152
3.10. Obcinanie okręgów 157
3.11. Obcinanie wielokątów 158
3.11.1. Algorytm Sutherlanda-Hodgmana obcinania wielokąta 158
3.12. Generowanie znaków 162
3.12.1. Definiowanie i obcinanie znaków 162
3.12.2. Implementacja prymitywu Text Output 164
3.13. SRGP_copyPixel 166
3.14. Metody usuwania zakłóceń 167
3.14.1. Zwiększanie rozdzielczości 167
3.14.2. Bezwagowe próbkowanie powierzchni 168
3.14.3. Wagowe próbkowanie powierzchni 170
3.15. Zaawansowane rozwiązania 173
Podsumowanie 175
Zadania 175

4. Sprzęt dla potrzeb grafiki komputerowej 178

4.1. Metody tworzenia kopii trwałych 179
4.2. Metody wyświetlania 185
4.3. Rastrowe systemy wyświetlania 192
4.3.1. Prosty rastrowy system wyświetlania 193
4.3.2. Rastrowy system wyświetlania z zewnętrznym procesorem wyświetlania 197
4.3.3. Dodatkowe funkcje procesora wyświetlania 200
4.3.4. Rastrowy system wyświetlania z wbudowanym procesorem wyświetlania 203

4.4. Sterownik wyświetlania 204
4.4.1. Mieszanie sygnałów wizyjnych 206
4.5. Urządzenia wejściowe do współpracy z operatorem . . . 207
4.5.1. Lokalizatory 207
4.5.2. Klawiatury 211
4.5.3. Urządzenia do wprowadzania wartości 211
4.5.4. Urządzenia wybierające 211
4.6. Skanery obrazów 212
Zadania 214

5. Przekształcenia geometryczne 215

5.1. Podstawy matematyczne 215
5.1.1. Wektory i ich właściwości 216
5.1.2. Iloczyn skalarny 218
5.1.3. Właściwości iloczynu skalarnego 220
5.1.4. Macierze 220
5.1.5. Mnożenie macierzy 220
5.1.6. Wyznaczniki 221
5.1.7. Transpozycja macierzy 222
5.1.8. Odwracanie macierzy 222
5.2. Przekształcenia 2D 224
5.3. Współrzędne jednorodne i macierzowa reprezentacja przekształceń 2D 226
5.4. Składanie przekształceń 2D 232
5.5. Przekształcenie okna w pole wizualizacji 234
5.6. Efektywność 237
5.7. Reprezentacja macierzowa przekształceń 3D 238
5.8. Składanie przekształceń 3D 242
5.9. Przekształcenia jako zmiana układu współrzędnych . . . 247
Zadania 251

6. Rzutowanie w przestrzeni 3D 253

6.1. Syntetyczna kamera i kroki rzutowania 3D 253
6.2. Rzuty 256
6.2.1. Rzuty perspektywiczne 257
6.2.2. Rzuty równoległe 259
6.3. Specyfikowanie dowolnego rzutu 3D 262

6.4. Przykłady rzutowania 3D 267
6.4.1. Rzuty perspektywiczne 268
6.4.2. Rzuty równoległe 273
6.4.3. Skończone bryły widzenia 274
6.5. Płaskie rzuty geometryczne 274
6.6. Implementacja płaskich rzutów geometrycznych 278
6.6.1. Przypadek rzutu równoległego 280
6.6.2. Przypadek rzutu perspektywicznego 285
6.6.3. Obcinanie w 3D przez kanoniczną bryłę widzenia . . . 291
6.6.4. Obcinanie we współrzędnych jednorodnych 292
6.6.5. Odwzorowanie na pole wizualizacji 296
6.6.6. Podsumowanie problemów realizacyjnych 297
6.7. Układy współrzędnych 298
Zadania 300

7. Hierarchia obiektów i SPHIGS 303

7.1. Modelowanie geometryczne 305
7.1.1. Modele geometryczne 307
7.1.2. Hierarchia modeli geometrycznych 307
7.1.3. Zależność między modelem, programem użytkowym i systemem graficznym 311
7.2. Charakterystyka pakietów graficznych wykorzystujących tryb podtrzymywania 313
7.2.1. Główna pamięć struktury i jej zalety 313
7.2.2. Ograniczenia pakietów działających w trybie podtrzymania 314
7.3. Struktury dla definiowania i wyświetlania 315
7.3.1. Otwieranie i zamykanie struktur 316
7.3.2. Specyfikowanie prymitywów wyjściowych i ich atrybutów 317
7.3.3. Wysyłanie struktur w celu wyświetlenia 320
7.3.4. Rzutowanie 321
7.3.5. Zastosowania graficzne wykorzystające ekran na zasadzie zarządzania oknami 325
7.4. Przekształcenia modelowania 326
7.5. Hierarchiczne sieci struktur 331
7.5.1. Hierarchia dwupoziomowa 331
7.5.2. Prosta hierarchia trójpoziomowa 333
7.5.3. Konstruowanie robota metodą wstępującą 335
7.5.4. Interakcyjne programy modelowania 339

7.6. Składanie macierzy przy przeglądaniu w celu wyświetlenia 339
7.7. Obsługa atrybutów wyglądu w hierarchii 344
7.7.1. Reguły dziedziczenia 344
7.7.2. Atrybuty SPHIGS i tekst nie modyfikowane przez przekształcenia 348
7.8. Uaktualnianie ekranu i tryby renderingu 348
7.9. Edycja struktury sieci dla efektów dynamicznych 350
7.9.1. Dostęp do elementów za pomocą indeksów i etykiet 350
7.9.2. Operacje edycji wewnątrzstrukturalnej 351
7.9.3. Bloki kopii 353
7.9.4. Sterowanie automatyczną regeneracją obrazu na ekranie 354
7.10. Interakcja 355
7.10.1. Lokalizatory 355
7.10.2. Korelacja wskazywania 356
7.11. Zaawansowane problemy 363
7.11.1. Dodatkowe funkcje wyjściowe 364
7.11.2. Problemy implementacyjne 364
7.11.3. Optymalizacja wyświetlania modeli hierarchicznych . . 367
7.11.4. Ograniczenia modelowania hierarchicznego w PHIGS . 367
7.11.5. Alternatywne formy modelowania hierarchicznego . . 368
7.11.6. Inne (przemysłowe) standardy 368
Podsumowanie 370
Zadania 371

8. Urządzenia wejściowe, metody i zadania interakcji 373

8.1. Sprzęt dla interakcji 375
8.1.1. Lokalizatory 375
8.1.2. Klawiatury 377
8.1.3. Urządzenia do wprowadzania wartości 378
8.1.4. Urządzenia wybierające 378
8.1.5. Inne urządzenia 378
8.1.6. Urządzenia interakcyjne 3D 379
8.2. Podstawowe zadania interakcyjne 382
8.2.1. Zadanie interakcyjnego pozycjonowania 382
8.2.2. Wybór interakcyjny – zbiór wyborów o zmiennej wielkości 383

8.2.3. Wybór interakcyjny – zbiór wyborów o relatywnie stałej wielkości 387
8.2.4. Wybór interakcyjny metodą wprowadzania tekstu . . . 391
8.2.5. Wybór interakcyjny wartości 391
8.2.6. Zadania interakcyjne 3D 392
8.3. Złożone zadania interakcyjne 395
8.3.1. Pola dialogowe 395
8.3.2. Metody konstrukcyjne 395
8.3.3. Manipulacje dynamiczne 397
8.4. Narzędzia wspomagające metody interakcji 399
Podsumowanie 400
Zadania 400

9. Reprezentowanie krzywych i powierzchni 402

9.1. Siatki wielokątowe 404
9.1.1. Reprezentowanie siatek wielokątowych 405
9.1.2. Równania płaszczyzny 407
• 9.2. Parametryczne krzywe trzeciego stopnia 410
9.2.1. Podstawowe charakterystyki 412
9.2.2. Krzywe Hermite'a 416
• 9.2.3. Krzywe Béziera 421
9.2.4. Jednorodne nieułamkowe krzywe B-sklejane 427
9.2.5. Niejednorodne nieułamkowe krzywe B-sklejane 432
9.2.6. Niejednorodne ułamkowe segmenty wielomianowej krzywej trzeciego stopnia 435
9.2.7. Dopasowywanie krzywych do zbioru punktów 436
9.2.8. Porównanie krzywych trzeciego stopnia 436
9.3. Parametryczne powierzchnie bikubiczne 438
9.3.1. Powierzchnie Hermite'a 439
9.3.2. Powierzchnie Béziera 441
9.3.3. Powierzchnie B-sklejane 442
9.3.4. Normalne do powierzchni 443
9.3.5. Wyświetlanie powierzchni bikubicznych 444
9.4. Powierzchnie drugiego stopnia 446
9.5. Specjalizowane metody modelowania 447
9.5.1. Modele fraktalne 447
9.5.2. Modele wykorzystujące gramatyki 453
Podsumowanie 456
Zadania 457

10. Modelowanie brył 459

10.1. Reprezentowanie brył 460
10.2. Regularyzowane operacje boolowskie 461
10.3. Kopiowanie prymitywów 466
10.4. Reprezentacje z przesuwaniem 467
10.5. Reprezentacje brzegowe 468
10.5.1. Wielościan i reguła Eulera 469
10.5.2. Operacje boolowskie 471
10.6. Reprezentacje z podziałem przestrzennym 473
10.6.1. Dekompozycja na komórki 473
10.6.2. Reprezentacja wokselowa 474
10.6.3. Drzewa ósemkowe 475
10.6.4. Drzewa BSP 479
10.7. Konstruktywna geometria brył 481
10.8. Porównanie reprezentacji 483
10.9. Interfejsy użytkownika dla systemu modelowania brył . . 486
Podsumowanie 487
Zadania 487

11. Światła achromatyczne i barwne 489

11.1. Światło achromatyczne 489
11.1.1. Wybór natężenia 490
11.1.2. Aproksymacja półtonowa 493
11.2. Barwa 497
11.2.1. Psychofizyka 498
11.2.2. Wykres chromatyczność CIE 502
11.3. Modele barw dla grafiki rastrowej 506
11.3.1. Model barw RGB 507
11.3.2. Model barw CMY 508
11.3.3. Model barw YIQ 509
11.3.4. Model barw HSV 511
11.3.5. Interakcyjny wybór barwy 514
11.3.6. Interpolacja w przestrzeni barw 516
11.4. Wykorzystanie barwy w grafice komputerowej 517
Podsumowanie 520
Zadania 521

12. Dążenie do wizualnego realizmu 523

12.1. Dlaczego realizm? 524

12.2. Podstawowe trudności 526

12.3. Metody renderingu dla rysunków odcinkowych 528

12.3.1. Wielokrotne rzuty prostokątne 528
12.3.2. Rzuty perspektywiczne 529
12.3.3. Wskazówki na temat głębokości (odległości) 529
12.3.4. Obcinanie w funkcji odległości 530
12.3.5. Tekstura 531
12.3.6. Barwa 531
12.3.7. Określanie linii widocznych 531

12.4. Metody renderingu dla obrazów cieniowanych 532

12.4.1. Określanie powierzchni widocznych 532
12.4.2. Oświetlanie i cieniowanie 532
12.4.3. Cieniowanie interpolacyjne 533
12.4.4. Właściwości materiału 534
12.4.5. Modelowanie powierzchni krzywoliniowych 534
12.4.6. Ulepszone oświetlanie i cieniowanie 534
12.4.7. Tekstura 534
12.4.8. Cienie 535
12.4.9. Przezroczystość i odbicia 535
12.4.10. Ulepszone modele kamery 536

12.5. Ulepszone modele obiektów 536

12.6. Dynamika i animacja 537

12.6.1. Znaczenie ruchu 537
12.6.2. Animacja 538

12.7. Stereopsja 541

12.8. Ulepszone wyświetlacze 542

12.9. Interakcja z innymi zmysłami 542

Podsumowanie 543

Zadania 544

13. Wyznaczanie powierzchni widocznych 545

13.1. Metody dla efektywnych algorytmów wyznaczania powierzchni widocznych 547

13.1.1. Spójność 548
13.1.2. Przekształcenie perspektywiczne 549
13.1.3. Prostokąty i bryły ograniczające 552

13.1.4. Wybieranie tylnych ścian 554
13.1.5. Podział przestrzenny 555
13.1.6. Hierarchia 556
13.2. Algorytm z-bufora 557
13.3. Algorytmy przeglądania 561
13.4. Wyznaczanie powierzchni widocznych metodą śledzenia promieni 566
13.4.1. Obliczanie przecięć 567
13.4.2. Problemy efektywności metody śledzenia promieni przy wyznaczaniu powierzchni widocznych . . . 571
13.5. Inne rozwiązania 574
13.5.1. Algorytmy z listą priorytetów 574
13.5.2. Algorytmy podziału powierzchni 579
13.5.3. Algorytmy wyznaczania powierzchni krzywoliniowych . 582
Podsumowanie 583
Zadania 586

14. Oświetlenie i cieniowanie 589

14.1. Modele oświetlenia 591
14.1.1. Światło otoczenia 591
14.1.2. Odbicie rozproszone 592
14.1.3. Tłumienie atmosferyczne 597
14.1.4. Odbicie zwierciadlane 598
14.1.5. Ulepszenie modelu punktowego źródła światła 601
14.1.6. Wiele źródeł światła 603
14.1.7. Modele oświetlenia mające podłoże fizyczne 604
14.2. Modele cieniowania wielokątów 607
14.2.1. Cieniowanie stałą wartością 607
14.2.2. Cieniowanie z interpolacją 608
14.2.3. Cieniowanie siatki wielokątowej 609
• 14.2.4. Cieniowanie Gourauda 610
• 14.2.5. Cieniowanie Phonga 611
14.2.6. Problemy z cieniowaniem z interpolacją 613
14.3. Szczegóły powierzchni 615
14.3.1. Detale wielokątowe 615
14.3.2. Odwzorowanie tekstury 616
14.3.3. Odwzorowanie nierówności powierzchni 618
14.3.4. Inne podejścia 618
14.4. Cienie 619

14.4.1. Generowanie cieni metodą przeglądania wierszy . . . 620
14.4.2. Bryły cienia 621
14.5. Przezroczystość 623
14.5.1. Przezroczystość bez załamań 624
14.5.2. Przezroczystość z załamaniem 626
14.6. Algorytmy oświetlenia globalnego 628
14.7. Rekursywna metoda śledzenia promieni 630
14.8. Metody energetyczne 635
14.8.1. Równanie energetyczne 636
14.8.2. Obliczanie współczynników sprzężenia 638
14.8.3. Progresywne ulepszanie 641
14.9. Potok renderingu 643
14.9.1. Potoki dla oświetlenia lokalnego 643
14.9.2. Potoki dla oświetlenia globalnego 647
14.9.3. Metoda progresywnych ulepszeń 648
Podsumowanie 648
Zadania 649

Bibliografia 651

Skorowidz 668

1. Wprowadzenie: grafika komputerowa

Witamy w grafice komputerowej – jednym z najbardziej ekscytujących działów informatyki. Tak, grafika komputerowa jest działem informatyki, ale jej oddziaływanie sięga daleko poza tę specjalistyczną dziedzinę. W krótkim czasie grafika komputerowa przyciągnęła kilku z najbardziej twórczych ludzi na świecie. Przyszli oni z różnych dyscyplin – sztuki, nauki, muzyki, tańca, filmu i wielu innych. Żeby wymienić choć kilku: animatorzy z firmy Disney korzystali z grafiki komputerowej w celu nadania specjalnego uroku scenie w sali balowej w *Beauty and the Beast.* Choreograf Merce Cunningham korzysta z grafiki komputerowej do zapisywania ruchu – scenariuszy, na podstawie których tancerze uczą się kroków. David Em, uznany artysta korzystający z konwencjonalnych mediów, obecnie w swojej pracy używa stale grafiki komputerowej. Ponieważ zakres i różnorodność grafiki komputerowej najlepiej jest przedstawić na przykładach jej zastosowań, przyjrzyjmy się dokładniej kilku takim zastosowaniom.

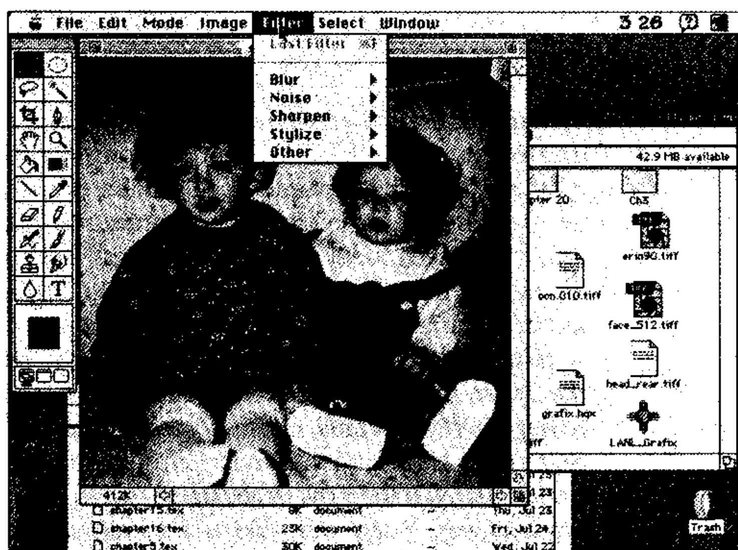
1.1. Przykłady zastosowań grafiki komputerowej

Obecnie grafika komputerowa jest stosowana w wielu różnych obszarach przemysłu, biznesu, w instytucjach rządowych, nauczaniu i rozrywce. Lista zastosowań jest ogromna i szybko wzrasta wraz z rozpowszechnianiem się komputerów wyposażonych w możliwości graficzne. Poniżej podano krótki przegląd kilku zastosowań.

▻ **Interfejsy użytkownika.** Prawdopodobnie już korzystaliśmy z grafiki komputerowej, być może nieświadomie. Jeżeli pracowaliśmy na

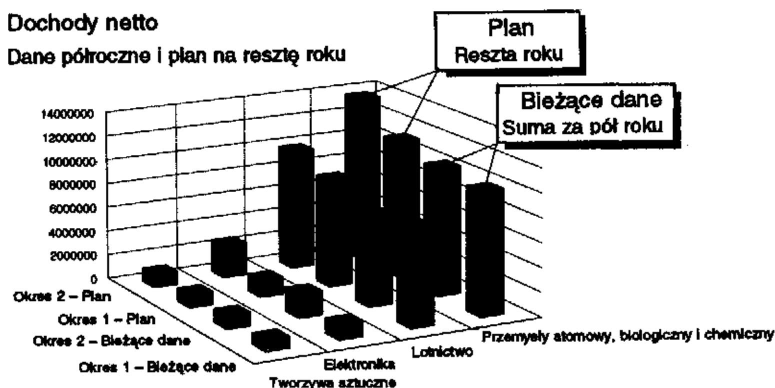
komputerach Macintosh albo zgodnych z IBM z systemem Windows 3.1, to jesteśmy prawdopodobnie wytrawnymi użytkownikami grafiki.

Większość programów użytkowych wykonywanych na komputerach osobistych i na stacjach roboczych ma interfejsy użytkownika (podobne do pokazanych na rys. 1.1) z systemem okien zarządzającym licznymi równoczesnymi czynnościami i z możliwościami wskazywania, pozwalającymi użytkownikowi wybierać opcje z menu, ikony i obiekty na ekranie. Programy przetwarzania tekstu, arkusze kalkulacyjne i programy przygotowania publikacji, to typowe zastosowania korzystające z metod interfejsu użytkownika. Jak zobaczymy, grafika odgrywa istotną rolę zarówno w funkcjach wejściowych, jak i wyjściowych interfejsu użytkownika.



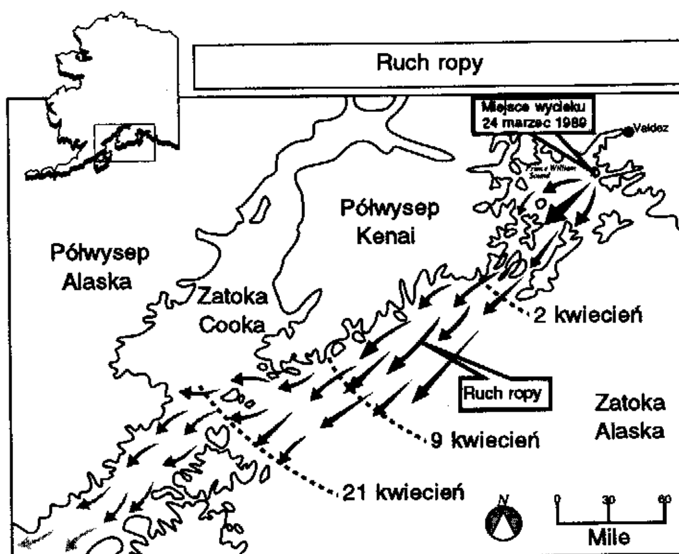
Rys. 1.1. Ekran komputera Macintosh z widocznymi oknami, menu i ikonami

▷ **Wykresy w biznesie, nauce i technologii.** Następnym bardzo popularnym obszarem zastosowań dzisiejszej grafiki jest tworzenie wykresów 2D i 3D funkcji matematycznych, fizycznych i ekonomicznych; histogramów i wykresów kołowych; wykresów harmonogramowania zadań; wykresów wielkości zapasów i produkcji itd. Wszystkie te wykresy są używane do prezentowania w przejrzysty i zwięzły sposób tendencji i wzorów uzyskanych z danych, tak żeby wyjaśniać złożone zjawiska i ułatwić podejmowanie decyzji. Na rysunku 1.2 pokazano typowy przykład trójwymiarowego wykresu danych ekonomicznych.



Rys. 1.2. Dane ekonomiczne przedstawione w postaci wykresu słupkowego 3D

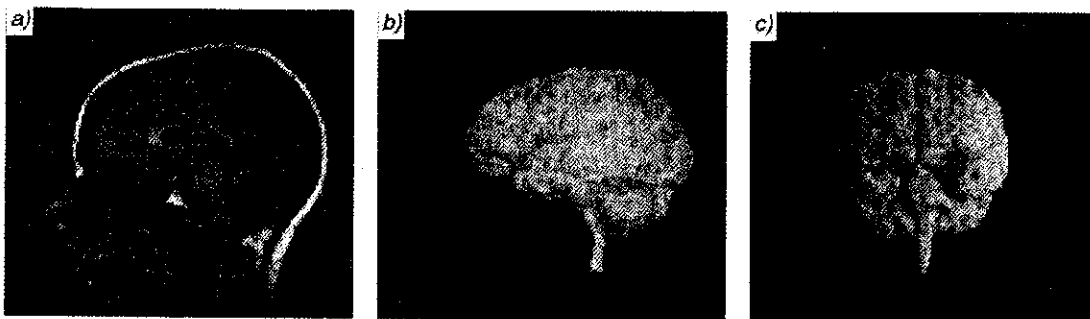
▷ **Kartografia.** Grafika komputerowa jest używana do tworzenia zarówno dokładnych, jak i uproszczonych informacji geograficznych oraz o różnych zjawiskach naturalnych na podstawie danych pomiarowych. Przykładami mogą być mapy geograficzne, mapy plastyczne, mapy eksploracyjne dla wierceń i prowadzenia prac górniczych, wykresy oceanograficzne, mapy pogody, mapy warstwicowe, mapy demograficzne. Na rysunku 1.3 pokazano mapę, którą opracowano w celu zilustrowania niektórych aspektów katastrofy spowodowanej wyciekiem ropy w 1989 r. pod Valdez. Ta mapa jest przykładem



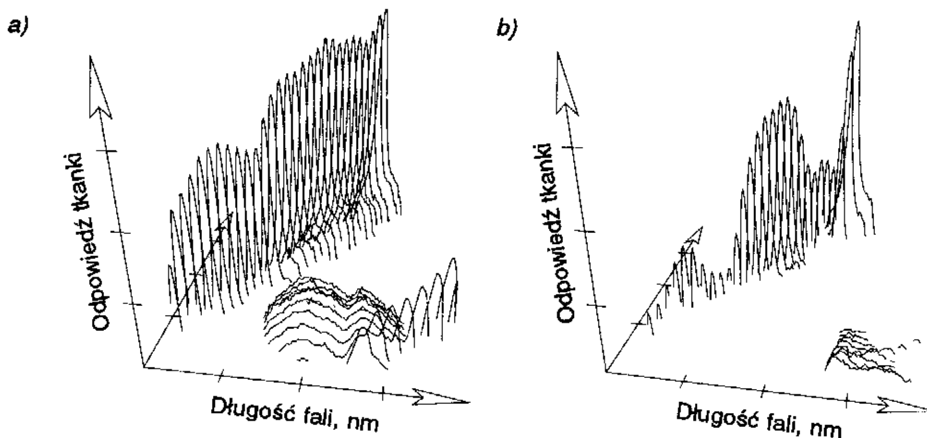
Rys. 1.3. Wykorzystanie podłoża kartograficznego do przedstawienia danych o charakterze geograficznym (za zgodą Toma Poikera z Simon Fraser University)

odwzorowania tematycznego, gdzie wartości danych – takie jak przemieszczanie się ropy – są nałożone na mapę stanowiącą w tym przypadku tło.

▻ **Medycyna.** Grafika komputerowa odgrywa coraz większą rolę w takich dziedzinach jak diagnostyka medyczna lub planowanie operacji. W tym ostatnim przypadku chirurdzy korzystają z grafiki do wspomagania kierowania przyrządami i do dokładnego określania, w którym miejscu należy usunąć chorą tkankę. Przykład zastosowania grafiki komputerowej w diagnostyce pokazano na rys. 1.4. Widać na nim trzy obrazy mózgu człowieka otrzymane metodą rezonansu magnetycznego. Na podstawie szeregu równoległych obrazów pokazanych na rys. 1.4a diagnosta może stworzyć reprezentacje 3D mózgu pokazane na rys. 1.4b i c. Użytkownik może interakcyjnie



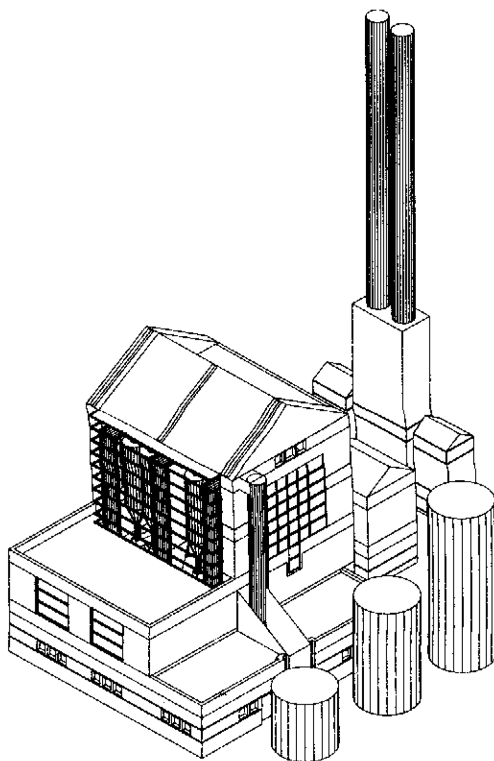
Rys. 1.4. 3D reprezentacja mózgu człowieka skonstruowana z obrazów uzyskanych metodą rezonansu magnetycznego. Na podstawie zestawu obrazów z rys. a) w wyniku przetwarzania uzyskuje się obrazy 3D pokazane na rys. b) (widok z boku) i c) (widok z tyłu) (za zgodą Johna George'a z Los Alamos National Laboratory)



Rys. 1.5. Wyniki biopsji optycznej: a) sygnatura normalnej tkanki wątroby; b) sygnatura zrakowaciałej tkanki wątroby (za zgodą Thomasa Loree'a z Los Alamos National Laboratory)

manipulować modelem i uzyskać szczegółowe informacje o stanie mózgu. Inne wyjątkowo ekscytujące zastosowanie grafiki w medycynie pokazano na rys. 1.5. Widać tu wykresy 3D danych otrzymanych po oświetleniu żywej tkanki światłem laserowym; metoda ta jest określana jako *biopsja optyczna*. W tym przypadku widać dużą różnicę w optycznych sygnaturach normalnej a) i zrakowaciałej b) wątroby psa; stwarza to nadzieję na możliwość diagnozy niechirurgicznej.

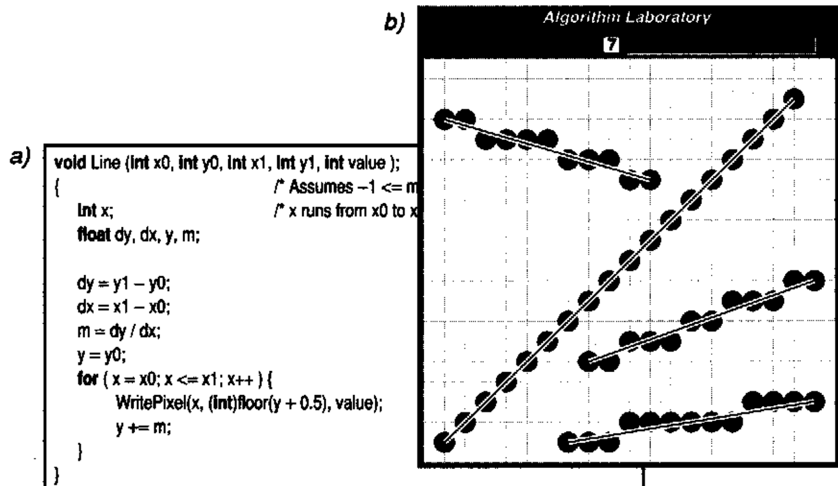
▻ **Kreślenie i projektowanie wspomagane komputerowo.** W projektowaniu wspomaganym komputerowo (CAD) użytkownik korzysta z grafiki interakcyjnej do projektowania elementów i systemów mechanicznych, elektrycznych, elektromechanicznych i elementów elektronicznych, w tym takich struktur jak budynki, karoserie samochodów, kadłuby samolotów i statków, struktury o bardzo dużym stopniu scalenia (VLSI) oraz sieci telefoniczne i komputerowe. Zazwyczaj nacisk jest kładziony na interakcję z modelem komputerowym projektowanego elementu albo systemu, ale niekiedy użytkownik chce szybko uzyskać dokładne rysunki elementów i zespołów,



Rys. 1.6. Rysunek elektrowni uzyskany za pomocą systemu wspomagania architekta (za zgodą Harolda Borkina z Architecture and Planning Research Lab, University of Michigan)

np. szkice lub projekty architektoniczne. Na rysunku 1.6 pokazano rysunek elektrowni; jest to przykład wykorzystania systemu CAD przez architektów.

▻ **Systemy multimedialne.** Grafika komputerowa odgrywa krytyczną rolę w szybko powiększającym się obszarze systemów multimedialnych. Jak już wynika z nazwy, multimedia to więcej niż jedno medium komunikacyjne. W takich systemach na ogół mamy tekst, grafikę i dźwięk; mogą występować również inne media [PHIL91]. Na rysunku 1.7 pokazano, jak niekonwencjonalne elementy multimedialne mogą być wykorzystane w nauczaniu. Rysunek jest zaczerpnięty z hipotetycznego multimedialnego podręcznika grafiki komputerowej [PHIL92]. Na rysunku 1.7a pokazano procedurę rysowania odcinka 2D (znaczenie tego kodu będziemy omawiali dokładnie w rozdz. 3). Kod wyświetlany na multimedialnej stronie jest gotowy do wykonania; oznacza to, że czytelnik wskazując go może go wykonać. Pojawia się wtedy okno robocze, pokazane na rys. 1.7b, które umożliwia użytkownikowi wypróbowanie różnych kombinacji położenia końców odcinka i obserwowanie działania algorytmu.



Rys. 1.7. Interakcyjny algorytm z multimedialnego podręcznika: a) kod komputerowy realizujący algorytm; b) interakcyjne okno, które pojawia się gdy użytkownik wskaże kod. Określając końce odcinka, można spowodować wykonanie kodu

▻ **Symulacja i animacja dla wizualizacji naukowej i rozrywki.** W wizualizacji naukowej i inżynierskiej coraz popularniejsze stają się obrazy i filmy animowane generowane komputerowo, pokazujące zmienne w czasie zachowanie się rzeczywistych i symulowanych obiektów (fot. 1). Z narzędzi takich można korzystać przy badaniu abstrak-

cyjnych wielkości matematycznych i modeli matematycznych takich zjawisk jak przepływ cieczy, teoria względności, reakcje jądrowe i chemiczne, systemy fizjologiczne i działanie organów, deformacje struktur mechanicznych pod wpływem różnych obciążeń. Inną dziedziną zaawansowanej technologii jest produkcja specjalnych efektów w filmach (fot. 2 i 3). Dostępne są wyrafinowane mechanizmy modelowania obiektów i reprezentowania świateł i cieni.

1.2. Krótka historia grafiki komputerowej

W tej książce skoncentrowano się na podstawowych zasadach i metodach, które opracowano w przeszłości, a które wciąż są stosowane i będą stosowane w przyszłości. W tym punkcie przyjrzymy się historycznemu rozwojowi grafiki komputerowej, tak żeby móc umiejscowić dzisiejsze systemy w jakimś kontekście. Pełniejsze omówienie interesującej ewolucji tej dziedziny można znaleźć w pracach [PRIN71], [MACH78], [CHAS81] i [CACM84]. Łatwiej jest podać chronologię rozwoju sprzętu niż oprogramowania, ponieważ ewolucja sprzętu miała większy wpływ na rozwój tej dziedziny. Dlatego zaczniemy od sprzętu.

Już w pierwszych dniach istnienia informatyki wykonywano niedoskonałe rysunki na urządzeniach drukujących takich jak dalekopisy i drukarki wierszowe. Komputer Whirlwind opracowany w 1950 r. w MIT miał wyjściowe urządzenie wyświetlające z elektronopromieniową lampą CRT sterowaną przez komputer, przeznaczoną dla operatora oraz dla tworzenia kopii za pomocą aparatu fotograficznego. Początki nowoczesnej grafiki interakcyjnej można znaleźć w brzemiennej w skutki pracy doktorskiej Ivana Sutherlanda poświęconej systemowi rysującemu Sketchpad [SUTH63]. Wprowadził on struktury danych dla pamiętania hierarchii symboli budowanych przez powielanie standardowych elementów, technikę podobną do tej, jaka jest używana w plastykowych szablonach do rysowania symboli układów. Sutherland opracował również metody interakcji za pomocą klawiatury i pióra świetlnego (ręczne urządzenie wskazujące, reagujące na światło emitowane przez obiekt znajdujący się na ekranie) do dokonywania wyborów, wskazywania i rysowania oraz sformułował wiele innych podstawowych idei i metod, które wciąż są stosowane. Wiele funkcji wprowadzonych w systemie Sketchpad można znaleźć w pakiecie graficznym PHIGS omawianym w rozdz. 7.

W tym samym czasie producenci komputerów, samochodów i sprzętu lotniczego zaczeli doceniać ogromny potencjał systemów CAD i CAM (komputerowe wspomaganie projektowania i produkcji)

w zakresie automatyzacji rysowania i innych czynności wymagających wykonywania wielu rysunków. Systemy CAD z General Motors [JACK64] do projektowania samochodów oraz Itek Digitek [CHAS81] dla projektowania soczewek były pionierskimi programami, które pokazały użyteczność interakcji graficznej w iteracyjnych cyklach projektowania, popularnych w praktyce inżynierskiej. W połowie lat sześćdziesiątych pojawiło się kilka projektów badawczych i komercyjnych produktów.

W tamtych czasach, informacje wprowadzano i wyprowadzano za pomocą kart dziurkowanych w trybie wsadowym i wiązano duże nadzieje z wprowadzeniem interakcyjnej komunikacji użytkownik-komputer. Grafika komputerowa jako „okno komputera" miała stać się integralną częścią ogromnie przyspieszonego cyklu interakcyjnego projektowania. Jednak wynik nie był nawet w przybliżeniu taki, jakiego oczekiwano, ponieważ grafika interakcyjna ze względu na wysokie koszty pozostawała wówczas poza zasięgiem większości potencjalnych użytkowników (z wyjątkiem organizacji najbardziej zaawansowanych technologicznie).

Aż do początku lat osiemdziesiątych grafika komputerowa była wąską specjalizacją, głównie ze względu na koszty sprzętu i niewielką liczbę programów użytkowych korzystających z grafiki. Później komputery osobiste z wbudowanymi rastrowymi urządzeniami wyświetlającymi – np. Apple Macintosh i IBM PC oraz jego klony – spopularyzowały korzystanie z grafiki z *mapą bitową* w interakcji użytkownika z komputerem. *Mapa bitowa* jest to zero-jedynkowa reprezentacja prostokątnej tablicy punktów na ekranie, nazywanych *pikselami* albo *pelami* (skrót od *picture elements*). Wkrótce po tym, jak dostępna stała się grafika z mapą bitową, nastąpiła eksplozja łatwych w użytkowaniu i tanich zastosowań grafiki. Interfejsy użytkownika wykorzystujące grafikę umożliwiły milionom nowych użytkowników korzystanie z prostych, tanich programów użytkowych takich jak arkusze kalkulacyjne, procesory tekstów i programy rysujące.

Koncepcja biurka stała się popularną metaforą organizowania powierzchni ekranu. Dzięki programowi zarządzania oknami użytkownik może tworzyć, umieszczać w odpowiednim miejscu na biurku i zmieniać wielkość prostokątnych powierzchni zwanych *oknami*, które zachowują się jak wirtualne terminale graficzne, z których każdy wykonuje jakiś program użytkowy. Takie podejście pozwala użytkownikom przełączać się między licznymi zadaniami na zasadzie wskazywania potrzebnego okna, zazwyczaj za pomocą urządzenia wskazującego nazywanego *myszką*. Podobnie jak kawałki papieru na nieuporządkowanym biurku, okna mogłyby pokryć całą powierzchnię. Częścią tej metafory biurka było również wyświetlanie ikon, które reprezentowały nie tylko pliki

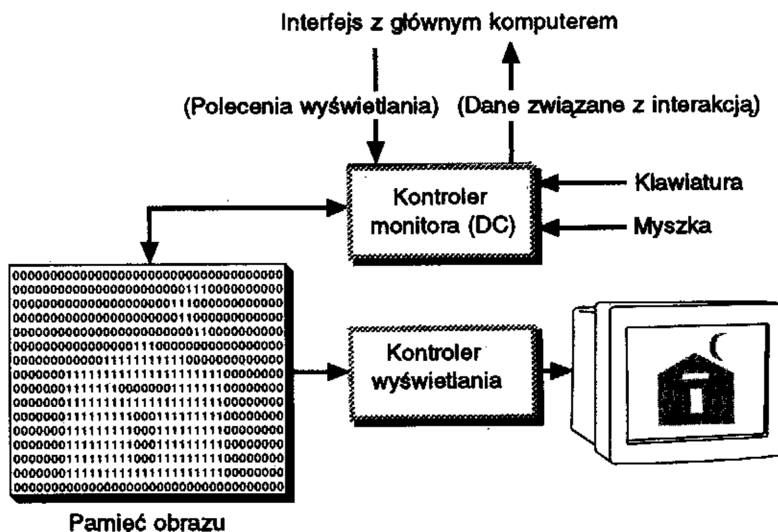
danych i programy użytkowe, ale również sprzęty zwykle znajdujące się w biurze – szafki na akta, pudełka na pocztę, drukarki i pojemnik na śmieci – które odgrywają rolę komputerowych odpowiedników ich rzeczywistych pierwowzorów (rys. 1.1).

Bezpośrednie manipulowanie obiektami na zasadzie ich *wskazywania* i *naciskania* na myszkę zastąpiło w znacznym stopniu wypisywanie tajemniczych poleceń stosowanych we wcześniejszych komputerach. W ten sposób użytkownicy mogą wybierając ikony uruchamiać odpowiednie programy lub obiekty albo naciskać przyciski na rozwijanych albo chwilowych menu w celu dokonywania wyborów. Obecnie prawie wszystkie interakcyjne programy użytkowe, nawet te do manipulowania tekstem (na przykład procesory tekstów) albo danych numerycznych (na przykład programy arkuszy kalkulacyjnych), powszechnie wykorzystują grafikę w interfejsie użytkownika i dla wizualizacji i manipulowania obiektami specyficznymi dla zastosowania.

1.2.1. Wyprowadzanie informacji

Urządzenia wyświetlające opracowane w połowie lat sześćdziesiątych i wykorzystywane do połowy lat osiemdziesiątych są określane jako *monitory wektorowe, kreskowe, rysujące odcinki* albo *kaligraficzne*. Tutaj określenie wektor jest używane jako synonim odcinka; kreska jest krótkim odcinkiem i znaki są tworzone z ciągów takich kresek. Typowy system wektorowy zawiera procesor monitora przyłączony jako zewnętrzne urządzenie wejścia/wyjścia do centralnej jednostki przetwarzającej (CPU), pamięci buforowej monitora i CRT. Istotą systemu wektorowego jest to, że strumień elektronów, który pisze po pokryciu luminoforowym CRT (zob. rozdz. 4), jest odchylany od jednego końca odcinka do drugiego, zależnie od arbitralnej kolejności poleceń wyświetlania; taka metoda jest określana jako *przeszukiwanie przypadkowe* (rys. 1.10b). Ponieważ światło emitowane przez luminofor zanika w ciągu dziesiątek albo najwyżej setek mikrosekund, procesor monitora musi cyklicznie przebiegać listę wyświetlanych elementów w celu odświeżania luminoforu przynajmniej 30 razy na sekundę (30 Hz) po to, żeby uniknąć migotania.

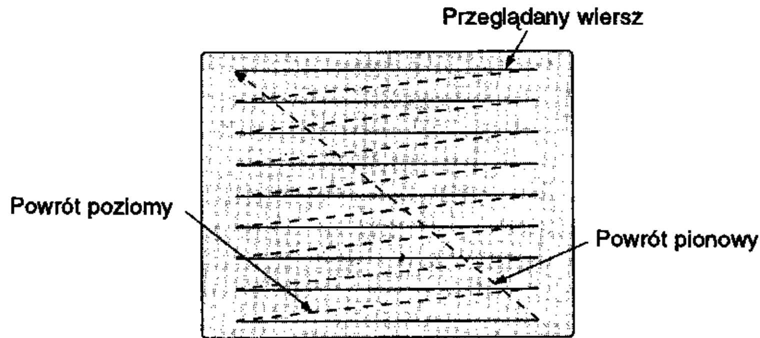
Opracowanie na początku lat siedemdziesiątych taniej grafiki rastrowej wykorzystującej technologię telewizyjną miało znacznie większy wpływ na rozwój tej dziedziny niż jakakolwiek inna technologia. *Monitory rastrowe* pamiętają wyświetlane *prymitywy* (np. odcinki, znaki i obszary wypełnione w sposób ciągły albo za pomocą wzorów) w pamięci ekranu w postaci pikseli tworzących określony prymityw (rys. 1.8). W niektórych monitorach rastrowych sprzętowy kontroler monitora



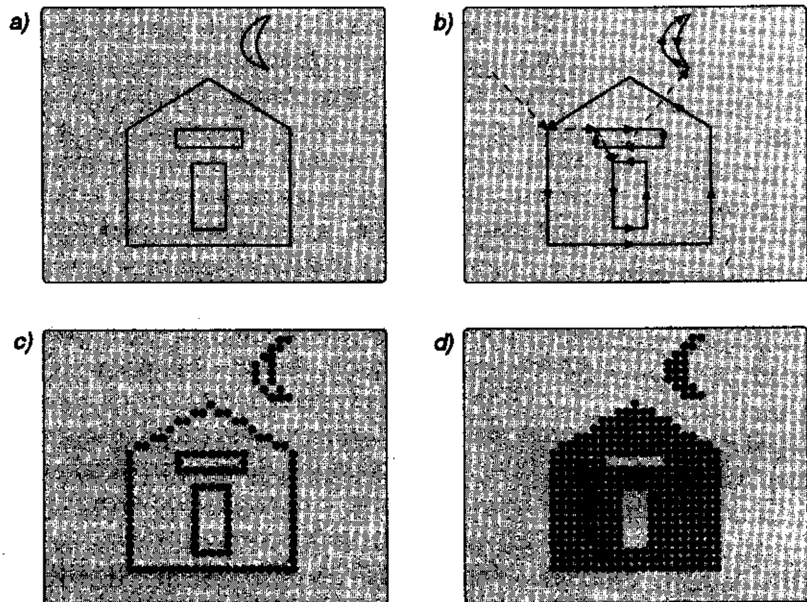
Rys. 1.8. Architektura monitora rastrowego

(taki jak na rysunku) odbiera i interpretuje sekwencje poleceń wyjściowych; w prostszych systemach, np. w komputerach osobistych, kontroler monitora istnieje tylko jako element programowy biblioteki graficznej, a pamięć ekranu jest po prostu częścią pamięci CPU, która może być odczytywana przez podsystem wyświetlania obrazu (określany często jako *sterownik wyświetlania*), który tworzy aktualny obraz na ekranie.

Kompletny obraz na monitorze rastrowym jest tworzony na bazie rastra, czyli zbioru poziomych linii składających się z pikseli. Raster jest zapamiętywany jako tablica pikseli reprezentujących całą powierzchnię ekranu. Cały obraz jest tworzony sekwencyjnie przez sterownik wyświetlania, linia po linii z góry na dół i potem ponownie od góry (rys. 1.9). Dla każdego piksela natężenie strumienia jest tak ustawiane, żeby odzwierciedlić jasność piksela; w systemach barwnych są sterowane trzy strumienie – po jednym dla każdej barwy podstawowej: czerwonej, zielonej i niebieskiej – zgodnie ze specyfikacją trzech składowych barwy dla każdej wartości piksela (zob. rozdz. 4 i 11). Na rysunku 1.10 pokazano różnicę między przeszukiwaniem przypadkowym b) i rastrowym c) na przykładzie wyświetlania prostego rysunku domu. Na rysunku 1.10b strzałki pokazują przypadkowe odchylanie promienia. Linie przerywane oznaczają odchylenie promienia, który nie jest włączony (jest *wygaszony*); wtedy nie jest rysowany żaden wektor. Na rysunku 1.10c pokazano nie wypełniony dom utworzony z prostokątów, wielokątów i łuków; na rys. 1.10d pokazano wersję wypełnioną. Zwróćmy uwagę na postrzępiony wygląd odcinków i łuków w obrazach rastrowych na rys. 1.10c i d; niżej krótko omówimy to zakłócenie (artefakt).



Rys. 1.9. Przeglądanie rastrowe



Rys. 1.10. Przeszukiwanie przypadkowe a przeszukiwanie rastrowe. Ekran jest przedstawiony symbolicznie w postaci jasnoszarego prostokąta z zaokrąglonymi rogami; barwa wypełniająca prostokąt reprezentuje białe tło, na którym jest narysowany czarny obraz: a) rysunek wykonany za pomocą idealnych linii; b) przeglądanie przypadkowe; c) przeglądanie rastrowe dla prymitywów krawędziowych; d) przeglądanie rastrowe dla prymitywów rastrowych

W systemie rastrowym cała siatka pikseli (na przykład 1024 linie po 1024 piksele) musi być bezpośrednio zapamiętana. Na początku lat siedemdziesiątych brak tanich półprzewodnikowych pamięci typu RAM potrzebnych do budowy pamięci dla mapy bitowej stanowił istotne ograniczenie rozwoju grafiki rastrowej i uniemożliwiał jej zdobycie dominującej pozycji. *Dwupoziomowe* monitory CRT (określane również jako *monochromatyczne*) rysowały obrazy biało-czarne albo

czarno-zielone; w niektórych wyświetlaczach plazmowych są wykorzystywane barwy czarna i pomarańczowa. W dwupoziomowej mapie bitowej każdemu pikselowi jest przyporządkowany 1 bit i cała mapa bitowa ekranu o rozdzielczości 1024 na 1024 piksele liczy 2^{20} bitów albo ok. 128 000 bajtów. W prostych systemach barwnych jest 8 bitów na piksel, dzięki czemu dostępnych jest równocześnie 256 barw; w droższych systemach są 24 bity na piksel i istnieje możliwość wybrania jednej z 16 milionów barw; dostępne są pamięci obrazu z 32 bitami na piksel i rozdzielczością ekranu 1280 na 1024 piksele, nawet w komputerach osobistych. Z tych 32 bitów, 24 są przeznaczone na reprezentowanie barwy a 8 jest wykorzystywanych do celów sterowania; omawiamy to w rozdz. 4. Typowy system barwny o rozdzielczości 1280 na 1024 z 24 bitami na piksel wymaga pamięci RAM o pojemności 3,75 MB, która przy dzisiejszych standardach jest niedroga. Dokładniej mówiąc, pojęcie mapy bitowej odnosi się tylko do systemów dwupoziomowych, gdzie jest 1 bit na piksel; dla systemów, gdzie jest wiele bitów dla jednego piksela, używamy bardziej ogólnego pojecia *mapy pikselowej*. W potocznym języku mapa pikselowa odnosi się zarówno do zawartości pamięci obrazu, jak i do samej pamięci.

Główne zalety grafiki rastrowej w porównaniu z grafiką wektorową to niższy koszt i możliwość wyświetlania obszarów wypełnionych jednolitą barwą albo wzorami – jest to niesłychanie ważna funkcja przy tworzeniu realistycznych obrazów obiektów 3D.

Główna wada systemów rastrowych w porównaniu z systemami wektorowymi jest związana z dyskretną naturą reprezentacji piksela. Prymitywy, takie jak odcinki i wielokąty, są określane przez parametry ich końców (wierzchołków) i muszą być odwzorowane w pamięci obrazu za pomocą pikseli. Ten proces odwzorowania, czy też konwersji, jest określany dalej jako rasteryzacja. W efekcie programista określa współrzędne końców odcinka albo wierzchołków w trybie przeglądania przypadkowego, a system zapewnia przejście do reprezentacji pikselowej przed wyświetleniem w trybie przeglądania rastrowego. W komputerach osobistych i tanich stacjach roboczych, gdzie mikroprocesor CPU jest odpowiedzialny za całą grafikę, rasteryzacja jest często wykonywana programowo.

Inna wada systemów rastrowych wynika z natury rastra. Podczas gdy system wektorowy może rysować ciągłe gładkie odcinki (a nawet gładkie krzywe) w zasadzie od dowolnego punktu na ekranie CRT do dowolnego innego punktu, system rastrowy może wyświetlać matematycznie gładkie linie, wielokąty i brzegi krzywoliniowych prymitywów, takich jak okręgi i elipsy, tylko na zasadzie ich aproksymacji za pomocą pikseli należących do siatki rastra. Taka aproksymacja może powodować powstawanie dobrze znanego problemu „schodków" albo „ząbków", jak to widać na rys. 1.10c i d. To wizualne zakłócenie jest kon-

sekwencją błędu próbkowania określanego w teorii przetwarzania sygnałów jako *aliasing*; takie zakłócenia pojawiają się wówczas, gdy funkcja ciągłej zmiennej zawierająca ostre zmiany jasności jest aproksymowana za pomocą dyskretnych próbek. We współczesnej grafice komputerowej są stosowane metody zwalczania zakłóceń (tak zwany *antyaliasing*) w systemach ze skalą szarości i w systemach barwnych. Te metody określają gradację jasności sąsiednich pikseli na granicach prymitywów (a nie ograniczają się do ustawiania tylko maksymalnej albo zerowej jasności); omówienie tego ważnego zagadnienia jest w rozdz. 3.

1.2.2. Wprowadzanie informacji

Z upływem lat ulepszono również metody wprowadzania informacji. Niewygodne i delikatne pióro świetlne stosowane w technice wektorowej zastąpiono wszechobecną myszką (pierwotnie opracowaną przez pioniera techniki biurowej Douga Engelbarta w połowie lat sześćdziesiątych [ENGE68]), tabliczkę, ekrany czułe na dotyk. Stają się również dostępne urządzenia wejściowe, które określają nie tylko położenie współrzędnych (x, y) na ekranie, ale również współrzędne 3D, a nawet więcej wymiarowe wartości wejściowe (stopnie swobody); problem ten jest omawiany w rozdz. 8. Komunikacja dźwiękowa ma również duże potencjalne możliwości, ponieważ umożliwia wprowadzenie informacji bez pomocy rąk i w sposób naturalny wyprowadzenie prostych poleceń, realizowanie sprzężenia zwrotnego itd. Przy standardowych urządzeniach wejściowych użytkownik może określić operacje albo elementy obrazu pisząc lub rysując nową informację albo wskazując informację istniejącą już na ekranie. Ta interakcja nie wymaga znajomości programowania – trzeba tylko umieć posługiwać się klawiaturą: użytkownik dokonuje wyborów po prostu na zasadzie wybierania przycisków albo ikon menu, odpowiada na pytania przez oznaczanie opcji albo wpisanie kilku znaków w formularzu, umieszcza na ekranie kopii predefiniowane symbole, rysuje na zasadzie wskazywania kolejnych punktów końcowych, które mają być połączone odcinkami albo interpolowane przez gładkie krzywe, maluje przesuwając kursor po ekranie i wypełnia wielokąty albo zamalowuje wnętrza konturów poziomami szarości, barwami albo różnymi wzorami.

1.2.3. Przenośność oprogramowania i standardy graficzne

Postęp w technologii sprzętu umożliwił ewolucję monitorów graficznych od czegoś w rodzaju specjalizowanych urządzeń wyjściowych do standardowego interfejsu użytkownika z komputerem. Możemy się

zastanawiać, czy oprogramowanie dotrzymało kroku. Na przykład, w jakim stopniu rozwiązaliśmy początkowe trudności ze zbyt złożonymi, niewygodnymi i drogimi systemami graficznymi i programami użytkowymi. Przeszliśmy od pakietów niskiego poziomu zależnych od urządzeń, dostarczanych przez producentów i przystosowanych do specyficznych urządzeń wyświetlających, do pakietów wyższego poziomu niezależnych od urządzeń. Te pakiety mogą sterować różnymi urządzeniami wyświetlającymi, poczynając od drukarek laserowych i ploterów a kończąc na naświetlarkach filmów i dobrej jakości interakcyjnych monitorach. Głównym celem korzystania z pakietów niezależnych od sprzętu w połączeniu z językami wysokiego poziomu jest sprzyjanie przenośności programów użytkowych. Pakiet zapewnia przenośność w zasadzie w taki sam sposób jak w przypadku języka wysokiego poziomu (np. Fortran, Pascal albo C) niezależnego od komputera przez izolowanie programisty od szczegółów związanych z maszyną i przez udostępnienie funkcji łatwych do implementacji na wielu różnych procesorach.

Ogólna świadomość potrzeby standardów dla grafiki niezależnej od urządzeń pojawiła się w połowie lat siedemdziesiątych; punktem kulminacyjnym była specyfikacja 3D Core Graphics System (w skrócie Core) opracowana przez ACM SIGGRAPH[1] Committee w 1977 r. [GSPC77] i ulepszona w 1979 r. [GSPC79].

Specyfikacja systemu Core odegrała zamierzoną rolę specyfikacji bazowej. Core nie tylko miał wiele implementacji, ale był również wykorzystany jako wyjściowy system dla oficjalnych (rządowych) projektów standardów w ramach ANSI (American National Standards Institute) i ISO (International Standards Organization). Pierwszą specyfikacją graficzną, która została przyjęta jako oficjalny standard, był GKS (Graphical Kernel System) [ANSI85], dopracowana wersja systemu Core, jednak w przeciwieństwie do tego systemu ograniczona do 2D. W 1988 r. oficjalnym standardem stał się GKS-3D [INTE88], rozszerzenie GKS-u w kierunku 3D; podobnie oficjalnym standardem stał się bardziej złożony system graficzny PHIGS (Programmer's Hierarchical Interactive Graphics System [ANSI88]). W systemie GKS są wyróżnione *segmenty* grupujące logicznie powiązane ze sobą prymitywy, np. odcinki, wielokąty i ciągi znaków i ich atrybuty; te segmenty nie

[1] SIGGRAPH (Special Interest Group on Graphics) jest to jedna z profesjonalnych grup działających w ramach ACM (Association for Computing Machinery). ACM jest jednym z dwóch największych profesjonalnych stowarzyszeń profesjonalnych informatyków; drugie to IEEE Computer Society. SIGGRAPH publikuje naukowe pisma i sponsoruje coroczną konferencję, na której są prezentowane prace badawcze z zakresu grafiki i z którą jest związana wystawa sprzętu. IEEE Computer Society również publikuje pismo naukowe z zakresu grafiki.

mogą być zagnieżdżane. PHIGS, jak to już z jego nazwy wynika, dopuszcza zagnieżdżane hierarchiczne grupy podprymitywów 3D, nazywane *strukturami.* W systemie PHIGS wszystkie prymitywy, włącznie z przywoływaniem struktur, podlegają przekształceniom geometrycznym (skalowanie, obroty i przesunięcia), dzięki czemu jest możliwy ruch. PHIGS dopuszcza bazy danych struktur, które programista może selektywnie poddawać edycji; PHIGS automatycznie uaktualnia ekran po zmianie bazy danych. PHIGS został rozszerzony o zestaw funkcji dla nowoczesnego pseudorealistycznego renderingu[2] obiektów na monitorach rastrowych; to rozszerzenie zostało określone jako PHIGS+ [PHIG88], zanim zostało skierowane do ANSI/ISO oraz PHIGS PLUS wewnątrz ISO [PHIG92]. Ze względu na dużą liczbę funkcji i złożoność specyfikacji, implementacje systemu PHIGS są dużymi pakietami. Implementacje systemu PHIGS, a zwłaszcza PHIGS PLUS są wykonywane najszybciej wówczas, gdy przekształcenia, obcinanie i funkcje renderingu są wspomagane sprzętowo.

Obok oficjalnych standardów publikowanych przez narodowe, międzynarodowe albo profesjonalne organizacje standaryzacyjne istnieją standardy nieoficjalne. Te tak zwane standardy przemysłowe albo de facto są opracowywane, promowane i udostępniane na zasadzie licencji przez poszczególne firmy albo konsorcja firm i uniwersytety. Do dobrze znanych graficznych standardów przemysłowych należą: PostScript firmy Adobe, OpenGL firmy Silicon Graphics, HOOPS firmy Ithaca Software i X Window System z X-Consortium kierowanego przez MIT i jego rozszerzenie PEX (zob. rozdz. 7) w kierunku grafiki 3D (protokoł klient-serwer). Standardy przemysłowe mogą być bardziej rozpowszechnione i dlatego są ważniejsze z punktu widzenia komercyjnego niż oficjalne standardy, ponieważ mogą być szybciej uaktualniane, zwłaszcza te, które są kluczowymi produktami komercyjnymi firmy i stąd mają za sobą znaczne zasoby.

Standardy oprogramowania graficznego są dokładnie omówione w książce. Na początku w rozdz. 2 omawiamy SRGP (Simple Raster Graphics Package), który zapożycza funkcje od popularnego całkowitoliczbowego pakietu QuickDraw firmy Apple [ROSE85] i X Window System z MIT [SCHE88], jeśli chodzi o wyjście, i od GKS-u i PHIGS-a dla wejścia. Po przyjrzeniu się prostym zastosowaniom tego pakietu grafiki rastrowej niskiego poziomu, omówimy algorytmy rasteryzacji

[2] Z *renderingiem pseudorealistycznym* mamy do czynienia wówczas, gdy do opisu odbijania światła od obiektów są wykorzystywane proste prawa optyki. W *renderingu fotorealistycznym* są wykorzystywane dokładniejsze aproksymacje sposobów odbijania i załamywania światła przez obiekty; takie aproksymacje wymagają większych nakładów obliczeniowych, ale prowadzą do generowania obrazów, których jakość jest bliższa jakości fotografii.

i obcinania wykorzystywane w takich pakietach do generowania obrazów prymitywów w pamięci obrazu. Następnie po wprowadzeniu matematycznych podstaw dla przekształceń geometrycznych 2D i 3D w rozdz. 5 i dla rzutowania równoległego i perspektywicznego w rozdz. 6 omówimy w rozdz. 7 znacznie silniejszy pakiet SPHIGS (Simple PHIGS). SPHIGS jest podzbiorem systemu PHIGS, który działa na prymitywach zdefiniowanych w zmiennopozycyjnym, abstrakcyjnym systemie współrzędnych świata 3D, niezależnym od rodzaju technologii wyświetlania i który ma niektóre funkcje systemu PHIGS PLUS. Orientujemy naszą dyskusję na PHIGS i PHIGS PLUS, ponieważ wierzymy, że będą one miały znacznie większy wpływ na interakcyjną grafikę 3D niż GKS-3D, zwłaszcza w kontekście rosnącej dostępności sprzętu, który wspomaga wykonywanie przekształceń w czasie rzeczywistym i rendering obrazów pseudorealistycznych.

1.3. Zalety grafiki interakcyjnej

Grafika zapewnia jeden z najbardziej naturalnych środków komunikacji z komputerem, ponieważ nasze wysoce rozwinięte zdolności rozpoznawania obrazów 2D i 3D umożliwiają nam odbieranie i przetwarzanie danych obrazowych szybko i wydajnie. Dzisiaj w wielu projektach, implementacjach i procesach konstrukcyjnych informacja, jaką niesie obraz, jest w zasadzie niezbędna. Wizualizacja naukowa stała się ważnym obszarem prac w końcu lat osiemdziesiątych, kiedy naukowcy i inżynierowie doszli do wniosku, że nie mogą interpretować ogromnych ilości danych produkowanych przez superkomputery bez przedstawiania danych i uwypuklania trendów i zjawisk za pomocą różnego rodzaju reprezentacji graficznych.

Interakcyjna grafika komputerowa jest najważniejszym środkiem tworzenia obrazów od wynalezienia fotografii i telewizji; ma ona tę dodatkową zaletę, że korzystając z komputera możemy tworzyć obrazy nie tylko istniejących rzeczywistych obiektów, ale również abstrakcyjnych, syntetycznych obiektów i danych, które nie mają wewnętrznej geometrii, tak jak wyniki pomiarów. Dalej, nie musimy ograniczać się do obrazów statycznych. Chociaż obrazy statyczne są dobrym środkiem przekazywania informacji, często obrazy zmieniające się dynamicznie są znacznie efektywniejsze, zwłaszcza dla zjawisk zmiennych w czasie (na przykład wygięcie skrzydła samolotu w locie ponaddźwiękowym albo ewolucja twarzy ludzkiej od dzieciństwa do wieku starczego), zarówno rzeczywistych jak i abstrakcyjnych (na przykład trendy wzrostu, takie jak zużycie energii jądrowej w Stanach Zjednoczonych albo przemieszczanie się ludności z miast do przedmieść i z powrotem do miast).

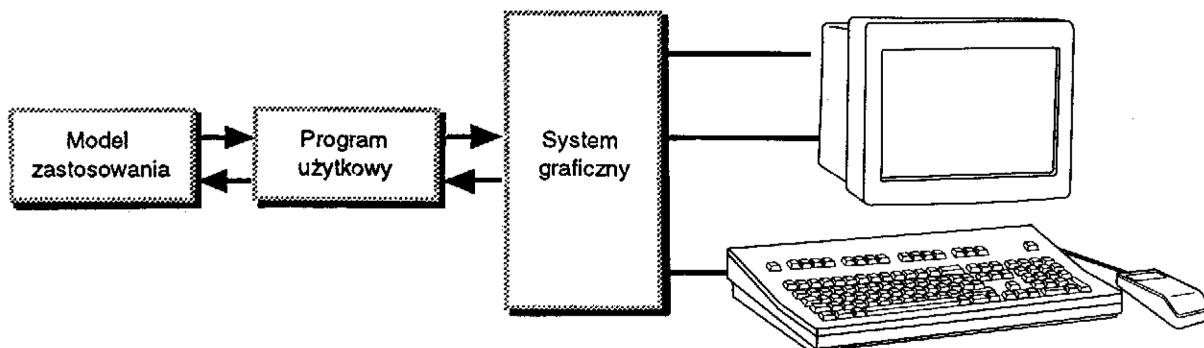
Jeżeli jest możliwe przedstawianie *dynamiki ruchu*, to obiekty mogą być przesuwane i obracane wokół nieruchomego obserwatora. Również obiekty mogą pozostać nieruchome, a obserwator może się poruszać wokół nich. Można także przesuwać kamerę i wybierać odpowiedni fragment pola wizualizacji oraz robić albo zbliżenie, albo oddalenie kamery po to, żeby uzyskać więcej albo mniej szczegółów, tak jak przy obserwacji przez wizjer szybko poruszającej się kamery wideo. W wielu przypadkach porusza się i obiekt i kamera. Typowy przykład to symulator lotu (fot. 4a i 4b), w którym mechaniczna platforma imitująca kokpit jest połączona z ekranami do wyświetlania widoków widzianych przez okna kokpitu. Komputer steruje ruchem platformy, przyrządami pomiarowymi i symulowanym światem ze stacjonarnymi i ruchomymi obiektami. W lunaparkach w różnych symulatorach można podróżować po różnych symulowanych ziemskich i pozaziemskich terenach. Salony gier oferują gry zręcznościowe wykorzystujące grafikę oraz wyścigi samochodowe w symulatorach, a także gry wideo wykorzystujące interakcyjny ruch (fot. 5).

Uaktualnianie dynamiki ruchu polega na zmienianiu kształtu, barwy i innych własności oglądanych albo modelowanych obiektów. Na przykład system może wyświetlać odkształcenia struktury samolotu w czasie lotu albo zmiany stanów w schemacie blokowym reaktora jądrowego w odpowiedzi na sterowanie przez operatora różnymi mechanizmami reprezentowanymi w postaci graficznej. Im zmiany są płynniejsze, tym bardziej realistyczny i wartościowy jest wynik.

Interakcyjna grafika komputerowa umożliwia wielostronną współpracę człowieka z komputerem. Taka współpraca w istotny sposób zwiększa naszą zdolność rozumienia danych, wychwytywania tendencji i wizualizowania rzeczywistych i nierzeczywistych obiektów.

1.4. Ogólny schemat grafiki interakcyjnej

Pokazany na rys. 1.11 ogólny schemat może być wykorzystany do niemal każdego interakcyjnego systemu graficznego. Na poziomie sprzętu (nie pokazanym bezpośrednio na rysunku) komputer odbiera sygnały wejściowe od współdziałających urządzeń i wysyła obrazy do urządzenia wyświetlającego. Oprogramowanie ma trzy elementy składowe. Pierwszy to *program użytkowy*, który tworzy informacje, zapamiętuje je w drugim elemencie i odzyskuje je od niego; drugi element to *model zastosowania*, który reprezentuje dane albo obiekty, jakie mają być wyświetlone na ekranie. Program użytkowy obsługuje również wejście użytkownika. Ten program tworzy obrazy dzięki wysyłaniu do trzeciego elementu, którym jest *system graficzny*, ciągów wyjściowych poleceń



Rys. 1.11. Ogólny schemat systemu grafiki interakcyjnej

graficznych, które zawierają zarówno szczegółowy opis geometryczny tego co ma być wyświetlone, jak i atrybuty opisujące wygląd obiektu. System graficzny jest odpowiedzialny za faktyczne utworzenie obrazu na podstawie szczegółowych opisów i za przekazanie wejścia użytkownika do przetwarzania przez program użytkowy.

System graficzny jest więc pośrednikiem między programem użytkowym a sprzętem wyświetlającym; system wykonuje przekształcenia wyjściowe: od obiektów w modelu zastosowania do obrazu modelu. Symetrycznie, wykonuje on przekształcenia wejściowe: od akcji użytkownika do wejścia do programu użytkowego, który spowoduje wykonanie zmian w modelu albo obrazie. Podstawowym zadaniem projektanta programu użytkowego z interakcyjną grafiką jest określenie klas danych wejściowych albo obiektów, które mają być generowane i reprezentowane obrazowo, oraz określenie, jak ma przebiegać interakcja między użytkownikiem a programem użytkowym przy tworzeniu i modyfikowaniu modelu i jego wizualnej reprezentacji. Większość zadań programisty koncentruje się raczej na tworzeniu i edycji modelu oraz obsłudze interakcji użytkownika niż na faktycznym tworzeniu obrazów, ponieważ to zadanie jest wykonywane przez system graficzny.

1.4.1. Model zastosowania

Pojęcie modelu zastosowania obejmuje wszystkie dane i obiekty jak również zależności między nimi, które są istotne dla części programu użytkowego związanej z wyświetlaniem i interakcją oraz dla wszystkich niegraficznych modułów przetwarzania końcowego. Przykładami takich modułów przetwarzania końcowego są: analiza stanów przejściowych układu albo naprężeń w skrzydle samolotu, symulacja modelu

populacji albo systemu pogody i obliczanie kosztorysów budów. W klasie zastosowań typowych dla programów malarskich takich jak MacPaint i PCPaint, przeznaczeniem programu jest tworzenie obrazu przez użytkownika, który ma możliwość ustawiania lub modyfikowania pikseli. Tutaj bezpośredni model zastosowania nie jest potrzebny – obraz jest zarówno środkiem, jak i efektem końcowym, a wyświetlana mapa bitowa albo pikselowa służy jako model zastosowania.

Częściej jednak można zidentyfikować model zastosowania reprezentujący obiekty użytkowe za pomocą kombinacji danych oraz opisu proceduralnego, który jest niezależny od określonego urządzenia wyświetlającego. Opisy proceduralne są używane na przykład do definiowania fraktali, tak jak to opisano w p. 9.5.1. Model danych może być tak elementarny jak tablica punktów danych albo tak złożony jak powiązana lista reprezentująca strukturę sieci danych albo relacyjną bazę danych pamiętającą zbiór relacji. Często mówi się o pamiętaniu *modelu zastosowania* w *bazie danych zastosowania*; tutaj używamy tych pojęć zamiennie. Modele na ogół pamiętają opisy prymitywów (punkty, odcinki, krzywe i wielokąty w 2D i 3D oraz wielościany i powierzchnie w 3D), które definiują kształty elementów obiektów; *atrybuty* obiektów, takie jak styl linii, barwa albo tekstura powierzchni; relacje *spójności* i rozmieszczenie danych, które opisują sposób łączenia elementów ze sobą.

W modelu zastosowania danym geometrycznym często towarzyszą informacje o właściwościach niegeometrycznych, tekstowe albo numeryczne, użyteczne dla programu przetwarzania końcowego albo dla współpracującego użytkownika. W zastosowaniach CAD-owskich takimi przykładami są dane produkcyjne; dane o cenach i dostawcach; właściwości termiczne, mechaniczne, elektryczne albo elektroniczne; oraz mechaniczne albo elektroniczne tolerancje.

1.4.2. Wyświetlanie modelu

Program użytkowy tworzy model zastosowania albo w wyniku wcześniejszych obliczeń, jak to ma miejsce w symulacji inżynierskiej lub naukowej na superkomputerach, albo w wyniku interakcyjnej sesji przy urządzeniu wyświetlającym, w czasie której użytkownik kieruje procesem konstruowania krok po kroku, wybierając elementy i właściwości geometryczne i niegeometryczne. W dowolnym czasie użytkownik może zażądać, żeby program użytkowy pokazał widok dotychczas utworzonego modelu. (Wyraz widok został tutaj użyty świadomie, zarówno w sensie wizualnego renderingu właściwości geometrycznych modelowanych obiektów, jak i w technicznym sensie bazodanowym prezentacji 2D właściwości podzbioru modelu.)

Modele zależą od zastosowania i mogą być tworzone niezależnie od systemu wyświetlania. Dlatego program użytkowy musi dokonać konwersji opisu części modelu, która ma być pokazana, z wewnętrznej reprezentacji geometrii (zapamiętanej bezpośrednio w modelu albo uzyskanej na bieżąco) na odpowiednie procedury albo polecenia wykorzystywane przez system graficzny przy tworzeniu obrazu. Ten proces konwersji ma dwie fazy. Po pierwsze, program użytkowy korzystając z pewnych kryteriów wyboru albo pytań przeszukuje bazę danych zastosowania, w której jest zapamiętany model w celu wybrania części, które mają być pokazane. Po drugie, wybrane dane albo geometria, plus atrybuty są zamieniane na format, który może być wysłany do systemu graficznego. Kryteria wyboru mogą być geometryczne (na przykład część modelu, która ma być pokazana, została przesunięta za pomocą graficznego odpowiednika przesuwania albo powiększania za pomocą kamery) albo też mogą być podobne do tradycyjnych kryteriów wyszukiwania w bazie danych.

Dane uzyskane w czasie przeszukiwania bazy danych muszą być albo danymi geometrycznymi, albo muszą być zamienione na dane geometryczne; dane mogą być przedstawione systemowi graficznemu albo w postaci prymitywów, które system może wyświetlić bezpośrednio, albo atrybutów, które określają wygląd prymitywów. Prymitywy wyświetlania na ogół odpowiadają prymitywom zapamiętanym w modelach geometrycznych: odcinkom, prostokątom, wielokątom, okręgom, elipsom i tekstom w 2D oraz wielokątom, wielościanom i tekstom w 3D.

System graficzny na ogół składa się ze zbioru wyjściowych podprogramów, odpowiadających różnym prymitywom, atrybutom i innym elementom. Są one zebrane w *bibliotece podprogramów graficznych* albo w *pakiecie*, który może być wywołany z języków wyższego poziomu, np. C, Pascal albo LISP. Program użytkowy określa prymitywy geometryczne i atrybuty potrzebne dla tych podprogramów, a te z kolei sterują odpowiednim urządzeniem wyświetlającym i powodują wyświetlenie obrazu. Podobnie jak konwencjonalne systemy wejścia/wyjścia tworzą logiczne jednostki wejścia/wyjścia po to, żeby uchronić programistę przed wieloma szczegółami dotyczącymi sprzętu i programów sterujących urządzeń, systemy graficzne tworzą *logiczne urządzenia wyświetlające*. Ta abstrakcja urządzeń wyświetlających odnosi się zarówno do wyprowadzania obrazów, jak i do interakcji za pomocą logicznych urządzeń wejściowych. Na przykład myszka, tabliczka, ekran dotykowy, drążek sterowniczy 2D albo kula śledząca mogą być traktowane jako logiczne wejściowe urządzenie wskazujące, które podaje współrzędne (x, y) na ekranie. Program użytkowy może zażądać od systemu graficznego albo podania próbki z urządzenia wejściowego (to znaczy zapytać o bieżącą wartość) albo czekać w określonym miejscu, aż wystąpi zdarzenie polegające na aktywacji urządzenia przez użytkownika.

1.4.3. Obsługa interakcji

W programie użytkowym typowym schematem obsługi interakcji jest *pętla sterowana zdarzeniami*. Można ją przedstawić jako automat skończony z centralnym stanem oczekiwania i przejściami do innych stanów wywoływanych przez zdarzenia będące wynikiem akcji użytkownika. Do przetwarzania polecenia mogą być potrzebne zagnieżdżone pętle zdarzeń, mające taki sam format, swoje własne stany i przejścia wywoływane przez wejście. Program użytkowy może również próbkować urządzenia wejściowe typu lokalizator, pytając w dowolnej chwili o ich wartości; otrzymaną wartość program traktuje z kolei jako wejście do procedury przetwarzania, która również zmienia stan programu użytkowego, obraz albo bazę danych. Pętlę sterowaną zdarzeniami charakteryzuje następujący schemat zapisany w postaci pseudokodu:

```
generowanie początkowego obrazu na podstawie modelu zastosowania
do {
  możliwość wyboru poleceń albo obiektów
  /* Program jest w stanie oczekiwania na akcję użytkownika */
  wait oczekiwanie na akcję użytkownika
  switch ( rozejście zależnie od akcji ) {
     obsługa wybranej akcji, uaktualnienie modelu i ekranu,
     jeżeli jest to konieczne
  }
}
while ( !quit )  /* Użytkownik nie wybrał opcji „quit" */
```

Zanalizujmy dokładniej reakcję programu użytkowego na informację z wejścia. Na ogół program użytkowy odpowiada na akcję użytkownika na jeden z dwóch sposobów. Akcja użytkownika może wymagać tylko uaktualnienia ekranu – na przykład system może odpowiedzieć podświetlając wybrany obiekt albo udostępniając nowe menu. Wtedy program użytkowy musi tylko uaktualnić swój wewnętrzny stan i wywołać pakiet graficzny po to, żeby uaktualnić ekran; nie ma potrzeby uaktualniania bazy danych. Jeżeli jednak akcja użytkownika wymaga zmiany modelu – na przykład dodania albo usunięcia elementu – to program użytkowy musi uaktualnić model, a następnie wywołać program graficzny w celu uaktualnienia ekranu na podstawie modelu. W celu zregenerowania obrazu od razu jest przeszukiwany cały model albo dla bardziej wyrafinowanych algorytmów przyrostowo-uaktualniających ekran jest uaktualniany selektywnie. Trzeba pamiętać, że na ekranie nie może nastąpić żadna istotna zmiana obiektu bez odpowiedniej zmiany w modelu. W istocie ekran jest oknem komputera, w którym w ogólnym przypadku, użytkownik manipuluje raczej

modelem niż obrazem, przy czym model formalnie i symbolicznie jest poza obrazem. Jedynie w zastosowaniach malarskich i związanych z ulepszaniem obrazu model i obraz są identyczne. Interpretacja wejścia użytkownika należy do programu użytkowego. System graficzny nie odpowiada za tworzenie albo modyfikowanie modelu ani początkowo, ani w odpowiedzi na akcję użytkownika; jego jedynym zadaniem jest tworzenie obrazu na podstawie opisu geometrycznego i przekazywanie danych wejściowych użytkownika.

Model z pętlą zdarzenia jest podstawowym modelem w obecnej praktyce grafiki komputerowej; niemniej jest on ograniczony w tym sensie, że dialog użytkownik-komputer jest sekwencyjnym modelem typu ping-pong, w którym na zmianę zachodzi akcja użytkownika i reakcja komputera. W przyszłości możemy się spodziewać konwersacji równoległych, w których będzie następowało równoczesne wprowadzanie i wyprowadzanie przy wykorzystaniu licznych kanałów komunikacyjnych – na przykład zarówno grafiki jak i głosu. Formalizm dla takiej swobodnej konwersacji, nie wspominając już o konstrukcjach języków programowania, jeszcze nie został dobrze opracowany i dalej nie będziemy rozwijali tego wątku.

Podsumowanie

Standardowym środkiem interakcji użytkownik-komputer stały się interfejsy graficzne, które zastąpiły interfejsy tekstowe. Grafika stała się również kluczową technologią przekazywania pomysłów, danych i trendów w biznesie, nauce, inżynierii i edukacji. Korzystając z grafiki możemy utworzyć sztuczne (albo wirtualne) światy, które mogą służyć jako komputerowe środowiska badawcze do badania obiektów i zjawisk w naturalny i intuicyjny sposób, wykorzystujący nasze wysoce rozwinięte umiejętności wzrokowego rozpoznawania obrazów.

Do końca lat osiemdziesiątych przeważająca liczba zastosowań grafiki komputerowej dotyczyła obiektów 2D; zastosowania 3D były dość rzadkie, ponieważ oprogramowanie dla zastosowań 3D jest znacznie bardziej złożone niż w przypadku 2D oraz ze względu na to, że do uzyskania obrazów pseudorealistycznych potrzebne są duże moce obliczeniowe. Do niedawna interakcja użytkownika w czasie rzeczywistym z modelami 3D i obrazami pseudorealistycznymi była możliwa jedynie na bardzo drogich, wysoce wydajnych stacjach roboczych z dedykowanym specjalizowanym sprzętem graficznym. Spektakularny postęp w technologii półprzewodnikowych układów VLSI, który spowodował pojawienie się tanich mikroprocesorów i pamięci, doprowadził na początku lat osiemdziesiątych do utworzenia interfejsów dla komputerów

osobistych wykorzystujących grafikę 2D z mapami bitowymi. Ta sama technologia umożliwiła, niecałą dekadę później, stworzenie podsystemów zawierających jedynie kilka układów VLSI, które umożliwiają uzyskanie animacji 3D w czasie rzeczywistym, z barwnymi obrazami złożonych obiektów, na ogół opisywanych za pomocą tysięcy wielokątów. Takie podsystemy mogą być dodawane jako akceleratory 3D do stacji roboczych albo nawet do komputerów osobistych, które wykorzystują zwykłe mikroprocesory. Jest oczywiste, że można się spodziewać eksplozji zastosowań 3D, podobnie jak to wystąpiło w przypadku zastosowań 2D. Zagadnienia takie jak fotorealistyczny rendering, które jeszcze niedawno wydawały się zupełnie egzotyczne, są obecnie dobrze znane i standardowo dostępne w oprogramowaniu graficznym i coraz częściej w sprzęcie grafiki.

Wiele zadań tworzenia efektywnej komunikacji graficznej, 2D czy też 3D, jest związanych z modelowaniem obiektów, których obrazy chcemy utworzyć. System graficzny działa jako pośrednik między modelem zastosowania a urządzeniem wyjściowym. Program użytkowy jest odpowiedzialny za tworzenie i uaktualnianie modelu bazującego na interakcji użytkownika; system graficzny wykonuje dobrze określoną, w większości rutynową, część pracy przy tworzeniu widoków obiektów i przekazywaniu zdarzeń użytkownika do programu użytkowego. Wzrastająca liczba pozycji literatury na temat różnych modeli wykorzystujących opis fizyczny pokazuje, że grafika rozwija się w kierunku, który będzie wykraczał poza rendering i obsługę interakcji. Obrazy i animacja już nie są tylko ilustracją w nauce i inżynierii – stały się częścią nauki i techniki i mają wpływ na codzienną pracę naukowca i inżyniera.

Zadania

1.1. Wymień wszystkie interakcyjne programy graficzne wykorzystywane w codziennej pracy do: pisania, wykonywania obliczeń, robienia wykresów, programowania, uruchamiania itd. Które z tych programów mogłyby tak samo dobrze pracować na terminalu alfanumerycznym? Które byłyby prawie bezużyteczne bez funkcji graficznych? Uzasadnij odpowiedź.

1.2. W odniesieniu do interfejsów użytkownika w programach graficznych często używa się określenia zobacz i wykonaj. Wymień główne elementy – takie jak ikony, okna, paski do przewijania i menu – które są związane z określeniem „zobacz” występujące w interfejsie graficznym ulubionego programu przetwarzania tekstów albo zarządzania oknami. Wymień funkcje graficzne potrzebne do realizowania tych elementów. Jak można wykorzystać barwę i opis 3D do realizacji cechy „zobacz”?

W jaki sposób można z „zaśmieconego biura" uczynić lepszą metaforę dla organizowania dostępu do informacji, niż to występuje w przypadku „zawalonego biurka"?

1.3. W tym samym duchu jak w zadaniu 1.2 zastanów się, jakie możliwości mogłyby dać dynamiczne ikony w sensie rozszerzenia, a nawet zastąpienia ikon statycznych z obecnej metafory biurka.

1.4. W ulubionym graficznym programie użytkowym wydziel główne moduły korzystając z modelu pokazanego na rys. 1.11 jako wskazówki. Jaka część zastosowania ma związek z grafiką? Jaka część ma związek z tworzeniem struktur danych i ich obsługą? Jaka część ma związek z obliczeniami, takimi jak symulacja?

1.5. Pojęcia *symulacja* i *animacja* są w grafice komputerowej często używane razem, a nawet zamiennie. Jest to naturalne wówczas, gdy wizualizujemy zmiany zachowania (albo strukturalne) w czasie w systemie fizycznym albo abstrakcyjnym. Podaj trzy przykłady systemów, dla których można zastosować taką wizualizację. Podaj formę symulacji, jakiej należałoby użyć, oraz sposób jej realizacji. Podaj przykład, który pokaże różnice między symulacją a animacją.

1.6. Jako wariant zadania 1.5 stwórz projekt na wysokim poziomie graficznego środowiska badawczego dla nietrywialnego problemu naukowego, matematycznego albo technicznego. Omów, jak będzie wyglądała sekwencja interakcji i jakie możliwości powinien mieć użytkownik, aby wykonać eksperymenty.

1.7. Nie zaglądając do rozdz. 3 zaproponuj prosty algorytm rysowania w pierwszej ćwiartce odcinka o nachyleniu nie większym niż 45°.

1.8. Aliasing jest poważnym problemem, który tworzy nieprzyjemne albo nawet mylące artefakty wizualne. Wymień sytuacje, kiedy te artefakty mają znaczenie, oraz takie, kiedy nie są one istotne. Omów różne sposoby minimalizowania wpływu zakłóceń i wyjaśnij, z jakimi kosztami mogą się one wiązać.

2. Pakiet SRGP

W rozdziale 1 pokazaliśmy, że monitory wektorowe i rastrowe to dwie zasadniczo różne technologie tworzenia obrazów na ekranie. Wyświetlanie rastrowe jest obecnie dominującą technologią, ze względu na kilka cech istotnych dla większości nowoczesnych zastosowań. Po pierwsze, monitory rastrowe umożliwiają wypełnienie obszarów jednolitą barwą albo powtarzalnymi wzorami o jednej lub więcej barwach; monitory wektorowe mogą w najlepszym przypadku symulować wypełnianie powierzchni za pomocą równoległych wektorów narysowanych blisko siebie. Po drugie, monitory rastrowe pamiętają informacje w sposób, który umożliwia manipulowanie na najniższym poziomie: można odczytać i zapisać piksele i można kopiować i przesuwać dowolne fragmenty obrazu.

Pierwszy omawiany pakiet graficzny SRGP (Simple Raster Graphics Package) jest niezależny od urządzenia i wykorzystuje możliwości techniki rastrowej. Zestaw prymitywów graficznych w SRGP (odcinki, prostokąty, okręgi i elipsy oraz ciągi tekstowe) jest podobny do zestawu w popularnym pakiecie rastrowym QuickDraw firmy Macintosh i pakiecie Xlib systemu X Window. Z drugiej strony funkcje obsługi interakcji w SRGP są podzbiorem pakietu graficznego wysokiego poziomu SPHIGS umożliwiającego wyświetlanie prymitywów 3D (omawianego w rozdz. 7). SPHIGS (Simple PHIGS) jest uproszczonym dialektem standardowego pakietu graficznego PHIGS (Programmer's Hierarchical Interactive Graphics System) zaprojektowanego dla sprzętu rastrowego i wektorowego. Chociaż SRGP i SPHIGS zostały napisane specjalnie dla potrzeb tej książki, zachowują wiele z koncepcji głównych pakietów graficznych i większość tego, czego się tutaj nauczymy, jest do

natychmiastowego zastosowania w komercyjnych pakietach. W tej książce omawiamy ogólnie oba te pakiety; pełniejszy opis można znaleźć w podręcznikach dostarczanych z pakietami programów.

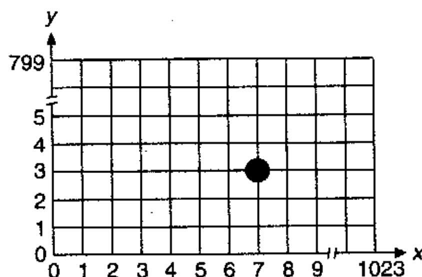
Naszą dyskusję zaczynamy od pakietu SRGP i od omówienia operacji wykonywanych przez program użytkowy przy rysowaniu na ekranie: specyfikacji prymitywów i atrybutów, które wpływają na ich wygląd. (Ponieważ drukarki graficzne przedstawiają informacje w zasadzie tak samo jak monitory rastrowe, nie będziemy zatem się nimi teraz zajmować; wrócimy do nich w rozdz. 4, gdy będzie mowa o sprzęcie.) Następnie nauczymy się, jak korzystając z funkcji wejścia w SRGP zapewnić programowi użytkowemu interakcyjność. Potem zajmiemy się możliwościami manipulowania pikselami, dostępnymi tylko w monitorach rastrowych. Zakończymy dyskusją na temat pewnych ograniczeń pakietów interakcyjnej grafiki rastrowej, na przykład SRGP.

Chociaż w naszej dyskusji zakładamy, że SRGP kontroluje cały ekran, pakiet został tak zaprojektowany, żeby mógł działać w środowisku okien (por. rozdz. 10) i sterować wnętrzem okna, tak jak gdyby było ono wirtualnym ekranem. Dlatego programista nie musi się zajmować szczegółami związanymi z pracą pod kontrolą programu zarządzania oknami.

2.1. Rysowanie za pomocą pakietu SRGP

2.1.1. Określanie prymitywów graficznych

Rysowanie za pomocą całkowitoliczbowych pakietów grafiki rastrowej, na przykład SRGP, to tak jak rysowanie wykresów na papierze z bardzo drobną siatką. Gęstość siatki zmienia się od 80 do 120 punktów na cal w konwencjonalnym monitorze, do 300 lub więcej w monitorze o dużej rozdzielczości. Im większa jest rozdzielczość, tym lepiej wyglądają drobne szczegóły. Na rysunku 2.1 pokazano ekran monitora (albo



Rys. 2.1. Kartezjański układ współrzędnych ekranu o szerokości 1024 i wysokości 800 pikseli. Pokazano piksel (7, 3)

powierzchnię papieru drukarki lub filmu) narysowany w całkowitoliczbowym kartezjańskim układzie współrzędnych SRGP. Zauważmy, że w SRGP piksele leżą na przecięciach linii siatki.

Początek układu (0, 0) jest na dole z lewej strony ekranu; wartości x wzrastają w kierunku na prawo, a wartości y – w kierunku ku górze. Piksel w górnym prawym rogu ma współrzędne (szerokość – 1, wysokość – 1), przy czym szerokość i wysokość określają wielkość ekranu i zależą od konkretnego urządzenia.

Na papierze milimetrowym możemy narysować ciągłą linię między dwoma dowolnymi punktami; na monitorach rastrowych możemy jednak rysować odcinki tylko między punktami siatki i odcinki muszą być aproksymowane przez wybrane punkty siatki – piksele – należące do odcinka lub leżące blisko niego. Podobnie figury, np. wypełnione wielokąty albo okręgi, są tworzone przez wybranie pikseli należących do ich wnętrz i brzegów. Ponieważ określanie każdego piksela odcinka albo zamkniętej figury byłoby zbyt uciążliwe, pakiety graficzne umożliwiają programiście określenie prymitywów takich jak odcinki lub wielokąty za pomocą ich wierzchołków; pakiet później uzupełnia brakujące szczegóły za pomocą odpowiednich algorytmów rasteryzacji omawianych w rozdz. 3.

SRGP zawiera podstawowy zestaw prymitywów: odcinki, wielokąty, okręgi i elipsy oraz tekst[1]. W celu określenia prymitywu program użytkowy wysyła współrzędne definiujące kształt prymitywu do odpowiedniej funkcji SRGP generowania prymitywów. Dopuszcza się, żeby określony punkt leżał poza prostokątnym obszarem ekranu; oczywiście widoczne będą tylko te części prymitywu, które leżą wewnątrz ekranu.

Korzystamy z języka ANSI C z następującymi konwencjami drukarskimi. Słowa kluczowe C, typy wbudowane i typy definiowane przez użytkownika są pogrubione. Zmienne używane w tekście są pisane kursywą. Stałe symboliczne są pisane dużymi literami. Komentarze są wewnątrz ograniczników /*...*/, a pseudokod jest pisany kursywą. Ze względu na zwięzłość deklaracje stałych i zmiennych są pomijane tam, gdzie nie budzi to wątpliwości. Zmienne boolowskie są typu **unsigned char**, przy czym TRUE i FALSE są zdefiniowane odpowiednio jako 1 i 0. Po zdefiniowaniu nowy typ na przykład **point** jest używany w dalszych fragmentach kodu bez ponownego wprowadzania.

Odcinki i łamane. Następująca funkcja SRGP rysuje odcinek od ($x1$, $y1$) do ($x2$, $y2$):

```
void SRGP_lineCoord ( int x1, int y1, int x2, int y2 );
```

[1] Specjalne funkcje, które rysują piksel albo tablicę pikseli są opisane w podręczniku SRGP.

Jeżeli chcemy narysować odcinek między (0, 0) a (100, 300), to po prostu wywołujemy

```
SRGP_lineCoord (0, 0, 100, 300);
```

Ponieważ często bardziej naturalne jest operowanie końcami odcinków niż współrzędnymi x i y, SRGP ma inną funkcję rysowania odcinka:

```
void SRGP_line ( point pt1, point pt2 );
```

Tutaj „point" jest typem strukturalnym, który zawiera dwie liczby całkowite określające wartości współrzędnych x i y punktu:

```
typedef struct {
   int x, y;
} point;
```

Sekwencja odcinków łączących kolejne wierzchołki jest nazywana *łamaną*. Chociaż łamaną można utworzyć powtarzając wywołania funkcji rysowania odcinka, SRGP zawiera je jako specjalny przypadek. Są dwie funkcje rysowania łamanych, podobnie do funkcji rysowania odcinka z określaniem współrzędnych i punktów. Parametrami są tutaj tablice:

```
void SRGP_polyLineCoord ( int vertexCount, vertexCoordinateList xArray,
        vertexCoordinateList yArray);
void SRGP_polyLine ( int vertexCount, vertexList vertices );
```

przy czym *vertexCoordinateList* (lista współrzędnych wierzchołków) i *vertexList* (lista wierzchołków) są odpowiednio tablicami liczb całkowitych i punktów.

Pierwszy parametr w obu tych wywołaniach określa, ilu wierzchołków SRGP ma się spodziewać. W pierwszym wywołaniu drugi i trzeci parametr są tablicami par wartości całkowitych x i y, a łamana jest rysowana od wierzchołka (xArray[0], yArray[0]) do wierzchołka (xArray[1], yArray[1]), do wierzchołka xArray[2], yArray[2]) itd. Ta forma jest wygodna na przykład przy rysowaniu danych w standardowym układzie współrzędnych, gdzie *xArray* jest wcześniej określonym zbiorem wartości zmiennej niezależnej, a *yArray* jest zbiorem wartości obliczanych albo wprowadzanych przez użytkownika.

Rys. 2.2. Wykres dla danych zapisanych w tablicy

Dla przykładu narysujmy wykres dla programu analizy ekonomicznej, który oblicza dane o trendach miesięcznych i zapamiętuje je w 12-pozycyjnej tablicy danych całkowitych *balanceOfTrade* (bilans obrotów). Zaczniemy nasz wykres w punkcie (200, 200). Żeby można było obserwować różnice między kolejnymi punktami, będziemy je rysowali w odległościach co 10 pikseli na osi x. Utworzymy całkowitoliczbową tablicę *months* (miesiące), która będzie reprezentowała 12 miesięcy i której pozycje będą zawierały potrzebne wartości x: 200, 210, ..., 310.

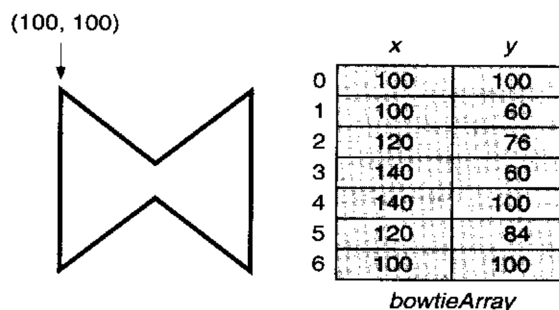
Każdą wartość w tablicy danych musimy zwiększyć o 200, żeby umieścić dwanaście współrzędnych y na właściwym miejscu. Następnie rysujemy wykres jak na rys. 2.2, korzystając z następującego kodu:

```
/* Rysowanie osi */
SRGP_lineCoord (175, 200, 320, 200);
SRGP_lineCoord (200, 140, 200, 280);

/* Rysowanie danych */
SRGP_polyLineCoord (12, months, balanceOfTrade);
```

Możemy użyć drugiej formy rysowania łamanej, określając pary x i y jako punkty i przekazując tablicę takich punktów do SRGP. Muszkę z rys. 2.3 możemy narysować wywołując

```
SRGP_polyLine (7, bowtieArray);
```

Tablica pokazana na rys. 2.3 ilustruje sposób definiowania tablicy bowtieArray.



	x	y
0	100	100
1	100	60
2	120	76
3	140	60
4	140	100
5	120	84
6	100	100

Rys. 2.3. Rysowanie łamanej

Znaczniki i ciągi znaczników. Często jest wygodnie umieszczać znaczniki (na przykład kropki, gwiazdki albo kółeczka) na wykresie w punktach reprezentujących dane. SRGP oferuje odpowiednie funkcje. Następująca funkcja utworzy symbol znacznika o środku w punkcie (x, y):

```
void SRGP_markerCoord ( int x, int y );
void SRGP_marker ( point pt );
```

Styl i wielkość znacznika można zmieniać; wyjaśnimy to w p. 2.1.2. Aby utworzyć ciąg identycznych znaczników dla zbioru punktów, wywołujemy albo

```
void SRGP_polyMarkerCoord( int vertexCount, vertexCoordinateList xArray,
    vertexCoordinateList yArray );
```

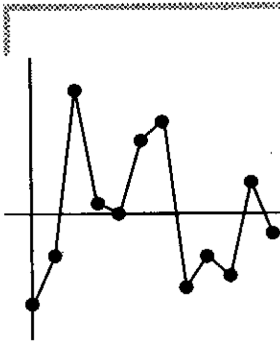
albo

```
void SRGP_polyMarker ( int vertexCount, vertexList vertices );
```

Dodatkowe wywołanie

```
SRGP_polyMarkerCoord (12, months, balanceOfTrade);
```

spowoduje dodanie znaczników do wykresu z rys. 2.2; otrzymamy wykres pokazany na rys. 2.4.



Rys. 2.4. Wykres dla danych zapisanych w tablicy z wykorzystaniem znaczników

Wielokąty i prostokąty. Aby narysować obwód wielokąta, możemy albo określić łamaną zamkniętą, w której pierwszy i ostatni wierzchołek są identyczne (tak jak to zrobiliśmy rysując muszkę z rys. 2.3), albo możemy użyć następującego specjalnego wywołania SRGP:

```
void SRGP_polygon ( int vertexCount, vertexList vertices );
```

Tutaj obwód zostaje automatycznie zamknięty dzięki narysowaniu odcinka od ostatniego wierzchołka do pierwszego. Aby narysować muszkę z rys. 2.3 jako wielokąt, korzystamy z wywołania

```
SRGP_polygon (6, bowtieArray);
```

przy czym bowtieArray jest tablicą zawierającą tylko sześć punktów.

Dowolny prostokąt można określić jako wielokąt o czterech wierzchołkach; prostokąt o bokach równoległych do boków ekranu można określić również za pomocą prymitywu *rectangle*, który wymaga podania tylko dwóch wierzchołków (dolny lewy róg i górny prawy róg):

```
void SRGP_rectangleCoord ( int leftX, int bottomY, int rightX, int topY );
void SRGP_rectanglePt ( point bottomLeft, point topRight );
void SRGP_rectangle (rectangle rect );
```

Struktura *rectangle* pamięta rogi lewy dolny i prawy górny:

```
typedef struct {
   point bottomLeft, topRight;
} rectangle;
```

Prostokąt o szerokości 101 pikseli i wysokości 151 pikseli i bokach równoległych do boków ekranu można narysować za pomocą wywołania:

```
SRGP_rectangleCoord (50, 25, 150, 175);
```

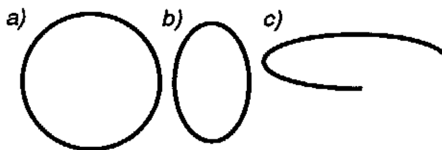
SRGP daje następujące narzędzia do tworzenia prostokątów i punktów na podstawie danych o współrzędnych

```
point SRGP_defPoint ( int x, int y );
rectangle SRGP_defRectangle ( int leftX, int bottomY, int rightX, int topY );
```

Nasz przykładowy prostokąt można więc narysować w następujący sposób:

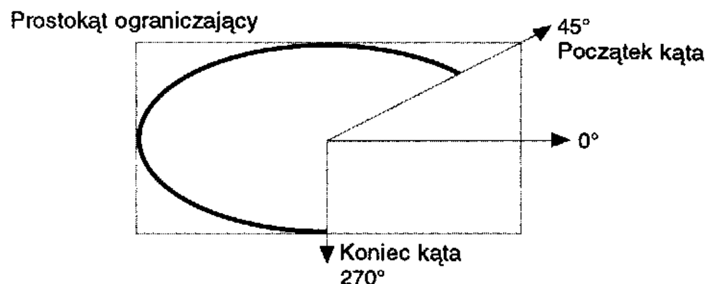
```
rect = SRGP_defRectangle ( 50, 25, 150, 175 );
SRGP_rectangle (rect);
```

Okręgi i elipsy. Na rysunku 2.5 pokazano łuki kołowe i eliptyczne narysowane przez SRGP. Ponieważ okręgi są specjalnymi przypadkami elips, dla wszystkich tych form łuków, kołowych i eliptycznych, otwartych i zamkniętych będziemy używali pojęcia *łuk eliptyczny*. SRGP może rysować tylko standardowe elipsy, których osie mała i wielka są równoległe do osi współrzędnych.



Rys. 2.5. Rysowanie łuków eliptycznych: a) kołowy, b) zamknięty eliptyczny; c) eliptyczny

Chociaż jest wiele metod, matematycznie równoważnych, specyfikowania łuków eliptycznych, dla programisty jest wygodnie określać łuki za pomocą opisanych na nich prostokątów o bokach równoległych do osi układu współrzędnych (rys. 2.6); takie prostokąty są określane jako *prostokąty ograniczające*.



Rys. 2.6. Specyfikowanie łuków eliptycznych

Ogólna postać funkcji dla elipsy jest następująca

```
void SRGP_ellipseArc ( rectangle extentRect, float startAngle, float endAngle );
```

Szerokość i wysokość prostokąta ograniczającego określają kształt elipsy. To, czy łuk jest zamknięty, zależy od pary kątów, które określają, gdzie łuk zaczyna się i kończy. Dla wygody każdy kąt jest mierzony w *stopniach prostokątnych*, które są liczone w kierunku przeciwnym

względem ruchu wskazówek zegara, przy czym 0° odpowiada dodatniej części osi x, 90° dodatniej części osi y, a 45° „przekątnej" wychodzącej z początku układu współrzędnych w kierunku górnego rogu prostokąta. Oczywiście stopnie prostokątne są równoważne zwykłym stopniom kołowym jedynie wówczas, gdy prostokąt ograniczający jest kwadratem. Ogólna zależność między kątami prostokątnymi a kołowymi ma postać

$$\textit{kąt prostokątny} = \operatorname{arctg}\left(\operatorname{tg}(\textit{kąt kołowy}) \cdot \frac{\textit{szerokość}}{\textit{wysokość}}\right) + \textit{poprawka}$$

przy czym kąty są w radianach i

$$\textit{poprawka} = 0 \text{ dla } 0 \leqslant \textit{kąt kołowy} < \frac{\pi}{2}$$

$$\textit{poprawka} = \pi \text{ dla } \frac{\pi}{2} \leqslant \textit{kąt kołowy} < \frac{3\pi}{2}$$

$$\textit{poprawka} = 2\pi \text{ dla } \frac{3\pi}{2} \leqslant \textit{kąt kołowy} < 2\pi$$

2.1.2. Atrybuty

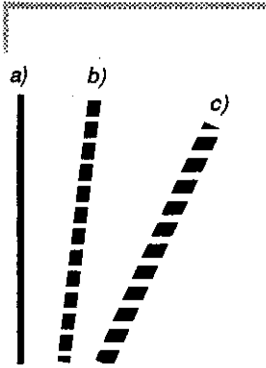
Styl linii i szerokość linii. Wygląd prymitywu można zmieniać za pomocą jego atrybutów[2]. W SRGP do określania wyglądu odcinków, łamanych, wielokątów, prostokątów i łuków eliptycznych służą następujące atrybuty: *styl linii*, *szerokość linii*, *barwa* i *styl pióra*.

Atrybuty są ustawiane *modalnie*, to znaczy są globalnymi zmiennymi stanu, które zachowują swoje wartości dopóty, dopóki nie zostaną bezpośrednio zmienione. W efekcie, przy rysowaniu prymitywów obowiązują atrybuty, które były ustawione w czasie specyfikowania prymitywów; dlatego zmiana wartości atrybutu w żaden sposób nie wpływa na wcześniej utworzone prymitywy – oddziaływanie jest tylko na te prymitywy, które zostały określone po zmianie wartości atrybutu. Modalne atrybuty są wygodne, ponieważ programiści nie muszą specyfikować długiej listy parametrów dla każdego prymitywu (w systemach produkcyjnych mogą być tysiące różnych atrybutów).

[2] Tutaj opis atrybutów SRGP nie zawsze jest szczegółowy, zwłaszcza jeśli chodzi o związki między poszczególnymi atrybutami. Niektóre szczegóły są pominięte, ponieważ dokładny wpływ atrybutu zależy od jego implementacji, a ze względu na parametry systemów w różnych systemach są używane różne implementacje; odpowiednie informacje można znaleźć w materiałach referencyjnych określonej implementacji.

Styl i szerokość linii są ustalane przez wywołania:

```
void SRGP_setLineStyle ( lineStyle CONTINUOUS / DASHED / DOTTED / ... );3)
void SRGP_setLineWidth ( int widthValue );
```



Rys. 2.7. Linie o różnych szerokościach i stylach

Szerokość linii jest mierzona w jednostkach ekranowych, to jest w pikselach. Każdy atrybut ma wartość domniemaną: styl linii CONTINOUS (linia ciągła) i szerokość 1. Na rysunku 2.7 pokazano linie o różnych szerokościach i stylach; kod, który posłużył do wygenerowania tych linii, pokazano w programie 2.1.

Na styl linii możemy patrzeć w następujący sposób: jest to maska bitowa używana do selektywnego zapisywania pikseli w czasie rasteryzacji prymitywu przez SRGP. Zero w masce oznacza, że dany piksel nie powinien zostać zapisany i w pamięci obrazu zostaje zachowana poprzednia wartość. Można uważać, że ten piksel jest przezroczysty, ponieważ widać przez niego oryginalny piksel znajdujący się pod spodem. Dalej styl CONTINOUS odpowiada ciągowi samych jedynek, a DASHED (linia przerywana) odpowiada ciągowi 1111001111001111[....], przy czym kreska jest dwa razy dłuższa od przezroczystych segmentów między kreskami.

Każdy atrybut ma wartość domniemaną; na przykład wartością domniemaną dla stylu linii jest CONTINOUS, dla szerokości linii 1 itd. W początkowych przykładowych kodach nie ustawialiśmy stylu linii dla pierwszego rysowanego odcinka; dlatego korzystaliśmy z domniemanego stylu linii. W praktyce przyjmowanie założenia o bieżącym stanie atrybutów nie jest bezpieczne i w następnych przykładach w każdej funkcji atrybuty ustawiamy bezpośrednio, tak żeby funkcje były modularne, co ułatwia uruchamianie i pielęgnowanie. W punkcie 2.1.4 zobaczymy, że dla programisty jest bezpiecznie zapamiętywać i odtwarzać atrybuty bezpośrednio dla każdej funkcji.

Program 2.1 Kod użyty do generowania rysunku 2.7

```
SRGP_setLineWidth (5);
SRGP_lineCoord (55, 5, 55, 295);            /* Odcinek a */
SRGP_setLineStyle(DASHED);
SRGP_setLineWidth(10);
SRGP_lineCoord(105, 5, 155, 295);           /* Odcinek b */
SRGP_setLineWidth(15);
SRGP_setLineStyle(DOTTED);
SRGP_lineCoord(155, 5, 285, 255);           /* Odcinek c */
```

[3] Tutaj i w dalszym tekście używamy skróconej notacji. W SRGP te stałe symboliczne faktycznie są wartościami przeliczalnego typu danych **lineStyle**.

Dla znacznika można ustawiać następujące parametry:

```
void SRGP_setMarkerSize ( int sizeValue );
void SRGP_setMarkerStyle ( markerStyle MARKER_CIRCLE
    / MARKER_SQUARE / ... );
```

Wielkość znacznika określa w pikselach długość boku kwadratu, w którym zawiera się znacznik. Pełny zestaw styli znacznika można znaleźć w podręczniku referencyjnym; domniemanym stylem jest kółeczko jak na rys. 2.4.

Barwa. Każdy z dotychczas przedstawionych atrybutów wpływa tylko na niektóre prymitywy SRGP, natomiast atrybut barwy (wartości całkowitoliczbowe) wpływa na wszystkie prymitywy. Oczywiście znaczenie atrybutu barwy zależy w dużym stopniu od dostępnego sprzętu; 0 i 1 są to dwie barwy występujące w każdym systemie. W systemach dwupoziomowych, barwa jest łatwa do przewidzenia: w urządzeniach czarno-białych piksele o barwie 1 są czarne, a piksele o barwie 0 są białe, w urządzeniach zielono-czarnych 1 oznacza barwę zieloną, a 0 barwę czarną.

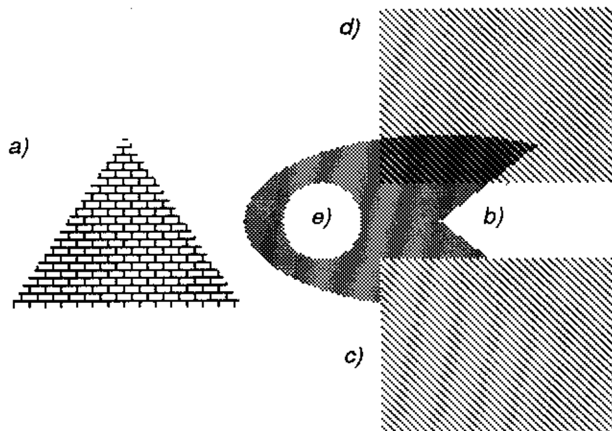
Całkowitoliczbowy atrybut barwy nie określa barwy bezpośrednio; jest to indeks do *tabeli barw* SRGP; każda pozycja tej tabeli definiuje barwę albo poziom szarości w sposób, którego programista korzystający z SRGP nie musi znać. W tabeli barw jest 2^d pozycji, przy czym d jest głębokością (liczbą bitów pamiętanych dla każdego piksela) pamięci obrazu. W implementacjach dwupoziomowych tabela barw jest realizowana sprzetowo; jednak w większości implementacji SRGP dopuszcza modyfikowanie tabeli. Szczegóły można znaleźć w podręcznikach referencyjnych. Niektóre z możliwości wykorzystania tabeli barw omówiono w rozdz. 4.

Program użytkowy może wykorzystywać każdą z dwóch metod określania barwy. W zastosowaniu, dla którego ważna jest niezależność od rodzaju komputera, powinno się używać bezpośrednio liczb 0 i 1; wtedy program użytkowy będzie mógł być wykorzystywany dla wszystkich monitorów dwupoziomowych i barwnych. Jeżeli program użytkowy zakłada wykorzystywanie barwy albo jest napisany dla określonego urządzenia wyświetlającego, to można wykorzystywać *nazwy barw* (zależne od implementacji) dostępne w SRGP. Te nazwy są to symboliczne stałe całkowitoliczbowe, które pokazują, gdzie zostały umieszczone pewne standardowe barwy w domniemanej tabeli barw dla danego urządzenia wyświetlającego. Na przykład, dla implementacji czarno-białej są dwie nazwy barw COLOR_BLACK (1) i COLOR_WHITE (0); korzystamy z tych wartości w przykładowych fragmentach kodu. Zauważmy, że nazwy barw nie są użyteczne dla zastosowań, które modyfikują tabelę barw. Barwę wybieramy za pomocą wywołania

```
void SRGP_setColor ( int colorIndex );
```

2.1.3. Wypełniane prymitywy i ich atrybuty

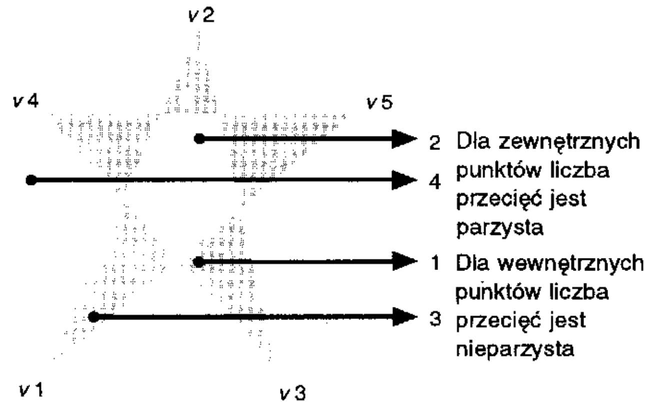
Prymitywy, które definiują obszar, można rysować albo jako *kontur*, albo jako *wypełnione wnętrze*. Funkcje opisane w poprzednim punkcie generowały zamknięte kontury z niewypełnionym wnętrzem. W SRGP wersje z wypełnianiem rysują piksele należące do wnętrza bez otaczającego konturu. Na rysunku 2.8 pokazano zestaw wypełnianych prymitywów dostępnych w SRGP, w tym wypełniony *łuk eliptyczny* (rys. 2.8b).



Rys. 2.8. Wzory map bitowych wypełnionych prymitywów: a), b), c) tryb nieprzezroczysty; d) tryb przezroczysty; (e) wypełnienie ciągłe

Zauważmy, że SRGP nie rysuje wokół wnętrza kontrastującego konturu, na przykład w postaci ciągłej linii o grubości 1 piksela; zastosowania, w których taki kontur jest potrzebny, muszą narysować go bezpośrednio. Ten problem omówimy dokładniej w p. 3.4.

W celu narysowania wypełnionego wielokąta używamy funkcji *SRGP_fillPolygon* albo *SRGP_fillPolygonCoord* z tą samą listą parametrów, jakiej używaliśmy przy wersjach bez wypełniania. Inne prymitywy wypełniające powierzchnię definiujemy w ten sam sposób, dodając prefiks „fill" do ich nazw. Ponieważ wielokąty mogą być wklęsłe a nawet samoprzecinające się, potrzebna jest reguła, która umożliwi określenie, które obszary stanowią wnętrze i powinny być wypełnione, a które są zewnętrzne. W SRGP obowiązuje reguła parzystości. W celu określenia, czy obszar leży wewnątrz, czy na zewnątrz danego wielokąta, należy wybrać jako punkt testowy dowolny punkt leżący wewnątrz określonego obszaru. Następnie z tego punktu należy poprowadzić półprostą w dowolnym kierunku nie przechodzącą przez żaden wierzchołek. Jeżeli ta półprosta przetnie brzeg wielokąta nieparzystą liczbę razy, to obszar uważamy za wewnętrzny (rys. 2.9).



Rys. 2.9. Reguła parzystości określania wnętrza wielokąta

Program 2.2. Kod użyty do wygenerowania rys. 2.8

```
SRGP_setFillStyle(BITMAP_PATTERN_OPAQUE);
SRGP_setFillBitmapPattern(BRICK_BIT_PATTERN);                /*Wzór typu cegła
SRGP_fillPolygon(3, triangle_coords);                        /* a */

SRGP_setFillBitmapPattern(MEDIUM_GRAY_BIT_PATTERN) ;         /* Poziom szarości
                                                                50 procent */
SRGP_fillEllipseArc(ellipseArcRect, 60.0, 290);              /* b */

SRGP_setFillBitmapPattern(DIAGONAL_BIT_PATTERN);
SRGP_fillRectangle(opaqueFilledRect);                        /* c */

SRGP_setFillStyle(BITMAP_PATTERN_TRANSPARENT);
SRGP_fillRectangle(transparentFilledRect);                   /* d */

SRGP_setFillStyle(SOLID);
SRGP_setColor(COLOR_WHITE);
SRGP_fillEllipse(circleRect);                                /* e */
```

W rzeczywistości w czasie rysowania SRGP nie wykonuje tego testu dla każdego piksela; pakiet wykorzystuje zoptymalizowaną technikę konwersji wielokąta opisaną w rozdz. 3, w której reguła parzystości jest wykorzystana do wszystkich sąsiednich pikseli leżących w danym wierszu albo wewnątrz, albo na zewnątrz. Metoda testu parzystości jest wykorzystywana również przy określaniu obiektu wskazywanego przez użytkownika za pomocą kursora (por. rozdz. 7).

Style i wzory wypełniania obszarów. Atrybut stylu wypełniania może być używany do określania wyglądu wnętrza wypełnianych prymitywów za pomocą jednego z czterech sposobów

```
void SRGP_setFillStyle ( drawStyle SOLID / BITMAP_PATTERN_OPAQUE /
     BITMAP_PATTERN_TRANSPARENT / PIXMAP_PATTERN);
```

W opcji SOLID prymitywy są wypełniane równomiernie barwą określoną przez bieżącą wartość atrybutu (rys. 2.8e z barwą ustawioną na COLOR_WHITE). Następne dwie opcje BITMAP_PATTERN_OPAQUE i BITMAP_PATTERN_TRANSPARENT wypełniają prymitywy regularnym, nieciągłym wzorem; w pierwszej z tych opcji wszystkie piksele wypełnianego obszaru są ponownie zapisywane albo z zachowaniem barwy, albo z inną barwą (rys. 2.8c); w drugiej z tych opcji niektóre piksele prymitywu są zapisywane z zachowaniem barwy, a pozostałe piksele stają się przezroczyste (rys. 2.8d). Ostatnia opcja PIXMAP_PATTERN zapisuje wzory z dowolną liczbą barw, zawsze w trybie nieprzezroczystości.

Mapy bitowe wzorów są to tablice zer i jedynek wybranych z tablicy dostępnych wzorów za pomocą funkcji

```
void SRGP_setFillBitmapPattern ( int patternIndex );
```

Każda pozycja w tablicy wzoru pamięta unikatowy wzór; w zestawie wzorów SRGP, zamieszczonym w podręczniku referencyjnym, są poziomy szarości (od prawie czarnego do prawie białego) oraz różne regularne i nieregularne wzory. W trybie przezroczystości te wzory są generowane w następujący sposób. Potraktujmy dowolny wzór tablicy wzorów jako małą mapę bitową – powiedzmy 8 na 8 – która może być powtarzana w miarę potrzeby, przy wypełnianiu prymitywu. W systemie dwupoziomowym bieżąca barwa (w efekcie barwa pierwszego planu) jest zapisywana tam, gdzie we wzorze jest 1; tam gdzie są zera – dziury – odpowiednie piksele oryginalnego obrazu nie są ponownie zapisywane i wobec tego są widoczne przez częściowo przezroczysty prymityw. Dlatego dla wzorów w trybie przezroczystym wzór w postaci mapy bitowej działa jak maska zezwolenia na zapis do pamięci, podobnie jak bit maski w stylu linii działał na prymitywy linii i konturów.

W częściej używanym trybie BITMAP_PATTERN_OPAQUE jedynki są zapisywane bieżącą barwą, natomiast zera są zapisywane inną barwą – *barwą tła* – ustawioną wcześniej za pomocą funkcji

```
void SRGP_setBackgroundColor ( int colorIndex );
```

W monitorach dwupoziomowych każdy wzór mapy bitowej w trybie OPAQUE może generować tylko dwa różne wzory wypełniania. Na przykład wzór mapy bitowej składający się głównie z jedynek może być używany w przypadku monitora czarno-białego do generowania ciemnoszarego wzoru wypełniania, jeżeli bieżącą barwą jest czarna (a tło jest białe) i jasnego wzoru wypełniania, jeżeli bieżącą barwą jest biała (a tło jest czarne). W monitorze kolorowym do uzyskania różnych efektów z dwoma odcieniami można używać różnych kombinacji barw pierwszego planu i tła. W typowym zastosowaniu dla monitora dwupoziomo-

wego zawsze ustawia się barwę tła wówczas, gdy ustawia się barwę pierwszego planu; gdyby te dwie barwy były sobie równe, to nieprzezroczyste wzory map bitowych byłyby niewidoczne. Program użytkowy mógłby utworzyć funkcję SetColor ustawiającą automatycznie barwę tła kontrastującą z pierwszym planem zawsze wówczas, gdy barwa pierwszego planu jest wybrana bezpośrednio.

Rysunek 2.8 został utworzony przez fragment kodu pokazany w programie 2.2. Zaletą wzorów map bitowych z dwoma poziomami jest to, że barwy nie są ustawiane bezpośrednio, a są określane przez atrybuty barwy i dlatego mogą być generowane w dowolnych kombinacjach. Wadą jest to, że mogą być generowane tylko dwie barwy; z tego też powodu SRGP ma wzory map pikselowych. Często chcielibyśmy wypełnić obszar za pomocą więcej niż dwóch barw, za pomocą bezpośrednio określonego wzoru. Podobnie jak wzór mapy bitowej jest małą mapą bitową wykorzystywaną do pokrywania prymitywu, tak mała mapa pikselowa może być wykorzystywana do wypełniania prymitywu, przy czym mapa pikselowa jest tablicą wzoru zawierającą indeksy do tabeli barw. Ponieważ każdy piksel jest bezpośrednio ustawiany w mapie pikselowej, nie ma koncepcji dziur i nie ma rozróżnienia między trybami wypełniania przezroczystym i nieprzezroczystym. Aby wypełnić obszar barwnym wzorem, wybieramy styl wypełniania PIXMAP_PATTERN i używamy odpowiedniej funkcji wyboru wzoru mapy pikselowej:

```
void SRGP_setFillPixmapPattern ( int patternIndex );
```

Ponieważ zarówno w przypadku wzorów mapy bitowej, jak i mapy pikselowej są generowane piksele o wartościach barw, które są indeksami do bieżącej tabeli barw, wygląd wypełnionych prymitywów zmienia się, gdy programista modyfikuje pozycje tabeli barw. W podręczniku referencyjnym SRGP pokazano sposób zmiany albo rozszerzenia tablic wzorów mapy bitowej i mapy pikselowej. Chociaż SRGP ma domniemane pozycje w tablicy wzorów mapy bitowej, nie ma domniemanej tablicy wzoru mapy pikselowej, ponieważ jest nieskończona liczba barwnych wzorów map pikselowych, które mogłyby być użyteczne.

Tło ekranu w programie użytkowym. Zdefiniowaliśmy barwę tła jako barwę bitów 0 we wzorach mapy bitowej używanych w trybie nieprzezroczystym; na ogół jednak określenie *tło* jest używane w innym znaczeniu. Użytkownik spodziewa się, że ekran będzie wyświetlał prymitywy na pewnym jednolitym wzorze tła, który pokrywa nieprzezroczyste okno albo cały ekran. Wzorem dla tła ekranu w programie użytkowym jest często stała barwa 0, ponieważ SRGP po inicjalizacji ustawia taką barwę na ekranie. Czasami jednak wzór tła nie jest ciągły albo

jest ciągły, lecz ma inną barwę; w takich przypadkach program użytkowy jest odpowiedzialny za ustawienie tła ekranu – przed narysowaniem jakichkolwiek prymitywów musi zostać narysowany prostokąt pokrywający cały ekran odpowiednim wzorem.

Często stosowana metoda *usuwania* prymitywów polega na ponownym ich narysowaniu z użyciem wzoru tła programu użytkowego, a nie na ponownym rysowaniu całego obrazu za każdym razem, gdy jest usuwany prymityw. Ta szybka metoda *uaktualniania* daje jednak niepoprawny obraz wówczas, gdy usuwany prymityw nakłada się na inne prymitywy.

Załóżmy na przykład, że wzór tła ekranu z rys. 2.8 jest cały biały i że usuwamy prostokąt w części c) rysując go ponownie za pomocą ciągłej barwy białej COLOR_WHITE. W rezultacie pozostanie biały fragment w wypełnionym łuku elipsy (część b)). Usunięcie tego defektu wymaga sięgnięcia do bazy danych programu użytkowego i zmiany parametrów prymitywów (zob. zadanie 2.8).

2.1.4. Zapamiętywanie i odtwarzanie atrybutów

SRGP dopuszcza wiele atrybutów dla różnych prymitywów. Atrybuty można zapamiętać w celu późniejszego odtworzenia; ta cecha jest specjalnie użyteczna przy projektowaniu funkcji programu użytkowego, który wykonuje swoje zadania bez ubocznych efektów – to znaczy nie wpływa na ogólny stan atrybutu. Dla wygody SRGP umożliwia odtwarzanie całego zbioru atrybutów – określanego jako *grupa atrybutów* – za pomocą

```
void SRGP_inquireAttributes ( attributeGroup *group );
void SRGP_setAttributes ( attributeGroup group );
```

Program użytkowy ma dostęp do wszystkich pól definiowanego przez SRGP rekordu „attributeGroup"; rekord otrzymany w wyniku realizacji funkcji zapytania może być użyty do selektywnego odtworzenia.

2.1.5. Tekst

W pakietach graficznych specyfikowanie i implementowanie rysowania tekstu jest zawsze złożonym zadaniem; wiąże się to z dużą liczba opcji i atrybutów, jakie może mieć tekst. Wśród nich są: styl albo krój znaków (Times, Helvetica, Bodoni itd.), ich wygląd (pismo proste, wytłuszczenie, kursywa, podkreślenie itd.), ich wielkość (typowo mierzona

w punktach[4]) i szerokość, odstęp między znakami, odstęp między kolejnymi wierszami, kąt, pod którym znaki są rysowane (poziomy, pionowy albo wybrany kąt) itd.

W prostym sprzęcie i oprogramowaniu znaki mają na ogół stałą szerokość, odstęp między nimi jest jednolity – wszystkie znaki zajmują taką samą szerokość i odstępy między nimi są stałe. Z drugiej strony, przy odstępach proporcjonalnych zmieniają się zarówno szerokości znaków, jak i odstępy między nimi, co czyni tekst bardziej czytelnym i estetycznym. W książkach, magazynach i gazetach są używane odstępy proporcjonalne; tak też jest w większości rastrowych monitorów graficznych i drukarek laserowych. W SRGP zastosowano rozwiązanie pośrednie: tekst jest wyrównany w poziomie, szerokość znaków jest zmienna, a odstęp między znakami jest stały. Ta prosta forma odstępu proporcjonalnego umożliwia opisywanie schematów i rysunków, realizację interakcji z użytkownikiem za pomocą menu tekstowych, pól dialogowych i form polegających na wpisywaniu, a nawet implementowaniu prostych procesorów tekstów. Jednak w takich zastosowaniach jak programy wydawnicze do tworzenia publikacji dobrej jakości konieczne jest stosowanie specjalizowanych pakietów, które zapewniają lepszą kontrolę nad specyfikacją tekstu i jego atrybutami niż SRGP. PostScript [ADOB85] oferuje wiele takich zaawansowanych możliwości; jest on standardem przemysłowym do opisywania tekstu i innych prymitywów za pomocą wielu różnych opcji i atrybutów. W SRGP tekst jest generowany przez wywołanie funkcji

```
void SRGP_text ( point origin, char *text );
```

Położenie prymitywu tekstu określa punkt początkowy (*origin*), określany również jako *punkt zaczepienia*. Współrzędna x punktu początkowego odpowiada lewemu brzegowi pierwszego znaku, a współrzędna y określa, gdzie powinna się pojawić *linia bazowa* ciągu znaków. (Linia bazowa jest to hipotetyczna linia, na której opierają się znaki, tak jak to pokazano na rys. 2.10 na przykładzie tekstowego przycisku menu. Niektóre znaki na przykład „y" i „q" mają ogonek, który schodzi poniżej linii bazowej.)

Wygląd prymitywu tekstu określają tylko dwa atrybuty – bieżąca barwa i krój. Krój jest to indeks do tablicy krojów o różnych wielkościach i stylach, zależnej od implementacji:

```
void SRGP_setFont ( int valueIndex );
```

Każdy znak w kroju pisma jest definiowany jako prostokątna mapa bitowa; SRGP rysuje znak wypełniając prostokąt za pomocą mapy

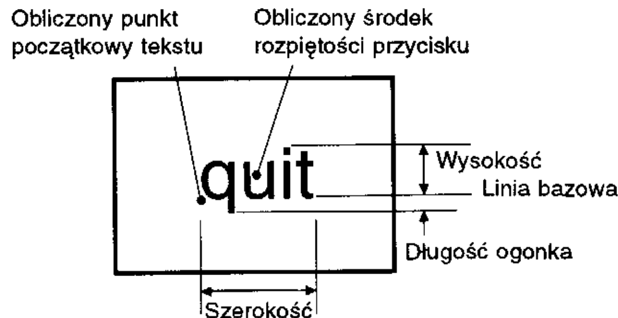
[4]) Punkt jest jednostką powszechnie używaną drukarstwie; jest równy ok. 1/72 cala.

bitowej znaku jako wzoru. Jedynki w mapie bitowej określają wnętrze znaku, a zera określają otoczenie i przerwy takie jak dziura w „o." (W niektórych bardziej zaawansowanych pakietach znaki są definiowane za pomocą map pikselowych, dzięki czemu jest możliwe pokrywanie wnętrza znaków wzorami.)

Formatowanie tekstu. Ponieważ implementacje SRGP oferują ograniczony repertuar krojów oraz wielkości i ponieważ implementacje na różnych komputerach rzadko oferują takie same repertuary, program użytkowy ma ograniczoną kontrolę nad wysokością i szerokością ciągów tekstowych. Ponieważ dla dobrego rozmieszczania tekstu jest potrzebna informacja o prostokącie opisanym na tekście (na przykład w celu umieszczenia tekstu, w środku prostokątnej ramki), SRGP ma następującą funkcję dla pytania o rozpiętość danego ciągu znaków wykorzystującą bieżącą wartość atrybutu kroju pisma:

```
void SRGP_inquireTextExtent( char*text, int*width, int*height, int*descent );
```

Chociaż SRGP nie ma trybu nieprzezroczystej mapy bitowej dla pisania znaków, można taki tryb łatwo naśladować. Na przykład, program 2.3 pokazuje, jak można wykorzystać informację o rozpiętości i atrybutach specyficznych dla tekstu do utworzenia czarnego tekstu, z wykorzystaniem bieżącego kroju pisma, do umieszczania tekstu



Rys. 2.10. Wymiary tekstu umieszczonego w prostokątnym przycisku i punkty obliczone na podstawie tych wymiarów umożliwiają umieszczenie tekstu pośrodku

w środku białego otaczającego prostokąta, jak to pokazano na rys. 2.10. Najpierw jest tworzone tło prostokąta przycisku o określonej wielkości i z oddzielnym konturem, a następnie w jego środku jest umieszczany tekst. Inny wariant tego problemu jest poruszony w zadaniu 2.9.

Program 2.3
Kod użyty do wygenerowania rys. 2.10

```
void MakeQuitButton( rectangle buttonRect )
{
   point centerOfButton, textOrigin;
   int width, height, descent;

   SRGP_setFillStyle(SOLID);
   SRGP_setColor(COLOR_WHITE);
   SRGP_fillRectangle(buttonRect);
   SRGP_setColor(COLOR_BLACK);
   SRGP_setLineWidth(2);
   SRGP_Rectangle(buttonRect);
   SRGP_inquireTextExtent("quit", &width, &height, &descent);
   centerOfButton.x = (buttonRect.bottomLeft.x + buttonRect.topRight.x)/2;
   centerOfButton.y = (buttonRect.bottomLeft.y + buttonRect.topRight.y)/2;
   textOrigin.x = centerOfButton.x – (width/2);
   textOrigin.y = centerOfButton.y – (height/2);
   SRGP_text(textOrigin, "quit" );
}
```

2.2. Podstawy obsługi interakcji

Teraz, kiedy już wiemy, jak narysować podstawowe kształty i tekst, naszym następnym krokiem jest pokazanie, jak można pisać programy interakcyjne, które komunikują się w efektywny sposób z użytkownikiem, poprzez urządzenia wejściowe, np. klawiaturę i myszkę. Najpierw poznamy ogólne wskazówki pisania efektywnych i przyjemnych w użytkowaniu programów interakcyjnych; następnie omówimy podstawowe pojęcia związane z logicznymi (abstrakcyjnymi) urządzeniami zewnętrznymi. Następnie pokażemy mechanizmy SRGP umożliwiające realizację różnych aspektów obsługi interakcji.

2.2.1. Czynniki ludzkie

Projektant programu interakcyjnego musi dać sobie radę z wieloma czynnikami, które nie występują w programach nieinterakcyjnych, wsadowych. Są one określane jako *czynniki ludzkie* programu i obejmują styl interakcji (określany często jako *patrz* i *czuj*) i łatwość uczenia się go oraz stosowania. Czynniki ludzkie są tak samo ważne jak kompletność i poprawność funkcjonalna. Metody interakcji użytkownik-komputer, które cechują dobre czynniki ludzkie, są omawiane dokładniej w rozdz. 8. Wśród dyskutowanych tam zaleceń są:

- Zapewnić *proste* i *zwarte sekwencje interakcji.*
- *Nie należy przeciążać użytkownika* zbyt dużą liczbą opcji i stylów.
- Na każdym etapie interakcji *pokazywać w przejrzysty sposób dostępne opcje.*
- Zapewnić użytkownikowi odpowiednie *sprzężenie zwrotne.*
- Umożliwiać użytkownikowi *łatwe wycofanie się* z pomyłki.

W naszych prostych programach próbujemy stosować się do tych zaleceń związanych z czynnikiem ludzkim. Na przykład z reguły używamy menu, które umożliwia użytkownikowi wskazanie za pomocą myszki przycisku tekstowego odpowiadającego funkcji, która powinna być wykonana. Powszechnie stosujemy również palety (menu składające się z ikon) z podstawowymi prymitywami geometrycznymi, symbolami specyficznymi dla zastosowania i wzorami wypełniania. Menu i palety spełniają nasze trzy pierwsze zalecenia; prezentują one użytkownikowi listę dostępnych opcji i zapewniają prosty i zwarty sposób wybierania tych opcji. Niedostępne opcje mogą być albo chwilowo usunięte, albo narysowane za pomocą wzoru z odcieniami szarości o małej intensywności, zamiast za pomocą ciągłej barwy.

Sprzężenie zwrotne pojawia się na każdym etapie działania menu i dzięki temu jest spełnione czwarte zalecenie: w celu zwrócenia uwagi program użytkowy podświetla wybraną pozycję menu albo wybrany obiekt – na przykład przez wyświetlanie inwersyjne albo wzięcie w ramkę. Sam pakiet może również generować *echo* podające natychmiastową odpowiedź na akcję urządzenia wejściowego. Na przykład w miejscu wskazywanym przez kursor natychmiast po naciśnięciu klawisza klawiatury pojawia się znak; po przesunięciu myszki po stole albo biurku kursor odpowiednio zmienia położenie na ekranie. Pakiety graficzne oferują wiele kształtów kursorów, które mogą być używane przez programy użytkowe do pokazywania stanu programu. W wielu systemach monitorowych kształt kursora może się zmieniać dynamicznie w funkcji pozycji kursora na ekranie. Na przykład, w wielu programach przetwarzania tekstów kursor jest pokazywany jako strzałka w obszarze menu i jako migająca pionowa kreska w obszarach tekstowych.

Nasze piąte zalecenie – łagodny powrót po zrobieniu błędu – jest realizowane za pomocą funkcji kasowania albo anulowania. Wymaga to pamiętania przez program użytkowy rekordu operacji czynności korekcyjnych.

2.2.2. Logiczne urządzenia wejściowe

Typy urządzeń w SRGP. Ważnym celem przy projektowaniu pakietów graficznych jest zapewnienie niezależności działania od rodzaju urzą-

dzenia – sprzyja to przenoszalności programu użytkowego. W SRGP cel ten uzyskuje się w odniesieniu do wyjściowych urządzeń graficznych dzięki specyfikowaniu prymitywów w abstrakcyjnym systemie współrzędnych całkowitych, co uwalnia program użytkowy od potrzeby ustawiania poszczególnych pikseli w pamięci obrazu. W celu uzyskania poziomu abstrakcji dla wejścia graficznego SRGP operuje zbiorem *logicznych urządzeń wejściowych*, które uwalniają program użytkowy od szczegółów związanych z dostępnym fizycznym urządzeniem wejściowym. SRGP dopuszcza dwa urządzenia logiczne:

- *Lokalizator* określający współrzędne ekranu i stan jednego lub kilku przycisków.
- *Klawiaturę* umożliwiającą wprowadzanie ciągów znaków.

SRGP odwzorowuje urządzenia logiczne na dostępne urządzenie fizyczne (na przykład lokalizator mógłby być odwzorowany na myszkę, drążek sterowniczy, tabliczkę albo ekran dotykowy). To odwzorowanie urządzenia logicznego na fizyczne jest podobne do stosowanego w konwencjonalnych językach proceduralnych i systemach operacyjnych, w których urządzenia we/wy, np. terminale, dyski i napędy taśmy, są rozważane w kategoriach logicznych plików danych, żeby uzyskać niezależność od urządzeń oraz prostotę programowania użytkowego.

Sposób traktowania urządzeń w innych pakietach. Model urządzeń wejściowych przyjęty w SRGP jest w zasadzie podzbiorem modeli w standardach GKS i PHIGS. W implementacjach SRGP dopuszcza się tylko jeden lokalizator i jedną klawiaturę, podczas gdy GKS i PHIGS dopuszczają wiele urządzeń każdego typu. Te pakiety dopuszczają również inne typy urządzeń: *kreskowe* (przekazują łamaną określoną przez ciąg pozycji kursora wprowadzonych za pomocą lokalizatora), *wybierające* (reprezentują zestaw kluczy funkcyjnych i przekazują identyfikator klucza), *waluatory* – urządzenia do wprowadzania wartości (reprezentują suwak potencjometru albo tarczę sterującą i przesyłają liczbę zmiennopozycyjną) oraz *wskazujące* (reprezentują urządzenie wskazujące, np. myszkę albo tabliczkę, z odpowiednim przyciskiem do potwierdzania wyboru, i przesyłają identyfikator wskazanej wielkości logicznej). Inne pakiety, np. QuickDraw i X Window System, traktują urządzenia wejściowe w sposób bardziej zależny od typu urządzenia; daje to programiście większą kontrolę nad poszczególnymi funkcjami urządzenia, kosztem większej złożoności programu użytkowego i zmniejszonymi możliwościami przenoszenia na inne typy komputerów.

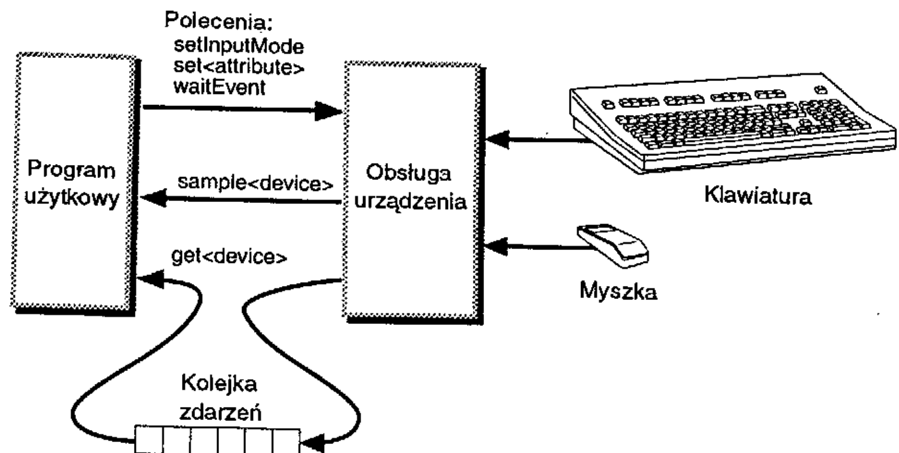
Inne właściwości urządzeń logicznych omówiono w rozdz. 8. Tutaj w skrócie omówimy ogólne tryby interakcji z urządzeniami logicznymi, a następnie dokładniej poznamy możliwości interakcyjne pakietu SRGP.

2.2.3. Próbkowanie a przetwarzanie sterowane zdarzeniami

Do odbierania informacji tworzonych w wyniku interakcji z użytkownikiem są wykorzystywane dwie podstawowe metody. W metodzie *próbkowania* (określanej również jako *odpytywanie urządzeń*) program użytkowy pyta o bieżącą wartość wejściowego urządzenia logicznego (określaną jako *stan urządzenia*) i kontynuuje działanie. Próbkowanie ma miejsce niezależnie od tego, czy stan urządzenia zmienił się od ostatniego próbkowania; tylko w wyniku ciągłego próbkowania stanu urządzenia program użytkowy może dowiedzieć się o zmianie stanu urządzenia. Ten tryb jest kosztowny dla zastosowań interakcyjnych, ponieważ duża ilość czasu pracy CPU jest tracona na wykonywanie pętli próbkowania w oczekiwaniu na zmianę stanu.

Alternatywą dla pętli sprawdzania jest interakcja na zasadzie *sterowania przerwaniami*; w tej metodzie program użytkowy zezwala na pracę jednego lub wielu urządzeń wejściowych, a potem kontynuuje normalną pracę dopóty, dopóki nie pojawi się przerwanie wywołane jakimś *zdarzeniem wejściowym* (zmiana stanu urządzenia wywołana akcją użytkownika); wtedy sterowanie przechodzi asynchronicznie do procedury przerwania, która odpowiada na to zdarzenie. Dla każdego urządzenia wejściowego jest przewidziany wskaźnik sygnalizujący, czy miała miejsce akcja użytkownika. Wskaźnik jest włączany przez naciśnięcie przycisku, na przykład przycisku myszki albo klawisza klawiatury.

W celu uwolnienia programisty piszącego program użytkowy od trudnych a czasami trikowych aspektów asynchronicznego przekazywania sterowania, wiele pakietów graficznych, w tym GKS, PHIGS



Rys. 2.11. Próbkowanie a obsługa zdarzeń przy wykorzystaniu kolejki zdarzeń

i SRGP, oferuje interakcję *sterowaną zdarzeniami*, będącą synchroniczną symulacją interakcji sterowanej przerwaniami. W tej metodzie program użytkowy daje zezwolenie urządzeniom i kontynuuje swoje działanie. W tle pakiet monitoruje urządzenia i zapamiętuje informacje o każdym zdarzeniu w kolejce zdarzeń (rys. 2.11). Ostatecznie program użytkowy określa, kiedy może odebrać przerwanie.

Program użytkowy sprawdzając kolejkę zdarzeń określa, czy chciałby przejść w stan oczekiwania. Jeżeli w kolejce jest informacja o jednym albo o większej liczbie zdarzeń, to jest usuwane zdarzenie znajdujące się na początku kolejki (a więc zdarzenie, które nadeszło najwcześniej) i związana z nim informacja jest przekazywana do programu użytkowego. Jeżeli kolejka jest pusta i przejście w stan oczekiwania nie jest pożądane, to program użytkowy otrzymuje informację, że nie ma żadnego zdarzenia i że może kontynuować pracę. Jeżeli kolejka jest pusta i stan oczekiwania jest pożądany, to program użytkowy pauzuje dopóty, dopóki nie pojawi się następne zdarzenie albo dopóki nie skończy się przewidziany przez program użytkowy maksymalny czas oczekiwania. W efekcie tryb zdarzeń zastępuje odpytywanie urządzeń wejściowych znacznie bardziej efektywnym oczekiwaniem na kolejkę zdarzeń.

Podsumowując, w trybie próbkowania urządzenie jest sprawdzane co pewien czas i zapamiętuje się stany zdarzeń, niezależnie od akcji użytkownika. W trybie zdarzeń program użytkowy albo otrzymuje raport o zdarzeniu związanym z wcześniejszą akcją użytkownika, albo oczekuje na akcję użytkownika (albo na zakończenie dopuszczalnego czasu oczekiwania). Zasadnicza różnica między trybem próbkowania a trybem sterowania zdarzeniami polega więc na tym, że w trybie sterowania zdarzeniami odpowiedź ma miejsce tylko wówczas, gdy użytkownik wykona jakąś akcję. Programowanie w trybie sterowania zdarzeniami wydaje się bardziej złożone niż przy próbkowaniu, ale znamy już podobną metodę, która była użyta w funkcji scanf w interakcyjnym programie C: C daje zezwolenie klawiaturze, a program użytkowy czeka w scanf dopóty, dopóki użytkownik nie zakończy wprowadzania wiersza tekstu. W niektórych środowiskach polecenie scanf może być wykorzystane do uzyskania dostępu do znaków, które były napisane i zapamiętane w kolejce, zanim polecenie scanf zostało wydane.

Proste programy sterowane zdarzeniami w SRGP i w innych pakietach działają według metody interakcji typu *ping-pong* wprowadzonej w p. 1.4.3 i w pseudokodzie w programie 2.4; taką interakcję można elegancko modelować w postaci automatu skończonego. Bardziej złożone style interakcji umożliwiające równoczesną aktywność programu i użytkownika omówiono w rozdz. 8.

Program 2.4 Schemat interakcji sterowanej zdarzeniami

```
inicjalizacja, włącznie z generowaniem początkowego obrazu;
ustawienie urządzenia (urządzeń) w tryb sterowania zdarzeniami;
do {        /* główna pętla zdarzenia */
  oczekiwanie na akcję użytkownika na każdym z kilku urządzeń;
  switch ( urządzenie, z którym było związane zdarzenie )
    case DEVICE_1: zbieranie dla DEVICE_1 danych o stanie zdarzenia,
                   przetwarzanie, odpowiedź;
    case DEVICE_2: zbieranie dla DEVICE_2 danych o stanie zdarzenia,
                   przetwarzanie, odpowiedź;
    ...
  }
}
while ( użytkownik nie chce wyjść );
```

Programy użytkowe sterowane zdarzeniami na ogół przez większość czasu są w stanie oczekiwania, ponieważ interakcja jest zdominowana przez czas myślenia, kiedy użytkownik decyduje, co robić dalej; nawet w szybkich grach zręcznościowych liczba zdarzeń, które użytkownik może wygenerować w ciągu sekundy, jest ułamkiem tego, co program użytkowy może obsłużyć. Ponieważ z reguły SRGP implementuje tryb zdarzeń za pomocą prawdziwych (sprzętowych) przerwań, stan oczekiwania efektywnie nie zużywa czasu CPU. W systemie wielozadaniowym zaleta jest oczywista: program użytkowy pracujący w trybie zdarzeń wykorzystuje czas CPU tylko w krótkich okresach aktywności występujących natychmiast po akcji użytkownika, zwalniając poza tym CPU do innych zadań.

Trzeba zwrócić uwagę na jeszcze inny problem związany z poprawnym wykorzystywaniem trybu zdarzeń. Chociaż mechanizm kolejkowy pozwala programowi i użytkownikowi działać asynchronicznie, użytkownikowi nie powinno się pozwolić odejść zbyt daleko od programu, ponieważ każde zdarzenie powinno w efekcie dawać echo jak również pewne sprzężenie zwrotne od programu użytkowego. Prawdą jest, że doświadczeni użytkownicy umieją „wyprzedzać", czyli wpisywać parametry takie jak nazwy plików albo nawet polecenia systemu operacyjnego wówczas, gdy system jeszcze przetwarza wcześniejsze żądania, zwłaszcza gdy przynajmniej echo typu znak po znaku pojawia się natychmiast. W przypadku poleceń graficznych „wyprzedzanie" za pomocą myszki w zasadzie nie jest użyteczne (i jest znacznie bardziej denerwujące), ponieważ użytkownik zazwyczaj chce widzieć uaktualniony ekran, zanim wykona następną czynność związaną z grafiką.

2.2.4. Tryb próbkowania

Aktywacja, deaktywacja i ustawianie trybu urządzenia. Do aktywacji lub deaktywacji urządzenia służy następująca funkcja, w której typ urządzenia i tryb są parametrami:

```
void SRGP_setInputMode ( inputDevice LOCATOR / KEYBOARD,
      inputMode INACTIVE / SAMPLE / EVENT);
```

Aby ustawić lokalizator w trybie próbkowania, wywołujemy

```
SRGP_setInputMode (LOCATOR, SAMPLE);
```

Początkowo oba urządzenie są nieaktywne. Ustawienie trybu urządzenia nie ma żadnego wpływu na inne urządzenie wejściowe – oba mogą być równocześnie aktywne i nawet wtedy nie muszą być w tym samym trybie.

Stan lokalizatora. Lokalizator jest logiczną abstrakcją myszki albo tabliczki, który przekazuje pozycję kursora jako parę (x, y) współrzędnych ekranu, numer przycisku, który ostatnio zmieniał stan i stan przycisków będących elementami tablicy **chord** (na raz może być naciśniętych kilka przycisków). Drugie pole informuje program użytkowy o tym, który przycisk wywołał zdarzenie.

```
typedef struct {
   point  position;
   enum {
      UP, DOWN
   } buttonChord[MAX_BUTTON_COUNT];          /* Typowo 1–3 */
   int buttonOfMostRecentTransition;
} locatorMeasure;
```

Po uaktywnieniu lokalizatora w trybie próbkowania za pomocą funkcji SRGP_setInputMode, możemy zapytać o jego bieżący stan korzystając z

```
void SRGP_sampleLocator ( locatorMeasure *measure );
```

Zanalizujmy przykładowy program próbkowania (program 2.5); jest to prosta pętla malowania wykorzystująca tylko przycisk 1 w lokalizatorze. Program pozostawia ślad farby w miejscach, nad którymi został przeciągnięty lokalizator przy naciśniętym przycisku 1; w trakcie przesuwania lokalizatora przez użytkownika w pętli jest próbkowany stan lokalizatora. Najpierw musimy stwierdzić, kiedy użytkownik zaczyna malować; w tym celu jest sprawdzany stan przycisku do chwili, gdy zostanie naciśnięty; następnie za każdym razem gdy ma miejsce

próbkowanie, umieszczamy farbę (w naszym prostym przykładzie wypełniony prostokąt) dopóty, dopóki użytkownik nie zwolni przycisku.

Program 2.5
Pętla próbkowania przy malowaniu

```
ustawienie atrybutów barwa/wzór i wielkości pędzla w halfBrushHeight
                                                    i halfBrushWidth
SRGP_setInputMode(LOCATOR, SAMPLE),

/* Próbkowanie dopóki przycisk nie zostanie naciśnięty. */
do {
   SRGP_sampleLocator(locMeasure);
} while (locMeasure.buttonChord[0] == UP);

/* Wykonanie pętli malowania:
      Ciągłe umieszczanie śladu pędzla i próbkowanie, dopóki przycisk
      nie zostanie zwolniony */
do {
   rect = SRGP_defRectangle( locMeasure.position.x – halfBrushWidth,
                  locMeasure.position.y – halfBrushHeight,
                  locMeasure.position.x + halfBrushWidth,
                  locMeasure.position.y + halfBrushHeight );
   SRGP_fillRectangle (rect );
   SRGP_sampleLocator( &locMeasure );
} while ( locMeasure.buttonChord[0] == DOWN );
```

Wyniki działania tej sekwencji nie są najlepsze: barwne prostokąty są w różnych odległościach od siebie, a ich gęstość zależy od wielkości przesunięcia lokalizatora między kolejnymi próbkowaniami. Szybkość próbkowania zależy w zasadzie od szybkości, z jaką CPU wykonuje rozkazy systemu operacyjnego, pakietu i programu użytkowego.

Tryb próbkowania jest dostępny dla obu urządzeń logicznych; ponieważ jednak klawiatura prawie zawsze jest obsługiwana w trybie zdarzeń, nie omawiamy tu metody próbkowania w odniesieniu do klawiatury.

2.2.5. Tryb zdarzeń

Wykorzystanie trybu próbkowania do inicjalizacji pętli próbkowania. Chociaż dwie pętle próbkowania w przykładzie malowania (jedna dla wykrycia momentu naciśnięcia przycisku, druga dla malowania, dopóki przycisk nie zostanie zwolniony) z pewnością spełniają swoje zadanie, stanowią niepotrzebne obciążenie dla CPU. Jeżeli jednak problem przeciążenia nie musi być krytyczny dla komputera osobistego, to powinno się go unikać w systemach wielozadaniowych działających na przykład

na zasadzie podziału czasu. O ile oczywiście stałe próbkowanie lokalizatora w samej pętli malowania jest konieczne (ponieważ musimy znać pozycję urządzenia przez cały czas kiedy przycisk jest naciśnięty), o tyle nie musimy korzystać z pętli próbkowania w czasie oczekiwania na zdarzenie polegające na naciśnięciu przycisku, które inicjuje proces interakcyjnego malowania. Omawiany dalej tryb zdarzeń może być użyty wówczas, gdy nie jest potrzebna znajomość stanu w czasie oczekiwania na zdarzenie.

SRGP_waitEvent. Po uaktywnieniu urządzenia w trybie zdarzeń przez polecenie SRGP_setInputMode, program może w dowolnej chwili sprawdzać kolejkę zdarzeń wprowadzając stan oczekiwania za pomocą

inputDevice SRGP_waitEvent (**int** maxWaitTime);

Jeżeli kolejka nie jest pusta, to funkcja natychmiast przesyła o tym fakcie informacje; w przeciwnym przypadku parametr określa maksymalny czas (mierzony z dokładnością do 1/60 sekundy) oczekiwania na zdarzenie pojawienia się czegoś w kolejce. Ujemna wartość *maxWaitTime* (specyfikowana przez symboliczną stałą INDEFINITE) oznacza nieskończenie długie oczekiwanie, a wartość zero powoduje natychmiastowy powrót, niezależnie od stanu kolejki.

Funkcja przesyła identyfikator urządzenia, które wywołało zdarzenie znajdujące się na początku kolejki, jako LOCATOR KEYBOARD albo NO_DEVICE. Specjalna wartość NO_DEVICE jest przesyłana wówczas, gdy w określonym czasie nie ma żadnego zdarzenia – to znaczy, jeżeli skończył się czas oczekiwania określony dla danego urządzenia. Można wtedy sprawdzić typ urządzenia po to, żeby określić, jak można uzyskać stan urządzenia znajdującego się na początku kolejki (opisujemy to w dalszym ciągu tego punktu).

Klawiatura. Dla klawiatury zdarzenie wyzwalające zależy od trybu przetwarzania, w jakim klawiatura została ustawiona. Tryb EDIT jest używany wówczas, gdy program użytkowy otrzymuje ciągi znaków (na przykład nazwy plików, polecenia) od użytkownika, który pisze ciąg znaków i dokonuje ich edycji, a potem naciska klawisz Return po to, żeby zainicjować zdarzenie. W trybie RAW używanym przy pracy interakcyjnej, kiedy klawiatura musi być dokładnie monitorowana, każde naciśnięcie klawisza powoduje powstanie zdarzenia. Do ustawiania trybu przetwarzania program użytkowy korzysta z następującej funkcji:

void SRGP_setKeyboardProcessingMode (**keyboardMode** EDIT / RAW);

W trybie EDIT użytkownik może pisać ciągi znaków, dokonywać w razie potrzeby korekcji za pomocą klawisza Backspace, a potem uży-

wać klawisza Return (albo Enter) do zainicjowania zdarzenia. Ten tryb jest używany wówczas, gdy użytkownik musi napisać ciąg, np. nazwę pliku albo etykietę rysunku. Wszystkie klawisze sterujące, oprócz Backspace i Return, są pomijane i stan jest określony przez ciąg znaków istniejący w czasie wyzwolenia. W trybie RAW każdy napisany znak, włącznie ze znakami sterującymi, inicjuje zdarzenie i jest przesyłany indywidualnie jako bieżący stan. Taki tryb jest używany wówczas, gdy znaki odgrywają rolę poleceń – na przykład przesunięcia kursora, wykonania zwykłych operacji edycyjnych w grach video. Tryb RAW nie daje echa, podczas gdy w trybie EDIT echo pojawia się na ekranie i jest wyświetlany *kursor tekstu* (na przykład znak podkreślenia albo znak prostokąta) w miejscu, gdzie pojawi się następny pisany znak. Każde naciśnięcie klawisza Backspace powoduje cofnięcie kursora tekstu z powrotem i wymazanie jednego znaku.

Gdy funkcja SRGP_waitEvent przesyła kod urządzenia KEYBOARD, wówczas program użytkowy uzyskuje stan związany ze zdarzeniem po wywołaniu

```
void SRGP_getKeyboard (char *measure , int buffersize);
```

Gdy klawiatura jest aktywna w trybie RAW, wówczas jej stan ma zawsze długość jednego znaku. W tym przypadku pierwszy znak ciągu znaków stanu określa stan RAW.

Program 2.6 ilustruje sposób użycia trybu EDIT. Program otrzymuje listę nazw plików od użytkownika i usuwa każdy tak wprowadzony plik. Gdy użytkownik wprowadza zerowy ciąg (naciskając klawisz Return bez napisania jakiegoś innego znaku), wówczas interakcja kończy się. W czasie interakcji program czeka, aż użytkownik wprowadzi następny ciąg znaków.

Chociaż ten kod bezpośrednio określa, gdzie ma się pojawić tekst zachęty, nie podaje, gdzie ma być wpisany przez użytkownika wejściowy ciąg znaków (i korygowany za pomocą klawisza Backspace). Miejsce dla tego echa klawiatury jest określane przez programistę; omawiamy to w p. 2.2.7.

Lokalizator. Zdarzeniem wyzwalającym dla lokalizatora jest naciśnięcie i zwolnienie myszki. Gdy SRGP_waitEvent przesyła kod urządzenia LOCATOR, wówczas program użytkowy uzyskuje stan związany ze zdarzeniem przez wywołanie

```
void SRGP_getLocator ( locatorMeasure *measure );
```

Na ogół pole **position** w słowie stanu jest wykorzystywane do określenia, w którym miejscu ekranu użytkownik umieścił punkt. Na przykład, jeżeli kursor lokalizatora jest w obszarze prostokątnym, w którym jest wyświetlony przycisk menu, to zdarzenie powinno być interpreto-

wane jako żądanie jakiejś akcji; jeżeli kursor jest w głównym polu rysowania, to punkt może się znaleźć albo wewnątrz poprzednio narysowanego obiektu, żeby wskazać, że powinien on być wybrany, albo w pustym obszarze, żeby wskazać, gdzie powinien być umieszczony nowy obiekt.

Program 2.6 Współpraca z kalwiaturą w trybie EDIT

```
#define KEYMEASURE_SIZE 80
SRGP_setInputMode(KEYBOARD, EVENT);      /* Zakładamy, że tylko klawiatura
                                            jest aktywna */
SRGP_setKeyboardProcessingMode(EDIT);
pt = SRGP_defPoint( 100, 100 );
SRGP_text( pt, "Specify one or more files to be deleted; to exit press Return\n" );

/* główna pętla zdarzenia */
do {
   inputDev = SRGP_waitEvent( INDEFINITE );
   SRGP_getKeyboard( measure , KEYMEASURE_SIZE );
   if (strcoll(measure, "" ))
      DeleteFile(measure);             /* DeleteFile oznacza potwierdzenie itd. */
}
while ( strcoll(measure, "" ) );
```

Pseudokod z programu 2.7 (podobny do pokazanego wcześniej dla klawiatury) pokazuje inny sposób wykorzystania lokalizatora, który umożliwia użytkownikowi określanie punktów umieszczania znaczników. Użytkownik kończy pętlę umieszczania znaczników naciskając przycisk lokalizatora wówczas, gdy kursor wskazuje przycisk ekranowy, w postaci prostokąta zawierającego tekst *quit*.

W tym przykładzie ważne jest tylko naciśnięcie przez użytkownika przycisku 1 lokalizatora; zwolnienie przycisku jest pomijane. Zauważmy, że przycisk musi zostać zwolniony, zanim pojawi się zdarzenie związane z naciśnięciem następnego przycisku – zdarzenie jest określone przez fakt zmiany stanu przycisku a nie przez fakt, że przycisk jest w określonym stanie. W celu zapewnienia tego, żeby zdarzenia przychodzące od innych przycisków nie zakłóciły tej interakcji, program użytkowy informuje SRGP, które przyciski mogą zainicjować zdarzenie związane z lokalizatorem; służy do tego wywołanie

```
void SRGP_setLocatorButtonMask ( int activeButtons );
```

Z maską przycisku są związane następujące wartości: LEFT_BUTTON_MASK, MIDDLE_BUTTON_MASK i RIGHT_BUTTON_MASK. Złożoną maskę uzyskuje się na zasadzie logicznego sumowania warto-

ści. Domniemaną wartością maski przycisku lokalizatora jest 1, ale niezależnie od tego, jaka jest maska, wszystkie przyciski zawsze mają swój stan. W implementacjach, które mają mniej niż trzy przyciski, odwołania do nie istniejących przycisków są po prostu pomijane przez SRGP i stan tych przycisków zawsze jest UP.

Funkcja PickedQuitButton porównuje wskazaną pozycję z granicami prostokąta przycisku Quit i przesyła wartość dwójkową sygnalizującą, czy użytkownik wybrał przycisk Quit. Ten proces jest prostym przykładem korelacji wskazywania omawianej w p. 2.2.6.

Program 2.7 Interakcja z lokalizatorem

```
#define QUIT 0
tworzenie przycisku Quit na ekranie;
SRGP_setLocatorButtonMask( LEFT_BUTTON_MASK );
SRGP_setInputMode( LOCATOR, EVENT );        /* Zakładamy, że aktywny jest
                                               tylko lokalizator */
/* główna pętla zdarzenia */
terminate = FALSE;
do {
   inputDev = SRGP_waitEvent( INDEFINITE );
   SRGP_getLocator( &measure );
   if (measure.buttonChord[QUIT] == DOWN ) {
      if PickedQuitButton( measure.position ) terminate = TRUE;
      else
         SRGP_marker( measure.position );
   }
}
while ( !terminate );
```

Oczekiwanie na kilka zdarzeń. Fragmenty kodu z programów 2.6 i 2.7 nie ilustrowały największej zalety trybu zdarzeń: możliwości czekania w tym samym czasie na więcej niż jedno urządzenie. SRGP ustawia zdarzenia przychodzące od aktywnych urządzeń w porządku chronologicznym i umożliwia programowi użytkowemu pobranie pierwszego zdarzenia z kolejki za pomocą wywołania SRGP_waitEvent. Stąd zdarzenia są obsługiwane dokładnie w porządku czasowym, inaczej niż w przypadku przerwań sprzętowych, które są przetwarzane w kolejności wynikającej z priorytetów. Program użytkowy sprawdza przesłany mu kod urządzenia i stwierdza, które urządzenie wygenerowało to zdarzenie.

Funkcja pokazana w programie 2.8 umożliwia użytkownikowi umieszczenie dowolnej liczby małych kółeczek (znaczników) w prostokątnym obszarze rysowania. W celu umieszczenia znacznika użytkownik wskazuje odpowiednią pozycję i naciska przycisk 1, a potem kończy współpracę naciskając przycisk 3 albo pisząc „q" lub „Q".

Program 2.8 Równoczesne korzystanie z kilku urządzeń

```
#define PLACE_BUTTON 0
#define QUIT_BUTTON 2

generowanie początkowego ekranu
SRGP_setInputMode( KEYBOARD, EVENT );
SRGP_setKeyboardProcessingMode( RAW );
SRGP_setInputMode( LOCATOR, EVENT );
SRGP_setLocatorButtonMask( LEFT_BUTTON_MASK | RIGHT_BUTTON_MASK );
                                        /* Nie uwzględniamy przycisku 2 */
/* Główna pętla zdarzenia */
terminate = FALSE;
do {
   device = SRGP_waitEvent( INDEFINITE );
   switch ( device ) {
      case KEYBOARD:
         SRGP_getKeyboard( keyMeasure, lbuf );
         terminate = (keyMeasure[0] == 'q') || (keyMeasure[0] == 'Q');
         break;
      case LOCATOR: {
         SRGP_getLocator( &locMeasure );
         switch ( locMeasure.buttonOfMostRecentTransition ) {
            case PLACE_BUTTON:
               if (( locMeasure.buttonChord[PLACE_BUTTON] == DOWN )
                  && InDrawingArea( locMeasure.position ))
                     SRGP_marker( locMeasure.position );
               break;
            case QUIT_BUTTON:
               terminate = TRUE;
               break;
         }  /* case lokalizator */
      }  /* case urządzenie */
   }  /* case przycisk */
}
while (!terminate );
```

2.2.6. Korelacja wskazywania przy obsłudze interakcji

Użytkowy program graficzny zazwyczaj dzieli powierzchnię ekranu na obszary dedykowane różnym zadaniom. Gdy użytkownik naciska przycisk urządzenia lokalizatora, wówczas program użytkowy musi dokładnie określić, który z obiektów został wybrany: przycisk ekranowy, ikona czy inny obiekt lub żaden, po to, żeby mógł odpowiedzieć poprawnie. Ta identyfikacja określana jako *korelacja wskazywania* jest zasadniczym elementem grafiki interakcyjnej.

Program użytkowy wykorzystujący SRGP wykonuje korelację wskazywania na zasadzie określenia, w którym obszarze jest zlokalizowany kursor, a następnie, który obiekt w tym obszarze wybiera użyt-

kownik, jeżeli w ogóle jakiś wybierze. Wskazanie pustego podobszaru może być pominięte (jeżeli na przykład punkt znajduje się między przyciskami w menu) albo może określać pozycję nowego obiektu (jeżeli punkt leży w głównym obszarze rysowania). Ponieważ większość obszarów na ekranie to prostokąty o bokach równoległych do boków ekranu, prawie cała praca związana z korelacją wskazywania może być wykonana za pomocą prostej często używanej funkcji boolowskiej, która sprawdza, czy dany punkt leży wewnątrz prostokąta. Pakiet GEOM rozprowadzany razem z SRGP zawierą tę funkcję (GEOM_ptInRect) oraz inne narzędzia dla arytmetyki dotyczącej współrzędnych. (Więcej informacji na temat korelacji wskazywania zawiera p. 7.11.2.)

Przyjrzyjmy się klasycznemu przykładowi korelacji wskazywania. Rozważmy program malowania z *paskiem menu* na górze ekranu. Ten pasek menu zawiera nagłówki – nazwy rozwijanych menu. Gdy użytkownik wskazuje nagłówek (umieszczając kursor na tekście nagłówka i naciskając przycisk urządzenia wskazującego), wówczas na ekranie pod nagłówkiem zostaje wyświetlone odpowiednie *ciało menu*, a sam nagłówek zostaje podświetlony. Po wybraniu przez użytkownika pozycji menu (w wyniku zwolnienia przycisku lokalizatora) ciało menu znika i znika również podświetlenie nagłówka. Reszta ekranu to pole do rysowania, na którym użytkownik może umieszczać i wskazywać obiekty. Program użytkowy tworząc obiekt przypisuje mu dodatni całkowitoliczbowy identyfikator (ID), który jest przesyłany przez funkcję korelacji wskazywania dla celów dalszego przetwarzania obiektu.

Program 2.9
Schemat interakcji wysokiego poziomu dla obsługi menu

```
void HighLevelInteractionHandler( locatorMeasure measureOfLocator )
{
   if ( GEOM_pointInRect( measureOfLocator.position, menuBarExtent ) {
/* Należy sprawdzić, który nagłówek menu (jeżeli w ogóle jakiś) wybrał użytkownik.
      Następnie rozwinąć to ciało menu */
      menuID = CorrelateMenuBar( measureOfLocator.position );
      if ( menuID > 0 ) {
         chosenItemIndex = PerformPulldownMenuInteraction( menuID );
         if (chosenItemIndex > 0 )
            PerformActionChosenFromMenu(menuID, chosenItemIndex);
      }
 2 else   /* Użytkownik wskazał pole do rysowania; sprawdzić, co zostało wskazane
            i zareagować */
   {
      objectID = CorrelateDrawingArea( measureOfLocator.position );
      if ( objectID > 0 ) ProcessObject( objectID );
   }
}
```

Gdy z lokalizatora po naciśnięciu przycisku zostaje odebrany punkt, wówczas są wykonywane czynności pokazane w programie 2.9; jest to w zasadzie procedura zarządzania, która korzysta z korelacji wskazywania w obrębie paska menu albo pola do rysowania do rozdziału pracy między funkcje wskazywania menu albo obiektu. Jeżeli kursor był w pasku menu, to procedura korelacji określa, czy użytkownik wybrał nagłówek w menu. Jeżeli tak, to jest wywoływana procedura (szczegóły w p. 2.3.1) wykonania interakcji z menu; przesyła ona indeks określający, który element z ciała menu (jeśli w ogóle jakiś) został wybrany. Numer identyfikatora menu razem z indeksem elementu jednoznacznie określają akcję, jaka powinna zostać podjęta. Jeżeli kursor nie był w pasku menu a w polu rysowania, to jest wywoływana inna procedura określająca, czy i który obiekt został wskazany. Jeżeli został wskazany obiekt, to jest wywoływana procedura przetwarzania, która zapewnia właściwą reakcję.

Funkcja CorrelateMenuBar dla każdego nagłówka menu w pasku menu wywołuje GEOM_pointInRect; sięga ona do struktury danych pamiętającej współrzędne pola na ekranie zarezerwowane dla każdego nagłówka. Funkcja CorrelateDrawingArea musi wykonywać bardziej wyrafinowaną operacje, ponieważ, na ogół, obiekty w polu rysowania mogą na siebie zachodzić i nie muszą być prostokątami.

2.2.7. Ustawianie stanu urządzenia i atrybutów

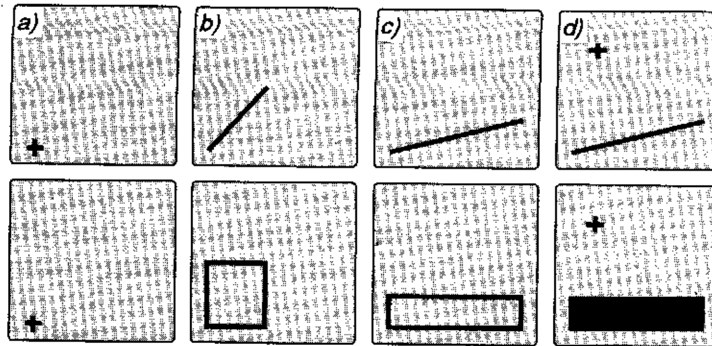
Każde urządzenie wejściowe ma swój własny zbiór atrybutów i program użytkowy może je tak ustawić, żeby uzyskać pożądane sprzężenie zwrotne prezentowane przez urządzenie użytkownikowi. (Omówiona wcześniej maska przycisku też jest atrybutem; różni się od atrybutów przedstawianych tutaj tym, że nie wpływa na sprzężenie zwrotne.) Podobnie jak atrybuty prymitywów wyjściowych, atrybuty urządzeń wejściowych są ustawiane modalnie przez określone funkcje. Atrybuty można ustawiać w dowolnym czasie niezależnie od tego, czy urządzenie jest w stanie aktywnym czy nie.

Dodatkowo stan każdego urządzenia wejściowego, zwykle określany przez akcje użytkownika, może być również określony przez program użytkowy. Inaczej niż w przypadku atrybutów urządzenia wejściowego, stan urządzenia wejściowego przy jego deaktywacji jest sprowadzany do wartości domniemanej; stąd po reaktywacji urządzenia mają na początku dobrze określone wartości, co bez wątpienia jest wygodne dla programisty i dla użytkownika. Stan uzyskany po automatycznym zerowaniu można zmienić ustawiając bezpośrednio inny stan w czasie, gdy urządzenie jest nieaktywne.

Atrybuty typu echo lokalizatorów. Dla lokalizatorów są użyteczne różnego rodzaju echa. Programista może sterować zarówno typem echa, jak i kształtem kursora za pomocą funkcji

void SRGP_setLocatorEchoType (**echoType** NO_ECHO / CURSOR / RUBBER_LINE /RUBBER_RECT);

Domniemaną wartością jest CURSOR; implementacje SRGP mają tablicę kursorów, z której program użytkowy wybiera odpowiedni kształt kursora (por. dokumentacja źródłowa). Powszechnie korzysta się z możliwości dynamicznego określania kształtu kursora zależnie od obszaru, w którym znajduje się kursor. Echa typu RUBBER_LINE i RUBBER_RECT są powszechnie używane do specyfikowania odcinka albo prostokąta. Jeżeli jest ustawiony ten rodzaj echa, to SRGP automatycznie rysuje uaktualniony odcinek albo prostokąt w ślad za tym, jak użytkownik porusza lokalizatorem. Odcinek albo prostokąt są definiowane przez dwa punkty: punkt zaczepienia (jeszcze jeden atrybut urządzenia lokalizatora) i bieżącą pozycję urządzenia lokalizującego. Na rysunku 2.12 pokazano sposób stosowania tych dwóch trybów specyfikowania przez użytkownika odcinka i prostokąta.



Rys. 2.12. Scenariusz dla echa typu gumka: a) naciśnięcie przycisku inicjuje echo; b) pojawienie się prymitywu typu gumka stanowi odpowiedź typu echo na akcję lokalizatora; c) pozycja lokalizatora jest przesyłana do programu użytkowego; d) program użytkowy rysuje odcinek i przywraca echo

Na rysunku 2.12a echo ma postać kursora w kształcie krzyżyka i użytkownik ma nacisnąć przycisk lokalizatora. Program użytkowy w odpowiedzi na naciśnięcie przycisku inicjuje echo typu gumka zaczepiona w bieżącej pozycji kursora. Na rysunkach 2.12b i c w reakcji na poruszenie przez użytkownika lokalizatora pojawia się echo w postaci prymitywu – gumki. Po zwolnieniu przez użytkownika przycisku pozycja lokalizatora na rys. 2.12c jest przesyłana do programu użytkowego, który odpowiada rysując prymityw – odcinek albo prostokąt – i przywracając normalną postać echa, czyli kursor (rys. 2.12d).

Punkt zaczepienia gumki jest ustawiany za pomocą

```
void SRGP_setLocatorEchoRubberAnchor ( point position );
```

Program użytkowy korzysta zazwyczaj z informacji w polu pozycji otrzymanej po ostatnim naciśnięciu przycisku lokalizatora jako danej pozycji zaczepienia – na ogół naciśnięcie tego przycisku inicjuje sekwencję echa typu gumka.

Sterowanie stanem lokalizatora. Gdy lokalizator jest deaktywowany, wówczas pole pozycji wchodzące w skład stanu lokalizatora jest automatycznie ustawiane na środek ekranu. Dopóki programista tego bezpośrednio nie zmieni, dopóty przy ponownej aktywacji jest ustawiony ten sam stan (i pozycja sprzężenia zwrotnego, jeżeli echo jest aktywne). W dowolnej chwili, niezależnie od tego, czy urządzenie jest aktywne czy nieaktywne, programista może przywrócić początkowy stan (w części pola pozycji; nie dotyczy to pól przycisków) za pomocą

```
void SRGP_setLocatorMeasure ( point position );
```

Zerowania stanu wówczas, gdy lokalizator jest nieaktywny nie ma natychmiastowego wpływu na ekran, natomiast zerowanie w czasie, gdy urządzenie jest aktywne, zmienia odpowiednio echo (jeżeli jest). Dlatego jeżeli program żąda, żeby kursor pojawił się początkowo na pozycji innej niż środek, to, gdy urządzenie jest aktywne, wywołanie SRGP_setLocatorMeasure z tą początkową pozycją musi poprzedzać wywołanie SRGP_setInputMode. Ta metoda jest powszechnie stosowana do zachowania ciągłości pozycji kursora: jest zapamiętywany ostatni stan przed deaktywacją urządzenia i kursor wraca na tę pozycję po ponownej aktywacji.

Atrybuty klawiatury i sterowanie stanem. W przeciwieństwie do lokalizatora, którego echo odzwierciedla ruchy fizycznego urządzenia, dla echa klawiatury nie ma oczywistej pozycji. Dlatego pozycja jest atrybutem (z wartością domniemaną, zależną od zastosowania) klawiatury, który można ustawić za pomocą funkcji

```
void SRGP_setKeyboardEchoOrigin ( point origin );
```

Po deaktywacji klawiatury domniemany stan klawiatury jest automatycznie ustawiany na ciąg zerowy. Bezpośrednie ustawienie stanu na początkową niezerową wartość, tuż przed aktywacją klawiatury, jest wygodnym sposobem przedstawienia użytkownikowi domniemanego ciągu wejściowego (wyświetlanego przez SRGP natychmiast po rozpoczęciu wysyłania echa), który może on zaakceptować albo zmienić, zanim naciśnie klawisz Return; minimalizuje się w ten spo-

sób ilość pisania. Stan klawiatury, ciąg znaków, jest ustawiany za pomocą funkcji

```
void SRGP_setKeyboardMeasure ( char *measure );
```

2.3. Cechy grafiki rastrowej

Dotychczas omówiliśmy większość funkcji SRGP. W tym punkcie omówimy pozostałe funkcje, które wykorzystują właściwości sprzętu rastrowego, zwłaszcza możliwość zapamiętania i odtworzenia części ekranu, gdy są one przesłaniane przez inne obrazy, np. okna i chwilowe menu. Takie manipulowanie obrazami jest wykonywane pod kontrolą programów użytkowych zarządzania oknem i menu. Wprowadzimy również mapy bitowe pozaekranowe do pamiętania okien i menu oraz omówimy sposoby korzystania z prostokątów obcinających.

2.3.1. Kanwy

Najlepszy sposób na to, żeby złożone ikony albo menu szybko się pojawiały i znikały, polega na tym, że tworzy się je raz w pamięci, a potem kopiuje się je na ekran wówczas, gdy jest to potrzebne. W pakietach grafiki rastrowej robi się to w ten sposób, że generuje się prymitywy w postaci niewidocznych, pozaekranowych map bitowych albo map pikselowych o potrzebnej wielkości, a następnie kopiuje się je do i z pamięci obrazu. W SRGP te mapy są określane jako *kanwy*. Ta metoda w istocie jest swego rodzaju buforowaniem. Przesyłanie bloków pikseli tam i z powrotem jest w ogólnym przypadku szybsze niż ciągłe generowanie informacji, pod warunkiem istnienia szybkiej operacji SRGP_copyPixel, jaką wkrótce omówimy.

W SRGP kanwa jest to struktura danych, która pamięta obraz jako dwuwymiarową tablicę pikseli. Jest pamiętana również informacja sterująca o wielkości i atrybutach obrazu. Każda kanwa reprezentuje swój obraz w swoim układzie współrzędnych kartezjańskich, który jest identyczny z układem pokazanym na rys. 2.1; sam ekran jest również kanwą, przy czym jest to jedyna kanwa, która jest wyświetlana. W celu wyświetlenia obrazu z kanwy pozaekranowej program użytkowy musi skopiować go do kanwy ekranu. Przedtem część obrazu ekranu, na której pojawi się nowy obraz – na przykład menu – musi być zapamiętana w kanwie pozaekranowej. Po wybraniu opcji w menu jest odtwarzany obraz na ekranie w wyniku ponownego skopiowania zapamiętanych pikseli.

W danej chwili aktywna jest tylko jedna kanwa: kanwa, w której są rysowane nowe prymitywy i do której odnoszą się nowe wartości atrybutów. Tą kanwą może być kanwa ekranu (domniemana wartość, z której korzystaliśmy) albo kanwa pozaekranowa. Współrzędne przekazywane do funkcji prymitywów są wyrażane w lokalnej przestrzeni współrzędnych aktywnej kanwy. Każda kanwa ma również swój własny kompletny zbiór atrybutów SRGP, który wpływa na cały rysunek w tej kanwie; przy tworzeniu kanwy są przyjmowane standardowe domniemane wartości. Wywołanie funkcji ustawiania atrybutu modyfikuje tylko atrybuty w aktywnej kanwie. Wygodnie jest traktować kanwę jak wirtualny ekran o wielkości określonej przez program, mający swoją mapę pikselową, układ współrzędnych i grupę atrybutów. Te właściwości kanwy są często określane jako *stan* albo *kontekst* kanwy.

Przy inicjalizacji SRGP automatycznie jest tworzona kanwa ekranu i ustawiana w stanie aktywnym. Wszystkie nasze dotychczasowe programy generowały swoje prymitywy tylko w tej kanwie. Jest to jedyna kanwa widoczna na ekranie i jej identyfikatorem jest stała SRGP SCREEN_CANVAS. Nowa kanwa pozaekranowa jest tworzona przez wywołanie następującej funkcji, która zwraca identyfikator (liczba całkowita) przypisany nowej kanwie:

```
int SRGP_createCanvas ( int width, int height );
```

Podobnie jak w przypadku ekranu, nowa kanwa ma lokalny układ współrzędnych o początku (0, 0) w lewym dolnym rogu, a prawy górny róg ma współrzędne (szerokość – 1, wysokość – 1). Stąd kanwa 1×1 ma wysokość i szerokość 1 i oba wierzchołki lewy dolny i prawy górny mają współrzędne (0, 0)! Jest to spójne z założeniem, że piksele znajdujące się na przecięciach siatki: jeden piksel w kanwie 1×1 ma współrzędne (0, 0).

Nowo utworzona kanwa jest automatycznie aktywowana i jej piksele mają ustawioną barwę 0 (tak jak w przypadku kanwy ekranu, zanim zostanie wyświetlony jakiś prymityw). Po utworzeniu kanwy jej wielkość nie może ulec zmianie. Również programista nie może zmienić liczby bitów na piksel w kanwie, ponieważ SRGP korzysta z tylu bitów na piksel, ile dopuszcza sprzęt. Atrybuty kanwy są przechowywane jako część informacji o jej stanie lokalnym; dlatego program nie musi bezpośrednio zapamiętać atrybutów bieżącej kanwy aktywnej przed utworzeniem nowej aktywnej kanwy.

Program użytkowy uaktywnia jedną z wcześniej utworzonych kanw za pomocą

```
void SRGP_useCanvas ( int canvasID );
```

Aktywacja kanwy w żadnym przypadku nie oznacza, że kanwa stała się widoczna; na to, żeby tak się stało, trzeba skopiować obraz z niewidocznej kanwy do kanwy ekranu (za pomocą funkcji SGRP_copyPixel).

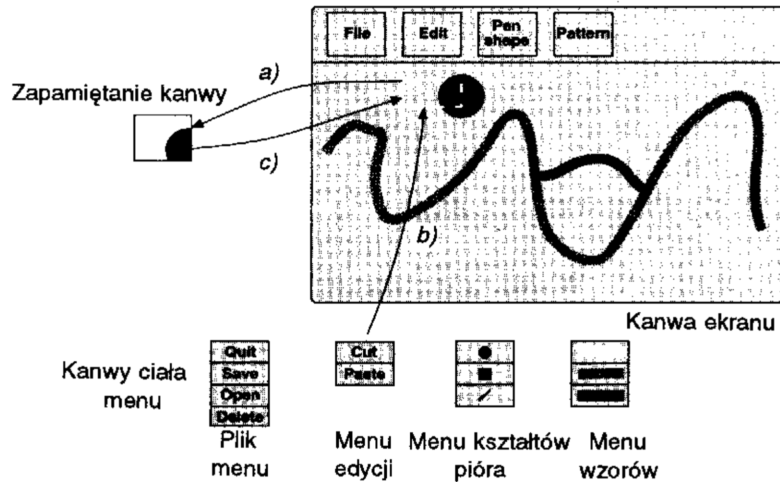
Kanwy są usuwane za pomocą następującej funkcji, która nie może być użyta do usunięcia kanwy ekranu albo kanwy aktywnej:

```
void SRGP_deleteCanvas ( int canvasID );
```

Następujące funkcje umożliwiają zapytanie o wielkość kanwy; jedna z nich przesyła informację o prostokącie, który definiuje system współrzędnych kanwy (lewy dolny punkt zawsze ma współrzędne (0, 0), a druga przesyła wysokość i szerokość jako niezależne wielkości:

```
rectangle SRGP_inquireCanvasExtent ( int canvasID );
void SRGP_inquireCanvasSize ( int canvasID, *width, *height );
```

Zobaczmy, w jaki sposób można wykorzystać kanwy do zrealizowania funkcji PerformPulldownMenuInteraction, wywoływanej przez program 2.9 obsługi interakcji wysokiego poziomu. Funkcja jest realizowana przez pseudokod z programu 2.10, a sekwencję jego działania



Rys. 2.13. Zapamiętanie i odtworzenie obszaru pokrytego przez ciało menu

pokazano na rys. 2.13. Każde menu ma swój unikatowy numer identyfikacyjny (zwracany przez funkcję CorrelateMenuBar), który może być użyty do lokalizowania rekordu danych zawierającego następującą informację o wyglądzie ciała menu:

▻ Identyfikator kanwy pamiętającej ciało menu.

▻ Prostokątne pole (w pseudokodzie jest określane jako *menuBodyScreenExtent*), określone we współrzędnych kanwy ekranu, w której powinno się pojawić ciało menu, gdy użytkownik rozwinie menu po wskazaniu nagłówka i naciśnięciu przycisku menu.

Program 2.10 Pseudokod dla PerformPulldownMenuInteraction

```
int PerformPulldownMenuInteraction( int menuID );
/* Zapamiętywanie i kopiowanie prostokątnych obszarów kanw jest opisane dalej */
{
 podświetlenie nagłówka menu w pasku menu;
 menuBodyScreenExtent=prostokąt obszaru ekranu, w którym powinno się pojawić
 ciało menu
   zapamiętanie bieżących pikseli z menuBodyScreenExtent w chwilowej kanwie
        /* zob. rys. 2.13a */
   kopiowanie obrazu ciała menu z kanwy ciała do menuBodyScreenExtent
        /* zob. rys. 2.13b i kod C w programie 2.11 */
   oczekiwanie na zwolnienie przycisku, co sygnalizuje, że użytkownik dokonał
   wyboru, a następnie pobranie stanu lokalizatora
   kopiowanie zapamiętanego obrazu z chwilowej kanwy z powrotem
   do menuBodyScreenExtent
        /* zob. rys. 2.13c */
   if ( GEOM_pointInRect(measureOfLocator.position, menuBodyScreenExtent )
     obliczenie i zwrócenie indeksu wybranego obiektu, z wykorzystaniem
     współrzędnej y pozycji stanu
   else
      return 0;
}
```

2.3.2. Prostokąty obcinające

W celu zabezpieczenia niektórych części kanwy często trzeba ograniczyć wpływ prymitywów graficznych do podobszaru kanwy aktywnej. Realizację operacji w SRGP zapewnia atrybut *clip rectangle*. Wszystkie prymitywy są obcinane do granic tego prostokąta; oznacza to, że prymitywy (albo części prymitywów) leżące na zewnątrz prostokąta obcinającego nie są rysowane. Podobnie jak każdy inny atrybut, prostokąt obcinający może być zmieniany w dowolnym czasie i jego ostatnia wartość jest zapamiętana razem z grupą atrybutów kanwy. Domniemany prostokąt obcinający (z którego dotychczas korzystaliśmy) jest to pełna kanwa; może on być mniejszy od kanwy, ale nie może wyjść poza granice kanwy. Odpowiednimi wywołaniami ustawiania i zapytywania dla prostokąta obcinającego są

```
void SRGP_setClipRectangle ( rectangle clipRect );
rectangle SRGP_inquireClipRectangle ();
```

Malarski program użytkowy z p. 2.2.4 mógłby używać prostokąta obcinającego do ograniczenia obszaru rysowania ekranu, zabezpieczając tym samym otaczające obszary menu przed zniszczeniem. SRGP oferuje tylko jeden rodzaj prostokąta obcinającego (prostokąt o bokach równoległych do boków ekranu), natomiast niektóre bardziej rozbudowane programy, np. PostScript, oferują wiele obszarów obcinanych o dowolnych kształtach.

2.3.3. Operacja SRGP_copyPixel

Polecenie SRGP_copyPixel jest typowym rastrowym poleceniem określanym często przy realizacji sprzętowej jako bitBlt (bit block transfer) albo pixBlt (pixel Blt); po raz pierwszy polecenie takie było dostępne w mikrokodzie pionierskiej bitmapowej stacji roboczej ALTO w Xerox Palo Alto Research Center na początku lat siedemdziesiątych [INGA81]. To polecenie jest używane do kopiowania tablicy pikseli z prostokątnego obszaru kanwy, obszaru źródłowego, do obszaru *przeznaczenia* w kanwie aktywnej (rys. 2.14). SRGP zapewnia jedynie ograniczoną

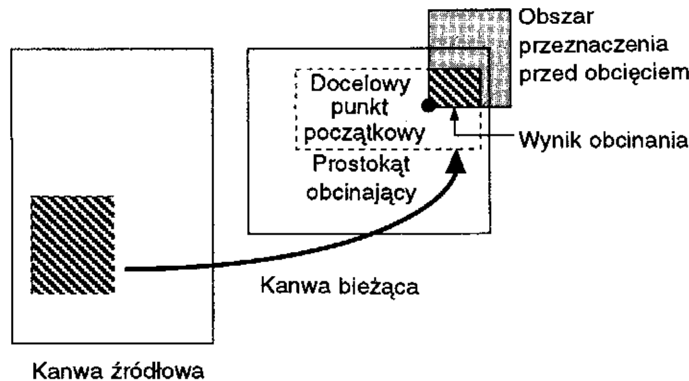
Rys. 2.14. Operacja SRGP_copyPixel

funkcjonalność w tym sensie, że prostokąt przeznaczenia musi być tej samej wielkości co prostokąt źródłowy. W bardziej rozbudowanych wersjach obszar źródłowy może być skopiowany do obszaru przeznaczenia o innej wielkości, z automatycznym skalowaniem. Mogą być również dostępne inne funkcje, takie jak *maski* do selektywnego zabezpieczania odpowiednich pikseli źródłowych albo docelowych przed kopiowaniem i *wzory półtonowe*, które mogą być użyte do przesłaniania (cieniowania) obszaru przeznaczenia.

SRGP_copyPixel może kopiować między dowolnymi dwiema kanwami i jest określona w następujący sposób:

```
void SRGP_copyPixel ( int sourceCanvasID, rectangle sourceRect,
    point destCorner );
```

SourceRect określa obszar źródłowy w dowolnej kanwie, a *destCorner* – lewy dolny róg docelowego prostokąta wewnątrz kanwy aktywnej, każdy w jego własnym systemie współrzędnych. Do operacji kopiowania odnosi się ten sam prostokąt obcinający, który zabezpiecza prymitywy przed generowaniem pikseli w zabezpieczonych obszarach kanwy. Dlatego obszar, do którego ostatecznie piksele są kopiowane, jest przecięciem docelowej kanwy, obszaru przeznaczenia i prostokąta obcinającego; obszar ten jest wyróżniony na rys. 2.15 liniami ukośnymi.



Rys. 2.15. Obcinanie w czasie operacji copyPixel

W celu pokazania sposobu korzystania z copyPixel do obsługi menu rozwijanych, zrealizujemy czwarte polecenie pseudokodu – *kopiowanie obrazu ciała menu* – z funkcji PerformPulldownMenuInteraction (program 2.10). W trzecim poleceniu pseudokodu zapamiętaliśmy w kanwie pozaekranowej obszar ekranu, do którego ma być przesłane ciało menu.

W programie 2.11 pokazano kod w języku C. Musimy umieć rozróżnić dwa prostokąty, które są tej samej wielkości, ale są wyrażone w różnych systemach współrzędnych. Pierwszy prostokąt, który w kodzie jest określany jako *menuBodyExtent*, jest po prostu rozszerzeniem kanwy ciała menu w jego własnym układzie współrzędnych. To rozszerzenie jest używane jako prostokąt źródłowy w operacji SRGP_copyPixel, która umieszcza menu na ekranie. Prostokąt *menuBodyScreenExtent* jest prostokątem o tej samej wielkości, który określa we współrzędnych ekranu pozycję, w której powinno się pojawić ciało menu; lewy dolny róg tego prostokąta jest na tym samym poziomie co

lewy bok nagłówka menu a jego prawy górny róg dotyka dolnego brzegu paska menu. (Na rysunku 2.13 prostokąt dla menu Edit na ekranie jest pokazany linią przerywaną, a prostokąt dla ciała menu – linią ciągłą). Dolny lewy róg prostokąta menuBodyScreenExtent jest używany do określenia obszaru przeznaczenia dla funkcji SRGP_copyPixel, która kopiuje ciało menu (rys. 2.15). Prostokąt ten jest również prostokątem źródłowym dla zapamiętania obszaru ekranu, który ma być przesłonięty przez ciało menu, i prostokątem docelowym dla końcowego odtworzenia.

Zauważmy, że stan programu użytkowego jest zapamiętywany i odtwarzany w celu wyeliminowania efektów ubocznych. Przed kopiowaniem ustawiamy ekranowy prostokąt obcinający jako SCREEN_EXTENT; alternatywnie moglibyśmy ustawić go dokładnie tak jak *menuBodyScreenExtent*.

Program 2.11 Kod kopiowania ciała menu na ekran

```
/* Ten fragment kodu kopiuje obraz ciała menu na ekran,
   w pozycji ekranu zapamiętanej w rekordzie ciała */

/* Zapamiętanie identyfikatora kanwy aktywnej, która nie musi być ekranem */
saveCanvasID = SRGP_inquireActiveCanvas();

/* Zapamiętanie wartości atrybutu prostokąta obcinanjącego kanwy ekranu */
SRGP_useCanvas( SCREEN_CANVAS );
saveClipRectangle = SRGP_inquireClipRectangle();

/* chwilowo tak ustawiamy prostokąt obcinający, żeby było możliwe zapisywanie
   na cały ekran */
SRGP_setClipRectangle( SCREEN_EXTENT );

/* Kopiowanie ciała menu z jego kanwy na obszar pod nagłówkiem menu */
SRGP_copyPixel( menuCanvasID, menuBodyExtent, menuBodyScreenExtent.lLeft );

/* Odtworzenie atrybutów ekranu i kanwy aktywnej */
SRGP_setClipRectangle( saveClipRectangle );
SRGP_useCanvas( saveCanvasID );
```

2.3.4. Tryb Write albo RasterOp

Funkcja SRGP_copyPixel może zrobić więcej niż tylko przesunąć tablicę pikseli z obszaru źródłowego do obszaru przeznaczenia. Może również wykonać logiczną (na poziomie bitów) operację dla każdej pary odpowiadających sobie pikseli w obszarach źródłowym i przeznaczenia, a potem umieścić wynik w obszarze przeznaczenia. Tę operację można symbolicznie zapisać jako

$$D \leftarrow S \text{ op } D$$

przy czym **op**, często określane jako *RasterOp* albo *tryb zapisu*, oznacza na ogół 16 operatorów boolowskich. Jedynie najbardziej popularne spośród nich – **replace**, **or**, **xor** i **and** – są dostępne w SRGP; działanie tych operatorów pokazano na rys. 2.16 dla przypadku, gdy do reprezentacji piksela jest wykorzystany 1 bit.

Tryb zapisu wpływa nie tylko na SRGP_copyPixel, ale również na każdy nowy prymityw zapisany w kanwie. Każdy piksel (albo z prostokąta źródłowego dla SRGP_copyPixel albo z prymitywu) jest zapamiętany w swojej komórce pamięci; jest on zapisany albo w destrukcyjnym trybie **replace**, albo jego wartość jest logicznie łączona z poprzednio zapisaną wartością piksela. (Bitowe operacje na wartościach źródłowej i docelowej są podobne do operacji arytmetycznych albo logicznych wykonywanych przez CPU na zawartości komórki pamięci w czasie cyklu pamięci odczyt-modyfikacja-zapis.). Najczęściej jest stosowany tryb replace, natomiast tryb **xor** jest użyteczny przy generowaniu obiektów dynamicznych, takich jak echo typu kursor albo gumka.

Atrybut trybu zapisu ustawiamy za pomocą funkcji:

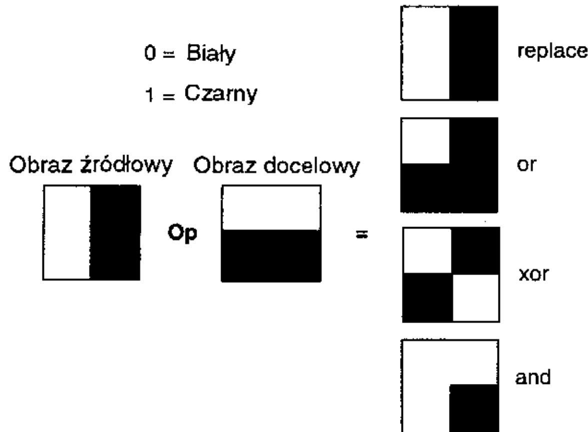
```
void SRGP_setWriteMode ( writeMode WRITE_REPLACE / WRITE_XOR /
    WRITE_OR / WRITE_AND );
```

Ponieważ wszystkie prymitywy są generowane zgodnie z bieżącym trybem zapisu, programista SRGP musi pamiętać o bezpośrednim ustawieniu tego trybu i nie może polegać na domniemanym ustawieniu WRITE_REPLACE.

W celu poznania sposobu działania operacji RasterOp zobaczmy, jak SRGP zapamiętuje i manipuluje pikselami; tylko tutaj aspekty sprzętowe i implementacyjne mają wpływ na abstrakcyjne spojrzenie na grafikę rastrową, jaką zajmowaliśmy się dotychczas.

Operacje RasterOp są wykonywane na wartościach pikseli, które są indeksami do tabeli barw, a nie na sprzętowej specyfikacji barw zapamiętanej jako pozycje w tabeli barw. Dlatego dla systemu dwupoziomowego (1 bit na piksel) operacja RasterOp jest wykonywana na dwóch indeksach 1-bitowych. W systemie barwnym z 8 bitami na piksel operacja RasterOp jest wykonywana jako bitowa operacja logiczna na dwóch indeksach 8-bitowych.

Chociaż interpretacja czterech podstawowych operacji na obrazach monochromatycznych (1 bit na piksel), pokazana na rys. 2.16 jest oczywista, wynik wszystkich trybów poza **replace** nie jast tak samo oczywisty dla obrazów z n bitami na piksel ($n > 1$), ponieważ bitowa operacja logiczna na indeksach źródłowym i docelowym daje trzeci indeks, dla którego wartość barwy może nie mieć żadnego związku z barwami źródłową i docelową.



Rys. 2.16. Tryby zapisu przy łączeniu pikseli źródłowego i docelowego

Tryb **replace** oznacza zapisywanie na tym, co już jest na ekranie (albo na innych kanwach). Ta niszcząca operacja zapisu jest wykorzystywana do rysowania prymitywów i jest powszechnie używana do przesuwania okien i do realizacji okien chwilowych. Może być również używana do wymazywania starych prymitywów na zasadzie rysowania na nich wzoru tła ekranu z danego zastosowania.

W trybie **or** na monitorze dwupoziomowym ma miejsce nieniszczące dodawanie do tego, co już jest na kanwie. Jeżeli 0 reprezentuje białe tło, a 1 plan przedni, to wykonanie operacji **or** szarego wzoru z białym tłem powoduje taką zmianę bitów tła, że pojawi się szary wzór na ekranie. Natomiast wykonanie operacji **or** z czarnym tłem nie da żadnego efektu na ekranie. Wykonanie operacji **or** dla jasnoszarej farby pędzla z wielokątem pokrytym wzorem cegły da w efekcie wzór cegły pokryty wzorem pędzla; czarne krawędzie cegieł nie zostają przy tym usunięte, tak jak by to było w przypadku trybu **replace**. Z tego powodu często malowanie wykonuje się w trybie **or** (zob. zadanie 2.6).

Tryb **xor** w monitorze dwupoziomowym może być użyty do wykonania inwersji obszaru przeznaczenia. Na przykład w celu uwypuklenia (podświetlenia) przycisku wybranego przez użytkownika ustawiamy tryb **xor** i generujemy prostokąt wypełniony barwą 1; w ten sposób dokonujemy zamiany wszystkich pikseli przycisku: 0 **xor** 1 = 1, 1 **xor** 1 = 0. W celu przywrócenia pierwotnego stanu przycisku po prostu ustawiamy tryb **xor** i rysujemy prostokąt jeszcze raz, zmieniając ponownie wartości bitów na przeciwne, co przywraca ich pierwotny stan. Ta metoda jest używana również w SRGP do realizowania echa typu gumowy odcinek lub gumowy prostokąt (zob. zadanie 2.1).

W wielu dwupoziomowych monitorach graficznych metoda **xor** jest wykorzystywana w rozwiązaniach sprzętowych (a czasami programo-

wych) do wyświetlania obrazu kursora lokalizatora w sposób nieniszczący. Są pewne wady związane z tą prostą metodą: jeżeli kursor jest na tle pokrytym drobnym wzorem zawierającym w prawie 50% barwę białą i w 50% barwę czarną, to kursor będzie ledwie widoczny. Dlatego w wielu monitorach dwukolorowych i w większości kolorowych jest wykorzystywany tryb **replace** do realizacji kursora, a sam kursor jest czarny i ma biały kontur, dzięki czemu jest widoczny na każdym tle. Brak możliwości korzystania z trybu **xor** komplikuje sprzęt i oprogramowanie realizujące funkcje echa (zob. zadanie 2.4).

Tryb **and** może być użyty na przykład do selektywnego zerowania pikseli w obszarze przeznaczenia (na barwę 0), co powoduje „wymazanie" tego piksela.

Przykład 2.1 **Problem**: Zrealizuj interakcję z echem typu gumowy prostokąt bez korzystania z wbudowanego echa lokalizatora. Zwróć uwagę na zakłócenia, zwłaszcza na początku i na końcu pętli interakcji.

Odpowiedź Zrealizujemy funkcję echa typu gumowy prostokąt. Podobnie można zrealizować echo typu gumowy odcinek. W czasie ruchu myszki, gdy końcowa wartość lokalizatora jeszcze nie została wybrana, funkcja interakcji będzie śledziła myszkę i rysowała gumkę echa. Pętla interakcji będzie wywoływana po naciśnięciu przez użytkownika przycisku myszki. Interakcja będzie umożliwiała ciągnięcie bieżącego punktu, z echem typu gumka, dopóty, dopóki użytkownik nie zwolni przycisku, sygnalizując tym samym koniec interakcji.

Przy powrocie z funkcji musimy odtworzyć stan, jaki był przed wejściem do funkcji, tak żeby wywołujący funkcję nie musiał się zajmować szczegółami implementacyjnymi funkcji.

Przy tworzeniu echa zasadniczą ideą jest rysowanie kształtu echa (prostokąt) w trybie **xor**, tak żeby przy ponownym rysowaniu tego samego kształtu mógł on znikać i tym samym żeby nie było konieczne wnikanie w to, na czym odbywa się rysowanie. Żeby pojawił się obraz echa, musimy narysować odpowiedni obraz. W celu odtworzenia stanu ekranu, jaki był przed narysowaniem echa, musimy narysować dokładnie ten sam kształt po raz drugi, tak żeby nie pozostał żaden ślad na ekranie.

Ponieważ w celu uaktualniania echa musimy zbierać dane z lokalizatora, za pierwszym razem pojawi się echo typu gumka, zanim zostanie spróbkowany stan lokalizatora; potem w pętli będziemy próbkowali, wymazywali stare echo i rysowali uaktualnione echo. Przy próbkowaniu, przed wymazaniem i ponownym narysowaniem, sprawdzamy stan przycisku, żeby stwierdzić, czy nie powinniśmy zakończyć interakcji. Ponadto, chociaż nie jest to konieczne, dobrze jest sprawdzać, czy myszka zmieniła swoje położenie od czasu ostatniego próbkownia, i wykonać cykl wymazanie-ponowne rysowanie tylko wówczas, gdy rzeczywiście ruch nastąpił.

Przedstawiona koncepcja rozwiązania problemu została zrealizowana w następującym programie w języku C (nie jest pokazana funkcja buttons_are_down(); sprawdza ona jedynie stan przycisków myszki):

Kod, który realizuje interakcję typu echo

```
void RubberRectInteract ( point anchor_pt, point curr_pt, int drag_flag,
        int buttonmask, locatorMeasure *final_loc, rectangle *final_rect)
{
    attributeGroup save_attributes;
    locatorMeasure curr_loc;
    int some_button_down;
    rectangle curr_rect;

    SRGP_inquireAttributes( &save_attributes );
    SRGP_setLineStyle( CONTINUOUS );
    SRGP_setFillStyle( SOLID );
    SRGP_setInputMode( LOCATOR, SAMPLE );
    SRGP_setWriteMode( WRITE_XOR );

    SRGP_setLocatorEchoType( CURSOR )                  /* albo NO_ECHO */;
    SRGP_setLocatorMeasure( curr_pt );

/*
 * Chcemy, żeby prostokąt nie był czuły na to, czy
 * punkt zaczepienia jest poniżej czy z lewej strony bieżącego punktu -
 * zapewnia to funkcja typu GEOM_utility.
*/
    curr_rect = GEOM_rectFromDiagPoints( anchor_pt, curr_pt );
/* Teraz po raz pierwszy pojawi się prostokąt typu gumka: */
    SRGP_rectangle( curr_rect );

    while( (buttons_are_down( buttonmask, curr_loc.button_chord )) ) {
        SRGP_sampleLocator( &curr_loc );
/* Uaktualniamy echo tylko wówczas, gdy myszka poruszyła się; */
            if (curr_loc.position.x != curr_pt.x || curr_loc.position.y != curr_pt.y) {
                SRGP_rectangle( curr_rect );
/* W tym miejscu znika prostokąt, który ostatnio rysowaliśmy */
                curr_pt = curr_loc.position;
                curr_rect = GEOM_rectFromDiagPoints( anchor_pt, curr_pt );
                 SRGP_rectangle( curr_rect );
/* Teraz pojawia się prostokąt określony przez punkt zaczepienia i nowe położenie */
            }
     }

/* W tym miejscu narysowaliśmy po raz pierwszy prostokąt dla ostatniego punktu */
/* Musimy go usunąć, rysując go ponownie */
     SRGP_rectangle( curr_rect );
     *final_loc = curr_loc;
     *final_rect = curr_rect;

/* Teraz odtwarzamy początkowy stan funkcji: */
     SRGP_setInputMode( LOCATOR, INACTIVE );
     SRGP_setAttributes( save_attributes );
  }
```

2.4. Ograniczenia SRGP

Chociaż SRGP jest silnym pakietem i wystarczającym dla dużej klasy zastosowań, ze względu na pewne wewnętrzne ograniczenia dla niektórych zastosowań nie jest optymalnym narzędziem. Przede wszystkim SRGP nie daje wsparcia, jeśli chodzi o wyświetlanie geometrii 3D. Bardziej subtelne ograniczenia wpływają nawet na wiele zastosowań 2D:

- Stosowany w SRGP układ współrzędnych całkowitych zależny od komputera jest zbyt mało elastyczny dla tych zastosowań, które wymagają większej precyzji, zakresu i wygody zmiennego przecinka.
- Podobnie jak w większości rastrowych bibliotek graficznych 2D, SRGP pamięta obraz w kanwie w sposób pozbawiony semantyki, w postaci tablicy nie powiązanych ze sobą wartości pikseli, a nie w postaci zbioru obiektów graficznych (prymitywów) i stąd nie ma operacji na poziomie obiektów, takich jak *delete*, *move* albo *change color*. Ponieważ SRGP nie przechowuje śladu akcji, które doprowadziły do bieżącego obrazu ekranu, nie może również odświeżyć ekranu, jeżeli obraz zostanie zniszczony przez inny program, ani nie może dokonać ponownej konwersji prymitywów w celu utworzenia obrazu, który ma być wyświetlony na urządzeniu o innej rozdzielczości.

2.4.1. Układ współrzędnych zastosowania

W rozdziale 1 przyjęliśmy, że dla większości zastosowań rysunki są tylko środkiem do uzyskania celu, a pierwszoplanową rolą bazy danych zastosowania jest wspieranie takich zadań jak analiza, symulacja, weryfikacja i wytwarzanie. Baza danych musi wobec tego pamiętać informację geometryczną korzystając z zakresu i precyzji potrzebnej dla tych zadań, niezależnie od układu współrzędnych i rozdzielczości urządzenia wyświetlającego. Na przykład program VLSI CAD/CAM może wymagać reprezentowania układów o długości 1 do 2 cm z dokładnością do połowy mikrona, podczas gdy program dla astronomii może potrzebować zakresu od 1 do 10^9 lat świetlnych z precyzją milionów kilometrów. Ze względu na maksymalną elastyczność i zakres wiele zastosowań używa zmiennopozycyjnych *współrzędnych świata* do pamiętania geometrii w swoich bazach danych.

Takie zastosowanie mogłoby samo wykonywać odwzorowanie ze współrzędnych świata do współrzędnych urządzenia; rozważając jednak złożoność takiego odwzorowania (o czym będziemy mówili w rozdz. 6) jest wygodnie korzystać z pakietu graficznego, który akceptuje prymity-

wy określone we współrzędnych świata i odwzorowuje je na urządzenie wyświetlające w sposób niezależny od rodzaju urządzenia. Dostępne obecnie tanie układy scalone zmiennopozycyjne zapewniające prawie tak samo dobre parametry jak dla arytmetyki całkowitoliczbowej istotnie obniżyły zwłokę czasową związaną ze stosowaniem obliczeń zmiennopozycyjnych – ich elastyczność jest warta kosztu, jaki ponosi potrzebujące ich zastosowanie.

Dla grafiki 2D najpopularniejszym oprogramowaniem, które dopuszcza współrzędne zmiennopozycyjne, jest PostScript firmy Adobe, używany zarówno jako standard języka opisu strony do sterowania drukarkami, jak i (dla rozszerzenia o nazwie Display PostScript) pakiet graficzny dla systemu okienek w niektórych stacjach graficznych. Dla zmiennopozycyjnej grafiki 3D obecnie są dostępne PHIGS i PHIGS+ i pojawiają się różne rozszerzenia 3D do PostScript-u.

2.4.2. Pamiętanie prymitywów dla celów ponownej specyfikacji

Zastanówmy się, co się stanie, gdy program użytkowy korzystający z SRGP musi narysować rysunek zmieniając jego wielkość albo przy tej samej wielkości narysować go na urządzeniu o innej rozdzielczości (na przykład na drukarce o dużej rozdzielczości). Ponieważ SRGP nie pamięta prymitywów, które zostały narysowane, program użytkowy po przeskalowaniu współrzędnych musi na nowo podać dla SRGP specyfikację całego zbioru prymitywów, po skalowaniu współrzędnych.

Gdyby rozszerzyć SRGP tak, żeby mógł pamiętać rekord wszystkich specyfikowanych parametrów, program użytkowy mógłby zlecić SRGP odtworzenie ich z własnej pamięci. Wtedy SRGP mógłby realizować inną, często potrzebną operację, a mianowicie odświeżanie ekranu zastosowania. W niektórych systemach graficznych obraz ekranu zastosowania może zostać zniszczony przez komunikaty przychodzące od innych użytkowników albo zastosowania. Dopóki kanwa ekranu nie będzie mogła być odświeżana z redundancyjnej zapamiętanej kopii w kanwie pozaekranowej, dopóty jedynym sposobem naprawienia szkody jest ponowne specyfikowanie prymitywów.

Najważniejszą zaletą istnienia w pakiecie pamięci prymitywów jest możliwość wykonywania operacji edycji, które są istotą programów rysujących albo konstrukcyjnych – jest to klasa programów całkowicie różna od programów malarskich, które były wykorzystywane w przykładach w tym rozdziale. *Program malarski* umożliwia użytkownikowi wykonanie dowolnego pociągnięcia za pomocą pędzli o różnych wielkościach, kształtach, barwach i wzorach. Bardziej rozbudowane programy malarskie umożliwiają również umieszczanie predefiniowanych

kształtów, np. prostokątów, wielokątów i kół. Każda część kanwy może być później edytowana na poziomie pikseli; farbą można pokryć część obiektu, ponadto dowolne regularne obszary kanwy mogą być skopiowane albo przesunięte w inne miejsce. Użytkownik nie może wskazać poprzednio narysowanego kształtu albo namalowanego elementu i potem usunąć go albo przesunąć jako spójny niepodzielny obiekt. Takie ograniczenie istnieje ze względu na to, że dzięki programowi malarskiemu obiekt raz umieszczony na kanwie może być okrojony albo podzielony, tracąc tym samym swoją tożsamość jako obiekt spójny. Na przykład, co mogłoby oznaczać dla użytkownika wskazanie fragmentu obiektu, który został podzielony na części, które były niezależnie rozmieszczone w różnych częściach ekranu? Czy należałoby uważać, że użytkownik odwołał się do fragmentu czy do całości oryginalnego obiektu? W istocie, możliwość oddziaływania na piksele czyni niemożliwą korelację wskazywania, a stąd wskazywania obiektu i edycji.

Program rysujący umożliwia użytkownikowi wskazywanie i edytowanie dowolnego obiektu w dowolnym czasie. Te programy użytkowe, nazywane również *edytorami rozmieszczania* albo *ilustratorami graficznymi*, umożliwiają użytkownikowi umieszczanie standardowych kształtów (nazywanych również symbolami, szablonami albo obiektami) i potem dokonywanie edycji rozmieszczenia na zasadzie usuwania, przesuwania, obracania, skalowania tych kształtów i zmiany ich atrybutów. Podobnie programy interakcyjne, które umożliwiają użytkownikowi tworzenie złożonych obiektów 3D z obiektów prostszych, są nazywane *edytorami geometrycznymi* albo *programami konstrukcyjnymi*.

Skalowanie, odświeżanie i edycja na poziomie obiektów wymagają pamiętania i ponownej specyfikacji prymitywów przez program użytkowy albo przez pakiet graficzny. Jeżeli program użytkowy pamięta prymitywy, to może ponownie dokonać ich specyfikacji; jednak te operacje są bardziej złożone, niż to mogłoby się wydawać na pierwszy rzut oka. Na przykład prymityw można usunąć w sposób trywialny na zasadzie wymazania ekranu i ponownego określenia wszystkich prymitywów (oczywiście poza tym usuniętym); efektywniejsza metoda polega na usunięciu obrazu prymitywu na zasadzie zamalowania go tłem i potem ponownego określenia wszystkich prymitywów, które mogły ulec uszkodzeniu. Ponieważ te operacje są zarówno złożone, jak i często potrzebne, jest to dobra motywacja do przesunięcia ich do pakietu graficznego.

Geometryczny pakiet graficzny na poziomie obiektów, taki jak PHIGS, umożliwia programowi użytkowemu definiowanie obiektów za pomocą zmiennopozycyjnego systemu współrzędnych 2D albo 3D. Pakiet sam pamięta obiekty, umożliwia programowi użytkowemu edycję zapamiętanych obiektów i uaktualnia ekran wówczas, gdy jest to po-

trzebne dla operacji edycji. Pakiet wykonuje również korelację wskazywania, generując identyfikator obiektu, gdy dane są współrzędne ekranowe. Ponieważ te pakiety operują na obiektach, nie zezwalają na operacje na poziomie pikseli (copyPixel i tryb zapisu) – jest to cena za zachowanie spójności obiektów. Tak więc ani pakiet grafiki rastrowej, ani pakiet grafiki geometrycznej z pamięcią prymitywu nie zaspokajają wszystkich potrzeb. W rozdziale 7 są dyskutowane zalety i wady zachowania prymitywów w pakiecie graficznym.

Skalowanie obrazu na zasadzie powielania piksela. Jeżeli ani program użytkowy, ani pakiet nie mają rekordu prymitywów (co jest typowe dla większości programów malarskich), to nie można dokonać skalowania na zasadzie pownownego określenia prymitywów o współrzędnych wierzchołków po skalowaniu. Wszystko, co można zrobić, to skalowanie zawartości kanwy za pomocą operacji czytaj piksel i zapisz piksel. Szybki i prosty sposób skalowania obrazu mapy bitowej/mapy pikselowej (w celu jej powiększenia) polega na *powielaniu piksela*, czyli na zastąpieniu każdego piksela przez blok pikseli $N \times N$, co powoduje powiększenie obrazu N razy.

Przy powielaniu piksela obraz powiększa się, ale równocześnie zatraca swoją ciągłość, ponieważ nie pojawia się żadna nowa informacja poza informacją zawartą w oryginalnej reprezentacji na poziomie pikselowym. Co więcej, powielanie pikseli może zwiększyć wielkość obrazu tylko całkowitą liczbę razy. W celu poprawnego skalowania powinniśmy korzystać z innej metody – próbkowania powierzchni i filtrowania (jest to omawiane w rozdz. 3). Filtrowanie działa najlepiej na mapach pikselowych o głębokości większej niż 1.

Problem skalowania obrazu pojawia się często zwłaszcza wtedy, kiedy obraz utworzony za pomocą programu malarskiego ma być wydrukowany. Rozważmy wysyłanie kanwy do drukarki, która ma dwa razy większą rozdzielczość niż ekran. Każdy piksel jest teraz dwa razy mniejszy; dlatego możemy pokazać oryginalny obraz o tej samej liczbie pikseli, lecz o połowę mniejszy, albo możemy wykorzystać powielanie pikseli w celu utworzenie obrazu o pierwotnej wielkości, ale bez wykorzystania pełnych możliwości drukarki. W każdym przypadku coś jest tracone – wielkość albo jakość – jedynym sposobem skalowania, który nie powoduje utraty jakości jest ponowna specyfikacja.

Podsumowanie

W tym rozdziale omówiliśmy prosty, ale mimo to skuteczny pakiet grafiki rastrowej SRGP. Umożliwia on programowi użytkowemu rysowanie prymitywów 2D z różnymi atrybutami wpływającymi na wygląd

tych prymitywów. Rysowanie może się odbywać bezpośrednio w kanwie ekranu albo w kanwie różnej od ekranu o dowolnej wielkości. Rysowanie może być ograniczone do obszaru prostokątnego kanwy dzięki atrybutowi prostokąta obcinającego. Obok standardowych kształtów 2D SRGP umożliwia kopiowanie prostokątnych obszarów wewnątrz kanwy i między kanwami. Na kopiowanie i rysowanie może mieć wpływ atrybut tryb zapisu, który pozwala na to, żeby bieżąca wartość docelowego piksela miała wpływ na określenie jego nowej wartości.

SRGP wprowadza również logiczne urządzenia wejściowe, które są abstrakcją fizycznych urządzeń wejściowych. Klawiatura SRGP jest abstrakcją fizycznej klawiatury, a lokalizator jest abstrakcją takich urządzeń jak myszka, tabliczka czy drążek sterowniczy. Urządzenia logiczne mogą działać albo w trybie próbkowania, albo w trybie zdarzeń. W trybie zdarzeń akcja użytkownika powoduje umieszczenie raportu o zdarzeniu w kolejce zdarzeń, którą program użytkowy może sprawdzać wtedy, kiedy jest to dla niego wygodne. W trybie próbkowania program użytkowy sprawdza stan urządzenia w sposób ciągły w celu wykrycia pojawienia się istotnych zmian.

Ponieważ SRGP dokonuje konwersji prymitywów do poziomu pikseli i nie zapamiętuje oryginalnej geometrii tych prymitywów, możliwości edycyjne SRGP ograniczają sie tylko do zmiany poszczególnych pikseli, rysowania nowych prymitywów albo korzystania z operacji copyPixel w odniesieniu do bloków pikseli. Operacje na obiektach, np. przesuwanie, usuwanie albo zmiana kształtu muszą być wykonywane przez program użytkowy, który musi podać ponowną specyfikację uaktualnionego obrazu.

Inne systemy oferują różne zestawy funkcji graficznych. Na przykład język PostScript oferuje prymitywy zmiennopozycyjne i atrybuty, włącznie z daleko bardziej ogólnymi kształtami krzywoliniowymi i funkcjami obcinającymi. PHIGS jest pakietem typu podprogram, który umożliwia manipulowanie hierarchicznymi modelami obiektów definiowanymi w trójwymiarowym systemie zmiennopozycyjnych współrzędnych świata. Te obiekty są zapamiętywane w edycyjnej bazie danych; pakiet automatycznie generuje obraz na nowo na podstawie reprezentacji zapamiętanej po każdej operacji edycyjnej.

SRGP jest pakietem typu podprogram; uważa się, że język interpretacyjny taki jak PostScript firmy Adobe daje maksymalną moc i elastyczność. Są również różne opinie co do tego, co powinno się stać standardem – pakiety typu podprogram (całkowitoliczbowe czy zmiennopozycyjne, z lub bez zachowywania prymitywów) czy języki wyświetlania, np. PostScript, które nie zachowują prymitywów. Każde rozwiązanie ma swój obszar zastosowań i spodziewamy się, że każde przetrwa przez pewien czas.

W następnym rozdziale zobaczymy, jak SRGP realizuje rysowanie metodą rasteryzacji i obcinania. W kolejnych rozdziałach, po zapoznaniu się z aspektami sprzętowymi, omawiamy matematykę przekształceń i rzutów 3D, co przygotuje nas do poznania standardu PHIGS.

Zadania

2.1. SRGP działa w środowisku okien, ale nie umożliwia programowi użytkowemu korzystania z zalet wielu okien. Kanwa ekranu jest odwzorowywana na jedno okno na ekranie i żadna inna kanwa nie jest widoczna. Jakie można zrobić zmiany w projekcie i programowym interfejsie programu użytkowego, żeby mógł on korzystać z zalet systemu okien?

2.2. Program użytkowy SRGP może być w pełni niezależny od komputera wówczas, gdy korzysta tylko z dwóch barw 0 i 1. Opracuj strategię takiego rozbudowania SRGP, żeby SRGP w razie potrzeby symulował barwę, co umożliwiłoby takie projektowanie programu użytkowego, żeby mógł on korzystać z zalet, jakie daje barwa, a mimo to wygodnie działać na monitorze dwupoziomowym. Przedyskutuj problemy i konflikty, jakie stwarza każda tego typu strategia.

2.3. Zrealizuj sekwencję animacji, w której kilka prostych obiektów porusza się i zmienia wielkość. Najpierw wygeneruj każdą ramkę na zasadzie wyzerowania ekranu i potem podania specyfikacji obiektów w ich nowych pozycjach. Następnie wypróbuj podwójne buforowanie: użyj kanwy pozaekranowej jako bufora, w którym jest rysowana każda ramka, zanim zostanie skopiowana do kanwy ekranu. Porównaj wyniki działania tych dwóch metod. Rozważ również możliwość użycia funkcji SRGP_copyPixel. W jakich ograniczonych warunkach jest ona użyteczna dla animacji?

2.4. Zrealizuj niedestrukcyjne śledzenie kursora korzystając z wbudowanego w SRGP kursora realizującego echo. Wykorzystaj wzór mapy bitowej albo pikselowej do pamiętania obrazu kursora, przy czym 0 we wzorze będzie odpowiadało przezroczystości. Zrealizuj kursor typu **xor** na monitorze dwupoziomowym i kursor działający w trybie replace na monitorze dwupoziomowym albo kolorowym. W celu testowania śledzenia należy zrealizować pętlę próbkowania z lokalizatorem SRGP i przesuwać kursor nad obszarem zawierającym informację zapisaną wcześniej.

2.5. Zastanów się nad implementacją następującej funkcji w użytkowym programie malarskim. Użytkownik może w trybie **xor** zmieniać barwę w obrębie śladu pędzla na przeciwną. Mogłoby się wydawać, że można to łatwo zaimplementować przez ustawienie trybu zapisu i następnie wykonywanie kodu programu 2.5. Jakie powstaną komplikacje? Zaproponuj rozwiązanie.

2.6. Niektóre użytkowe programy malarskie mają tryb malowania rozpylaczem, w którym przesuwanie pędzla nad powierzchnią wpływa losowo

na niewielką liczbę pikseli w tym obszarze. Za każdym razem, gdy pędzel przesuwa się nad obszarem, wpływa na inne piksele i po każdym przejściu pędzla pozostaje gęstszy ślad farby. Zrealizuj interakcję dla malowania rozpylaczem dla monitora dwupoziomego. (*Uwaga*: Najbardziej oczywiste algorytmy tworzą smugi albo nie dają w efekcie wzrastającej gęstości. Trzeba będzie stworzyć bibliotekę rzadkich map bitowych albo wzorów; w materiałach referencyjnych można znaleźć informacje na temat tworzenia typowych wzorów.)

2.7. Dla monitora dwupoziomego zrealizuj duży znak, którego wnętrze jest przezroczyste dla tła, bez korzystania z wbudowanego w SRGP prymitywu tekstu. Wykorzystaj kanwę pozaekranową do pamiętania kształtu mapy bitowej dla każdego znaku (uwzględnij najwyżej sześć znaków) – to nie jest lekcja projektowania kroju pisma! (*Wskazówka*: Być może, trzeba będzie użyć dwóch różnych algorytmów dla barw 0 i 1.)

2.8. Program rysujący może uaktualnić ekran po operacji usuwania na zasadzie wypełniania kształtu usuwanego obiektu wzorem tła ekranu zastosowania. Oczywiście ta metoda może zniszczyć inne obiekty na ekranie. Dlaczego nie wystarczy naprawić uszkodzenia na zasadzie zwykłego ponownego wyspecyfikowania wszystkich obiektów, dla których opisany na nich prostokąt przecina prostokąt opisany na usuniętym obiekcie? Przedyskutuj rozwiązania problemu optymalizacji naprawy uszkodzenia.

2.9. Zrealizuj funkcję, która rysuje przycisk z tekstem w środku nieprzezroczystego prostokąta o cienkim konturze. Umożliw wywołującemu określanie barwy tekstu, tła i konturu; pozycji na ekranie, w której powinien być umieszczony środek przycisku; parę wymiarów min/maks zarówno dla szerokości, jak i dla wysokości; krój pisma i sam ciąg znaków tekstu. Jeżeli ciąg znaków nie mieści się w jednym wierszu, w przycisku w jego najdłuższym wierszu, to podziel ciąg w odpowiednich miejscach (na przykład tam, gdzie są spacje) tak, żeby powstał tekst wielowierszowy dla danego przycisku.

2.10. Zrealizuj ekranowe logiczne urządzenie wejściowe przeznaczone do wprowadzania wartości, które umożliwi użytkownikowi określanie temperatury za pomocą myszki zmieniającej długość symulowanego słupka rtęci. Wsród atrybutów urządzenia powinien być zakres pomiaru, początkowa wielkość, wymagana dokładność pomiaru (na przykład 2°C), długość oraz pozycja obrazu termometru na ekranie. W celu przetestowania urządzenia wykorzystaj interakcję, która symuluje nieskończone oczekiwanie na zdarzenie (waitEvent) i jedynym aktywnym urządzeniem jest urządzenie do wprowadzania wartości.

2.11. Rozważ rozbudowę implementacji SRGP o ekranowe urządzenie do wprowadzania wartości (takie jak w przykładzie 2.10), które może działać zarówno w trybie próbkowania, jak i w trybie zdarzeń. Jakie mogą

się pojawić problemy, jeżeli implementacja zostanie zainstalowana na stacji roboczej, która ma tylko fizyczne urządzenie wskazujące? Zaproponuj rozwiązanie.

2.12. Zrealizuj prymityw prostokąta z zaokrąglonymi narożnikami o kształcie łuku elipsy o kącie 90°. Należy dopuścić rysowanie tylko konturu i wersji wypełnionej.

Projekty programowe

2.13. Zrealizuj pakiet rozwijanego menu, którego projekt wysokiego poziomu przedstawiono we fragmentach kodu w p. 2.2.6, 2.3.1 i 2.3.3. Inicjalizację paska menu i ciała menu zraealizuj na zasadzie czytania ciągów z pliku wejściowego. Program powinien móc deaktywować menu tak, żeby znikał nagłówek, i aktywować menu (parametrem jest położenie poziome na pasku menu) tak, żeby się ono pojawiło.

2.14. Ulepsz pakiet menu z projektu 2.13 przez włączenie funkcji blokowania wybranych elementów menu. Zablokowane elementy w ciele menu powinny pojawić się na szarym tle; ponieważ SRGP nie umożliwia rysowania tekstu za pomocą stylu pióra, w celu uzyskania pożądanego efektu na monitorze dwupoziomowym musimy malować na tekście za pomocą trybu zapisu.

2.15. Ulepsz pakiet menu z zadania 2.13 wprowadzając podświetlanie elementu wskazywanego przez lokalizator wówczas, gdy użytkownik wybiera element z ciała menu.

2.16. Zrealizuj program rozmieszczania, który umożliwi użytkownikowi umieszczenie obiektu w kwadratowym podobszarze ekranu. Powinna być możliwość umieszczania elips, prostokątów i trójkątów równobocznych. Użytkownik będzie wybierał typ obiektu i inicjował akcję (przerysowanie ekranu, zapamiętanie sceny w pliku, odtworzenie sceny z pliku, wyjście z programu), naciskając odpowiedni przycisk ekranowy.

2.17. Do programu rozmieszczania z zadania 2.16 dodaj edycję obiektu. Użytkownik musi mieć możliwość usunięcia, przesunięcia oraz zmiany wielkości albo przeskalowania obiektów. Wykorzystaj następującą prostą metodę korelacji wskazywania: przeszukaj obiekty w programie użytkowym i wybierz pierwszy obiekt, którego opisany prostokąt obejmuje pozycję wskazywaną przez lokalizator. (Pokaż, że ta naiwna metoda ma zakłócający efekt uboczny: istnieje możliwość, że nie będzie można wskazać widocznego obiektu!) Zwróć uwagę na to, żeby zapewnić użytkownikowi sprzężenie zwrotne na zasadzie podświetlania wybranego obiektu.

2.18. Do programu rozmieszczania z zadania 2.16 dodaj dodatkowe pole wymiaru wprowadzając priorytet nakładania. Użytkownik musi móc chować/odkrywać obiekt (wymuszać, żeby jego priorytet stał się najmniej-

szy/największy). Ulepsz wskazywanie korelacyjne o wykorzystywanie priorytetu nakładania do rozwiązywania konfliktów. W jaki sposób ta funkcja chowania/odkrywania wraz z priorytetowym działaniem wskazywania korelacyjnego umożliwia użytkownikowi unikania niedokładności wskazywania korelacyjnego?

2.19. Zoptymalizuj algorytm uaktualniania ekranu w programie rozmieszczania z zadania 2.16, wykorzystując wyniki zadania 2.8, tak żeby w odpowiedzi na operację edycji trzeba było podawać ponownie specyfikację dla możliwie małej liczby obiektów.

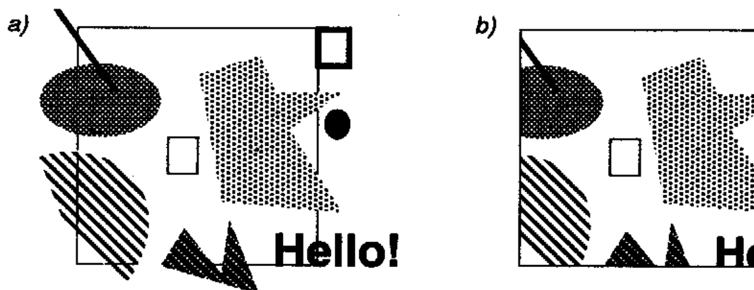
2.20. Ulepsz program rozmieszczania z zadania 2.16 tak, żeby klawiatura i lokalizator mogły być równocześnie aktywne po to, żeby popularnym operacjom mogły być przypisane skróty klawiaturowe. Na przykład naciskając klawisz „d" powinniśmy móc usunąć wskazany obiekt.

2.21. Zaprojektuj i zaimplementuj analityczne metody wskazywania korelacyjnego dla trzech typów obiektów dozwolonych w programie rozmieszczania z zadania 2.16. Nowe metody powinny zapewnić pełną dokładność; użytkownik nie powinien już korzystać z metody chowania/odkrywania do wskazywania widocznego obiektu o niskim priorytecie.

3. Podstawowe algorytmy rysowania prymitywów 2D w grafice rastrowej

Pakiet grafiki rastrowej aproksymuje prymitywy matematyczne (idealne), opisane przez wierzchołki siatki kartezjańskiej, za pomocą zbiorów pikseli o odpowiednim poziomie szarości lub barwie. Te piksele są pamiętane w postaci mapy bitowej albo pikselowej w pamięci CPU albo w pamięci obrazu. W poprzednim rozdziale poznaliśmy właściwości SRGP – typowego pakietu grafiki rastrowej – z punktu widzenia programisty piszącego programy użytkowe. Celem tego rozdziału jest spojrzenie na SRGP z punktu widzenia implementatora pakietu – to znaczy z punktu widzenia podstawowych algorytmów rasteryzacji prymitywów z uwzględnieniem atrybutów i obcinaniem przez prostokąt o bokach równoległych do boków ekranu. Na rysunku 3.1 pokazano przykłady prymitywów po rasteryzacji i obcinaniu.

W bardziej wyrafinowanych i złożonych pakietach są stosowane bardziej zaawansowane algorytmy niż te, które są wykorzystywane



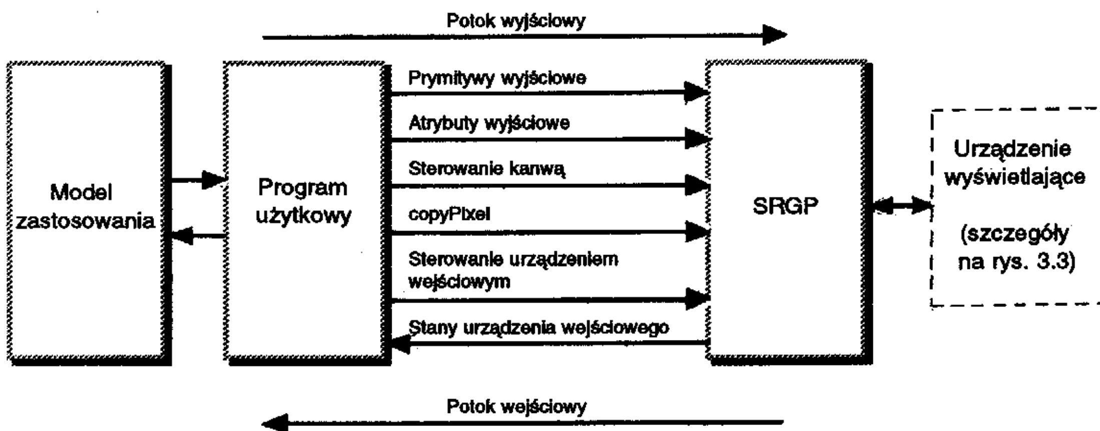
Rys. 3.1. Obcinanie prymitywów SRGP przez prostokątne okno: a) prymitywy i prostokąt obcinający; b) wynik obcinania

w SRGP. W tym rozdziale algorytmy są rozważane w kategoriach całkowitoliczbowej dwuwymiarowej siatki kartezjańskiej; większość algorytmów rasteryzacji może być rozszerzona na liczby zmiennopozycyjne, a algorytmy obcinania mogą być rozszerzone na liczby zmiennopozycyjne i na przestrzeń trójwymiarową. Końcowy punkt wprowadza koncepcję usuwania zakłóceń (antyaliasingu), to znaczy minimalizowania zakłóceń z wykorzystaniem zdolności systemu do zmieniania intensywności pikseli.

3.1. Przegląd

3.1.1. Wpływ architektury systemu wyświetlania

Podstawowy ogólny model z rozdz. 1 przedstawia pakiet graficzny jako system, który pośredniczy między programem użytkowym (i jego modelem/strukturą danych użytkowych) a sprzętem wyświetlającym. Pakiet zapewnia programowi użytkowemu interfejs ze sprzętem niezależny od urządzenia, tak jak to pokazano na rys. 3.2, gdzie funkcje SRGP są podzielone na te, które tworzą potok wyjściowy, i na te, które tworzą potok wejściowy.



Rys. 3.2. SRGP jako element pośredniczący między programem użytkowym a systemem graficznym, z potokami wyjściowym i wejściowym

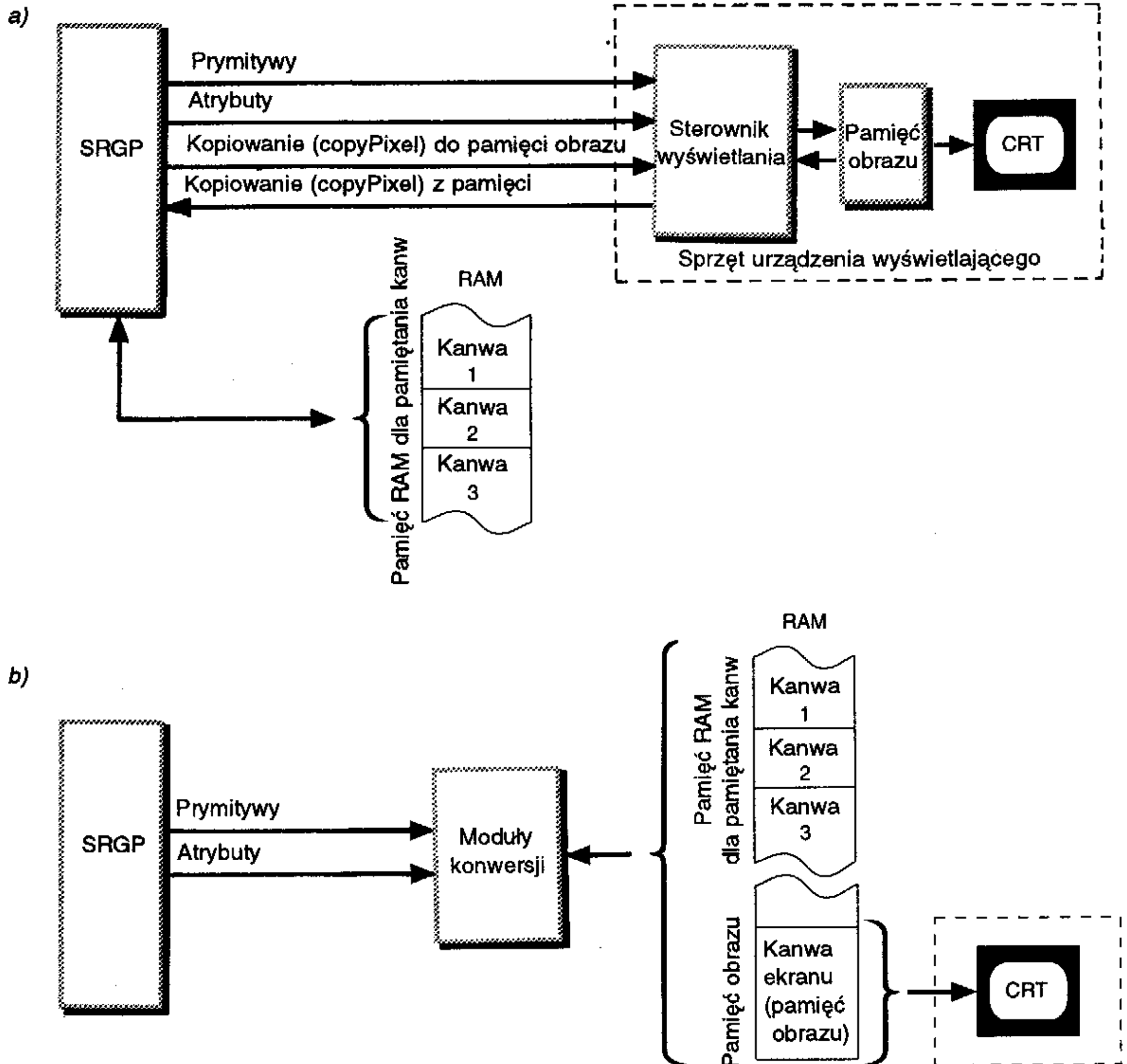
W *potoku wyjściowym* program użytkowy korzysta z opisów obiektów w postaci prymitywów i atrybutów zapamiętanych albo otrzymanych z modelu zastosowania albo ze struktury danych i specyfikuje je dla pakietu graficznego, który z kolei wykonuje obcinanie i konwersję

na piksele widziane na ekranie. Funkcje generowania prymitywów w pakiecie określają, co ma być generowane, funkcje atrybutu określają, jak prymitywy mają być generowane, funkcja SRGP_copyPixel określa, jak obrazy mają być modyfikowane, a funkcje sterowania kanwą określają, gdzie obrazy mają być generowane. W *potoku wejściowym* interakcja użytkownika z monitorem jest zamieniana na wartości stanu, przesyłane do programu użytkowego przez funkcje wejściowe pakietu próbkowania i obsługi zdarzeń; typowy potok wykorzystuje te wartości do modyfikowania modelu albo obrazu na ekranie. Do funkcji związanych z wejściem należą te, które inicjalizują urządzenia wejściowe i sterują nimi, i te, które rejestrują późniejsze stany w czasie interakcji. W książce nie zajmujemy się ani zarządzaniem kanwami w SRGP, ani obsługą wejść, ponieważ te zagadnienia niewiele mają wspólnego z grafiką rastrową i są głównie związane ze strukturą danych i z oprogramowaniem systemu na niskim poziomie.

Implementacja SRGP musi się komunikować z różnymi urządzeniami wyświetlającymi. Niektóre systemy wyświetlające są przyłączone jako urządzenia zewnętrzne ze swoimi własnymi wewnętrznymi pamięciami obrazu i sterownikami wyświetlania. Sterowniki wyświetlania są to specjalizowane procesory mające za zadanie interpretowanie i wykonywanie poleceń rysowania, które generują piksele do pamięci obrazu. Inne, prostsze systemy są odświeżane bezpośrednio z pamięci wykorzystywanej przez CPU. Podzbiory pakietu typu tylko wyjście mogą sterować rastrowymi urządzeniami kopiującymi. Różne rodzaje architektur sprzętowych będą dokładniej omówione w rozdz. 4. W dowolnej architekturze systemu wyświetlania CPU musi móc czytać i zapisywać piksele w pamięci obrazu. Dogodnie jest również móc przesuwać prostokątne bloki pikseli do i z pamięci obrazu w celu realizowania operacji typu copyPixel (bitBlt). Ta możliwość nie jest używana bezpośrednio do generowania prymitywów, ale do tego, żeby wyświetlać fragmenty map bitowych albo pozaekranowych i zapamiętywać albo odtwarzać fragmenty ekranu przy zarządzaniu oknami, obsłudze menu, przewijaniu itd.

Wszystkie implementacje dla systemów z odświeżaniem pamięci CPU są w zasadzie identyczne, ponieważ cała praca jest wykonywana programowo. Natomiast implementacje dla sterowników wyświetlania i systemów generujących trwałe kopie różnią się między sobą w znacznym stopniu, zależnie od tego, co mogą odpowiednie urządzenia zrobić same, a co pozostaje do zrobienia dla oprogramowania. Przyjrzyjmy się różnym architekturom i implementacjom.

Urządzenia wyświetlające z pamięcią obrazu i sterownikami wyświetlania. Jeżeli sterownik wyświetlania sam dokonuje konwersji i bezpośrednio obsługuje wszystkie prymitywy i atrybuty SRGP, to SRGP ma



Rys. 3.3. Sterowanie przez SRGP dwóch typów urządzeń wyświetlających: a) monitor jako urządzenie zewnętrzne ze sterownikiem wyświetlania i z pamięcią obrazu; b) brak sterownika wyświetlania, pamięć obrazu jako część ogólnego bloku pamięci

niewiele do roboty. W takim przypadku SRGP musi tylko wykonywać konwersję swojej wewnętrznej reprezentacji prymitywów, atrybutów i trybu zapisu na formaty akceptowane przez urządzenia wyświetlające, które faktycznie rysują prymitywy (rys. 3.3a).

Sterownik wyświetlania ma największą moc wówczas, gdy odwzorowanie pamięci umożliwia CPU bezpośredni dostęp do pamięci obra-

zu, a sterownik wyświetlania ma dostęp do pamięci CPU. Wtedy CPU może czytać i pisać piksele i kopiować (copyPixel) bloki pikseli za pomocą zwykłych rozkazów CPU, a sterownik wyświetlania może wykonywać konwersję kanw pozaekranowych i również używać polecenia copyPixel do przesuwania pikseli między dwiema pamięciami albo w ramach swojej własnej pamięci. Jeżeli CPU i sterownik wyświetlania mogą działać asynchronicznie, to trzeba zapewnić synchronizację dla unikania konfliktów pamięci. Często sterownik wyświetlania jest sterowany przez CPU jako koprocesor. Jeżeli sterownik wyświetlania urządzenia wyświetlającego może wykonywać tylko konwersję do swojej własnej pamięci obrazu i nie może zapisywać pikseli do pamięci CPU, to jest potrzebny sposób generowania prymitywów w kanwie pozaekranowej. Wtedy pakiet wykorzystuje sterownik wyświetlania do konwersji do kanwy ekranowej, ale musi wykonać własny program konwersji do kanw pozaekranowych. Oczywiście pakiet może wykonywać operację copyPixel w stosunku do obrazów po konwersji sprzętowej, z pamięci obrazu do kanw pozaekranowych.

Monitory mające tylko pamięć obrazu. W przypadku monitorów bez sterownika wyświetlania SRGP sam wykonuje konwersję zarówno do kanw pozaekranowych, jak i do pamięci obrazu. Na rysunku 3.3b pokazano typową organizację implementacji SRGP, która steruje pamięcią obrazu znajdującą się w obszarze pamięci dzielonej. Zauważmy, że pokazano tylko te części pamięci, które tworzą pamięć obrazu i pamiętają kanwy zarządzane przez SRGP; reszta pamięci jest zajęta przez zwykłe programy i dane, w tym przez sam SRGP.

Urządzenia tworzące trwałe kopie. Jak wyjaśnimy w rozdz. 4, istnieje wiele różnych urządzeń tworzących trwałe kopie, podobnie jak istnieje wiele różnych urządzeń wyświetlających. Najprostsze urządzenia przyjmują na raz tylko pojedynczy wiersz i program musi dostarczyć dokładną postać wiersza, gdy ma być on zobrazowany na filmie albo na papierze. Dla tak prostego sprzętu SRGP musi generować kompletną mapę bitową albo pikselową i wysyłać do urządzenia wyjściowego jeden wiersz na raz. Nieco lepsze urządzenia mogą przyjmować na raz całą ramkę (stronę). Jeszcze lepsze urządzenia mają wbudowane układy konwersji, nazywane czasami procesorami rasteryzacji obrazu (RIP). Najlepsze urządzenia, drukarki PostScriptowe, mają wewnętrzne układy, które czytają programy PostScriptowe opisujące strony w sposób niezależny od urządzenia; wynikiem interpretacji takich programów są prymitywy i atrybuty, które są potem poddawane konwersji. Podstawowe algorytmy obcinania i konwersji są w zasadzie niezależne od technologii wyprowadzania przez urządzenie rastrowe; dlatego nie musimy się dalej zajmować w tym rozdziale urządzeniami do tworzenia trwałych kopii.

3.1.2. Programowy potok wyjściowy

Omówimy teraz potok wyjściowy sterujący prostym monitorem z buforem ramki jedynie po to, żeby poruszyć problem programowego obcinania i konwersji. Wprowadzone różne algorytmy są omawiane na ogólnym, niezależnym od komputera poziomie i stosują się do implementacji sprzętowych (mikrokod) albo programowych.

Po napotkaniu przez SRGP prymitywu wyjściowego, pakiet dokonuje konwersji prymitywu; piksele są zapisywane do bieżącej kanwy zgodnie z przypisanymi im atrybutami i trybem zapisu. Prymityw jest również obcinany przez prostokąt obcinający; to znaczy piksele należące do prymitywu, ale leżące na zewnątrz obszaru obcinającego nie są wyświetlane. Jest kilka metod realizacji obcinania. Oczywista metoda polega na obcięciu prymitywu przed konwersją, na zasadzie obliczenia w drodze analitycznej przecięcia z granicami prostokąta obcinającego; punkty przecięcia są potem wykorzystywane do definiowania nowych wierzchołków dla obciętej wersji prymitywu. Zaletą obcinania przed konwersją jest to, że konwerter musi się zajmować tylko obciętą wersją prymitywu, a nie oryginalną (być może znacznie większą). Ta metoda jest używana najczęściej do obcinania odcinków, prostokątów i wielokątów, dla których algorytmy obcinania są dość proste i wydajne.

Najprostsza metoda obcinania określana jako *wycinanie*, polega na konwersji całego prymitywu i zapisie tylko widocznych pikseli do obszaru kanwy określonego przez prostokąt obcinający. W zasadzie ta procedura porównuje współrzędne każdego piksela z granicami (x, y) prostokąta przed zapisaniem tego piksela. W praktyce korzysta się z metod uproszczonych, które nie wymagają sprawdzania każdego piksela w wierszu. Ten typ obcinania jest wykonywany szybko; jeżeli porównywanie z granicami prostokąta może być wykonane szybko (na przykład przez małą wewnętrzną pętlę wykonywaną całkowicie w mikrokodzie albo pamięci podręcznej rozkazów), to takie podejście może być szybsze niż obcinanie prymitywu, a następnie konwersja wyników obcinania. Ponadto takie rozwiązanie można uogólnić na dowolne obszary obcinające.

Trzecia metoda polega na generowaniu całego zbioru prymitywów do tymczasowej kanwy, a następnie przepisaniu do kanwy docelowej za pomocą copyPixel tylko zawartości prostokąta obcinającego. Przy takim podejściu traci się zarówno miejsce, jak i czas, ale jest ono łatwe do implementacji i jest często używane w odniesieniu do tekstu.

Monitory rastrowe wywołują algorytmy obcinania i konwersji za każdym razem, gdy obraz jest tworzony albo modyfikowany. Dlatego te algorytmy nie tylko muszą tworzyć wzrokowo satysfakcjonujące ob-

razy, ale muszą być możliwie najszybsze. Jak pokażemy w następnym punkcie, w algorytmach konwersji wykorzystuje się *metody przyrostowe* w celu minimalizacji liczby obliczeń (zwłaszcza dzieleń i mnożeń) wykonywanych w czasie każdej iteracji; w tych obliczeniach korzysta się raczej z arytmetyki całkowitoliczbowej niż zmiennopozycyjnej. Szybkość można zwiększyć jeszcze bardziej korzystając z wielu procesorów równoległych do równoczesnej konwersji całych wyjściowych prymitywów albo ich fragmentów.

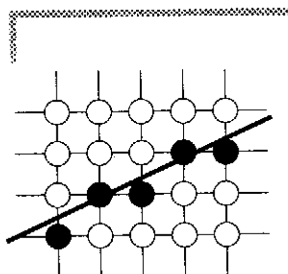
3.2. Konwersja odcinków

Algorytm konwersji odcinka oblicza współrzędne pikseli, które leżą na lub blisko idealnej nieskończenie cienkiej linii prostej nałożonej na siatkę dwuwymiarowego rastra. W zasadzie chcielibyśmy, żeby sekwencja pikseli leżała tak blisko idealnego odcinka, jak tylko jest to możliwe, i żeby był on możliwie prosty. Rozważmy przybliżenie idealnego odcinka, mające szerokość jednego piksela; jakie powinno mieć ono właściwości? Dla odcinków o nachyleniu z przedziału $[-1, 1]$ (włącznie) w każdej kolumnie powinien być wyświetlony tylko jeden piksel; dla odcinków o nachyleniu spoza tego przedziału dokładnie jeden piksel powinien być wyświetlony w każdym wierszu. Wszystkie odcinki powinny być rysowane ze stałą jasnością, niezależnie od długości i orientacji odcinka i tak szybko, jak to jest możliwe. Powinna również istnieć możliwość rysowania odcinków o grubości większej niż 1 piksel, których oś symetrii pokrywa się z idealnym odcinkiem, których wygląd zależy od atrybutów stylu linii oraz stylu pióra i spełnia wymagania stawiane ilustracjom dobrej jakości. Na przykład kształt obszarów, gdzie leżą końce odcinka, powinien być pod kontrolą programisty, który może wybrać prostokątne, zaokrąglone lub ścięte ukośnie zakończenie odcinka. Chcielibyśmy nawet mieć możliwość minimalizowania zębatych zakłóceń wynikających z dyskretnej aproksymacji idealnego odcinka, za pomocą metod odkłócania wykorzystujących możliwość ustalania jasności pikseli przy n-bitowej reprezentacji piksela.

Na razie ograniczymy się do odcinków o grubości 1 piksela, które w każdej kolumnie (albo w wierszu dla odcinków o nachyleniu $> +1$ albo < -1) mają dokładnie 1 piksel. W dalszej części rozdziału zajmiemy się grubymi prymitywami oraz stylami.

Przypomnijmy, że SRGP reprezentuje piksel jako kółko o środku w punkcie (x, y) w siatce całkowitoliczbowej. Taka reprezentacja jest wygodną aproksymacją mniej lub bardziej okrągłego przekroju strumienia elektronów w lampie kineskopowej; jednak odległości między plamkami na ekranie mogą się w znacznym stopniu zmieniać w różnych

systemach. W niektórych systemach sąsiednie plamki nakładają się; w innych mogą być przerwy między pikselami sąsiadującymi w pionie; w większości systemów odległości są mniejsze w poziomie niż w pionie. Inna reprezentacja występuje w systemach takich jak Macintosh, w których przyjmuje się, że piksele leżą w środkach prostokątnych pól między sąsiednimi liniami siatki, a nie na samych liniach siatki. Przy takiej reprezentacji prostokąty reprezentują wnętrze piksela (z punktu widzenia matematycznego prostokąty są określane przez dwa wierzchołki). Taka definicja dopuszcza kanwy o zerowej szerokości: prostokąt od (x, y) do (x, y) nie zawiera pikseli, w przeciwieństwie do kanwy SRGP, która ma piksel w tym punkcie. Na razie będziemy reprezentowali piksele jako rozłączne kółka umieszczone w węzłach siatki; pewne odstępstwa od tego zrobimy przy omawianiu usuwania zakłóceń.



Rys. 3.4. Odcinek po konwersji; wyświetlane piksele są zaznaczone jako czarne kółeczka

Na rysunku 3.4 pokazano silnie powiększony odcinek o grubości 1 piksela i aproksymowany idealny odcinek. Wyświetlane piksele są pokazane jako kółka wypełnione, a nie wybrane piksele jako kółka nie wypełnione. Na rzeczywistym ekranie średnica prawie okrągłego piksela jest większa od odstępu między pikselami i dlatego nasza symboliczna reprezentacja wyolbrzymia dyskretność pikseli.

Ponieważ prymitywy SRGP są definowane na siatce całkowitoliczbowej, współrzędne końców odcinka są liczbami całkowitymi. W rzeczywistości, jeżeli najpierw dokonamy obcięcia odcinka przez prostokąt obcinający, to punkt przecięcia odcinka z brzegiem prostokąta nie musi mieć współrzędnych całkowitych. To samo występuje, gdy korzystamy ze zmiennopozycyjnego pakietu grafiki rastrowej. (Te niecałkowitoliczbowe przecięcia omawiamy w p. 3.2.3). Załóżmy, że nasz odcinek ma nachylenie $|m| \leqslant 1$; rasteryzacja odcinków o innych nachyleniach wymaga wprowadzenia drobnych zmian w poniższych rozważaniach. Najczęściej występujące odcinki – poziome, pionowe i o nachyleniu ± 1 – mogą być potraktowane jako specjalne przypadki, ponieważ przechodzą wyłącznie przez środki pikseli (por. zadanie 3.1).

3.2.1. Podstawowy algorytm przyrostowy

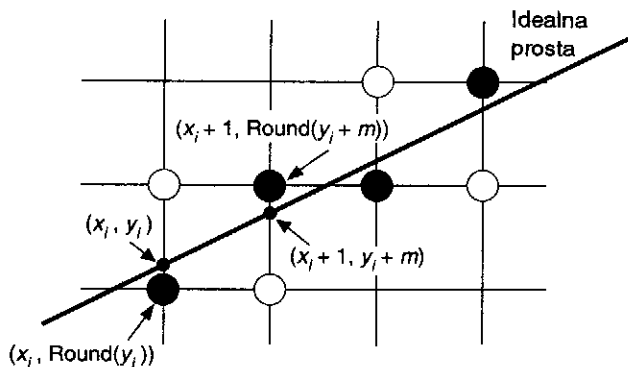
Najprostsza strategia konwersji odcinka polega na obliczeniu nachylenia m jako $\Delta y/\Delta x$, zwiększaniu wartości x o 1 zaczynając od punktu położonego z lewej strony, obliczaniu $y_i = mx_i + B$ dla każdego x_i i wyświetlaniu piksela w punkcie $(x_i, \text{Round}(y_i))$, przy czym $\text{Round}(y_i) = \text{Floor}(0{,}5 = y_i)$. W wyniku tych obliczeń jest wybierany najbliższy piksel, to znaczy ten piksel, który leży najbliżej prawdziwego odcinka. Ta bezpośrednia strategia nie jest efektywna, ponieważ w każdej iteracji jest wykonywane zmiennopozycyjne (albo ułamkowe) mno-

żenie, dodawanie i odwołanie do Floor. Mnożenie można wyeliminować, jeżeli zauważymy, że

$$y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\Delta x$$

i jeżeli $\Delta x = 1$, to $y_{i+1} = y_i + m$.

Wobec tego jednostkowej zmianie x towarzyszy zmiana y o m, które określa nachylenie odcinka. Dla wszystkich punktów (x_i, y_i) na odcinku (nie chodzi tu o punkty otrzymywane w wyniku rasteryzacji odcinka) wiemy, że jeżli $x_{i+1} = x_i + 1$, to $y_{i+1} = y_i + m$; oznacza to, że wartości x i y są zdefiniowane w zależności od poprzednich wartości (rys. 3.5). Na tym właśnie polega algorytm przyrostowy: w każdym kroku wykonujemy obliczenia przyrostowe korzystając z wyników poprzedniego kroku.



Rys. 3.5. Przyrostowe obliczanie (x_i, y_i)

Obliczenia przyrostowe zaczynamy od (x_0, y_0), całkowitych współrzędnych punktu końcowego. Zauważmy, że w tej przyrostowej metodzie nie trzeba bezpośrednio zajmować się współczynnikiem przesunięcia B wzdłuż osi y. Jeżeli $|m| > 1$, to krok w kierunku x tworzy przyrost w kierunku y większy niż 1. Musimy więc zamienić x i y rolami, przypisując krok jednostkowy do y i zwiększając x o $\Delta x = \Delta y/m = 1/m$. Funkcja Line w programie 3.1 realizuje metodę przyrostową. Punktem początkowym musi być lewy koniec odcinka. Przyjęte jest ograniczenie $-1 \leqslant m \leqslant 1$; dla innych nachyleń można postępować symetrycznie. Pominięte jest sprawdzanie przypadków specjalnych, czyli odcinków poziomych, pionowych i przekątnych.

Funkcja WritePixel, wykorzystywana przez Line, jest funkcją niskiego poziomu realizowaną na poziomie programu urządzenia; umieszcza ona w kanwie wartość piksela, którego współrzędne są podane jako

dwa pierwsze argumenty[1]. Zakładamy tutaj, że konwersja jest wykonywana wyłącznie w trybie replace; dla innych trybów zapisu w SRGP musimy korzystać z niskopoziomowej funkcji ReadPixel w celu odczytania piksela w miejscu docelowym, logicznego powiązania tego piksela ze źródłowym pikselem i potem zapisania wyniku do docelowego piksela za pomocą WritePixel.

Ten algorytm jest często określany jako algorytm *DDA* (*digital differential analyzer*). DDA jest mechanicznym urządzeniem, które rozwiązuje równania różniczkowe metodami numerycznymi: są śledzone kolejne wartości (x, y) otrzymywane w wyniku równoczesnego zwiększania x i y o małe przyrosty, proporcjonalne do pierwszych pochodnych po x i y. W naszym przypadku przyrost dla x jest równy 1, a przyrost dla y jest równy $dy/dx = m$. Ponieważ zmienne rzeczywiste mają ograniczoną precyzję, stałe dosumowywanie niedokładnej wartości m wprowadza kumulujący się błąd i możliwe jest odejście od poprawnej wartości Round(y_i); dla większości (krótkich) odcinków nie sprawia to kłopotu.

Program 3.1 Przyrostowy algorytm konwersji odcinka

```
void  Line(int x0, int y0, int x1, int y1, int value)
{                         /* Zakłada się, że –1 ≤ m ≤ 1, x0 < x1 */
  int    x;               /* x zmienia się od x0 do x1 z przyrostem jednostkowym */
  float  dy, dx, y, m;

  dy = y1 – y0;
  dx = x1 – x0;
  m = dy / dx;
  y = y0;
  for (x = x0; x <= x1; x++) {
     WritePixel(x, (int) floor(y + 0.5), value);     /* Ustawienie wartości piksela */
     y += m;                                         /* Zwiększenie y o nachylenie m */
  }
}
```

3.2.2. Rysowanie odcinka – algorytm z punktem środkowym

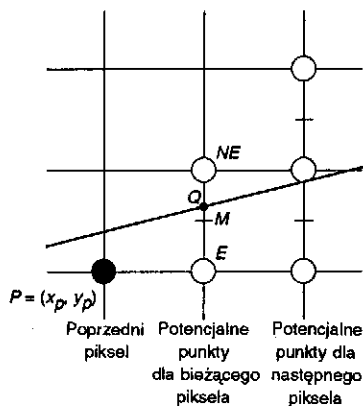
Wadą funkcji Line jest to, że zaokrąglanie y do wartości całkowitej zajmuje czas i że zmienne y i m muszą być rzeczywiste albo ułamkowe, ponieważ nachylenie jest ułamkiem. Bresenham opracował klasyczny algorytm [BRES65], który jest atrakcyjny z tego względu, że korzysta tylko z arytmetyki liczb całkowitych, dzięki czemu unika się funkcji

[1] Jeżeli taka funkcja niskiego poziomu nie jest dostępna, to można użyć funkcji SRGP_pointCoord, tak jak to opisano w podręczniku referencyjnym SRGP.

Round i umożliwia przyrostowe obliczanie wartości (x_{i+1}, y_{i+1}), to znaczy z wykorzystaniem wykonanych już obliczeń dla (x_i, y_i). Zmiennopozycyjna wersja tego algorytmu może być zastosowana do odcinków o dowolnych rzeczywistych wartościach współrzędnych punktów końcowych. Co więcej, przyrostowa metoda Bresenhama może być wykorzystana do całkowitoliczbowych obliczeń dla okręgu, chociaż nie daje się uogólnić łatwo na dowolne krzywe stożkowe. Tutaj korzystamy z nieco odmiennego rozwiązania: *metody punktu środkowego*, opublikowanej po raz pierwszy przez Pittewaya [PITT67] i zaadaptowanej przez Van Akena [VANA84] i innych badaczy. Jak pokazał Van Aken [VANA85], wersja z punktem środkowym dla odcinków i całkowitoliczbowych okręgów sprowadza się do sformułowania Bresenhama i generuje te same piksele. Bresenham pokazał, że jego algorytmy dla odcinka i całkowitoliczbowego okręgu dają najlepszą aproksymację dla idealnych odcinków i okręgów dzięki minimalizacji błędu (odległości) do idealnego prymitywu [BRES77]. Kappel omawia wpływ różnych kryteriów oceny błędów [KAPP85].

Zakładamy, że nachylenie odcinka jest między 0 i 1. W przypadku innych nachyleń można korzystać z odpowiednich odbić względem podstawowych osi. Przyjmujemy, że dolny lewy koniec odcinka ma współrzędne (x_0, y_0), a górny prawy koniec odcinka ma współrzędne (x_1, y_1).

Rozważmy odcinek z rys. 3.6, na którym wcześniej wybrany piksel pojawia się jako czarne kółko, a dwa piksele, spośród których należy dokonać wyboru w następnym kroku, są pokazane jako puste kółka. Załóżmy, że wybraliśmy piksel P o współrzędnych (x_P, y_P), a teraz musimy dokonać wyboru między pikselem znajdującym się o jedną jednostkę w prawo (określanym jako piksel wschodni – E) a pikselem znajdującym się o jedną jednostkę w prawo i o jedną jednostkę do góry



Rys. 3.6. Siatka pikseli dla algorytmu z punktem środkowym; pokazano punkt środkowy M oraz punkty E i NE, spośród których należy wybrać jeden

(określanym jako piksel północno-wschodni – *NE*). Niech *Q* będzie punktem przecięcia odcinka poddawanego konwersji z linią siatki $x = x_p + 1$. W sformułowaniu Bresenhama jest obliczana różnica odległości w pionie między *E* i *Q* oraz *NE* i *Q*. Znak różnicy umożliwia wybranie piksela, którego odległość od *Q* jest mniejsza – zapewnia to najlepszą aproksymację odcinka. W rozwiązaniu z punktem środkowym ustalamy, po której stronie odcinka leży punkt środkowy *M*. Można zauważyć, że jeżeli punkt środkowy leży powyżej odcinka, to piksel *E* leży bliżej odcinka; jeżeli punkt środkowy leży poniżej odcinka, to piksel *NE* jest bliżej odcinka. Odcinek może przechodzić między *E* i *NE* albo oba piksele mogą leżeć po jednej stronie – w każdym z tych przypadków test punktu środkowego wybierze najbliższy punkt. Błąd – to znaczy pionowa odległość między wybranym pikselem a idealnym odcinkiem – wynosi zawsze $\leqslant 1/2$.

W przypadku pokazanym na rys. 3.6 algorytm wybiera *NE* jako następny piksel. Teraz musimy jeszcze pokazać sposób obliczania, po której stronie odcinka leży punkt środkowy. Niech odcinek będzie reprezentowany przez funkcję uwikłaną[2] ze współczynnikami *a*, *b* i *c*: $F(x, y) = ax + by + c = 0$. (Współczynnik *b* przy *y* nie jest związany z wartością przesunięcia *B* w metodzie korzystającej z nachylenia i przesunięcia.) Jeżeli $dy = y_1 - y_0$ i $dx = x_1 - x_0$, to postać z nachyleniem i przesunięciem może być zapisana jako

$$y = \frac{dy}{dx}x + B$$

stąd

$$F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0$$

Zatem w postaci uwikłanej $a = dy$, $b = -dx$ oraz $c = B \cdot dx$[3].

Można łatwo sprawdzić, że funkcja $F(x, y)$ jest równa 0 na odcinku, dodatnia dla punktów poniżej odcinka i ujemna dla punktów powyżej odcinka. Żeby zastosować metodę punktu środkowego, trzeba jedynie obliczyć $F(M) = F(x_P + 1, y_P + 1/2)$ i sprawdzić znak. Ponieważ nasza decyzja zależy od wartości funkcji w punkcie $(x_P + 1, y_P + 1/2)$, definiujemy zmienną decyzyjną $d = F(x_P + 1, y_P + 1/2)$. Z definicji $d = a(x_P + 1) + b(y_P + 1/2) + c$. Jeżeli $d > 0$, to wybieramy *NE*; jeżeli $d < 0$, to wybieramy *E*; jeżeli $d = 0$, to możemy wybrać którykolwiek piksel i wybieramy *E*.

[2] Ta postać funkcjonalna daje się łatwo rozszerzyć na uwikłaną postać dla okręgu.

[3] Ze względu na poprawność działania algorytmu z punktem środkowym ważne jest, żeby *a* było dodatnie; warunek ten będzie spełniony, jeżeli *dy* jest dodatnie, czyli $y_1 > y_0$.

Następnie pytamy, co się stanie z położeniem M, a co za tym idzie z wartością d dla następnej linii siatki; obie wielkości zależą oczywiście od tego, czy wybierzemy E, czy NE. Jeżeli wybierzemy E, to M jest zwiększane o jeden krok w kierunku x. Wtedy

$$d_{\text{new}} = F\left(x_P + 2, y_P + \frac{1}{2}\right) = a(x_P + 2) + b\left(y_P + \frac{1}{2}\right) + c$$

ale

$$d_{\text{old}} = a(x_P + 1) + b\left(y_P + \frac{1}{2}\right) + c$$

Po odjęciu d_{old} od d_{new} otrzymujemy różnicę przyrostową i zapisujemy $d_{\text{new}} = d_{\text{old}} + a$.

Przyrost, który dodajemy po wybraniu E, oznaczamy jako Δ_E; $\Delta_E = a = dy$. Innymi słowy, możemy otrzymać wartość zmiennej decyzyjnej dla następnego kroku dodając przyrost Δ_E do wartości z bieżącego kroku bez konieczności bezpośredniego obliczania $F(M)$.

Jeżeli został wybrany przyrost NE, to M jest zwiększane o jednostkę w obu kierunkach x i y. Wtedy

$$d_{\text{new}} = F\left(x_P + 2, y_P + \frac{3}{2}\right) = a(x_P + 2) + b\left(y_P + \frac{3}{2}\right) + c$$

Po odjęciu d_{old} od d_{new} w celu uzyskania różnicy przyrostowej otrzymujemy

$$d_{\text{new}} = d_{\text{old}} + a + b$$

Przyrost, który trzeba dodać do d po wybraniu NE, oznaczamy jako Δ_{NE}; $\Delta_{NE} = a + b = dy - dx$.

Podsumujmy przyrostową metodę punktu środkowego. W każdym kroku algorytm wybiera jeden z dwóch pikseli na podstawie znaku zmiennej decyzyjnej obliczonej w poprzedniej iteracji; następnie uaktualnia się zmienną decyzyjną przez dodanie albo Δ_E, albo Δ_{NE} do starej wartości zależnie od tego, który piksel został wybrany.

Ponieważ pierwszym wybranym pikselem jest po prostu pierwszy koniec odcinka (x_0, y_0), możemy bezpośrednio obliczyć początkową wartość d, aby wybrać między E i NE. Pierwszy punkt środkowy jest w $(x_0 + 1, y_0 + 1/2)$ i

$$F\left(x_0 + 1, y_0 + \frac{1}{2}\right) = a(x_0 + 1) + b\left(y_0 + \frac{1}{2}\right) + c =$$

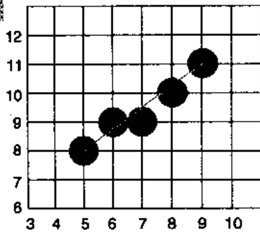
$$= ax_0 + by_0 + c + a + \frac{b}{2} = F(x_0, y_0) + a + \frac{b}{2}$$

Ale (x_0, y_0) jest punktem na odcinku i funkcja $F(x_0, y_0)$ jest równa 0; stąd d_{start} jest równe po prostu $a + b/2 = dy - dx/2$. Korzystając z d_{start} wybieramy drugi piksel itd. W celu wyeliminowania ułamka w d_{start} zmieniamy naszą oryginalną funkcję F mnożąc ją przez 2; $F(x, y) = 2(ax + by + c)$. Powoduje to pomnożenie każdej stałej i zmiennej decyzyjnej (i przyrostów Δ_E i Δ_{NE}) przez 2, nie wpływa natomiast na znak zmiennej decyzyjnej, który liczy się w teście punktu środkowego.

W każdym kroku obliczenia potrzebne do znalezienia d_{new} sprowadzają się do prostego dodawania całkowitoliczbowego. W ogóle nie ma czasochłonnego mnożenia. Wewnętrzna pętla jest bardzo prosta, co można zauważyć w programie 3.2 realizującym algorytm z punktem środkowym. Pierwsze polecenie w pętli, testowanie d, określa wybór piksela, faktycznie natomiast zwiększamy x i y do tego położenia piksela dopiero po uaktualnieniu zmiennej decyzyjnej (dla zapewnienia zgodności z algorytmem dla okręgu). Zauważmy, że ta wersja algorytmu działa tylko dla odcinków o nachyleniu między 0 i 1; uogólnienie algorytmu zostawiamy jako zadanie 3.2. W pracy [SPRO82] Sproull podaje eleganckie wyprowadzenie algorytmu Bresenhama w postaci ciągu przekształceń wychodząc z algorytmu bezpośredniego. Dotychczas nie pojawiło się równoważne wyprowadzenie dla okręgu, ale – jak zobaczymy – metoda punktu środkowego daje się uogólnić.

Program 3.2
Algorytm z punktem środkowym do konwersji odcinka

```
void MidpointLine(int x0, int y0, int x1, int y1, int value)
{
   int  dx, dy, incrE, incrNE, d, x, y;

   dx = x1 – x0;
   dy = y1 – y0;
   d = dy * 2 – dx;                /* Początkowa wartość d */
   incrE = dy * 2;                 /* Przyrost używany dla przejścia do E */
   incrNE = (dy – dx) * 2;            /* Przyrost używany dla przejścia do NE */
   x = x0;
   y = y0;
   WritePixel(x, y, value);           /* Piksel początkowy */
   while (x < x1) {
      if (d <= 0) {                   /* Wybór E */
         d += incrE;
         x++;
      } else {                        /* Wybór NE */
         d += incrNE;
         x++;
         y++;
      }
      WritePixel(x, y, value);        /* Wybrany najbliższy piksel */
   }
}
```



Rys. 3.7. Odcinek między punktami (5, 8) i (9, 11) rysowany metodą punktu środkowego

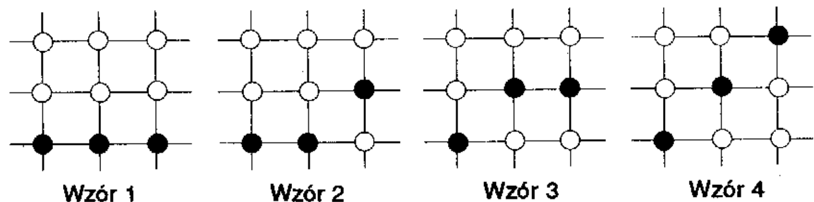
Dla odcinka od punktu (5, 8) do punktu (9, 11) kolejne wartości d są następujące: 2, 0, 6 i 4, co daje w efekcie następujące wybory: *NE*, *E*, *NE* i *NE*, jak to pokazano na rys. 3.7. Odcinek wydaje się nienormalnie zakłócony; jest to skutkiem powiększonej skali rysunku i sztucznie dużych odstępów między pikselami przyjętych dla lepszego zilustrowania algorytmu. Z tego samego powodu na rysunkach w następnych punktach prymitywy wyglądają gorzej niż w rzeczywistym obrazie na ekranie.

Przykład 3.1

Problem: Opracuj i przedstaw w postaci programu ulepszoną wersję algorytmu z punktem środkowym do rysowania odcinka, który zamiast jednego wybiera na raz dwa kolejne punkty.

Odpowiedź

Alternatywą dla algorytmu konwersji odcinka z wykorzystaniem punktu środkowego jest algorytm, który opracowali Wu i Rokne [WU87]. Podobnie jak metoda punktu środkowego jest to metoda przyrostowa; (x_{i+1}, y_{i+1}) można obliczyć znając (x_i, y_i) i korzystając tylko z arytmetyki całkowitoliczbowej. W algorytmie z punktem środkowym najpierw określa się kierunek (nachylenie odcinka), a potem w każdym kroku korzystając ze zmiennej decyzyjnej wybiera się jeden z dwóch możliwych pikseli. W algorytmie z podwójnym krokiem dwukrotnie zredukowana jest liczba decyzji, czyli wyborów następnych pikseli. Uzyskuje się to dzięki poszukiwaniu następnej pary pikseli, a nie jednego piksela. Wu [WU87] pokazał, że mogą wystąpić cztery kombinacje:



W pracy [WU87] pokazano, że wzory 1 i 4 nie mogą wystąpić w tym samym odcinku. Co więcej, jeżeli nachylenie odcinka jest większe niż 1/2, to nie może wystąpić wzór 1. Podobnie, jeżeli nachylenie odcinka jest mniejsze niż 1/2, to nie może wystąpić wzór 4. Dlatego testowanie nachylenia ogranicza wybór do jednego z trzech wzorów: 1, 2, 3 albo 2, 3, 4. Przyjrzyjmy się przypadkowi, kiedy nachylenie jest między 0 i 1/2. To wyklucza wzór 4, tak jak to już wcześniej zaznaczyliśmy. Trzeba zdecydować, czy należy wybrać wzór 1 czy 2 albo 3. Zmienna

decyzyjna początkowo przyjmuje wartość $d = 4dy - dx$. Potem dla każdego przyrostu (krok o dwie jednostki rastra), jeżeli okaże się, że $d < 0$, to należy wybrać wzór 1. Jeżeli d jest większe lub równe 0, to trzeba wybrać między 2 a 3. Decyduje o tym prosty test $d < 2dy$. W celu zwiększania d korzystamy z reguły:

$$d_{i+1} = d_i + 4dy \quad \text{jeżeli } d_i < 0 \text{ (wzór 1)}$$
$$d_{i+1} = d_i + 4dy - 2dx \quad \text{w przeciwnym przypadku (wzór 2 albo 3)}$$

Korzystając z tego ulepszenia można zmodyfikować algorytm z punktem środkowym w następujący sposób:

Kod realizujący ulepszony algorytm z punktem środkowym

```
void DoubleStep( int x0, int y0, int x1, int y1 )
{
  int current_x, incr_1, incr_2, cond, dx, dy, d;

/* Kod wewnętrznej pętli dla przypadku kiedy (0<nachylenie<1/2)

  Dla programu DrawPixels jest potrzebny zarówno wzór, jak i bieżącą pozycja x.
  Wynika to stąd, że w przypadku ostatniego piksela może być konieczne narysowanie
  tylko jednego piksela, a nie całego wzoru. Rysując cały wzór, otrzymalibyśmy zbyt długi
  odcinek */

/* Inicjalizacja wewnętrznej pętli */
   dx = x1 – x0;
   dy = y1 – y0;
   current_x = x0;
   incr_1 = 4*dy;
   incr_2 = 4*dy – 2*dx;
   cond = 2 * dy;
   d = 4*dy – dx;

   while( current_x < x1 ) {          /* potrzebna jest dalsza konwersja */
      if ( d < 0 ) {                  /* pierwsza decyzja */
         DrawPixels(PATTERN_1,current_x);
         d += incr_1;
      } else {
         if ( d < cond )        /* to nie był pierwszy przypadek, wybór między 2 a 3 */
            DrawPixels(PATTERN_2, current_x);
      else
            DrawPixels(PATTERN_3, current_x);
            d += incr_2;
      }
      current_x += 2;
   }
 }
```

Podobnie jak w algorytmie z punktem środkowym, wszystkie obliczenia mogą być wykonywane za pomocą całkowitoliczbowego doda-

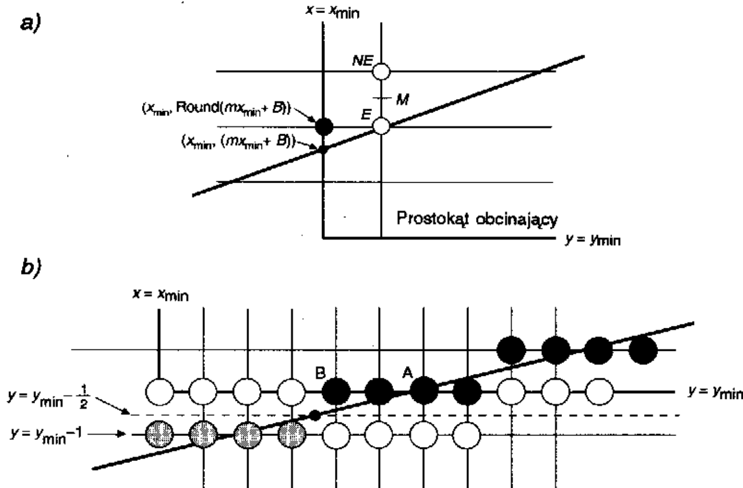
wania i mnożenia przez 2. Jeżeli monitor jest wielopoziomowy, to zamiast rozróżniać między wzorami 2 i 3, można po prostu narysować je oba z jasnością o połowę mniejszą, otrzymując dodatkowo pewną formę odkłócania! Wyvill [WYVI90] zauważył, że można wykorzystać cechę symetrii wokół punktu środkowego i dokonywać konwersji odcinka *równocześnie z obu końców, co dwukrotnie* przyspiesza algorytm. Kompletny algorytm wykorzystujący symetrię i podwójny krok zakodowany w języku C można znaleźć w pracy [WYVI90].

3.2.3. Dodatkowe problemy

Kolejność punktów końcowych. Musimy zapewnić, żeby odcinek rysowany od P_0 do P_1 zawierał ten sam zestaw pikseli co odcinek od P_1 do P_0, tak żeby wygląd odcinka był niezależny od kolejności podawania punktów końcowych. Jedyną sytuacją, kiedy wybór piksela zależy od kierunku odcinka, jest ta, gdy odcinek przechodzi dokładnie przez punkt środkowy i zmienna decyzyjna jest równa zeru. W takim przypadku, idąc z lewa na prawo, wybieramy punkt *E*. Na zasadzie symetrii, idąc z prawa na lewo spodziewalibyśmy się, że dla $d = 0$ będzie wybrany punkt *W*; jednak wtedy zostałby wybrany punkt leżący o jednostkę wyżej w stosunku do tego, który został wybrany przy konwersji z lewa na prawo. Dlatego przy konwersji z prawa na lewo dla $d = 0$ powinniśmy wybrać piksel *SW*. Podobną korektę trzeba zrobić dla odcinków o innych nachyleniach.

Jeżeli korzystamy ze stylów linii, to rozwiązanie z przełączaniem końców odcinka po to, żeby konwersja była dokonywana zawsze w tym samym kierunku, nie działa dobrze. Styl odcinka zawsze zaczepia określoną maskę zapisu w punkcie początkowym, którym powinien być lewy dolny punkt, niezależnie od kierunku odcinka. To nie musi dawać pożądanego efektu wizualnego. Zwłaszcza dla odcinka rysowanego linią przerywaną ze wzorem na przykład 111100 chcielibyśmy, żeby wzór zaczynał się tam, gdzie zostanie określony punkt początkowy, a nie automatycznie w lewym dolnym punkcie odcinka. Jeżeli algorytm zawsze ustawia punkty końcowe w porządku kanonicznym, to wzór mógłby iść z lewa na prawo dla jednego segmentu i z prawa na lewo dla sąsiedniego segmentu, zależnie od nachylenia drugiego odcinka; to mogłoby tworzyć niespodziewane nieciągłości we wspólnym wierzchołku, podczas gdy wzór powinien przechodzić gładko z jednego odcinka do drugiego.

Początek odcinka na krawędzi prostokąta obcinającego. Inny problem polega na tym, że algorytm powinien poprawnie rasteryzować odcinki, które zostały analitycznie obcięte za pomocą jednego z algorytmów



Rys. 3.8. Początek odcinka na krawędzi obszaru obcinającego: a) przecięcie z krawędzią pionową; b) przecięcie z krawędzią poziomą (szare punkty należą do odcinka, ale znajdują się na zewnątrz prostokąta obcinającego)

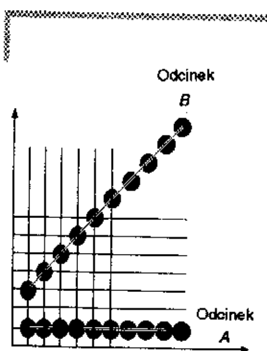
z p. 3.9. Na rysunku 3.8a pokazano odcinek obcięty przez lewą krawędź okna $x = x_{min}$. Punkt przecięcia odcinka z krawędzią ma całkowitą współrzędną x, ale rzeczywistą współrzędną y. Piksel (x_{min}, Round($mx_{min} + B$)), leżący na lewej krawędzi, jest tym pikselem, który byłby narysowany dla tej wartości x przez algorytm przyrostowy, dla nie obciętego odcinka[4]. Dla danego piksela początkowego musimy następnie zainicjalizować zmienną decyzyjną pośrodku między E i NE w następnej kolumnie. Ważne jest, że ta strategia daje poprawną sekwencję pikseli, podczas gdy obcinanie odcinka na krawędzi x, a potem dokonywanie konwersji obciętego odcinka od (x_{min}, Round($mx_{min} + B$)) do (x_1, y_1) za pomocą całkowitoliczbowego algorytmu z punktem środkowym nie – ten obcięty odcinek ma inne nachylenie!

Sytuacja jest bardziej skomplikowana jeżeli odcinek przecina poziomą, a nie pionową krawędź (rys. 3.8b). Dla odcinka o niewielkim nachyleniu w wierszu $y = y_{min}$ będzie kilka pikseli, które odpowiadają dolnej krawędzi prostokąta obcinającego. Chcemy każdy z tych pikseli potraktować jako leżący wewnątrz obszaru obcinającego, ale zwykłe obliczenie analitycznego przecięcia odcinka z linią $y = y_{min}$, a potem zaokrąglenie wartości x w punkcie przecięcia daje piksel A, a nie piksel B leżący z lewej strony ciągu pikseli w wierszu. Z rysunku widać, że piksel B jest tym pikselem, który leży nad i z prawej strony

[4] Jeżeli $mx_{min} + B$ leży dokładnie w połowie między poziomymi liniami siatki, to musimy zaokrąglać w dół. Jest to konsekwencją wyboru piksela E dla $d = 0$.

miejsca w siatce, w którym odcinek po raz pierwszy przecina pionową linię siatki ponad punktem środkowym $y = y_{min} - 1/2$. Dlatego po prostu znajdujemy przecięcie odcinka z linią poziomą $y = y_{min} - 1/2$ i zaokrąglamy wartość x; pierwszy piksel B jest to piksel (Round($x_{min} - 1/2$), y_{min}).

Algorytm przyrostowy z punktem środkowym działa nawet wówczas, gdy punkty końcowe są określone przez pakiet zmiennopozycyjnej grafiki rastrowej; jedyną różnicą jest to, że przyrosty są teraz liczbami rzeczywistymi i obliczenia są wykonywane dla liczb rzeczywistych.



Rys. 3.9. Zmiana jasności odcinka rastrowego w funkcji nachylenia

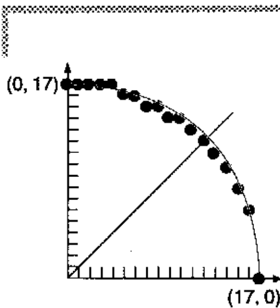
Zmiana jasności odcinka w funkcji nachylenia. Przyjrzyjmy się dwóm odcinkom po konwersji pokazanym na rys. 3.9. Odcinek B leżący na przekątnej siatki ma nachylenie 1 i dlatego jest $\sqrt{2}$ razy dłuższy od poziomego odcinka A, chociaż do narysowania odcinka została wykorzystana taka sama liczba pikseli (10). Jeżeli jasność każdego piksela jest I, to jasność na jednostkę długości dla odcinka A jest I, a dla odcinka B tylko $I/\sqrt{2}$; ta różnica jest łatwo zauważana przez obserwatora. Na monitorze dwupoziomowym nie można tego problemu rozwiązać, natomiast w systemie z n bitami na piksel możemy wprowadzić kompensację, uzależniając jasność od nachylenia odcinka. Przy usuwaniu zakłóceń omawianym w p. 3.14 uzyskuje się jeszcze lepsze wyniki, jeżeli potraktuje się odcinek jako wąski prostokąt i obliczy odpowiednie jasności dla kilku pikseli w kolumnie, leżących wewnątrz lub w pobliżu prostokąta.

Traktowanie odcinka jako prostokąta jest również sposobem na tworzenie grubych odcinków. W punkcie 3.7 pokazujemy, jak zmodyfikować podstawowy algorytm konwersji, żeby można było rysować grube prymitywy oraz prymitywy, których wygląd zależy od stylu linii i stylu pióra.

Prymitywy konturowe budowane z odcinków. Skoro już wiemy, jak dokonywać konwersji odcinków, zastanówmy się, jak dokonywać konwersji prymitywów budowanych z odcinków. W przypadku łamanej można dokonywać konwersji po kolei dla każdego segmentu. Konwersję prostokątów i wielokątów jako prymitywów definiujących obszar można wykonać na zasadzie kolejnego rozpatrywania odcinków, ale może to prowadzić do narysowania kilku pikseli, które leżą poza obszarem prymitywu – w p. 3.4 i 3.5 podano specjalne algorytmy rozwiązywania tego problemu. Trzeba uważać na to, żeby rysować wierzchołki łamanej tylko raz, ponieważ dwukrotne narysowanie wierzchołka powoduje zmianę barwy albo pojawienie się barwy tła, jeżeli jest wykorzystywany tryb zapisu **xor** na ekranie, albo podwojenie jasności w naświetlarce. W rzeczywistości również inne piksele mogą należeć równocześnie do dwóch segmentów, które leżą blisko siebie albo przecinają się. Dyskusję tego problemu oraz różnicy między łamaną a sekwencją połączonych odcinków odkładamy do zadania 3.7.

3.3. Konwersja okręgów

W SRGP nie ma takiego prymitywu jak okrąg. Przy rysowaniu okręgu korzysta się z kołowego łuku eliptycznego jako specjalnego przypadku oraz ośmiokrotnej symetrii. Równanie okręgu o środku w początku układu współrzędnych ma postać $x^2 + y^2 = R^2$. Okrąg, którego środek nie leży w początku układu współrzędnych, może być przesunięty do początku układu współrzędnych o pewną całkowitą wielkość; teraz można dokonać konwersji, przy czym piksele są zapisywane z uwzględnieniem przesunięcia. Jest kilka łatwych, ale nieefektywnych sposobów konwersji okręgu. Po rozwiązaniu ze względu na y uwikłanej postaci równania okręgu, otrzymujemy bezpośrednio $y = f(x)$ jako

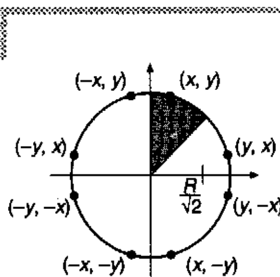


Rys. 3.10. Ćwiartka okręgu generowana z krokiem jednostkowym w kierunku x oraz obliczanym i zaokrąglanym y. Ponieważ dla każdego x jest jedna wartość y, powstają przerwy

$$y = \pm\sqrt{R^2 - x^2}$$

W celu narysowania ćwiartki okręgu (inne ćwiartki są rysowane na zasadzie symetrii) możemy zwiększać x od 0 do R z krokiem jednostkowym i znajdować dodatnie y dla każdego kroku. Takie postępowanie jest poprawne, ale nie jest efektywne ze względu na operacje mnożenia i obliczania pierwiastka. Okrąg będzie miał duże przerwy (chyba że R jest duże) dla wartości x bliskich R, ponieważ nachylenie okręgu staje się tutaj nieskończenie duże (rys. 3.10). Podobnie nieefektywna jest metoda polegająca na rysowaniu ($R\cos\theta$, $R\sin\theta$) dla krokowo zwiększanej wartości θ od 0 do 90° (chociaż unika się tu dużych przerw).

3.3.1. Ośmiokrotna symetria



Rys. 3.11. Osiem punktów symetrycznych na okręgu

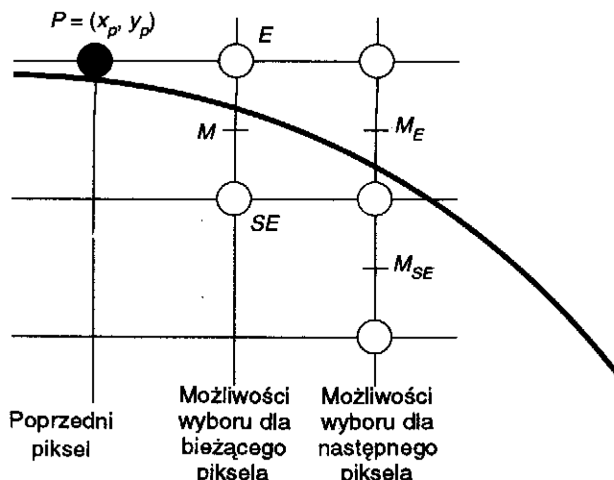
Można ulepszyć proces rysowania okręgu pokazany w poprzednim punkcie lepiej wykorzystując symetrię okręgu. Rozważmy na początek okrąg o środku w początku układu współrzędnych. Jeżeli punkt (x, y) należy do okręgu, to w trywialny sposób możemy obliczyć siedem innych punktów okręgu (rys. 3.11). Dlatego do pełnego określenia okręgu trzeba wykonać obliczenia tylko dla segmentu o kącie 45°. Dla okręgu o środku w początku układu współrzędnych osiem symetrycznych punktów można wyświetlić za pomocą funkcji CirclePoints (funkcję można łatwo uogólnić na przypadek okręgu o środku leżącym w dowolnym punkcie):

```
void CirclePoints (float x, float y, int value);
{
   WritePixel (x, y, value);
   WritePixel (y, x, value);
   WritePixel (y, –x, value);
   WritePixel (x, –y, value);
   WritePixel (–x, –y, value);
   WritePixel (–y, –x, value);
   WritePixel (–y, x, value);
   WritePixel (–x, y, value);
}
```

Dla $x = y$ nie chcemy wywoływać CirclePoint, ponieważ każdy z czterech pikseli byłby narysowany podwójnie; kod można łatwo tak zmodyfikować, żeby uwzględnić warunki brzegowe.

3.3.2. Rysowanie okręgu – algorytm z punktem środkowym

Bresenham [BRES77] opracował przyrostowy generator okręgu, bardziej efektywny niż wyżej omówione metody. Algorytm, opracowany dla współpracy z ploterami, generuje wszystkie punkty okręgu o środku w początku układu współrzędnych metodą przyrostową. Pokażemy podobny algorytm, wykorzystujący ponownie kryterium punktu środkowego, który dla przypadku całkowitoliczbowego środka okręgu i promienia generuje ten sam optymalny zestaw pikseli. Wynikowy kod jest w zasadzie taki sam jak podany w patencie USA 4 371 933 [BRES83].



Rys. 3.12. Siatka pikselowa dla algorytmu z punktem środkowym dla koła; pokazano punkt *M* oraz piksele *E* i *SE*, między którymi trzeba dokonać wyboru

Rozważamy jedynie 45° drugiego oktantu okręgu, od $x = 0$ do $x = y = R/\sqrt{2}$, i używamy funkcji CirclePoints do wyświetlania punktów na całym okręgu. Podobnie jak w przypadku algorytmu z punktem środkowym dla odcinka, strategia polega na wybraniu tego z dwóch pikseli, który jest bliższy okręgu, na zasadzie obliczania funkcji w punkcie środkowym między dwoma pikselami. W drugim oktancie, jeżeli piksel P w (x_P, y_P) został poprzednio wybrany jako bliższy okręgowi, to jako następny piksel może zostać wybrany piksel E lub SE (rys. 3.12).

Niech $F(x, y) = x^2 + y^2 - R^2$; ta funkcja jest równa 0 na okręgu, dodatnia na zewnątrz okręgu i ujemna wewnątrz okręgu. Można zauważyć, że jeżeli punkt środkowy między pikselami E i SE jest na zewnątrz okręgu, to piksel SE jest bliższy okręgu. Jeżeli punkt środkowy jest wewnątrz okręgu, to bliżej okręgu jest piksel E.

Podobnie jak dla odcinków, wyboru dokonujemy na podstawie zmiennej decyzyjnej d, która jest wartością funkcji w punkcie środkowym

$$d_{\text{old}} = F\left(x_P + 1, y_P - \frac{1}{2}\right) = (x_P + 1)^2 + \left(y_P - \frac{1}{2}\right)^2 - R^2$$

Jeżeli $d_{\text{old}} < 0$, to wybieramy E i następny punkt środkowy będzie w odległości jednego przyrostu wzdłuż współrzędnej x. Wtedy

$$d_{\text{new}} = F\left(x_P + 2, y_P - \frac{1}{2}\right) = (x_P + 2)^2 + \left(y_P - \frac{1}{2}\right)^2 - R^2$$

i $d_{\text{new}} = d_{\text{old}} + (2x_P + 3)$; stąd przyrost $\Delta_E = 2x_P + 3$.

Jeżeli $d_{\text{old}} \geqslant 0$, to wybieramy SE[5] i następny punkt będzie w odległości jednego przyrostu wzdłuż współrzędnej x i jednego ujemnego przyrostu wzdłuż współrzędnej y. Wtedy

$$d_{\text{new}} = F\left(x_P + 2, y_P - \frac{3}{2}\right) = (x_P + 2)^2 + \left(y_P - \frac{3}{2}\right)^2 - R^2$$

Ponieważ $d_{\text{new}} = d_{\text{old}} + (2x_P - 2y_P + 5)$, przyrost $\Delta_{SE} = 2x_P - 2y_P + 5$.

Przypomnijmy, że w przypadku liniowym Δ_E i Δ_{NE} były stałe; jednak w przypadku kwadratowym Δ_E i Δ_{SE} zmieniają się w każdym kroku i są funkcjami konkretnych wartości x_P i y_P dla piksela wybranego w poprzedniej iteracji. Ponieważ te funkcje są wyrażone w zależności od (x_P, y_P), P nazywamy *punktem odniesienia*. Funkcje Δ mogą być

[5] Wybór SE dla $d = 0$ różni się od naszego wyboru w algorytmie odcinka i jest arbitralny. Czytelnik może ręcznie zasymulować algorytm i zobaczyć, że dla $R = 17$ przy takim wyborze zmienia się jeden piksel.

obliczane bezpośrednio w każdym kroku przez wstawienie wartości x i y dla wybranego piksela w poprzedniej iteracji. To bezpośrednie obliczenie nie jest specjalnie kosztowne, ponieważ funkcje są tylko liniowe.

W sumie wykonujemy te same dwa kroki w każdej iteracji algorytmu, tak jak to robiliśmy dla odcinka: 1) należy wybrać piksel na bazie znaku zmiennej d obliczonej w czasie poprzedniej iteracji i 2) uaktualnić zmienną decyzyjną d odpowiadającą za wybór piksela o odpowiednią wartość Δ. Jedyna różnica w stosunku do algorytmu dla odcinka polega na tym, że uaktualniając d obliczamy funkcję liniową dla punktu odniesienia.

Teraz pozostaje już tylko obliczyć warunek początkowy. Ograniczając algorytm do przypadku promienia o wartości całkowitej w drugim oktancie, wiemy że piksel początkowy leży na okręgu w punkcie $(0, R)$. Następny punkt środkowy leży w $(1, R-1/2)$ i $F(1, R-1/2) = 1 + (R^2 - R + 1/4) - R^2 = -5/4 - R$. Teraz możemy bezpośrednio zaimplementować algorytm tak jak w programie 3.3. Zauważmy, jak podobna jest struktura tego algorytmu do struktury algorytmu dla odcinka.

Problem z tą wersją polega na tym, że musimy korzystać z arytmetyki zmiennopozycyjnej ze względu na ułamkową inicjalizację d. Chociaż funkcja może być łatwo zmodyfikowana tak, żeby można było rysować okręgi o środkach o współrzędnych, które nie są całkowite albo których promień nie jest całkowity, chcielibyśmy mieć efektywniejszą wersję w pełni całkowitoliczbową. W celu wyeliminowania ułamków dokonamy prostego przekształcenia programu.

Program 3.3
Algorytm konwersji z punktem środkowym dla okręgu

```
void MidpointCircle(int radius, int value)
{
  int   x, y;
  float d;

  x = 0;                    /* Inicjalizacja */
  y = radius;
  d = 5.0 / 4 – radius;
  CirclePoints(x, y, value);
  while (y > x) {
    if (d < 0) {            /* Wybrać E */
      d += x * 2.0 + 3;
      x++;
    } else {                /* Wybrać SE */
      d += (x – y) * 2.0 + 5;
      x++;
      y––;
    }
    CirclePoints(x, y, value);
  }
}
```

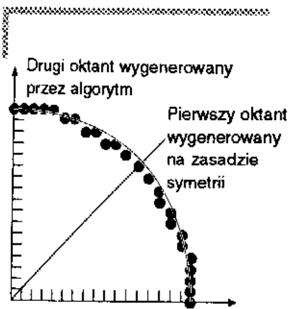
Najpierw definiujemy nową zmienną decyzyjną h jako $h = d - 1/4$ i podstawiamy w kodzie $h + 1/4$ zamiast d. Teraz do inicjalizacji mamy $h = 1 - R$ i porównanie $d < 0$ zostaje zastąpione przez $h < -1/4$. Jednak, ponieważ h zaczyna się od wartości całkowitej i jest zwiększane o wartości całkowite (Δ_E i Δ_{SE}), możemy zmienić porównanie na $h < 0$. Mamy teraz algorytm całkowitoliczbowy względem h; dla zgodności z algorytmem dla odcinka wszędzie podstawimy d zamiast h. Ostatecznie pełny całkowitoliczbowy algorytm jest pokazny w programie 3.4.

Program 3.4 Algorytm z punktem środkowym, całkowitoliczbowy, dla okręgu

```
void  MidpointCircle(int radius, int value)
{
   int   x, y, d;

   x = 0;                        /* Inicjalizacja */
   y = radius;
   d = 1 – radius;
   CirclePoints(x, y, value);
   while (y > x) {
      if (d < 0) {               /* Wybrać E */
         d += x * 2 + 3;
         x++;
      } else {                   /* Wybrać SE */
         d += (x – y) * 2 + 5;
         x++;
         y—;
      }
      CirclePoints(x, y, value);
   }
}
```

Na rysunku 3.13 pokazano drugi oktant okręgu o promieniu 17 generowany za pomocą algorytmu i pierwszy oktant uzyskany na zasadzie symetrii (porównajmy wyniki z rys. 3.10).

Różnice drugiego rzędu. Możemy jeszcze bardziej polepszyć parametry algorytmu z punktem środkowym dla okręgu, korzystając z metody obliczeń przyrostowych. Zauważmy, że funkcje Δ są równaniami liniowymi i obliczaliśmy je bezpośrednio. Jednak każdy wielomian możemy obliczyć metodą przyrostową, tak jak to robiliśmy dla zmiennych decyzyjnych zarówno dla odcinka, jak i dla okręgu. W efekcie obliczamy *różnice pierwszego* i *drugiego rzędu*, korzystając z metody, z którą ponownie spotkamy się w rozdz. 9. Strategia polega na tym, żeby bezpośrednio obliczyć funkcję w dwóch są-



Rys. 3.13. Drugi oktant okręgu wygenerowany za pomocą algorytmu z punktem środkowym i pierwszy oktant wygenerowany na zasadzie symetrii

siednich punktach, obliczyć różnicę (która dla wielomianów jest zawsze wielomianem niższego stopnia) i wykorzystać tę różnicę w każdej iteracji.

Jeżeli w bieżącej iteracji wybierzemy E, to obliczany punkt przesuwa się z (x_P, y_P) do $(x_P + 1, y_P)$. Jak widzieliśmy, dla (x_P, y_P) $\Delta_{Eold} = 2x_P + 3$. Stąd

$$\Delta_{Enew} = 2(x_P + 1) + 3 \text{ dla } (x_P + 1, y_P)$$

a różnica drugiego rzędu jest $\Delta_{Enew} - \Delta_{Eold} = 2$.

Podobnie, dla (x_P, y_P) $\Delta_{SEold} = 2x_P - 2y_P + 5$. Stąd

$$\Delta_{SEnew} = 2(x_P + 1) - 2y_P + 5 \text{ dla } (x_P + 1, y_P)$$

i różnica drugiego rzędu $\Delta_{SEnew} - \Delta_{SEold} = 2$.

Jeżeli w bieżącej iteracji wybierzemy SE, to analizowany punkt przesuwa się z (x_P, y_P) do $(x_P + 1, y_P - 1)$. Stąd

$$\Delta_{Enew} = 2(x_P + 1) + 3 \text{ dla } (x_P + 1, y_P - 1)$$

i różnica drugiego rzędu $\Delta_{Enew} - \Delta_{Eold} = 2$. Tak więc

$$\Delta_{SEnew} = 2(x_P + 1) - 2(y_P - 1) + 5 \text{ dla } (x_P + 1, y_P - 1)$$

i różnica drugiego rzędu wynosi $\Delta_{SEnew} - \Delta_{SEold} = 4$.

Zmodyfikowany algorytm składa się teraz z następujących kroków: 1) wybranie piksela na podstawie znaku zmiennej d obliczonej w czasie poprzedniej iteracji; 2) uaktualnienie zmiennej d albo o Δ_E albo Δ_{SE}, korzystając z odpowiedniej wartości obliczonej w czasie poprzedniej iteracji; 3) uaktualnienie delt z uwzględnieniem przejścia do nowego piksela, korzystając ze stałych różnic obliczonych wcześniej; 4) wykonanie przesunięcia. Δ_E i Δ_{SE} są inicjalizowane z wykorzystaniem punktu startowego $(0, R)$. Zmodyfikowaną funkcję wykorzystującą tę technikę pokazano w programie 3.5.

Program 3.5 Algorytm konwersji okręgu z wykorzystaniem punktu środkowego i różnic drugiego rzędu

```
void MidpointCircle(int radius, int value)
{
   /* W celu obliczenia przyrostów zmiennej decyzyjnej wykorzystuje się różnice drugiego */
   /* rzędu. Zakłada się, że środek okręgu jest w początku układu współrzędnych */
   int    x, y, d, deltaE, deltaSE;
```

```
    x = 0;                  /* Inicjalizacja */
    y = radius;
    d = 1 – radius;
    deltaE = 3;
    deltaSE = 5 – radius * 2;
    CirclePoints(x, y, value);
    while (y > x) {
      if (d < 0) {          /* Wybrać E */
        d += deltaE;
        deltaE += 2;
        deltaSE += 2;
        x++;
      } else {              /* Wybrać SE */
        d += deltaSE;
        deltaE += 2;
        deltaSE += 4;
        x++;
        y—;
      }
      CirclePoints(x, y, value);
    }
}
```

3.4. Wypełnianie prostokątów

Zadanie wypełniania prymitywów może być podzielone na dwie części: podjęcie decyzji o tym, które piksele należy wypełniać (to zależy od kształtu prymitywu, modyfikowalnego przez obcinanie), i podjęcie łatwiejszej decyzji o tym, jak należy te piksele wypełnić. Najpierw omówimy wypełnianie nie obcinanych prymitywów stałą barwą; wypełnianiem wzorem zajmiemy się w p. 3.6. Ogólnie określenie, które piksele trzeba wypełnić, składa się z rozpatrywania kolejnych linii przecinających prymityw i wypełnianiu segmentów sąsiednich pikseli leżących wewnątrz prymitywu, licząc od strony lewej do prawej.

W celu wypełnienia prostokąta stałą barwą każdemu pikselowi segmentu (poziomy odcinek zawarty w prostokącie) przypisujemy tę samą wartość; to znaczy, wypełniamy każdy segment od x_{min} do x_{max}. Te wewnętrzne segmenty charakteryzują się *przestrzenną spójnością*: chodzi o to, że często prymitywy nie zmieniają się od piksela do piksela w ramach segmentu ani między sąsiednimi liniami. Ogólnie, spójność wykorzystujemy w ten sposób, że szukamy tylko tych pikseli, dla których następują zmiany. Dla prymitywów wypełnianych stałą barwą wszystkie piksele należące do tego samego segmentu przyjmują tę samą wartość i mówimy o *spójności segmentowej*. Prostokąty wypełniane stałą barwą mają również silną *spójność międzywierszową*, co oznacza, że ko-

lejne segmenty prostokąta mają tę samą barwę; później będziemy również korzystali z pojęcia *spójności krawędziowej* dla krawędzi dowolnego wielokąta. Będziemy korzystali z różnych typów spójności nie tylko przy konwersji prymitywów 2D, ale również przy renderingu prymitywów 3D, o czym będzie mowa w p. 13.1.

Możliwość traktowania wielu pikseli w ramach segmentu w identyczny sposób jest szczególnie ważna, ponieważ umożliwia zapis do pamięci obrazu całego słowa, co pozwala minimalizować liczbę czasochłonnych dostępów do pamięci; jeżeli segmenty nie są dopasowane do rozkładu słów w pamięci, to algorytm musi zapewnić odpowiednie maskowanie słów nie zawierających pełnego zestawu pikseli. Potrzeba efektywnego zapisywania pamięci jest całkowicie podobna do implementacji copyPixel, co będzie pokrótce omówione w p. 3.13. W naszym kodzie koncentrujemy się na definiowaniu segmentów wewnętrznych i pomijamy problem efektywności zapisywania do pamięci (por. rozdz. 4 i zadanie 3.10).

W związku z powyższym konwersja prostokąta jest to po prostu zagnieżdżona pętla **for**:

```
for (y od ymin do ymaks prostokąta){     /* Dla linii */
   for (x od xmin do xmaks){      /* Dla piksela wewnętrznego */
      WritePixel(x, y, value);
   }
}
```

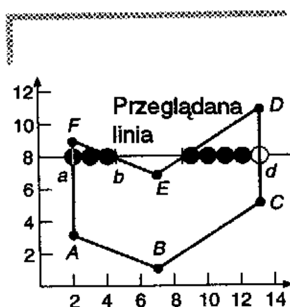
Dla tego bezpośredniego rozwiązania powstaje interesujący problem, podobny do problemu konwersji łamanej, w której segmenty mają wspólne piksele. Rozważmy dwa prostokąty o wspólnej krawędzi. Jeżeli dokonamy konwersji każdego prostokąta osobno, to dla wspólnej krawędzi zapiszemy piksele dwukrotnie, co – jak wcześniej wspominaliśmy – nie jest korzystne. Ten problem jest szczególnym przypadkiem większego problemu definiowania powierzchni prymitywów, a więc określania, które piksele należą do prymitywu, a które nie. Oczywiście te piksele, które należą do matematycznego wnętrza obszaru definiowanego prymitywu, należą do tego prymitywu. Ale co z pikselami leżącymi na brzegu? Jeżeli patrzymy na prostokąt (albo po prostu myślimy o problemie w kategoriach matematycznych), to narzucającą się odpowiedzią jest włączenie do prymitywu pikseli leżących na brzegu; ale jeżeli chcemy uniknąć problemu konwersji wspólnych krawędzi dwukrotnie, to musimy podać jakąś regułę, która będzie jednoznacznie określała, do którego prymitywu należą piksele brzegowe.

Prosta reguła mówi, że piksel brzegowy – to znaczy piksel leżący na krawędzi – nie jest traktowany jako część prymitywu, jeżeli półpłaszczyzna zdefiniowana przez tę krawędź i zawierająca prymityw leży po-

niżej albo z lewej strony krawędzi. Dlatego piksele na krawędziach lewych i dolnych będą rysowane, natomiast piksele, które leżą na górnych i prawych krawędziach, nie będą rysowane. A więc wspólna krawędź należy do tego z dwóch prostokątów, który leży z prawej strony. W efekcie segmenty w prostokącie reprezentują przedziały, które są domknięte z lewej strony i otwarte z prawej strony.

W odniesieniu do tej reguły trzeba podać kilka uwag. Po pierwsze, stosuje się ona do dowolnych wielokątów podobnie jak i do prostokątów. Po drugie, lewy dolny wierzchołek prostokąta i tak będzie rysowany podwójnie – dla tego specjalnego przypadku jest potrzebna inna reguła, o czym powiemy w następnym punkcie. Po trzecie, regułę możemy stosować również do nie wypełnionych prostokątów i wielokątów. Po czwarte, reguła powoduje, że w każdym segmencie będzie gubiony piksel leżący najbardziej z prawej strony i w każdym prostokącie będzie gubiona najwyższa linia. Te problemy pokazują, że nie ma idealnego rozwiązania problemu podwójnego rysowania pikseli na (potencjalnie) wspólnych krawędziach, ale implementatorzy na ogół uważają, że jest lepiej (jeśli idzie o wzrokowy odbiór) dopuszczać brak pikseli na prawej i górnej krawędzi, niż mieć piksele, które znikają lub przyjmują niespodziewane barwy w trybie **xor**.

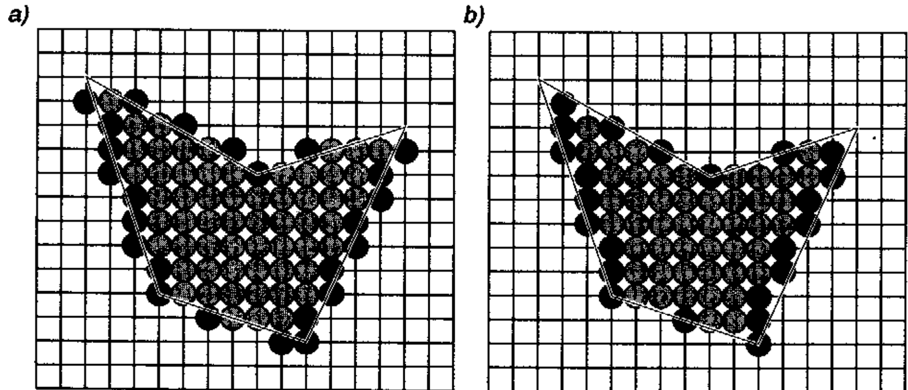
3.5. Wypełnianie wielokątów



Rys. 3.14. Wielokąt i przeglądana linia 8

Opisany dalej ogólny algorytm rasteryzacji wielokąta obejmuje wielokąty zarówno wypukłe, jak i niewypukłe, a nawet samoprzecinające się i z wewnętrznymi dziurami. Algorytm działa na zasadzie wyznaczania segmentów, które leżą między lewymi i prawymi krawędziami wielokąta. Końce segmentów są obliczane za pomocą algorytmu przyrostowego, który wyznacza przecięcie segmentu z krawędzią na podstawie informacji o przecięciu poprzedniego segmentu. Na rysunku 3.14, który ilustruje podstawowy proces rasteryzacji wielokąta, przedstawiono wielokąt i jeden jego segment. Przecięcia linii 8 z krawędziami FA i CD mają współrzędne całkowite, natomiast punkty przecięcia z krawędziami EF i DE nie; punkty przecięć są zaznaczone na rysunku pionowymi kreskami i są oznaczone literami od a do d.

Dla każdego wiersza musimy określić, które piksele leżą wewnątrz wielokąta i ustawić odpowiednie wartości pikseli (w tym przypadku segmenty od $x = 2$ do 4 i od 9 do 13). Powtarzając ten proces dla każdej linii, która przecina wielokąt, wykonamy konwersję całego wielokąta, tak jak pokazano to na przykładzie innego wielokąta na rys. 3.15.



● Punkty ekstremalne segmentów ◍ Inne punkty należące do segmentów

Rys. 3.15. Segmenty w wielokącie. Punkty ekstremalne są zaznaczone jako czarne, a piksele wewnętrzne jako szare: a) punkty ekstremalne wyznaczone za pomocą algorytmu z punktem środkowym; b) punkty ekstremalne leżące wewnątrz wielokąta

Na rysunku 3.15a na czarno są zaznaczone piksele definiujące punkty końcowe wewnętrznych segmentów, a na szaro – punkty wewnętrzne. Bezpośredni sposób wyznaczania punktów końcowych polega na użyciu algorytmu konwersji z punktem środkowym dla każdej krawędzi i utworzeniu tablicy punktów krańcowych dla każdego segmentu w każdej linii, a następnie uaktualnianiu pozycji tablicy, jeżeli zostanie wyznaczony piksel rozszerzający segment. Zauważmy, że przy tej strategii niektóre z generowanych pikseli ograniczających segmenty leżą na zewnątrz wielokąta; są one generowane przez algorytm konwersji odcinka, ponieważ leżą najbliżej krawędzi, niezależnie od tego, po której stronie krawędzi – algorytm nie zwraca uwagi na to, czy punkt jest wewnętrzny czy zewnętrzny. Nie chcemy jednak rysować takich punktów na zewnątrz krawędzi, która jest wspólna dla dwóch wielokątów – ponieważ mogłoby to prowadzić do zakłócania obszarów sąsiednich wielokątów i gdyby te wielokąty miały różne barwy, wynik byłby nienaturalny. Oczywiście lepiej jest rysować tylko te piksele, które należą do wnętrza obszaru, nawet jeżeli zewnętrzny piksel leżałby bliżej krawędzi. Musimy więc odpowiednio zmodyfikować algorytm konwersji; porównajmy rys. 3.15a i 3.15b i zauważmy, że kilka pikseli zewnętrznych w stosunku do idealnego prymitywu nie jest narysowanych na rys. 3.15b.

Przy takim postępowaniu wielokąt nie nakłada się (nawet jednym pikselem) na obszary zdefiniowane przez inne prymitywy. Dla spójności możemy zastosować tę samą metodę do nie wypełnionych wielokątów albo możemy się zdecydować na równoczesną konwersję prostokątów

i wielokątów po jednym segmencie na raz – wtedy nie wypełnione i wypełnione wielokąty nie zawierają tych samych pikseli brzegowych!

Podobnie jak w oryginalnym algorytmie z punktem środkowym, wykorzystujemy algorytm przyrostowy do obliczania końców segmentów w jednej linii na podstawie wartości obliczonych dla poprzedniego segmentu, bez konieczności analitycznego obliczania przecięć kolejnych linii z każdą krawędzią wielokąta. Na przykład dla linii 8 z rys. 3.14 wewnątrz wielokąta są dwa segmenty. Segmenty można wypełniać za pomocą trzyetapowego procesu:

1. Znajdź przecięcia rozpatrywanej linii ze wszystkimi krawędziami wielokąta.
2. Posortuj przecięcia według rosnącej wartości współrzędnej x.
3. Wypełnij między parą przecięć wszystkie piksele, które leżą wewnątrz wielokąta, korzystając przy tym z reguły parzystości do określania, który punkt jest wewnątrz obszaru: na początku przyjmujemy, że odpowiedni bit ma wartość parzystą, a następnie przy każdym przecięciu jego wartość zmienia się na przeciwną – rysujemy te punkty, dla których rozważany bit ma wartość nieparzystą, i nie rysujemy punktów, dla których ten bit ma wartość parzystą.

Pierwsze dwa kroki procesu, znajdowanie przecięć i ich sortowanie, są omówione w następnym punkcie. Przyjrzyjmy się teraz strategii wypełniania. Na rysunku 3.14 posortowana lista współrzędnych x jest następująca (2, 4,5, 8,5, 13). Krok 3 wymaga rozważenia czterech problemów:

3.1. Jak dla przecięcia o dowolnej ułamkowej wartości x określić, który piksel jest wewnętrzny?
3.2. Jak postępować w specjalnym przypadku przecięć dla całkowitych współrzędnych piksela?
3.3. Jak postępować w specjalnym przypadku z kroku 3.2 dla wspólnych wierzchołków?
3.4. Jak postępować w specjalnym przypadku z kroku 3.2, jeżeli wierzchołki definiują poziomą krawędź?

W przypadku 3.1 przyjmujemy, że jeżeli zbliżamy się do przecięcia dla ułamkowej wartości współrzędnej idąc w prawo i jesteśmy wewnątrz wielokąta, to w celu zdefiniowania punktu wewnętrznego zaokrąglamy współrzędną x w dół; jeżeli jesteśmy na zewnątrz wielokąta, to, żeby znaleźć się wewnątrz, zaokrąglamy w górę. Jeśli chodzi o przypadek 3.2, to stosujemy kryterium, z którego korzystaliśmy w celu uniknięcia konfliktów dla wspólnych krawędzi wielokątów: jeżeli pierwszy z lewej piksel segmentu ma współrzędną x całkowitą, to traktujemy go jako wewnętrzny; jeżeli piksel pierwszy z prawej ma współrzędną x całkowitą, to traktujemy go jako zewnętrzny. W przypadku 3.3 przy obliczaniu

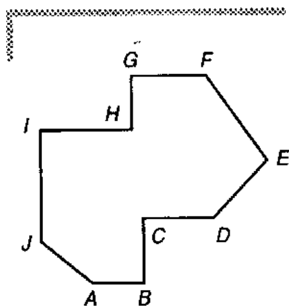
parzystości uwzględniamy wierzchołek y_{min} krawędzi, nie uwzględniamy natomiast wierzchołka y_{max}; dlatego wierzchołek y_{max} jest rysowany tylko wtedy, kiedy jest wierzchołkiem dla sąsiedniej krawędzi. Na przykład wierzchołek A na rys. 3.14 jest w obliczeniach parzystości uwzględniany tylko raz, ponieważ jest to wierzchołek y_{min} dla krawędzi FA i y_{max} dla krawędzi AB. Tak więc zarówno krawędzie, jak i segmenty są traktowane jako przedziały domknięte od strony wartości minimalnej i otwarte od strony maksymalnej. Oczywiście odwrotna reguła również działałaby poprawnie, jednak omawiana reguła wydaje się bardziej naturalna, ponieważ traktuje punkt końcowy minimalny jako punkt wejściowy, a maksymalny jako punkt wyjściowy. W przypadku 3.4 – krawędzi poziomych – pożądany jest taki efekt jak w przypadku prostokątów, a mianowicie, że dolne krawędzie są rysowane, a górne nie. Jak pokażemy w następnym punkcie, dzieje się to automatycznie, jeżeli nie uwzględniamy wierzchołków krawędzi, ponieważ one nie są ani wierzchołkami y_{min}, ani wierzchołkami y_{max}.

Zastosujmy te reguły do linii 8 na rys. 3.14, która nie przecina żadnego wierzchołka. Wypełniamy piksele od punktu a, piksel (2, 8), do pierwszego piksela z lewej strony punktu b, piksel(4, 8) i od pierwszego piksela z prawej strony punktu c, piksel (9, 8), do piksela pierwszego z lewej strony punktu d, piksel (12, 8). Dla linii 3 wierzchołek A jest liczony tylko raz, ponieważ jest to wierzchołek y_{min} dla krawędzi FA, ale także wierzchołek y_{max} dla krawędzi AB; pomocniczy bit przyjmuje wartość nieparzystą, a więc rysujemy odcinek aż do pierwszego piksela z lewej strony przecięcia z krawędzią CB, gdzie pomocniczy bit przyjmuje wartość parzystą i segment się kończy. Linia 1 dotyka tylko wierzchołka B. Obie krawędzie AB i BC mają wierzchołki y_{min} w B, który wobec tego jest liczony dwukrotnie i pomocniczy bit przyjmuje wartość parzystą. Ten wierzchołek jest traktowany jako segment zerowy – wejście w wierzchołku, narysowanie piksela i wyjście w wierzchołku. Chociaż w takim lokalnym minimum jest rysowany jeden piksel, w lokalnym maksimum nie jest rysowany żaden piksel, tak jak na przykład dla wierzchołka F wspólnego dla krawędzi FA i EF. Oba wierzchołki są wierzchołkami typu y_{max} i stąd nie wpływają na parzystość, która się nie zmienia i bit pomocniczy zachowuje wartość parzystą.

3.5.1. Krawędzie poziome

W przypadku krawędzi poziomych postępujemy poprawnie, jeżeli nie uwzględniamy ich wierzchołków; możemy to sprawdzić rozpatrując różne przypadki pokazane na rys. 3.16. Weźmy pod uwagę dolną krawędź AB. Wierzchołek A jest wierzchołkiem typu y_{min} dla krawędzi JA i AB nie

wnosi żadnych zmian. Dlatego pomocniczy bit ma wartość nieparzystą i odcinek *AB* jest rysowany. Pionowa krawędź *BC* ma swój wierzchołek y_{min} w *B* i znowu odcinek *AB* nie jest istotny. Bit pomocniczy przyjmuje wartość parzystą i odcinek poziomy kończy się. W wierzchołku *J* krawędź *IJ* ma wierzchołek typu y_{min}, natomiast krawędź *JA* nie, a więc bit parzystości przyjmuje wartość nieparzystą i jest rysowany segment do krawędzi *BC*. Segment, który zaczyna się na krawędzi *IJ* i trafia na *C*, nie widzi zmiany w *C*, ponieważ jest to wierzchołek typu y_{max} dla *BC* i segment jest prowadzony dalej wzdłuż dolnej krawędzi *CD*. Jednak w *D* krawędź *DE* ma wierzchołek typu y_{min} i bit parzystości jest zerowany na wartość parzystą i segment się kończy. W *I* krawędź *IJ* ma wierzchołek y_{max} i krawędź *HI* nie wnosi nic nowego i bit parzystości w dalszym ciągu ma wartość parzystą i górna krawędź *IH* nie jest rysowana. Jednak w *H* krawędź *GH* ma wierzchołek typu y_{min}, bit parzystości przyjmuje wartość nieparzystą i jest rysowany odcinek od *H* do piksela z lewej strony przecięcia z krawędzią *EF*. W *G* nie ma wierzchołka typu y_{min}, podobnie w *F*; dlatego górna krawędź *FG* nie jest rysowana.

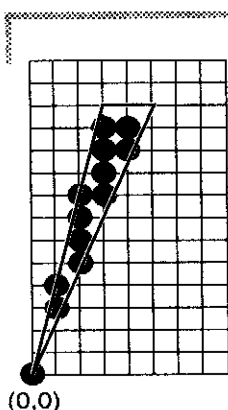


Rys. 3.16. Poziome krawędzie w wielokącie

Omawiany algorytm uwzględnia wspólne wierzchołki w wielokącie, krawędzie wspólne dla dwóch sąsiednich wielokątów i krawędzie poziome. Dopuszcza on samoprzecinające się wielokąty. Jak sygnalizowaliśmy, nie działa on idealnie, ponieważ pomija piksele. Co gorzej, nie potrafi on uniknąć całkowicie wielokrotnego zapisywania wspólnych pikseli bez pamiętania historii: weźmy pod uwagę krawędzie wspólne dla więcej niż dwóch wielokątów albo wierzchołek typu y_{min} wspólny dla dwóch, poza tym rozłącznych trójkątów (por. zadanie 3.11).

3.5.2. Drzazgi

Jest jeszcze jeden problem związany z naszym algorytmem konwersji, który nie daje się rozwiązać tak satysfakcjonująco jak w przypadku poziomych krawędzi: wielokąty o krawędziach, które leżą dostatecznie blisko siebie, tworzą drzazgę – obszar wielokąta tak wąski, że jego wnętrze nie zawiera segmentów w każdej linii. Rozważmy na przykład trójkąt o wierzchołkach (0,0), (3, 12) i (5, 12), pokazany na rys. 3.17. Ze względu na regułę, że są rysowane tylko piksele, które leżą wewnątrz albo na lewej, albo na dolnej krawędzi, będzie wiele linii, w których segmenty będą albo jednopikselowe, albo nie będzie ich wcale. Problem gubienia pikseli jest jeszcze jednym przykładem problemu usuwania zakłóceń, czyli reprezentowania sygnału ciągłego za pomocą aproksymacji dyskretnej. Gdybyśmy mieli wiele bitów na piksel, moglibyśmy użyć metod usuwania zakłóceń, takich jak metoda pokazana w p. 3.14 dla odcinka. Usuwanie zakłóceń mogłoby osłabić naszą regułę rysowania



Rys. 3.17. Konwersja wielokąta typu drzazga

tylko tych pikseli, które leżą wewnątrz albo na lewej, albo na dolnej krawędzi, i dopuścić, żeby brzegowe piksele, a nawet zewnętrzne piksele przyjmowały wartości jasności, które zmieniają się w funkcji odległości między środkiem piksela a prymitywem; wtedy na wartość piksela może mieć wpływ kilka prymitywów.

3.5.3. Spójność krawędziowa a algorytm konwersji

Pierwszy krok naszej procedury – obliczanie przecięć – musi być wykonany szybko. W szczególności musimy unikać metody bezpośredniej testowania przecięcia każdej krawędzi z każdą nową linią. Bardzo często dla danej linii jedynie kilka krawędzi jest interesujących. Możemy również zauważyć, że wiele krawędzi przecinanych przez linię i jest również przecinanych przez linię $i+1$. Ta spójność krawędziowa występuje na krawędzi dla tylu linii, ile przecina krawędź. Gdy przesuwamy się z jednej linii do następnej, możemy obliczyć nową współrzędną x przecięcia z krawędzią, wykorzystując znajomość poprzedniej współrzędnej x przecięcia, podobnie jak to robiliśmy przy obliczaniu następnego piksela na podstawie znajomości bieżącego piksela w algorytmie z punktem środkowym konwersji odcinka, korzystając z zależności

$$x_{i+1} = x_i + \frac{1}{m}$$

przy czym m jest nachyleniem krawędzi. W algorytmie z punktem środkowym konwersji odcinków uniknęliśmy arytmetyki ułamkowej dzięki obliczaniu całkowitoliczbowej zmiennej decyzyjnej i sprawdzaniu tylko jej znaku po to, żeby wybrać piksel najbliższy odcinkowi idealnemu; tutaj chcielibyśmy korzystać z arytmetyki całkowitoliczbowej do wykonywania odpowiedniego zaokrąglania w celu obliczenia najbliższego punktu wewnętrznego.

Rozważmy odcinki o nachyleniu większym niż $+1$, będące lewymi krawędziami; prawe krawędzie i inne nachylenia są rozpatrywane w podobny, choć nieco trikowy sposób, a krawędzie pionowe są traktowane jako specjalne przypadki. (Krawędzie poziome są przetwarzane niejawnie za pomocą reguł dla segmentów, jak to już omawialiśmy.) W punkcie końcowym (x_{min}, y_{min}) musimy narysować piksel. Przy zwiększeniu y, współrzędna x punktu leżącego na idealnym odcinku wzrasta o $1/m$, gdzie $m = (y_{max} - y_{min})/(x_{max} - x_{min})$ jest nachyleniem odcinka. Ten przyrost da w wyniku x mające część całkowitą i ułamkową, co może być wyrażone jako ułamek z mianownikiem $y_{max} - y_{min}$. Przy iteracyjnym powtarzaniu tego procesu, pojawi się nadmiar dla części ułamko-

wej i część całkowita zostanie zwiększona. Na przykład, jeżeli nachylenie wynosi 5/2, a x_{min} wynosi 3, to sekwencja wartości x będzie następująca 3, $3\frac{2}{5}$, $3\frac{4}{5}$, $3\frac{6}{5} = 4\frac{1}{5}$ itd. Jeżeli część ułamkowa x jest równa 0, to możemy narysować piksel (x, y), który leży na odcinku, ale jeżeli część ułamkowa x nie jest zerowa, to musimy zaokrąglić w górę w celu otrzymania piksela, który leży całkowicie wewnątrz odcinka. Gdy część ułamkowa staje się większa niż 1, to zwiększamy x i odejmujemy 1 od części ułamkowej; musimy również przesunąć się o 1 piksel w prawo. Jeżeli po dodaniu przyrostu znajdziemy się dokładnie w pikselu, to rysujemy ten piksel, ale musimy zmniejszyć ułamek o 1 po to, żeby stał się on mniejszy niż 1.

Możemy uniknąć korzystania z ułamków, przechowując tylko licznik ułamka i zauważając, że część ułamkowa jest większa niż 1, gdy licznik jest większy niż mianownik. Wykorzystujemy ten algorytm w programie 3.6, przy czym zmienna *increment* jest wykorzystywana do śledzenia kolejnego zwiększania licznika dopóty, dopóki nie wystąpi nadmiar w stosunku do mianownika, kiedy to następuje zmniejszenie licznika o wartość mianownika i x jest zwiększane.

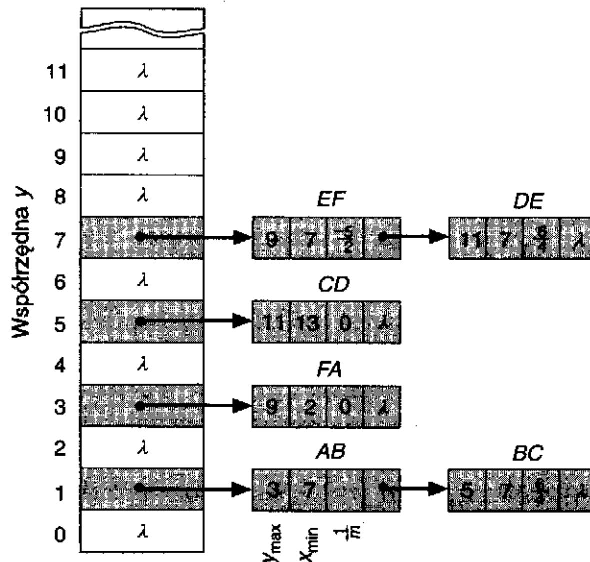
Program 3.6 Konwersja lewej krawędzi wielokąta

```
void LeftEdgeScan(int xmin,int ymin,int xmax,int ymax,int value);
{
  int   x, y, numerator, denominator, increment;

  x = xmin;
  numerator = xmax – xmin;
  denominator = ymax – ymin;
  increment = denominator;

  for (y = ymin; y < ymax; y++) {
    WritePixel(x, y, value);
    increment += numerator;
    if (increment > denominator) {
      /* chwilowo tak ustawiamy prostokąt obcinający, żeby było możliwe zapisywanie
         na cały ekran */
      x += 1;
      increment –= denominator;
    }
  }
}
```

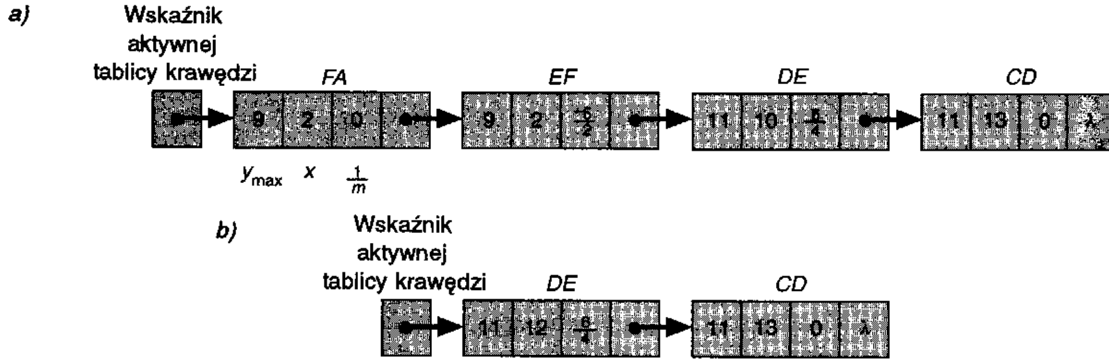
Omówimy teraz *algorytm przeglądania linii*, który wykorzystuje spójność krawędziową i dla każdej linii pamięta zbiór przecinanych krawędzi i punkty przecięć w strukturze danych określanej jako *tablica aktywnych krawędzi* (AET). Krawędzie w tej tablicy są sortowane ze względu

na wartości x przecięć; dzięki temu możemy wypełniać segmenty określone przez pary wartości przecięć (odpowiednio zaokrąglonych) – to znaczy krańcowe wartości segmentów. Gdy przesuwamy się do następnej linii $y + 1$, tablica AET jest uaktualniana. Po pierwsze, krawędzie, które są w tablicy, a nie są przecinane przez następną linię (to znaczy te, dla których $y_{max} = y$), są usuwane. Po drugie, każda nowa krawędź przecinana przez następną linię (to znaczy taka, dla której $y_{min} = y + 1$) jest dodawana do tablicy. Wreszcie, dla krawędzi, które były w tablicy, ale dla których nie zostało zakończone przeglądanie, są obliczane nowe przecięcia za pomocą poprzedniego przyrostowego algorytmu krawędziowego.

W celu efektywnego dodawania krawędzi do tablicy AET, najpierw tworzymy globalną *tablicę krawędzi* (ET), zawierającą wszystkie krawędzie posortowane ze względu na ich najmniejsze współrzędne y. Ta globalna tablica jest na ogół budowana przy wykorzystaniu sortowania grupowego, z tyloma grupami, ile jest linii. W ramach każdej grupy, krawędzie są w porządku wynikającym z rosnącej współrzędnej x dolnego punktu końcowego. Każda pozycja w tablicy globalnej ET zawiera współrzędną y_{max} krawędzi, współrzędną x dolnego punktu końcowego (x_{min}) i przyrost x wykorzystywany przy przejściu od jednej linii do drugiej, czyli $1/m$. Na rysunku 3.18 pokazano, jak przebiega proces sortowania sześciu krawędzi wielokąta z rys. 3.14, a na rys. 3.19 pokazano linie 9 i 10 aktywnej tablicy krawędzi (AET) wielokąta. (W rzeczywistej implementacji prawdopodobnie dodalibyśmy wskaźnik wyróżniający krawędzie lewe i prawe.)



Rys. 3.18. Tablica krawędzi wielokąta z rys. 3.14 posortowana metodą grup



Rys. 3.19. Tablica krawędzi aktywnych dla wielokąta z rys. 3.14: a) linia 9; b) linia 10. (Zauważmy, że współrzędna x krawędzi DE na rysunku b) została zaokrąglona dla tej lewej krawędzi.)

Po utworzeniu globalnej tablicy krawędzi są wykonywane następujące kroki przetwarzania dla algorytmu przeglądania wierszami:

1. Ustaw y na najmniejszej współrzędnej y, która występuje w globalnej tablicy krawędzi, to znaczy y dla pierwszej niepustej grupy.
2. Wyzeruj tablicę aktywnych krawędzi.
3. Powtarzaj dopóty, dopóki tablice aktywna i globalna nie będą puste:
   - 3.1. Przesuń z grupy y tablicy globalnej do tablicy krawędzi aktywnych te krawędzie, dla których $y_{\min} = y$ (wchodzące krawędzie), i posortuj tablicę krawędzi aktywnych ze względu na x (co jest łatwe, ponieważ globalna tablica jest posortowana).
   - 3.2. Wypełnij piksele odpowiednią wartością w przeglądanej linii y, wykorzystując pary współrzędnych x z tablicy aktywnych krawędzi.
   - 3.3. Usuń z tablicy aktywnych krawędzi te pozycje, dla których $y = y_{\max}$ (krawędzie nie mające znaczenia dla następnej linii).
   - 3.4. Zwiększ y o 1 (do wartości współrzędnej dla następnej przeglądanej linii).
   - 3.5. Dla każdej krawędzi, która nie jest pionowa i jest w tablicy aktywnych krawędzi, uaktualnij x dla nowego y.

Ten algorytm wykorzystuje spójność krawędzi przy obliczaniu współrzędnych x przecięć oraz spójność linii (razem z sortowaniem) przy obliczaniu segmentów. Ponieważ sortowaniu jest poddawana nieduża liczba krawędzi i ponowne sortowanie z p. 3.1 ma miejsce w stosunku do listy w znacznym stopniu albo całkowicie posortowanej, można użyć nawet algortmu wstawiania albo bąbelkowego. W rozdziałach 13 i 14 zobaczymy, jak można rozszerzyć ten algorytm, żeby równocześnie zajmować się kilkoma wielokątami w czasie określania powierzchni widocznych, włącznie z przypadkiem wielokątów, które są przezroczyste.

Dla celów konwersji trójkąty i trapezoidy mogą być traktowane jako specjalne przypadki wielokątów, ponieważ mają one tylko dwie krawędzie dla każdej linii (przy założeniu, że krawędzie poziome nie są poddawane jawnej konwersji). Ponieważ dowolny wielokąt może być rozłożony na siatkę trójkątów o wspólnych wierzchołkach i krawędziach (por. zadanie 3.14), moglibyśmy dokonywać konwersji dowolnych wielokątów dekomponując je najpierw na siatki trójkątów, a potem wykonując konwersję trójkątów składowych. Taki podział na trójkąty należy do klasycznych problemów geometrii obliczeniowej [PREP85] i jest łatwy do przeprowadzenia dla wielokątów wypukłych; efektywna realizacja tego w odniesieniu do wielokątów, które nie są wypukłe, jest trudna.

Zauważmy, że gdy bieżąca iteracja algorytmu konwersji w kroku 3.5 generuje kilka pikseli leżących na tej linii, wówczas trzeba odpowiednio uaktualniać wartości ekstremalne segmentów. (Obliczenia związane z krawędziami, które się krzyżują i dla drzazg wymagają specjalnego potraktowania.) Możemy obliczać wszystkie segmenty w jednym przebiegu i wypełniać je w drugim przebiegu albo obliczać segment i wypełniać go od razu po jego znalezieniu. Inną zaletą korzystania z segmentów jest to, że obcinanie może być wykonywane w tym samym czasie co wyznaczanie tych segmentów: segmenty mogą być indywidualnie obcinane przez lewą lub prawą krawędź prostokąta obcinającego.

3.6. Wypełnianie wzorami

W poprzednich punktach wypełnialiśmy wnętrza prymitywów SRGP definiujących obszary jedną barwą, która była podana w polu **value** funkcji WritePixel. Teraz zajmiemy się wypełnianiem wzorem; w tym celu dodajemy dodatkowy parametr do części algorytmu konwersji, która realizuje zapis do każdego piksela. Dla wzorów z mapą pikselową ten parametr powoduje pobranie wartości barwy z odpowiedniej pozycji we wzorze mapy pikselowej, tak jak to pokazujemy dalej. W celu zapisywania wzorów mapy bitowej w sposób przezroczysty, wykonujemy funkcję WritePixel z barwą przedniego planu dla 1 we wzorze, a nie wykonujemy jej dla 0, tak jak przy stylach linii. Jeżeli wzór mapy bitowej jest podawany w trybie nieprzezroczystym, to jedynki i zera wybierają odpowiednio barwę planu przedniego i tła.

3.6.1. Wypełnianie wzorami przy konwersji wierszowej

Głównym problemem przy wypełnianiu wzorami jest relacja między obszarem wzoru a obszarem prymitywu. Innymi słowy, musimy zdecydo-

wać, gdzie wzór ma być zaczepiony, tak żebyśmy wiedzieli, który piksel wzoru odpowiada bieżącemu pikselowi prymitywu.

Pierwsza metoda polega na zaczepieniu wzoru w wierzchołku wielokąta; umieszczamy tam piksel znajdujący się w pierwszym rzędzie wzoru z lewej strony. W tej metodzie wzór może sie poruszać razem z prymitywem, co da efekt wzrokowy, jakiego można by się spodziewać dla wzorów o silnej organizacji geometrycznej, takich jak kreskowanie używane często w rysunkach. Jednak w wielokącie nie ma żadnego punktu, w którym można by wzór w oczywisty sposób zaczepić, nie ma również takich punktów we wszystkich gładko zmieniających się prymitywach takich jak okręgi. Dlatego programista musi określić punkt zaczepienia jako punkt na lub wewnątrz prymitywu. W niektórych systemach punkt zaczepienia może być nawet wspólny dla grupy prymitywów.

Druga metoda, wykorzystywana w SRGP, polega na tym, że traktuje się cały ekran jako obszar wypełniony wzorem, a prymityw jako kontur albo jako obszar wypełniony przezroczystymi bitami, przez które wzór jest widoczny. Standardowym punktem zaczepienia jest początek układu współrzędnych. Piksele prymitywu są wtedy traktowane jako jedynki i są iloczynowane ze wzorem. Ubocznym efektem tej metody jest to, że wzór nie jest związany z prymitywem wówczas, gdy jest on przesuwany. Prymityw porusza się tak jak wykrój nad tłem o pewnym wzorze i wobec tego jego wygląd może się zmieniać w trakcie poruszania się; dla regularnych wzorów bez silnej geometrycznej orientacji użytkownicy mogą nawet tego nie zauważyć. Metoda bezwzględnego zaczepiania jest nie tylko efektywna obliczeniowo, ale umożliwia również nakładanie się prymitywów i stykanie bez widocznych miejsc styku.

W celu pokrycia prymitywu wzorem indeksujemy wzór za pomocą współrzędnych piksela (x, y). Ponieważ wzory są definiowane jako małe mapy bitowe albo pikselowe o rozmiarze $M \times N$, w celu powtarzania wzoru korzystamy z arytmetyki modulo. Przyjmujemy, że piksel *pattern*[0, 0] jest związany z początkiem ekranu[6] i wzór mapy bitowej w trybie przezroczystym możemy zapisać na przykład w następujący sposób:

```
if ( pattern [x % M, y % N] )WritePixel(x, y, value)
```

Jeżeli wypełniamy cały segment w trybie **replace**, to możemy skopiować od razu cały wiersz wzoru, zakładając, że jest dostępna niskopoziomowa wersja funkcji copyPixel umożliwiająca zapisywanie wielu pikseli. Załóżmy na przykład, że mamy wzór w postaci tablicy 8×8.

[6] W systemach okienkowych wzór jest często zaczepiony w początku układu współrzędnych okna.

Wtedy powtarza się on w wierszu co każde 8 pikseli. Jeżeli lewy punkt segmentu jest wyrównany z początkiem bajtu – to znaczy, jeżeli wartość x pierwszego piksela mod 8 jest równa 0 – to cały pierwszy rząd wzoru może być pobrany za pomocą copyPixel z tablicy 1 na 8; ta procedura jest powtarzana dopóty, dopóki segment nie zostanie wypełniony. Jeżeli któryś z końców segmentu nie jest wyrównany z granicą bajtu, to piksele, które nie należą do odcinka, muszą zostać zamaskowane. Implementatorzy spędzają wiele czasu nad opracowaniem szczególnie efektywnych algorytmów rastrowych dla przypadków specjalnych; na przykład dążą do wyeliminowania pętli wewnętrznych i piszą ręcznie optymalizowane kody asemblerowe dla pętli wewnętrznych, które wykorzystują zalety specjalnych funkcji sprzętowych, takich jak pamięć podręczna albo szczególnie efektywne instrukcje pętli.

3.6.2. Wypełnianie wzorami bez wielokrotnej konwersji wierszowej

Dotychczas omawialiśmy wypełnianie w kontekście konwersji wierszowej. Inna metoda polega na tym, że najpierw dokonuje się konwersji prymitywu do prostokątnego obszaru roboczego, a potem zapisuje każdy piksel z tej mapy bitowej na właściwe miejsce w kanwie. Ten tak zwany *prostokątny zapis* do kanwy to po prostu zagnieżdżona pętla **for**, w której dla 1 jest zapisywana bieżąca barwa, a dla 0 nic nie jest zapisywane (dla przezroczystości) albo jest zapisywana barwa tła (dla nieprzezroczystości). Ten dwustopniowy proces jest prawie dwa razy tak pracochłonny jak wypełnianie z konwersją i dlatego nie nadaje się do prymitywów, które występują i są poddawane konwersji tylko raz. Opłaca się jednak dla prymitywów, które w przeciwnym przypadku byłyby poddawane konwersji wiele razy. Tak jest w przypadku znaków danego kształtu i wielkości, które mogą być poddane konwersji zawczasu na podstawie ich konturów. W stosunku do znaków zdefiniowanych tylko w postaci map bitowych albo innych obiektów, takich jak ikony i logo, które są malowane albo skanowane jako obrazy w postaci map bitowych, konwersja w ogóle nie jest używana i prostokątny zapis ich map bitowych jest jedyną możliwą metodą wypełniania. Zaletą mapy bitowej uzyskanej we wcześniejszym procesie konwersji jest to, że oczywiście szybciej można zapisać każdy piksel w prostokątnym obszarze, bez konieczności wykonywania jakiegokolwiek obcinania czy obliczeń związanych z wyznaczaniem segmentów niż wykonywanie konwersji prymitywu za każdym razem od początku przy wykonywaniu obcinania.

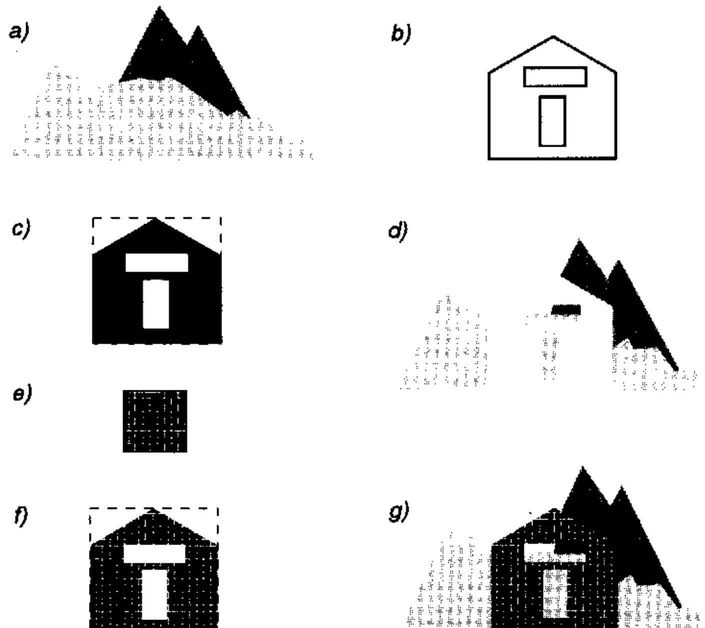
Jeżeli musimy zapisać prostokątną mapę bitową do kanwy, to czemu nie wykonać operacji copyPiksel bezpośrednio w stosunku do mapy bitowej, zamiast zapisywać 1 piksel za każdym razem? Dla monitorów

dwupoziomowych przy zapisie bieżącej barwy 1 copyPixel działa dobrze: dla trybu przezroczystego używamy trybu zapisu **or**; dla trybu nieprzezroczystego używamy trybu zapisu **replace**. Dla monitorów wielopoziomowych nie możemy zapisać mapy bitowej bezpośrednio za pomocą 1 bitu na piksel, musimy natomiast zamienić każdy bit na n-bitową wartość barwy, która jest potem zapisywana.

W niektórych systemach silniejsza funkcja copyPixel umożliwia wykonywanie operacji zapisu z uwzględnieniem masek albo dla informacji źródłowej, albo dla informacji zapisywanej. Można to dobrze wykorzystać w trybie przezroczystym (używanym dla znaków w SRGP), jeżeli możemy określić mapę bitową jako maskę dla zapisu i źródło jako tablicę o stałej (bieżącej) barwie. Wtedy piksele są zapisywane z bieżącą barwą tylko tam, gdzie mapa bitowa maski zapisu ma jedynki; mapa bitowa maski zapisu działa jak dowolny obszar obcinania. W pewnym sensie zagnieżdżona pętla **for** dla implementacji zapisu prostokątnego w systemie z n bitami na piksel symuluje tę silniejszą możliwość wykonywania *copyPixel z maską zapisu.*

Rozważmy teraz inny wariant. Chcemy narysować literę albo jakiś inny kształt wypełnione wzorem, a nie w sposób ciągły. Na przykład chcielibyśmy utworzyć grubą literę „P" wypełnioną 50-procentowym wzorem szarych kropek (nadających literze szarą barwę) albo ikonę domu ze wzorem dwubarwnej cegły i zaprawy. Jak możemy zapisać taki obiekt w trybie nieprzezroczystym bez wykonywania konwersji za każdym razem? Problem polega na tym, że wnętrza dziur w obszarze, gdzie są zera w mapie bitowej, powinny być zapisane barwą tła, podczas gdy dziury na zewnątrz obszaru (takie jak wnęka w literze „P") muszą być zapisane w sposób przezroczysty, tak żeby nie zmienić znajdującego się pod spodem obrazu. Innymi słowy, chcemy, żeby zera we wnętrzu kształtu oznaczały barwę tła, a zera na zewnątrz niego, w tym wszelkie wnęki, należały do maski zapisu używanej do ochrony pikseli na zewnątrz kształtu. Jeżeli dokonujemy konwersji z przeglądaniem wierszy na bieżąco, to nie powstaje problem związany z tym, że zera oznaczają różne rzeczy w różnych obszarach mapy bitowej, ponieważ nigdy nie patrzymy na piksele leżące na zewnątrz granicy kształtu.

W celu uniknięcia ciągłej konwersji wierszowej korzystamy z czteroetapowego procesu, tak jak to pokazano na przykładzie gór na rys. 3.20a. Korzystając z konturu ikony pokazanej na rys. 3.20b, w pierwszym kroku tworzymy pełną mapę bitową, która będzie używana jako maska zapisu/obszaru obcinania, z jedynkami dla pikseli znajdujących się wewnątrz obiektu i z zerami na zewnątrz; jest to pokazane na rys. 3.20c, na którym biały obszar reprezentuje piksele tła (0), a czarny reprezentuje jedynki. Ta konwersja wierszowa jest wykonywana tylko raz. W drugim kroku, za każdym razem gdy jest potrzebna kopia



Rys. 3.20. Zapisywanie w trybie nieprzezroczystym obiektu pokrytego wzorem z dwoma zapisami przezroczystymi: a) scena gór; b) kontur ikony domu; c) mapa bitowa pełnej wersji ikony domu; d) wyzerowanie sceny na skutek zapisania tła; e) wzór ceglasty; f) ikona domu pokryta wzorem ceglastym; g) zapisywanie na ekran w sposób przezroczysty z ikoną domu pokrytą wzorem

obiektu wypełnionego wzorem, zapisujemy w kanwach tę pełną mapę bitową w sposób przezroczysty z barwą tła (w trybie **replace**). W ten sposób ustawiamy barwę tła w obszarze kształtu obiektu, tak jak to pokazano na rys. 3.20d, na którym na tle istniejącego obrazu gór ustawiono białe tło odpowiadające kształtowi domu. Trzeci krok polega na utworzeniu wersji pełnej mapy bitowej obiektu ze wzorem; w tym celu wykonujemy operację copyPiksel prostokąta wzoru (rys. 3.20e) do pełnej mapy bitowej korzystając z trybu **and**. To powoduje zmianę niektórych pikseli leżących wewnątrz kształtu obiektu z 1 na 0 (rys. 3.20f) *i można to traktować jako wycięcie kawałka dowolnie dużego wzoru* w kształcie obiektu. Wreszcie ponownie zapisujemy tę nową mapę bitową w sposób przezroczysty na to samo miejsce w kanwie, ale tym razem z bieżącą barwą pierwszego planu, tak jak to pokazano na rys. 3.20g. Tak jak przy pierwszym zapisie do kanwy wszystkie piksele na zewnątrz obszaru obiektu są zerami po to, żeby zabezpieczyć zewnętrzne piksele obszaru; natomiast zera wewnątrz obszaru nie wpływają na poprzednie zapisane (białe) tło; plan przedni (czarny) jest zapisywany tylko tam, gdzie są jedynki. W celu zapisania domu ze wzorem ceglastym i z zaprawą powinniśmy zapisać mapę bitową barwą

szarą i mapę bitową ze wzorem barwą czerwoną; wzór powinien mieć jedynki wszędzie poza małymi paskami zer reprezentującymi zaprawę. W rezultecie zredukowaliśmy funkcję zapisu prostokątnego, która miała zapisać dwubarwny obiekt do maski zapisu, do dwóch funkcji zapisu, które zapisują w sposób przezroczysty, albo do funkcji copyPiksel z maską zapisu.

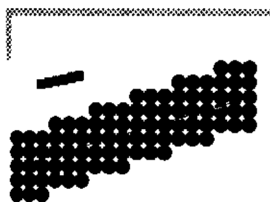
3.7. Pogrubione prymitywy

Pogrubione prymitywy tworzymy na zasadzie śledzenia jednopikselowego konturu prymitywu uzyskanego w wyniku konwersji. Umieszczamy środek pędzla o określonym przekroju (albo inny wyróżniony punkt, np. lewy górny róg prostokątnego pędzla) w każdym pikselu wybranym w trakcie realizacji algorytmu konwersji. Każda linia o szerokości 1 piksela może być traktowana jako narysowana pędzlem o szerokości 1 piksela. Jednak ten prosty opis ukrywa kilka problemów. Po pierwsze, jaki kształt ma pędzel? W typowych implementacjach wykorzystuje się pędzle okrągłe i prostokątne. Po drugie, jaka jest orientacja pędzla, który nie jest okrągły? Czy prostokątne pióro zawsze jest ustawione równolegle do krawędzi ekranu, dzięki czemu pióro ma stałą szerokość, czy też obraca się w trakcie poruszania się po konturze prymitywu, tak że pionowa oś pędzla jest styczna do prymitywu? Jak wyglądają końce grubego odcinka, zarówno idealnego jak i na siatce całkowitoliczbowej? Co dzieje się w wierzchołku pogrubionego wielokąta? Jak współdziałają ze sobą styl linii i styl pióra? W tym punkcie odpowiemy na prostsze z tych pytań. Inne są omawiane w rozdz. 19 pracy [FOLE90].

Cztery podstawowe metody rysowania pogrubionych prymitywów pokazano na rys. 3.21 do 3.24. Idealne linie prymitywów są narysowane w postaci czarnych konturów z białym środkiem, a piksele generowane w czasie konwersji dla linii o grubości jednego piksela są zaznaczone na czarno; piksele dodane w celu uzyskania efektu pogrubienia są szare. Wersje w zmniejszonej skali pokazują, jak wygląda pogrubiony prymityw dla zbyt małej rozdzielczości, ze wszystkmi pikselami ustawionymi na czarno. W pierwszej metodzie, stanowiącej grube przybliżenie, w czasie konwersji wykorzystuje się więcej niż 1 piksel w każdej kolumnie (albo wierszu). W drugiej metodzie rysuje się wzdłuż jednopikselowego konturu prymitywu piórem o jakimś przekroju. W trzeciej metodzie rysuje się dwie kopie prymitywu w odległości t od siebie i wypełnia się obszar między zewnętrzną a wewnętrzną linią. Czwarta metoda aproksymuje wszystkie prymitywy łamanymi i wykorzystuje pogrubione odcinki dla każdego segmentu łamanej.

Przyjrzyjmy się każdej z tych metod i rozważmy ich zalety i wady. Wszystkie metody dają efekty, które są satysfakcjonujące dla wielu, jeżeli nie dla większości przypadków, przynajmniej dla oglądania na ekranie. Do drukowania powinno się korzystać z większej rozdzielczości, zwłaszcza że wtedy szybkość algorytmu nie jest tak krytyczna jak przy generowaniu prymitywów na bieżąco. Możemy wtedy korzystać z bardziej złożonych algorytmów w celu uzyskania lepszych wizualnie efektów. Pakiet może wręcz korzystać z różnych metod dla różnych prymitywów. Na przykład, QuickDraw do rysowania odcinków wykorzystuje prostokątne pióro (równoległe do osi ekranu), a metodę wypełniania segmentów dla obszaru między wewnętrzną a zewnętrzną elipsą.

3.7.1. Powielanie pikseli

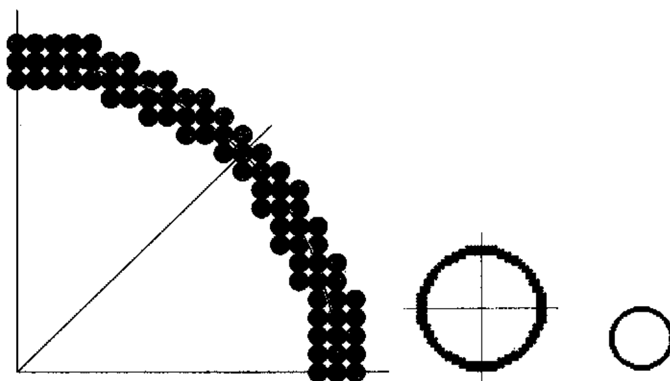


Rys. 3.21. Pogrubiony odcinek narysowany metodą powielania kolumn

Rozszerzenie wewnętrznej pętli w programie konwersji tak, żeby dla każdego obliczonego piksela rysować wiele pikseli, daje dobre rezultaty w przypadku odcinków; dla odcinków o nachyleniu od -1 do 1 piksele są powielane w kolumnach, a dla pozostałych nachyleń w wierszach. Skutek jednak jest taki, że końce odcinków są zawsze poziome albo pionowe, co nie wygląda ładnie dla grubych odcinków (rys. 3.21).

Ten algorytm powielania pikseli tworzy również zauważalne przerwy w miejscach, gdzie odcinki przecinają się, i gubi piksele w miejscach, gdzie następuje przejście od powielania poziomego do pionowego w związku ze zmianą nachylenia. Ten ostatni efekt uwidocznia się jako nienormalne zwężenie łuku elipsy na granicy między oktantami (rys. 3.22).

Grubość odcinków poziomego i pionowego różni się od grubości odcinka rysowanego pod pewnym kątem, dla którego grubość prymitywu jest zdefiniowana jako odległość między brzegami prymitywu



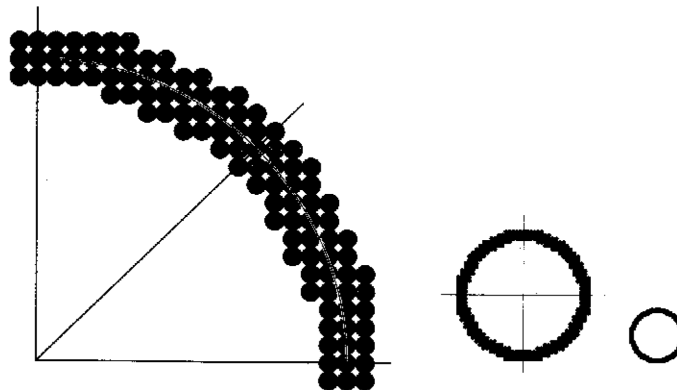
Rys. 3.22. Pogrubiony okrąg narysowany metodą powielania kolumn

liczona wzdłuż prostopadłej do stycznej. Jeżeli parametrem oznaczającym grubość jest t, to odcinki poziomy i pionowy mają grubość t, a odcinek narysowany pod kątem 45° ma średnią grubość $t/\sqrt{2}$. Ta różnica to inny skutek tego, że odcinek rysowany pod pewnym kątem ma mniej pikseli, na co po raz pierwszy zwróciliśmy uwagę w p. 3.2.3; zmniejszeniu ulega kontrast jasności w stosunku do odcinków poziomego i pionowego o tej samej grubości. Jeszcze innym problemem związanym z powielaniem pikseli jest szerokość o parzystej wartości; z powielanym pikselem nie możemy związać środka powielanej kolumny albo wiersza i musimy znaleźć tę stronę prymitywu, która ma mieć dodatkowy piksel. W sumie powielanie piksela jest efektywną, ale zgrubną aproksymacją, która jest dobra dla niezbyt grubych prymitywów.

3.7.2. Ruchome pióro

Rys. 3.23. Pogrubiony odcinek narysowany za pomocą prostokątnego pióra

Prostokątne pióro, którego środek albo róg porusza się wzdłuż jednopikselowego konturu prymitywu, działa dobrze dla odcinków; otrzymuje się odcinek pokazany na rys. 3.23. Zauważmy, że ten odcinek jest podobny do tego, który został otrzymany metodą powielania piksela, ale jest grubszy na końcach. Podobnie jak w przypadku powielania piksela, ze względu na to, że pióro jest zawsze ustawione pionowo, obserwowana grubość prymitywu zmienia się w funkcji kąta prymitywu, ale w przeciwnym kierunku: szerokość jest mniejsza dla odcinków poziomych i większa dla odcinków o nachyleniu ± 1. Na przykład łuk elipsy zmienia grubość wzdłuż całej trajektorii i ma określoną szerokość wówczas, gdy styczna jest prawie pozioma albo pionowa i powiększona $\sqrt{2}$ razy w okolicach kątów $\pm 45°$ (por. rys. 3.24). Ten problem mógłby



Rys. 3.24. Pogrubiony okrąg narysowany za pomocą prostokątnego pióra

zostać wyeliminowany, gdyby prostokąt mógł się obracać wzdłuż konturu, lepiej jest jednak skorzystać z przekroju kołowego, tak żeby szerokość była niezależna od kąta.

Zobaczmy teraz, jak można zrealizować algorytm poruszającego się pióra o przekroju prostokąta o bokach równoległych do boków ekranu albo o przekroju kołowym. Najprostsze rozwiązanie polega na użyciu copyPixel dla wybranego przekroju (określanego również jako *ślad*) o stałej barwie albo wypełnionego wzorem, tak żeby jego środek albo róg były w wybranym pikselu; dla kołowego śladu i wzoru rysowanego w trybie nieprzezroczystym musimy dodatkowo maskować bity poza obszarem kołowym, co nie jest łatwe, jeżeli funkcja niskiego poziomu copyPixel nie ma maski zapisu dla obszaru przeznaczenia. Przy najprostszym rozwiązaniu copyPixel zapisuje piksele więcej niż raz, ponieważ ślad pióra nakłada się na sąsiednie piksele. Lepsza metoda, która również rozwiązuje problem kołowego przekroju, polega na tym, żeby wykorzystywać segmenty śladu do obliczania segmentów następnych śladów w sąsiednich pikselach. Podobnie jak w przypadku wypełniania prymitywów, takie kombinowanie segmentów w linii rastra, to w zasadzie nic innego jak łączenie segmentów wymagające śledzenia minimalnych i maksymalnych wartości x akumulowanych segmentów dla każdej linii rastra.

Inne metody wyświetlania pogrubionych prymitywów, np. wypełnianie powierzchni między zewnętrznym i wewnętrznym konturami znajdującymi się w odległości $t/2$ po każdej stronie jednopikselowej trajektorii, są omawiane w rozdz. 3 i 19 pracy [FOLE90].

3.8. Obcinanie w technice rastrowej

Jak zauważyliśmy we wstępie do tego rozdziału, ważne jest, żeby zarówno obcinanie, jak i konwersja były wykonywane tak szybko, jak to tylko jest możliwe, po to, żeby użytkownik mógł jak najszybciej zobaczyć efekty wynikające ze zmian w modelu zastosowania. Obcinanie może być wykonane analitycznie, w trakcie konwersji, albo jako część operacji copyPixel z odpowiednim prostokątem obcinającym przy przesyłaniu z kanwy zawierającej nie obcięte prymitywy do kanwy docelowej. Ta trzecia metoda może być użyteczna wówczas, gdy najpierw można wygenerować dużą kanwę, a później użytkownik może sprawdzać jej fragmenty przesuwając prostokąt obcinający bez uaktualniania zawartości kanwy.

Połączenie konwersji i obcinania (określane jako *wycinanie*) można łatwo zrealizować w stosunku do wypełnionych albo pogrubionych prymitywów: obcięte muszą być tylko wartości krańcowe; natomiast nie muszą być sprawdzane żadne punkty wewnętrzne. Wycinanie pokazuje

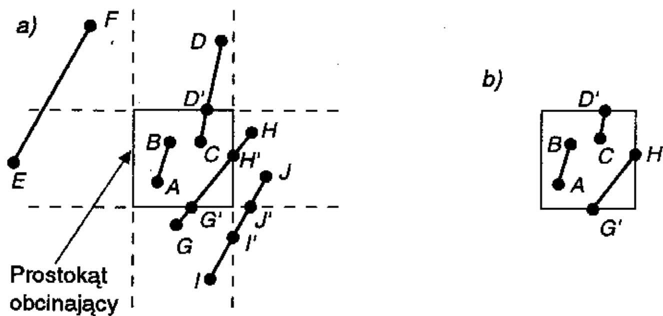
jeszcze jedną zaletę spójności wewnątrzsegmentowej. Jeżeli prymityw nie jest dużo większy od prostokąta obcinającego, to względnie niewiele pikseli znajdzie się poza obszarem obcinania. W takim przypadku może się okazać, że szybciej jest generować każdy piksel i obcinać go (to znaczy zapisywać warunkowo), niż wykonywać wcześniej obcinanie analityczne. W szczególności, chociaż odpowiedni test jest w pętli wewnętrznej, unika się czasochłonnych zapisów dla zewnętrznych pikseli i zarówno obliczenia przyrostowe, jak i testowanie mogą być wykonywane całkowicie w szybkiej pamięci, takiej jak pamięć podręczna rozkazów CPU albo pamięć mikrorozkazów kontrolera monitora.

Mogą być użyteczne również inne triki. Na przykład, można zatrzymać się na przecięciu odcinka z krawędzią obszaru obcinającego, wykonując standardowy algorytm konwersji z punktem środkowym dla co i-tego piksela i sprawdzając, czy wybrany piksel leży w granicach prostokąta, dopóki nie zostanie znaleziony pierwszy piksel, który leży wewnątrz tego obszaru. Wtedy algorytm musi wrócić, znaleźć pierwszy wewnętrzny piksel i wykonać normalną konwersję. W podobny sposób można określić ostatni piksel wewnętrzny. Można też sprawdzać każdy piksel w trakcie wykonywania wewnętrznej pętli konwersji i konwersja mogłaby być zatrzymywana, gdy test da wynik negatywny. Testowanie co ósmego piksela daje zadowalające wyniki, ponieważ jest to dobry kompromis między zbyt dużą liczbą testów a zbyt dużą liczbą pikseli, o które trzeba się cofać.

Dla pakietów graficznych, które wykorzystują zmienny przecinek, lepiej jest obcinać analitycznie w systemie współrzędnych zmiennopozycyjnych, a potem dokonywać konwersji obciętych prymitywów, uważając na poprawne inicjalizowanie zmiennych decyzyjnych, tak jak to zrobiliśmy dla odcinków w p. 3.2.3. W całkowitoliczbowych pakietach graficznych, takich jak SRGP, można wybierać między wstępnym obcinaniem a późniejszą konwersją albo wykonywaniem obcinania w czasie konwersji. Ponieważ jest dość łatwo wykonywać obcinanie analityczne dla odcinków i wielokątów, obcinanie tych prymitywów jest często wykonywane przed konwersją, podczas gdy obcinanie innych prymitywów wykonuje się szybciej w czasie konwersji. W zmiennopozycyjnych pakietach graficznych często wykonuje się obcinanie analityczne w ich układach współrzędnych, a potem wywołuje się program konwersji niskiego poziomu, który generuje obcięte prymitywy; ten całkowitoliczbowy program graficzny mógłby później wykonywać dodatkowo obcinanie rastrowe w granicach prostokątnego (a nawet dowolnego) okna. Ponieważ analityczne obcinanie prymitywów jest użyteczne dla całkowitoliczbowych pakietów graficznych i istotne dla zmiennopozycyjnych pakietów grafiki 2D i 3D, w tym rozdziale omówimy podstawowe algorytmy obcinania analitycznego.

3.9. Obcinanie odcinków

W tym punkcie omówimy metody analitycznego obcinania odcinków przez prostokąty; algorytmy obcinania innych prymitywów są omawiane w kolejnych punktach. Chociaż istnieją specjalne algorytmy obcinania prostokątów i wielokątów, musimy zauważyć, że prymitywy SRGP są budowane z odcinków (łamane, nie wypełnione prostokąty i wielokąty) i mogą być obcinane na zasadzie wielokrotnego stosowania obcinania odcinka. Co więcej, okręgi mogą być aproksymowane za pomocą ciągu bardzo krótkich odcinków i obwody mogą być traktowane jako łamane albo wielokąty zarówno z punktu widzenia konwersji, jak i obcinania. W niektórych systemach krzywe stożkowe są reprezentowane jako ilorazy wielomianów parametrycznych (rozdz. 9) – reprezentacji, która nadaje się do przyrostowej aproksymacji odcinkami dla algorytmu obcinania. Obcinanie prostokąta przez prostokąt daje najwyżej jeden prostokąt. Obcinanie wypukłego wielokąta przez prostokąt daje najwyżej jeden wypukły wielokąt, ale obcinanie niewypukłego wielokąta może dać w efekcie więcej niż jeden niewypukły wielokąt. Obcinanie okręgu przez prostokąt może dać nawet cztery łuki.



Rys. 3.25. Różne przypadki obcinania odcinków

Odcinki obcinane przez prostokątny obszar obcinający dają zawsze jeden segment; odcinki, które leżą na brzegu prostokąta obcinającego, są traktowane jako leżące wewnątrz i są wyświetlane. Na rysunku 3.25 pokazano kilka przykładów obciętych odcinków.

3.9.1. Obcinanie punktów

Zanim omówimy obcinanie odcinków, przyjrzyjmy się prostszemu problemowi obcinania punktów. Jeżeli współrzędnymi x prostokąta obci-

nającego są x_{min} i x_{max}, a granicznymi współrzędnymi y są y_{min} i y_{max}, to na to, żeby punkty (x, y) były wewnątrz prostokąta obcinającego, muszą być spełnione cztery nierówności:

$$x_{min} \leqslant x \leqslant x_{max}, \qquad y_{min} \leqslant y \leqslant y_{max}$$

Jeżeli jedna z tych czterech nierówności nie jest spełniona, to punkt leży na zewnątrz prostokąta obcinającego.

3.9.2. Obcinanie odcinków na zasadzie rozwiązywania układu równań

W celu obcięcia odcinka musimy wziąć pod uwagę tylko jego punkty końcowe, a nie nieskończenie wiele punktów wewnętrznych. Jeżeli oba punkty końcowe odcinka leżą wewnątrz prostokąta obcinającego (na przykład *AB* na rys. 3.25), to cały odcinek leży wewnątrz prostokąta obcinającego i może być bezpośrednio zaakceptowany. Jeżeli jeden koniec odcinka leży wewnątrz prostokąta obcinającego, a drugi na zewnątrz (na przykład *CD* na rys. 3.25), to odcinek przecina prostokąt obcinający i musimy obliczyć punkt przecięcia. Jeżeli oba końce odcinka leżą na zewnątrz prostokąta obcinającego, to odcinek może (ale nie musi) przecinać prostokąt obcinający (*EF*, *GH* i *IJ*) i musimy przeprowadzić dalsze obliczenia w celu ustalenia, czy są inne przecięcia, a jeżeli są, to gdzie się znajdują.

Narzucające się rozwiązanie zadania obcinania odcinka, trudne jednak do zaakceptowania, polega na przecięciu tego odcinka z każdą z czterech krawędzi prostokąta obcinającego po to, żeby sprawdzić, czy któryś z punktów przecięcia nie leży na tych krawędziach; jeżeli tak, to odcinek przecina prostokąt obcinający i częściowo leży wewnątrz tego prostokąta. Dla każdego odcinka i krawędzi prostokąta obcinającego bierzemy więc dwie matematycznie nieskończone linie, które je zawierają, i znajdujemy ich przecięcie. Następnie sprawdzamy, czy ten punkt przecięcia jest punktem wewnętrznym – to znaczy, czy leży zarówno na krawędzi prostokąta obcinającego, jak i na odcinku; jeżeli tak, to jest przecięcie z prostokątem obcinającym. Na rysunku 3.25 punkty G' i H' są punktami wewnętrznymi, a I' i J' nie. Korzystając z takiego podejścia musimy rozwiązać układ równań i wykonać mnożenie i dzielenie dla każdej pary ⟨krawędź, odcinek⟩. Chociaż można by skorzystać z równania prostej ze współczynnikiem nachylenia i przesunięciem, znanego z geometrii analitycznej, dotyczy ono jednak nieskończenie długiej linii prostej, podczas gdy w grafice i przy obcinaniu mamy do czynienia z odcinkami. Ponadto to równanie prostej nie obejmuje przypadku linii pionowych, co jest poważnym problemem, jeśli prostokąt obcinający

ma boki równoległe do boków ekranu. Żaden z tych problemów nie występuje w reprezentacji parametrycznej:

$$x = x_0 + t(x_1 - x_0), \qquad y = y_0 + t(y_1 - y_0)$$

Te równania opisują położenie punktu (x, y) na odcinku od (x_0, y_0) do (x_1, y_1) dla parametru t w przedziale [0, 1], co łatwo jest sprawdzić wstawiając odpowiednie wartości t do równań. Układ równań zapisanych w postaci parametrycznej może być rozwiązany ze względu na parametry t_{edge} dla krawędzi i t_{line} dla odcinka. Potem można sprawdzić, czy wartości t_{edge} i t_{line} należą do przedziału [0, 1]; jeżeli tak, to punkt przecięcia należy do obu odcinków i jest wynikiem obcinania przez prostokąt. Przed rozwiązaniem układu równań trzeba sprawdzić, czy nie występuje specjalny przypadek linii równoległej do krawędzi prostokąta obcinającego. W sumie bezpośrednie podejście wiąże się z dużą liczbą obliczeń oraz sprawdzeń i dlatego jest nieefektywne.

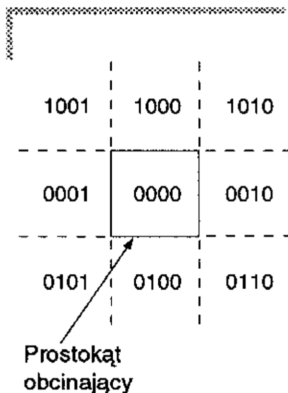
3.9.3. Algorytm Cohena-Sutherlanda obcinania odcinków

Efektywniejszy algorytm Cohena-Sutherlanda wykonuje testy początkowe dla odcinka w celu określenia, czy nie można uniknąć obliczania przecięć. Najpierw sprawdza się, czy końce odcinków mogą być bezpośrednio zaakceptowane. Jeżeli odcinka nie można bezpośrednio zaakceptować, to wykonuje się sprawdzanie obszarów. Na przykład, dwa proste porównania współrzędnych x pokazują, że oba punkty końcowe odcinka EF na rys. 3.25 mają współrzędne x mniejsze od x_{min} i wobec tego leżą w obszarze na lewo od prostokąta obcinającego (to znaczy w zewnętrznej półpłaszczyźnie zdefiniowanej przez prostokąt obcinający); wobec tego odcinek EF można bezpośrednio odrzucić i nie musi on być ani obcinany, ani wyświetlany. Podobnie możemy bezpośrednio odrzucić odcinki o obu końcach w obszarze na prawo od x_{max}, poniżej y_{min} i powyżej y_{max}.

Jeżeli odcinek nie może być bezpośrednio ani zaakceptowany, ani odrzucony, to jest on dzielony na dwie części przez krawędź obcinającą, tak, żeby jedna część mogła być bezpośrednio odrzucona. Tak więc odcinek jest iteracyjnie obcinany i sprawdza się, czy może być już bezpośrednio odrzucony albo zaakceptowany; jeżeli żaden test nie jest pozytywny, to odcinek jest dzielony dopóty, dopóki to, co pozostanie, nie będzie się zawierało całkowicie w prostokącie obcinającym albo będzie mogło być bezpośrednio odrzucone. Algorytm jest szczególnie efektywny dla dwóch przypadków. W pierwszym przypadku, gdy prostokąt obcinający jest duży i obejmuje całość lub większość pola wyświetlania,

większość prymitywów może być bezpośrednio zaakceptowana. W drugim przypadku, gdy prostokąt obcinający jest mały, prawie wszystkie odcinki można bezpośrednio odrzucić. Ten ostatni przypadek występuje w standardowej metodzie korelacji wskazywania, w której mały prostokąt otaczający kursor, określany jako *okno wskazywania*, jest używany do obcinania prymitywów po to, żeby określić, które prymitywy leżą wewnątrz małego (prostokątnego) sąsiedztwa punktu wskazywanego przez kursor (por. p. 7.11.2).

W celu wykonywania testów akceptacji-odrzucenia przedłużamy krawędzie prostokąta obcinającego tak, żeby podzieliły płaszczyznę, na której leży prostokąt obcinający, na dziewięć obszarów (rys. 3.26). Każdemu obszarowi jest przypisany 4-bitowy kod, wynikający z położenia obszaru w stosunku do półpłaszczyzn zewnętrznych dla krawędzi prostokąta obcinającego. Każdy bit w kodzie przyjmuje wartość 1 (prawda) albo 0 (fałsz); 4 bity kodu spełniają następujące warunki:



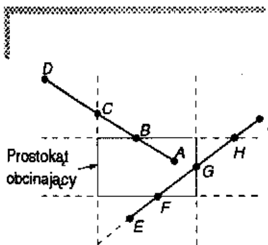
Rys. 3.26. Kody obszarów

Pierwszy bit	półpłaszczyzna powyżej górnej krawędzi $y > y_{max}$
Drugi bit	półpłaszczyzna poniżej dolnej krawędzi $y < y_{min}$
Trzeci bit	półpłaszczyzna na prawo od prawej krawędzi $x > x_{max}$
Czwarty bit	półpłaszczyzna na lewo od lewej krawędzi $x < x_{min}$

Ponieważ obszar powyżej i z lewej strony prostokąta obcinającego leży w zewnętrznej półpłaszczyźnie dla krawędzi górnej i lewej, przypisujemy mu kod 1001. Szczególnie efektywna metoda obliczania kodu wynika ze spostrzeżenia, że bit 1 jest bitem znaku $(y_{max} - y)$; bit 2 jest bitem znaku dla $(y - y_{min})$; bit 3 jest bitem znaku dla $(x_{max} - x)$; bit 4 jest bitem znaku dla $(x - x_{min})$. Każdemu punktowi końcowemu odcinka jest przypisywany kod obszaru, do którego ten koniec należy. Teraz możemy wykorzystać kody punktów końcowych do określenia, czy odcinek leży całkowicie w prostokącie obcinającym albo w zewnętrznej półpłaszczyźnie krawędzi. Jeżeli oba 4-bitowe kody końców odcinka są zerowe, to odcinek leży całkowicie wewnątrz prostokąta. Jeżeli jednak oba końce leżą w zewnętrznej półpłaszczyźnie określonej krawędzi (na przykład *EF* na rys. 3.25), to każdy z kodów końców odcinka ma ustawiony bit wskazujący, że punkt leży w zewnętrznej półpłaszczyźnie krawędzi. Dla *EF* odpowiednie kody są następujące: 0001 i 1001 i ich czwarte bity pokazują, że odcinek leży w zewnętrznej półpłaszczyźnie lewej krawędzi. Dlatego, jeżeli logiczny iloczyn (**and**) kodów punktów końcowych nie jest równy zeru, to odcinek można bezpośrednio odrzucić.

Jeżeli odcinek nie może być bezpośrednio odrzucony albo zaakceptowany, to musimy podzielić go na dwa odcinki w taki sposób, żeby

jeden z nich albo oba mogły być odrzucone. Tego podziału dokonujemy wykorzystując krawędź, którą odcinek przecina: odcinek leżący w zewnętrznej półpłaszczyźnie krawędzi jest odrzucany. Możemy wybrać dowolny porządek testowania krawędzi, ale w algorytmie za każdym razem musimy korzystać zawsze z tego samego porządku; będziemy korzystali z następującego porządku: góra-dół, prawa strona-lewa strona. Istotną cechą kodu jest to, że bity, które mają wartość różną od zera, odpowiadają przecinanym krawędziom; jeżeli jeden punkt końcowy leży w zewnętrznej półpłaszczyźnie krawędzi i odcinek nie spełnia bezpośrednich testów odrzucenia, to drugi punkt musi leżeć w wewnętrznej półpłaszczyźnie tej krawędzi i odcinek musi ją przecinać. Dlatego algorytm zawsze wybiera punkt, który leży na zewnątrz i potem wykorzystuje ustawiony bit kodu do określenia krawędzi obcinającej; krawędź jest wybierana zgodnie z porządkiem góra-dół i prawo-lewo, to znaczy w kolejności określonej przez pierwszy z lewej strony ustawiony bit w kodzie. Algorytm działa następująco. Obliczamy kody dla obu końców i sprawdzamy, czy odcinek można bezpośrednio odrzucić albo zaakceptować. Jeżeli żaden z testów nie jest udany, znajdujemy ten koniec, który leży na zewnątrz (przynajmniej jeden musi leżeć na zewnątrz) i potem na podstawie kodu znajdujemy przecinaną krawędź i określamy odpowiedni punkt przecięcia. W punkcie przecięcia odcinamy odcinek od strony końca zewnętrznego, zastępując koniec zewnętrzny punktem przecięcia i obliczamy kod tego nowego punktu końcowego w celu przygotowania się do następnej iteracji.



Rys. 3.27. Ilustracja metody Cohena-Sutherlanda obcinania odcinka

Rozważmy odcinek *AD* z rys. 3.27. Punkt *A* ma kod 0000, a punkt *D* ma kod 1001. Odcinek nie może być bezpośrednio zaakceptowany ani odrzucony. Algorytm wybiera punkt *D* jako punkt zewnętrzny, kod punktu *D* wskazuje, że odcinek przecina krawędzie górną i lewą. Zgodnie z naszym porządkiem testowania najpierw obcinamy odcinek *AD* przez krawędź górną i otrzymujemy odcinek *AB*, a następnie wyznaczamy kod punktu *B* jako 0000. W następnej iteracji do odcinka *AB* stosujemy bezpośredni test akceptacji/odrzucenia i jest on bezpośrednio zaakceptowany i wyświetlony.

Odcinek *EI* wymaga wielu iteracji. Pierwszy punkt końcowy *E* ma kod 0100; algorytm wybiera go jako punkt zewnętrzny i sprawdzając kod stwierdza, że pierwszą krawędzią obcinającą odcinek jest dolna krawędź; *EI* zostaje obcięty do *FI*. W drugiej iteracji odcinek *FI* nie może być bezpośrednio zaakceptowany albo odrzucony. Pierwszy punkt końcowy *F* ma kod 0000 i algorytm wybiera punkt zewnętrzny *I* o kodzie 1010. Odcinek jest obcinany przez górną krawędź i otrzymujemy *FH*. Punkt *H* otrzymuje kod 0010; w trzeciej iteracji ma miejsce obcięcie przez prawą krawędź do odcinka *FG*. Ten odcinek jest akceptowany w czwartej i końcowej iteracji i zostaje wyświetlony. Inna

sekwencja obcinania występowałaby, gdybyśmy jako początkowy wybrali punkt *I*. Korzystając z kodu tego punktu najpierw obcinalibyśmy względem górnej krawędzi, potem prawej i na końcu dolnej.

W kodzie programu 3.7 korzystamy ze struktury języka C z polami do reprezentowania kodu końca odcinka; taka reprezentacja jest bardziej naturalna od tablicy czterech liczb całkowitych. Ze względu na modularność do obliczania końca odcinka wykorzystujemy funkcję zewnętrzną; aby poprawić efektywność powinniśmy oczywiście wstawić ten kod do programu.

Program 3.7 Algorytm Cohena-Sutherlanda obcinania odcinka

```
typedef struct {
   unsigned all;
   unsigned left:1;
   unsigned right:1;
   unsigned bottom:1;
   unsigned top:1;
} outcode;

void  CohenSutherlandLineClipAndDraw(float x0, float y0, float x1, float y1,
           float xmin, float xmax, float ymin, float ymax, int value)
/* Algorytm Cohena-Sutherlanda obcinania odcinka od P0=(x0, y0) do P1=(x1, y1)
   i obcinania prostokąta z przekątną od (xmin, ymin) do (xmax, ymax) */
{
   boolean  accept, done;
   outcode outcode0, outcode1, outcodeOut, CompOutCode();
   float  x, y;

   accept = FALSE;
   done = TRUE;
   outcode0 = CompOutCode(x0, y0, xmin, xmax, ymin, ymax);
   outcode1 = CompOutCode(x1, y1, xmin, xmax, ymin, ymax);
   do {
if (outcode0.all == 0 && outcode1.all == 0) {
   accept = TRUE;
   done = TRUE;
} else if (outcode0.all & outcode1.all != 0)
   done = TRUE:   /* Podział logiczny jest true więc bezpośrednie odrzucenie
                     i wyjście */
else {
   if (outcode0.all != 0)
      outcodeOut = outcode0;
   else
      outcodeOut = outcode1;
   if (outcodeOut.top) {   /* Podział odcinka na górze prostokąta obcinającego */
      x = x0 + (x1 – x0) * (ymax – y0) / (y1 – y0);
      y = ymax;
   } else if (outcodeOut.bottom) { /* Podział odcinka na dole prostokąta obcinającego */
```

```
                x = x0 + (x1 – x0) * (ymin – y0) / (y1 – y0);
                y = ymin;
            } else if (outcodeOut.right) { /* Podział odcinka przez prawą krawędź
                                              prostokąta obcinającego */
                y = y0 + (y1 – y0) * (xmax – x0) / (x1 – x0);
                x = xmax;
            } else if (outcodeOut.left) {     /* Podział odcinka przez lewą krawędź */
                y = y0 + (y1 – y0) * (xmin – x0) / (x1 – x0);
                x = xmin;
            }
            if (outcodeOut.all == outcode0.all) {
                x0 = x;
                y0 = y;
                outcode0 = CompOutCode(x0, y0, xmin, xmax, ymin, ymax);
            } else {
                x1 = x;
                y1 = y;
                outcode1 = CompOutCode(x1, y1, xmin, xmax, ymin, ymax);
            }
        }                         /* Podział */
    } while (!done);
    if (accept)
        MidpointLineReal(x0, y0, x1, y1, value) /* Wersja dla współrzędnych
                                                   zmiennopozycyjnych */
}

outcode CompOutCode (float x, float y,
        float xmin, float xmax, float ymin, float ymax)
{
    outcode code;
    code.all = 0;
    if (y > ymax) {
        code.top = 1;
        code.all += code.top;
    } else if (y < ymin) {
        code.bottom = 1;
        code.all += code.bottom;
    }
    if (x > xmax) {
        code.right = 1;
        code.all += code.right;
    } else if (x < xmin) {
        code.left = 1;
        code.all += code.left;
    }
    return code;
}
```

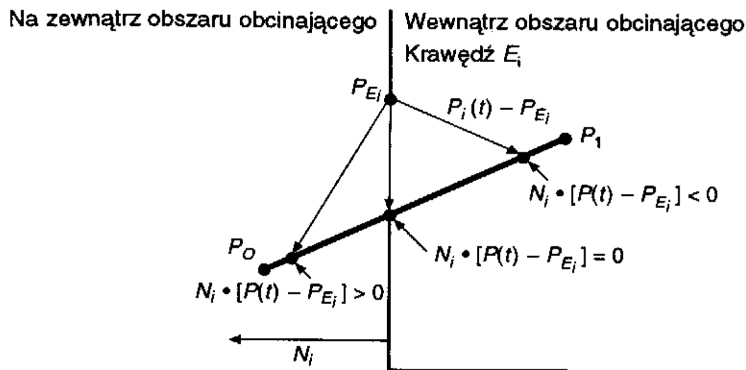
Możemy poprawić trochę efektywność algorytmu, jeżeli nie będziemy ponownie obliczali nachyleń (por. zadanie 3.22). Jednak nawet z tym ulepszeniem nie jest to najefektywniejszy algorytm. Ponieważ testowanie i obcinanie są wykonywane w ustalonym porządku, algorytm czasami będzie wykonywał niepotrzebne obcinanie. Takie obcinanie pojawia się, gdy przecięcie z krawędzią prostokąta jest zewnętrznym przecięciem, tzn. jeżeli nie leży na granicy prostokąta obcinającego (np. punkt *H* na odcinku *EI* na rys. 3.27). Dla kontrastu algorytm Nicholla, Lee i Nicholla [NICH87] unika obliczania zewnętrznych przecięć dzięki podziałowi płaszczyzny na większą liczbę obszarów; jest on omawiany w rozdz. 19 pracy [FOLE90]. Zaletą znacznie prostszego algorytmu Cohena-Sutherlanda jest to, że może on być bezpośrednio rozszerzony dla prostokątnej bryły widzenia 3D, jak to zobaczymy w p. 6.6.3.

3.9.4. Parametryczny algorytm obcinania odcinków

Algorytm Cohena-Sutherlanda jest prawdopodobnie wciąż najczęściej używanym algorytmem obcinania odcinka, ponieważ jest stosowany najdłużej i był szeroko publikowany. W 1978 r. Cyrus i Beck opublikowali algorytm, który wykorzystuje całkowicie inne i efektywniejsze podejście do obcinania odcinka [CYRU78]. Metoda Cyrusa-Becka może być używana do obcinania odcinka 2D przez prostokąt albo wielokąt wypukły na płaszczyźnie albo odcinka 3D przez dowolny wypukły wielościan w przestrzeni 3D. Później Liang i Barsky niezależnie opracowali efektywniejszy algorytm obcinania odcinka parametrycznego; algorytm ten jest wyjątkowo szybki w szczególnym przypadku obcinania przez obszary 2D i 3D o bokach równoległych do osi układu współrzędnych [LIAN84]. Niezależnie od wykorzystania zalet wynikających z tak prostych granic obcinania wprowadzili oni efektywniejsze testy bezpośredniego odrzucania, które działają dla ogólnych obszarów obcinania. Tutaj przedstawimy oryginalne opracowanie Cyrusa-Becka po to, żeby pokazać obcinanie parametryczne. Ponieważ jednak jesteśmy zainteresowani tylko obcinaniem przez prostokąty o bokach równoległych do boków ekranu, na końcu wywodu redukujemy sformułowanie Cyrusa-Becka do bardziej efektywnego przypadku Lianga-Barsky'ego.

Przypomnijmy, że algorytm Cohena-Sutherlanda dla odcinków, które nie mogą być bezpośrednio zaakceptowane albo odrzucone, oblicza przecięcie (x, y) odcinka z krawędzią obcinającą na zasadzie podstawienia znanej wartości x albo y do krawędzi obcinających odpowiednio pionowej lub poziomej. Algorytm parametryczny znajduje wartość

parametru t, w reprezentacji parametrycznej odcinka, dla punktu, w którym odcinek przecina prostą, na której leży krawędź obcinająca. Ponieważ w ogólnym przypadku wszystkie krawędzie obcinające mogą być przecinane przez odcinek, są obliczane cztery wartości t. W celu określenia, które (jeżeli w ogóle któreś) z czterech wartości odpowiadają faktycznemu przecięciu, korzysta się z prostych porównań. Jedynie wtedy oblicza się wartości (x, y) dla jednego lub dwóch faktycznych przecięć. Ogólnie to podejście jest oszczędniejsze czasowo w porównaniu z algorytmem Cohena-Sutherlanda obliczania przecięć, ponieważ unika się powtarzania pętli potrzebnych przy obcinaniu przez kilka krawędzi prostokąta obcinającego. Ponadto obliczenia w jednowymiarowej przestrzeni parametrów są prostsze od podobnych obliczeń w przestrzeni trójwymiarowej. Liang i Barsky ulepszyli algorytm Cyrusa-Becka; sprawdzają oni każdą wartość t wówczas, gdy jest ona generowana w celu odrzucenia niektórych odcinków, zanim zostaną obliczone cztery wartości t.



Rys. 3.28. Iloczyny skalarne dla trzech punktów zewnętrzngo, wewnętrznego i na granicy obszaru obcinającego

Algorytm Cyrusa-Becka wykorzystuje następujące określenie przecięcia dwóch linii. Na rysunku 3.28 pokazano jedną krawędź E_i prostokąta obcinającego i normalną do tej krawędzi N_i skierowaną na zewnątrz (to znaczy na zewnątrz prostokąta obcinającego[7]) oraz odcinek od P_0 do P_1, który ma być obcięty przez krawędź. Zarówno krawędź, jak i odcinek mogą być przedłużone w celu znalezienia punktu przecięcia.

[7] Cyrus i Beck korzystali z normalnej skierowanej do wnętrza; tutaj wolimy jednak korzystać z normalnej skierowanej na zewnątrz ze względu na zgodność z normalnymi do płaszczyzny w przestrzeni 3D, które są skierowane na zewnątrz. Nasze sformułowanie różni się jedynie testowaniem znaku.

Tak jak poprzednio, odcinek jest reprezentowany parametrycznie w postaci

$$P(t) = P_0 + t(P_1 - P_0)$$

przy czym $t = 0$ w P_0 i $t = 1$ w P_1. Weźmy teraz dowolny punkt P_{E_i} na krawędzi E_i i rozważmy trzy wektory $P(t) - P_{E_i}$ poprowadzone od P_{E_i} do trzech punktów na odcinku od P_0 do P_1: poszukiwanego punktu przecięcia, końca odcinka w wewnętrznej półpłaszczyźnie wyznaczonej przez krawędź i końca odcinka w zewnętrznej półpłaszczyźnie krawędzi. To, w którym obszarze leży punkt, możemy określić na podstawie wartości iloczynu skalarnego $N_i \cdot [P(t) - P_{E_i}]$. Ta wartość jest ujemna dla punktu w wewnętrznej półpłaszczyźnie, równa zeru dla punktu na odcinku należącego do krawędzi i ujemna dla punktu, który leży w zewnętrznej półpłaszczyźnie. Definicje wewnętrznej i zewnętrznej półpłaszczyzny określonej przez krawędź odpowiadają numerowaniu krawędzi prostokąta obcinającego w kierunku przeciwnym względem ruchu wskazówek zegara; jest to konwencja, z której korzystamy w całej książce. Teraz możemy znaleźć wartość t dla przecięcia P_0P_1 z krawędzią:

$$N_i \cdot [P(t) - P_{E_i}] = 0$$

Najpierw podstawiamy zamiast $P(t)$

$$N_i \cdot [P_0 + t(P_1 - P_0) - P_{E_i}] = 0$$

Po prostych przekształceniach otrzymujemy

$$N_i \cdot [P_0 - P_{E_i}] + N_i t[P_1 - P_0] = 0$$

Niech $D = (P_1 - P_0)$ będzie wektorem od P_0 do P_1; rozwiązując ze względu na t otrzymujemy

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D} \tag{3.1}$$

Zauważmy, ze poprawną wartość t otrzymujemy tylko wówczas, gdy mianownik wyrażenia jest niezerowy. W celu sprawdzenia tego algorytm sprawdza, czy

$N_i \neq 0$ (to znaczy, że normalna nie może być równa 0; to mogłoby się zdarzyć tylko przez pomyłkę),

$D \neq 0$ (to znaczy $P_1 \neq P_0$),

$N_i \cdot D \neq 0$ (to znaczy, że krawędź E_i i odcinek od P_0 do P_1 nie są równoległe. Gdyby były równoległe, dla tej krawędzi nie ma przecięcia i algorytm przechodzi do następnego przypadku).

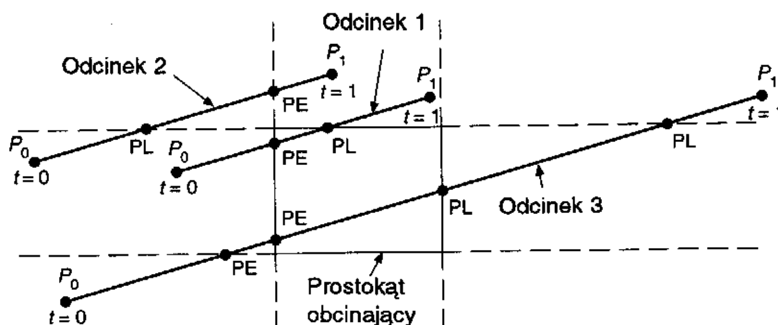
Równanie (3.1) może być wykorzystane do znalezienia przecięcia między P_0P_1 a każdą krawędzią prostokąta obcinającego. Wykonujemy te obliczenia określając normalną i dowolny punkt P_{E_i} – powiedzmy koniec krawędzi – dla każdej krawędzi obcinającej; te wartości wykorzysujemy później dla wszystkich odcinków. Po znalezieniu czterech wartości t dla odcinka, następny krok polega na określeniu, które (jeżeli w ogóle któraś) z wartości odpowiadają wewnętrznym przecięciom odcinka z krawędziami prostokąta obcinającego. W pierwszym kroku każda wartość t spoza przedziału [0, 1] jest odrzucana, ponieważ leży na zewnątrz $P_0 P_1$. Następnie musimy określić, czy przecięcie leży na granicy obcinania.

Moglibyśmy po prostu spróbować sortować pozostałe wartości t, wybierając jako punkty przecięcia wartości pośrednie, tak jak to sugeruje rys. 3.29 dla odcinka 1. Ale jak odróżnić ten przypadek od przypadku odcinka takiego jak odcinek 2, dla którego żaden fragment nie leży wewnątrz prostokąta obcinającego i pośrednie wartości t odpowiadają punktom, które nie leżą na granicy obcinania? Ponadto, które z czterech przecięć odcinka 3 leżą na granicy obcinania?

Przecięcia na rys. 3.29 są klasyfikowane jako *potencjalnie wchodzące* (PE) albo *potencjalnie wychodzące* (PL) z prostokąta obcinającego w następujacy sposób: jeżeli poruszając się od P_0 do P_1 przecinamy określoną krawędź, tak że wchodzimy do wewnętrznej półpłaszczyzny krawędzi, to przecięcie jest typu PE; jeżeli natomiast wychodzimy z wewnętrznej półpłaszczyzny krawędzi, to jest to przecięcie typu PL. Zauważmy, że przy tym rozróżnieniu dwa wewnętrzne punkty przecięcia odcinka przecinającego prostokąt obcinający mają przeciwne etykiety.

Formalnie przecięcia mogą być klasyfikowane jako PE albo PL na podstawie kąta między $P_0 P_1$ i N_i: jeżeli kąt jest mniejszy niż 90°, to przecięcie jest typu PL; jeżeli jest większy niż 90°, to przecięcie jest typu PE. Ta informacja jest zawarta w znaku iloczynu skalarnego dla N_i i $P_0 P_1$:

$$N_i \cdot D < 0 \Rightarrow \text{PE (kąt większy niż 90°)},$$
$$N_i \cdot D > 0 \Rightarrow \text{PL (kąt mniejszy niż 90°)}.$$



Rys. 3.29. Odcinki leżące ukośnie względem prostokąta obcinającego

Zauważmy, że $N_i \cdot D$ jest w zasadzie mianownikiem równania (3.1), co oznacza, że w procesie obliczania t można od razu klasyfikować przecięcie. Przy takim klasyfikowaniu odcinek 3 na rys. 3.29 sugeruje ostatni krok procesu. Musimy wybrać parę (PE, PL), która określa obcięty odcinek. Część nieskończonej linii przechodzącej przez P_0P_1, która leży wewnątrz obszaru obcinającego jest ograniczona przez przecięcie PE o największej wartości t, którą określamy jako t_E, i przecięcie PL o najmniejszej wartości t, t_L. Obcięty odcinek jest określony przez przedział (t_E, t_L). Ponieważ jednak jesteśmy zainteresowani przecięciem P_0P_1 a nie nieskończenie długiej linii, trzeba dalej zmodyfikować definicję zakresu tak żeby $t = 0$ było dolną granicą dla t_E oraz $t = 1$ było górną granicą dla t_L. Co będzie jeżeli $t_E > t_L$? Jest to dokładnie przypadek odcinka 2. Oznacza to, że żadna część P_0P_1 nie jest wewnątrz prostokąta obcinającego i cały odcinek jest odrzucany. Wartości t_E i t_L, które odpowiadają faktycznym przecięciom, są używane do obliczenia odpowiednich współrzędnych x i y.

Kompletny algorytm obcinania przez prostokąt o bokach równoległych do boków ekranu jest zapisany w postaci pseudokodu w programie 3.8. Kompletna wersja kodu, adaptowana z pracy [LIAN84] jest podana w pracy [FOLE90] na rys. 3.45.

Program 3.8 Pseudokod dla parametrycznego algorytmu Cyrusa-Becka obcinania odcinka

```
{
   wstępne obliczenie Ni i wybranie PEi dla każdej krawędzi

   for ( każdy obcinany odcinek )
      if (P1 = P0)
         zdegenerowany odcinek jest obcinany jako punkt;
      else {
         tE = 0;
         tL = 1;
         for ( każde potencjalne przecięcie z krawędzią obcinającą )
            if (Ni · D != 0) {        /* Pomijamy krawędzie równoległe do odcinka */
               obliczanie t;
               wykorzystanie Ni · D do klasyfikowania jako PE albo PL;
               if (PE) tE = max(tE, t);
               if (PL) tL = min(tL, t);
            }
         }
         if (tE > tL)
            return nil;
         else
            return P(tE) and P(tL) jako prawdziwe przecięcia dla obcinania;
      }
   }
}
```

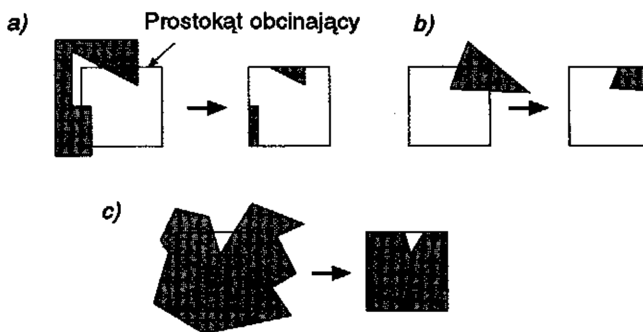
Algorytm Cohena-Sutherlanda jest efektywny wówczas, gdy testowanie kodu obszaru można wykonać tanio (na przykład wykonując operacje na bitach w języku asemblera) i bezpośrednie odrzucanie albo akceptację można zastosować do większości odcinków. Parametryczne obcinanie odcinka wygrywa, gdy trzeba obciąć wiele odcinków i testowanie można wykonać w odniesieniu do wartości parametru. Jednak to obliczanie parametru jest wykonywane nawet dla końców, które byłyby bezpośrednio akceptowane w strategii Cohena-Sutherlanda. Algorytm Lianga-Barsky'ego jest efektywniejszy niż algorytm Cyrusa-Becka ze względu na dodatkowe bezpośrednie testowanie odrzucania, dzięki czemu można uniknąć obliczania wszystkich czterech wartości parametru dla odcinków, które nie przecinają prostokąta obcinającego. Dla odcinków, które nie mogą być bezpośrednio odrzucone przez algorytm Cohena-Sutherlanda, ponieważ nie leżą w niewidocznej półpłaszczyźnie, testy odrzucające Lianga-Barsky'ego są oczywiście lepsze w porównaniu z powtarzanym obcinaniem wymaganym przez Cohena-Sutherlanda. Algorytm Nicholla i in. [NICH87] ogólnie jest lepszy w porównaniu z algorytmami Cohena-Sutherlanda czy Lianga-Barsky'ego, ale nie daje się uogólnić na przypadek 3D jak przy obcinaniu parametrycznym. Metody przyśpieszenia algorytmu Cohena-Sutherlanda są omawiane w pracy [DUVA90].

3.10. Obcinanie okręgów

W celu obcięcia okręgu przez prostokąt możemy najpierw wykonać prosty test akceptacji/odrzucenia, polegający na przecięciu kwadratu opisanego na okręgu z prostokątem obcinającym, wykorzystując do tego algorytm obcinania wielokąta opisany w następnym punkcie. Jeżeli okrąg przecina prostokąt, to dzielimy go na kwadraty i wykonujemy dla każdego z nich prosty test akceptacji/odrzucenia. Te testy mogą z kolei doprowadzić do testowania oktantów. Następnie możemy analitycznie obliczyć przecięcie okręgu z krawędzią rozwiązując odpowiedni układ równań; potem możemy dokonać konwersji otrzymanego łuku za pomocą odpowiedniego algorytmu dla obliczonych (i odpowiednio zaokrąglonych) punktów początkowego i końcowego. Jeżeli konwersja jest szybka albo jeżeli okrąg nie jest zbyt duży, to, być może, lepiej jest dokonać wycinania na zasadzie piksel po pikselu, testując każdy piksel brzegowy z granicami prostokąta (przed zapisaniem). W każdym przypadku użyteczne będzie sprawdzenie opisanego prostokąta. Jeżeli okrąg jest wypełniony, to piksele należące do segmentów mogą być wypełnione bez sprawdzania względem granic; oznacza to, że każdy segment może być obcięty, a potem można wypełnić należące do niego piksele.

3.11. Obcinanie wielokątów

Algorytm obcinania wielokątów musi uzględniać różne przypadki (rys. 3.30). Na szczególną uwagę zasługuje przypadek a) z tego względu, że wielokąt niewypukły jest dzielony na dwa oddzielne wielokąty. W ogóle zadanie obcinania wydaje się raczej złożone. Każda krawędź wielokąta musi być testowana względem każdej krawędzi prostokąta obcinającego; trzeba dodawać nowe krawędzie, a istniejące krawędzie mogą być odrzucone, zachowane albo podzielone. Z obcinania jednego wielokąta może powstać kilka wielokątów. Musimy sobie poradzić z tymi wszystkimi przypadkami w sposób zorganizowany.

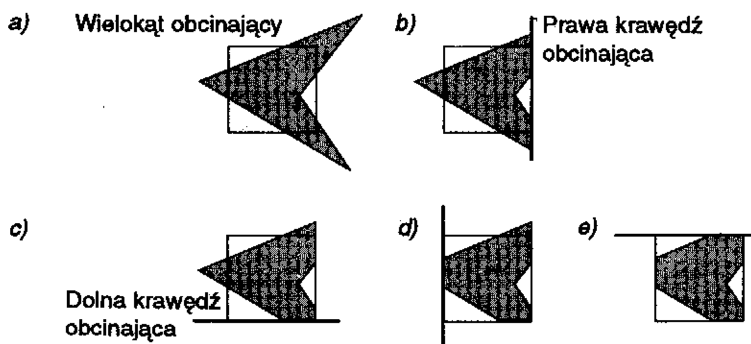


Rys. 3.30. Przykłady obcinania wielokąta: a) wiele elementów; b) prosty przypadek wielokąta wypukłego; c) przypadek wielokąta niewypukłego z wieloma zewnętrznymi krawędziami

3.11.1. Algorytm Sutherlanda-Hodgmana obcinania wielokąta

Algorytm Sutherlanda-Hodgmana obcinania wielokąta [SUTH74b] wykorzystuje strategię *dziel* i *zwyciężaj*: rozwiązuje się szereg prostych identycznych problemów, które po połączeniu dają wynik dla całego problemu. Prosty problem polega na obcinaniu wielokąta przez jedną nieskończenie długą krawędź obcinającą. Cztery krawędzie obcinające, każda zdefiniowana przez jeden z boków prostokąta obcinającego (rys. 3.31), kolejno obcinają wielokąt dając w efekcie obcięcie przez prostokąt.

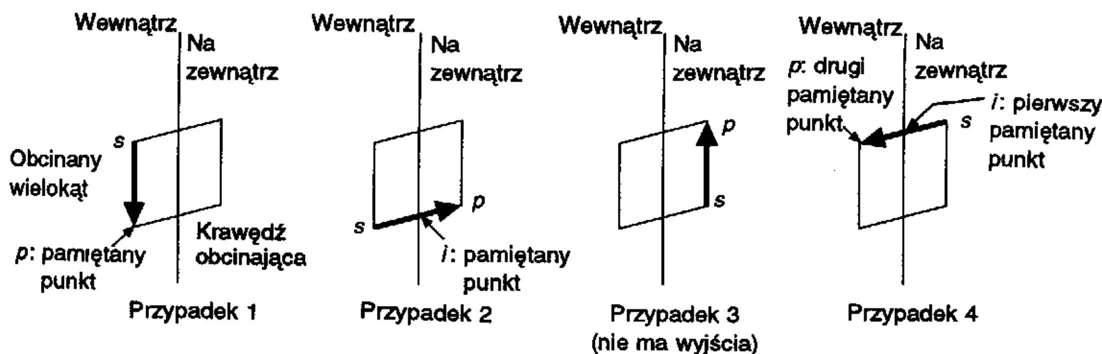
Zauważmy różnicę między tą strategią dla wielokąta a algorytmem Cohena-Sutherlanda obcinania odcinka: w przypadku wielokąta kolejno obcina się przez cztery krawędzie, podczas gdy w przypadku odcinka testuje się kod w celu sprawdzenia, która krawędź obcina, i w razie



Rys. 3.31. Obcinanie wielokąta krawędź po krawędzi: a) przed obcinaniem; b) obcinanie przez prawą krawędź; c) obcinanie przez dolną krawędź; d) obcinanie przez lewą krawędź; e) obcinanie przez górną krawędź; wielokąt jest całkowicie obcięty

potrzeby dokonuje się obcięcia. Rzeczywisty algorytm Sutherlanda-Hodgmana jest ogólniejszy: wielokąt (wypukły albo niewypukły) można obciąć przez dowolny inny wypukły wielokąt; w 3D wielokąty mogą być obcinane przez wypukłe wielościany zdefiniowane przez płaszczyzny. Algorytm akceptuje ciąg wierzchołków wielokąta $v_1, v_2, \ldots, v_n$. W 2D wierzchołki definiują krawędzie wielokąta od v_i do v_{i+1} i od v_n do v_1. Algorytm obcina względem jednej nieskończenie długiej krawędzi obcinającej i oblicza ciąg wierzchołków definiujących obcięty wielokąt. W drugim przebiegu częściowo obcięty wielokąt jest obcinany przez drugą krawędź obcinającą itd.

Algorytm porusza się wokół wielokąta od v_n do v_1 i potem z powrotem do v_n, sprawdzając na każdym kroku zależność między kolejnymi wierzchołkami i krawędzią obcinającą. Trzeba brać pod uwagę cztery możliwe przypadki pokazane na rys. 3.32.



Rys. 3.32. Cztery przypadki obcinania wielokąta

Weźmy pod uwagę krawędź wielokąta od wierzchołka *s* do wierzchołka *p* na rys. 3.32. Załóżmy, że w poprzedniej iteracji został rozpatrzony punkt *s*. W przypadku 1, kiedy krawędź wielokąta jest całkowicie po wewnętrznej stronie granicy obcinania, do listy wyjściowej jest dodawany wierzchołek *p*. W przypadku 2 punkt przecięcia *i* jest zapisywany jako wierzchołek, ponieważ krawędź przecina brzeg. W przypadku 3 oba wierzchołki są na zewnątrz granicy i w związku z tym nie jest zapisywany żaden punkt. W przypadku 4 oba punkty, punkt przecięcia oraz wierzchołek *p* są dodawane do listy wyjściowej.

Funkcja SutherlandHodgmanPolygonClip() w programie 3.9 na podstawie tablicy wierzchołków *inVertexArray* tworzy inną tablicę wierzchołków *outVertexArray*. W celu zachowania prostoty kodu nie pokazujemy kontroli rozmiaru tablicy i wykorzystujemy funkcję Output() do umieszczenia wierzchołka w tablicy *outVertexArray*. Funkcja Intersect() oblicza przecięcie krawędzi wielokąta od wierzchołka *s* do wierzchołka *p* z *clipBoundary*, która jest zdefiniowana przez dwa wierzchołki na brzegu wielokąta obcinającego. Funkcja Inside() zwraca wartość TRUE, jeżeli wierzchołek jest po wewnętrznej stronie brzegu obcinania, przy czym strona wewnętrzna jest zdefiniowana jako „obszar leżący z lewej strony krawędzi obcinającej, jeżeli patrzymy od pierwszego do drugiego wierzchołka krawędzi obcinającej". Odpowiada to numerowaniu krawędzi w kierunku przeciwnym względem ruchu wskazówek zegara. W celu sprawdzenia, czy punkt leży na zewnątrz krawędzi obcinającej możemy sprawdzić znak iloczynu skalarnego normalnej do krawędzi obcinającej i krawędzi wielokąta, tak jak to opisano w p. 3.9.4. (Dla prostego przypadku prostokąta obcinającego o bokach równoległych do boków ekranu wystarczy tylko sprawdzić znak odległości pionowej albo poziomej od jego brzegu.)

Sutherland i Hodgman pokazali strukturę algorytmu [SUTH74b]. Jak tylko pojawi się wierzchołek, program obcinania wywołuje się sam z tym wierzchołkiem. Obcinanie wykonuje się względem następnej krawędzi obcinającej i nie jest potrzebna żadna pomocnicza pamięć do przechowywania częściowo obciętego wielokąta: w istocie wielokąt jest przepuszczany przez potok programów obcinających. Każdy krok może być zrealizowany za pomocą specjalizowanego sprzętu bez żadnej dodatkowej przestrzeni buforowej. Dzięki tej właściwości (i jej ogólności) algorytm nadaje się do realizacji sprzętowych. Jednak w algorytmie o dotychczasowej postaci na brzegu obcinającego wielokąta mogą się pojawić nowe krawędzie. Rozważmy rys. 3.30a – jest wprowadzana nowa krawędź łącząca lewy górny punkt trójkąta i lewy górny punkt prostokąta. Takie krawędzie można usunąć w przetwarzaniu końcowym.

Program 3.9 Algorytm Sutherlanda-Hodgmana obcinania wielokąta

```
typedef struct vertex {
   float  x, y;
} vertex;

typedef vertex edge[2];
typedef vertex vertexArray[MAX];  /* MAX jest to deklarowana stała */

void   Intersect(vertex first, vertex second, vertex *clipBoundary,
               vertex *intersectPt)
{
   if (clipBoundary[0].y == clipBoundary[1].y) {           /* pozioma */
      intersectPt->y = clipBoundary[0].y;
      intersectPt->x = first.x + (clipBoundary[0].y - first.y) *
         (second.x - first.x) / (second.y - first.y);
   } else {                                                 /* pionowa */
      intersectPt->x = clipBoundary[0].x;
      intersectPt->y = first.y + (clipBoundary[0].x - first.x) *
         (second.y - first.y) / (second.x - first.x);
   }
}

boolean   Inside(vertex testVertex, vertex *clipBoundary)
{
   if (clipBoundary[1].x > clipBoundary[0].x)              /* dolna */
      if (testVertex.y >= clipBoundary[0].y) return TRUE;
   if (clipBoundary[1].x < clipBoundary[0].x)              /* górna */
      if (testVertex.y <= clipBoundary[0].y) return TRUE;
   if (clipBoundary[1].y > clipBoundary[0].y)              /* prawa */
      if (testVertex.x <= clipBoundary[1].x) return TRUE;
   if (clipBoundary[1].y < clipBoundary[0].y)              /* lewa */
      if (testVertex.x >= clipBoundary[1].x) return TRUE;
   return FALSE;
}

void   Output(vertex newVertex, int *outLength, vertex *outVertexArray)
{
   (*outLength)++;
   outVertexArray[*outLength - 1].x = newVertex.x;
   outVertexArray[*outLength - 1].y = newVertex.y;
}

void   SutherlandHodgmanPolygonClip(vertex *inVertexArray,
       vertex *outVertexArray, int inLength, int *outLength, vertex *clip_boundary)
{
   vertex s, p, i;
   int    j;
```

```
        *outLength = 0;
        s = inVertexArray[inLength – 1]; /* Start z ostatnim wierzchołkiem w inVertexArray*/
        for (j = 0; j < inLength; j++) {
           p = inVertexArray[j]; /* Teraz s i p odpowiadają wierzchołkom na rys. 3.33 */
           if (Inside(p, clip_boundary)) {                        /* Przypadki 1 i 4 */
              if (Inside(s, clip_boundary))
                 Output(p, outLength, outVertexArray);            /* Przypadek 1 */
              else {                                              /* Przypadek 4 */
                 Intersect(s, p, clip_boundary, &i);
                 Output(i, outLength, outVertexArray);
                 Output(p, outLength, outVertexArray);
              }
           } else if (Inside(s, clip_boundary)) {                 /* Przypadki 2 i 3 */
              Intersect(s, p, clip_boundary, &i);                 /* Przypadek 2 */
              Output(i, outLength, outVertexArray);
           }                                             /* Brak akcji dla przypadku 3 */
           s = p;                           /* Przejście do następnej pary wierzchołków */
        }
     }
```

3.12. Generowanie znaków

3.12.1. Definiowanie i obcinanie znaków

Są dwie podstawowe metody generowania znaków. Najogólniejsza, ale równocześnie najdroższa obliczeniowo metoda polega na definiowaniu każdego znaku jako krzywej albo konturu wielokątowego i w razie potrzeby dokonywaniu konwersji. Najpierw omówimy inną prostszą metodę, w której każdy znak w danym kroju pisma jest specyfikowany jako mała prostokątna mapa bitowa. Generowanie znaku wymaga wtedy tylko zwykłego użycia funkcji copyPixel do skopiowania obrazu znaku z kanwy pozaekranowej, określanej jako kanwa *kroju pisma*, do bufora ramki w wymaganym miejscu.

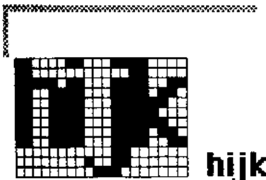
Pamięć podręczna dla kroju pisma może znajdować się w pamięci obrazu. W większości systemów graficznych, w których monitor jest odświeżany z własnej pamięci obrazu, ta pamięć jest większa niż rzeczywiście potrzebna dla wyświetlanego obrazu. Na przykład, piksele dla prostokątnego obrazu mogą być zapamiętane w kwadratowej pamięci i pozostaje wtedy prostokątny pasek niewidocznej pamięci obrazu. W innym przypadku może być dostatecznie dużo pamięci na zapamiętanie dwóch ekranów: jednego, który jest odświeżany, i drugiego, który jest rysowany; mamy wtedy do czynienia z podwójnym buforem. Pamięć podręczna dla bieżąco wyświetlanego kroju pisma (krojów pisma)

często znajduje się w takiej niewidocznej pamięci obrazu, ponieważ copyPixel kontrolera monitora działa szybciej w obrębie lokalnej pamięci obrazu. Podobne zastosowanie takiej niewidocznej pamięci wiąże się z zapamiętywaniem obszarów ekranu czasowo przesłanianych przez chwilowe obrazy, np. okna, menu itp.

Mapy bitowe dla pamięci podręcznej kroju pisma są zazwyczaj tworzone na zasadzie skanowania powiększonych obrazów znaków z drukarskich krojów pisma o różnych wielkościach; projektant czcionek, jeżeli zajdzie taka potrzeba, może z kolei użyć programu malarskiego do poprawienia poszczególnych pikseli w każdej mapie bitowej znaku. Projektant może użyć programu malarskiego do tworzenia od nowa krojów pisma, które są specjalnie projektowane dla monitorów i drukarek małej rozdzielczości. Ponieważ skalowanie małej mapy bitowej nie jest efektywne, dla danego znaku w danym kroju pisma, w celu uzyskania różnych standardowych wielkości, trzeba zdefiniować więcej niż jedną mapę bitową. Ponadto dla każdej czcionki jest potrzebny zestaw map bitowych; dlatego dla każdego kroju pisma załadowanego przez program użytkownika jest potrzebna odrębna pamięć podręczna.

Mapy bitowe znaków są automatycznie obcinane przez SRGP w ramach implementacji copyPixel. Każdy znak jest obcinany do docelowego prostokąta na zasadzie piksel po pikselu, metodą, która umożliwia obcinanie znaku w dowolnym wierszu lub kolumnie jego mapy bitowej. Dla systemów z wolną operacją copyPixel znacznie szybszą, chociaż zgrubną, jest metoda obcinania znaku albo nawet całego tekstu na zasadzie wszystko albo nic, w wyniku bezpośredniej akceptacji znaku albo prostokąta opisanego na znaku. copyPixel w stosunku do znaku albo ciągu znaków wykonuje się tylko wówczas, gdy opisany na nim prostokąt został zaakceptowany w sposób bezpośredni. Nawet dla systemów z szybszą operacją copyPixel opłaca się robić bezpośredni test typu akceptacja/odrzucenie prostokąta opisanego na ciągu znaków, jako poprzednika obcinania znaków w czasie operacji copyPixel.

W SRGP, w metodzie z pamięcią podręczną, dla kroju pisma w kanwie są pamiętane proste mapy bitowe znaków, jedna obok drugiej; ta kanwa jest szeroka, jej wysokość natomiast jest określona przez najwyższy znak; na rys. 3.33 pokazano fragment pamięci podręcznej oraz dyskretne kopie tych samych znaków przy małej rozdzielczości. Każdy załadowany krój pisma jest opisany przez strukturę (deklaracja jest w programie 3.10) zawierającą wskazanie kanwy, która pamięta obrazy znaków wraz z informacją o wysokości znaków i ilością miejsca, jakie musi być zostawione między sąsiednimi znakami w ciągu tekstowym. (W niektórych pakietach odstęp między znakami jest pamiętany jako część szerokości znaku, co pozwala na zmienne odległości między znakami.)



Rys. 3.33. Fragment przykładowej pamięci podręcznej

Program 3.10 Deklaracje typu dla pamięci podręcznej kroju pisma

```
typedef struct charLocation {
   int leftX, width;          /* Położenie w poziomie, szerokość obrazu w pamięci
                                 podręcznej kroju pisma */
} charLocation;

typedef struct fontCacheDescriptor {
   canvasIndexInteger cache;
   int descenderHeight, totalHeight; /* Wysokość jest stała; szerokość się zmienia */
   int interCharacterSpacing;        /* Mierzone w pikselach */
   charLocation locationTable[128];
} fontCacheDescriptor;
```

Jak opisano w p. 2.1.5, wielkość części znajdującej się poniżej linii bazowej i ogólna wysokość liter są dla danego kroju pisma stałe – ta pierwsza jest liczbą wierszy pikseli na dole pamięci podręcznej kroju pisma, ta druga jest po prostu wysokością kanwy pamięci podręcznej. Szerokość znaku nie jest traktowana jako wielkość stała; dlatego znak może zajmować tyle miejsca, ile trzeba, a nie musi być dopasowywany do pola o stałej szerokości. W trakcie pisania tekstu SRGP wstawia stały odstęp między znakami; wielkość tego odstępu jest określona jako część deskryptora każdego kroju pisma. W zastosowaniach związanych z przetwarzaniem tekstu wiersze tekstu mogą być wyświetlane za pomocą SRGP, który wyświetla poszczególne wyrazy tekstu; wiersze mogą być wyrównywane do prawej strony dzięki zmiennym odstępom między wyrazami – po znakach interpunkcyjnych wiersze są tak wypełniane, żeby znaki ostatnie z prawej strony były wyrównane do prawego marginesu. W takim programie użytkowym wykorzystuje się prostokąty opisane na tekście do określenia, gdzie jest prawy koniec każdego wyrazu, i z kolei obliczenia początku następnego wyrazu. Oczywiście możliwości redagowania tekstu są w SRGP bardzo ograniczone i odbiegają od możliwości dostępnych w wyrafinowanych procesorach tekstu, nie mówiąc już o programach do składania tekstu, w których potrzebna jest znacznie dokładniejsza kontrola nad odstępami między poszczególnymi literami po to, żeby było możliwe realizowanie takich efektów jak indeksy dolne i górne, regulowanie odstępów między poszczególnymi literami i drukowanie tekstu, który nie jest wyrównany w poziomie.

3.12.2. Implementacja prymitywu Text Output

W kodzie z programu 3.11 pokazaliśmy, jak w SRGP tekst jest implementowany wewnętrznie: każdy znak w danym ciągu jest umieszczany indywidualnie, a odstęp między znakami jest określony przez od-

powiednie pole w deskryptorze kroju pisma. Zauważmy, że problemy takie jak mieszanie krojów pisma w ciągu znaków muszą być rozwiązywane przez program użytkowy.

Program 3.11 Implementacja umieszczania znaku w prymitywie tekstu w SRGP

```
void SRGP_characterText(point origin, char stringToPrint,
        fontCacheDescriptor fontInfo)
/* punkt odniesienia dla umieszczenia znaku w bieżącej kanwie */
{
   rectangle fontCacheRectangle;
   char charToPrint;
   int i;
   charLocation *fp;

/* Punkt odniesienia określony przez program użytkowy dla linii bazowej; nie jest
   uwzględniona dolna część litery */
   origin.y -= fontInfo.descenderHeight;

   for (i = 0; i < strlen(stringToPrint); i++) {
      charToPrint = stringToPrint[i];
      fp = &fontInfo.locationTable[charToPrint];
/* Wyznaczenie prostokątnego obszaru w pamięci podręcznej, w którym leży znak */
      SRGP_defPoint(fp->leftX, 0, fontCacheRectangle.bottomLeft);
      SRGP_defPoint(fp->leftX + fp->width - 1, fontInfo.totalHeight - 1,
         fontCacheRectangle.topRight);
      SRGP_copyPixel(fontInfo.cache, fontCacheRectangle, origin);
 /* Takie uaktualnienie punktu odniesienia, żeby przesunąć się poza dotychczasowy
    znak plus odstęp między znakami */
      origin.x += fp->width + interCharacterSpacing;
   }
}
```

Wspomnieliśmy, że metoda mapy bitowej wymaga różnych pamięci podręcznych dla każdej kombinacji kroju pisma, wielkości i czcionki dla różnych rozdzielczości ekranu i urządzenia zewnętrznego. Jeden krój pisma dla ośmiu wielkości i czterech czcionek (zwykła, pogrubiona, kursywa i pogrubiona kursywa) wymaga 32 podręcznych pamięci! Jeden ze sposobów rozwiązania tego problemu polega na tym, żeby reprezentować znak w abstrakcyjnej formie niezależnej od urządzenia zewnętrznego, wykorzystując do tego wielomianowe albo krzywoliniowe opisy kształtów zdefiniowanych przez zmiennopozycyjne parametry, a potem ich odpowiednie przekształcanie. Funkcje wielomianowe, określane jako *krzywe sklejane* (por. rozdz. 9), dają gładkie krzywe z ciągłymi pochodnymi, pierwszą oraz wyższymi, i są powszechnie używane do kodowania konturów tekstu. Chociaż każdy opis znaku zajmuje więcej miejsca niż reprezentacja w pamięci podręcznej, jedna zapamiętana

reprezentacja umożliwia otrzymanie reprezentacji znaków o różnych wielkościach na zasadzie odpowiedniego skalowania; również aproksymację kursywy można szybko otrzymać na zasadzie pochylania konturu. Inną wielką zaletą pamiętania znaków w postaci całkowicie niezależnej od urządzenia jest to, że kontury mogą być dowolnie przesuwane, obracane, skalowane i obcinane (albo same mogą być używane jako obszary obcinające).

Efektywność pamięciowa reprezentowania znaków krzywymi sklejanymi nie jest tak wielka, jak może sugerować ten opis. Na przykład na zasadzie skalowania jednego kształtu nie można otrzymać wszystkich wielkości wyrażonych w punktach – kształt kroju pisma o odpowiednim wyglądzie estetycznym z reguły jest funkcją wielkości wyrażonej w punktach; dlatego dla każdego kroju pisma jest tylko ograniczona liczba wielkości wyrażonych w punktach. Ponadto konwersja tekstu z krzywymi sklejanymi wymaga znacznie więcej obliczeń niż prosta implementacja copyPixel, ponieważ forma niezależna od urządzenia musi być zamieniona na współrzędne pikseli z uwzględnieniem bieżącej wielkości, czcionki i atrybutów przekształcania. Dlatego metoda pamięci podręcznej dla krojów pisma jest wciąż najbardziej rozpowszechniona w komputerach osobistych, a nawet jest używana w wielu stacjach roboczych. Strategia, która wykorzystuje zalety obu metod, polega na tym, żeby pomiętać kroje pisma w postaci konturów, natomiast wykonywać konwersję krojów używanych w danym zastosowaniu do odpowiednika w postaci mapy bitowej – na przykład do tworzenia zawartości pamięci podręcznej na bieżąco. Dokładniejszy opis tekstów z krzywymi sklejanymi można znaleźć w pracy [FOLE90], p. 19.4.

3.13. SRGP_copyPixel

Jeżeli są dostępne tylko funkcje niskiego poziomu WritePixel i ReadPixel, to funkcja SRGP_copyPixel może być zaimplementowana dla każdego piksela jako podwójnie zagnieżdżona pętla **for**. Dla uproszczenia załóżmy najpierw, że pracujemy na monitorze dwupoziomowym i nie musimy się zajmować problemami niskiego poziomu zapisu bitów, które nie są wyrównane w słowach. W wewnętrznej pętli naszej prostej funkcji SRGP_copyPixel wykonujemy operację ReadPixel pikseli źródłowego i docelowego, łączymy je logicznie zgodnie z trybem zapisu SRGP i wykonujemy operację zapisu wyniku (WritePixel). Traktując tryb **replace**, najpopularniejszy tryb zapisu, jako specjalny przypadek, możemy użyć prostszej pętli wewnętrznej, która wykonuje jedynie ReadPixel/WritePixel źródła do miejsca docelowego bez konieczności

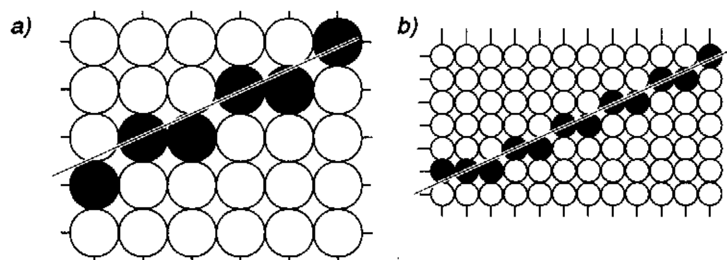
wykonywania operacji logicznej. W czasie obliczania adresu wykorzystuje się prostokąt obcinający do ograniczenia obszaru, do którego są zapisywane piksele docelowe.

3.14. Metody usuwania zakłóceń

3.14.1. Zwiększanie rozdzielczości

Dotychczas przy rysowaniu prymitywów był jeden wspólny problem: miały one „zębate" krawędzie. Ten niepożądany efekt jest wynikiem podejścia do procesu rasteryzacji na zasadzie wszystko albo nic, zgodnie z którą barwa każdego piksela jest albo zastępowana przez barwę prymitywu, albo pozostaje bez zmiany. Zakłócenia tego typu są przykładem zjawiska określanego mianem *aliasingu*. Metody redukcji albo eliminacji aliasingu są określane jako metody antyaliasingu (odkłócania), a obiekty lub obrazy otrzymane z wykorzystaniem takich metod są określane jako odkłócone. W rozdziale 14 pracy [FOLE90] omówiono podstawowe idee z zakresu przetwarzania sygnałów, które wyjaśniają pochodzenie nazwy, źródła i sposoby ich redukcji albo eliminacji przy tworzeniu obrazów. Tutaj ograniczymy się do bardziej intuicyjnego wyjaśnienia, dlaczego w prymitywach SRGP występuje aliasing i opiszemy, jak zmodyfikować algorytm rasteryzacji odcinka podany w tym rozdziale, żeby było możliwe generowanie odkłóconych odcinków.

Weźmy pod uwagę algorytm z punktem środkowym rysujący na białym tle czarny odcinek o grubości 1 piksela, o nachyleniu między 0 a 1. W każdej kolumnie, przez którą przechodzi odcinek, algorytm wstawia barwę piksela, który jest najbliższy odcinka. Za każdym razem, kiedy odcinek przesuwa się między kolumnami, w których piksele najbliższe odcinka nie są w tym samym wierszu, w odcinku zapisywanym do kanwy jest ostry ząbek, co widać na rys. 3.34a. To samo



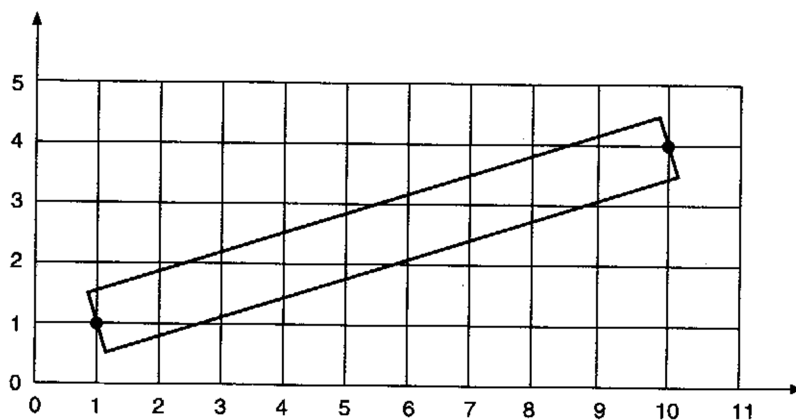
Rys. 3.34. Standardowy odcinek narysowany metodą z punktem środkowym na monitorze dwupoziomowym (a); ten sam odcinek na monitorze o dwa razy większej rozdzielczości liniowej (b)

jest prawdziwe dla innych rasteryzowanych prymitywów, w których pikselowi można przypisać tylko jedną z dwóch wartości jasności.

Załóżmy, że korzystamy teraz z urządzenia wyświetlającego o rozdzielczości dwa razy większej w pionie i dwa razy większej w poziomie. Jak pokazano na rys. 3.34b, odcinek przechodzi przez dwa razy więcej kolumn i dlatego ma dwa razy więcej ząbków, ale każdy ząbek jest o połowę mniejszy w kierunku x i w kierunku y. Chociaż otrzymany obraz wygląda lepiej, polepszenie uzyskujemy za cenę czterokrotnego zwiększenia kosztu pamięci, pasma pamięci i czasu rasteryzacji. Wzrost rozdzielczości jest kosztownym rozwiązaniem, które zmniejsza jedynie problem ząbków, ale nie eliminuje go. W następnych punktach pokażemy mniej kosztowne metody usuwania zakłóceń, a mimo to dające istotnie lepsze obrazy.

3.14.2. Bezwagowe próbkowanie powierzchni

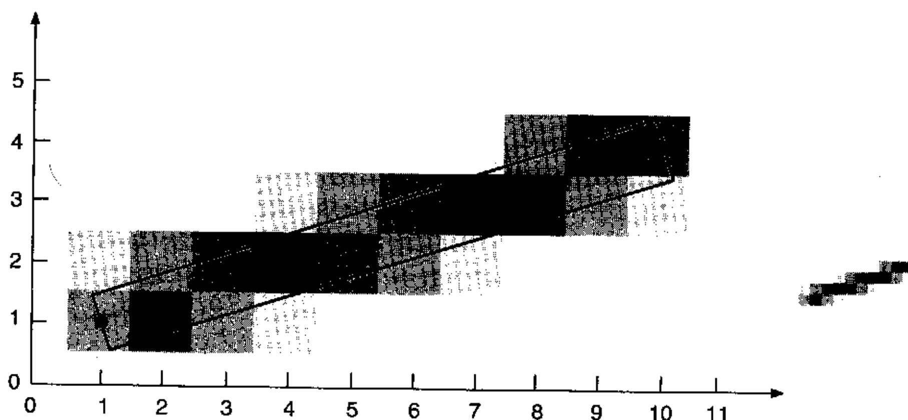
Pierwsza możliwość polepszenia jakości obrazu może wynikać ze spostrzeżenia, że chociaż idealny prymityw, np. odcinek, ma zerową szerokość, to prymityw, który rysujemy, ma niezerową szerokość. Prymityw po rasteryzacji zajmuje skończoną powierzchnię na ekranie – nawet najcieńsza linia pozioma lub pionowa zajmuje na powierzchni ekranu szerokość 1 piksela, a odcinki nachylone pod innymi kątami mają szerokości zmienne wzdłuż prymitywu. Dlatego o dowolnym odcinku myślimy jak o prostokącie o wymaganej szerokości, pokrywającym część siatki (rys. 3.35). Z rysunku wynika, że odcinek nie powinien mieć tylko jednego czarnego piksela w kolumnie, lecz powinien raczej wnosić pewien udział do jasności każdego piksela w kolumnie, którego powierzchnię



Rys. 3.35. Odcinek o niezerowej szerokości o końcach w punktach (1, 1) oraz (10, 4)

przecina. (Taka zmieniająca się jasność może być pokazana oczywiście tylko na monitorach z wieloma bitami na piksel). Wobec tego dla odcinków o szerokości 1 piksela tylko dla odcinków pionowych i poziomych w kolumnie albo w wierszu będzie ustawiony dokładnie 1 piksel. Dla odcinków o innych kątach w kolumnie albo w wierszu powinny być ustawiane przynajmniej 2 piksele, każdy z odpowiednią jasnością.

Ale jaka jest geometria piksela? Jaka jest jego wielkość? Jaką jasność powinien wnosić odcinek do każdego przecinanego piksela? Z punktu widzenia obliczeniowego jest wygodnie przyjąć, że piksele tworzą tablicę rozłącznych kwadratów o środkach w punktach siatki pokrywających ekran, a nie są rozłącznymi kółkami, tak jak to było przyjęte wcześniej w tym rozdziale. (Gdy mówimy o prymitywie pokrywającym całość lub część piksela, mamy na myśli to, że pokrywa on całość lub część kwadratu; dla podkreślenia tego czasami mówimy o kwadraciku jak o obszarze reprezentowanym przez piksel.) Zakładamy również, że odcinek wnosi do jasności każdego piksela wielkość proporcjonalną do procentowego pokrycia kwadratu przez odcinek. Piksel pokryty w całości na monitorze czarno-białym będzie czarny, podczas gdy częściowo pokryty będzie szary o jasności zależnej od stopnia pokrycia piksela przez odcinek. Tę metodę zastosowaną do odcinka z rys. 3.35 pokazano na rys. 3.36.



Rys. 3.36. Jasność piksela jest proporcjonalna do jego powierzchni pokrytej przez odcinek

Dla czarnego odcinka na białym tle piksel (2, 1) jest czarny w około 70%, natomiast piksel (2, 2) jest czarny w około 25%. Piksele (2, 3) nie przecięte przez odcinek są całkowicie białe. Dobranie jasności piksela proporcjonalnie do powierzchni pokrytej przez prymityw łagodzi ostry charakter (typu piksel włączony-wyłączony) krawędzi prymitywu i daje łagodniejsze przejście między pikselami w pełni włączonymi

i w pełni wyłączonymi. Przy takim rozmyciu z pewnej odległości odcinek wygląda lepiej, chociaż w wierszu albo w kolumnie jest włączona większa liczba pikseli. Pierwsze przybliżenie dla stopnia pokrycia piksela można znaleźć dzieląc piksel na drobniejszą siatkę prostokątnych podpikseli, a potem zliczenie podpikseli wewnątrz odcinka – na przykład poniżej górnej krawędzi odcinka albo powyżej dolnej krawędzi (zob. zadanie 3.25).

Metodę dobierania jasności proporcjonalnie do pokrytej powierzchni określamy jako *bezwagowe próbkowanie powierzchni.* Ta metoda daje znacząco lepsze wyniki niż metoda dobierania tylko pełnej albo zerowej jasności; jeszcze efektywniejsza jest metoda określana jako *wagowe próbkowanie powierzchni.* W celu wyjaśnienia różnicy między tymi dwoma sposobami próbkowania powierzchni zauważmy, że bezwagowe próbkowanie powierzchni ma trzy następujące właściwości. Po pierwsze, jasność piksela przeciętego przez krawędź odcinka zmniejsza się w funkcji odległości od środka piksela; im prymityw jest dalej, tym ma mniejszy wpływ na jasność piksela. Ta zależność jest słuszna, ponieważ jasność maleje wówczas, gdy pole pokrycia maleje, a to pole z kolei maleje w miarę oddalania krawędzi odcinka od środka piksela w kierunku brzegu piksela. Jeżeli odcinek całkowicie pokrywa piksel, to pole pokrycia, a co za tym idzie jasność, mają wartości maksymalne; jeżeli krawędź prymitywu jest styczna do brzegu piksela, to pole i jasność są równe zeru.

Druga właściwość bezwagowego próbkowania powierzchni polega na tym, że prymityw nie może wpływać na jasność piksela, jeżeli nie przecina on piksela, to znaczy, jeżeli nie przecina kwadratu reprezentującego piksel. Trzecia właściwość bezwagowego próbkowania polega na tym, że równe pola wnoszą równe jasności, niezależnie od odległości między środkiem piksela a powierzchnią; istotna jest tylko całkowita wielkość pokrytej powierzchni. Dlatego mała powierzchnia w rogu piksela wnosi taki sam udział jak taka sama powierzchnia blisko środka piksela.

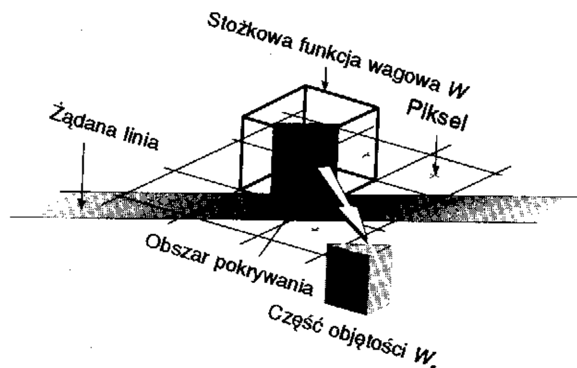
3.14.3. Wagowe próbkowanie powierzchni

Przy wagowym próbkowaniu powierzchni zachowujemy pierwsze dwie właściwości bezwagowego próbkowania (jasność maleje wraz z maleniem wielkości pola pokrycia i prymityw ma udział jedynie wówczas, gdy pokrywa pole reprezentowane przez piksel), a zmieniamy właściwość trzecią. Zgadzamy się na to, żeby udział takich samych powierzchni nie był taki sam: mała powierzchnia blisko środka piksela ma większy wpływ niż powierzchnia znajdująca się w większej odległości. Teoretyczną podstawę takiego podejścia podano w rozdz. 14 pracy [FOLE90], gdzie próbkowanie wagowe jest dyskutowane w kontekście teorii filtrowania.

W celu zachowania drugiej właściwości musimy przyjąć następującą zmianę w geometrii piksela. Przy bezwagowym próbkowaniu, jeżeli krawędź prymitywu jest blisko brzegu kwadratu, z którego dotychczas korzystaliśmy do reprezentowania piksela, ale go nie przecina, to prymityw nie wnosi nic do jasności piksela. W naszym nowym podejściu piksel jest reprezentowany przez koło o powierzchni większej niż powierzchnia kwadratu; prymityw będzie przecinał tę większą powierzchnię; dlatego będzie miał udział w jasności piksela. Zauważmy, że oznacza to, że powierzchnie związane z sąsiednimi pikselami faktycznie nachodzą na siebie.

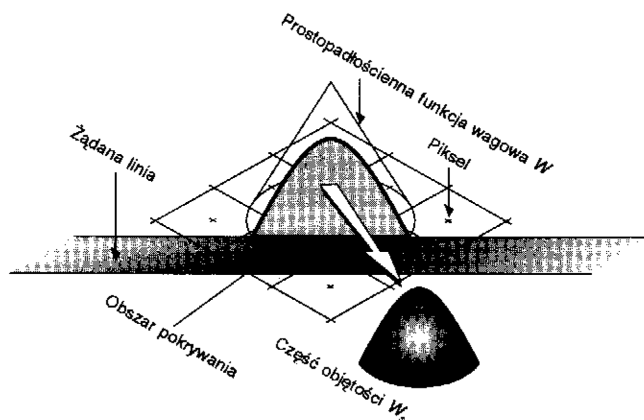
W celu wyjaśnienia pochodzenia przymiotników bezwagowy i wagowy, definiujemy *funkcję wagową*, która określa wpływ danej małej powierzchni dA prymitywu na jasność piksela, jako funkcję odległości od środka piksela. Ta funkcja jest stała dla próbkowania bezwagowego i maleje ze wzrostem odległości dla próbkowania wagowego. Pomyślmy o funkcji wagowej jako o funkcji $W(x, y)$ na płaszczyźnie, której wysokość nad płaszczyzną (x, y) daje wagę dla powierzchni dA w punkcie (x, y). Dla bezwagowego próbkowania, gdy piksele są reprezentowane przez prostokąty, wykresem W jest prostopadłościan (rys. 3.37).

Na rysunku pokazano prostokątne piksele o środkach zaznaczonych krzyżykami na przecięciach linii siatki; funkcję wagową pokazano jako prostopadłościan, którego podstawą jest bieżący piksel. Jasność wnoszona przez powierzchnię piksela pokrytą przez prymityw jest sumą jasności wnoszonych przez wszystkie małe powierzchnie w obszarze nakładania się prymitywu i piksela. Jasność wnoszona przez każde małe pole jest proporcjonalna do powierzchni pomnożonej przez wagę. Dlatego cała jasność jest całką funkcji wagowej po pokrytej powierzchni. Część objętości W_s reprezentowana przez tę całkę jest zawsze ułamkiem o wartości między 0 a 1, a jasność piksela I jest równa $I_{max}W_s$. Na rysunku 3.37 W_s jest klinem wyciętym z prostopadłościanu. Funkcja



Rys. 3.37. Filtr prostopadłościenny dla kwadratowego piksela

wagowa jest również określana jako *funkcja filtrująca*, a prostopadłościan jako *filtr prostopadłościenny*. Dla bezwagowego próbkowania wysokość prostopadłościanu ma znormalizowaną wartość 1 i dlatego objętość prostopadłościanu jest równa 1; w efekcie szeroka linia pokrywająca cały piksel ma jasność $I = I_{max} \cdot 1 = I_{max}$.



Rys. 3.38. Filtr stożkowy dla kołowego piksela o średnicy dwóch skoków siatki

Skonstruujmy teraz funkcję wagową dla próbkowania wagowego; musi ona przypisać mniejszą wagę małym obszarom odległym od środka piksela niż bliższym obszarom. Weźmy jako funkcję wagową najprostszą malejącą funkcję odległości; na przykład wybierzmy funkcję, która ma maksimum w środku piksela i maleje liniowo wraz ze wzrostem odległości od środka. Ze względu na symetrię obrotową wykres tej funkcji tworzy stożek. Podstawa stożka (często nazywana *podstawą filtru*) powinna mieć promień większy, niż moglibyśmy oczekiwać; teoria filtrowania mówi, że dobrze jest przyjąć promień równy skokowi siatki całkowitoliczbowej. Wobec tego prymityw dość odległy od środka piksela może wciąż oddziaływać na jasność tego piksela; ponadto podstawy związane z sąsiednimi pikselami nakładają się i dlatego mały fragment prymitywu może wpływać na kilka różnych pikseli (rys. 3.38). Dzięki temu nakładaniu każdy fragment powierzchni siatki jest pokryty przez jakiś piksel; gdyby okrągły piksel miał promień równy połowie skoku siatki, to pozostawałyby wolne miejsca[8].

[8] Jak zauważono w p. 3.2, piksele wyświetlane na monitorze mają prawie kołowy przekrój i sąsiednie piksele na ogół przecinają się; jednak model nakładających się kółek wykorzystywany w próbkowaniu wagowym nie ma bezpośredniego związku z tym faktem i jest słuszny również dla innych technologii wyświetlania, takich jak ekrany LCD albo ekrany plazmowe, w których rzeczywiste piksele są rozłącznymi kwadratami.

Tak jak w przypadku filtru prostopadłościennego, dla filtru stożkowego suma wszystkich wnoszonych jasności jest reprezentowana przez objętość W_s części stożka leżącej nad częścią wspólną podstawy stożka i prymitywu (rys. 3.38). Podobnie jak w przypadku filtru prostopadłościennego, wysokość stożka jest najpierw tak normalizowana, żeby objętość całego stożka wynosiła 1; dzięki temu piksel, który jest całkowicie pokryty przez prymityw, jest wyświetlany z maksymalną jasnością. Chociaż udziały wnoszone przez części powierzchni prymitywu odległe od środka piksela, ale przecinające podstawę filtru są małe, odcinek wnosi pewien udział do jasności piksela, którego środek jest dostatecznie blisko odcinka. I odwrotnie, piksel, który w modelu kwadratowym był całkowicie pokryty przez odcinek o szerokości jednostkowej[9], nie jest tak jasny jak przy tamtym modelu. Ostateczny efekt próbkowania wagowego polega na zmniejszeniu kontrastu między sąsiednimi pikselami w celu polepszenia gładkości przejścia. W szczególności, przy próbkowaniu wagowym poziomy albo pionowy odcinek o jednostkowej szerokości ma więcej niż jeden podświetlony piksel w każdej kolumnie albo wierszu, co nie występowałoby przy próbkowaniu bezwagowym.

Filtr stożkowy ma dwie użyteczne właściwości: symetrię obrotową i liniowy spadek wartości funkcji ze wzrostem promienia. Preferujemy symetrię obrotową, ponieważ zapewnia obliczenia niezależne od kąta nachylenia odcinka; ponadto jest ona optymalna z punktu widzenia teoretycznego. Zauważmy jednak, że liniowe nachylenie stożka i jego promień są jedynie aproksymacjami optymalnej funkcji filtrującej; ale i tak filtr stożkowy jest lepszy niż filtr prostopadłościenny [FOLE90]. Filtry optymalne są najkosztowniejsze obliczeniowo, podczas gdy filtry prostopadłościenne są najtańsze, i wobec tego filtry stożkowe są rozsądnym kompromisem między kosztem a jakością. Moglibyśmy dość łatwo wbudować filtr stożkowy do naszych algorytmów rasteryzacji. Szczegóły takiego zabiegu można znaleźć w pracy [FOLE90].

3.15. Zaawansowane problemy

W tym rozdziale zaledwie dotknęliśmy koncepcji obcinania i rasteryzacji. W praktyce powstaje wiele złożonych sytuacji, które wymagają stosowania zaawansowanych metod. Są one omawiane w rozdz. 19 pracy [FOLE90], ale warto je tutaj chociaż wymienić.

[9] Mówimy tutaj raczej „odcinek o jednostkowej szerokości" niż „odcinek o grubości 1 piksela" po to, żeby podkreślić, że jednostka szerokości linii jest wciąż tą z siatki SRGP, podczas gdy podstawa filtru ma średnicę równą dwóm jednostkom.

Obcinanie. O ile algorytmy obcinania, które omówiliśmy w tym rozdziale, będą w większości przypadków działały poprawnie, to nie zawsze będą robiły to efektywnie. Ponadto, w niektórych sytuacjach nie będą precyzyjne albo wręcz nie dadzą poprawnej odpowiedzi. Jednym z ulepszonych algorytmów jest algorytm Nicholla-Lee-Nicholla obcinania 2D, który daje ogromne przyspieszenie w stosunku do algorytmów Lianga-Barsky'ego i Cohena-Sutherlanda. Powstają również takie sytuacje obcinania, których nawet nie rozważaliśmy, na przykład obcinanie dowolnego wielokąta przez dowolny wielokąt. W takich sytuacjach można korzystać z algorytmu Weilera.

Rasteryzacja prymitywów. Dotychczas rozważyliśmy rasteryzację prostych prymitywów – odcinków, okręgów i wielokątów. Znane są dokładniejsze i efektywniejsze algorytmy dla tych prymitywów, a także metody rasteryzacji bardziej złożonych prymitywów, np. elipsy, łuki eliptyczne, krzywe sześcienne i przekroje stożkowe. Są również algorytmy dla pogrubionych prymitywów, w których brzeg nie jest po prostu obszarem matematycznym, ale ma jakąś szerokość. Do tej klasy problemów należy efektywne łączenie pogrubionych prymitywów. Wreszcie często trzeba wypełniać samoprzecinające się wielokąty, dla których nie jest oczywiste, co jest wewnątrz, a co na zewnątrz.

Usuwanie zakłóceń. Jest znacznie więcej sytuacji wymagających usuwania zakłóceń niż w rozważonych przez nas przypadkach odcinków. Musimy mieć również znacznie lepszą znajomość teorii próbkowania po to, żeby dobrze usuwać zakłócenia. Są potrzebne specjalne algorytmy dla okręgów, przekrojów stożkowych, dowolnych krzywych oraz dla prostokątów, wielokątów i końców odcinków.

Tekst. Tekst jest wysoce specjalizowanym obiektem i omówione wcześniej metody zazwyczaj nie wystarczają. W tym rozdziale omówiliśmy wykorzystanie pamięci podręcznej do pamiętania znaków, które mogłyby być później kopiowane bezpośrednio do mapy bitowej, ale sygnalizowaliśmy również pewne ograniczenia takiego podejścia: dla każdej wielkości tekstu może być potrzebna inna pamięć podręczna i odstępy między znakami są stałe. Ponadto, chociaż przy korzystaniu z takiej podręcznej pamięci kroju pisma można utworzyć wersje tekstu pogrubionego albo pochylonego, nie są one zadowalające. Nawet jeżeli mamy dokładny rysunek geometrii znaku, w wersji dostarczonej przez projektanta kroju pisma, nie możemy dokonać rasteryzacji na zasadzie segment po segmencie. Ogólnie wyniki będą nieakceptowalne. Do wyświetlania tekstu zostały opracowane specjalne metody, z uwzględnieniem usuwania zakłóceń.

Algorytmy wypełniania. Czasami, po narysowaniu sekwencji prymitywów, możemy chcieć je pokolorować albo pokolorowć obszar narysowany odręcznie. Na przykład może być łatwiej zrobić wzór mozaiki, tworząc siatkę i potem wypełniając ją różnymi barwami, niż rozmieszczać

równo pokolorowane wcześniej kwadraty. Zauważmy, że jeżeli jest wykorzystywana pierwsza metoda, to nie są rysowane żadne prymitywy 2D: używamy tylko obszarów 2D wypełnionych tłem, które powstały po narysowaniu odcinków. W celu określenia, jak duże pole jest do pokolorowania, trzeba wykryć moment osiągnięcia brzegu. Algorytmy, które wykonują taką operację, są nazywane *algorytmami wypełniającymi*. Wśród nich wyróżnia się algorytmy wypełniania konturu, wypełniania wnętrza określanego przez barwę wybranego piksela albo wnętrza określanego przez zanikanie określonej barwy. Każdy algorytm ma swoje zastosowanie; wiele systemów graficznych oferuje wszystkie te możliwości.

Podsumowanie

W rozdziale omówiono podstawowe algorytmy obcinania i rasteryzacji, które wchodzą w skład pakietów grafiki rastrowej. Pokazaliśmy jedynie podstawy; w bardziej zaawansowanych implementacjach muszą być uwzględnione bardziej wyszukane i specjalne przypadki. Dalsze szczegóły można znaleźć w rozdziałach 14, 17 i 19 w pracy [FOLE90].

Najważniejszą ideą tego rozdziału jest to, że – ponieważ dla interakcyjnej grafiki rastrowej istotna jest szybkość – zazwyczaj najlepsze są przyrostowe algorytmy rasteryzacji wykorzystujące w swoich wewnętrznych pętlach tylko operacje całkowitoliczbowe. Podstawowe algorytmy można rozszerzyć tak, żeby uwzględnić grubość oraz wzory dla krawędzi i dla wypełniania obszarów. Podczas gdy podstawowe algorytmy rasteryzacji prymitywów o szerokości 1 piksela dążą do minimalizacji błędu między wybranym pikselem na siatce kartezjańskiej a idealnym prymitywem zdefiniowanym na płaszczyźnie, to algorytmy dla pogrubionych prymitywów mogą być tworzone na zasadzie kompromisu między jakością i poprawnością a szybkością. Chociaż dzisiaj większość grafiki rastrowej 2D wciąż operuje, nawet na monitorach kolorowych, prymitywami z jednym bitem na piksel, spodziewamy się, że wkrótce rozpowszechnią się metody z usuwaniem zakłóceń działające w czasie rzeczywistym.

Zadania

3.1. Napisz specjalny fragment kodu dla rasteryzacji odcinków poziomych oraz pionowych i odcinków o nachyleniach ± 1.

3.2. Zmodyfikuj algorytm z punktem środkowym rasteryzacji odcinka (program 3.2) tak, żeby działał dla odcinków o dowolnym nachyleniu.

3.3. Pokaż, dlaczego dla algorytmu rasteryzacji odcinka z punktem środkowym błąd punkt-odcinek jest zawsze $\leqslant 1/2$.

3.4. Zmodyfikuj algorytm rasteryzacji odcinków z punktem środkowym z zadania 3.2 tak, żeby uwzględniał kolejność punktów końcowych i przecięcia z krawędziami obcinającymi, jak to omówiono w p. 3.2.3.

3.5. Zmodyfikuj algorytm z punktem środkowym rasteryzacji odcinka (zadanie 3.2) tak, żeby jasność zapisywanych pikseli zmieniała się w funkcji nachylenia odcinka.

3.6. Zmodyfikuj algorytm z punktem środkowym rasteryzacji odcinków (zadanie 3.2) tak, żeby punkty końcowe odcinków nie musiały mieć współrzędnych całkowitych – jest to łatwiejsze, jeżeli w całym algorytmie będą używane wielkości zmiennopozycyjne. Jako trudniejsze zadanie rozważ możliwość uwzględniania odcinków o wymiernych współrzędnych końców, korzystając tylko z liczb całkowitych.

3.7. Pokaż, że łamane mogą mieć wspólne nie tylko punkty wierzchołkowe. Opracuj algorytm, który umożliwia uniknięcie dwukrotnego zapisywania pikseli. *Wskazówka*: Weź pod uwagę rasteryzację i zapis do kanwy w trybie **xor** jako niezależne fazy.

3.8. Opracuj metodę rasteryzacji okręgu alternatywną dla algorytmu z punktem środkowym omówionego w p. 3.3.2, wykorzystując aproksymację okręgu za pomocą łamanej.

3.9. Opracuj algorytm rasteryzacji nie wypełnionego prostokąta z zaokrąglonymi rogami w postaci ćwiartek okręgu o określonym promieniu.

3.10. Napisz funkcję rasteryzacji prostokąta o bokach równoległych do boków ekranu i wypełnionego stałą barwą, umieszczanego w dowolnym miejscu ekranu, który efektywnie zapisuje dwupoziomową pamięć obrazu; chodzi o równoczesne zapisywanie całego słowa pikseli.

3.11. Skonstruuj przykłady pikseli, które są gubione albo zapisywane wiele razy, korzystając z reguł z p. 3.5. Opracuj inne, być może, bardziej złożone reguły, które nie rysują dwukrotnie pikseli znajdujących się na wspólnych krawędziach, bez możliwości gubienia pikseli. Czy te reguły są warte zwiększonego nakładu pracy?

3.12. Zaimplementuj pseudokod z p. 3.5 do rasteryzacji wielokąta, biorąc pod uwagę przy zapamiętywaniu segmentów wystąpienie wielokątów typu drzazga.

3.13. Opracuj algorytmy rasteryzacji dla trójkątów i trapezów, wykorzystujące prostotę ich kształtów. Takie algorytmy są popularne w rozwiązaniach sprzętowych.

3.14. Zbadaj algorytmy triangulacji dla dekompozycji dowolnego, być może niewypukłego albo samoprzecinającego się wielokąta, na siatkę trójkątów o wspólnych wierzchołkach. Czy ułatwi coś ograniczenie klasy wielokątów do niewypukłych wielokątów bez samoprzecinania i wewnętrznych dziur [PREP85]?

3.15. Rozszerz algorytm rasteryzacji okręgów z punktem środkowym (program 3.4) tak, żeby działał dla wypełnionych okręgów i wycinków kołowych (klinów dla wykresów kołowych); wykorzystaj tablice segmentów.

3.16. Zaimplementuj algorytmy wypełniania wielokątów wzorem dla przypadków zaczepiania wzorów bezwzględnego i względnego, omówionych w p. 3.7; porównaj algorytmy ze względu na efekty wizualne i efektywność obliczeniową.

3.17. Wykorzystaj metodę z rys. 3.20 do zapisywania znaków wypełnianych wzorem w trybie nieprzezroczystym. Pokaż, jak funkcja copyPixel z maską zapisu może być wykorzystana w tej klasie problemów.

3.18. Zaproponuj metodę rysowania różnych symboli, np. ikon kursora, reprezentowanych przez małe mapy bitowe tak, żeby mogły być widoczne niezależnie od tła, na którym są rysowane. *Wskazówka*: Dla każdego symbolu zdefiniuj maskę, która „otacza" symbol – to znaczy, pokrywa więcej pikseli niż zawiera symbol – i która rysuje maski i symbole w niezależnych przebiegach.

3.19. Zaimplementuj algorytm rysowania pogrubionych odcinków korzystając z metod wymienionych w p. 3.7. Porównaj ich efektywność i jakość tworzonych obrazów.

3.20. Rozszerz algorytm rasteryzacji okręgu z punktem środkowym (program 3.4) tak, żeby było możliwe rysowanie pogrubionych okręgów.

3.21. Zaimplementuj algorytm rysowania pogrubionych linii tak, żeby możliwe było uwzględnianie stylu linii oraz stylu pióra i wzoru.

3.22. Zmodyfikuj algorytm Cohena-Sutherlanda obcinania odcinka (program 3.7) tak, żeby unikać ponownego obliczania nachyleń w kolejnych przebiegach. Ponadto przedefiniuj strukturę *outcode* tak, żeby był to związek zmiennej **unsigned int** *all* i czterech jednobitowych znaczników *left*, *right*, *bottom* i *top*.

3.23. Rozważ wypukły wielokąt o n wierzchołkach obcinany przez prostokąt obcinający. Jaka jest maksymalna liczba wierzchołków obciętego wielokąta? Jaka jest minimalna liczba takich wierzchołków? Rozważ ten sam problem dla niewypukłego wielokąta. Ile może powstać wielokątów? Jeżeli powstanie jeden wielokąt, to jaką największą liczbę wierzchołków może mieć?

3.24. Wyjaśnij, dlaczego algorytm Sutherlanda-Hodgmana obcinania wielokątów działa tylko dla wypukłych obszarów obcinających.

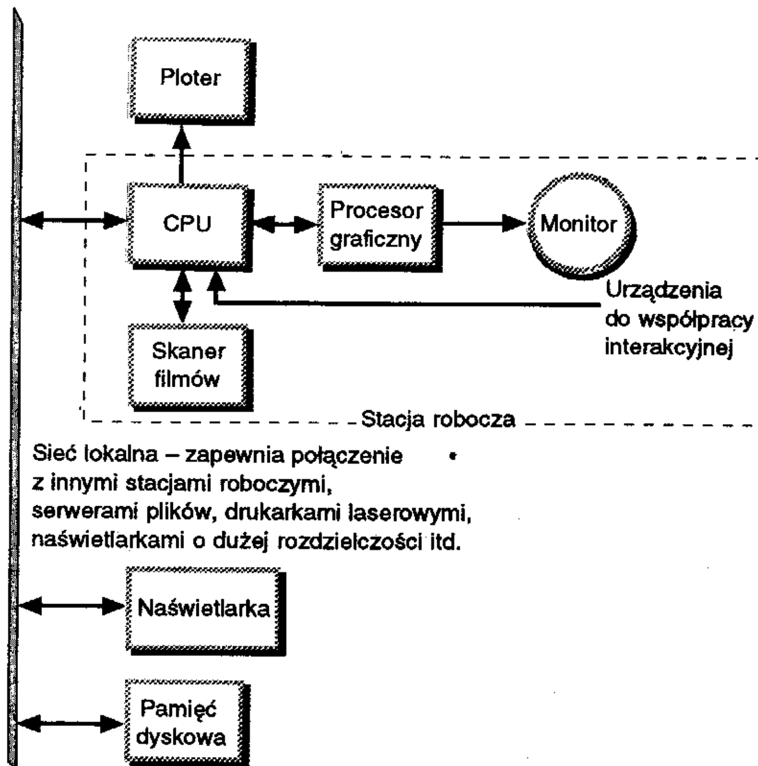
3.25. Zaproponuj strategię podziału piksela i zliczania pokrytych (przynajmniej w znacznym stopniu) podpikseli przez odcinek, w celu wykorzystania w algorytmie rysowania odcinka metodą bezwagowego próbkowania powierzchni.

4. Sprzęt dla potrzeb grafiki komputerowej

W rozdziale opisano, jak działają ważne elementy sprzętowe systemu wyświetlania grafiki komputerowej. W punkcie 4.1 omówiono urządzenia do wykonywania trwałych kopii: drukarki, plotery, drukarki laserowe, drukarki atramentowe i naświetlarki. Krótko są opisane podstawowe koncepcje technologiczne wykorzystywane w poszczególnych urządzeniach, a w zakończeniu porównano różne urządzenia. W punkcie 4.2, poświęconym urządzeniom wyświetlającym, omówiono monitory monochromatyczne i kolorowe, ekrany ciekłokrystaliczne (LCD) i elektroluminescencyjne. Na końcu podano zalety i wady różnych technologii wyświetlania.

Systemy wyświetlania rastrowego, które mogą wykorzystywać każdą z omawianych tu technologii wyświetlania, opisano w p. 4.3. Najpierw wprowadzono prosty system rastrowy, a potem rozbudowano go ze względu na funkcjonalność graficzną i integrację procesorów rastrowych i uniwersalnych w przestrzeni adresowej systemu. W punkcie 4.4 opisano rolę tablicy pośredniej i sterownika wyświetlaniem w monitorze, sterowanie barwą i mieszanie obrazów. W punkcie 4.5 omówiono urządzenia umożliwiające interakcyjną współpracę z użytkownikiem, np. tabliczki, myszkę, ekrany dotykowe itd. Położono nacisk raczej na koncepcje działania i wykorzystywania niż na szczegóły technologiczne. W punkcie 4.6 krótko omówiono urządzenia do wprowadzania obrazów, np. skanery, za pomocą których można istniejący obraz wprowadzić do komputera.

Na rysunku 4.1 pokazano powiązanie różnych urządzeń ze sobą. Kluczowym elementem jest zintegrowany układ procesora CPU i procesora wyświetlania określany jako *stacja graficzna*; na ogół zawiera ona



Rys. 4.1. Elementy typowego interakcyjnego systemu graficznego

CPU zdolne do wykonywania 20-100 milionów instrukcji na sekundę (MIPS) i monitor o rozdzielczości przynajmniej 1000 × 800. Lokalna sieć łączy wiele stacji i umożliwia wspólne wykorzystywanie plików, korzystanie z poczty elektronicznej i dostęp do wspólnych urządzeń zewnętrznych, np. dobrej jakości naświetlarki filmów, duże dyski, bramy do innych sieci i komputery dużej mocy.

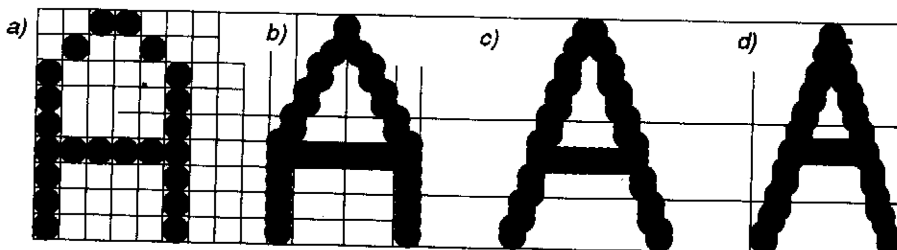
4.1. Metody tworzenia kopii trwałych

W tym punkcie omówimy różne metody tworzenia kopii trwałych, a potem podsumujemy ich właściwości. Na początku musimy jednak zdefiniować kilka istotnych pojęć.

Jakość obrazu uzyskiwana za pomocą urządzeń wyświetlających zależy zarówno od rozdzielczości urządzenia, jak i od wielkości plamki. *Wielkość plamki* jest to średnica jednej plamki tworzonej przez urządzenie. *Adresowalność* jest to liczba plamek, które może utworzyć urządzenie na odcinku 1 cala; adresowalności w poziomie i w pionie mogą się

różnić. Adresowalność w kierunku osi x to po prostu odwrotność odległości między środkami punktów o adresach (x, y) i $(x + 1, y)$; adresowalność w kierunku osi y jest definiowana podobnie. *Odległość między plamkami* jest odwrotnością adresowalności.

Zazwyczaj dobrze jest, żeby wielkość plamki była trochę większa niż odległość między plamkami; umożliwia to tworzenie gładkich kształtów. Wyjaśnia to rys. 4.2. Oczywiście są potrzebne kompromisy: za pomocą plamki kilka razy większej niż odległość między plamkami można drukować bardzo gładkie kształty, natomiast za pomocą małej plamki można pokazać drobniejsze elementy.



Rys. 4.2. Wpływ stosunku wielkości plamki do odstępu między plamkami: a) odstęp między plamkami równy wielkości plamki; b) odstęp między plamkami równy połowie wielkości plamki; c) odstęp między plamkami równy jednej trzeciej wielkości plamki; d) odstęp między plamkami równy jednej czwartej wielkości plamki

Rozdzielczość, która jest związana z wielkością plamki i nie może być większa niż adresowalność, jest to liczba rozróżnialnych linii na cal, które urządzenie może utworzyć. Rozdzielczość jest definiowana jako najmniejsza odległość, przy której sąsiednie linie czarna i biała mogą być rozróżnione przez obserwatora (to znowu oznacza, że rozdzielczości pionowa i pozioma mogą się różnić). Jeżeli na odcinku 1 cala można rozróżnić 40 czarnych linii przeplecionych z 40 białymi liniami, to rozdzielczość wynosi 80 linii na cal (mówi się również o 40 parach linii na cal). Rozdzielczość zależy również od rozkładu jasności wzdłuż przekroju plamki. Plamka o ostro zarysowanym brzegu daje większą rozdzielczość niż plamka z rozmytym brzegiem.

Wiele z omawianych urządzeń może tworzyć jedynie kilka barw w dowolnym punkcie. Inne barwy można otrzymać metodą mikrowzorów, opisaną w rozdz. 11, kosztem zmniejszenia rozdzielczości przestrzennej wynikowego obrazu.

W *drukarkach matrycowych* są używane głowice drukujące z 7...24 *iglami* (cienkimi sztyftami), z których każda może indywidualnie być pobudzona i uderzyć w papier przez taśmę. Głowica drukująca porusza się wzdłuż papieru skokowo, papier jest wysuwany o wiersz i głowica ponownie przesuwa się wzdłuż papieru. Ponieważ takie drukarki są

urządzeniami rastrowymi, przed drukowaniem jest potrzebna konwersja obrazów wektorowych.

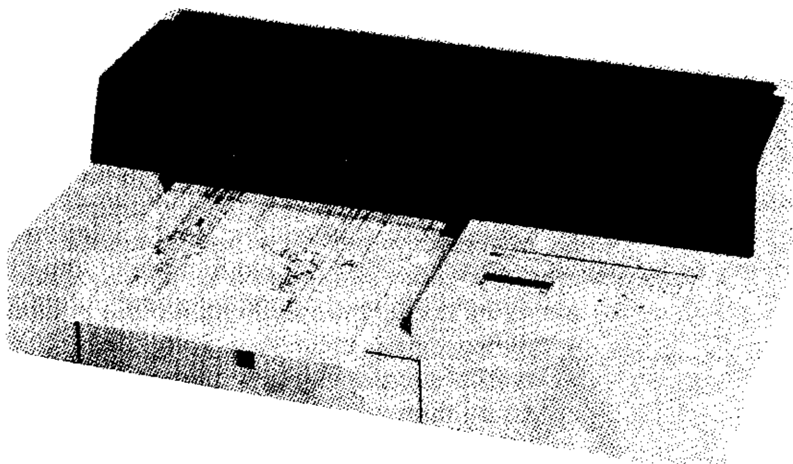
Używając taśm barwnych można uzyskać barwną kopię. Możliwe są dwa rozwiązania. W pierwszym wykorzystuje się kilka głowic drukujących, przy czym każda głowica ma przypisaną taśmę o innej barwie. Częściej jest używane rozwiązanie z jedną głowicą i z wielobarwną taśmą. Jeżeli dla danej plamki uderzy się igłą w dwie różne barwy, to można utworzyć więcej barw niż ma taśma. Barwa będąca na wierzchu może być nieco silniejsza od tej pod spodem. Mając do dyspozycji trzy barwy na taśmie można w tym samym punkcie uzyskać do ośmiu barw – typowo cyjan, magenta i żółty. Jednak barwa czarna uzyskiwana w wyniku uderzenia igłą w trzy różne barwy jest dość słaba, dlatego często do taśmy dodaje się barwę czarną.

W ploterze piórowym pióro porusza się nad papierem w swobodny sposób, jak przy rysowaniu wektorów. W celu narysowania odcinka pióro jest umieszczane nad miejscem, gdzie ma być początek odcinka, potem jest opuszczane tak, żeby dotykało papieru, i następnie przesuwane po linii prostej w kierunku końca odcinka, gdzie jest podnoszone i przesuwane do punktu początkowego nowego odcinka.

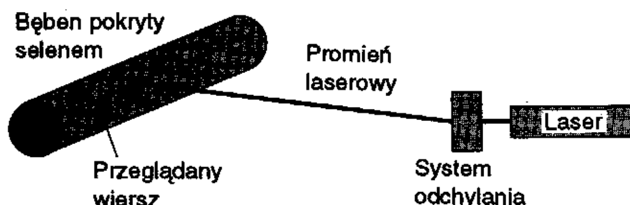
Są dwie odmiany ploterów piórowych. W *ploterze płaskim* pióro jest poruszane w kierunkach x i y nad kartką papieru rozłożoną na płaskiej powierzchni; przyleganie papieru do podłoża jest zapewniane metodą ładunku elektrostatycznego, metodą próżniową albo po prostu na skutek dokładnego rozłożenia. Karetka porusza się wzdłuż jednej osi nad papierem, a pióro porusza się wzdłuż karetki, a więc w kierunku prostopadłym do kierunku ruchu karetki. Pióro może być podnoszone i opuszczane. Powierzchnie dostępne do rysowania mają wymiary od 30 × 45 cm do 1,8 × 3 m i więcej.

W *ploterach bębnowych* papier przesuwa się wzdłuż jednej osi, a pióro wzdłuż drugiej. Papier dokładnie przylega do bębna. Wypusty na bębnie współpracują z dziurkami na brzegach papieru, dzięki czemu papier nie ślizga się po powierzchni bębna. Bęben może poruszać się tam i z powrotem. W wielu *ploterach stołowych* papier jest przesuwany tam i z powrotem między rolkami dociskowymi, natomiast pióro porusza się w poprzek papieru (rys. 4.3).

W *drukarkach laserowych* promień laserowy jest przesuwany wzdłuż dodatnio naładowanego bębna obrotowego pokrytego selenem. Obszary trafione przez promień tracą swój ładunek; ładunek dodatni pozostaje tylko tam, gdzie kopia ma być czarna. Ujemnie naładowany toner przylega do dodatnio naładowanych obszarów bębna i jest potem przenoszony na papier, co w efekcie tworzy kopię. W kserografii barwnej ten proces jest powtarzany trzy razy, raz dla każdej barwy podstawowej. Uproszczony schemat drukarki laserowej pokazano na



Rys. 4.3. Ploter stołowy (Za zgodą firmy Hewlett-Packard Company)



Rys. 4.4. Organizacja drukarki laserowej (mechanizm dostarczania tonera i podajnik papieru nie są pokazane)

rys. 4.4. W każdym punkcie bębna albo jest ładunek dodatni, albo go nie ma i odpowiednio na kopii uzyskujemy czarne plamki. Dlatego drukarka laserowa jest dwupoziomowym urządzeniem monochromatycznym albo urządzeniem kolorowym z ośmioma barwami.

W drukarce laserowej jest mikroprocesor, który dokonuje rasteryzacji i steruje drukarką. Coraz większa liczba drukarek laserowych akceptuje opis PostScriptowy, a więc w języku opisu dokumentu i obrazu, będącym standardem de facto [ADOB85]. PostScript daje proceduralny opis obrazu, który ma być wydrukowany i może być również wykorzystywany do pamiętania opisu obrazu. Większość drukarek laserowych korzysta z papieru 8,5 × 11 albo 8,5 × 14 cali; są również dostępne drukarki laserowe ze znacznie szerszym papierem (30 cali) dla tworzenia rysunków inżynierskich albo map.

Drukarki atramentowe natryskują na papier atramenty o barwach cyjan, magenta i żółta, a czasami czarna. W większości przypadków wyrzutnie atramentu są montowane na głowicy przypominającej głowicę drukarki igłowej. Głowica porusza się w poprzek papieru rysując jedną linię; w czasie powrotu głowicy papier jest wysuwany o jeden odstęp między liniami i jest rysowana następna linia. Jeżeli papier prze-

suwa się trochę więcej albo trochę mniej, to mogą powstać pewne nieregularności w odstępach między liniami. Wszystkie barwniki są osadzane jednocześnie; jest to inaczej niż w wieloprzebiegowych ploterach laserowych albo w drukarkach. Większość drukarek atramentowych jest ograniczona do dwupoziomowego sterowania każdym pikselem; nieliczne mają zmienną wielkość plamki.

Drukarki termiczne to inne urządzenia rastrowe do wykonywania kopii trwałych; wykorzystuje się w nich dokładnie rozmieszczone podgrzewające ostrza (typowo 200 na cal) do przenoszenia pigmentu z barwnego woskowego papieru na czysty papier. Papiery woskowy i czysty są przeciągane razem nad taśmą z selektywnie podgrzewanymi igłami; w efekcie uzyskuje się przenoszenie pigmentu. W celu drukowania barwnego (jest to najczęściej stosowana technologia) papier woskowy jest na rolce z umieszczonymi kolejno taśmami o barwach cyjan, magenta, żółta i czarna, każda o długości równej długości papieru. Ponieważ igły nagrzewają się i stygną bardzo szybko, jednobarwna kopia obrazu może powstać w czasie krótszym niż 1 min. Niektóre drukarki termiczne akceptują sygnały wizyjne i cyfrowe mapy bitowe, dzięki czemu są wygodne do tworzenia kopii obrazów wizyjnych.

Drukarki termiczne przenoszące barwnik na zasadzie *sublimacji* działają podobnie jak zwykłe drukarki termiczne, z tą różnicą, że proces ogrzewania i przenoszenia farby umożliwia przeniesienie 256 intensywności każdej z barw cyjan, magenta i żółta, dzięki czemu uzyskuje się dobrej jakości pełnobarwne obrazy o rozdzielczości przestrzennej 200 plamek na cal. Proces jest wolniejszy niż przy przenoszeniu wosku, ale jakość jest niemal fotograficzna – dzięki temu drukarki te mogą być wykorzystywane do tworzenia próbnych pełnobarwnych obrazów.

Innym urządzeniem do tworzenia kopii trwałych może być *aparat fotograficzny* wykonujący zdjęcia obrazu wyświetlonego na ekranie monitora. Jest to najpopularniejsza z omawianych przez nas technologii, która zapewnia otrzymywanie kopii wielobarwnych.

Są dwie podstawowe metody stosowane w naświetlarkach – rejestratorach filmów barwnych. W jednej metodzie aparat rejestruje barwny obraz bezpośrednio z ekranu monitora kolorowego. Rozdzielczość obrazu jest ograniczona ze względu na maskę monitora kolorowego (p. 4.2) i konieczność rasteryzacji na monitorze kolorowym. W drugim podejściu czarno-biały ekran monitora jest fotografowany przez filtry kolorowe i sekwencyjnie są wyświetlane różne składowe barwy obrazu. Ta metoda daje obrazy rastrowe i wektorowe o bardzo dobrej jakości. Barwy uzyskuje się dzięki kolejnemu naświetlaniu obrazu przez dwa lub więcej filtrów, zazwyczaj przy różnych jasnościach lampy monitora.

Wejściem do naświetlarki może być rastrowy sygnał wizyjny, mapa bitowa albo instrukcje typu wektorowego. Monitor kolorowy może być

sterowany bezpośrednio przez sygnał wizyjny albo składowe sygnału *R*, *G* i *B* mogą być elektronicznie rozdzielone i wyświetlone kolejno z użyciem odpowiednich filtrów. W każdym przypadku sygnał wizyjny musi być stały w czasie całego procesu rejestracji, co może trwać nawet minutę, jeżeli korzysta się z filmu o małej czułości.

W tablicy 4.1 są zebrane różnice między urządzeniami wytwarzającymi barwne kopie trwałe. Wiele szczegółów na temat metod wykorzystywanych w takich urządzeniach można znaleźć w pracy [DURB88]. Szybkość pojawiania się nowości technologicznych jest obecnie tak duża, że względne zalety i wady niektórych z tych urządzeń z pewnością się zmienią. Ceny urządzeń również wahają się w dużym przedziale. Na przykład naświetlarki i plotery piórowe mogą kosztować od 500 do 100 000$.

Tablica 4.1. Porównanie urządzeń do tworzenia kopii trwałych*)

Parametr	Ploter piórowy	Drukarka matrycowa	Drukarka laserowa	Drukarka atramentowa	Fotografia
Liczba barw na plamkę	do 16	8	8	8–wiele	wiele
Adresowalność, punktów na cal	1000+	do 250	do 1500	do 200	do 800
Wielkość plamki, tysięczne cala	15–6	18–10	5	20–8	20–6
Zakres względnych kosztów	L–M	VL	M–H	L–M	M–H
Względny koszt/obraz	L	VL	M	L	H
Jakość obrazu	L–M	L	H	M	M–H
Szybkość	L	L–M	M	M	L

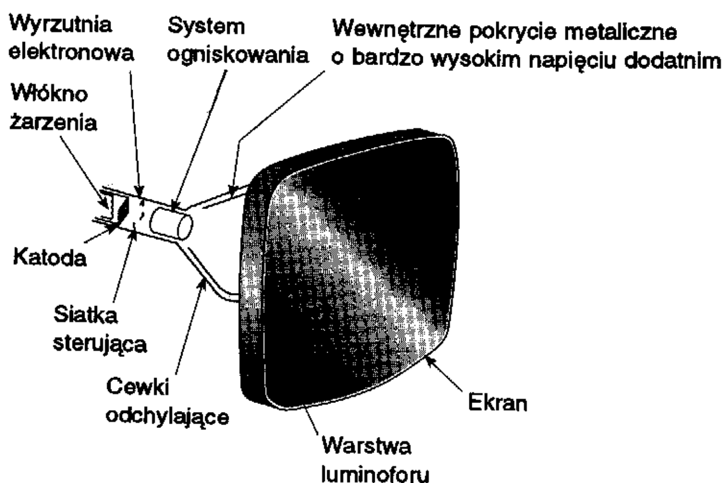
*) VL – bardzo mały; L – mały; M – średni; H – wysoki.

Zauważmy, że ze wszystkich urządzeń kolorowych tylko naświetlarki, drukarki z termiczną sublimacją barwnika i niektóre drukarki atramentowe umożliwiają reprodukcję szerokiej gamy barw. We wszystkich innych technologiach w zasadzie wykorzystuje się dwupoziomowe sterowanie dla trzech albo czterech barw, które mogą być rejestrowane bezpośrednio. Zauważmy również, że sterowanie barwą jest złożone; nie ma gwarancji, że osiem barw na jednym urządzeniu będzie wyglądało tak samo jak osiem barw na monitorze albo na innym urządzeniu tworzącym trwałe kopie. W punkcie 13.4 książki [FOLE90] można znaleźć omówienie trudności związanych z reprodukcją barwy.

4.2. Metody wyświetlania

Interakcyjna grafika komputerowa wymaga urządzeń wyświetlających, na których obrazy mogą się zmieniać szybko. Monitory umożliwiają zmianę obrazów, dzięki czemu jest możliwe uzyskanie dynamicznego ruchu części obrazu. Monitory CRT są bez wątpienia najpopularniejszymi urządzeniami wyświetlającymi i ta sytuacja jeszcze się długo nie zmieni. Niemniej są opracowywane technologie półprzewodnikowe, które z czasem mogą istotnie ograniczyć dominację monitorów CRT.

Monitory monochromatyczne, wykorzystywane w zastosowaniach graficznych, to w zasadzie takie same urządzenia jak domowe odbiorniki telewizji czarno-białej. Na rysunku 4.5 pokazano wysoce stylizowany przekrój lampy CRT. Wyrzutnia elektronowa emituje strumień elektronów, które są przyspieszane w kierunku ekranu pokrytego luminoforem, przez wysokie napięcie dodatnie wytworzone w pobliżu ekranu.



Rys. 4.5. Przekrój lampy CRT (skala nie jest zachowana)

Na drodze do ekranu wiązka jest formowana przez układ ogniskowania i jest kierowana przez pole magnetyczne wytwarzane przez cewki odchylające do określonego punktu ekranu. Gdy elektrony uderzają w ekran, luminofor emituje światło widzialne. Ponieważ światło emitowane przez luminofor zanika wykładniczo w funkcji czasu, cały obraz musi być *odświeżany* (ponownie rysowany) wiele razy na sekundę, tak żeby obserwator widział stały, nie migający obraz.

Częstotliwość odświeżania w monitorach rastrowych nie zależy od złożoności obrazu. W systemach wektorowych częstotliwość odświeżania zależy bezpośrednio od złożoności obrazu (liczby odcinków,

punktów i znaków): im większa jest złożoność, tym dłuższy jest czas jednego cyklu odświeżania i mniejsza częstotliwość odświeżania.

Strumień elektronów z podgrzanej katody jest przyspieszany w kierunku luminoforu przez wysokie napięcie, typowo 15 000 do 20 000 V, które określa szybkość uzyskaną przez elektrony tuż przed uderzeniem w luminofor. Napięcie siatki sterującej określa liczba elektronów w strumieniu elektronów. Im bardziej ujemne jest napięcie siatki sterującej, tym mniej elektronów przechodzi przez siatkę. Dzięki temu można sterować jasnością plamki – ilość światła emitowanego przez luminofor maleje wraz z maleniem liczby elektronów w strumieniu.

System ogniskowania koncentruje strumień elektronów tak, żeby zbiegał się w małym punkcie w chwili uderzania w warstwę luminoforu. Nie wystarczy, żeby elektrony w wiązce poruszały się równolegle jeden do drugiego. Ze względu na odpychanie elektrony rozbiegałyby się – system ogniskowania musi temu zapobiec. Z wyjątkiem tej tendencji do rozbiegania się, ogniskowanie wiązki elektronów jest analogiczne do ogniskowania strumienia światła.

Gdy strumień elektronów uderza w ekran lampy CRT pokryty luminoforem, wówczas poszczególne elektrony poruszają się z energią kinetyczną proporcjonalną do napięcia przyspieszającego. Część tej energii jest tracona w postaci ciepła, reszta natomiast jest przekazywana elektronom atomów luminoforu, co powoduje przeskakiwanie elektronów na wyższe poziomy energetyczne. Przy powrocie do swoich poprzednich poziomów energetycznych pobudzone elektrony oddają nadmiar energii w postaci światła o częstotliwościach (tj. barwach) wynikających z teorii kwantowej. Każdy rodzaj luminoforu ma kilka różnych poziomów kwantowych, do których mogą zostać elektrony pobudzone. Elektrony na niektórych poziomach są mniej stabilne i wracają do stanu podstawowego szybciej niż inne. *Fluorescencja* luminoforu, to światło emitowane wówczas, gdy te bardzo niestabilne elektrony tracą swoją nadmiarową energię, gdy luminofor jest uderzany przez elektrony. *Fosforescencja* jest to światło emitowane przy powrocie względnie bardziej stabilnych pobudzonych elektronów do ich stanu niepobudzonego, już po usunięciu pobudzenia przez wiązkę elektronów. Dla typowych luminoforów większość światła jest emitowana na zasadzie fosforescencji, ponieważ pobudzenie i fluorescencja zazwyczaj trwają tylko ułamek mikrosekundy. *Poświata* luminoforu jest definiowana jako czas od usunięcia pobudzenia do chwili, kiedy fosforescencja spadnie do 10% początkowej emisji światła. Wartość poświaty dla różnych luminoforów może sięgać wielu sekund, ale dla większości luminoforów wykorzystywanych w urządzeniach graficznych wynosi zazwyczaj 10 do 60 μs. Zmniejszenie ilości emitowanego światła jest wykładnicze w funkcji czasu. Charakterystyki luminoforów omówiono w pracy [SHER93].

Częstotliwość odświeżania monitora CRT jest to liczba określająca, ile razy jest pobudzony ekran w ciągu sekundy; w monitorach rastrowych typowa wartość wynosi 60 Hz. Jeżeli częstotliwość odświeżania maleje, to pojawia się *migotanie*, ponieważ oko przestaje całkować impulsy światła przychodzące od piksela. Częstotliwość odświeżania, powyżej której obraz przestaje migotać i zaczyna sprawiać wrażenie obrazu stałego, jest określana jako *częstotliwość krytyczna* zaniku migotania. Efekt ten jest nam wszystkim dobrze znany; występuje w czasie oglądania telewizji albo w kinie. Obraz pozbawiony migotania sprawia na obserwatorze wrażenie obrazu stałego, nawet gdy w rzeczywistości każdy punkt jest wyłączony znacznie dłużej niż jest włączony.

Jednym z czynników mających wpływ na częstotliwość krytyczną jest poświata luminoforu: im ta poświata jest dłuższa, tym mniejsza jest wartość częstotliwości krytycznej. Relacja między częstotliwością krytyczną a poświatą jest nieliniowa: podwojenie poświaty nie powoduje zmniejszenia częstotliwości krytycznej o połowę. Jeżeli poświata wzrasta do wartości kilku sekund, to częstotliwość krytyczna staje się mała. Z drugiej strony może zostać użyty nawet luminofor, który w ogóle nie ma poświaty, ponieważ, to czego oko w rzeczywistości wymaga, to to żeby docierała do niego pewna ilość światła przez krótki czas, w sposób powtarzalny z częstotliwością powyżej częstotliwości krytycznej.

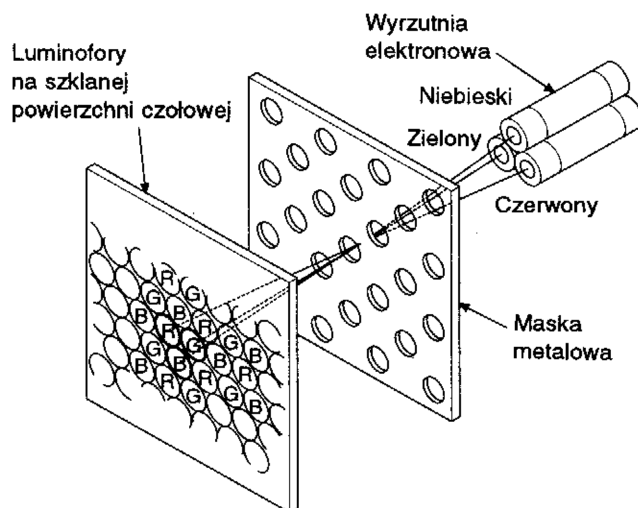
Szybkość przeglądania poziomego jest to liczba linii w ciągu sekundy, które mogą wyświetlić układy sterujące monitora. Ta szybkość jest w przybliżeniu równa iloczynowi częstotliwości odświeżania i liczby linii. Dla danej szybkości przeglądania wzrost częstotliwości odświeżania oznacza zmalenie liczby wyświetlanych linii.

Pasmo monitora ma związek z szybkością, z jaką wyrzutnia elektronowa może być włączana i wyłączana. W celu uzyskania rozdzielczości poziomej n pikseli w linii, musi być możliwe włączenie wyrzutni elektronowej przynajmniej $n/2$ razy i wyłączenie jej również $n/2$ razy w ramach jednej linii po to, żeby utworzyć na przemian linie włączone i wyłączone. Rozważmy raster o 1000 liniach po 1000 pikseli wyświetlanych z częstotliwością odświeżania 60 Hz. Z prostego obliczenia wynika, że czas potrzebny na narysowanie piksela jest równy odwrotności liczby pikseli (1000 pikseli/linię × 1000 linii/ramkę × 60 ramek/s) wynosi ok. 16 ns. Dokładniej biorąc, ze względu na pewien narzut związany z odświeżaniem pionowym i poziomym, jeden piksel jest rysowany w czasie ok. 11 ns [WHIT84]. Wobec tego okres cyklu włączania/wyłączania wynosi ok. 22 ns, co odpowiada częstotliwości 45 MHz. Ta częstotliwość to minimalne pasmo potrzebne do uzyskania rozdzielczości 1000 linii (500 par linii); nie jest to rzeczywiste pasmo, ponieważ pominęliśmy wpływ wielkości plamki. Niezerowa wielkość plamki musi być skompensowana przez większe pasmo, co powoduje, że strumień musi się

szybciej przełączać między stanami włączenia i wyłączenia, co daje ostrzejsze krawędzie plamki. Nie jest niczym niezwykłym rzeczywiste pasmo 100 MHz w monitorze 1000 × 1000.

W odbiornikach telewizyjnych i w monitorach kolorowych są wykorzystywane elektronowe lampy promieniowe CRT z *maską*. Wewnętrzna powierzchnia ekranu lampy jest pokryta ciasno upakowanymi grupami plamek luminoforów *R*, *G* i *B*. Te grupy plamek są tak małe, że światło emitowane przez poszczególne plamki jest odbierane przez obserwatora jako mieszanina trzech barw. Każda grupa może wytworzyć wiele różnych barw zależnie od tego, jak silnie zostaną pobudzone poszczególne plamki luminoforu. Maska jest to cienka metalowa płyta z wieloma małymi otworami zamontowana blisko ekranu; jest ona tak wykonana, żeby każdy z trzech strumieni elektronów (po jednym dla *R*, *G* i *B*) mógł uderzyć tylko w plamkę jednego rodzaju. Dlatego plamki mogą być wybierane selektywnie.

Na rysunku 4.6 pokazano jeden z najpopularniejszych typów masek CRT, a mianowicie maskę typu *delta-delta*. Plamki luminoforu są tak rozmieszczone, że tworzą trójkątne wzory – triady; podobnie są rozmieszczone wyrzutnie elektronowe. Strumienie w poszczególnych wyrzutniach są odchylane razem, tak żeby skupiały się w tym samym punkcie ekranu. Maska ma jeden mały otwór dla każdej triady. Otwory są dokładnie rozmieszczone w stosunku do triad i wyrzutni elektronowych, tak żeby każda plamka w triadzie była eksponowana na elektrony tylko z jednej wyrzutni. W kineskopach dużej precyzji typu delta-



Rys. 4.6. Maska typu delta-delta. Trzy wyrzutnie i plamki luminoforu są uporządkowane w trójkątny wzór (delta). Maska pozwala elektronom z każdej wyrzutni uderzyć tylko w odpowiednie plamki luminoforu

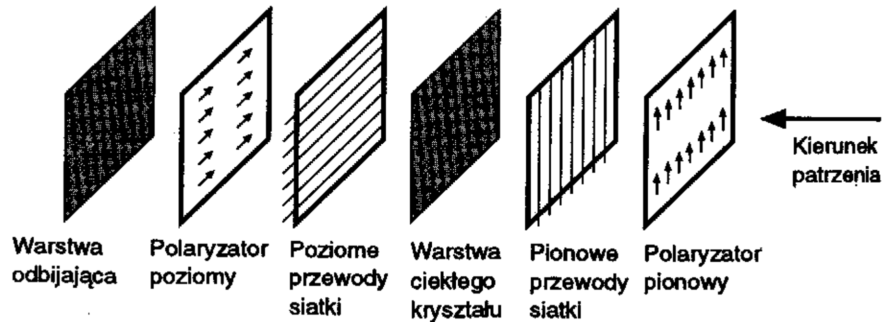
-delta jest szczególnie trudno utrzymać precyzję wzajemnego rozmieszczenia. Inne rozwiązanie jest stosowane w kineskopach typu *delta PIL* (precision in-line), w których jest łatwiej zapewnić zbieżność i które są łatwiejsze w produkcji; ta technologia jest wykorzystywana w monitorach dużej precyzji (1000 linii). Ponieważ jednak rozwiązanie typu delta-delta zapewnia większą rozdzielczość, może znowu stać się dominującą technologią dla telewizji o podwyższonej rozdzielczości (HDTV). W badaniach laboratoryjnych, z dużymi szansami na wejście na rynek, jest płaski kolorowy kineskop CRT, w którym strumień elektronów porusza się równolegle do ekranu i którego tor jest zakrzywiany pod kątem 90° przed uderzeniem w powierzchnię.

Konieczność istnienia maski i triad ogranicza rozdzielczość kolorowego kineskopu; nie ma tego w kineskopach monochromatycznych. W lampach o bardzo dużej rozdzielczości triady są rozmieszczone w centrach oddalonych o 0,21 mm; w kineskopach telewizorów domowych te odległości wynoszą ok. 0,60 mm (ten odstęp jest określany jako tak zwana *podziałka* lampy). Ponieważ dokładnie zogniskowany strumień nie daje gwarancji idealnego trafienia w środek otworu maski, średnica strumienia (zdefiniowana jako średnica, dla której natężenie wynosi 50% maksimum) musi wynosić ok. 7/4 wartości podziałki. Wobec tego dla maski o odstępie między triadami 0,25 mm (0,01 cala) strumień ma szerokość 0,018 cala i rozdzielczość nie może być większa niż ok. 1/0,018 = 55 linii na cal. Dla odstępu 0,25 i monitora o przekątnej 19 cali (szerokość ok. 15,5 cala i wysokość ok. 11,6 cala) [CONR85] osiągalna rozdzielczość wynosi tylko 15,5 × 55 = 850 na 11,6 × 55 = = 638. Ta wartość zgadza się z typową adresowalnością 1280 × 1024 albo 1024 × 800. Jak pokazano na rys. 4.2, dobrze jest, jeżeli rozdzielczość jest trochę mniejsza niż adresowalność.

Większość dobrej jakości lamp CRT z maską ma przekątne od 15 do 21 cali i nieco zakrzywioną powierzchnię ekranu, co stwarza optyczne zakłócenie dla obserwatora. Obecnie stają się dostępne różnego rodzaju lampy CRT z płaskim ekranem.

Wyświetlacz ciekłokrystaliczny (LCD) jest wykonany z kilku (do sześciu) warstw (rys. 4.7). Przednią warstwę tworzy płyta polaryzatora pionowego. Następna warstwa to siatka cienkich przewodów osadzonych elektrolitycznie na powierzchni sąsiadującej z kryształami. Dalej jest cienka (ok. 0,0005 cala) warstwa ciekłego kryształu. Potem jest warstwa z pionowymi przewodami siatki. Z kolei jest polaryzator poziomy i wreszcie reflektor.

Materiał ciekłokrystaliczny składa się z długich cząsteczek krystalicznych. Poszczególne cząsteczki są normalnie uporządkowane spiralnie tak, że kierunek polaryzacji przechodzącego spolaryzowanego światła jest obrócony o 90°. Światło wchodzące przez warstwę przednią jest



Rys. 4.7. Warstwy wyświetlacza ciekłokrystalicznego (LCD); wszystkie te warstwy są łączone tworząc cienki panel

polaryzowane pionowo. Gdy światło przechodzi przez ciekły kryształ, płaszczyzna polaryzacji jest skręcana o 90° w stosunku do poziomu; teraz światło przechodzi przez tylny polaryzator poziomy, jest odbijane i wraca przez dwa polaryzatory i kryształ.

Gdy kryształy są w polu elektrycznym, wówczas wszystkie ustawiają się w tym samym kierunku i nie ma efektu skręcenia płaszczyzny polaryzacji. Dlatego kryształy w polu elektrycznym nie zmieniają polaryzacji przesyłanego światła; światło pozostaje spolaryzowane poziomo i nie przechodzi przez tylny polaryzator: światło jest absorbowane i obserwator widzi ciemną plamkę na ekranie.

Ciemną plamkę w punkcie (x_1, y_1) tworzy się za pomocą adresowania matrycowego. Punkt jest wybierany przez podanie ujemnego napięcia $-V$ na przewód x_1 poziomej siatki i dodatniego napięcia $+V$ na przewód y_1 pionowej siatki: ani $-V$ ani $+V$ nie są dostatecznie duże na to, żeby spowodować uporządkowanie kryształów, ale ich różnica wystarcza. Teraz kryształy w (x_1, y_1) nie zmieniają kierunku polaryzacji przesyłanego światła i światło pozostaje spolaryzowane pionowo i nie przechodzi przez tylny polaryzator: światło jest absorbowane i obserwator widzi czarną plamkę na ekranie.

Ekrany LCD z aktywną matrycą mają tranzystor w każdym punkcie (x, y) siatki. Tranzystory są wykorzystywane do wywoływania szybkich zmian stanów kryształów i do sterowania stopniem, w jakim te stany zostaną zmienione. Dzięki tym dwóm właściwościom wyświetlacze LCD mogą być wykorzytywane w miniaturowych odbiornikach telewizyjnych z obrazami o odcieniach zmieniających się w sposób ciągły. Kryształy mogą być również barwione. Najważniejsze jest to, że tranzystor może służyć jako pamięć stanu komórki i może podtrzymywać stan komórki do następnej zmiany. To znaczy, pamięć realizowana przez tranzystor umożliwia pozostanie komórce w stanie włączenia przez cały czas i tym samym zwiększenie jej jasności, w porównaniu

z sytuacją, kiedy byłaby odświeżana okresowo. Zbudowano barwne ekrany LCD o rozdzielczości do 800 × 1000 przy przekątnej 14 cali.

Zaletami wyświetlaczy LCD są: mały koszt, mały ciężar i mały pobór mocy. Dawniej główną wadą wyświetlaczy LCD była ich pasywność wyrażająca się w tym, że odbijały tylko światło padające i same nie generowały światła (chociaż można to obejść stosując podświetlanie od tyłu): nawet przy niewielkim oświetleniu zewnętrznym obraz był niewidoczny. Ostatnio problem ten został wyeliminowany dzięki użyciu aktywnych ekranów. W komputerach przenośnych z monitorami kolorowymi – dostępnymi od niedawna – są wykorzystywane zarówno aktywne, jak i pasywne wyświetlacze LCD. Ponieważ wyświetlacze LCD są małe i lekkie, mogą być używane jako elementy wyświetlające w hełmach, o jakich jest mowa w p. 8.1.6. Ponieważ wielkości ekranów LCD zwiększają się, a ich koszt maleje, mogą stanowić w przyszłości konkurencję dla monitorów kolorowych CTR.

Wyświetlacze elektroluminescencyjne (EL) mają taką samą strukturę siatkową, jaka jest wykorzystywana w wyświetlaczach LCD i plazmowych. Między przednim i tylnym panelem jest cienka (typowo 500 nm) warstwa materiału elektroluminescencyjnego, takiego jak siarczek cynku domieszkowany manganem, który emituje światło pod wpływem pola elektrycznego o dużym natężeniu (ok. 10^6 V/cm). Punkt na panelu jest wyświetlany na zasadzie adresowania matrycowego, przy czym między liniami wybierającymi poziomą i pionową jest napięcie kilkuset woltów. Są również dostępne barwne wyświetlacze elektroluminescencyjne.

Takie wyświetlacze są jasne i mogą być przełączane szybko; do pamiętania obrazu mogą być również wykorzystywane tranzystory związane z poszczególnymi pikselami. Typowe wymiary paneli to 6 × 8 cali do 12 × 16 cali z 70 adresowalnymi punktami na cal. Główną wadą takich wyświetlaczy jest to, że pobór mocy jest większy niż w panelach LCD. Jednak dzięki dużej jasności znajdują one zastosowanie w niektórych przenośnych komputerach.

W większości wyświetlaczy z dużym ekranem wykorzystuje się *projekcyjne lampy CRT*, w których światło z małego (o średnicy kilku cali), ale bardzo jasnego monochromatycznego monitora CRT jest wzmacniane i rzucane na ekran poprzez zakrzywione zwierciadło. W systemach kolorowych są używane projektory z filtrami czerwonym, zielonym i niebieskim. Kineskopy z maską nie wytwarzają dostatecznej ilości światła na to, żeby można było rzutować obraz na duży (o przekątnej 2 m) ekran.

System projekcyjny z zaworem świetlnym firmy General Electric jest używany do bardzo dużych ekranów, gdzie światło z projekcyjnego monitora CRT byłoby niewystarczające. Zawór świetlny jest to mechanizm sterowania ilością światła przepuszczanego przez zawór. Źródło światła może mieć znacznie większą jasność niż CRT. W najpopularniejszym

rozwiązaniu strumień elektronów tworzy obraz na cienkiej warstwie olejowej na płytce szklanej. Ładunek elektryczny powoduje zmianę grubości warstwy. Światło ze źródła o dużej jasności jest kierowane na szkło i załamywane w różnych kierunkach zależnie od grubości warstwy. Specjalny układ optyczny rzuca na ekran światło załamane w określonych kierunkach; pozostałe promienie są rozproszone. W tych systemach jest możliwa projekcja obrazów kolorowych, wymaga to jednak stosowania albo trzech projektorów, albo bardziej rozbudowanego układu optycznego z jednym projektorem. Więcej szczegółów można znaleźć w pracy [SHER93].

W tablicy 4.2 zebrano charakterystyki trzech głównych metod wyświetlania. Jednak tempo rozwoju technologii jest takie, że niektóre relacje mogą się zmienić w ciągu kilku następnych lat. Zauważmy również, że dane dla ciekłych kryształów są dla adresowania pasywnego; dla adresowania z aktywną matrycą są dostępne poziomy szarości i barwy.

Dokładniejsze informacje o tych metodach wyświetlania można znaleźć w [APT85; BALD85; CONR85; PERR85; SHER93; TANN85].

Tablica 4.2. Porównanie metod wyświetlania

Parametr	CRT	Eektroluminescencyjna	Ciekłe kryształy
Pobór mocy	dostateczny	dostateczny-dobry	doskonały
Wielkość ekranu	doskonała	dobra	dostateczna
Głębokość	zła	doskonała	doskonała
Ciężar	zły	doskonały	doskonały
Ostrość	dostateczna-dobra	dobra-wspaniała	wspaniała
Jasność	doskonała	doskonała	dostateczna-dobra
Adresowalność	dobra-doskonała	dobra	dostateczna-dobra
Kontrast	dobry-doskonały	dobry	dostateczny
Liczba poziomów jasności na plamkę	doskonała	dostateczna	dostateczna
Kąt widzenia	dostateczny	dobry	zły
Barwa	doskonała	dobra	dobra
Zakres względnych kosztów	mały	średni-wysoki	mały

4.3. Rastrowe systemy wyświetlania

Podstawowe koncepcje rastrowych systemów graficznych przedstawiono w rozdz. 1, a rozdz. 2 dał dalszy wgląd w rodzaje możliwych operacji w wyświetlaczu rastrowym. W tym punkcie omówimy różne elementy

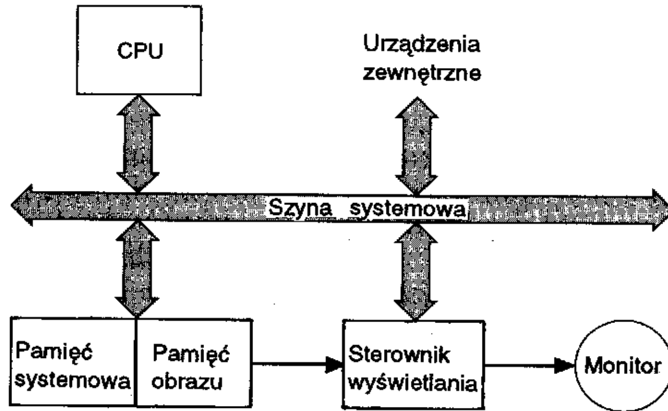
wyświetlaczy rastrowych i zwrócimy uwagę na dwa podstawowe elementy odróżniające od siebie różne systemy rastrowe.

Po pierwsze, większość rastrowych systemów wyświetlania ma specjalizowany sprzęt do rasteryzacji wyjściowych prymitywów do postaci mapy bitowej i do wykonywania rastrowych operacji przesuwania, kopiowania i modyfikowania pikseli albo bloków pikseli. Sprzęt ten określamy mianem *graficznego procesora urządzenia wyświetlającego*. Podstawowa różnica między systemami wyświetlania jest związana z podziałem zadań między procesorem urządzenia wyświetlającego a podprogramem pakietu graficznego wykonywanego na uniwersalnym CPU, który steruje rastrowym urządzeniem wyświetlającym. Zauważmy, że graficzny procesor urządzenia wyświetlającego jest określany również jako *sterownik graficzny* (dla podkreślenia podobieństwa do jednostek sterujących w innych urządzeniach zewnętrznych) albo *koprocesor urządzenia wyświetlającego*. Drugim kluczowym elementem różnicującym systemy rastrowe jest zależność między mapą pikselową a przestrzenią adresową pamięci uniwersalnego komputera – istotne jest to, czy mapa bitowa jest częścią pamięci komputera uniwersalnego, czy też jest niezależną pamięcią.

W punkcie 4.3.1 wprowadziliśmy prosty system rastrowy składający się z CPU z mapą pikselową w jego pamięci i sterownika wyświetlania dla CRT. Nie ma tam procesora urządzenia wyświetlającego i CPU wykonuje zadania programu użytkowego i graficzne. W punkcie 4.3.2 wprowadzono procesor graficzny z niezależną mapą pikselową; szeroki zakres możliwości funkcjonalnych procesora graficznego omówiono w p. 4.3.3. W punkcie 4.3.4 omówiono sposoby ponownego włączenia mapy pikselowej do przestrzeni adresowej CPU, przy istniejącym procesorze graficznym.

4.3.1. Prosty rastrowy system wyświetlania

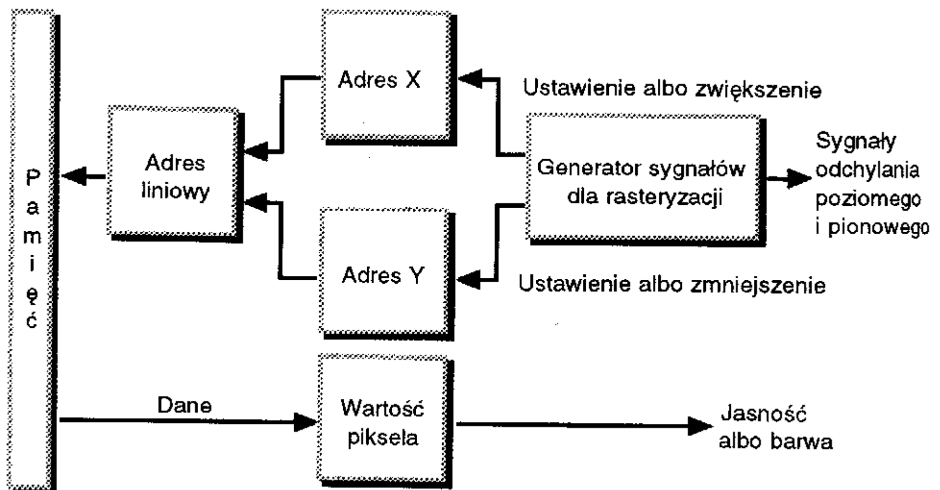
Na rysunku 4.8 pokazano najpopularniejszą organizację rastrowego systemu wyświetlania. Relacja między pamięcią a CPU jest dokładnie taka sama jak w systemach komputerowych bez grafiki. Część pamięci służy również jako mapa pikselowa. Sterownik wizyjny wyświetla obraz definiowany w pamięci obrazu; dostęp do pamięci jest przez oddzielny port z szybkością określoną przez układ rasteryzacji. W wielu systemach dla pamięci obrazu jest przydzielona stała część pamięci, natomiast w niektórych systemach jest kilka wymiennych obszarów pamięci (określanych czasami dla komputerów osobistych jako *strony*). W innych systemach na pamięć obrazu jest przeznaczana dowolna część pamięci (przez rejestr).



Rys. 4.8. Popularna architektura rastrowego systemu wyświetlania. Dedykowana część pamięci systemu jest pamięcią z podwójnym dostępem, tak że sterownik wyświetlania może mieć do niej bezpośredni dostęp bez konieczności angażowania szyny systemowej

Program użytkowy i podprogram programu graficznego wspólnie korzystają z pamięci systemu i są wykonywane przez CPU. Pakiet graficzny zawiera procedury rasteryzacji; gdy program użytkowy wywołuje, powiedzmy, SRGP_lineCoord ($x1$, $y1$, $x2$, $y2$) pakiet graficzny może ustawić odpowiednie piksele w pamięci obrazu (szczegóły na temat procedur rasteryzacji podano w rozdz. 3). Ponieważ pamięć obrazu jest w przestrzeni adresowej CPU, pakiet graficzny ma łatwy dostęp do pikseli i łatwa jest implementacja instrukcji PixBlt opisanej w rozdz. 2.

Sterownik wyświetlania przegląda pamięć obrazu wiersz po wierszu, na ogół z częstotliwością 60 Hz. Adresy odwołań do pamięci są



Rys. 4.9. Logiczna organizacja sterownika wyświetlania

generowane synchronicznie z przeglądaniem rastra i zawartość pamięci jest wykorzystywana do sterowania jasnością strumienia albo barwą monitora CRT. Sterownik wyświetlania jest zorganizowany w sposób pokazany na rys. 4.9. Generator sterujący przeglądaniem rastrowym wytwarza sygnały odchylania; steruje on również rejestrami adresowymi X i Y, które z kolei określają następną komórkę pamięci, do której ma nastąpić dostęp.

Załóżmy, że pamięć obrazu jest adresowana według współrzędnych x (od 0 do x_{max}) i y (od 0 do y_{max}); na początku cyklu odświeżania rejestr adresowy X jest zerowany, a do rejestru adresowego Y jest wstawiana wartość y_{max} (górna linia rastra). W czasie przeglądania pierwszej linii adres X jest zwiększany co jeden do wartości x_{max}. Odczytywane wartości kolejnych pikseli są używane do sterowania plamki. Po zakończeniu przeglądania pierwszej linii adres X jest zerowany, a adres Y jest zmniejszany o jeden. Proces powtarza się dopóty, dopóki nie zostanie wygenerowana ostatnia linia ($y = 0$).

W tej uproszczonej sytuacji dla każdego wyświetlanego piksela jest jeden dostęp do pamięci. Dla monitora o dużej rozdzielczości o 1000 liniach po 1000 pikseli z odświeżaniem z częstotliwością 60 Hz można oszacować czas na wyświetlenie jednego piksela jednobitowego: $1/(1000 \times 1000 \times 60) = 16$ ns. W tym obliczeniu pomija się fakt, że w systemie rastrowym piksele nie są wyświetlane w czasie powrotów poziomego (raz na linię) i pionowego (raz na ramkę). Natomiast typowe układy pamięci RAM mają czasy cyklu ok. 80 ns: dostęp do nich nie może odbywać się co 16 ns. Dlatego sterownik wyświetlania musi pobierać z pamięci wiele wartości pikseli w jednym cyklu pamięci. W przykładowym rozwiązaniu sterownik mógłby pobierać 16 bitów w jednym cyklu pamięci, co umożliwiałoby uzyskanie czasu oświeżania 16 pikseli $\times$ 16 ns/piksel $=$ 256 ns. 16 bitów jest ładowanych do rejestru sterownika wyświetlania; potem są one wysuwane co 16 ns w celu sterowania jasnością strumienia CRT. W 256 nanosekundach mieszczą się trzy cykle pamięci: jeden dla sterownika wyświetlania i dwa dla CPU. Przy takim podziale może się zdarzyć, że CPU będzie czekało na dostęp do pamięci, co potencjalnie proporcjonalnie zmniejsza szybkość CPU. Oczywiście pamięć podręczna CPU może polepszyć tę sytuację. Inne podejście polega na tym, żeby do budowy pamięci obrazu stosować niekonwencjonalne układy pamięci. Na przykład, włączenie wszystkich pikseli w wierszu redukuje liczbę cykli pamięci potrzebnych do przeglądania linii, zwłaszcza dla wypełnionych obszarów. W pamięciach wizyjnych VRAM opracowanych w firmie Texas Instruments można w jednym cyklu odczytać wszystkie piksele z jednej linii, co redukuje liczbę cykli pamięci potrzebnych do odświeżania ekranu.

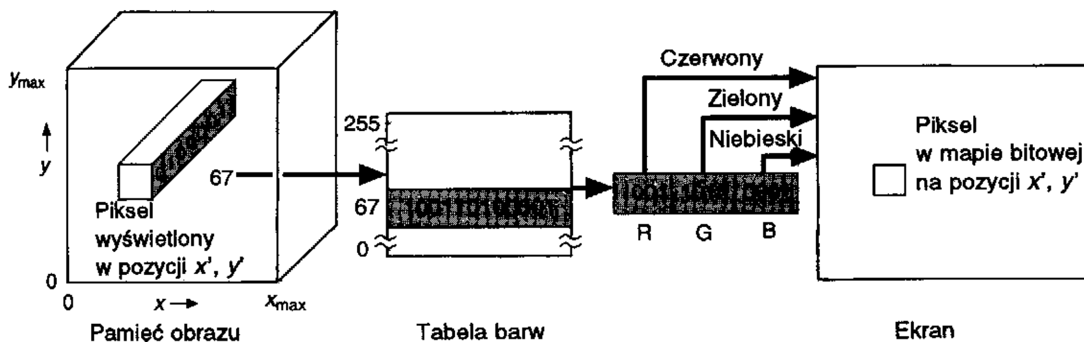
Dotychczas zakładaliśmy, że mapy bitowe są monochromatyczne jednopikselowe. Takie założenie jest dobre dla niektórych zastosowań, jest natomiast wysoce niezadowalające dla innych. Dodatkowe sterowanie jasnością każdego piksela wymaga pamiętania wielu bitów dla każdego piksela; przy dwóch bitach mamy cztery poziomy jasności itd. Bity mogą być wykorzystywane nie tylko do sterowania jasnością, ale również do sterowania barwą. Ile bitów na piksel trzeba na to, żeby zapamiętany obraz był odbierany jako obraz z ciągłymi poziomami szarości? Często wystarcza 5 albo 6 bitów; czasami jednak trzeba 8 i więcej bitów. Uproszczone rozumowanie prowadzi do wniosku, że dla monitorów kolorowych trzeba trzy razy tyle bitów: 8 bitów dla każdej z trzech addytywnych barw podstawowych czerwonej, niebieskiej i zielonej.

Systemy z 24 bitami na piksel są dość tanie, niemniej wiele zastosowań nie potrzebuje 2^{24} różnych barw w jednym obrazie (który typowo ma 2^{18} do 2^{20} pikseli). Często istnieje potrzeba, żeby było mało barw w obrazie albo zastosowaniu i żeby istniała możliwość zmiany barw z obrazu na obraz albo od zastosowania do zastosowania. W wielu zastosowaniach związanych z analizą obrazów i z ulepszaniem obrazów dobrze jest zmienić wygląd obrazu bez zmiany danych definiujących obraz; na przykład czasami chodzi o wyświetlenie wszystkich wartości poniżej pewnego progu jako czarnych, o rozszerzenie zakresu jasności albo o wyświetlanie obrazów monochromatycznych z pseudobarwami.

Ze względu na te różne przyczyny sterownik wyświetlania dla rastrowych urządzeń wyświetlających często zawiera *tabelę barw* (LUT). Tabela barw ma tyle pozycji, ile jest wartości dla jednego piksela. Wartość piksela nie jest używana do bezpośredniego sterowania strumieniem, a jest indeksem do tabeli barw. Wartości pamiętane w pozycjach tabeli są używane do sterowania jasnością albo barwą ekranu CRT. Wartość 67 przypisana pikselowi określa dostęp do zawartości pozycji 67 tabeli; zawarta tam informacja jest wykorzystywana do sterowania strumieniem w kineskopie CRT. Ta operacja na tabeli jest wykonywana dla każdego piksela w każdym cyklu wyświetlania; dlatego dostęp do tabeli musi być szybki i CPU musi mieć możliwość załadowania zawartości tabeli po otrzymaniu odpowiedniego polecenia.

Na rysunku 4.10 tabela barw jest umieszczona między pamięcią obrazu a monitorem CRT. Pamięć obrazu ma 8 bitów na piksel i dlatego tabela barw ma 256 pozycji.

Prosta organizacja rastrowego systemu wyświetlania pokazana na rys. 4.8 jest stosowana w wielu tanich komputerach osobistych. Taki system można tanio zbudować; ma on jednak sporo niedogodności. Po pierwsze, programowa rasteryzacja jest wolna. Na przykład trzeba obliczyć adres (x, y) każdego piksela; potem musi on być przekodowany na adres pamięci składający się z pary bajt i pozycja bitu w bajcie.



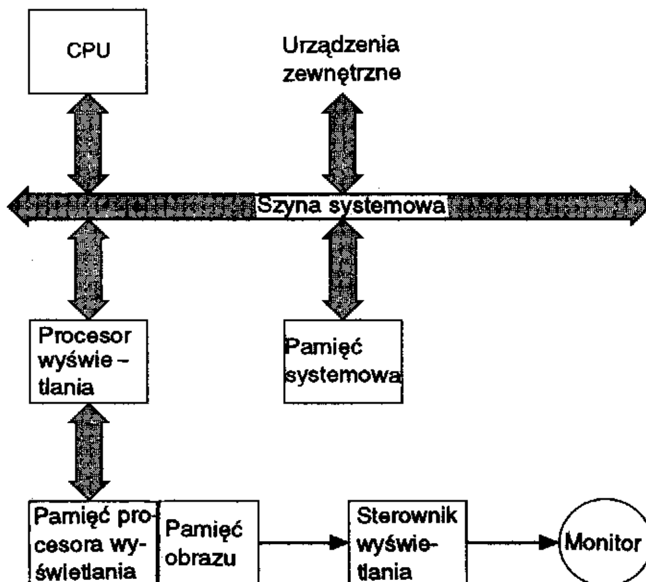
Rys. 4.10. Organizacja tabeli barw. Piksel o wartości 67 (dwójkowo 01000011 0) jest wyświetlany na ekranie z następującymi nastawami: 9/15 maksymalnej jasności dla strumienia wyrzutni czerwonego, 10/15 dla zielonego i 1/15 dla niebieskiego. Pokazana tabela barw ma po 12 bitów w każdej pozycji. Używa się tabel z 24 bitami w jednej pozycji

Chociaż każdy krok jest prosty, jest on powtarzany wiele razy. Programowa rasteryzacja zwalnia ogólną szybkość współpracy użytkownika z programem użytkowym, co stwarza pewien dyskomfort dla użytkownika.

Drugą wadą tej architektury jest to, że ze wzrostem liczby pikseli albo częstotliwości odświeżania monitora rośnie również liczba dostępów do pamięci wykonywanych przez sterownik wyświetlania, co ogranicza liczbę cykli pamięci dostępnych dla CPU. Wskutek tego zwalnia się szybkość pracy CPU, zwłaszcza w przypadku architektury, w której dostęp do pamięci obrazu jest przez szynę systemową. Przy podwójnym dostępie do części pamięci systemu (rys. 4.8) zwolnienie to pojawia się tylko wówczas, gdy CPU realizuje dostęp do pamięci obrazu, zazwyczaj dla wykonania rasteryzacji albo operacji rastrowych. Te dwie wady trzeba wziąć pod uwagę w kontekście łatwości, z jaką CPU może uzyskiwać dostęp do pamięci obrazu, i prostoty architektury systemu.

4.3.2. Rastrowy system wyświetlania z zewnętrznym procesorem wyświetlania

Rastrowy system wyświetlania z zewnętrznym procesorem wyświetlania jest popularnym rozwiązaniem (rys. 4.11), w którym unika się wad prostego wyświetlania rastrowego dzięki wprowadzeniu oddzielnego procesora graficznego do wykonywania takich funkcji graficznych, jak rasteryzacja i operacje rastrowe, oraz oddzielnej pamięci obrazu do odświeżania obrazu. Teraz mamy dwa procesory: uniwersalny procesor CPU i specjalizowany procesor wyświetlania. Ponadto mamy trzy obszary pamięci: pamięć systemową, pamięć procesora wyświetlania i pamięć obrazu. Pamięć systemowa przechowuje dane oraz te programy,



Rys. 4.11. Architektura systemu rastrowego z zewnętrznym procesorem wyświetlania

które wykonuje CPU: program użytkowy, pakiet graficzny i system operacyjny. Podobnie, pamięć procesora wyświetlania przechowuje dane oraz programy rasteryzacji i operacji rastrowych. Pamięć obrazu zawiera obraz gotowy do wyświetlenia, utworzony w wyniku rasteryzacji i operacji rastrowych.

W prostych przypadkach procesor wyświetlania może się składać ze specjalizowanych układów do wykonywania odwzorowania ze współrzędnych 2D (x, y) na liniowe adresy pamięci. W tym przypadku rasteryzacja i operacje rastrowe są w dalszym ciągu wykonywane przez CPU i pamięć procesora wyświetlania nie jest potrzebna; obecna jest tylko pamięć obrazu. Większość zewnętrznych procesorów wyświetlania wykonuje również rasteryzację. W tym punkcie przedstawiamy system prototypowy. Jego właściwości funkcjonalne (czasami uproszczone) są złożeniem wielu typowych systemów dostępnych w handlu, np. karty graficzne do komputerów typu IBM PC.

Pamięć obrazu ma 1024 × 1024 8-bitowych pikseli; jest również 256-pozycyjna tabela barw po 12 bitów na każdą barwę (po 4 bity na R, G, B). Początek jest z lewej strony na dole; wyświetlanych jest tylko pierwszych 768 wierszy mapy pikselowej (y w przedziale od 0 do 767). Sześć rejestrów stanu jest ustawianych przez różne instrukcje; te rejestry mają wpływ na wykonywanie innych instrukcji. Są to rejestry: CP (złożony z rejestrów pozycji X i Y), FILL, INDEX, WMODE, MASK i PATTERN. Ich działanie jest wyjaśnione niżej.

Wśród instrukcji prostego rastrowego urządzenia wyświetlającego są:

Move (x, y). W rejestrach X i Y, które definiują bieżącą pozycję (CP) są ustawiane wartości x i y. Ponieważ mamy mapę pikselową 1024×1024, to x i y muszą być w przedziale 0 do 1023.

MoveR (dx, dy). Wartości dx i dy są dodawane do rejestrów X i Y, co definiuje nowe CP. Wartości dx i dy muszą się zawierać w przedziale -1024 do $+1023$ i są reprezentowane w kodzie uzupełnień do 2. Dodawanie może spowodować nadmiar i dlatego ma miejsce cykliczne przeniesienie wartości rejestrów X i Y.

Line (x, y). Jest rysowany odcinek od CP do (x, y) i ta pozycja staje się nową pozycją CP.

LineR (dx, dy). Jest rysowany odcinek od CP do CP + (dx, dy) i ta pozycja staje się nową pozycją CP.

Point (x, y). Jest ustawiany piksel (x, y) i ta pozycja staje się nową pozycją CP.

PointR (dx, dy). Jest ustawiany piksel w CP + (dx, dy) i ta pozycja staje się nową pozycją CP.

Rect (x, y). Jest rysowany prostokąt między CP i (x, y). CP nie ulega zmianie.

RectR (dx, dy). Jest rysowany prostokąt między CP i CP + (dx, dy). O parametrze dx można myśleć jak o szerokości prostokąta, a o dy jak o wysokości prostokąta. CP nie ulega zmianie.

Text (*n*, *address*). Wyświetlanych jest *n* znaków z pamięci o adresie pamięci *address*, poczynając od CP. Znaki są definiowane na siatce 7×9 pikseli, z dwoma dodatkowymi pikselami odstępu pionowego i poziomego w celu oddzielenia znaków i wierszy. CP jest uaktualniane na wartość lewego dolnego rogu obszaru, w którym powinien być wyświetlony znak $n + 1$.

Circle (*radius*). Jest rysowany okrąg o środku w CP. CP nie ulega zmianie.

Polygon (*n*, *address*). Pod adresem *address* jest lista wierzchołków $(x_1, y_1, x_2, y_2, x_3, y_3, ..., x_n, y_n)$. Poczynając od (x_1, y_1) jest rysowany wielokąt, przechodzący przez wszystkie wierzchołki aż do (x_n, y_n) i z powrotem do (x_1, y_1). CP nie ulega zmianie.

AreaFill (*flag*). Parametr *flag* jest wykorzystywany do ustawienia wskaźnika FILL. Gdy wskaźnik ma wartość ON (niezerowa wartość parametru *flag*), wówczas wszystkie obszary tworzone przez polecenia Rect, RectR, Circle, CircleSector, Polygon są wypełniane w czasie tworzenia wzorem zdefiniowanym za pomocą polecenia Pattern.

RasterOp $(dx, dy$, *xdest*, *ydest*). Prostokątny obszar pamięci obrazu określony przez CP i CP + (dx, dy) jest zapisywany na miejsce obszaru o tej samej wielkości z lewym dolnym rogiem w (*xdest*, *ydest*). Operacja jest kontrolowana przez rejestr WMODE.

Polecenia i dane bezpośrednie są przesyłane do procesora wyświetlania przez bufor *FIFO* (to znaczy kolejkę) w dedykowanej części przestrzeni adresowej CPU. Pakiet graficzny umieszcza polecenia w kolejce, skąd są one kolejno pobierane i wykonywane. Wskaźniki początku i końca bufora znajdują się również w określonych miejscach pamięci, dostępnych zarówno dla CPU, jak i monitora. Wskaźnik początku bufora jest modyfikowany przez procesor wyświetlania za każdym razem, kiedy bajt jest usuwany; wskaźnik końca bufora jest modyfikowany przez CPU za każdym razem, kiedy bajt jest dodawany. Odpowiednie testowanie zabezpiecza przed odczytem pustego bufora i zapisem pełnego bufora. Do pobierania danych dla instrukcji jest wykorzystywana metoda bezpośredniego dostępu do pamięci.

Kolejka jest bardziej atrakcyjna dla przekazywania poleceń niż rejestr jednej instrukcji albo miejsce w pamięci dostępne dla monitora. Po pierwsze, zmienna długość instrukcji faworyzuje koncepcję kolejki. Po drugie, CPU może wyprzedzać monitor i ustawiać w kolejce kilka poleceń wyświetlania. Gdy CPU skończy wysyłanie poleceń wyświetlania, może przejść do innego zadania w czasie, gdy monitor opróżnia kolejkę.

Programowanie monitora jest podobne jak w pakiecie SRGP z rozdz. 2. Kilka przykładowych programów pokazano w rozdz. 4 książki [FOLE90].

4.3.3. Dodatkowe funkcje procesora wyświetlania

Nasz prosty procesor wyświetlania wykonuje tylko niektóre z możliwych do zrealizowania operacji graficznych. To, na czym może zależeć projektantowi systemu, to coraz większe odciążanie CPU przez dodawanie funkcji do procesora wyświetlania, takich jak pamiętanie list instrukcji wyświetlania w lokalnej pamięci, dodanie obcinania i przekształceń okno-pole wizualizacji i, być może, dodanie logiki korelacji wskazywania i automatycznego sprzężenia zwrotnego po wskazaniu elementu graficznego. Ostatecznie procesor wyświetlania staje się drugim uniwersalnym CPU, wykonującym interakcję graficzną i projektant znowu chciałby dodać specjalizowany sprzęt w celu odciążenia procesora wyświetlania.

To *koło reinkarnacji* zostało zidentyfikowane przez Myera i Sutherlanda w 1968 r. [MYER68]. Ich tezą było, że istnieje kompromis między funkcjami uniwersalnymi i specjalizowanymi. Sprzęt specjalizowany zazwyczaj wykonuje prace szybciej niż procesor uniwersalny. Sprzęt specjalizowany jest jednak droższy i nie może być używany do innych celów. Ten kompromis jest stałym tematem przy projektowaniu systemów graficznych.

Jeżeli do procesora wyświetlania dodamy obcinanie (rozdz. 3), to prymitywy wyjściowe mogą być przesyłane do procesora w innym układzie współrzędnych niż układ współrzędnych urządzenia. Ta specyfikacja może być we współrzędnych zmiennopozycyjnych, chociaż niektóre procesory wyświetlania operują tylko na liczbach całkowitych (to się szybko zmienia, w miarę jak pojawiają się nowe zmiennopozycyjne układy scalone). Jeżeli są używane tylko liczby całkowite, to współrzędne używane przez program użytkowy muszą być całkowite albo program graficzny musi zamieniać współrzędne zmiennopozycyjne na współrzędne całkowitoliczbowe. Żeby takie odwzorowanie było możliwe, program użytkowy musi przekazać pakietowi graficznemu prostokąt, w którym na pewno znajdują się współrzędne wszystkich prymitywów wyjściowych pakietu. Prostokąt musi zostać odwzorowany na największy zakres liczb całkowitych, tak żeby wszystko co jest w prostokącie, było w zakresie współrzędnych całkowitoliczbowych.

Jeżeli pakiet dopuszcza obiekty 3D, to procesor wyświetlania może wykonywać bardziej złożone przekształcenia geometryczne 3D i obcinanie opisane w rozdz. 5 i 6. Jeżeli pakiet zawiera prymitywy powierzchniowe 3D, np. powierzchnie wielokątowe, to procesor wyświetlania może również wykonywać określanie powierzchni widocznych i kroki renderingu opisane w rozdz. 13 i 14. W rozdziale 18 książki [FOLE90] są omawiane pewne podstawowe podejścia do organizacji układów VLSI, uniwersalnych i specjalizowanych do szybkiego wykonywania tych kroków. Wiele handlowo dostępnych monitorów ma takie funkcje.

Inną funkcją, która często jest dodawana do procesora wyświetlania, jest lokalna pamięć segmentów, określana również jako *pamięć listy wyświetlania*. Instrukcje wyświetlania, zgrupowane we wspomniane segmenty i mające nie obcięte współrzędne całkowite, są pamiętane w pamięci procesora wyświetlania, dzięki czemu może on działać bardziej autonomicznie w stosunku do CPU.

Co na prawdę może zrobić procesor wyświetlania z tymi zapamiętanymi segmentami? Może dokonać ich przekształcenia, na przykład powiększenia albo przewijania, i przerysowania. Można zrealizować lokalne przeciągnięcie segmentu na nową pozycję. Można zrealizować lokalne wskazywanie na zasadzie porównania pozycji kursora ze wszystkimi prymitywami graficznymi (bardziej efektywne metody wykonywania tego są omawiane w rozdz. 7). Pamięć segmentów umożliwia również wykonanie regeneracji potrzebnej do wypełnienia dziur tworzonych w czasie wymazywania segmentu. Segmenty można tworzyć, usuwać, edytować i czynić widocznymi i niewidocznymi.

Segmenty można kopiować i można się do nich odwoływać, zmniejszając ilość informacji, która musi być wysłana z CPU do procesora wyświetlania, oraz zwiększając ekonomikę wykorzystania pamięci w sa-

mym procesorze wyświetlania. Korzystając z tej możliwości można również budować złożone hierarchiczne struktury danych; wiele komercyjnych procesorów wyświetlania z lokalną pamięcią segmentów umożliwia kopiowanie i odwoływanie się do innych segmentów. Gdy segment jest wyświetlany, wówczas odniesienie do innego segmentu musi być poprzedzone przez zapamiętanie bieżącego stanu procesora wyświetlania, tak jak wywołanie podprogramu musi być poprzedzone zapamiętaniem bieżącego stanu CPU. Odwołania mogą być zagnieżdżane, co prowadzi do *strukturalnego pliku wyświetlania* albo do *hierarchicznej struktury wyświetlania*, takiej jak w PHIGS [ANSI88], która jest omawiana w rozdz. 7.

Chociaż ta architektura rastrowego systemu wyświetlania z własnym monitorem graficznym i oddzielną pamięcią obrazu ma wiele zalet w porównaniu z prostym rastrowym systemem wyświetlania z p. 4.3.1, ma również parę wad. Jeżeli jednostka centralna uzyskuje dostęp do procesora wyświetlania jak do urządzenia zewnętrznego przez kanał bezpośredniego dostępu do pamięci, to trzeba uwzględnić czas na działanie systemu operacyjnego za każdym razem, gdy instrukcja jest przekazywana do procesora wyświetlania (nie występuje to w procesorze wyświetlania, w którym rejestr instrukcji jest odwzorowany w przestrzeni adresowej CPU, ponieważ wtedy pakiet graficzny może łatwo bezpośrednio ustawiać rejestry).

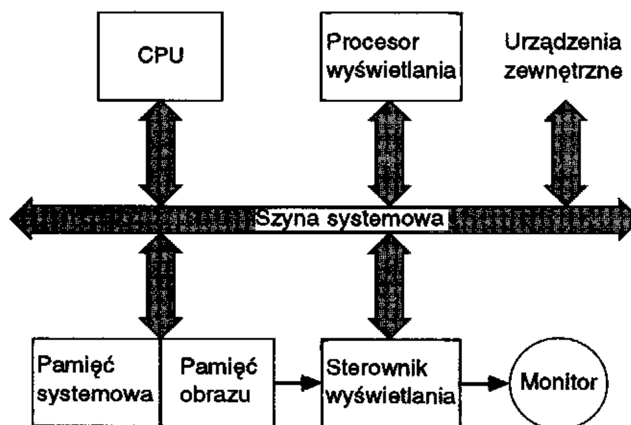
Polecenie związane z operacją rastrową jest szczególnie trudne. Koncepcyjnie powinno ono mieć cztery pary potencjalnych źródeł i miejsc docelowych: pamięć systemowa do pamięci systemowej, pamięć systemowa do pamięci obrazu, pamięć obrazu do pamięci systemowej i pamięć obrazu do pamięci obrazu (tutaj pamięć obrazu i pamięć procesora wyświetlania z rys. 4.11 są traktowane identycznie, ponieważ są one w tej samej przestrzeni adresowej). Jednak w systemach monitor-procesor różne pary źródło-miejsce docelowe są traktowane rozmaicie i przypadek pamięć systemu do pamięci systemu może nie istnieć. Ten brak symetrii komplikuje zadania programisty i zmniejsza elastyczność. Na przykład, jeżeli pozaekranowa część mapy pikselowej jest wypełniana przez menu, kroje pisma itp., to trudno jest wykorzystać pamięć główną jako obszar przepełnienia. Ponieważ korzystanie z mapy pikselowej jest tak rozpowszechnione, brak możliwości wykonywania operacji rastrowych na mapach pikselowych zapamiętanych w głównej pamięci nie może być zaakceptowany.

Procesor wyświetlania zdefiniowany wcześniej w tym punkcie, podobnie jak wiele rzeczywistych procesorów wyświetlania, przesyła obrazy rastrowe między pamięcią systemu a pamięcią obrazu na zasadzie przesłań we/wy po szynie systemu. Niestety, to przesyłanie może być zbyt wolne dla operacji w czasie rzeczywistym, takich jak animowanie, przeciąganie czy chwilowe wyświetlanie okien i menu – na przeszkodzie

stoi czas potrzebny na przesłanie szyną po zainicjowaniu przesłania przez system operacyjny. Ten problem może być częściowo złagodzony przez zwiększenie pamięci procesora wyświetlania, tak żeby można było przechowywać więcej pozaekranowych map pikselowych, ale wtedy ta pamięć nie jest dostępna dla innych celów – i prawie nigdy nie ma dostatecznie dużo pamięci!

4.3.4. Rastrowy system wyświetlania z wbudowanym procesorem wyświetlania

Możemy usunąć wiele ograniczeń zewnętrznego procesora wyświetlania, omawianego w poprzednim punkcie, poświęcając specjalną część pamięci systemu na pamięć obrazu i realizując drugi kanał dostępu do pamięci obrazu dla sterownika wyświetlania; w ten sposób powstaje architektura systemu wyświetlania z jedną przestrzenią adresową (SAS), jak na rys. 4.12. Tutaj procesor wyświetlania, CPU i sterownik wyświetlania są połączone z szyną systemową i wszystkie te podzespoły mają dostęp do pamięci systemu. Początek i czasami wielkość pamięci obrazu są pamiętane w rejestrach, dzięki czemu podwójne buforowanie staje się proste w realizacji i wymaga jedynie przeładowania rejestru początku. Wyniki rasteryzacji mogą być przesyłane albo do pamięci obrazu do natychmiastowego wyświetlenia, albo do pamięci systemu do późniejszego wyświetlenia. Podobnie, źródło i miejsce przeznaczenia dla operacji rastrowych wykonywanych przez procesor wyświetlania mogą być gdziekolwiek w pamięci systemu (teraz jest to jedyna interesująca nas



Rys. 4.12. Popularna architektura systemu rastrowego wyświetlania z jedną przestrzenią adresową (SAS) z integralnym procesorem wyświetlania. Procesor wyświetlania może mieć własną pamięć dla algorytmów i pamięć roboczą. Część pamięci systemu jest pamięcią dwuportową i dostęp do niej może być bezpośrednio ze sterownika wyświetlania, bez angażowania szyny systemu

pamięć). Taka konfiguracja jest również atrakcyjna ze względu na to, że CPU może bezpośrednio manipulować pikselami w pamięci obrazu po prostu na zasadzie zapisywania i odczytywania odpowiednich bitów.

Architektura SAS ma jednak kilka wad. Najpoważniejszą jest problem współzawodniczenia o dostęp do pamięci systemu. Jedno z rozwiązań tego problemu polega na użyciu układu CPU zawierającego podręczne pamięci instrukcji i danych, co umożliwia zredukowanie zależności CPU od częstych i szybkich dostępów do pamieci systemu. Oczywiście te i inne rozwiązania mogą być wykorzystywane na wiele różnych pomysłowych sposobów; dalsza dyskusja jest w rozdz. 18 książki [FOLE90].

Inna komplikacja projektowa pojawia się, gdy CPU ma wirtualną przestrzeń adresową, tak jak to jest powszechnie używane w rodzinach Motorola 680X0 i Intel 80X86 i różnych procesorach ze zredukowaną listą instrukcji (RISC). W tym przypadku adresy generowane przez procesor wyświetlania muszą przejść przez tę samą dynamiczną translację adresów tak jak inne adresy pamięci. Wiele architektur CPU wyróżnia wirtualną przestrzeń adresową rdzenia systemu operacyjnego i wirtualną przestrzeń adresową programu użytkowego. Często dobrze jest, żeby pamięć obrazu (kanwa 0 w terminologii SRGP) była w przestrzeni rdzenia po to, żeby sterownik urządzenia wyświetlającego systemu operacyjnego miał do niej bezpośredni dostęp. Jednak kanwy alokowane przez program użytkowy muszą być w przestrzeni adresowej. Dlatego instrukcje wyświetlania, które korzystają z pamięci obrazu, muszą rozróżniać między przestrzeniami adresowymi rdzenia i zastosowania. Jeżeli ma być dostęp do rdzenia, to dostęp do instrukcji wyświetlania musi nastąpić przez czasochłonne wywołanie usługi systemu operacyjnego, a nie przez proste wywołanie podprogramu.

Mimo tych potencjalnych komplikacji, coraz więcej rastrowych systemów wyświetlania ma architekturę z jedną przestrzenią adresową, na ogół takiego typu jak na rys. 4.12. Elastyczność związana z umożliwieniem zarówno CPU, jak i procesorowi wyświetlania dostępu do dowolnej części pamięci w jednolity i jednorodny sposób jest bardzo zachęcająca i upraszcza programowanie.

4.4. Sterownik wyświetlania

Najważniejszym zadaniem sterownika wyświetlania jest stałe odświeżanie ekranu. Są dwa podstawowe typy odświeżania: z wybieraniem międzyliniowym i bez wybierania międzyliniowego. Ten pierwszy rodzaj jest stosowany w telewizji i w rastrowych urządzeniach wyświetlających wykorzystujących zwykłe odbiorniki telewizyjne. Cykl odświeżania jest

podzielony na dwa pola trwające po 1/60 sekundy każde; dlatego całe odświeżanie trwa 1/30 sekundy. Wszystkie nieparzyste linie są wyświetlane w pierwszym polu, a wszystkie parzyste w drugim. Przy wyświetlaniu z wybieraniem międzyliniowym chodzi o to, żeby można było umieścić nową informację we wszystkich obszarach ekranu z częstotliwością 60 Hz, ponieważ przy odświeżaniu z częstotliwością 30 Hz może powstawać efekt migotania. Skutkiem stosowania wybierania międzyliniowego ma być utworzenie obrazu, którego efektywna szybkość odświeżania jest bliższa 60 niż 30 Hz. Ta metoda działa poprawnie dopóty, dopóki w sąsiednich wierszach jest wyświetlana podobna informacja; obraz składający się z poziomych linii w kolejnych wierszach rasteryzacji będzie wyglądał gorzej. Większość sterowników wyświetlania zapewnia odświeżanie z częstotliwością 60 Hz oraz większą i nie korzysta z metody wybierania międzyliniowego.

Wyjście ze sterownika wyświetlania ma jedną z trzech postaci: RGB, monochromatyczna albo NTSC. W przypadku wyjścia RGB odzielne przewody przenoszą sygnały dla barw czerwonej, zielonej i niebieskiej służące do sterowania wyrzutniami elektronowymi monitora CRT, a odrębny przewód przenosi sygnał synchronizacji odpowiedzialny za start pionowego i poziomego powrotu. Odpowiednie standardy określają napięcia, kształty przebiegów i zależności czasowe dla synchronizacji sygnałów *R*, *G*, *B*. Dla sygnałów monochromatycznych i ekranu o 480 liniach standardem jest RS-170; dla sygnałów barwnych RS-170A; dla sygnałów monochromatycznych i o dużej rozdzielczości RS-343. Często sygnały synchronizacji są przesyłane tym samym przewodem co sygnał dla barwy zielonej – w tym przypadku mówimy o całkowitym sygnale wizyjnym. Przy przesyłaniu sygnałów monochromatycznych korzysta się z tych samych standardów, z tym, że wykorzystuje się tylko przewody niosące informacje o jasności i synchronizacji, albo wręcz tylko jeden przewód przenoszący całkowity sygnał niosący informację o jasności i synchronizacji.

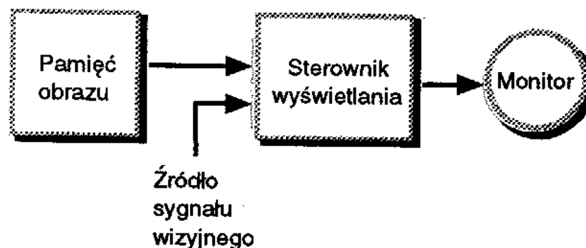
Format sygnału wizyjnego NTSC (National Television System Committee) jest używany w komercyjnej telewizji w Ameryce Północnej. Barwa, jasność i informacja synchronizująca są połączone w sygnał o pasmie ok. 5 MHz, wysyłany jako 525 linii w dwóch polach po 262,5 linii każde. Widocznych jest tylko 480 linii; reszta występuje w czasie pionowego powrotu na końcu każdego pola. Odbiornik telewizji monochromatycznej wykorzystuje informację o jasności i synchronizacji; odbiornik telewizji kolorowej wykorzystuje również informację o barwie do sterowania trzema wyrzutniami. Dzięki ograniczeniu pasma jest możliwe nadawanie wielu różnych kanałów telewizyjnych w zakresie częstotliwości przydzielonych telewizji. Niestety takie pasmo ogranicza jakość obrazu do efektywnej rozdzielczości ok. 350 × 350. Niemniej

NTSC jest standardem dla taniego sprzętu magnetowidowego. W droższych rejestratorach oddzielne składowe sygnału barwy są pamiętane w postaci analogowej albo cyfrowej. W Europie i Rosji są wykorzystywane standardy SECAM i PAL (625 linii i 50 Hz).

Niektóre sterowniki wyświetlania dodają programowany kursor, pamiętany w mapie pikselowej 16 × 16 albo 32 × 32 niezależnej od pamięci obrazu. Dzięki temu unika się operacji PixBlt przesyłającej wzór kursora do pamieci obrazu w każdym cyklu wyświetlania, co trochę zmniejsza narzut wnoszony przez CPU. Podobnie niektóre sterowniki wyświetlania dodają liczne małe mapy pikselowe o stałej wielkości (nazywane *sprajtami* – duszkami) pamiętane w niezależnej pamięci. Ta funkcja jest często wykorzystywana w grach komputerowych.

4.4.1. Mieszanie sygnałów wizyjnych

Inną użyteczną funkcją sterownika wyświetlania jest mieszanie sygnałów wizyjnych. Dwa obrazy, jeden zdefiniowany w pamięci obrazu i drugi będący sygnałem wizyjnym przychodzącym z kamery, magnetowidu lub innego źródła, mogą być zmieszane i utworzyć złożony obraz. Przykłady takiego mieszania są widoczne stale w telewizji w dziennikach, wiadomościach sportowych i prognozach pogody. Na rysunku 4.13 pokazano podstawową konfigurację systemu.



Rys. 4.13. Sterownik wyświetlania miksujący obrazy z niezależnej pamięci i ze źródła sygnału wizyjnego

Sa dwa rodzaje mieszania. W jednym obraz graficzny jest wstawiany w obraz wideo. Typowym przykładem tego jest wykres wyświetlany nad ramieniem prezentera. Miksowanie wykonuje się sprzętowo; odpowiednią wyróżnioną wartość piksela w pamięci obrazu traktuje się jako wskaźnik informujący o tym, że zamiast sygnału z pamięci obrazu należy pokazać sygnał wideo. Normalnie wartością wyróżnioną jest barwa tła obrazu w pamięci obrazu; interesujące efekty można uzyskać również korzystając z jakiejś innej wartości piksela.

Przy drugim typie miksowania sygnał wideo jest wstawiany w obraz z pamięci obrazu, tak jak na przykład wówczas, gdy prezenter staje przed pełnoekranową mapą pogody. Prezenter faktycznie stoi przed kurtyną, której barwa (najczęściej niebieska) jest używana do sterowania miksowaniem: gdy nadchodzący obraz jest niebieski, wówczas jest pokazywana zawartość pamięci obrazu; w przeciwnym przypadku jest pokazywany obraz wideo. Ta metoda działa dobrze pod warunkiem, że prezenter nie nosi niebieskiej koszulki albo krawata!

4.5. Urządzenia wejściowe do współpracy z operatorem

W tym punkcie opiszemy działanie najpopularniejszych urządzeń wejściowych. Przedstawimy krótką dyskusję na temat działania dostępnych rodzajów urządzeń. W rozdziale 8 omawiamy zalety i wady różnych urządzeń i opisujemy niektóre bardziej zaawansowane urządzenia.

Nasza prezentacja jest uporządkowana zgodnie z koncepcją *urządzeń logicznych* wprowadzoną w rozdz. 2 przy omawianiu SRGP i omawianą dalej w rozdz. 7. Jest pięć podstawowych urządzeń logicznych: *lokalizatory*, wskazujące pozycję lub orientację; *urządzenia wskazujące* do wybierania wyświetlanego obiektu; *urządzenia do wprowadzania wartości*; *klawiatura* do wprowadzania ciągu znaków; *urządzenia wybierające* jedną z możliwych akcji lub opcji wyborów. Koncepcja urządzenia logicznego definiuje klasy równoważności urządzeń ze względu na typ informacji przekazywanej do programu użytkowego.

4.5.1. Lokalizatory

Tabliczka. Tabliczka jest to płaskie urządzenie o powierzchni użytkowej od 6 × 6 cali do 48 × 72 cali lub większej, które może umożliwiać ustalenie położenia ruchomego wskaźnika przypominającego pióro albo wskaźnika optycznego. Na rysunku 4.14 pokazano małą tabliczkę oraz oba rodzaje wskaźników (dalej będziemy mówili głównie o wskaźniku przypominającym pióro; niemniej dyskusja dotyczy obu rodzajów wskaźników). W większości wskaźników do określania pozycji wskaźnika wykorzystuje się czujnik elektryczny. W przykładowym rozwiązaniu pod powierzchnią tabliczki znajduje się siatka przewodów o skoku 1/4 do 1/2 cala. Sygnały elektromagnetyczne, generowane przez impulsy elektryczne doprowadzane kolejno do przewodów siatki, indukują sygnały elektryczne w cewce wskaźnika. Wartość indukowanego sygnału przez każdy impuls jest wykorzystywana do określenia pozycji wskaźnika. Wartość sygnału jest również wykorzystywana do zgrubnego określenia, jak wysoko nad tabliczką znajduje się wskaźnik (daleko, blisko, to jest nie dalej niż 1/2 cala, czy też dotyka powierzchni). Jeżeli

Rys. 4.14. Tabliczka ze wskaźnikami typu pióro i wskaźnikiem optycznym. Z końcówką pióra jest związany przycisk czuły na nacisk, który zwiera po naciśnięciu pióra. Wskaźnik optyczny ma kilka przycisków do wprowadzania poleceń i kursor w kształcie krzyżyka umożliwiający uzyskanie dużej dokładności przy wprowadzaniu danych z rysunku leżącego na tabliczce (Za zgodą Summagraphics Corporation.)

odpowiedź jest, że blisko albo że dotyka powierzchni, to zazwyczaj na ekranie jest pokazywany kursor i tym samym istnieje wzrokowe sprzężenie zwrotne dla użytkownika. Jeżeli końcówka wskaźnika zostaje dociśnięta do powierzchni tabliczki albo jeżeli zostanie naciśnięty przycisk wskaźnika optycznego (takie wskaźniki mają do 16 przycisków), to do komputera jest wysyłany sygnał. Pozycja (x, y) na tabliczce, stan przycisku i stan wysokości (jeżeli jest to stan daleko, to nie jest wysyłana pozycja (x, y)) są wysyłane 30 do 60 razy na sekundę.

Parametrami charakterystycznymi tabliczki i innych lokalizatorów są: rozdzielczość (liczba rozróżnialnych punktów na cal), liniowość, powtarzalność, wielkość oraz zakres. Te parametry są szczególnie istotne dla wprowadzania cyfrowych danych z map i innych rysunków; są one mniej istotne, jeżeli urządzenie jest stosowane tylko do umieszczania kursora ekranu, ponieważ wtedy użytkownik ma zapewnione sprzężenie zwrotne w postaci widocznego kursora i może na tej podstawie sterować ruchami ręki oraz ponieważ rozdzielczość typowego monitora jest znacznie mniejsza niż nawet najtańszej tabliczki. W innych tabliczkach są stosowane metody sprzężenia akustycznego albo sprzężenia rezystywnego.

Niektóre typy tabliczek są przezroczyste i mogą być podświetlone od dołu, dzięki czemu jest możliwe wprowadzanie danych z klisz rentgenowskich i negatywów fotograficznych; mogą one być również umieszczane bezpośrednio przed ekranem monitora. Do tego szczególnie nadają się tabliczki rezystywne, ponieważ można je dopasować do kształtu ekranu.

Myszka. Myszka jest to małe urządzenie wygodne do manipulowania ręką, dla którego można mierzyć względny ruch po powierzchni. Myszki różnią się liczbą przycisków i sposobem wykrywania względnego ruchu. Wiele zastosowań myszek w różnych zadaniach interakcyjnych omówiono w p. 8.1. Ruch kulki w obudowie *myszki mechanicznej* jest zamieniany na wartości cyfrowe, które są wykorzystywane do określania kierunku i wielkości ruchu. *Myszka optyczna* jest umieszczana na specjalnej podstawce z siatką jasnych i ciemnych linii umieszczonych na przemian. Dioda emitująca światło (LED) znajdująca się na spodzie myszki wysyła światło w dół na podstawkę, od której jest ono odbijane i wykrywane przez detektory na spodzie myszki. W czasie ruchu myszki odbity promień światła jest przerywany za każdym razem, gdy jest przecinana ciemna linia. Liczba generowanych w ten sposób impulsów, która jest równa liczbie przeciętych linii, jest używana do przekazywania informacji o ruchu myszki do komputera.

Ponieważ myszka jest urządzeniem przekazującym dane względne, może być podnoszona, przesuwana i ponownie opuszczana bez zmieniania podawanej pozycji. Komputer musi pamiętać bieżącą pozycję myszki, która jest zwiększana albo zmniejszana w wyniku ruchów myszki.

Kula śledząca. Kula śledząca jest często opisywana jako mechanicza myszka odwrócona do góry nogami. Ruch kuli, która obraca się swobodnie w obudowie, jest śledzony przez potencjometry albo koder obrotowy. Na ogół użytkownik obraca kulę poruszając dłonią opartą na kuli. Zwykle w zasięgu palców są montowane różne przełączniki używane podobnie jak w przypadku myszki albo wskaźnika optycznego tabliczki.

Drążek sterowniczy. Drążek sterowniczy pokazany na rys. 4.15 może być przesuwany w lewo albo w prawo, do przodu albo do tyłu; ruch

Rys. 4.15. Drążek sterowniczy z trzecim stopniem swobody. Drążek może być obracany w kierunku zgodnym z ruchem wskazówek zegara albo przeciwnym (Za zgodą Measurement Systems, Inc.)

jest śledzony przez odpowiednie potencjometry. Często są używane sprężyny zapewniające powrót drążka do pozycji spoczynkowej. Niektóre drążki, w tym ten pokazany na rysunku, mają trzeci stopień swobody; drążek może być obracany w kierunku zgodnym z ruchem wskazówek zegara albo w kierunku przeciwnym.

Za pomocą drążka trudno jest bezpośrednio sterować bezwzględnym położeniem kursora na ekranie, ponieważ niewielki ruch krótkiego (zazwyczaj) ramienia jest wzmacniany pięć do dziesięciu razy przy przejściu na ruch kursora. Skutkiem tego są nierównomierne ruchy kursora na ekranie i nie można szybko i dokładnie ustawić pozycji kursora. Dlatego drążki są używane raczej do sterowania szybkością ruchu kursora niż do określania bezwzględnej pozycji kursora. Oznacza to, że bieżąca pozycja kursora na ekranie zmienia się z szybkością określoną przez drążek.

Ekran dotykowy. Myszka, kula śledząca i drążek sterowniczy wykorzystują obszar powierzchni roboczej. Ekran dotykowy umożliwia użytkownikowi wskazywanie palcem bezpośrednio na ekranie i poruszanie kursorem po ekranie. Stosuje się kilka różnych technologii. W ekranach o niedużej rozdzielczości (10 do 50 rozróżnialnych punktów w każdym kierunku) diody LED działające w podczerwieni i czujniki światła (fotodiody albo fototranzystory) tworzą siatkę niewidocznych promieni światła nad powierzchnią ekranu. Dotknięcie ekranu powoduje przerwanie jednego albo dwóch promieni światła, poziomego albo pionowego, i określenie tym samym pozycji palca. Jeżeli zostaną przerwane dwa równoległe strumienie, to przyjmuje się, że palec jest pośrodku między nimi; jeżeli zostanie przerwany tylko jeden strumień, to przyjmuje się, że palec jest na linii strumienia.

Ekran dotykowy ze sprzężeniem pojemnościowym może dawać ponad 100 rozróżnialnych pozycji w każdym kierunku. Gdy użytkownik dotyka szklanego ekranu pokrytego materiałem przewodzącym, wówczas układy elektroniczne wykrywają miejsce dotknięcia wskutek zmiany impedancji przewodzącego pokrycia [INTE85].

Główne parametry ekranu dotykowego to rozdzielczość, wielkość nacisku potrzebnego do aktywacji (nie odnosi się to do ekranu ze strumieniem światła) i przezroczystość (znowu nie odnosi się to do ekranu ze strumieniem światła). Ważną cechą niektórych technologii jest paralaksa: jeżeli ekran dotykowy jest w odległości 1/2 cala od ekranu, to użytkownik dotyka miejsca na ekranie, które wynika z położenia odpowiedniego punktu na ekranie i z pozycji oczu, a nie miejsca na ekranie leżącego dokładnie na prostopadłej do ekranu i przechodzącej przez punkt na ekranie.

Użytkownicy są przyzwyczajeni do jakiegoś rodzaju sprzężenia dotykowego, natomiast ekran dotykowy nie daje żadnego sprzężenia tego

typu. Jest więc szczególnie istotne realizowanie innych rodzajów natychmiastowego sprzężenia zwrotnego, takich jak dźwiękowe albo podświetlanie odpowiedniej pozycji.

4.5.2. Klawiatury

Prototypowym urządzeniem do wprowadzania danych tekstowych jest *klawiatura alfanumeryczna*. Do wykrywania faktu naciśnięcia klawisza jest używanych kilka metod, w tym mechaniczne zwarcie kontaktu, zmiana pojemności, sprzężenie magnetyczne. Ważną cechą funkcjonalną klawiatury jest to, że generuje ona kod, na przykład ASCII, jednoznacznie odpowiadający naciśniętemu klawiszowi. Czasami dobrze jest nacisnąć równocześnie kilka klawiszy klawiatury alfanumerycznej po to, żeby dać doświadczonym użytkownikom szybki dostęp do wielu różnych poleceń. W ogólnym przypadku nie jest to możliwe; w *blokowanych klawiaturach* po naciśnięciu dowolnego znaku jest przesyłany znak ASCII, a po równoczesnym naciśnięciu dwóch klawiszy (chyba że jednym z nich jest klawisz shift, control albo inny klawisz specjalny) nic nie jest przesyłane. W *nieblokowanych klawiaturach* są przesyłane kody wszystkich klawiszy, które zostały równocześnie naciśnięte.

4.5.3. Urządzenia do wprowadzania wartości

W większości urządzeń do wprowadzania wartości skalarnych wykorzystuje się potencjometry, podobnie jak przy regulacji głośności i tonu w odbiorniku stereofonicznym. Urządzenia do wprowadzania wartości wykorzystują zazwyczaj potencjometry obrotowe montowane w grupach po 8 albo 10. Zwykłe potencjometry obrotowe mogą być obracane o ok. 330°; może to być niewystarczające do uzyskania odpowiedniego zakresu i rozdzielczości. Potencjometry umożliwiające ciągły obrót mogą być obracane swobodnie w każdym kierunku i dlatego mają nieograniczony zakres. Potencjometry liniowe, które z konieczności mają ograniczenia, są rzadko używane w systemach graficznych.

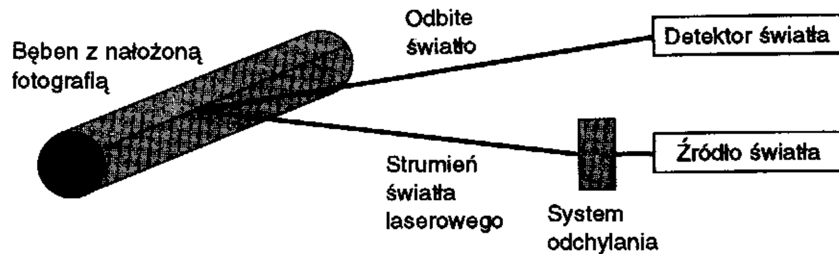
4.5.4. Urządzenia wybierające

Najpopularniejszymi urządzeniami wybierającymi są *klawisze funkcyjne*. Czasami są one wykonywane jako niezależne jednostki, ale częściej są połączone z klawiaturą. Inne urządzenia wybierające to *przyciski*

występujące we wskaźnikach optycznych tabliczek i w myszkach. Urządzenia wybierające są na ogół wykorzystywane w programach graficznych do wprowadzania poleceń albo do wybierania opcji menu. W specjalizowanych systemach można używać klawiszy funkcyjnych z odpowiednimi etykietami. Żeby etykiety można było wymieniać, obok klawiszy funkcyjnych lub przycisków albo w nich samych mogą być umieszczone małe wyświetlacze LCD albo LED. Inne rozwiązanie polega na umieszczeniu przycisków na brzegu monitora, tak żeby etykiety przycisków mogły być pokazywane na ekranie, tuż obok fizycznego przycisku.

4.6. Skanery obrazów

Chociaż tabliczki mogą być używane do ręcznego wprowadzania danych cyfrowych związanych z określonym rysunkiem, jest to proces wolny i męczący oraz nieprzydatny do wprowadzania więcej niż kilku prostych rysunków, a także do obróbki obrazów półtonowych. Efektywne rozwiązanie problemu zapewniają skanery obrazu. Kamera telewizyjna w połączeniu z układem cyfrowej akwizycji obrazów daje tanie rozwiązanie uzyskiwania obrazów rastrowych o niezbyt dużej rozdzielczości (1000 × 1000, z wieloma poziomami jasności) dla fotografii czarno-białych i barwnych. Kamery z układami CCD (układy ze sprzężeniami ładunkowymi) mogą tworzyć obraz o rozdzielczości 2000 × 2000 w ciągu 30 s. W jeszcze tańszym rozwiązaniu wykorzystuje się głowicę skanującą, składającą się z siatki światłoczułych komórek, montowanej na głowicy drukującej drukarki; można w ten sposób dokonać skanowania obrazów z rozdzielczością ok. 80 punktów na cal. Takie rozdzielczości nie mogą być jednak zaakceptowane w zastosowaniach związanych z przygotowaniem publikacji dobrej jakości. W takich przypadkach używa się *fotoskanera*. Fotografia jest montowana na obracającym się bębnie. Na fotografię zostaje skierowany strumień świetlny i fotokomórka mierzy ilość odbitego światła. W przypadku negatywów mierzy się przepuszczone światło za pomocą fotokomórki znajdującej się wewnątrz przezroczystego bębna. W czasie obrotu bębna (rys. 4.16) źródło światła wolno przesuwa się od jednego końca do drugiego i jest przeglądana cała fotografia. W przypadku barwnych zdjęć, w celu rozdzielenia różnych barw, przegląda się zdjęcia kilka razy, umieszczając za każdym razem inny filtr przed fotokomórką. W skanerach o największej rozdzielczości wykorzystuje się laserowe źródła światła; rozdzielczość tych skanerów przekracza 2000 punktów na cal.



Rys. 4.16. Fotoskaner. Źródło światła jest odchylane wzdłuż osi bębna i jest mierzona ilość odbitego światła

W innej klasie skanerów są używane długie zestawy elementów CCD, tak zwane *tablice CCD*. Rysunek podlega digitalizacji w czasie przesuwania go pod tablicą elementów CCD, przy czym wielkość przesuwu rysunku jest dostosowana do wymaganej rozdzielczości. Dlatego do obróbki dużego rysunku wystarczy jeden przebieg, który trwa 1 albo 2 min. Rozdzielczość tablic CCD wynosi 200 do 1000 punktów na cal, a więc mniej niż w fotoskanerze.

Rysunki można łatwo skanować za pomocą każdego z opisanych rozwiązań. Trudnym elementem jest wydobycie informacji ze zbioru powstających pikseli. *Wektoryzacja* jest to proces wykrywania odcinków, znaków i innych prymitywów geometrycznych z obrazu rastrowego. Takie zadanie wymaga stosowania odpowiednich algorytmów i w zasadzie jest problemem przetwarzania obrazów, którego rozwiązanie wymaga wykonania kilku kroków. Najpierw do wyeliminowania plam i zabrudzeń oraz wypełnienia przerw wykorzystuje się operacje progowania i uwypuklania krawędzi. Z kolei są stosowane algorytmy wykrywania krawędzi w celu połączenia sąsiednich włączonych pikseli w prymitywy geometryczne, takie jak odcinki. Na drugim poziomie złożoności są stosowane algorytmy rozpoznawania wzorów do łączenia prostych prymitywów w łuki, litery, symbole itd. W celu rozwiązania niejednoznaczności wynikających z przerw w odcinkach, ciemnych zanieczyszczeń i przecięć wielokrotnych odcinków leżących obok siebie może być konieczna ingerencja użytkownika.

Trudniejszy problem polega na zorganizowaniu zbioru prymitywów geometrycznych w sensowne struktury danych. Nieuporządkowany zbiór odcinków nie jest specjalnie przydatny jako wejście do programu typu CAD albo topograficznego. Muszą być rozpoznane konstrukcje geometryczne wysokiego poziomu reprezentowane przez rysunki. Odcinki określające granice państwa powinny być zorganizowane w prymitywy takie jak wielokąty, a mały „+" reprezentujący środek łuku powinien być związany z samym łukiem. Znane są częściowe rozwiązania tych problemów i ciągle pojawiają się coraz lepsze algorytmy.

Niemniej przy trudnych problemach systemy komercyjne odwołują się do interwencji użytkownika.

Zadania

4.1. Jeżeli luminofory o długiej poświacie zmniejszają częstotliwość zlewania się, to dlaczego nie używa się ich powszechnie?

4.2. Napisz program wyświetlania wzorów testowych na monitorze rastrowym. Powinny być uwzględnione trzy wzory: 1) pionowe linie o szerokości 1 piksela, w odstępach co 0, 1, 2 albo 3 piksele; 2) pionowe linie o szerokości 1 piksela w odstępach co 0, 1, 2 albo 3 piksele; 3) siatka jednopikselowych punktów na siatce ze skokiem 5-pikselowym. Każdy wzór powinien być wyświetlany z barwą białą, czerwoną, zieloną albo niebieską oraz w postaci pasków o różnych barwach. Jak to, co się obserwuje przy wyświetlaniu wzorów, ma się do dyskusji na temat rozdzielczości rastra?

4.3. Ile czasu zajmie załadowanie mapy pikselowej 512 × 512 × 1, jeżeli piksele są upakowane po 8 w bajcie i jeżeli bajty mogą być przesyłane i rozpakowywane z szybkością 100 000 bajtów na sekundę? Jak długo potrwa załadowanie mapy bitowej 1024 × 1280 × 1?

4.4. Zaprojektuj sprzętowy układ do konwersji adresów rastra 2D na adresy o postaci bajt i bit w bajcie. Wejścia układu są następujące: 1) adres rastrowy (x, y); 2) baza – adres bajtu pamięci, w którym jest adres rastra (0, 0) w bicie 0; i 3) x_{max}, maksymalny adres rastra x (0 jest to minimum). Wyjściami z układu są: 1) bajt – adres bajtu, który zawiera (x, y) w jednym ze swoich bitów; 2) bit – bit w bajcie, który zawiera (x, y). Jakie są możliwe uproszczenia, jeżeli $x_{max} + 1$ jest potęgą 2?

5. Przekształcenia geometryczne

W rozdziale przedstawiono podstawowe przekształcenia geometryczne 2D i 3D wykorzystywane w grafice komputerowej. Omawiane przekształcenia przesunięcia, skalowania i obrotów są wykorzystywane w wielu pakietach graficznych i będą często wykorzystywane w następnych rozdziałach.

Przekształcenia są wykorzystywane bezpośrednio w programach użytkowych i w wielu podprogramach pakietów graficznych. Program wspomagający planowanie miasta wykorzystywałby przesuwanie do rozmieszczania symboli budynków i drzew na właściwych miejscach, obroty do odpowiedniego zorientowania symboli i skalowanie do dobrania odpowiedniej wielkości symboli. Ogólnie, wiele programów użytkowych wykorzystuje przekształcenia geometryczne do zmieniania pozycji, orientacji i wielkości obiektów (określanych również jako symbole albo szablony) na rysunkach. W rozdziale 6 obroty 3D, przesunięcia 3D i skalowanie 3D zostaną użyte jako elementy procesu tworzenia dwuwymiarowych obrazów obiektów trójwymiarowych. W rozdziale 7 zobaczymy, jak współczesne pakiety graficzne wykorzystują przekształcenia w swoich implementacjach i jak udostępniają je programom użytkowym.

5.1. Podstawy matematyczne

W tym punkcie podano najważniejsze elementy matematyki wykorzystywane w książce – a zwłaszcza wektory i macierze. W żadnym przypadku ten przegląd nie ma na celu pełnej prezentacji algebry liniowej czy

geometrii; nie jest również celem wprowadzanie w te zagadnienia. Przyjmujemy założenie, że czytelnik ma za sobą kurs matematyki równoważny wykładowi na pierwszym roku studiów, ale część wiadomości z algebry i geometrii uleciała mu z pamięci. Jeżeli czytelnik zna zagadnienia omawiane w tym punkcie, to może przejść od razu do p. 5.2. Jeżeli czytelnik nie pamięta opisanych tutaj koncepcji albo jeżeli nie zetknął się z nimi, to mamy nadzieję, że ten punkt będzie mógł służyć jako swego rodzaju „książka kucharska", do której można zajrzeć w trakcie czytania książki. Tych, którzy nie są pewni swoich umiejętności, jeśli chodzi o zrozumienie i stosowanie przypominanych tu koncepcji, możemy zapewnić, że każda operacja wektorowa czy macierzowa jest po prostu skróconym zapisem odpowiedniej postaci algebraicznej. Ponieważ jednak ta notacja jest tak wygodna, namawiamy do poświęcenia czasu na jej poznanie. Stwierdziliśmy, że dobrym narzędziem do nauki jest program *Mathematica*™ [WOLF91], który umożliwia interakcyjne wykonywanie zarówno symbolicznych, jak i numerycznych operacji na wektorach i macierzach. Czytelników zainteresowanych dokładniejszym poznaniem tego materiału odsyłamy do pozycji [BANC83; HOFF61; MARS85].

5.1.1. Wektory i ich właściwości

Być może, pierwszy kontakt z koncepcją wektora nastąpił w ramach kursu fzyki albo mechaniki, na przykład przy okazji omawiania prędkości i kierunku lotu piłki baseballowej w chwili, gdy zostaje uderzona przez kij. W tym momencie stan piłki może być reprezentowany przez odcinek ze strzałką. Odcinek wskazuje kierunek ruchu piłki, a jego długość reprezentuje prędkość. Ten skierowany odcinek reprezentuje *wektor prędkości* piłki. Wektor prędkości to tylko jeden z wielu przykładów wektorów występujących w problemach fizycznych. Inne przykłady wektorów to siła, przyspieszenie i pęd.

Koncepcja wektora okazała się bardzo przydatna w fizyce i w matematyce. W grafice komputerowej wektory są również często wykorzystywane. Stosujemy je do reprezentowania położenia punktu w zbiorze danych we współrzędnych świata, do oznaczania orientacji powierzchni w przestrzeni, do opisu zachowania światła przy kontakcie z bryłami przezroczystymi i nieprzezroczystymi i do wielu innych celów. W następnych rozdziałach często będziemy mieli do czynienia z wektorami. W miarę opisywania ich właściwości będziemy pokazywali, gdzie i jak wykorzystywać wektory.

Chociaż w niektórych podręcznikach wektory są opisywane wyłącznie z punktu widzenia geometrycznego, będziemy zwracali uwagę na ich dwoistą naturę i wykorzystywali również ich definicje algebraiczne.

Takie podejście prowadzi nas do zwartych i zwięzłych sformułowań algebry liniowej, którą faworyzujemy jako narzędzie matematyczne grafiki komputerowej. Będziemy jednak podawać interpretacje geometryczne wszędzie tam, gdzie ułatwi to zrozumienie koncepcji.

Chociaż są bardziej ścisłe definicje, dla naszych potrzeb ograniczymy się do stwierdzenia, że *wektor* jest to n-tka liczb rzeczywistych, przy czym n jest równe 2 dla przestrzeni 2D, 3 dla przestrzeni 3D itd. Wektory będziemy oznaczali literami pisanymi kursywą[1], w tym rozdziale zazwyczaj jako u, v albo w. Dla wektorów są określone dwie operacje: dodawanie wektorów i mnożenie wektorów przez liczby rzeczywiste, określane jako *mnożenie skalarne*[2]. Te operacje mają pewne właściwości. Dodawanie jest przemienne i łączne oraz istnieje wektor, tradycyjnie określany jako 0, o takiej właściwości, że dla dowolnego wektora v, $0 + v = v$. Jest również operacja odwrotna (to znaczy, dla każdego wektora v jest inny wektor w o takiej właściwości, że $v + w = 0$; w jest zapisywane jako „$-v$"). Dla mnożenia skalarnego są spełnione następujące reguły: $(\alpha\beta)v = \alpha(\beta v)$, $1v = v$, $(\alpha + \beta)v = \alpha v + \beta v$ oraz $\alpha(v + w) = \alpha v + \alpha w$.

Dodawanie jest definiowane na poziomie składowych, podobnie jak mnożenie przez skalar. Wektory są zapisywane kolumnowo; przykładowy wektor 3D jest zapisywany jako

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

Można sumować składowe tak jak w następującym przykładzie:

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 7 \end{bmatrix}$$

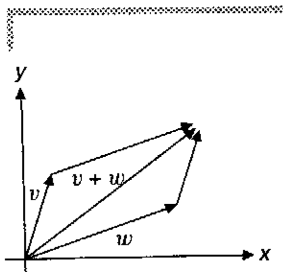
Większość zadań grafiki jest wykonywana w przestrzeni 2D, przestrzeni 3D albo, jak to zobaczymy w/p. 5.7 i 6.6.4, w przestrzeni 4D.

W tym miejscu jest ważne, żeby rozróżnić wektory i punkty. Możemy przyjąć, że punkt jest zdefiniowany przez wektor kończący się w nim; taka reprezentacja wymaga określenia, skąd wektor jest

[1] W dalszej części książki dołożyliśmy starań, żeby stosowane konwencje notacyjne były przystosowane do przyjętych w literaturze naukowej; chociaż oczywiście nie zawsze taka zgodność występuje we wszystkich pracach. Ze względu na te różnice, czasami do oznaczenia wektorów są stosowane duże litery, a czasami małe.

[2] Skalary (to znaczy liczby rzeczywiste) będą oznaczane literami greckimi, głównie z początku alfabetu.

poprowadzony, to znaczy trzeba zdefiniować coś, co zazwyczaj jest określane jako początek układu współrzędnych. Jednak na punktach można wykonywać wiele operacji, które nie wymagają określania początku układu współrzędnych; są operacje wektorowe, na przykład mnożenie, które nie są zdefiniowane w odniesieniu do punktów. Dlatego najlepiej jest nie mieszać punktów i wektorów i traktować je jako oddzielne koncepcje. Niemniej są operacje – np. tworzenie wektora będącego różnicą dwóch punktów – które zawsze mają sens.



Rys. 5.1. Dodawanie wektorów na płaszczyźnie

Po tym zwróceniu uwagi na różnice między wektorami i punktami możemy podać użyteczne właściwości traktując punkty jako wektory. Jako przykład rozważmy przestrzeń 2D o określonym początku układu współrzędnych. Dodawanie wektorów możemy określić za pomocą znanej reguły równoległoboku: w celu dodania wektorów v i w prowadzimy strzałkę od początku układu do w i przesuwamy ją tak, żeby jej początek znalazł się w punkcie v i definiujemy $v + w$ jako nowy koniec strzałki. Jeżeli narysujemy również strzałkę z początku układu współrzędnych do v i powtórzymy powyższe operacje, to otrzymamy równoległobok (rys. 5.1). Mnożenie skalarne przez liczbę rzeczywistą α jest definiowane podobnie: rysujemy strzałkę z początku układu współrzędnych do punktu v i wydłużamy ją α razy, przy czym jej początek zostaje w początku układu współrzędnych; wtedy αv jest definiowane przez koniec otrzymanej strzałki. Oczywiście taką samą definicję można podać dla przestrzeni 3D.

Mając takie dwie operacje zdefiniowane w przestrzeni wektorów, można wykonywać inne obliczenia na wektorach. Jedno z takich obliczeń to tworzenie *kombinacji liniowej*. Kombinacja liniowa wektorów $v_1, ..., v_n$ jest to dowolny wektor o postaci $\alpha_1 v_1 + \alpha_2 v_2 + ... + \alpha_n v_n$. Kombinacje liniowe wektorów są używane do opisywania wielu obiektów. W rozdziale 9 opisano zastosowania kombinacji liniowej do reprezentowania krzywych i powierzchni.

5.1.2. Iloczyn skalarny

Dla dwóch n-wymiarowych wektorów

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \text{oraz} \quad \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

definiujemy *iloczyn skalarny* jako $x_1 y_1 + ... + x_n y_n$. Iloczyn skalarny wektorów v i w jest oznaczany jako $v \cdot w$.

Odległość punktu (x, y) na płaszczyźnie od początku układu współrzędnych wynosi $\sqrt{x^2 + y^2}$. Ogólnie odległość punktu $(x_1, ..., x_n)$ od początku układu współrzędnych wynosi $\sqrt{x_1^2 + ... + x_n^2}$. Jeżeli v jest wektorem

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

to odległość tę możemy zapisać jako $\sqrt{v \cdot v}$. Jest to nasza definicja długości n-wymiarowego wektora. Oznaczamy tę długość jako $\|v\|$. Odległość między dwoma punktami w standardowej n-przestrzeni jest definiowana podobnie: odległość między P i Q jest to długość wektora $Q - P$.

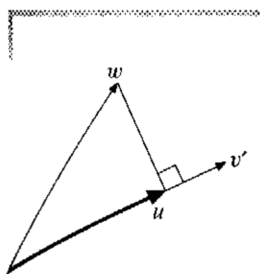
5.1.3. Właściwości iloczynu skalarnego

Iloczyn skalarny ma kilka interesujących właściwości. Po pierwsze, jest *symetryczny*: $v \cdot w = w \cdot v$. Po drugie, jest *niedegenerujący*: $v \cdot v = 0$ tylko wówczas, gdy $v = 0$. Po trzecie, jest *biliniowy*: $v \cdot (u + \alpha w) = v \cdot u + \alpha(v \cdot w)$.

Iloczyn skalarny może być używany do generowania wektorów o długości 1 (jest to tak zwane *normalizowanie wektora*). W celu znormalizowania wektora v po prostu obliczamy $v' = v/\|v\|$. Otrzymany wektor ma długość 1 i jest nazywany *wektorem jednostkowym.*

Iloczyn skalarny może być również wykorzystany do mierzenia kątów. Kąt między wektorami v i w jest równy

$$\cos^{-1}\left(\frac{v \cdot w}{\|v\|\|w\|}\right)$$



Rys. 5.2. Rzut w na wektor jednostkowy v' jest wektorem u o długości $\|w\|$ razy kosinus kąta między u i v'

Zauważmy, że jeżeli v i w są wektorami jednostkowymi, to dzielenie nie jest potrzebne.

Jeżeli mamy wektor jednostkowy v' i inny wektor w i zrzutujemy w prostopadle na v', tak jak na rys. 5.2, a wynik oznaczymy przez u, to długość u powinna być długością w pomnożoną przez $\cos\theta$, przy czym θ jest kątem między v i w. Ponieważ długość v jest równa 1, otrzymujemy

$$\|u\| = \|w\| \cos\theta = \|w\| \left(\frac{v \cdot w}{\|v\| \; \|w\|}\right) = v \cdot w$$

Daje to nam nową interpretację iloczynu skalarnego; iloczyn skalarny wektorów v i w jest równy długości rzutu w na v przy założeniu, że v jest wektorem jednostkowym. Spotkamy się z wieloma zastosowaniami iloczynu skalarnego, zwłaszcza w rozdz. 14, w którym jest wykorzystywany do opisania, jak światło oddziałuje z powierzchniami.

5.1.4. Macierze

Macierz jest to prostokątna tablica liczb, z której często korzystamy przy operacjach na punktach i wektorach. O macierzy możemy myśleć jak o reprezentacji reguły przekształcania jej argumentów przez przekształcenie liniowe. Jej elementy – na ogół liczby rzeczywiste – mają podwójne indeksy i na zasadzie umowy pierwszy indeks oznacza wiersz, a drugi kolumnę. Zgodnie z konwencją matematyczną indeksy zaczynają się od 1; w niektórych językach programowania indeksy zaczynają się od 0. Zostawiamy programistom korzystającym z tych języków przesuwanie wszystkich indeksów o 1. Jeżeli A jest macierzą, to $a_{3,2}$ oznacza element w trzecim wierszu i drugiej kolumnie. Gdy używa się symbolicznych oznaczeń, na przykład a_{ij}, to przecinek między nimi jest pomijany.

Wektor w n-przestrzeni, który zapisywaliśmy w postaci

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

może być traktowany jako macierz $n \times 1$ i jest określany jako *wektor kolumnowy*. W znacznej części książki będziemy korzystali z macierzy i z odpowiednich operacji (p. 5.1.5–5.1.8). Macierze odgrywają istotną rolę w przekształceniach geometrycznych (ten rozdział), w rzutowaniu 3D (rozdz. 6), w pakietach grafiki 3D (rozdz. 7) i w opisie krzywych i powierzchni (rozdz. 9).

5.1.5. Mnożenie macierzy

Macierze mnoży się zgodnie z następującą regułą: jeżeli A jest macierzą $n \times m$ z elementami a_{ij} i B jest macierzą $m \times p$ z elementami b_{ij}, to istnieje AB i jest to macierz $n \times p$ z elementami c_{ij}, przy czym $c_{ij} = \sum_{s=1}^{m} a_{is} b_{sj}$. Jeżeli myślimy o kolumnach B jak o wektorach, $B_1, \ldots, B_p$ i wierszach A jak o wektorach $A_1, \ldots, A_n$ (ale obróconych o 90°, tak żeby

stały się poziome), to możemy zauważyć, że c_{ij} jest równe $A_i \cdot B_j$. Obowiązują zwykłe właściwości mnożenia poza tym, że mnożenie nie jest przemienne: AB na ogół różni się od BA. Mnożenie jest rozłączne względem dodawania: $A(B + C) = AB + AC$; jest również element identycznościowy dla mnożenia, a mianowicie *macierz jednostkowa* – jest to macierz kwadratowa, której wszystkie elementy są równe 0 poza elementami, które leżą na przekątnej i są równe 1 (to znaczy elementami są δ_{ij}, przy czym $\delta_{ij} = 0$ dla $i \neq j$, a $\delta_{ii} = 1$). Przykład 5.1 ilustruje mnożenie macierzy.

5.1.6. Wyznaczniki

Wyznacznik macierzy kwadratowej jest to jedna liczba utworzona z elementów macierzy. Obliczenie wyznacznika jest nieco kłopotliwe ze względu na rekurencyjną definicję. Wyznacznik macierzy 2×2 $\begin{bmatrix} a & c \\ b & d \end{bmatrix}$ jest równy po prostu $ad - bc$. Wyznacznik macierzy $n \times n$ jest zdefiniowany w zależności od wyznaczników mniejszych macierzy. Niech A_{1i} oznacza wyznacznik macierzy $(n - 1) \times (n - 1)$, którą otrzymujemy po usunięciu pierwszego wiersza oraz i-tej kolumny z macierzy A o wymiarze $n \times n$. Wtedy wyznacznik A jest zdefiniowany jako

$$\det A = \sum_{i=1}^{n} (-1)^{1+i} A_{1i}$$

Jedno z zastosowań wyznacznika w przestrzeni 3D występuje przy obliczaniu *iloczynu wektorowego*. Iloczyn wektorowy dwóch wektorów

$$v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \text{oraz} \quad w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

obliczamy wyznaczając wyznacznik macierzy

$$\begin{bmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix}$$

przy czym i, j oraz k reprezentują wektory jednostkowe skierowane wzdłuż trzech osi współrzędnych. Wynikiem jest liniowa kombinacja zmiennych i, j oraz k dająca wektor

$$\begin{bmatrix} v_2w_3 - v_3w_2 \\ v_3w_1 - v_1w_3 \\ v_1w_2 - v_2w_1 \end{bmatrix}$$

który jest oznaczony jako $v \times w$. Wektor ten jest prostopadły do płaszczyzny określonej przez wektory v i w, a jego długość jest równa iloczynowi $\|v\| \, \|w\| \, |\sin\theta|$, przy czym θ jest kątem między v i w. Właściwości iloczynu wektorowego wykorzystamy w rozdz. 9, w którym pokażemy, że może on być stosowany do określenia równania płaszczyzny wielokąta.

5.1.7. Transpozycja macierzy

Macierz $n \times k$ może być odbita zwierciadlanie wokół przekątnej dając w wyniku macierz $k \times n$. Jeżeli elementami pierwszej macierzy są elementy a_{ij} ($i = 1, ..., n; j = 1, ..., k$), to elementami wynikowej macierzy są elementy b_{ij} ($i = 1, ..., k, j = 1, ..., n$), przy czym $b_{ij} = a_{ji}$. Ta nowa macierz jest nazywana *macierzą transponowaną* w stosunku do oryginalnej macierzy. Transpozycję macierzy A zapisujemy jako A^T. Jeżeli potraktujemy wektor w n-wymiarowej przestrzeni jako macierz $n \times 1$, to macierz transponowana jest macierzą $1 \times n$ (czasami używa się nazwy wektor wierszowy). Korzystając z pojęcia transpozycji możemy otrzymać nowy opis iloczynu skalarnego; a mianowicie $u \cdot v = u^T v$.

5.1.8. Odwracanie macierzy

Mnożenie macierzy różni się od zwykłego mnożenia tym, że nie jest przemienne. Odwracanie macierzy jest definiowane tylko dla macierzy kwadratowych, przy czym nie wszystkie takie macierze mają macierz odwrotną. Mówiąc dokładniej, macierz odwrotną mają tylko takie macierze kwadratowe, których wyznaczniki są różne od zera.

Jeżeli A i B są macierzami $n \times n$ i $AB = BA = I$, przy czym I jest macierzą jednostkową $n \times n$, to można powiedzieć, że macierz B jest odwrotna względem macierzy A, co się zapisuje jako A^{-1}. Dla macierzy $n \times n$, której elementami są liczby rzeczywiste, wystarczy pokazać, że albo $AB = I$ albo $BA = I$ – jeżeli jedno jest prawdziwe, to drugie również.

Jeżeli mamy macierz $n \times n$, to preferuje się sposób znajdowania macierzy odwrotnej metodą eliminacji Gaussa, zwłaszcza dla dowolnej macierzy większej niż 3×3. Metoda ta jest dobrze opisana w pracy [PRES88]; tam też podano działające programy.

Przykład 5.1 W punkcie 5.7 poznamy znaczenie macierzy 4 × 4 w grafice komputerowej; są one powszechnie używane przy przekształceniach 3D.

Problem: a. Znajdź iloczyn macierzy $C = AB$ dla macierzy

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/k & 1 \end{bmatrix} \quad \text{oraz} \quad B = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & m \\ -\sin\theta & 0 & \cos\theta & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Jak zobaczymy w p. 6.5 i 5.7, te macierze określają rzut perspektywiczny, obrót i dwa przesunięcia.

b. Napisz program w języku C, który wyznacza iloczyn C dwóch macierzy 4 × 4 A i B.

Odpowiedź a. Regułę mnożenia macierzy podaną w p. 5.1.5 można zilustrować w następujący sposób:

$$c_{ij} = \sum_{s=1}^{m} a_{is} b_{sj} \qquad \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

Pokazano moment wyznaczania elementu c_{43}. Z równania mnożenia wynika, że powinniśmy pomnożyć elementy z czwartego wiersza macierzy A przez elementy z trzeciej kolumny macierzy B. W wyniku tej operacji otrzymujemy, że $c_{43} = \cos\theta/k$. Stosując tę procedurę do każdego elementu C otrzymamy następujący wynik:

$$C = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & m \\ 0 & 0 & 0 & 0 \\ -\sin\theta/k & 0 & \cos\theta/k & n/k + 1 \end{bmatrix}$$

b.
```
typedef struct Matrix4Struct {
    double element[4][4];
}Matrix4;

/* mnożenie macierzy c = ab */
/* zauważmy, że c nie musi wskazywać na żadną z macierzy wejściowych */
Matrix4 *V3MatMul(a, b, c)
Matrix4 *a, *b, *c;
{
    int    i, j, k;
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4, j++) {
            c->element[i][j] = 0.0;
            for (k = 0; k < 4; k++)
                c->element[i][j] +=
                    a->element[i][k] * b->element[k][j];
        }
    }
    return (c);
}
```

5.2. Przekształcenia 2D

Punkty na płaszczyźnie (x, y) możemy przesunąć na nową pozycję dodając do współrzędnych punktów wielkość przesunięcia. Dla każdego punktu $P(x, y)$, który ma być przesunięty do nowego punktu $P'(x', y')$ o d_x jednostek wzdłuż osi x i o d_y jednostek wzdłuż osi y, możemy napisać:

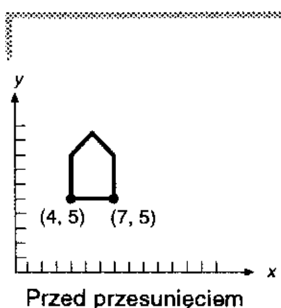
$$x' = x + d_x, \qquad y' = y + d_y \tag{5.1}$$

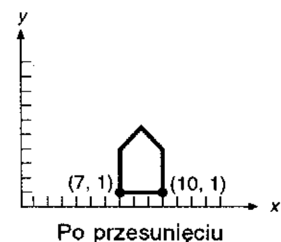
Jeżeli zdefiniujemy wektory kolumnowe:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad T = \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{5.2}$$

to równanie (5.1) może być wyrażone w bardziej zwarty sposób jako

$$P' = P + T \tag{5.3}$$





Rys. 5.3. Przesuwanie domu

Obiekt powinniśmy przesuwać stosując równanie (5.3) do każdego punktu obiektu. Ponieważ jednak każdy odcinek obiektu składa się z nieskończonej liczby punktów, taki proces trwałby nieskończenie długo. Na szczęście możemy przesunąć wszystkie punkty odcinka przesuwając tylko jego końce i rysując nowy odcinek między przesuniętymi końcami; ta obserwacja odnosi się także do skalowania (rozciągania) i obrotów. Na rysunku 5.3 pokazano efekt przesuwania konturu domu o (3, −4).

Punkty mogą być *skalowane* ze współczynnikiem s_x wzdłuż osi x i ze współczynnikiem s_y wzdłuż osi y przez mnożenie

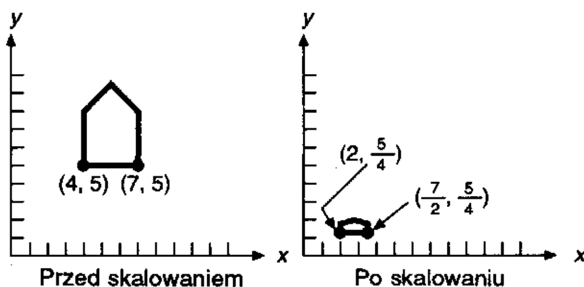
$$x' = s_x \cdot x, \qquad y' = s_y \cdot y \tag{5.4}$$

W postaci macierzowej można to zapisać następująco:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{albo} \quad P' = S \cdot P \tag{5.5}$$

przy czym S jest macierzą w równaniu (5.5).

Na rysunku 5.4 dom jest skalowany ze współczynnikiem 1/2 w kierunku osi x i ze współczynnikiem 1/4 w kierunku osi y. Zauważmy, że skalowanie odbywa się względem początku układu współrzędnych: dom zmniejsza się i jest bliżej początku układu współrzędnych. Gdyby współczynnik skalowania był większy niż 1, wówczas dom byłby większy i znajdowałby się dalej od początku układu współrzędnych. Metody skalowania względem punktu innego niż początek układu współrzędnych są omawiane w p. 5.3. Proporcje domu również zmieniły się, ponieważ dokonaliśmy skalowania *niejednorodnego*, dla którego $s_x \neq s_y$. Przy skalowaniu *jednorodnym*, dla którego $s_x = s_y$, proporcje nie ulegają zmianie.



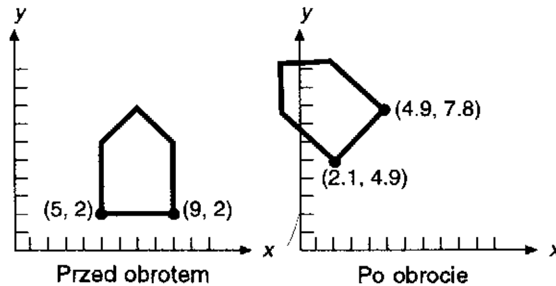
Rys. 5.4. Skalowanie domu. Skalowanie jest niejednorodne i dom zmienia położenie

Punkty mogą być obracane o kąt θ wokół początku układu współrzędnych. Matematycznie obrót jest zdefiniowany następująco:

$$x' = x \cdot \cos\theta - y \cdot \sin\theta, \quad y' = x \cdot \sin\theta + y \cdot \cos\theta \tag{5.6}$$

W postaci macierzowej można to zapisać następująco:

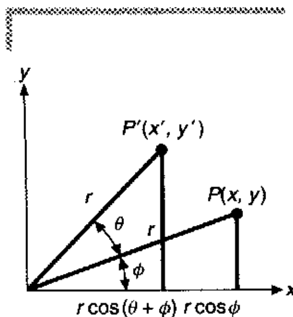
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{albo} \quad P' = R \cdot P \tag{5.7}$$



Rys. 5.5. Obrót domu. Dom zmienia również pozycję

przy czym R w równaniu (5.7) jest macierzą obrotu. Na rysunku 5.5 pokazano obrót domu o kąt 45°. Podobnie jak dla skalowania obrót następuje względem początku układu współrzędnych; obrót wokół dowolnego punktu omówiono w p. 5.3.

Kąty dodatnie są mierzone w kierunku przeciwnym względem kierunku ruchu wskazówek zegara od x do y. Dla kątów ujemnych (zgodnych z kierunkiem ruchu wskazówek zegara) można w równaniach (5.6) i (5.7) skorzystać z tożsamości $\cos(-\theta) = \cos\theta$ oraz $\sin(-\theta) = -\sin\theta$.



Rys. 5.6. Wyprowadzenie równania obrotu

Równanie (5.6) można łatwo wyprowadzić, korzystając z rys. 5.6, na którym obrót o θ przekształca punkt $P(x, y)$ w punkt $P'(x', y')$. Ponieważ obrót następuje wokół początku układu współrzędnych, odległości od początku układu współrzędnych do P i P' są sobie równe i są oznaczone na rysunku przez r. Korzystając z prostych przekształceń trygonometrycznych otrzymujemy

$$x = r \cdot \cos\Phi, \quad y = r \cdot \sin\Phi \tag{5.8}$$

oraz

$$\left.\begin{aligned} x' &= r \cdot \cos(\theta + \Phi) = r \cdot \cos\Phi \cdot \cos\theta - r \cdot \sin\Phi \cdot \sin\theta \\ y' &= r \cdot \sin(\theta + \Phi) = r \cdot \cos\Phi \cdot \sin\theta + r \cdot \sin\Phi \cdot \cos\theta \end{aligned}\right\} \tag{5.9}$$

Po podstawieniu równania (5.8) do równania (5.9) otrzymamy równanie (5.6).

5.3. Współrzędne jednorodne i macierzowa reprezentacja przekształceń 2D

Reprezentacje macierzowe przekształceń przesunięcia, skalowania i obrotu mają następującą postać:

$$P' = T + P \tag{5.3}$$

$$P' = S \cdot P \tag{5.5}$$

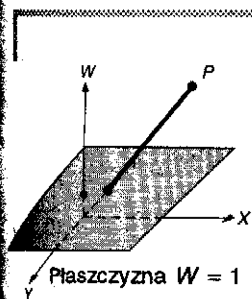
$$P' = R \cdot P \tag{5.7}$$

Niestety przesunięcie jest traktowane inaczej niż skalowanie i obrót. Chcielibyśmy móc traktować wszystkie trzy przekształcenia w jednolity sposób, tak żeby można je było łatwo łączyć ze sobą.

Jeżeli punkty są wyrażone we *współrzędnych jednorodnych*, to wszystkie trzy przekształcenia można traktować jako mnożenia. Współrzędne jednorodne zostały wprowadzone po raz pierwszy w geometrii [MAXW46;MAXW51], a potem zostały zastosowane w grafice [ROBE65;BLIN77b;BLIN78a]. Liczne pakiety graficzne i procesory wyświetlania korzystają ze współrzędnych i przekształceń jednorodnych.

We współrzędnych jednorodnych dodajemy trzecią współrzędną dla punktu. Punkt reprezentowany przez parę liczb (x, y) jest teraz reprezentowany przez trójkę (x, y, W). Mówimy, że dwa zestawy współrzędnych jednorodnych (x, y, W) i (x', y', W') reprezentują ten sam punkt wtedy i tylko wtedy, gdy jeden jest wielokrotnością drugiego. I tak (2, 3, 6) i (4, 6, 12) są tymi samymi punktami reprezentowanymi przez różne trójki współrzędnych. To znaczy, że każdy punkt ma wiele różnych reprezentacji we współrzędnych jednorodnych. Ponadto, przynajmniej jedna ze współrzędnych jednorodnych musi być różna od zera: reprezentacja (0, 0, 0) nie jest dopuszczalna. Jeżeli współrzędna W jest różna od zera, to możemy przez nią podzielić: (x, y, W) reprezentuje ten sam punkt co $(x/W, y/W, 1)$. Jeżeli W nie jest równe 0, to dzielenie wykonujemy normalnie i liczby x/W i y/W są nazywane współrzędnymi kartezjańskimi punktu jednorodnego. Punkty z $W = 0$ są nazywane punktami w nieskończoności i rzadko będą pojawiały się w naszych dyskusjach.

Trójki współrzędnych na ogół reprezentują punkty w przestrzeni trójwymiarowej, tutaj natomiast używamy ich do reprezentowania punktów w przestrzeni dwuwymiarowej. Powiązanie jest następujące: jeżeli weźmiemy wszystkie trójki reprezentujące ten sam punkt – to jest trójki o postaci (tx, ty, tW) przy $t \neq 0$ – otrzymamy linię w przestrzeni trójwymiarowej. Tak więc każdy *punkt* jednorodny reprezentuje *linię* w przestrzeni trójwymiarowej. Jeżeli punkt przedstawimy w postaci jednorodnej (podzielimy przez W), to otrzymamy punkt o postaci $(x, y, 1)$. Tak więc punkty jednorodne tworzą płaszczyznę zdefiniowaną przez równanie $W = 1$ w przestrzeni (x, y, W). Na rysunku 5.7 pokazano tę zależność. Punkty w nieskończoności nie są reprezentowane na tej płaszczyźnie.



Rys. 5.7. Przestrzeń XYW współrzędnych jednorodnych z płaszczyzną $W = 1$ i punktem $P(X, Y, W)$ zrzutowanym na płaszczyznę $W = 1$

Ponieważ punkty są teraz trzyelementowymi wektorami kolumnowymi, macierz przekształcenia, przez którą się mnoży, musi być macierzą 3 × 3. Równania (5.1) przekształcenia typu przesunięcie przyjmują we współrzędnych jednorodnych postać

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.10}$$

Zwracamy uwagę, że w niektórych podręcznikach z zakresu grafiki komputerowej, w tym w pracy [FOLE82], jest stosowana konwencja mnożenia wektorów wierszowych przez macierze zamiast mnożenia macierzy przez wektory kolumnowe. Przy przejściu od jednej konwencji do drugiej trzeba dokonać transponowania macierzy

$$(P \cdot M)^{\mathrm{T}} = M^{\mathrm{T}} \cdot P^{\mathrm{T}}$$

Równanie (5.10) można zapisać inaczej w postaci

$$P' = T(d_x, d_y) \cdot P \tag{5.11}$$

przy czym

$$T(d_x, d_y) = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.12}$$

Co się stanie, jeżeli punkt P jest przesuwany o $T(d_{x1}, d_{y1})$ do P', a potem o $T(d_{x2}, d_{y2})$ do P''? Wynik, którego się spodziewamy intuicyjnie, to łączne przesunięcie $T(d_{x1} + d_{x2}, d_{y1} + d_{y2})$. Dla potwierdzenia tego przypuszczenia zaczynamy od danych początkowych:

$$P' = T(d_{x1}, d_{y1}) \cdot P \tag{5.13}$$

$$P'' = T(d_{x2}, d_{y2}) \cdot P' \tag{5.14}$$

Teraz po podstawieniu równania (5.13) do równania (5.14) otrzymujemy

$$P'' = T(d_{x2}, d_{y2})(T(d_{x1}, d_{y1}) \cdot P) = (T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})) \cdot P \tag{5.15}$$

Iloczyn macierzy $T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})$ jest następujący:

$$\begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_{x1} + d_{x2} \\ 0 & 1 & d_{y1} + d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \tag{5.16}$$

Końcowe przesunięcie jest rzeczywiście równe $T(d_{x1} + d_{x2}, d_{y1} + d_{y2})$. Iloczyn macierzy jest czasami określany jako złożenie albo konkatenacja $T(d_{x1}, d_{y1})$ i $T(d_{x2}, d_{y2})$. Tutaj na ogół będziemy korzystali z określenia *złożenie*.

Podobnie równanie skalowania (5.4) w postaci macierzowej przyjmuje postać

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.17}$$

Definiując

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.18}$$

otrzymujemy

$$P' = S(s_x, s_y) \cdot P \tag{5.19}$$

Kolejne przesunięcia są addytywne, tutaj natomiast spodziewamy się, że kolejne skalowania powinny być multyplikatywne. Jeżeli są dane:

$$P' = S(s_{x1}, s_{y1}) \cdot P \tag{5.20}$$

$$P'' = S(s_{x2}, s_{y2}) \cdot P' \tag{5.21}$$

to po podstawieniu równania (5.20) do równania (5.21) otrzymujemy

$$P'' = S(s_{x2}, s_{y2}) \cdot (S(s_{x1}, s_{y1}) \cdot P) = (S(s_{x2}, s_{y2}) \cdot (S(s_{x1}, s_{y1})) \cdot P \tag{5.22}$$

Iloczyn macierzy $S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})$ jest równy

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1}s_{x2} & 0 & 0 \\ 0 & s_{y1}s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.23}$$

A więc skalowanie jest rzeczywiście multyplikatywne.

Wreszcie równanie obrotu (5.6) może być reprezentowane jako

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.24}$$

Oznaczając

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.25}$$

otrzymujemy

$$P' = R(\theta) \cdot P \tag{5.26}$$

Wykazanie, że dwa kolejne obroty są addytywne, zostawiamy jako zadanie 5.2.

W górnej lewej podmacierzy 2×2 z równania (5.25) potraktujmy każdy z dwóch wierszy jako wektor. Można wykazać, że te wektory mają następujące trzy właściwości:

1. Każdy jest wektorem jednostkowym.
2. Każdy jest prostopadły do drugiego (ich iloczyn skalarny jest równy 0).
3. Na to, żeby wektory pierwszy i drugi leżały odpowiednio na osiach x i y, muszą zostać obrócone o $R(\theta)$ (przy spełnieniu warunków 1 i 2 ta właściwość jest równoważna temu, że podmacierz ma wyznacznik równy 1).

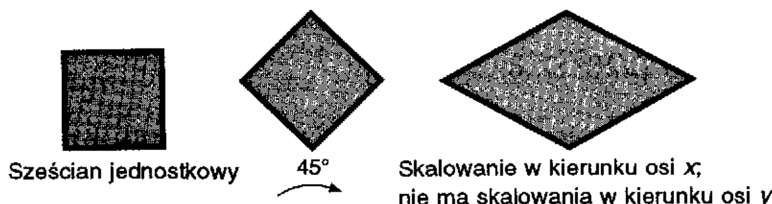
Pierwsze dwie właściwości są również prawdziwe dla kolumn podmacierzy 2×2. Te dwa kierunki, o których mowa, to te, na które są obracane wektory osi dodatnich x i y. Te właściwości sugerują dwie użyteczne metody wyznaczania macierzy obrotu, gdy znamy pożądany efekt obrotu. Macierz o takich właściwościach jest określana jako *ortonormalna*.

Macierz przekształcenia o postaci

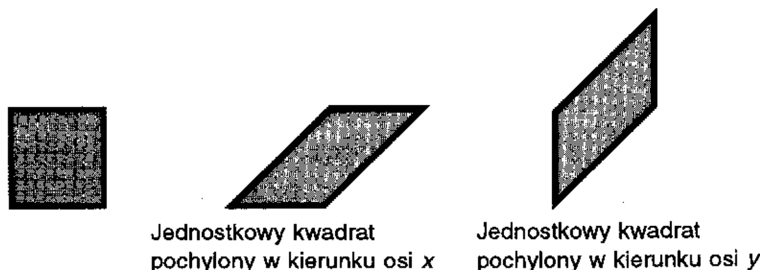
$$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.27}$$

przy czym górna podmacierz 2×2 jest ortogonalna, zachowuje kąty i długości. To znaczy, kwadrat jednostkowy pozostaje kwadratem jednostkowym i nie staje się rombem o boku jednostkowym ani kwadratem o boku różnym od jednostki. Takie przekształcenia są również określane jako przekształcenia *ciała sztywnego*, ponieważ ciało albo obiekt poddawane przekształceniu w żaden sposób nie jest odkształcane. Dowolne złożenie macierzy obrotu i przesunięcia tworzy macierz o tej postaci.

Co można powiedzieć o iloczynie dowolnej sekwencji macierzy obrotu, przesunięcia i skalowania? Są one określane jako *przekształcenia afiniczne* i mają właściwość zachowania równoległości linii; nie odnosi się to do długości i kątów. Na rysunku 5.8 pokazano przykład obrotu jednostkowego kwadratu o $-45°$, a potem skalowania niejednorodnego. Widać, że ani kąty, ani długości nie zostały zachowane w wyniku tej sekwencji, natomiast odcinki równoległe pozostały równoległe. Dalsze operacje obrotu, skalowania i przesuwania nie spowodują tego, że odcinki równoległe przestaną być równoległe. Przekształcenia $R(\theta)$, $S(s_x, s_y)$ i $T(d_x, d_y)$ są również afiniczne.



Rys. 5.8. Jednostkowy kwadrat jest obracany o $-45°$ i jest skalowany niejednorodnie. Wynikiem jest afiniczne przekształcenie jednostkowego sześcianu, w którym linie równoległe pozostają równoległe, natomiast kąty i długości nie zostają zachowane



Rys. 5.9. Wynik zastosowania operacji pochylenia do jednostkowego kwadratu. W każdym przypadku długość ukośnych boków jest większa niż 1

Innym przekształceniem podstawowym jest *przekształcenie pochylające*; jest to również przekształcenie afiniczne. Na płaszczyźnie są dwa rodzaje przekształceń pochylających: pochylenie wzdłuż osi x i pochylenie wzdłuż osi y. Na rysunku 5.9 pokazano efekt pochylenia jednostkowego kwadratu wzdłuż obu osi. Operacja pochylania jest reprezentowana przez macierz

$$SH_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.28}$$

Wyraz a w macierzy pochylania jest współczynnikiem proporcjonalności. Zauważmy, że iloczyn $SH_x[x\ y\ 1]^T$ jest równy $[x + ay\ y\ 1]^T$, co demonstruje proporcjonalność zmiany x w funkcji y.

Podobnie macierz

$$SH_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.29}$$

pochyla wzdłuż osi y.

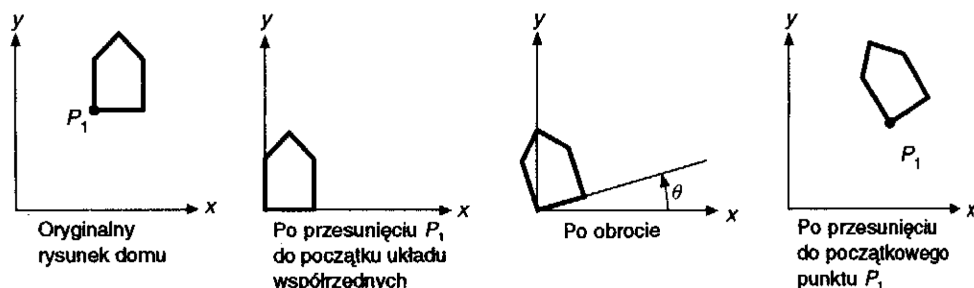
5.4. Składanie przekształceń 2D

Idea składania została wprowadzona w p. 5.3. Tutaj stosujemy składanie do łączenia podstawowych macierzy R, S i T w celu uzyskania pożądanego wyniku. Podstawowym celem składania przekształceń jest zwiększenie efektywności – zamiast stosować ciąg przekształceń jedno po drugim, można stosować jedną macierz złożoną.

Rozważmy obrót obiektu wokół pewnego dowolnego punktu P_1. Ponieważ wiemy tylko, jak wykonywać obrót wokół początku układu współrzędnych, zamieniamy nasz oryginalny (trudny) problem na trzy oddzielne (łatwe) problemy. Dlatego, żeby obrócić punkt wokół P_1, potrzebujemy sekwencji trzech przekształceń:

1. Takie przesunięcie, żeby punkt P_1 znalazł się w początku układu współrzędnych.
2. Obrót.
3. Takie przesunięcie, żeby punkt znajdujący się w początku układu współrzędnych wrócił do P_1.

Ta sekwencja jest zilustrowana na rys. 5.10, na którym dom zostaje obrócony wokół $P_1(x_1, y_1)$. Pierwsze przesunięcie jest o $(-x_1, -y_1)$,



Rys. 5.10. Obracanie domu wokół punktu P_1 o kąt θ

a drugie odwrotnie o (x_1, y_1). Wynik jest różny od tego, jaki powstałby w wyniku zastosowania tylko obrotu.

Całe przekształcenie wygląda następująco:

$$T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot$$

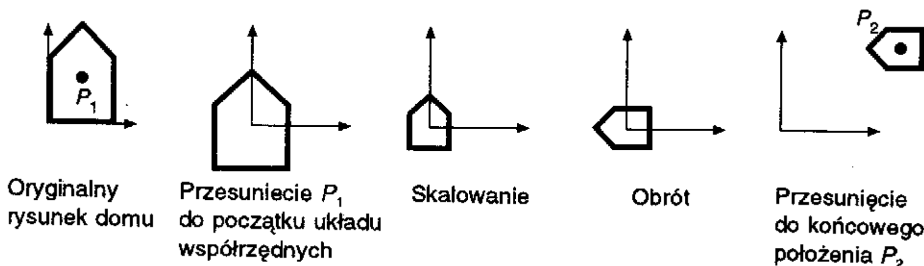
$$\cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_1(1-\cos\theta) + y_1\sin\theta \\ \sin\theta & \cos\theta & y_1(1-\cos\theta) - x_1\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{5.30}$$

Podobne podejście jest wykorzystane przy skalowaniu obiektu wokół dowolnego punktu P_1. Najpierw dokonujemy takiego przesunięcia, żeby punkt P_1 znalazł się w początku układu współrzędnych, potem wykonujemy skalowanie i ponownie przesunięcie do P_1. W tym przypadku całkowite przkształcenie ma postać

$$T(x_1, y_1) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) =$$

$$= \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_1(1-s_x) \\ 0 & s_y & y_1(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \tag{5.31}$$

Załóżmy, że chcielibyśmy dokonać skalowania, obrotu i przesunięcia zarysu domu pokazanego na rys. 5.11, z punktem P_1 jako środkiem obrotu i skalowania. Sekwencja działań jest następująca: przesunięcie



Rys. 5.11. Obrót rysunku domu dookoła punktu P_1 i takie przesunięcie, żeby punkt P_1 znalazł się w punkcie P_2

punktu P_1 do początku układu współrzędnych, skalowanie i obrót, przesunięcie z początku układu współrzędnych do nowej pozycji P_2. Struktura danych, która pamięta to przekształcenie, mogłaby zawierać współczynnik(i) skalowania, kąt obrotu i wielkość przesunięcia oraz kolejność wykonywania przekształceń albo też mogłaby po prostu zawierać złożoną macierz przekształcenia

$$T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) \tag{5.32}$$

Jeżeli M_1 i M_2 reprezentują podstawowe przekształcenia przesunięcia, skalowania albo obrotu, to, czy $M_1 \cdot M_2 = M_2 \cdot M_1$? To znaczy, czy M_1 i M_2 mogą być zamienione miejscami? Na ogół mnożenie macierzy nie jest przemienne. Łatwo jednak wykazać, że w następujących specjalnych przypadkach przemienność obowiązuje:

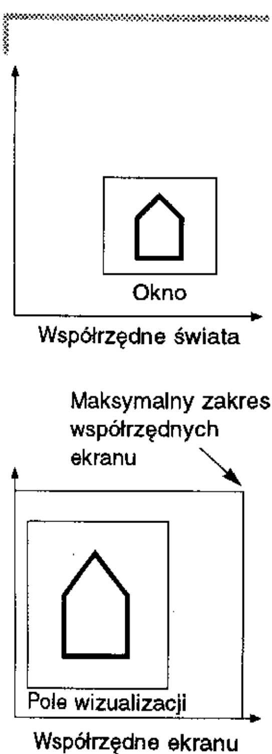
M_1	M_2
Przesunięcie	Przesunięcie
Skalowanie	Skalowanie
Obrót	Obrót
Skalowanie (z $s_x = s_y$)	Obrót

W tych przypadkach nie musimy dbać o kolejność składania macierzy.

5.5. Przekształcenie okna w pole wizualizacji

Niektóre pakiety graficzne umożliwiają programiście określenie współrzędnych wyjściowego prymitywu w systemie zmiennopozycyjnych *współrzędnych świata*, wykorzystujących jednostki takie, jakie są wygodne dla programu użytkowego: angstremy, mikrony, metry, mile, lata świetlne itd. Używany jest wyraz *świat* z tego względu, że program użytkowy reprezentuje świat, który jest interakcyjnie tworzony albo wyświetlany przez użytkownika.

Jeżeli prymitywy wyjściowe są określane we współrzędnych świata, to pakiet graficzny musi określić, jak odwzorować współrzędne świata na *współrzędne ekranu* (używamy określenia współrzędne ekranu, żeby naszą dyskusję powiązać z SRGP; równie dobrze moglibyśmy mówić o urządzeniu wyjściowym do tworzenia kopii trwałych i wtedy właściwe byłoby mówienie o *współrzędnych urządzenia*). Moglibyśmy wykonywać to odwzorowanie korzystając z macierzy przekształcenia umieszczonej przez programistę w programie użytkowym. Inny sposób polega na



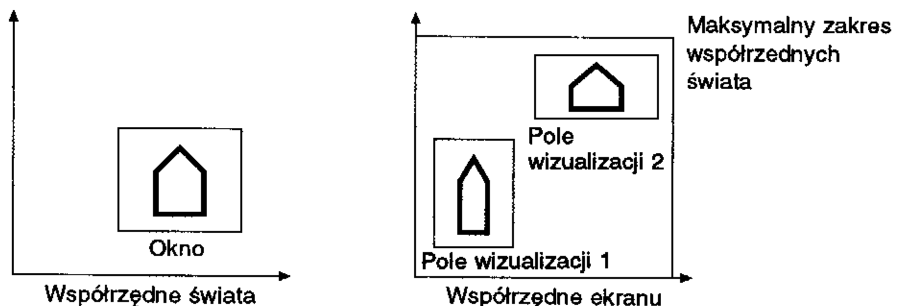
Rys. 5.12. Okno we współrzędnych świata i pole wizualizacji we współrzędnych ekranu określają odwzorowanie stosowane do wszystkich prymitywów wyjściowych we współrzędnych świata

tym, żeby program użytkowy określał prostokątny obszar we współrzędnych świata nazywany *oknem we współrzędnych świata* i odpowiedni obszar prostokątny we współrzędnych ekranu, określany jako *pole wizualizacji*, na który będzie odwzorowane okno we współrzędnych świata. Przekształcenie, które odwzorowuje okno na pole wizualizacji, jest stosowane do wszystkich prymitywów wyjściowych we współrzędnych świata, dzięki czemu zostają one odwzorowane na współrzędne ekranu. Koncepcję tę ilustruje rys. 5.12. Jak można zauważyć na rysunku, jeżeli okno i pole wizualizacji nie mają takich samych stosunków wysokości do szerokości, to ma miejsce niejednorodne skalowanie. Jeżeli program użytkowy zmieni okno albo pole wizualizacji, to ta zmiana będzie miała wpływ na nowe prymitywy wyjściowe rysowane na ekranie. Taka zmiana nie ma wpływu na istniejące prymitywy wyjściowe.

Mówiąc o oknie dodajemy określenie we „współrzędnych świata" po to, żeby podkreślić, że nie mówimy o oknie związanym z programem zarządzania oknami, które jest inną i nowszą koncepcją i które niestety ma tę samą nazwę. Tam, gdzie nie będzie wątpliwości, będziemy opuszczali to dodatkowe określenie.

Gdyby pakiet SRGP dostarczał prymitywy wyjściowe we współrzędnych świata, wówczas pole wizualizacji byłoby w bieżącej kanwie, przy czym domniemaną kanwą 0 jest ekran. Program użytkowy mógłby zmieniać okno albo pole wizualizacji w dowolnej chwili i wtedy prymitywy wyjściowe określone później podlegałyby nowym przekształceniom. Gdyby zmiana obejmowała inne pole wizualizacji, wówczas nowy prymityw wyjściowy byłby umieszczony w kanwie w innym miejscu niż poprzednio, tak jak to pokazano na rys. 5.13.

Program zarządzania oknami mógłby odwzorować kanwę 0 SRGP na okno mniejsze od pełnego ekranu i wtedy niekoniecznie cała kanwa, a nawet pole wizualizacji byłyby widoczne.



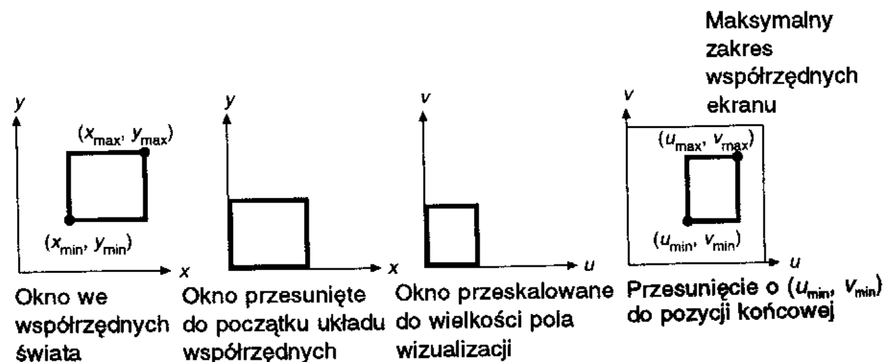
Rys. 5.13. Efekt rysowania prymitywów wyjściowych z dwoma polami wizualizacji. Prymitywy wyjściowe określające dom zostały najpierw narysowane w polu wizualizacji 1, potem pole wizualizacji zostało zmienione na pole wizualizacji 2, a następnie program użytkowy znowu wywołał pakiet graficzny, żeby narysować prymitywy wyjściowe

Jeżeli dane jest okno i pole wizualizacji, to jaka jest macierz przekształceń, która odwzorowuje okno ze współrzędnych świata na pole wizualizacji we współrzędnych ekranu? Taką macierz można otrzymać w wyniku złożenia trzech kolejnych przekształceń, tak jak to sugeruje rys. 5.14. Okno, określone przez narożniki dolny lewy i górny prawy, najpierw jest przesuwane do początku układu współrzędnych. Następnie okno jest skalowane tak, żeby dopasować się do wymiarów pola wizualizacji. Wreszcie po zastosowaniu przesunięcia pole wizualizacji zostaje umieszczone na właściwej pozycji. Cała macierz M_{WV} otrzymana po złożeniu dwóch macierzy przesuwania i macierzy skalowania ma postać

$$M_{WV} = T(u_{\min}, v_{\min}) \cdot S\left(\frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}}, \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}\right) \cdot T(-x_{\min}, -y_{\min}) =$$

$$= \begin{bmatrix} 1 & 0 & u_{\min} \\ 0 & 1 & v_{\min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & -x_{\min} \cdot \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & -y_{\min} \cdot \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\min} \\ 0 & 0 & 1 \end{bmatrix} \tag{5.33}$$

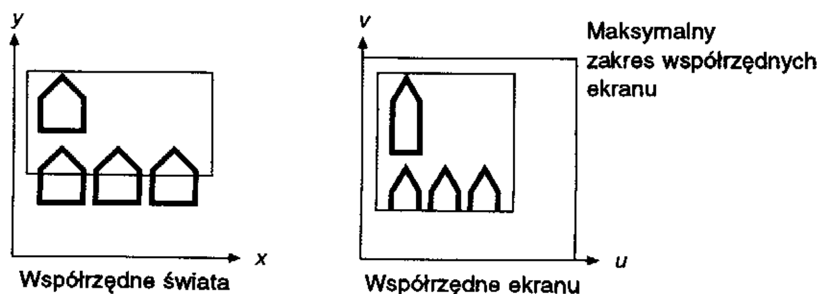


Rys. 5.14. Kroki procesu przekształcania okna we współrzędnych świata na pole wizualizacji

Końcowy wynik otrzymujemy po wykonaniu mnożenia $P = M_{WV}[x\ y\ 1]^T$:

$$P = \left[(x - x_{min}) \cdot \frac{u_{max} - u_{min}}{x_{max} - x_{min}} + u_{min} \quad (y - y_{min}) \cdot \frac{v_{max} - v_{min}}{y_{max} - y_{min}} + v_{min} \quad 1 \right] \tag{5.34}$$

W wielu pakietach graficznych przekształcenie okna w pole wizualizacji jest łączone z obcinaniem prymitywów wyjściowych przez okno. Koncepcję obcinania wprowadzono w rozdz. 3; na rysunku 5.15 pokazano obcinanie w kontekście okien i pól wizualizacji.



Rys. 5.15. Prymitywy wyjściowe we współrzędnych świata są obcinane przez okno. Te części, które pozostają, są wyświetlane w polu wizualizacji

5.6. Efektywność

Najogólniejsze złożenie operacji R, S i T daje macierz o postaci

$$M = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.35}$$

Górna podmacierz 2 × 2 jest złożoną macierzą obrotu i skalowania, a t_x i t_y są złożonymi przesunięciami. Obliczenie $M \cdot P$ jako wektora pomnożonego lewostronnie przez macierz 3 × 3 wymaga dziewięciu mnożeń i sześciu dodawań. Stała struktura ostatniego wiersza równania (5.35) umożliwia uproszczenie wykonywanych operacji do:

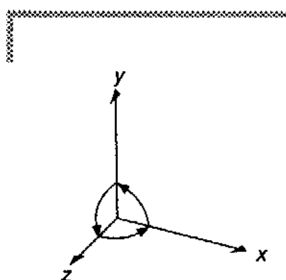
$$\left.\begin{aligned} x' &= x \cdot r_{11} + y \cdot r_{12} + t_x \\ y' &= x \cdot r_{21} + y \cdot r_{22} + t_y \end{aligned}\right\} \tag{5.36}$$

co redukuje proces do czterech mnożeń i czterech dodawań – jest to istotne przyspieszenie, zwłaszcza że operacja może być wykonywana w stosunku do setek, a nawet tysięcy punktów jednego obrazu. Dlatego, chociaż macierze 3×3 są wygodne i użyteczne do składania przekształceń 2D, możemy wykorzystać końcową macierz bardziej efektywnie w programie uwzględniając jej specyficzną strukturę. Niektóre sprzętowe rozwiązania układów mnożenia macierzy mają równoległe układy sumowania i mnożenia, co zmniejsza albo usuwa ten problem.

5.7. Reprezentacja macierzowa przekształceń 3D

Podobnie jak przekształcenia 2D mogą być reprezentowane za pomocą macierzy 3×3, przekształcenia 3D mogą być reprezentowane za pomocą macierzy 4×4, jeżeli wykorzystamy również współrzędne jednorodne do reprezentowania punktów z przestrzeni trójwymiarowej. Dlatego, zamiast reprezentować punkt jako (x, y, z), reprezentujmy go jako (x, y, z, W), przy czym dwie takie czwórki reprezentują ten sam punkt, jeżeli jeden z nich jest niezerową wielokrotnością drugiego; czwórka $(0, 0, 0, 0)$ jest zabroniona. Podobnie jak w przestrzeni 2D standardowa reprezentacja punktu (x, y, z, W) przy $W \neq 0$ jest dana jako $(x/W, y/W, z/W, 1)$. Jak poprzednio mówimy, że punkt jest we współrzędnych jednorodnych. Punkty, dla których współrzędna W jest równa 0, są określane jako punkty w nieskończoności. Jest również interpretacja geometryczna. Każdy punkt w przestrzeni trójwymiarowej jest reprezentowany przez odcinek przechodzący przez początek układu współrzędnych w przestrzeni czterowymiarowej, a reprezentacje tych punktów we współrzędnych jednorodnych tworzą w przestrzeni czterowymiarowej podprzestrzeń trójwymiarową, która jest określona przez jedno równanie $W = 1$.

W książce korzystamy z prawoskrętnego układu współrzędnych 3D, pokazanego na rys. 5.16. Zgodnie z konwencją za dodatni obrót w układzie prawoskrętnym uważa się taki, dla którego patrząc z dodatniego kierunku osi w kierunku początku układu współrzędnych obrót o 90° w kierunku przeciwnym względem ruchu wskazówek zegara przekształci jedną dodatnią oś w drugą. Z tej konwencji wynika następująca tablica

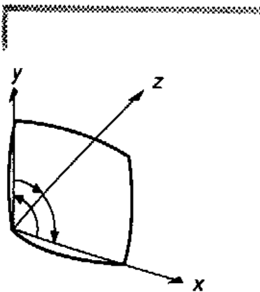


(widok w kierunku na zewnątrz strony)

Rys. 5.16. Prawoskrętny układ współrzędnych

Oś obrotu	Kierunek dodatniego obrotu
x	y na z
y	z na x
z	x na y

Te dodatnie kierunki są również zaznaczone na rys. 5.16. Pamiętajmy, że nie we wszystkich podręcznikach stosuje się tę konwencję.



Rys. 5.17. Lewostronny układ współrzędnych z nałożonym ekranem monitora

Korzystamy tutaj z układu prawoskrętnego, ponieważ jest to standardowa konwencja matematyczna, nawet jeżeli w grafice 3D jest wygodniej myśleć o układzie lewoskrętnym dla ekranu monitora (rys. 5.17), bowiem układ lewoskrętny daje bardziej naturalną interpretację, w której wartości z są większe dla punktów leżących dalej od obserwatora. Zauważmy, że w układzie lewoskrętnym dodatnie obroty są w kierunku zgodnym z ruchem wskazówek zegara, gdy patrzymy od strony dodatniej osi w kierunku początku układu współrzędnych. Ta definicja kierunku dodatniego obrotu umożliwia korzystanie z tej samej macierzy obrotu podanej w tym punkcie zarówno dla prawoskrętnego, jak i dla lewoskrętnego układu współrzędnych. Konwersję z jednej konwencji na drugą omówiono w p. 5.9.

Przesunięcie w 3D jest zwykłym rozszerzeniem przesunięcia w 2D:

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.37}$$

To znaczy

$$T(d_x, d_y, d_z) \cdot [x\ y\ z\ 1]^T = [x + d_x \quad y + d_y \quad z + d_z\ 1]^T$$

Podobnie rozszerza się operację skalowania

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.38}$$

Można zauważyć, że

$$S(s_x, s_y, s_z) \cdot [x\ y\ z\ 1]^T = [s_x \cdot x \quad s_y \cdot y \quad s_z \cdot z \quad 1]^T$$

Obrót 2D w równaniu (5.26) jest po prostu obrotem 3D wokół osi z

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.39}$$

Tę obserwację można łatwo zweryfikować: obrót o 90° jednostkowego wektora osi x $[1\ 0\ 0\ 1]^T$ powinien dać jednostkowy wektor $[0\ 1\ 0\ 1]^T$ osi y. Obliczając iloczyn

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{5.40}$$

otrzymamy przewidywany wynik $[0\ 1\ 0\ 1]^T$.

Macierz obrotu wokół osi x ma następującą postać:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.41}$$

Macierz obrotu wokół osi y ma następującą postać:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.42}$$

Kolumny (i wiersze) lewej górnej podmacierzy 3×3 macierzy $R_z(\theta)$, $R_x(\theta)$ i $R_y(\theta)$ są wzajemnie prostopadłymi wektorami jednostkowymi i podmacierz ma wyznacznik równy 1, co oznacza, że te trzy macierze są ortonormalne, tak jak to omawialiśmy w p. 5.3. Również lewa górna podmacierz 3×3 utworzona przez dowolną sekwencję obrotów jest ortonormalna. Przypomnijmy, że przekształcenia ortogonalne zachowują odległości i kąty.

Dla wszystkich tych macierzy przekształceń istnieją macierze odwrotne. Dla macierzy T macierz odwrotną otrzymujemy negując d_x, d_y i d_z; w przypadku macierzy S musimy zastąpić s_x, s_y i s_z przez ich odwrotności; dla macierzy obrotu musimy negować kąt obrotu.

Macierz odwrotna dla dowolnej macierzy ortogonalnej B jest to po prostu macierz transponowana: $B^{-1} = B^T$. W rzeczywistości wyznaczenie macierzy transponowanej nie wymaga nawet wymiany elementów tablicy pamiętającej macierz – wystarczy tylko zamienić indeksy wierszy i kolumn przy dostępie do tablicy. Zauważmy, że ta metoda znajdowania macierzy odwrotnej jest zgodna z wynikiem negowania θ przy znajdowaniu macierzy odwrotnej dla R_x, R_y, i R_z.

Można pomnożyć przez siebie dowolną liczbę macierzy obrotu, skalowania i przesuwania. Wynik zawsze ma postać

$$M = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \tag{5.43}$$

Tak jak w przypadku 2D, lewa górna podmacierz 3×3 macierzy R zawiera zagregowaną informację o obrocie i skalowaniu, natomiast T zawiera zagregowaną informację o przesunięciu. Zyskujemy trochę na efektywności obliczeń wykonując przekształcenia bezpośrednio jako

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + T \tag{5.44}$$

przy czym R i T są podmacierzami z równania (5.43).

Macierzom dla operacji pochylenia na płaszczyźnie z p. 5.2 odpowiadają trzy macierze dla operacji pochylenia w przestrzeni trójwymiarowej. Macierz dla operacji pochylenia (x, y) ma postać

$$SH_{xy}(sh_x, sh_y) = \begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.45}$$

Stosując operację SHx_y do punktu $[x \ \ y \ \ z \ \ 1]^T$, otrzymujemy $[x + sh_x \cdot z \quad y + sh_y \cdot z \ \ z \ \ 1]^T$. Macierze pochylenia wzdłuż osi x i y mają podobną postać.

Dotychczas zajmowaliśmy się przekształceniami w odniesieniu do poszczególnych punktów. Przekształcanie odcinków wykonujemy, dokonując przekształceń dla punktów końcowych. Płaszczyzny, jeżeli są zdefiniowane przez trzy punkty, można przekształcać w ten sam sposób; zazwyczaj jednak płaszczyzny są określone za pomocą równań płaszczyzn i współczynniki takich równań muszą być przekształcane w inny sposób. Może również zaistnieć potrzeba przekształcenia normalnej do płaszczyzny. Niech płaszczyzna będzie reprezentowana jako wektor kolumnowy współczynników $N = [A \ \ B \ \ C \ \ D]^T$. Wtedy płaszczyzna jest zdefiniowana jako zbiór punktów, dla których $N \cdot P = 0$, przy czym symbol „$\cdot$" reprezentuje iloczyn skalarny i $P = [x \ y \ z \ 1]^T$. Ten iloczyn

skalarny umożliwia przejście do znanego równania płaszczyzny $Ax + By + Cz + D = 0$, które również może być wyrażone jako iloczyn wektora wierszowego ze współczynnikami równania płaszczyzny przez wektor kolumnowy P: $N^T \cdot P = 0$. Załóżmy teraz, że przekształcamy wszystkie punkty P płaszczyzny za pomocą pewnej macierzy M. Żeby zapewnić spełnienie warunku $N^T \cdot P = 0$ dla punktów poddawanych przekształceniu, musimy przekształcić N za pomocą pewnej (do wyznaczenia) macierzy Q, co prowadziłoby do równania $(Q \cdot N)^T \cdot M \cdot P = 0$. To wyrażenie może z kolei być zapisane w postaci $N^T \cdot Q^T \cdot M \cdot P = 0$; korzystamy przy tym z tożsamości $(Q \cdot N)^T = N^T \cdot Q^T$. Równanie będzie słuszne, jeżeli $Q^T \cdot M$ jest wielokrotnością macierzy jednostkowej. Jeżeli mnożnik jest 1, to ta sytuacja prowadzi do $Q^T = M^{-1}$ albo $Q = (M^{-1})^T$. Dlatego wektor kolumnowy N' współczynników dla płaszczyzny przekształcanej za pomocą macierzy M jest dany jako

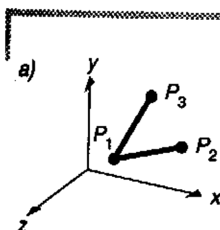
$$N' = (M^{-1})^T \cdot N \tag{5.46}$$

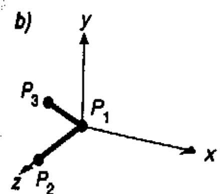
Macierz $(M^{-1})^T$ nie musi istnieć w ogólnym przypadku, ponieważ wyznacznik M może być równy 0. Taka sytuacja pojawiłaby się, gdyby macierz M obejmowała rzutowanie (moglibyśmy zbadać wpływ rzutowania perspektywicznego na płaszczyznę).

Jeżeli przekształceniu ma być poddana tylko normalna do płaszczyzny (na przykład w celu wykonania obliczeń związanych z wyznaczaniem cieniowania, omawianych w rozdz. 14) i jeżeli M składa się tylko ze złożenia macierzy przesuwania, obrotu i jednorodnego skalowania, to obliczenia są jeszcze prostsze. N' z równania (5.46) może być uproszczone do $[A' \; B' \; C' \; 0]^T$. (Przy zerowej składowej W punkt we współrzędnych jednorodnych reprezentuje punkt w nieskończoności, o którym można myśleć jak o kierunku).

5.8. Składanie przekształceń 3D

W tym punkcie omawiamy sposoby składania macierzy przekształceń 3D; wykorzystamy przy tym przykład, który będzie przydatny w p. 6.5. Celem jest dokonanie przekształcenia odcinków skierowanych P_1P_2 i P_1P_3 z początkowego położenia jak na rys. 5.18a do końcowego położenia jak na rys. 5.18b. Punkt P_1 ma być przesunięty do początku układu współrzędnych, P_1P_2 ma leżeć na dodatniej osi z, a P_1P_3 ma leżeć na półpłaszczyźnie (y, z), na której leży dodatnia oś y. Przekształcenie powinno nie zmienić długości odcinków.





Rys. 5.18. Przesunięcie P_1, P_2 i P_3 od pozycji początkowych (a) do pozycji końcowych (b)

Przedstawiono dwa sposoby otrzymania żądanego przekształcenia. Pierwsze podejście polega na złożeniu podstawowych przekształceń T, R_x, R_y i R_z. To podejście, chociaż nudne, można łatwo zilustrować i jego zrozumienie pomoże nam w przyswojeniu idei przekształceń. Drugie podejście, wykorzystujące właściwości specjalnych macierzy ortogonalnych opisanych w p. 5.7, jest wyjaśnione w sposób bardziej zwięzły, ale jest bardziej abstrakcyjne.

Aby skorzystać z podstawowych przekształceń, ponownie rozbijamy trudny problem na prostsze podproblemy. W tym przypadku potrzebne przekształcenia można wykonać w czterech krokach:

1. Przesunięcie P_1 do początku układu współrzędnych.
2. Taki obrót wokół osi y, żeby odcinek P_1P_2 znalazł się na płaszczyźnie (y, z).
3. Taki obrót wokół osi x, żeby odcinek P_1P_2 znalazł się na osi z.
4. Taki obrót wokół osi z, żeby odcinek P_1P_3 znalazł się na płaszczyźnie (y, z).

Krok 1. Przesunięcie P_1 do początku układu współrzędnych. Przesunięcie jest następujące:

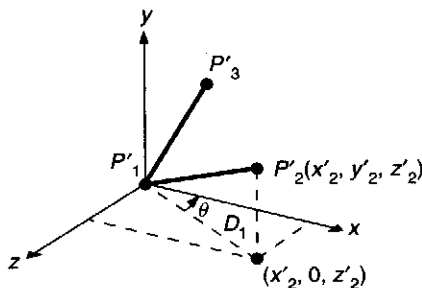
$$T(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.47}$$

Wykonując przesunięcie T w stosunku do P_1, P_2 i P_3 otrzymujemy:

$$P_1' = T(-x_1, -y_1, -z_1) \cdot P_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{5.48}$$

$$P_2' = T(-x_1, -y_1, -z_1) \cdot P_2 = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix} \tag{5.49}$$

$$P_3' = T(-x_1, -y_1, -z_1) \cdot P_3 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{bmatrix} \tag{5.50}$$



Rys. 5.19. Obrót wokół osi y: Rzut $P_1' P_2'$ o długości D_1 jest obracany na oś z. Kąt θ pokazuje dodatni kierunek obrotu wokół osi y; kąt faktycznie użyty wynosi $-(90-\theta)$

Krok 2. Obrót wokół osi y. Na rysunku 5.19 pokazano odcinek P_1P_2 po kroku 1 i po zrzutowaniu P_1P_2 na płaszczyznę (x, z). Kąt obrotu wynosi $-(90-\theta)=\theta-90$. Wtedy:

$$\cos(\theta-90)=\sin\theta=\frac{z_2'}{D_1}=\frac{z_2-z_1}{D_1}$$

$$\sin(\theta-90)=-\cos\theta=-\frac{x_2'}{D_1}=-\frac{x_2-x_1}{D_1} \tag{5.51}$$

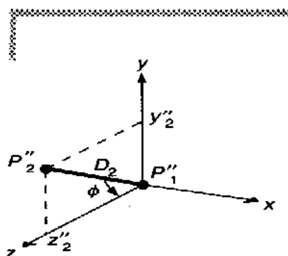
przy czym

$$D_1=\sqrt{(z_2')^2+(x_2')^2}=\sqrt{(z_2-z_1)^2+(x_2-x_1)^2} \tag{5.52}$$

Jeżeli te wartości podstawimy do równania (5.42), to otrzymamy

$$P_2''=R_y(\theta-90)\cdot P_2'=[0\ \ y_2-y_1\ \ D_1\ \ 1]^{\mathrm{T}} \tag{5.53}$$

Tak jak należało się spodziewać, współrzędna x punktu P_2'' jest równa zeru, a współrzędna z jest równa długości D_1.



Rys. 5.20. Obrót wokół osi x: $P_1''P_2''$ jest obracany na oś z o dodatni kąt Φ. D_2 jest długością odcinka. Odcinek $P_1''P_3''$ nie jest pokazany, ponieważ nie jest wykorzystany do określenia kątów obrotu. Oba odcinki są obrócone o $R_x(\Phi)$

Krok 3. Obrót wokół osi x. Na rysunku 5.20 pokazano odcinek P_1P_2 po kroku 2. Kąt obrotu jest równy Φ, przy czym:

$$\cos\Phi=\frac{z_2''}{D_2'},\quad \sin\Phi=\frac{y_2''}{D_2'} \tag{5.54}$$

gdzie $D_2=|P_1''P_2''|$ jest długością odcinka $P_1''P_2''$. Ponieważ przekształcenia obrotu i przesunięcia zachowują długość, zatem

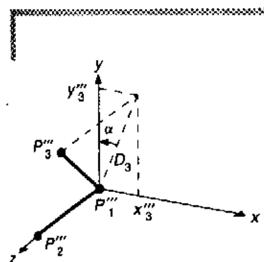
$$D_2+|P_1''P_2''|=|P_1P_2|=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2} \tag{5.55}$$

Wynik obrotu w kroku 3 jest następujący:

$$P_2''' = R_x(\Phi) \cdot P_2'' = R_x(\Phi) \cdot R_y(\theta - 90) \cdot P_2' =$$
$$= R_x(\Phi) \cdot R_y(\theta - 90) \cdot T \cdot P_2 = [0\ 0\ |P_1 P_2|\ 1]^T \qquad (5.56)$$

Oznacza to, że odcinek P_1P_2 leży teraz na osi dodatniej z.

Krok 4. Obrót wokół osi z. Na rysunku 5.21 pokazano odcinki P_1P_2 i P_1P_3 po kroku 3, przy czym P_2''' jest na osi z, a P_3''' w punkcie



Rys. 5.21. Obrót wokół osi z: rzut odcinka $P_1'P_3'$ o długości D_3 jest obrócony o dodatni kąt α na oś y; sam odcinek znajduje się na płaszczyźnie (y, z). D_3 jest długością rzutu

$$P_3''' = [x_3'''\ y_3'''\ z_3''']^T = R_x(\Phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) \cdot P_3 \qquad (5.57)$$

Obrót jest wykonywany o dodatni kąt α, przy czym:

$$\cos\alpha = \frac{y_3'''}{D_3}, \quad \sin\alpha = \frac{x_3'''}{D_3}, \quad D_3 = \sqrt{x_3'''^2 + y_3'''^2} \qquad (5.58)$$

W kroku 4 uzyskuje się rezultat pokazany na rys. 5.18b.

Złożona macierz o postaci

$$R_z(\alpha) \cdot R_x(\Phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) = R \cdot T \qquad (5.59)$$

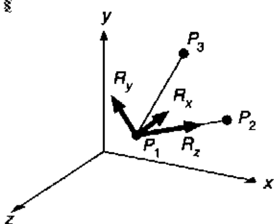
jest szukaną macierzą przekształcenia, przy czym $R = R_z(\alpha) \cdot R_x(\Phi) \cdot R_y(\theta - 90)$. Pozostawiamy czytelnikowi zastosowanie tego przekształcenia do P_1, P_2 i P_3 w celu sprawdzenia, że P_1 jest przesunięte do początku układu współrzędnych, P_2 jest przesunięte na dodatnią oś z, a P_3 jest na półpłaszczyźnie (y, z), na której leży dodatnia oś y.

Drugi sposób otrzymania macierzy R polega na użyciu właściwości macierzy ortogonalnych omówionej w p. 5.3. Przypomnijmy, że jednostkowe wektory wierszowe macierzy R obracają się na główne osie. Po zastąpieniu, ze względu na wygodę notacji, drugich indeksów w równaniu (5.43) przez x, y i z, otrzymujemy

$$R = \begin{bmatrix} r_{1_x} & r_{2_x} & r_{3_x} \\ r_{1_y} & r_{2_y} & r_{3_y} \\ r_{1_z} & r_{2_z} & r_{3_z} \end{bmatrix} \qquad (5.60)$$

Ponieważ R_z jest wektorem jednostkowym wzdłuż P_1P_2, który obróci się na dodatnią oś z, zatem

$$R_z = [r_{1_z}\ r_{2_z}\ r_{3_z}]^T = \frac{P_1 P_2}{|P_1 P_2|} \qquad (5.61)$$



Rys. 5.22. Wektory jednostkowe R_x, R_y i R_z, które są przekształcone na główne osie

Dodatkowo wektor jednostkowy R_x jest prostopadły do płaszczyzny wyznaczonej przez punkty P_1, P_2, P_3 i obróci się na dodatnią oś x; stąd R_x musi być znormalizowanym iloczynem wektorowym dwóch wektorów na płaszczyźnie

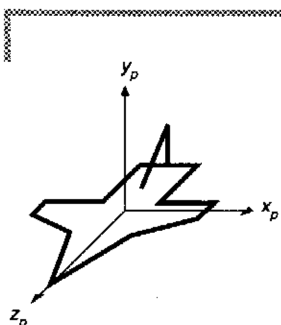
$$R_x = [r_{1x}\ r_{2x}\ r_{3x}]^{\mathrm{T}} = \frac{P_1P_3 \times P_1P_2}{|P_1P_3 \times P_1P_2|} \tag{5.62}$$

Wreszcie

$$R_y = [r_{1y}\ r_{2y}\ r_{3y}]^{\mathrm{T}} = R_z \times R_x \tag{5.63}$$

obróci się na dodatnią oś y. Złożona macierz ma postać

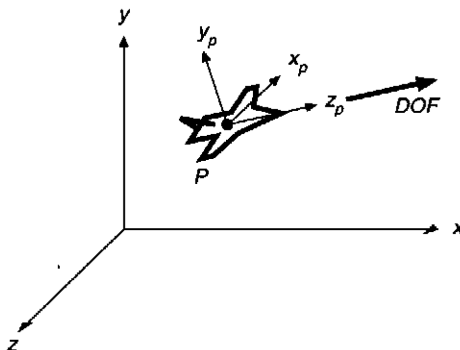
$$\begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & 0 \\ r_{1y} & r_{2y} & r_{3y} & 0 \\ r_{1z} & r_{2z} & r_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot T(-x_1,\ -y_1,\ -z_1) = R \cdot T \tag{5.64}$$



Rys. 5.23. Samolot w systemie współrzędnych (x_p, y_p, z_p)

przy czym R i T są takie jak w równaniu (5.59). Na rysunku 5.22 pokazano wektory R_x, R_y i R_z.

Rozważmy teraz inny przykład. Na rysunku 5.23 pokazano samolot zdefiniowany w układzie współrzędnych x_p, y_p, z_p i o środku w początku układu współrzędnych. Chcemy dokonać takiego przekształcenia samolotu, żeby jego dziób był skierowany w kierunku wektora *DOF* (kierunek lotu), a środek był w punkcie P i żeby nie był on pochylony, czyli żeby wyglądał tak jak na rys. 5.24. Przekształcenie do wykonania takiej reorientacji składa się z obrotu dziobu samolotu we właściwym kierunku i przesunięciu od początku układu współrzędnych do punktu P. Aby znaleźć macierz obrotu, określamy, w jakim kierunku jest



Rys. 5.24. Samolot z rys. 5.23 umieszczony w punkcie P lecący w kierunku *DOF*

skierowana na rys. 5.24 każda z osi x_p, y_p i z_p, sprawdzamy, czy kierunki są znormalizowane, i wykorzystujemy je jako wektory kolumnowe w macierzy obrotu.

Oś z_p musi być tak przekształcona, żeby pokryła się z kierunkiem lotu DOF, a oś x_p tak, żeby pokryła się z poziomym wektorem prostopadłym do DOF – to znaczy w kierunku $y \times DOF$ określonym przez iloczyn wektorowy y i DOF. Kierunek y_p jest określony przez $z_p \times x_p = DOF \times (y \times DOF)$, iloczyn wektorowy z_p i x_p; trzy kolumny macierzy obrotu są znormalizowanymi wektorami $|y \times DOF|$, $|DO \times (y \times DOF)|$ i $|DOF|$:

$$R = \begin{bmatrix} |y \times DOF| & |DOF \times (y \times DOF)| & |DOF| & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.65}$$

Sytuacja, kiedy DOF jest w kierunku osi y, jest sytuacją zdegenerowaną, ponieważ jest nieskończony zbiór możliwych wektorów dla wektora poziomego. To zdegenerowanie znajduje odbicie w algebrze, ponieważ iloczyny wektorowe $y \times DOF$ i $DOF \times (y \times DOF)$ są zerami. W tym specjalnym przypadku R nie jest macierzą obrotu.

5.9. Przekształcenia jako zmiana układu współrzędnych

Omawialiśmy przekształcenie zbioru punktów należących do obiektu w inny zbiór punktów, przy założeniu, że oba zbiory są w tym samym układzie współrzędnych. Przy takim podejściu układ współrzędnych pozostaje nie zmieniony i obiekty są przekształcane w odniesieniu do początku układu współrzędnych. Alternatywny, ale równoważny sposób myślenia o przekształcaniu polega na zmianie układu współrzędnych. Takie podejście jest przydatne wówczas, gdy łączy się wiele obiektów, z których każdy jest zdefiniowany w swoim własnym układzie współrzędnych i chcemy te układy współrzędnych obiektów wyrazić w jednym globalnym układzie współrzędnych. Taką sytuację opisano w rozdz. 7.

Zdefiniujmy $M_{i \leftarrow j}$ jako przekształcenie, które zamienia reprezentację punktu w układzie współrzędnych j na reprezentację w układzie współrzędnych i.

Oznaczmy przez $P^{(i)}$ reprezentację punktu w układzie współrzędnych i, przez $P^{(j)}$ reprezentację punktu w układzie współrzędnych j i przez $P^{(k)}$ reprezentację punktu w układzie współrzędnych k; wtedy:

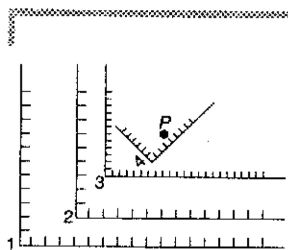
$$P^{(i)} = M_{i \leftarrow j} \cdot P^{(j)}, \quad P^{(j)} = M_{j \leftarrow k} \cdot P^{(k)} \tag{5.66}$$

Po podstawieniu otrzymujemy

$$P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)} = M_{i\leftarrow j} \cdot M_{j\leftarrow k} \cdot P^{(k)} = M_{i\leftarrow k} \cdot P^{(k)} \tag{5.67}$$

oraz

$$M_{i\leftarrow k} = M_{i\leftarrow j} \cdot M_{j\leftarrow k} \tag{5.68}$$



Rys. 5.25. Punkt P i układy współrzędnych 1, 2, 3 i 4

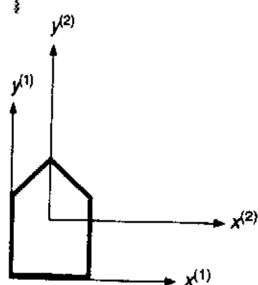
Na rysunku 5.25 pokazano cztery różne układy współrzędnych. Można zauważyć, że przekształcenie z układu współrzędnych 2 w układ 1 jest określone przez $M_{1\leftarrow 2} = T(4, 2)$. Podobnie, $M_{2\leftarrow 3} = T(2, 3) \cdot S(0{,}5, 0{,}5)$ i $M_{3\leftarrow 4} = T(6{,}7, 1{,}8) \cdot R(-45°)$. Wtedy $M_{1\leftarrow 3} = M_{1\leftarrow 2} \cdot M_{2\leftarrow 3} = T(4, 2) \cdot T(2, 3) \cdot S(0{,}5, 0{,}5)$. Na rysunku pokazano również punkt P, który w różnych układach współrzędnych jest określony jako $P^{(1)} = (10, 8)$, $P^{(2)} = (6, 6)$, $P^{(3)} = (8, 6)$ i $P^{(4)} = (4, 2)$. Można łatwo sprawdzić, że $P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)}$ dla $1 \leqslant i, j \leqslant 4$.

Zauważmy również, że $M_{i\leftarrow j} = M_{j\leftarrow i}^{-1}$, skąd $M_{2\leftarrow 1} = M_{1\leftarrow 2}^{-1} = T(-4, -2)$. Ponieważ $M_{1\leftarrow 3} = M_{1\leftarrow 2} \cdot M_{2\leftarrow 3}$, zatem $M_{1\leftarrow 3}^{-1} = M_{2\leftarrow 3}^{-1} \cdot M_{1\leftarrow 2}^{-1} = M_{3\leftarrow 2} \cdot M_{3\leftarrow 1}$.

W punkcie 5.7 omawialiśmy układy współrzędnych lewoskrętny i prawoskrętny. Macierz, która umożliwia dokonanie konwersji z jednego rodzaju układu współrzędnych na drugi, ma postać (jest to macierz odwrotna do siebie samej)

$$M_{R\leftarrow L} = M_{L\leftarrow R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.69}$$

Podejście użyte w poprzednim punkcie – określenie wszystkich obiektów w układzie współrzędnych świata i potem takie ich przekształcenie, żeby znalazły się we właściwym miejscu – zawiera dość nierealistyczne założenie, że wszystkie obiekty zostały pierwotnie określone w tym samym układzie współrzędnych świata. Bardziej naturalne jest myślenie o każdym obiekcie jak o obiekcie określonym w jego własnym układzie współrzędnych, a potem skalowanym, obracanym i przesuwanym na zasadzie przedefiniowania układu współrzędnych do nowego systemu współrzędnych świata. Przy takim podejściu w sposób naturalny myślimy o oddzielnych kartkach papieru z obiektem na każdej z nich, ściskanych lub rozciąganych, obracanych albo umieszczanych na płaszczyźnie współrzędnych świata. Możemy również wyobrazić sobie, że to płaszczyzna jest ścieśniana lub rozciągana, pochylana lub przesuwana względem każdej kartki papieru. Matematycznie wszystkie te podejścia są równoważne.



Rys. 5.26. Dom i dwa układy współrzędnych. Współrzędne punktów domu mogą być reprezentowane w każdym z układów współrzędnych

Rozważmy prosty przypadek przesunięcia zbioru punktów, które określają dom pokazany na rys. 5.10, do początku układu współrzędnych. Jest to przekształcenie $T(-x_1, -y_1)$. Oznaczając te dwa układy współrzędnych jak na rys. 5.26 możemy zauważyć, że przesunięcie, które odwzorowuje układ współrzędnych 1 na 2 – to jest $M_{2\leftarrow 1}$ – ma postać $T(x_1, y_1)$, co jest równe $T(-x_1, -y_1)^{-1}$. Ogólną regułą jest, że przekształcenie, które przekształca zbiór punktów w jednym układzie współrzędnych jest po prostu odwrotnością odpowiedniego przekształcenia zmieniającego układ współrzędnych, w którym punkt jest reprezentowany. Tę zależność można zauważyć na rys. 5.27, otrzymanym bezpośrednio z rys. 5.11. Przekształcenie dla punktów reprezentowanych w jednym układzie współrzędnych jest określane jako

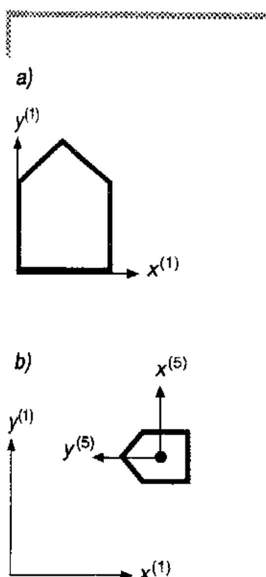
$$T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) \tag{5.32}$$

Na rysunku 5.27 przekształcenie układu współrzędnych jest określane jako

$$\begin{aligned} M_{5\leftarrow 1} &= M_{5\leftarrow 4} M_{4\leftarrow 3} M_{3\leftarrow 2} M_{2\leftarrow 1} = \\ &= (T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1))^{-1} = \\ &= T(x_1, y_1) \cdot S(s_x^{-1}, s_y^{-1}) \cdot R(-\theta) \cdot T(-x_2, -y_2) \end{aligned} \tag{5.70}$$

a więc

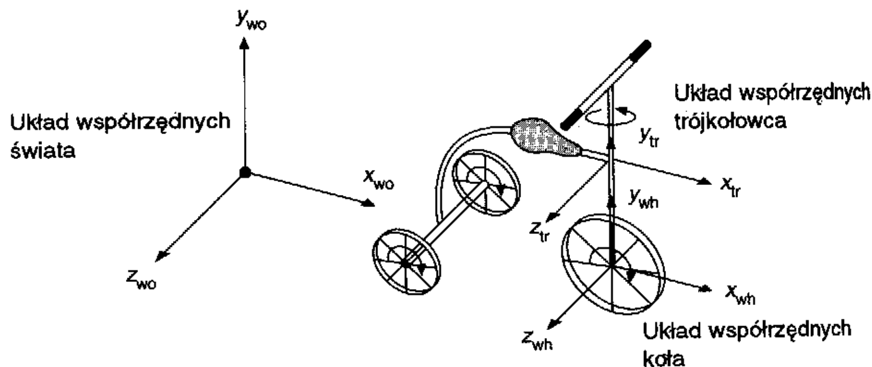
$$P^{(5)} = M_{5\leftarrow 1} P^{(1)} = T(x_1, y_1) \cdot S(s_x^{-1}, s_y^{-1}) \cdot R(-\theta) \cdot T(-x_2, -y_2) \cdot P^{(1)} \tag{5.71}$$



Rys. 5.27. Oryginalny dom w jego układzie współrzędnych (a); dom po przekształceniu w jego układzie współrzędnych w odniesieniu do oryginalnego układu współrzędnych (b)

Ważnym zagadnieniem związanym ze zmianą układu współrzędnych jest sposób zmiany przekształceń. Załóżmy, że $Q^{(j)}$ jest przekształceniem w układzie współrzędnych j. Mogłoby to być na przykład jedno ze złożonych przekształceń otrzymane w poprzednich punktach. Załóżmy, że chcielibyśmy znaleźć przekształcenie $Q^{(i)}$ w układzie współrzędnych i, które mogłoby być zastosowane do punktów $P^{(i)}$ w układzie i, i dawać dokładnie takie same wyniki, jak gdyby $Q^{(j)}$ było zastosowane do odpowiednich punktów $P^{(j)}$ w układzie j. Ta odpowiedniość jest reprezentowana przez $Q^{(i)} \cdot P^{(i)} = M_{i\leftarrow j} \cdot Q^{(j)} \cdot P^{(j)}$. Po podstawieniu $P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)}$ to wyrażenie przyjmuje postać $Q^{(i)} \cdot M_{i\leftarrow j} \cdot P^{(j)} = M_{i\leftarrow j} \cdot Q^{(j)} \cdot P^{(j)}$. Upraszczając otrzymujemy $Q^{(i)} = M_{i\leftarrow j} \cdot Q^{(j)} \cdot M_{i\leftarrow j}^{-1}$.

Rozumowanie w kategoriach zmiany układu współrzędnych jest przydatne wtedy, kiedy dodatkowa informacja dla podobiektu jest określona w lokalnym systemie współrzędnych podobiektu. Na przykład,



Rys. 5.28. Stylizowany trójkołowiec i trzy układy współrzędnych

jeżeli przednie koło trójkołowca z rys. 5.28 ma się obracać wokół jego osi z_{wh}, to wszystkie koła muszą się obracać w podobny sposób i musimy wiedzieć, jak trójkołowiec jako całość porusza się w układzie współrzędnych świata. Jest to złożony problem, ponieważ pojawiają się różne kolejne zmiany układów współrzędnych. Po pierwsze, układy współrzędnych trójkołowca i przedniego koła mają początkowe pozycje w układzie współrzędnych świata. Gdy rower porusza się do przodu, przednie koło obraca się wokół osi z układu współrzędnych koła i równocześnie układy współrzędnych koła i trójkołowca poruszają się względem układu współrzędnych świata. Układy współrzędnych koła i trójkołowca są powiązane z układem współrzędnych świata przez zmieniające się w czasie przesunięcia w kierunku x oraz z plus obrót wokół y. Układy współrzędnych trójkołowca i koła są związane ze sobą przez zmienny w czasie obrót wokół y, gdy kierownica zostaje skręcona. (Układ współrzędnych trójkołowca jest związany raczej z ramą niż z kierownicą.)

Żeby trochę ułatwić problem, zakładamy, że osie koła i trójkołowca są równoległe do osi układu współrzędnych i że koło porusza się po linii prostej równoległej do osi x układu współrzędnych świata. Gdy koło obraca się o kąt α, wówczas punkt P na kole przesuwa się o αr, przy czym r jest promieniem koła. Ponieważ koło jest na podłożu, trójkołowiec porusza się do przodu o αr jednostek. Punkt P na obwodzie koła porusza się i obraca w odniesieniu do początkowego układu współrzędnych koła i w efekcie przesuwa się o αr i obraca o kąt α. Jego nowe współrzędne P' w oryginalnym układzie współrzędnych koła są następujące:

$$P'^{(wh)} = T(\alpha r, 0, 0) \cdot R_z(\alpha) \cdot P^{(wh)} \tag{5.72}$$

i jego współrzędne w nowym (przesuniętym) układzie współrzędnych koła są wynikiem obrotu

$$P'^{(wh')} = R_z(\alpha) \cdot P^{(wh)} \tag{5.73}$$

W celu znalezienia punktów $P^{(wo)}$ i $P'^{(wo)}$ w układzie współrzędnych koła, przechodzimy do układu współrzędnych świata

$$P^{(wo)} = M_{wo \leftarrow wh} \cdot P^{(wh)} = M_{wo \leftarrow tr} \cdot M_{tr \leftarrow wh} \cdot P^{(wh)} \qquad (5.74)$$

przy czym $M_{wo \leftarrow tr}$ i $M_{tr \leftarrow wh}$ są to przesunięcia określone przez początkowe pozycje trójkołowca i koła.

$P'^{(wo)}$ obliczamy z równań (5.72) i (5.74)

$$P'^{(wo)} = M_{wo \leftarrow wh} \cdot P'^{(wh)} = M_{wo \leftarrow wh} \cdot T(\alpha r, 0, 0) \cdot R_z(\alpha) \cdot P^{(wh)} \qquad (5.75)$$

Możemy również zauważyć, że $M_{wo \leftarrow wh}$ zostało zamienione na $M_{wo \leftarrow wh'}$ w wyniku przesunięcia układu współrzędnych koła. Otrzymujemy ten sam wynik co w równaniu (5.75), ale w inny sposób

$$P'^{(wo)} = M_{wo \leftarrow wh'} \cdot P'^{(wh')} = (M_{wo \leftarrow wh} \cdot M_{wh \leftarrow wh'}) \cdot (R_z(\alpha) \cdot P^{(wh)}) \qquad (5.76)$$

Ogólnie otrzymujemy nowe $M_{wo \leftarrow wh'}$ i $M_{tr' \leftarrow wh'}$ na podstawie ich poprzednich wartości, stosując odpowiednie przekształcenia z równań ruchu części trójkołowca. Następnie te nowe przekształcenia stosujemy do uaktualnionych punktów w lokalnym układzie współrzędnych i otrzymujemy równoważne punkty w układach współrzędnych świata.

Zadania

5.1. Pokaż, że możemy przekształcić odcinek przekształcając jego punkty końcowe i potem konstruując nowy odcinek między przekształconymi końcami odcinka.

5.2. Pokaż, że dwa kolejne obroty 2D są addytywne: $R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$.

5.3. Pokaż, że obrót 2D i skalowanie są przemienne, jeżeli $s_x = s_y$ albo jeżeli $\theta = n\pi$ dla n całkowitego, i że w przeciwnych przypadkach ta właściwość nie jest prawdziwa.

5.4. Zastosuj przekształcenia wyprowadzone w p. 5.8 do punktów P_1, P_2 i P_3 w celu sprawdzenia, że te punkty są przekształcane zgodnie z zamierzeniem.

5.5. Przeanalizuj ponownie p. 5.8 zakładając, że $|P_1 P_2| = 1$, $|P_1 P_3| = 1$ i że dane są kosinusy kierunkowe odcinków P_1P_2 i P_1P_3 (*kosinusy kierunkowe* odcinka są to kosinusy kątów między odcinkiem a osiami x, y, z). Dla odcinka między początkiem układu współrzędnych a (x, y, z) kosinusy kierunkowe są równe $(x/d, y/d, z/d)$, przy czym d jest długością odcinka.

5.6. Pokaż, że równania (5.59) i (5.64) są równoważne.

5.7. Dla sześcianu jednostkowego o wierzchołku (0, 0, 0) i przeciwnym wierzchołku (1, 1, 1) wyprowadź przekształcenia potrzebne do obrócenia sześcianu o θ stopni wokół głównej przekątnej (od (0, 0, 0) do (1, 1, 1)) w kierunku przeciwnym względem ruchu wskazówek zegara, gdy patrzymy wzdłuż przekątnej w kierunku początku układu współrzędnych.

5.8. Załóż, że podstawa okna jest obracana o kąt θ od osi x, tak jak w systemie Core [GSPC79]. Jakie jest odwzorowanie okno-pole wizualizacji? Sprawdź odpowiedź stosując przekształcenie do każdego rogu okna i weryfikując, czy te rogi zostały przekształcone we właściwe rogi pola wizualizacji.

5.9. Weź odcinek między początkiem układu współrzędnych (w prawoskrętnym układzie współrzędnych) a punktem $P(x, y, z)$. Znajdź macierze przekształcenia potrzebne do obrócenia odcinka na dodatnią oś z dwoma różnymi sposobami i pokaż za pomocą przekształceń algebraicznych, że w każdym przypadku P znajdzie się na osi z. Dla każdej metody oblicz sinus i kosinus kąta obrotu.

a. Obróć wokół osi y na płaszczyznę (y, z), a potem obróć wokół osi x na oś z.

b. Obróć wokół osi z na płaszczyznę (x, z), a potem wokół osi y na oś z.

5.10. Obiekt ma być przeskalowany ze współczynnikiem S w kierunku, dla którego kosinusy kierunkowe są (α, β, γ). Wyprowadź macierz przekształcenia.

5.11. Znajdź macierz przekształcenia 4×4 dla obrotu o kąt wokół dowolnego kierunku określonego przez wektor kierunkowy $U = (u_x, u_y, u_z)$. Wykonaj to zadanie tworząc złożoną macierz dla przekształceń: obrót U na oś z (za pomocą macierzy M), obrót o $R_z(\theta)$, obrót za pomocą macierzy M^{-1}. Wynik powinien być następujący:

$$\begin{bmatrix} u_x^2 + \cos\theta(1 - u_x^2) & u_x u_y(1 - \cos\theta) - u_z \sin\theta & u_z u_x(1 - \cos\theta) + u_y \sin\theta & 0 \\ u_x u_y(1 - \cos\theta) + u_z \sin\theta & u_y^2 + \cos\theta(1 - u_y^2) & u_y u_z(1 - \cos\theta) - u_x \sin\theta & 0 \\ u_z u_x(1 - \cos\theta) - u_y \sin\theta & u_y u_z(1 - \cos\theta) + u_x \sin\theta & u_z^2 + \cos\theta(1 - u_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.77}$$

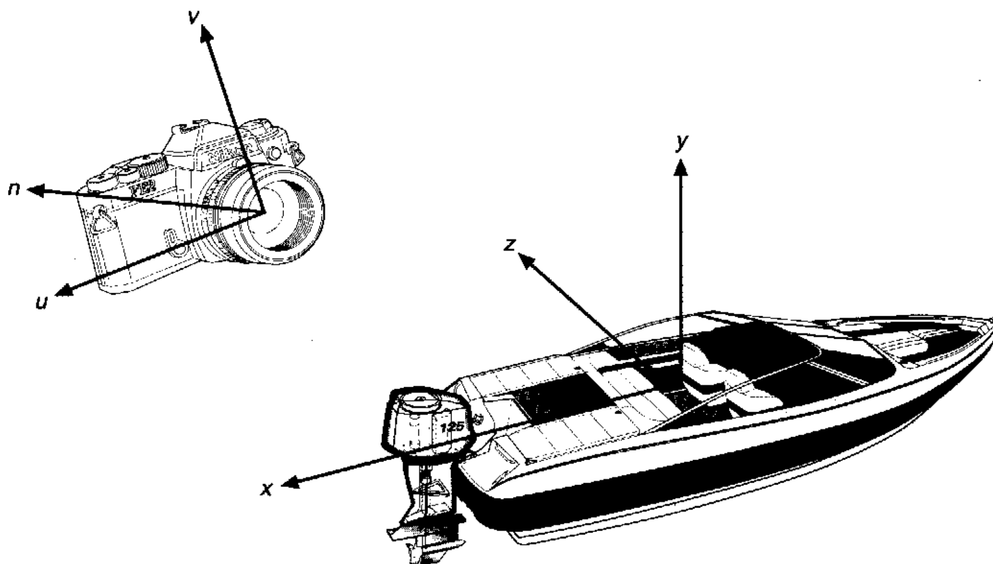
Sprawdź, że jeżeli U jest główną osią, to macierz redukuje się do R_x, R_y albo R_z. W pracy [FAUX79] jest wyprowadzenie oparte na operacjach wektorowych. Zauważmy, że zanegowanie U i θ nie zmienia wyniku. Wyjaśnij, dlaczego tak jest?

6. Rzutowanie w przestrzeni 3D

Proces rzutowania w 3D jest z natury bardziej złożony niż proces rzutowania w 2D. W 2D po prostu określamy okno na świat 2D i pole wizualizacji na rzutni 2D. Koncepcyjnie, obiekty świata są obcinane przez okno i przekształcane w pole wizualizacji przed wyświetleniem. Dodatkowa złożoność rzutowania 3D częściowo wynika z dodatkowego wymiaru, a częściowo z faktu, że urządzenia wyświetlające są tylko urządzeniami 2D. Chociaż rzutowanie w 3D może początkowo wydawać się kłopotliwe, problem staje się bardziej zrozumiały, jeżeli rozpatrzy się go jako ciąg kroków, z których wiele omówiliśmy w poprzednich rozdziałach. Zaczynamy od sprecyzowania, co rozumiemy przez rzutowanie w 3D; ułatwi to nam zrozumienie następnych punktów tego rozdziału.

6.1. Syntetyczna kamera i kroki rzutowania 3D

Użyteczną metaforą do tworzenia scen 3D jest pojęcie *syntetycznej kamery*, której koncepcję wyjaśniono na rys. 6.1. Wyobraźmy sobie, że możemy przesunąć kamerę do dowolnego miejsca, zorientować ją w dowolny sposób i otwierając migawkę utworzyć płaski obraz obiektu 3D – w tym przypadku obraz motorówki. Kamerze można nadać ruch – umożliwi to nam tworzenie animowanych sekwencji, które pokazują obiekty z różnych punktów obserwacji i przy różnych powiększeniach. Kamera jest to po prostu program komputerowy, który tworzy obrazy na ekranie monitora, a obiekty 3D są zbiorami danych zawierającymi



Rys. 6.1. Syntetyczna kamera fotografująca obiekt 3D

zbiory punktów, odcinków i powierzchni. Na rysunku 6.1 widać również, że kamera i obiekt 3D mają swoje własne układy współrzędnych: u, v, n dla kamery i x, y, z dla obiektu. Znaczenie tych układów współrzędnych omówimy w dalszej części rozdziału. Teraz zauważmy tylko, że dają one ważną niezależność reprezentacji.

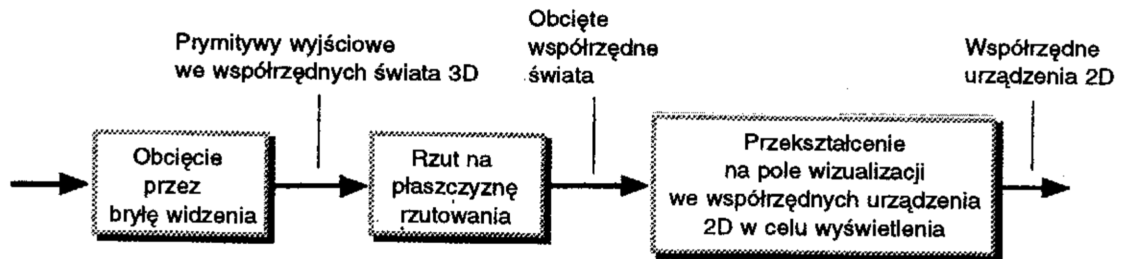
O ile kamera syntetyczna jest użyteczną koncepcją, o tyle do utworzenia obrazu potrzeba coś więcej niż tylko nacisnąć przycisk. Tworzenie „fotografii" faktycznie składa się z kilku kroków, które teraz opiszemy.

- *Określenie rodzaju rzutu.* Niezgodność między obiektami 3D i ich obrazami 2D eliminujemy wprowadzając rzutowanie obiektów 3D na płaszczyznę rzutowania 2D. Większość tego rozdziału jest poświęcona rzutom: rodzajom rzutów, matematyce związanej z rzutowaniem i wykorzystaniu rzutów w pakiecie graficznym PHIGS [ANSI88]. Koncentrujemy się na dwóch najważniejszych rzutach: *perspektywicznym* i *równoległym prostokątnym.* Rzutowanie jest omawiane również w rozdz. 7.
- *Określanie parametrów rzutowania.* Po wybraniu rodzaju rzutowania musimy określić warunki, w jakich chcemy oglądać zbiór danych rzeczywistego świata 3D, albo scenę, która ma być odtworzona. Jeżeli dane są współrzędne świata dla zbioru danych, to określają one położenie oka obserwatora i położenie płaszczyzny rzutowania

– powierzchni, na której ostatecznie rzut jest wyświetlany. Będziemy korzystali z dwóch układów współrzędnych – jednego dla sceny i drugiego, który określamy jako *układ współrzędnych oka* albo *rzutowania*. Zmieniając jeden albo wszystkie z tych parametrów możemy otrzymać dowolną potrzebną nam reprezentację sceny, włączając w to rzutowanie jej wnętrza, jeżeli będzie to miało sens.

▻ *Obcinanie w trzech wymiarach*. Tak jak musimy ograniczyć wyświetlaną scenę do granic określonego przez nas okna, tak musimy odrzucić te części sceny 3D, które nie są przeznaczone do wyświetlania. W rzeczywistości możemy chcieć pominąć te części sceny, które są poza nami albo są zbyt daleko na to, żeby mogły być dobrze widoczne. Wymaga to obcinania przez bryłę widzenia – bardziej złożony proces niż w dotychczas omawianych algorytmach. Ze względu na dużą różnorodność brył widzenia, włożymy trochę wysiłku w zdefiniowanie kanonicznej bryły, względem której możemy efektywnie zastosować standardowy algorytm obcinania.

▻ *Rzutowanie i wyświetlanie*. Zawartość bryły widzenia, która ma być rzutowana na płaszczyznę rzutowania określaną jako *okno*, jest przekształcana (odwzorowywana) w pole wizualizacji.



Rys. 6.2. Koncepcyjny model procesu rzutowania 3D

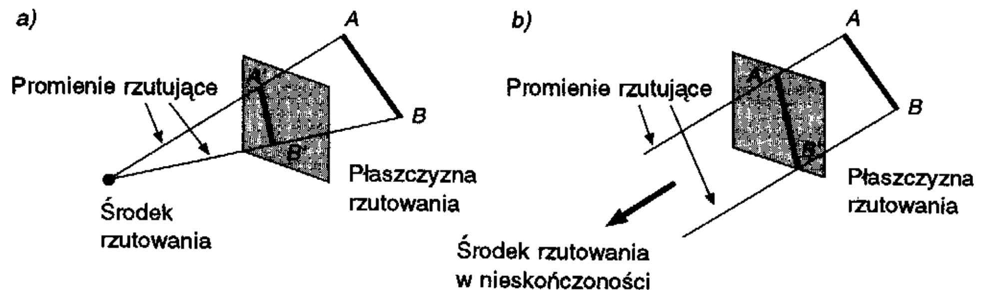
Na rysunku 6.2 pokazano główne kroki tego koncepcyjnego modelu procesu rzutowania 3D; jest to model prezentowany użytkownikom wielu graficznych programów 3D. Podobnie jak w przypadku 2D, można wykorzystywać różne strategie przy realizacji procesu rzutowania. Te strategie nie muszą być identyczne jak w modelu koncepcyjnym, jeżeli tylko wyniki są takie same jak dla tego modelu. Typową strategię implementacji dla rysunków szkieletowych opisano w p. 6.6. W systemach graficznych, w których są wykonywane algorytmy wykrywania powierzchni widocznych i cieniowanie, wykorzystuje się nieco inną sekwencję operacji; omawiamy to w rozdz. 14.

6.2. Rzuty

Ogólnie, rzuty przekształcają punkty w n-wymiarowym układzie współrzędnych w punkty w układzie współrzędnych o wymiarze mniejszym niż n. Grafika komputerowa od dawna była stosowana do badania obiektów n-wymiarowych przez rzutowanie ich na płaszczyznę 2D [NOLL67]. Tu ograniczymy się do rzutowania z przestrzeni 3D na płaszczyznę 2D. Rzut obiektu 3D jest określony przez promienie rzutujące, wychodzące ze środka rzutowania, przechodzące przez każdy punkt obiektu i przecinające płaszczyznę rzutowania. Ogólnie, środek rzutowania jest w skończonej odległości od rzutni. Dla niektórych rodzajów rzutowania wygodnie jest myśleć w kategoriach środka rzutowania, który ma tendencję do znajdowania się nieskończenie daleko; rozwiniemy tę koncepcję w p. 6.2.1. Na rysunku 6.3 pokazano dwa różne rzuty tego samego odcinka. Rzutem odcinka jest odcinek i wystarczy rzutować tylko końce odcinków.

Klasa rzutów, którymi będziemy się tutaj zajmowali, jest znana jako *planarne rzuty geometryczne*; wynika to stąd, że rzutowanie odbywa się na płaszczyznę, a nie na powierzchnie krzywoliniowe i korzysta się z promieni rzutujących prostoliniowych, a nie krzywoliniowych. Wiele rzutów kartograficznych jest albo nieplanarnych, albo niegeometrycznych.

Planarne rzuty geometryczne, nazywane dalej po prostu *rzutami*, można podzielić na dwie podstawowe klasy: *perspektywiczne* i *równoległe*. Różnica tkwi w relacji między środkiem rzutowania a płaszczyzną rzutowania. Jeżeli odległość między nimi jest skończona, to rzut jest perspektywiczny; jeżeli środek rzutowania coraz bardziej oddala się, to promienie rzutujące przechodzące przez dowolny obiekt są coraz bardziej zbliżone do promieni równoległych. Na rysunku 6.3 pokazano oba te przypadki. Nazwa rzutowanie równoległe pochodzi stąd, że jeżeli środek rzutowania jest nieskończenie daleko, to promienie rzutujące są równoległe. Gdy definiujemy rzut perspektywiczny, wówczas bezpośrednio określamy jego



Rys. 6.3. Dwa różne rzuty tego samego odcinka: a) odcinek AB i jego rzut perspektywiczny $A'B'$; b) odcinek AB i jego rzut równoległy $A'B'$. Promienie rzutujące AA' i BB' są równoległe

środek rzutowania; dla rzutu równoległego określamy kierunek rzutowania. Środek rzutowania jest punktem i jego współrzędne jednorodne mają postać $(x, y, z, 1)$. Ponieważ kierunek rzutowania jest wektorem (to jest różnicą między punktami), możemy go obliczyć odejmując od siebie dwa punkty $d = (x, y, z, 1) - (x', y', z', 1) = (a, b, c, 0)$. Stąd kierunki i punkty w nieskończoności odpowiadają sobie w naturalny sposób. W granicy, rzut perspektywiczny, którego środek rzutowania oddala się do nieskończoności, staje się rzutem równoległym.

Efekt wizualny rzutu perspektywicznego jest podobny do efektów w systemach fotograficznych i w systemie wzrokowym człowieka i jest znany jako *skrót perspektywiczny*: wielkość rzutu perspektywicznego obiektu zmienia się odwrotnie proporcjonalnie do odległości obiektu od środka rzutowania. Dlatego, chociaż rzut perspektywiczny obiektów sprawia wrażenie realistycznego, nie jest on szczególnie użyteczny do rejestrowania dokładnego kształtu i pomiarów obiektów; na podstawie rzutu nie można oceniać odległości, kąty są zachowane tylko dla tych płaszczyzn, które są równoległe do rzutni, i linie równoległe w ogólnym przypadku nie rzutują się na linie równoległe.

Rzut równoległy daje obraz mniej realistyczny ponieważ brak jest skrótu perspektywicznego, chociaż mogą być stałe skróty wzdłuż każdej z osi. Rzut może być używany do wykonywania dokładnych pomiarów i linie równoległe pozostają równoległe. Podobnie jak w rzucie perspektywicznym, kąty są zachowane tylko w płaszczyznach równoległych do rzutni.

Różne rzuty perspektywiczne i równoległe są wyczerpująco omówione i zilustrowane w pracy Carlboma i Paciorka [CARL78]. W punktach 6.2.1 i 6.2.2 podajemy podstawowe definicje i cechy najczęściej używanych rzutów; w punkcie 6.3 przechodzimy do przedstawienia sposobu określania rzutów w standardzie PHIGS.

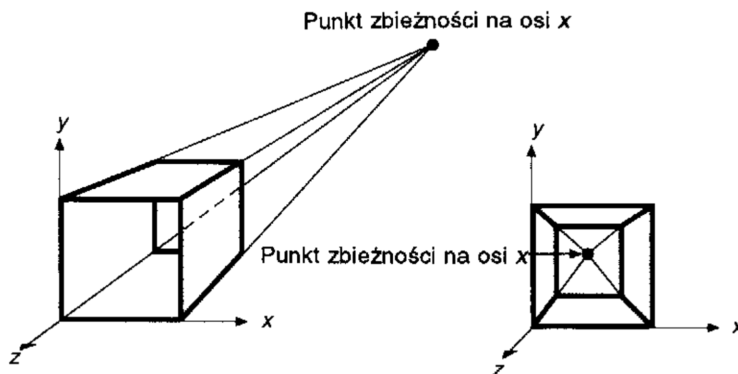
6.2.1. Rzuty perspektywiczne

Rzuty perspektywiczne dowolnego zbioru linii równoległych, które nie są równoległe do rzutni, zbiegają się w *punkcie zbieżności*. W 3D linie równoległe spotykają się tylko w nieskończoności i o punkcie zbieżności można myśleć jak o rzucie punktu w nieskończoności. Oczywiście jest nieskończenie dużo punktów zbieżności, po jednym dla każdego z nieskończenie wielu kierunków, w jakich może być zorientowany odcinek.

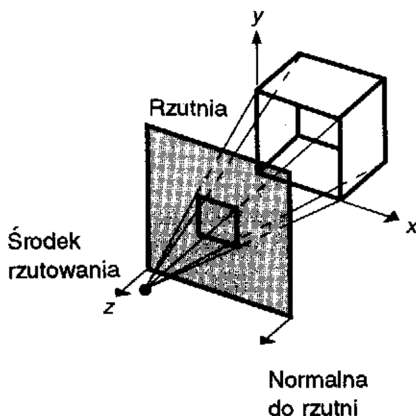
Jeżeli zbiór odcinków jest równoległy do jednej z trzech osi, to punkt zbieżności jest określany jako *osiowy punkt zbieżności*. Są najwyżej trzy takie punkty, odpowiadające liczbie podstawowych osi przecinanych przez rzutnię. Na przykład, jeżeli rzutnia przecina tylko oś z

(i wobec tego jest prostopadła do niej), tylko oś z ma podstawowy punkt zbieżności, ponieważ linie równoległe do osi y albo do osi z są również równoległe do rzutni i nie mają punktu zbieżności.

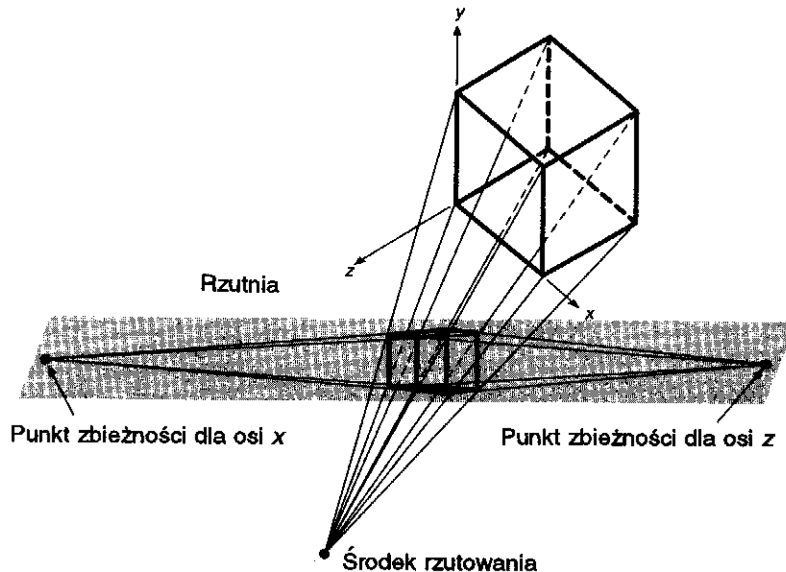
Rzuty perspektywiczne są dzielone ze względu na liczbę podstawowych punktów zbieżności, a tym samym ze względu na liczbę osi przecinanych przez rzutnię. Na rysunku 6.4 pokazano dwa różne jednopunktowe rzuty perspektywiczne sześcianu. Jest oczywiste, że są to rzuty jednopunktowe, ponieważ linie równoległe do osi x i y nie zbiegają się; dzieje się tak tylko dla linii równoległych do osi z. Na rysunku 6.5 pokazano konstrukcję jednopunktowego rzutu perspektywicznego, kilka promieni rzutujących i płaszczyznę rzutowania przecinającą tylko oś z.



Rys. 6.4. Jednopunktowy rzut perspektywiczny sześcianu na płaszczyznę przecinającą oś z; pokazano punkt zbieżności linii prostopadłych do płaszczyzny rzutowania



Rys. 6.5. Konstrukcja jednopunktowego rzutu perspektywicznego sześcianu na rzutnię przecinającą oś z. Normalna do rzutni jest równoległa do osi z. (Adaptacja z pracy [CARL78] za zgodą Association for Computing Machinery, Inc.)



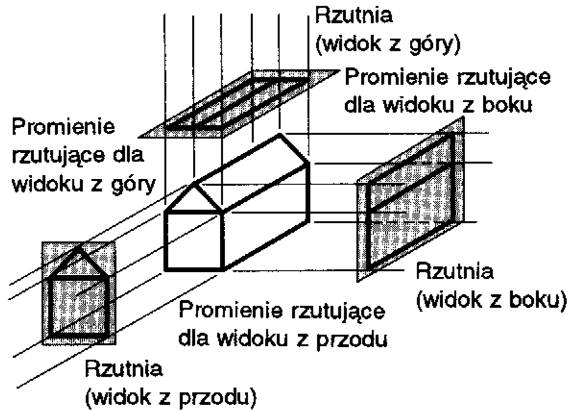
Rys. 6.6. Rzut sześcianu w perspektywie dwupunktowej. Rzutnia przecina osie x i z

Na rysunku 6.6 pokazano konstrukcję dwupunktowej perspektywy. Perspektywa dwupunktowa jest powszechnie używana w rysunkach architektonicznych, inżynierskich, w projektowaniu przemysłowym i w reklamach. Perspektywy trzypunktowe są używane rzadziej, ponieważ wnoszą one niewiele realizmu w stosunku do perspektywy dwupunktowej.

6.2.2. Rzuty równoległe

Wśród rzutów równoległych wyróżnia się dwa rodzaje, zależnie od relacji między kierunkiem rzutowania a normalną do rzutni. W równoległym rzucie prostokątnym te kierunki są takie same (albo są odwrotne względem siebie) i kierunek rzutowania jest prostopadły do rzutni. W równoległym rzucie ukośnym tak nie jest.

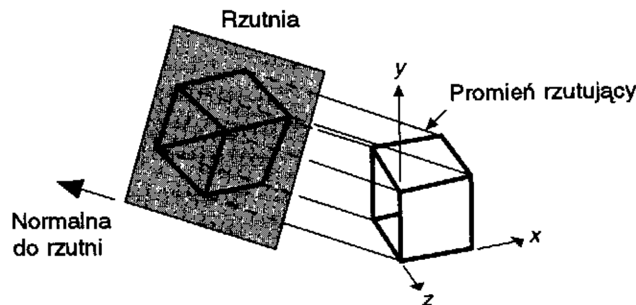
Najbardziej typowymi rzutami ortogonalnymi są rzuty *przedni*, *górny* i *boczny*. W tych wszystkich przypadkach rzutnia jest prostopadła do głównej osi, która w związku z tym określa kierunek rzutowania. Na rysunku 6.7 pokazano sposób tworzenia tych trzech rzutów; są one często wykorzystywane w rysunkach inżynierskich do obrazowania części maszyn, zespołów i budynków, ponieważ umożliwiają mierzenie odległości i kątów. Ponieważ jednak każdy rzut przedstawia tylko jeden widok obiektu, czasami trudno jest wyobrazić sobie strukturę przestrzenną rzutowanego obiektu, nawet jeżeli równocześnie ogląda się kilka rzutów tego samego obiektu.



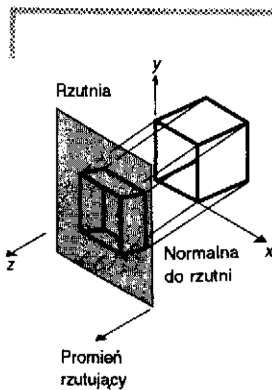
Rys. 6.7. Konstrukcja trzech rzutów prostokątnych

W *aksonometrycznych rzutach prostokątnych* rzutnia nie jest prostopadła do głównej osi i dlatego równocześnie widać kilka stron obiektu. W tym sensie przypominają one rzut perspektywiczny, różnica polega natomiast na tym, że skrót jest równomierny, a nie jest związany z odległością od środka rzutu. Równoległość linii jest zachowana, równość kątów natomiast nie, a odległości mogą być mierzone wzdłuż każdej głównej osi (w ogólnym przypadku z różnymi współczynnikami skali).

Często stosowanym rzutem aksonometrycznym jest *rzut izometryczny*. Normalna do rzutni (i stąd kierunek rzutowania) tworzy równe kąty z głównymi osiami. Jeżeli normalna do rzutni ma współrzędne (d_x, d_y, d_z), to chcemy, żeby $|d_x| = |d_y| = |d_z|$ albo $\pm d_x = \pm d_y = \pm d_z$. Jest dokładnie osiem kierunków (po jednym w każdym oktancie), które spełniają ten warunek. Na rysunku 6.8 pokazano konstrukcję rzutu izometrycznego wzdłuż jednego kierunku $[1, -1, -1]$.



Rys. 6.8. Konstrukcja rzutu izometrycznego dla sześcianu jednostkowego. (Adaptacja z pracy [CARL78] za zgodą Association for Computing Machinery, Inc.)

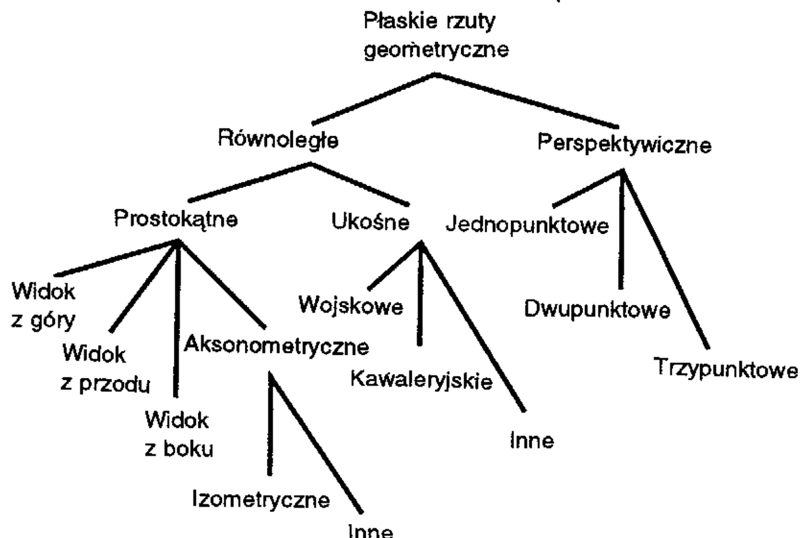


Rys. 6.9. Konstrukcja rzutu ukośnego (Adaptacja z pracy [CARL78] za zgodą Association for Computing Machinery, Inc.)

Rzut izometryczny ma taką użyteczną właściwość, że wszystkie trzy podstawowe osie są jednakowo skrócone; dzięki temu można wykonywać pomiary wzdłuż wszystkich osi w tej samej skali (stąd nazwa rzutu: *iso* od równy i *metric* od mierzyć). Ponadto kąty między rzutami głównych osi są takie same i wynoszą 120°.

Rzuty ukośne – druga klasa rzutów równoległych – różnią się od rzutów prostokątnych tym, że normalna do rzutni i kierunek rzutowania różnią się. Rzuty ukośne łączą właściwości rzutów prostokątnych czołowego, górnego i bocznego z właściwościami rzutu aksonometrycznego: rzutnia jest prostopadła do głównej osi i na rzucie płaszczyzny obiektu równoległej do rzutni można wykonywać pomiary kątów i odległości. Na rzutach innych płaszczyzn obiektu można dokonywać pomiarów odległości wzdłuż głównych osi, nie dotyczy to natomiast kątów. Ze względu na te właściwości oraz łatwość rysowania w książce używamy głównie rzutów ukośnych. Na rysunku 6.9 pokazano konstrukcję rzutu ukośnego. Zauważmy, że normalna do rzutni i kierunek rzutowania to nie to samo. Kilka rodzajów rzutów ukośnych jest opisanych w pracy [FOLE90].

Na rysunku 6.10 pokazano logiczne zależności między różnymi rzutami. Wspólną cechą tych wszystkich rzutów jest to, że występuje w nich rzutnia i albo środek rzutowania w rzucie perspektywicznym, albo kierunek rzutowania dla rzutu równoległego. Możemy dalej ujednolicić przypadki rzutów równoległych i perspektywicznych przyjmując, że środek rzutowania jest wyznaczony przez kierunek z jakiegoś punktu odniesienia do środka rzutowania i odległość do punktu odniesienia. Gdy



Rys. 6.10. Klasyfikacja płaskich rzutów geometrycznych

ta odległość rośnie do nieskończoności, rzut staje się rzutem równoległym. Dlatego możemy również powiedzieć, że wspólnym elementem łączącym te rzuty jest to, że wykorzystują płaszczyznę rzutów, kierunek do środka rzutowania i odległość do środka rzutowania. W punkcie 6.3 pokażemy, jak połączyć niektóre rodzaje rzutów w jeden proces rzutowania 3D.

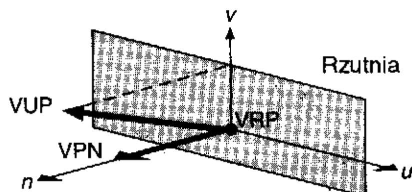
6.3. Specyfikowanie dowolnego rzutu 3D

Tak jak to sugeruje rys. 6.2, proces tworzenia obrazu 3D to nie tylko rzutowanie, ale również obcinanie sceny 3D przez bryłę widzenia. Rodzaj rzutu i bryła widzenia to pełna informacja potrzebna nam do obcięcia i wykonania rzutu na płaszczyznę 2D. Potem przekształcenie 2D we współrzędne fizycznego urządzenia jest już bezpośrednie. Korzystając z koncepcji geometrycznego rzutowania planarnego wprowadzonych w p. 6.2 pokażemy teraz, jak określić bryłę widzenia. Proponowane podejście i użyta terminologia są takie same jak w standardzie PHIGS.

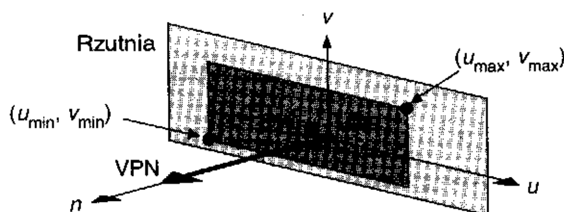
Rzutnia, określana również w literaturze związanej z grafiką jako *płaszczyzna rzutowania*, jest określona przez punkt na płaszczyźnie określany jako *punkt odniesienia rzutni* (VRP) i normalną do płaszczyzny określaną jako *normalna do rzutni* (VPN). Rzutnia może być gdziekolwiek w stosunku do obiektów świata, które mają być rzutowane: może być przed obiektami, może je przecinać i może być za nimi.

Dla danej rzutni trzeba określić okno. Rola okna jest podobna do roli okna w 2D: jego zawartość jest odwzorowywana na pole wizualizacji i dowolna część sceny 3D, której rzut wypada poza okno, nie jest wyświetlana. Zobaczymy, że okno odgrywa również ważną rolę przy definiowaniu bryły widzenia.

W celu określenia okna na rzutni musimy określić minimalne i maksymalne współrzędne okna i dwie prostopadłe osie na rzutni, wzdłuż których należy mierzyć te współrzędne. Te osie są częścią układu współrzędnych rzutowania (VRC). Początek układu VRC jest w punkcie VRP. Jedną z osi układu VRC jest oś VPN; ta oś jest określana jako oś n. Drugą oś układu VRC znajdziemy znając wektor skierowany ku górze (VUP); określa on kierunek osi v na rzutni. Oś v jest tak określona, że rzut VUP na rzutnię w kierunku równoległym do VPN jest koincydentny z osią v. Kierunek osi u jest tak określony, żeby u, v i n tworzyły prawoskrętny układ współrzędnych (rys. 6.11). VRP i dwa wektory kierunkowe VPN i VUP są określone w prawoskrętnym układzie współrzędnych świata. (W niektórych pakietach graficznych oś y jest używana jako VUP, ale ta konwencja jest zbyt restrykcyjna i zawodzi, jeżeli oś VPN jest równoległa do osi y, kiedy to rzut wektora VUP na płaszczyznę ruchu jest niezdefiniowany.)



Rys. 6.11. Rzutnia jest określana przez wektor VPN i punkt VRP; oś *v* jest określana przez rzut wektora VUP równolegle do VPN na rzutnię. Oś *u* tworzy prawoskrętny układ współrzędnych VRC razem z wektorem VPN i *v*



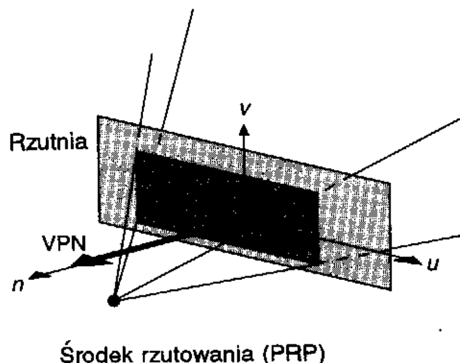
Rys. 6.12. Układ współrzędnych rzutowania (VRC) jest układem prawoskrętnym utworzonym przez osie *u*, *v* i *n*. Oś *n* jest zawsze określona przez wektor VPN. CW jest środkiem okna

Dla przyjętego układu VRC można określić współrzędne *u* i *v* okna, minimalne i maksymalne, tak jak na rys. 6.12. Z rysunku widać, że okno nie musi być symetryczne względem VRP; na rysunku środek okna jest oznaczony jako CW.

Środek rzutowania i kierunek rzutowania (DOP) są określone przez punkt odniesienia rzutowania (PRP) i wskaźnik rodzaju rzutu. Jeżeli mamy do czynienia z rzutem perspektywicznym, to PRP jest środkiem rzutowania. Jeżeli mamy rzutowanie równoległe, to kierunek DOP jest od punktu odniesienia rzutowania PRP do środka okna CW. W ogólnym przypadku punkt CW nie pokrywa się z punktem VRP, który nie musi być nawet w granicach okna.

Punkt PRP jest określony we współrzędnych rzutowania VRC, a nie we współrzędnych świata; dlatego pozycja PRP względem VRP nie zmienia się, gdy przesuwa się wektor VUP albo punkt VRP. Zaletą tego schematu jest to, że programista może określić potrzebny kierunek rzutowania i potem zmienić oś VPN i wektor VUP (zmieniając tym samym układ VRC), bez konieczności ponownego obliczania punktu PRP potrzebnego do uzyskania odpowiedniego rzutu. Przesuwanie punktu PRP w celu uzyskania różnych widoków obiektu może być jednak trudniejsze.

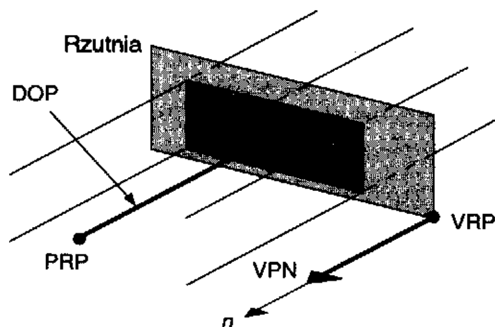
Bryła widzenia ogranicza tę część świata, która ma być wycięta i zrzutowana na rzutnię. Dla rzutu perspektywicznego bryła widzenia



Rys. 6.13. Otwarty ostrosłup widzenia dla rzutu perspektywicznego. CW jest środkiem okna

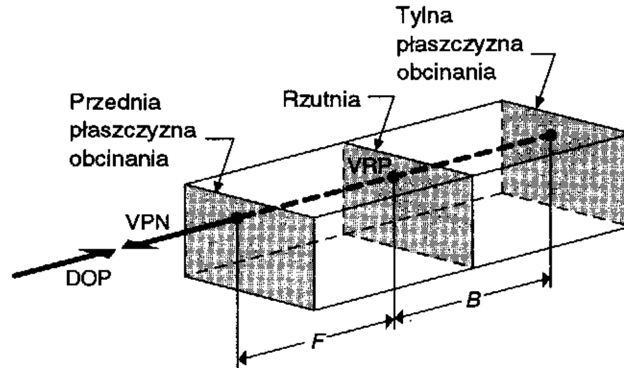
jest otwartym ostrosłupem z wierzchołkiem w PRP i krawędziach przechodzących przez rogi okna. Na rysunku 6.13 pokazano bryłę widzenia dla rzutu perspektywicznego.

Obszar z tyłu za środkiem rzutu nie jest włączany do bryły widzenia i nie jest w związku z tym rzutowany. W rzeczywistości nasze oczy widzą nieregularnie ukształtowaną bryłę widzenia zbliżoną do stożka. Ostrosłup widzenia jest jednak łatwiejszy z punktu widzenia matematycznego i jest spójny z koncepcją prostokątnego pola wizualizacji.

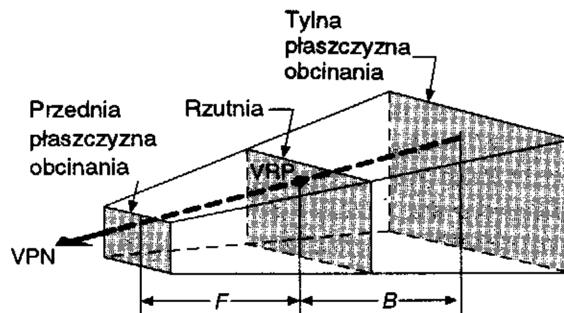


Rys. 6.14. Nieskończony równoległościan widzenia dla rzutu równoległego prostokątnego. Wektor VPN i kierunek rzutowania (DOP) są równoległe. Wektor DOP jest skierowany od PRP do CW i jest równoległy do wektora VPN

Dla rzutów równoległych bryła widzenia jest nieskończonym równoległościanem o bokach równoległych do kierunku rzutowania; jest to kierunek od punktu PRP do środka okna. Na rysunku 6.14 pokazano bryłę widzenia dla rzutu równoległego i jej relację z rzutnią, oknem i punktem PRP.



Rys. 6.15. Obcięta bryła widzenia dla rzutu równoległego prostokątnego. DOP jest kierunkiem rzutu



Rys. 6.16. Obcięta bryła widzenia dla rzutu perspektywicznego

Niestety ze względu na chęć ograniczenia liczby rzutowanych prymitywów wyjściowych może nam zależeć na tym, żeby bryła widzenia była skończona. Na rysunkach 6.15 i 6.16 pokazano, jak można ograniczyć bryłę widzenia za pomocą płaszczyzn obcinających przedniej i tylnej (bliższej i dalszej). Te płaszczyzny są równoległe do rzutni. Normalna do nich to wektor VPN. Płaszczyzny te są określone przez wielkości ze znakiem: odległość przednia (F) i odległość tylna (B) odniesione do VRP i skierowane zgodnie z wektorem VPN (zwrot wektora wskazuje kierunek wzrostu odległości). Na to, żeby bryła widzenia nie była pusta, odległość przednia musi być algebraicznie większa od odległości tylnej.

Takie ograniczenie bryły widzenia może być przydatne przy eliminowaniu niepotrzebnych obiektów i umożliwia użytkownikowi skoncentrowanie się na określonym fragmencie świata. Dynamiczna modyfikacja przedniej lub tylnej odległości może dać obserwatorowi dobre wrażenie o przestrzennych zależnościach między różnymi częściami obiektu wówczas, gdy te części pojawiają się i giną z pola wizualizacji (por. rozdz. 12). Dla rzutów perspektywicznych dochodzi dalsza mo-

tywacja. Obiekt bardzo odległy od środka rzutowania pojawia się na rzutni jako plama o nieokreślonym kształcie. Przy wyprowadzaniu takiego obiektu na ploter piórko może przedrzeć papier; na monitorze wektorowym luminofor CRT może zostać wypalony przez strumień elektronów; a na wektorowym rejestratorze filmów wskutek dużej koncentracji światła mogą pojawić się rozmyte białe obszary. Również obiekt leżący bardzo blisko środka rzutowania może rozszerzyć się na całe okno. Właściwe określenie bryły widzenia może wyeliminować takie problemy.

W jaki sposób zawartość bryły widzenia jest odwzorowywana na rzutnię? Najpierw weźmy pod uwagę sześcian jednostkowy rozciągający się od 0 do 1 w każdym z trzech kierunków *znormalizowanych współrzędnych rzutowania* (NPC). Bryła widzenia jest przekształcana w prostopadłościan w układzie NPC o współrzędnych: od x_{min} do x_{max} w kierunku osi x, od y_{min} do y_{max} w kierunku osi y i od z_{min} do z_{max} w kierunku osi z. Przednia płaszczyzna obcinania staje się płaszczyzną z_{min}. Podobnie ściana u_{min} bryły widzenia staje się płaszczyzną x_{min}, a ściana u_{max} staje się płaszczyzną x_{max}. Wreszcie ściana v_{min} bryły widzenia staje się płaszczyzną y_{min}, a bok v_{max} staje się płaszczyzną y_{max}. Ten prostopadłościan we współrzędnych NPC znajduje się w sześcianie jednostkowym i jest określany jako pole wizualizacji 3D.

Z kolei ściana $z = 1$ tego jednostkowego sześcianu jest odwzorowywana na największy kwadrat, jaki może zostać wyświetlony na ekranie. W celu utworzenia szkieletowego obrazu zawartości pola wizualizacji 3D (zawartości bryły widzenia) przy wyświetlaniu po prostu pomija się współrzędne z każdego prymitywu wyjściowego. W rozdziale 13 zobaczymy, że przy usuwaniu powierzchni niewidocznych wykorzystuje się współrzędną z do określenia, które prymitywy wyjściowe są bliższe obserwatora i wobec tego są widoczne.

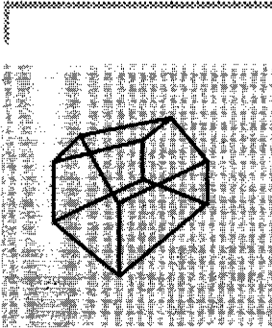
W standardzie PHIGS wykorzystuje się dwie macierze 4×4: macierz orientacji rzutu i macierz odwzorowania rzutu do reprezentowania pełnego zbioru parametrów rzutowania. VRP, VPN i VUP razem tworzą *macierz orientacji rzutu*, która przekształca pozycje reprezentowane we współrzędnych świata w pozycje reprezentowane w układzie VRC. Przy tym przekształceniu osie u, v i n przechodzą odpowiednio na osie x, y i z.

Parametry bryły widzenia określone przez PRP, u_{min}, u_{max}, y_{min}, F i B razem z parametrami pola wizualizacji 3D określonymi przez x_{min}, x_{max}, y_{min}, z_{min} i z_{max} tworzą *macierz odwzorowania rzutu*, która przekształca punkty z układu VRC w punkty w znormalizowanych współrzędnych rzutowania. Wywołania podprogramów, które tworzą macierz orientacji rzutu i macierz odwzorowania rzutu, są omawiane w p. 7.3.4.

W punkcie 6.4 pokażemy, jak uzyskać różne rzuty korzystając z koncepcji pokazanych w tym punkcie. W punkcie 6.5 wprowadzono

podstawową matematykę dla rzutów geometrii płaskiej, natomiast w p. 6.6 podano wiadomości z matematyki i algorytmy potrzebne dla całej operacji rzutowania.

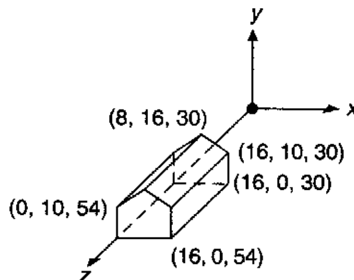
6.4. Przykłady rzutowania 3D



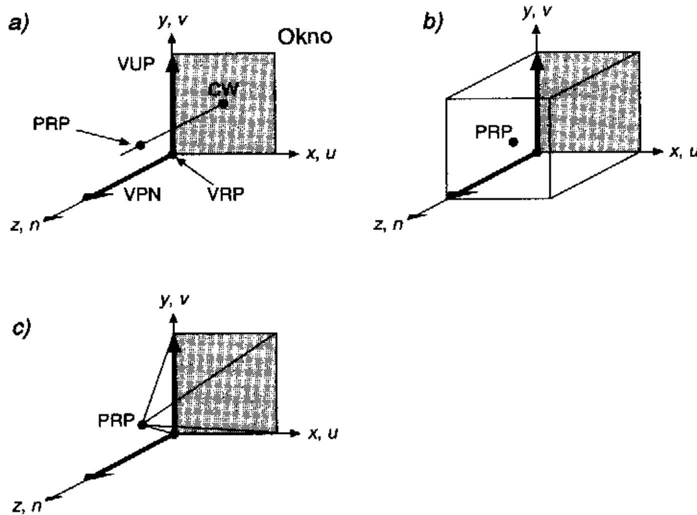
Rys. 6.17. Dwupunktowy rzut perspektywiczny domu

W tym punkcie zastanowimy się, jak można zastosować podstawowe koncepcje rzutowania wprowadzone w p. 6.3 do tworzenia różnych rzutów, takich jak na przykład ten z rys. 6.17. Ponieważ z domu pokazanego na tym rysunku korzystamy w całym tym punkcie, warto zapamiętać jego wymiary i położenie (rys. 6.18). Dla każdego omawianego rzutu pokazujemy tablicę zawierającą parametry VRP, VPN, VUP, PRP, okno i rodzaj rzutu (perspektywiczny czy równoległy). W całym punkcie przyjmujemy domniemane pole wizualizacji 3D, którym jest sześcian jednostkowy w układzie NPC. W tablicy jest podawane oznaczenie (WC) albo (VRC), które ma przypominać, w jakim układzie współrzędnych są dane parametry rzutowania. Postać tej tablicy zawiera domniemane wartości jak w standardzie PHIGS. Bryłę widzenia odpowiadającą tym domniemanym wartościom pokazano na rys. 6.19b. Gdybyśmy użyli rzutu perspektywicznego zamiast równoległego, wówczas bryła widzenia byłaby ostrosłupem pokazanym na rys. 6.19c.

Parametr rzutowania	Wartość	Komentarz
VRP(WC)	(0, 0, 0)	początek układu
VPN(WC)	(0, 0, 1)	oś z
VUP(WC)	(0, 1, 0)	oś y
PRP(VRC)	(0,5, 0,5, 1,0)	
okno(VRC)	(0, 1, 0, 1)	
rodzaj rzutu	równoległy	



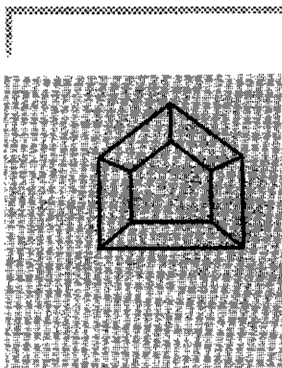
Rys. 6.18. Ten dom jest używany w tym rozdziale jako przykład zestawu danych we współrzędnych świata. Jego współrzędne zawierają się w przedziałach od 30 do 54 dla osi z, od 0 do 16 dla osi x i od 0 do 16 dla osi y



Rys. 6.19. Związek między współrzędnymi układu rzutowania a współrzędnymi układu świata: a) domniemane parametry widzenia: punkt VRP jest w początku układu współrzędnych, wektor VUP pokrywa się z osią y, a normalna VPN z osią z; przy takim ustawieniu układ VRC ze współrzędnymi u, v i n pokrywa się z układem współrzędnych świata x, y, z; okno rozciąga się od 0 do 1 wzdłuż u i v, a punkt PRP ma współrzędne (0,5, 0,5, 1,0); b) domniemana bryła widzenia dla rzutu równoległego; c) bryła widzenia dla rzutu perspektywicznego

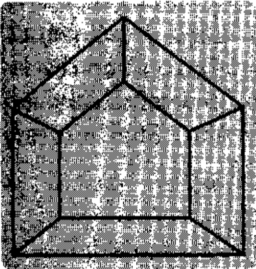
6.4.1. Rzuty perspektywiczne

W celu otrzymania rzutu perspektywicznego domu z jednym punktem zbieżności, pokazanego na rysunku 6.20, na przednią płaszczyznę bryły widzenia (ten i inne podobne rysunki są wykonane za pomocą programu SPHIGS omawianego w rozdz. 7) umieszczamy środek rzutowania (który może być traktowany jako pozycja obserwatora) w punkcie $x = 8$, $y = 6$ i $z = 84$. Wartość x jest tak wybrana, żeby była w środku poziomego wymiaru domu, a wartość y jest tak wybrana, żeby odpowiadała przybliżonemu poziomowi oka obserwatora stojącego na płaszczyźnie (x, z); wartość z jest dowolna. W tym przypadku wartość z jest przesunięta o 30 jednostek przed dom (płaszczyzna $z = 54$). Okno wybrano na tyle duże, żeby uzyskać gwarancję, że dom zmieści się w bryle widzenia. Wszystkie inne parametry rzutowania mają wartości domniemane i cały zbiór parametrów rzutowania wygląda następująco:



Rys. 6.20. Rzut domu w perspektywie jednopunktowej

VRP(WC)	(0, 0, 0)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	(8, 6, 84)
okno(VRC)	(−50, 50, −50, 50)
rodzaj rzutu	perspektywiczny

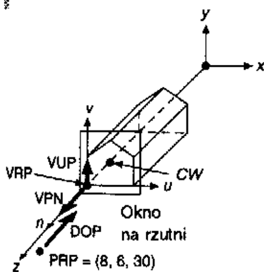


Rys. 6.21. Wypośrodkowany rzut perspektywiczny domu

Chociaż obraz z rys. 6.20 jest naprawdę perspektywicznym rzutem domu, jest bardzo mały i nie znajduje się w środku rzutni. Wolelibyśmy, żeby obraz był umieszczony pośrodku i dokładniej wypełniał rzutnię, tak jak na rys. 6.21. Efekt ten można najłatwiej uzyskać wówczas, gdy rzutnia i przednia ściana domu pokrywają się. Teraz przód domu rozciąga się wzdłuż osi x i y od wartości 0 do wartości 16, a okno wzdłuż osi x i y od -1 do 17; jest to wynik rozsądny.

Umieszczamy rzutnię na płaszczyźnie ściany przedniej domu przyjmując punkt VRP gdzieś na płaszczyźnie $z=54$; (0, 0, 54) jest dobrym wyborem dla lewego dolnego przedniego rogu domu. Na to, żeby środek rzutu był taki sam jak na rys. 6.20, punkt PRP określony w układzie VRC powinien mieć współrzędne (8, 6, 30). Na rysunku 6.22 pokazano nowy zestaw parametrów rzutowania:

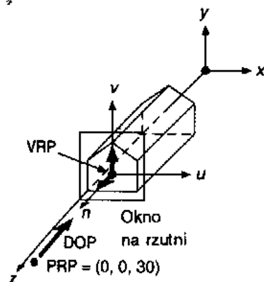
VRP(WC)	(0, 0, 54)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	(8, 6, 30)
okno(VRC)	(−1, 17, −1, 17)
rodzaj rzutu	perspektywiczny



Rys. 6.22. Parametry rzutowania dla rys. 6.21

Ten sam wynik można otrzymać na wiele innych sposobów. Na przykład, dla punktu VRP o współrzędnych (8, 6, 54) (rys. 6.23) środek rzutu określony przez PRP jest w punkcie (0, 0, 30). Trzeba również zmienić okno, ponieważ jego definicja bazuje na układzie VRC, którego początkiem jest punkt VRP. Odpowiednie okno rozciąga się od -9 do 9 wzdłuż osi u i od -7 do 11 wzdłuż osi v. W odniesieniu do domu jest to to samo okno, jakie było użyte w poprzednim przykładzie, ale jest teraz określone w innym układzie VRC. Ponieważ kierunek do góry jest zgodny z osią y, osie u i x są równoległe, podobnie jak osie v i y. W sumie, przy parametrach z rys. 6.23 można otrzymać rys. 6.21:

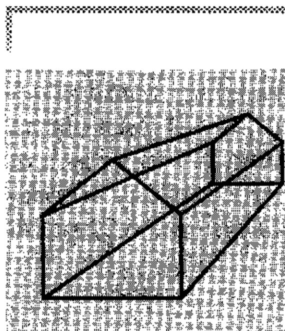
VRP(WC)	(8, 6, 54)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	(0, 0, 30)
okno(VRC)	(−9, 9, −7, 11)
rodzaj rzutu	perspektywiczny



Rys. 6.23. Alternatywne parametry rzutowania dla rys. 6.21

Spróbujmy teraz otrzymać rzut perspektywiczny z dwoma punktami zbieżności pokazany na rys. 6.17. Środek rzutowania jest analogiem pozycji kamery, która robi zdjęcia obiektów świata rzeczywistego. Pamiętając o tej analogii, wydaje się, że środek rzutowania na rys. 6.17 jest trochę powyżej i na prawo od domu, gdy patrzymy wzdłuż dodat-

niej osi z. Dokładny środek rzutowania jest w punkcie (36, 25, 74). Teraz, jeżeli punkt VRP zostanie umieszczony w rogu domu (16, 0, 54), to środek rzutowania jest w (20, 25, 20) względem niego. Jeżeli rzutnia pokrywa się z przodem budynku (płaszczyzna $z = 54$), to okno rozciągające się od -20 do 20 wzdłuż osi u i od -5 do 35 wzdłuż osi v jest dostatecznie duże na to, żeby rzut zmieścił się w nim. Stąd możemy określić następujące parametry dla rzutu z rys. 6.24:



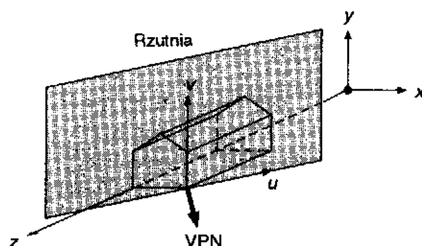
Rys. 6.24. Rzut perspektywiczny domu z (36, 25, 74) z punktu VPN; wektor normalny jest równoległy do osi z

VRP(WC)	(16, 0, 54)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	(20, 25, 20)
okno(VRC)	$(-20, 20, -5, 35)$
rodzaj rzutu	perspektywiczny

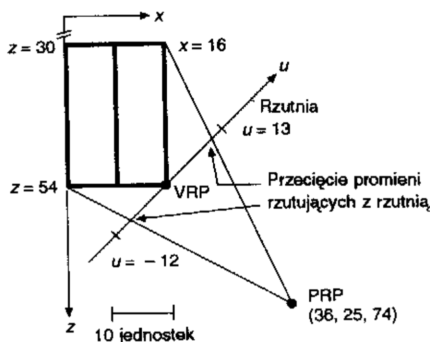
Obraz jest podobny do pokazanego na rys. 6.17, ale oczywiście nie jest taki sam. Jedyna różnica polega na tym, że rys. 6.17 pokazuje rzut perspektywiczny z dwoma punktami zbieżności, a rys. 6.24 – rzut perspektywiczny z jednym punktem zbieżności. Widać, że zwykłe przesunięcie środka rzutowania nie wystarcza do utworzenia rys. 6.17. W istocie, musimy tak zmienić orientację rzutni, żeby przecinała obie osie x i y, wybierając wektor VPN o współrzędnych (1, 0, 1). Stąd parametry rzutowania dla rys. 6.17 są następujące:

VRP(WC)	(16, 0, 54)
VPN(WC)	(1, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	$(0, 25, 20\sqrt{2})$
okno(VRC)	$(-20, 20, -5, 35)$
rodzaj rzutu	perspektywiczny

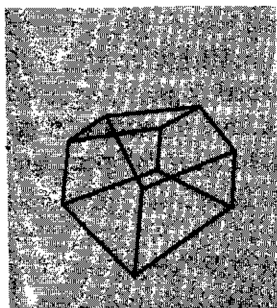
Na rysunku 6.25 pokazano rzutnię otrzymaną dla tej normalnej VPN. Współrzędna $20\sqrt{2}$ współrzędnej n punktu PRP zapewnia to, że środek rzutowania jest w odległości $20\sqrt{2}$ od punktu VRP w płaszczyźnie (x, y) (rys. 6.26).



Rys. 6.25. Rzutnia i układ współrzędnych VRC odpowiadający rys. 6.17



Rys. 6.26. Rzut z góry domu dla określenia odpowiedniej wielkości okna



Rys. 6.27. Rzut domu uzyskany po obrocie wektora VUP

Są dwa sposoby wybrania okna, które całkowicie zawiera rzut, tak jak w przypadku okna na rys. 6.17. Możemy oszacować wielkość rzutu domu na rzutnię wykonując szkic taki jak na rys. 6.26, w celu obliczenia przecięć promieni rzutujących z rzutnią. Lepszym rozwiązaniem jest przyjęcie w programie zmiennych granic okna, które są określane interakcyjnie za pomocą urządzeń typu lokalizator lub urządzenia do wprowadzania wartości.

Rysunek 6.27 powstał w wyniku tego samego rzutu co rys. 6.17, z tą różnicą, że okno ma inną orientację. We wszystkich poprzednich przykładach oś v układu VRC była równoległa do osi y układu współrzędnych świata; dlatego okno (którego dwa boki są równoległe do osi v) było dobrze ustawione w stosunku do pionowych boków domu. Na rysunku 6.27 są dokładnie takie same parametry widzenia jak na rys. 6.17, przy czym wektor VUP został obrócony w stosunku do osi y o kąt ok. 10°.

Przykład 6.1

Problem: Po ustaleniu układu współrzędnych VRC wszystkie kolejne operacje graficzne są wykonywane w tym układzie; procedura ta będzie omawiana dokładniej w p. 6.6. Przed tymi krokami przetwarzania musimy przejść ze współrzędnych świata naszych danych na współrzędne VRC. To przekształcenie może być wykonane za pomocą jednej macierzy, która realizuje zarówno obrót, jak i przesunięcie. Jaka jest ogólna postać takiej macierzy? Jakie konkretne wartości powinny mieć elementy tej macierzy dla rzutowania z rys. 6.17?

Odpowiedź Podejście, którym się zajmiemy było zasugerowane w p. 5.8 i zilustrowane równaniami od (5.60) do (5.64). Wtedy dowolny zbiór odcinków był przesuwany i obracany do nowej pozycji w układzie współrzędnych x, y, z za pomocą macierzy M będącej złożeniem macierzy przesunięcia T i macierzy R. Chcemy skorzystać z tej samej procedury, lecz

tym razem chcemy zmienić układ współrzędnych u, v, n tak, żeby zgadzał się z układem współrzędnych świata. Macierz, która wykonuje te przekształcenia, jest szukaną macierzą.

Po pierwsze, musimy przesunąć układ VRC do początku układu współrzędnych. Podobnie jak w p. 5.8 wykonujemy to przesunięcie za pomocą macierzy T o postaci

$$\begin{bmatrix} 1 & 0 & 0 & -\mathrm{VRP}_x \\ 0 & 1 & 0 & -\mathrm{VRP}_y \\ 0 & 0 & 1 & -\mathrm{VRP}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Użyjemy tutaj podejścia takiego jak w p. 5.8, zgodnie z którym szukamy macierzy obrotu (specjalnej macierzy ortogonalnej) określając, gdzie powinny być trzy główne osie. Elementami takiej macierzy są składowe wektorów jednostkowych, które leżą wzdłuż kierunków u, v i n. Stąd możemy znaleźć elementy macierzy obrotu R zauważając, że wektor VPN ma być obrócony na oś z, oś u ma być prostopadła do wektorów VUP i VPN, a oś v ma być prostopadła do n i u. Wobec tego:

$$n = \frac{\mathrm{VPN}}{\|\mathrm{VPN}\|}, \quad u = \frac{\mathrm{VUP} \times \mathrm{VPN}}{\|\mathrm{VUP} \times \mathrm{VPN}\|}, \quad v = n \times u$$

przy czym u, v i n reprezentują wektory jednostkowe. Wynikowa macierz obrotu ma postać

$$R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

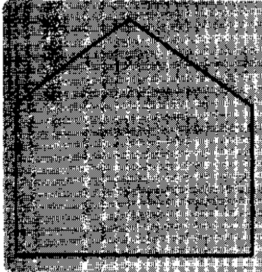
Stąd macierz, której szukamy ma postać

$$M = R \cdot T = \begin{bmatrix} u_x & u_y & u_z & -(u_x \cdot \mathrm{VRP}_x + u_y \cdot \mathrm{VRP}_y + u_z \cdot \mathrm{VRP}_z) \\ v_x & v_y & v_z & -(v_x \cdot \mathrm{VRP}_x + v_y \cdot \mathrm{VRP}_y + v_z \cdot \mathrm{VRP}_z) \\ n_x & n_y & n_z & -(n_x \cdot \mathrm{VRP}_x + n_y \cdot \mathrm{VRP}_y + n_z \cdot \mathrm{VRP}_z) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

W celu określenia wartości, które odpowiadają rys. 6.17, zuważamy, że $n = [\sqrt{2}/2, 0, \sqrt{2}/2]^T$, $u = [\sqrt{2}/2, 0, -\sqrt{2}/2]^T$ oraz $v = [0, 1, 0]^T$.

Ponieważ współrzędnymi punktu VRP są 16,0, 0,0 i 54, można zauważyć, że wyrazami macierzy M odpowiadającymi za przesuniecie są 26,8701, 0,0 i −49,4975.

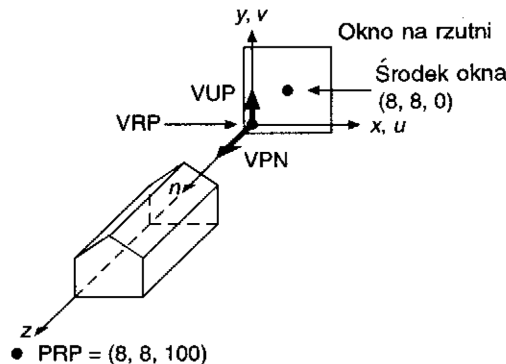
6.4.2. Rzuty równoległe



Rys. 6.28. Czołowy rzut równoległy domu

Przyjmując, że kierunek rzutowania jest równoległy do osi z, tworzymy czołowy rzut równoległy domu (rys. 6.28). Przypomnijmy, że kierunek rzutowania jest określony przez punkt PRP i przez środek okna. Dla domniemanego układu VRC i okna (−1, 17, −1, 17) środek okna ma współrzędne (8, 8, 0). Punkt PRP o współrzędnych (8, 8, 100) określa kierunek rzutowania równoległy do osi z. Na rysunku 6.29 pokazano sytuację wyjściową dla rzutu z rys. 6.28. Parametry rzutowania są następujące:

VRP(WC)	(0, 0, 0)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	(8, 8, 100)
okno(VRC)	(−1, 17, −1, 17)
rodzaj rzutu	równoległy



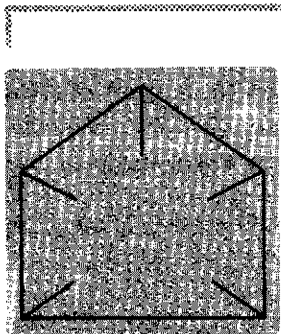
Rys. 6.29. Parametry rzutowania dla rys. 6.28 z czołowym rzutem domu: Punkt PRP może być w dowolnym miejscu, dla którego $x = 8$, $y = 8$

W celu utworzenia rzutu bocznego powinniśmy jako rzutni użyć płaszczyzny (x, z) (albo innej płaszczyzny do niej równoległej). Górny rzut domu tworzymy przy wykorzystaniu płaszczyzny (x, z) jako rzutni i wektora VPN jako osi y. Trzeba jednak zmienić domniemany kierunek dodatniej osi $+y$. Zamiast tego wykorzystamy ujemną oś x.

W pracy [FOLE90] można znaleźć szczegółowe omówienie przypadków rzutów bocznego i górnego, a także przykłady rzutu ukośnego.

6.4.3. Skończone bryły widzenia

We wszystkich dotychczasowych przykładach zakładaliśmy, że bryła widzenia jest nieskończona. Przednia i tylna płaszczyzna obcinania, opisane w p. 6.3, pomagają określić skończoną bryłę widzenia. Obie te płaszczyzny są równoległe do rzutni i znajdują się w odległościach F i B od punktu VRP, mierząc wzdłuż wektora VPN. Aby uniknąć ujemnych brył widzenia, musimy zapewnić, żeby F było algebraicznie większe od B.



Rys. 6.30. Rzut perspektywiczny domu z tylną płaszczyzną obcinającą $z = 31$

Czołowy rzut perspektywiczny domu z odciętą tylną ścianą (rys. 6.30) otrzymuje się dla następujących parametrów rzutowania, do których zostały dodane wartości F i B. Jeżeli dana jest odległość, to zakłada się, że obcinanie następuje przez odpowiednią płaszczyznę; w przeciwnym przypadku nie ma obcinania. Parametry rzutowania są następujące:

VRP(WC)	(0, 0, 54)	dolny lewy róg domu
VPN(WC)	(0, 0, 1)	oś z
VUP(WC)	(0, 1, 0)	oś y
PRP(VRC)	(8, 6, 30)	
okno(VRC)	(−1, 17, −1, 17)	
rodzaj rzutu	perspektywiczny	
F(VRC)	+1	jedna jednostka z przodu domu dla $z = 54 + 1 = 55$
B(VRC)	−23	jedna jednostka z tyłu domu dla $z = 54 - 23 = 31$

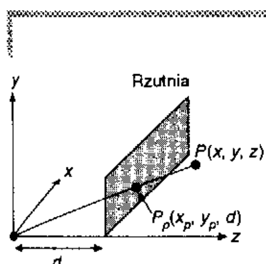
W tym przypadku konfiguracja rzutowania jest taka sama jak na rys. 6.22, przy czym zostały dodane płaszczyzny obcinające.

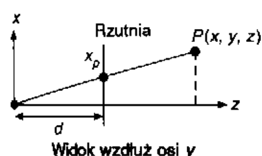
Poruszajac płaszczyznami obcinającymi przednią i tylną można zapewnić lepsze rozpoznanie struktury 3D rzutowanego obiektu niż przy rzucie statycznym.

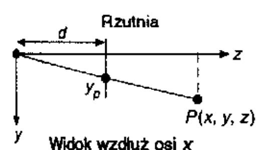
6.5. Płaskie rzuty geometryczne

Podamy teraz podstawowe wiadomości z matematyki związane z płaskimi rzutami geometrycznymi. Dla uproszczenia założymy na początku, że w rzucie perspektywicznym rzutnia jest prostopadła do osi z w punkcie $z = d$ oraz że w rzucie równoległym rzutnią jest płaszczyzna $z = 0$. Każdy z rzutów może być określony przez macierz 4×4. Taka reprezentacja jest wygodna, ponieważ macierz rzutowania może być złożona z macierzy przekształceń, co umożliwia reprezentowanie dwóch operacji (przesunięcia i obrotu) za pomocą jednej macierzy. W punkcie 6.6 omawiamy dowolne rzutnie.

W tym punkcie wyprowadzamy macierze 4 × 4 dla kilku rzutów, zaczynając od rzutni równoległej do płaszczyzny xy i przecinającej oś z w punkcie $z = d$, a więc w odległości $|d|$ od początku układu współrzędnych; punkt P ma być zrzutowany na tę płaszczyznę. W celu obliczenia $P_p = (x_p, y_p, z_p)$ – rzutu perspektywicznego (x, y, z) na rzutnię dla $z = d$, korzystamy z podobieństwa trójkątów (rys. 6.31); stąd można napisać proporcje







Rys. 6.31. Rzut perspektywiczny

$$\frac{x_p}{d} = \frac{x}{z}, \quad \frac{y_p}{d} = \frac{y}{z} \tag{6.1}$$

Mnożąc obie strony przez d otrzymujemy

$$x_p = \frac{dx}{z} = \frac{x}{z/d}, \quad y_p = \frac{dy}{z} = \frac{y}{z/d} \tag{6.2}$$

Odległość d jest po prostu współczynnikiem skalowania zastosowanym do x_p i y_p. Dzielenie przez z powoduje, że rzut perspektywiczny odleglejszych obiektów jest mniejszy niż bliższych obiektów. Dopuszczalne są wszystkie wartości z oprócz $z = 0$. Punkty mogą być poza środkiem rzutowania na ujemnej osi z albo między środkiem rzutowania a płaszczyzną rzutowania.

Równanie (6.2) może być przedstawione w postaci macierzy 4 × 4

$$M_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \tag{6.3}$$

Mnożąc współrzędne punktu $P = [x \ y \ z \ 1]^T$ przez macierz M_{per} otrzymujemy punkt we współrzędnych jednorodnych $[X \ Y \ Z \ W]^T$

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = M_{per} \cdot P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6.4}$$

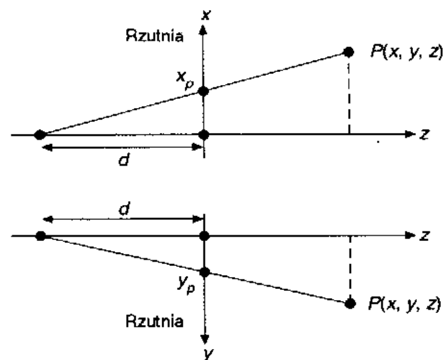
albo

$$[X \ Y \ Z \ W]^T = \left[x \ y \ z \ \frac{z}{d} \right]^T \tag{6.5}$$

Dzieląc przez $W (W = z/d)$ i pomijając czwartą współrzędną powracamy do przestrzeni 3D i otrzymujemy

$$\left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}\right) = (x_p, y_p, z_p) = \left(\frac{x}{z/d}, \frac{y}{z/d}, d\right) \tag{6.6}$$

Są to poprawne wyniki zgodne z równaniem (6.1) plus przekształcona współrzędna z o d, przy czym d określa położenie rzutni na osi z.



Rys. 6.32. Alternatywny rzut perspektywiczny

Przy alternatywnym sformułowaniu rzutu perspektywicznego rzutnia znajduje się w punkcie $z = 0$ i środek rzutowania jest w punkcie $z = -d$ (rys. 6.32). Z podobieństwa trójkątów otrzymujemy:

$$\frac{x_p}{d} = \frac{x}{z+d}, \qquad \frac{y_p}{d} = \frac{y}{z+d} \tag{6.7}$$

Mnożąc przez d otrzymujemy:

$$x_p = \frac{d \cdot x}{z+d} = \frac{x}{(z/d)+1}, \qquad y_p = \frac{d \cdot y}{z+d} = \frac{y}{(z/d)+1} \tag{6.8}$$

Macierz ma postać

$$M'_{\text{per}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \tag{6.9}$$

Przy tym sformułowaniu d, czyli odległość środka rzutowania, może wzrastać do nieskończoności.

Rzut prostokątny na rzutnię $z = 0$ jest oczywisty. Kierunek rzutowania jest zgodny z normalną do rzutni, w tym przypadku z osią z. Stąd rzut punktu P jest następujący:

$$x_p = x, \quad y_p = y, \quad z_p = 0 \tag{6.10}$$

Ten rzut jest wyrażony przez macierz

$$M_{\text{ort}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.11}$$

Zauważmy, że jeżeli d w równaniu (6.9) dąży do nieskończoności, to równanie (6.9) staje się równaniem (6.11). Jest tak, ponieważ rzut prostokątny jest specjalnym przypadkiem rzutu perspektywicznego.

Macierz M_{per} stosuje się tylko w specjalnym przypadku, gdy środek rzutowania jest w początku układu współrzędnych; macierz M_{ort} stosuje się tylko wówczas, gdy kierunek rzutowania jest równoległy do osi z. Ogólniejsze sformułowanie podane w pracy [FOLE90] nie tylko usuwa te ograniczenia, ale łączy rzuty równoległy i perspektywiczny w jednym wzorze.

W tym punkcie zobaczyliśmy, jak można określić macierze M_{per}, M'_{per} i M_{ort}, słuszne przy założeniu, że rzutnia jest prostopadła do osi z. W punkcie 6.6 usuwamy to ograniczenie i zajmujemy się obcinaniem wynikającym z ograniczonej bryły widzenia.

Przykład 6.2 **Problem**: Macierz M_{per} definiuje rzut perspektywiczny jednopunktowy. Podaj macierz dla rzutu perspektywicznego dwupunktowego i jej związek z macierzą M_{per}, którą właśnie wyprowadziliśmy. Jak wygląda macierz dla perspektywy trzypunktowej?

Odpowiedź Jak to sugerowano w p. 6.4.1, musimy w tym przypadku tak zorientować rzutnię, żeby przecinała więcej niż jedną oś – w tym przypadku oś z. Na przykład określimy obrót wokół osi y tak, żeby rzutnia przecięła obie osie x i z. Nową macierz otrzymamy mnożąc macierz M_{per} przez macierz

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

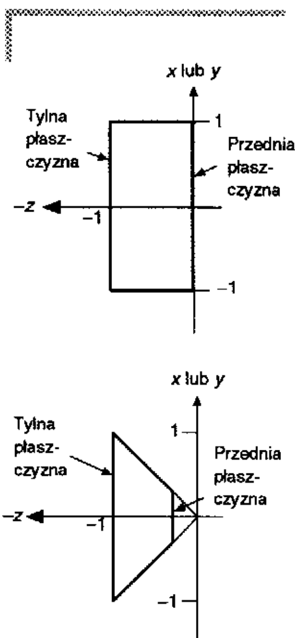
przy czym θ jest kątem obrotu wokół osi y. Wynikowa macierz ma postać

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ -\sin\theta/d & 0 & \cos\theta/d & 0 \end{bmatrix}$$

Zauważmy pojawienie się niezerowego elementu na pozycji a_{41} w wynikowej macierzy. Oznacza to istnienie punktu zbieżności na osi x. Gdybyśmy mieli wykonać podobną operację z obrotem wokół osi x, wówczas niezerowy element na pozycji a_{42} wskazywałby na punkt zbieżności na osi y. Łącząc obroty wokół osi x i y otrzymalibyśmy perspektywę trzypunktową.

6.6. Implementacja płaskich rzutów geometrycznych

Załóżmy, że dane są bryła widzenia oraz rodzaj rzutu i zastanówmy się, jak wykonuje się operacje obcinania i rzutowania. Jak wynika z koncepcyjnego modelu procesu rzutowania (rys. 6.2), moglibyśmy obcinać odcinki przez bryłę widzenia na zasadzie obliczania ich przecięć z każdą z sześciu płaszczyzn, które określają bryłę widzenia. Odcinki, które pozostają po obcięciu, powinny być zrzutowane na rzutnię na zasadzie rozwiązania układu równań dla przecięcia promieni rzutujących z rzutnią. Współrzędne zostaną przekształcone z układu współrzędnych świata 3D w układ współrzędnych urządzenia 2D. Mamy tu do czynienia z dużą liczbą obliczeń powtarzanych wiele razy. Na szczęście jest bardziej efektywna procedura, wykorzystująca strategię typu dziel i zwyciężaj, umożliwiająca podzielenie trudnego problemu na prostsze problemy.



Rys. 6.33. Dwie kanoniczne bryły widzenia dla rzutów: a) równoległego; b) perspektywicznego

Obcinanie dla niektórych brył widzenia jest łatwiejsze niż obcinanie dla dowolnej bryły widzenia (algorytmy obcinania są omawiane w p. 6.6.3). Na przykład można łatwo policzyć przecięcie odcinka z każdą z płaszczyzn bryły widzenia dla rzutu równoległego, określonej przez sześć płaszczyzn:

$$x = -1, \quad x = 1, \quad y = -1, \quad y = 1, \quad z = 0, \quad z = -1 \qquad (6.12)$$

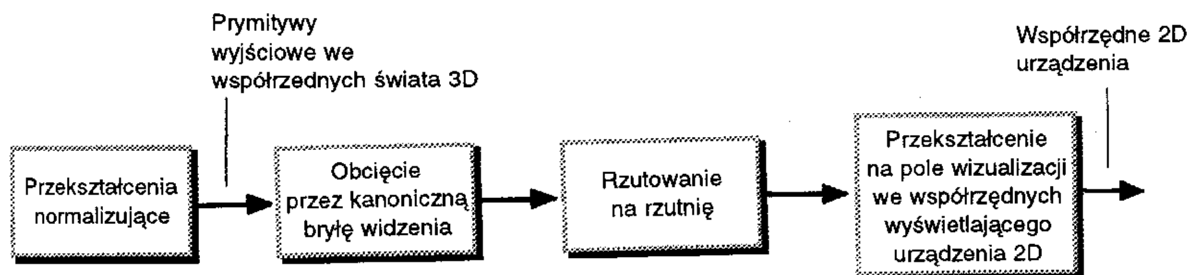
Jest to również prawdziwe dla rzutu perspektywicznego i bryły widzenia określonej przez płaszczyzny

$$x = z, \quad x = -z, \quad y = z, \quad y = -z, \quad z = -z_{\min}, \quad z = -1 \qquad (6.13)$$

Te *kanoniczne bryły widzenia* pokazano na rys. 6.33.

Nasza strategia polega na znalezieniu przekształceń normalizujących N_{par} i N_{per}, które przekształcają dowolną bryłę widzenia dla rzutu równoległego albo perspektywicznego w kanoniczne bryły widzenia, odpowiednio równoległą i perspektywiczną. Następnie jest wykonywane obcinanie, po którym następuje rzutowanie na płaszczyznę 2D za pomocą macierzy z p. 6.5. Przy takiej strategii ryzykuje się wysiłek wkładany w przekształcanie punktów, które potem są obcinane, ale przynajmniej obcinanie jest łatwe do wykonania.

Na rysunku 6.34 pokazano wykorzystywaną tu sekwencję. Możemy ją zredukować do sekwencji przekształcenie-obcinanie-przekształcenie łącząc kroki 3 i 4 w jedną macierz przekształcenia. Przy rzutach perspektywicznych potrzebne jest również dzielenie do odwzorowania współrzędnych jednorodnych z powrotem na współrzędne 3D. To dzielenie następuje po drugim przekształceniu skróconej sekwencji. Alternatywną strategię polegającą na obcinaniu we współrzędnych jednorodnych omówiono w p. 6.6.4.



Rys. 6.34. Implementacja rzutowania 3D

Czytelnicy znający standard PHIGS zauważą, że kanoniczne bryły widzenia z równań (6.12) i (6.13) różnią się od domniemanych brył widzenia standardu PHIGS: sześcian jednostkowy o bokach od 0 do 1 dla współrzędnych x, y, z dla równoległego rzutowania i ostrosłup o wierzchołku (0,5, 0,5, 1,0) i ścianach bocznych przechodzących przez boki kwadratu jednostkowego o współrzędnych od 0 do 1 na osiach x i y na płaszczyźnie $z = 0$ dla rzutu perspektywicznego. Kanoniczne bryły widzenia są określane w celu uproszczenia równań obcinania i uzyskania zgodności między rzutami równoległym i perspektywicznym omawianej w p. 6.6.4. W standardzie PHIGS domniemane bryły widzenia są zdefiniowane tak, żeby rzutowanie w 2D było szczególnym przypadkiem rzutowania 3D.

W punktach 6.6.1 i 6.6.2 wyprowadzimy przekształcenia normalizujące dla rzutów perspektywicznego i równoległego, które są używane w pierwszym kroku sekwencji przekształcenie-obcinanie-przekształcenie.

6.6.1. Przypadek rzutu równoległego

W tym punkcie wyprowadzamy przekształcenie normalizujące N_{par} dla rzutów równoległych w celu takiego przekształcania pozycji we współrzędnych świata, żeby bryła widzenia była przekształcana w kanoniczną bryłę widzenia określoną równaniem (6.12). Przekształcone współrzędne są obcinane przez tę kanoniczną bryłę widzenia, a wyniki obcinania są rzutowane na płaszczyznę $z = 0$ i potem przekształcane w pole wizualizacji w celu wyświetlenia.

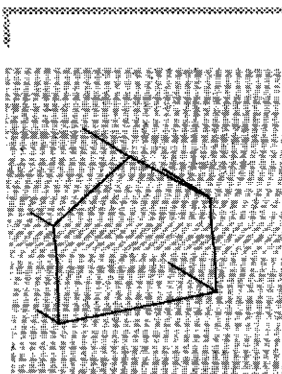
Przekształcenie N_{par} jest wyprowadzone dla najogólniejszego przypadku równoległego rzutu ukośnego (a nie prostokątnego). N_{par} zawiera przekształcenie pochylenia, które powoduje, że kierunek rzutowania we współrzędnych rzutowania jest równoległy do osi z, nawet jeżeli we współrzędnych (u, v, n) nie jest równoległy do wektora normalnego VPN. Włączając to pochylenie możemy wykonać rzutowanie na płaszczyznę $z = 0$, przyjmując po prostu $z = 0$. Jeżeli rzut równoległy jest rzutem prostokątnym, to składowa pochylenia w przekształceniu normalizującym staje się składową identycznościową.

Ciąg przekształceń składających się na N_{par} jest następujący:

1. Przesunięcie punktu VRP do początku układu współrzędnych.
2. Obrót układu VRC tak, żeby oś n (VPN) pokryła się z osią z, oś u pokryła się z osią x i oś v pokryła się z osią y.
3. Takie pochylenie, żeby kierunek rzutowania stał się równoległy do osi z.
4. Przesunięcie i przeskalowanie do kanonicznej bryły widzenia dla rzutu równoległego z równania (6.12).

W standardzie PHIGS kroki 1 i 2 definiują macierz orientacji rzutu, a kroki 3 i 4 definiują macierz odwzorowania rzutu.

Na rysunku 6.37 pokazano tę sekwencję przekształceń w odniesieniu do bryły widzenia dla rzutu równoległego i do konturu domu; na rysunku 6.35 pokazano wynikowy rzut równoległy.



Rys. 6.35. Końcowy rzut równoległy obciętego domu

Krok 1 to po prostu przekształcenie $T(-\mathrm{VRP})$. Dla kroku 2 korzystamy z właściwości specjalnych macierzy ortogonalnych omówionych w p. 5.3 i 5.7 i zilustrowanych przy wyprowadzaniu równań (5.64) i (5.65). Wektory wierszy macierzy obrotu dla wykonania kroku 2 są jednostkowymi wektorami, które są obracane przez R na osie x, y i z. Wektor VPN jest obracany na oś z, a więc

$$R_z = \frac{\mathrm{VPN}}{\|\mathrm{VPN}\|} \qquad (6.14)$$

Oś u jest prostopadła do wektorów VUP i VPN, jest zatem iloczynem wektorowym wektora jednostkowego VUP i R_z (który ma ten sam kierunek co wektor VPN) i jest obracana na oś x, a więc

$$R_z = \frac{\mathrm{VUP} \times R_z}{\|\mathrm{VUP} \times R_z\|} \tag{6.15}$$

Podobnie oś v prostopadła do R_z i R_x jest obracana na oś y, a więc

$$R_y = R_z \times R_x \tag{6.16}$$

Stąd obrót w kroku 2 jest dany przez macierz

$$R = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & 0 \\ r_{1y} & r_{2y} & r_{3y} & 0 \\ r_{1z} & r_{2z} & r_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.17}$$

przy czym r_{1x} jest pierwszym elementem R_x itd.

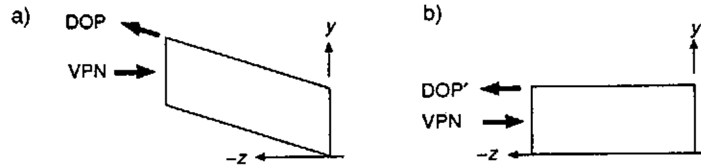
Trzeci krok polega na takim pochyleniu bryły widzenia wzdłuż osi z, żeby jej wszystkie płaszczyzny stały się prostopadłe do jednej z osi układu współrzędnych. Wykonujemy ten krok określając pochylenie, które ma być wykonane w kierunku rzutowania (DOP) tak, żeby kierunek DOP stał się zgodny z kierunkiem osi z. Przypomnijmy, że kierunek DOP jest określony przez wektor od punktu PRP do środka okna (CW) i że punkt PRP jest zadany w układzie VRC. Dwa pierwsze kroki przekształcenia doprowadziły układ VRC do zgodności z układem współrzędnych świata, tak że teraz również punkt PRP jest we współrzędnych świata. Dlatego kierunek DOP jest określony przez CW − PRP. Dla:

$$\mathrm{DOP} = \begin{bmatrix} dop_x \\ dop_y \\ dop_z \\ 0 \end{bmatrix}, \quad \mathrm{CW} = \begin{bmatrix} \frac{u_{\max} + u_{\min}}{2} \\ \frac{v_{\max} + v_{\min}}{2} \\ 0 \\ 1 \end{bmatrix}, \quad \mathrm{PRP} = \begin{bmatrix} prp_u \\ prp_v \\ prp_n \\ 1 \end{bmatrix} \tag{6.18}$$

mamy

$$\mathrm{DOP} = \mathrm{CW} - \mathrm{PRP} =$$

$$= \begin{bmatrix} \frac{u_{\max} + u_{\min}}{2} & \frac{v_{\max} + v_{\min}}{2} & 0 & 1 \end{bmatrix}^{\mathrm{T}} - [prp_u \; prp_v \; prp_n \; 1]^{\mathrm{T}} \tag{6.19}$$

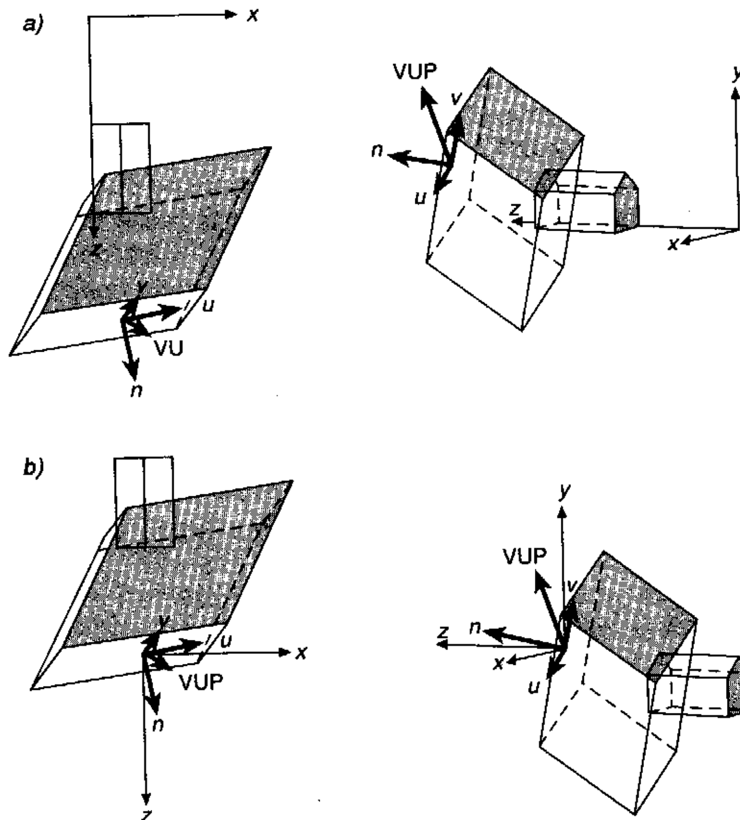


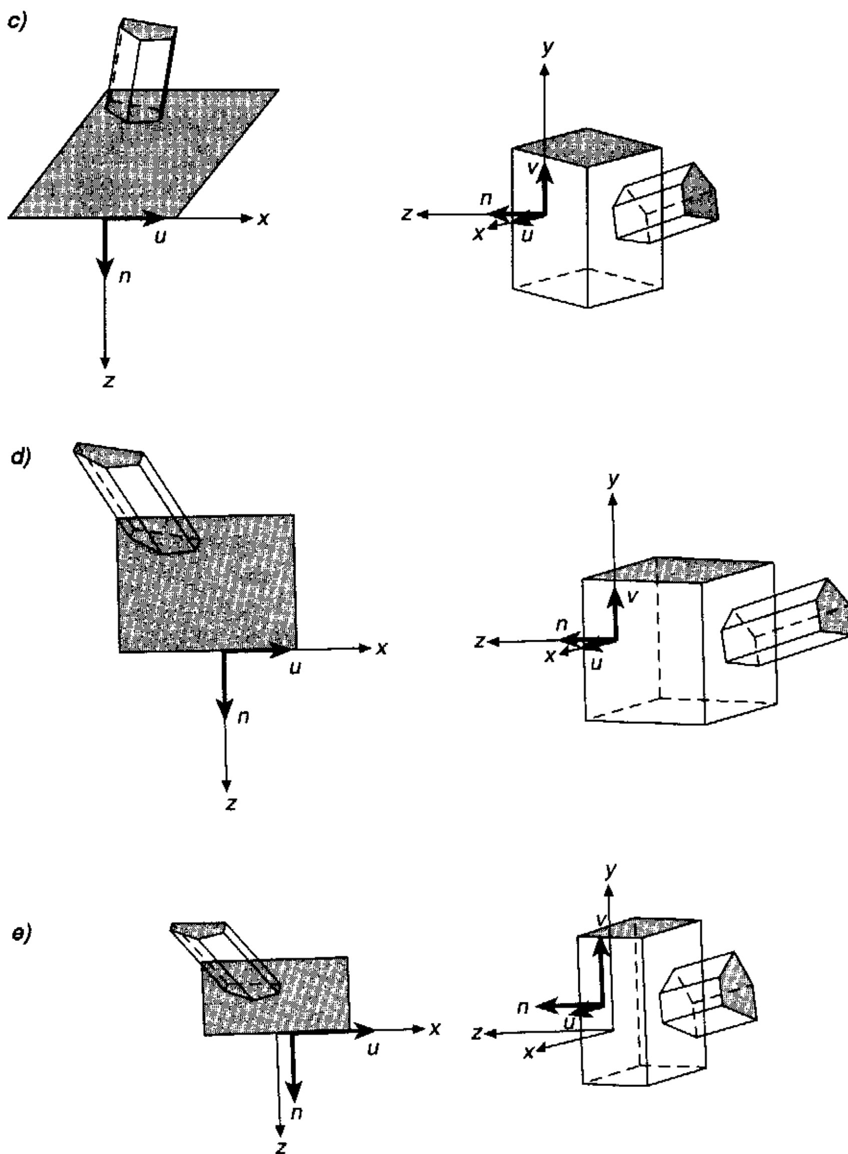
Rys. 6.36. Ilustracja pochylania na przykładzie bocznego rzutu bryły widzenia. Równoległobok z rysunku a) po pochyleniu staje się prostokątem z rysunku b); wektor VPN nie zmienia się, ponieważ jest równoległy do osi z

Tak określony kierunek DOP pokazano na rys. 6.36; pokazano również wymagany przekształcony kierunek DOP'.

Pochylenie można wykonać za pomocą macierzy pochylenia (x, y) z p. 5.7 – równanie (5.45). Dla współczynników shx_{par} i shy_{par} macierz ma postać

$$SH_{par} = SH_{xy}(shx_{par}, shy_{par}) = \begin{bmatrix} 1 & 0 & shx_{par} & 0 \\ 0 & 1 & shy_{par} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.20}$$





Rys. 6.37. Wyniki poszczególnych kroków sekwencji rzutowania przy rzucie równoległym. Za każdym razem jest pokazany widok z góry dla rzutu równoległego i z pozycji nie leżącej na osi: a) widok dla sytuacji początkowej; b) punkt VRP został przesunięty do początku układu współrzędnych; c) układ współrzędnych (u, v, n) został obrócony tak, żeby pokrył się z układem współrzędnych (x, y, z); d) bryła widzenia została tak pochylona, żeby kierunek rzutowania (DOP) stał się równoległy do osi z; e) bryła widzenia została przesunięta i przeskalowana do kanonicznej bryły widzenia dla rzutu równoległego. Parametry widzenia są następujące: VRP = (0,325, 0,8, 4,15), VPN = (0,227, 0,267, 1,0), VUP = (0,293, 1,0, 0,227), PRP = (0,6, 0,0, −1,0), Okno = (−1,425, 1,0, −1,0, 1,0), F = 0,0, B = −1,75. (Rysunki zostały wykonane za pomocą programu napisanego przez L. Lu z George Washington University.)

Jak opisano w p. 5.7, SH_{xy} nie zmienia wartości z, dodaje natomiast do x i y wyrazy $z \cdot shx_{par}$ i $z \cdot shy_{par}$. Chcemy znaleźć takie shx_{par} i shy_{par}, żeby

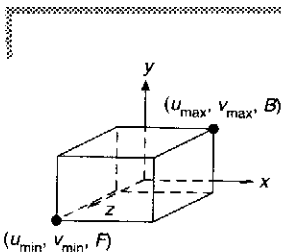
$$\text{DOP}' = [0\ 0\ dop_z\ 0]^T = SH_{par} \cdot \text{DOP} \tag{6.21}$$

Wykonując mnożenie z równania (6.21) oraz przekształcenia algebraiczne otrzymujemy, że równość pojawia się wówczas, gdy

$$shx_{par} = \frac{dop_x}{dop_z}, \qquad shy_{par} = \frac{dop_y}{dop_z} \tag{6.22}$$

Zauważmy, że dla rzutu prostokątnego, $dop_x = dop_y = 0$ i $shx_{par} = shy_{par} = 0$ i macierz pochylenia staje się macierzą jednostkową.

Na rysunku 6.38 pokazano bryłę widzenia po tych trzech krokach przekształcania. Bryła widzenia jest ograniczona przez:



Rys. 6.38. Bryła widzenia po trzech krokach przekształcania

$$u_{min} \leqslant x \leqslant u_{max}, \quad v_{min} \leqslant y \leqslant v_{max}, \quad B \leqslant z \leqslant F \tag{6.23}$$

przy czym F i B są odległościami od punktu VRP wzdłuż wektora VPN odpowiednio do przedniej i tylnej płaszczyzny obcinania.

Czwarty i ostatni krok procesu polega na przekszatałceniu pochylonej bryły widzenia w kanoniczną bryłę widzenia. Wykonujemy ten krok przesuwając środek przedniej ściany bryły widzenia z równania (6.23) do początku układu współrzędnych i potem skalując do wielkości $2 \times 2 \times 1$ końcowej kanonicznej bryły widzenia z równania (6.12). Przekształcenia są następujące:

$$T_{par} = T\left(-\frac{u_{max} + u_{min}}{2}, -\frac{v_{max} + v_{min}}{2}, -F\right) \tag{6.24}$$

$$S_{par} = S\left(\frac{2}{u_{max} + u_{min}}, \frac{2}{v_{max} + v_{min}}, \frac{1}{F - B}\right) \tag{6.25}$$

Gdyby wartości F i B nie były określone (ze względu na brak płaszczyzn obcinających przedniej i tylnej), można by użyć dowolnych wartości, które spełniają warunek $B < F$. Wartości 0 i 1 są dobre.

W sumie mamy

$$N_{par} = S_{par} \cdot T_{par} \cdot SH_{par} \cdot R \cdot T(-\text{VRP}) \tag{6.26}$$

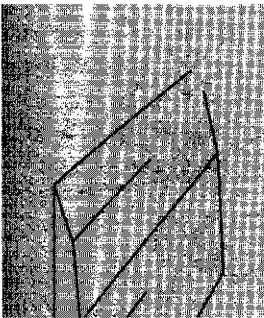
N_{par} dokonuje przekształcenia dowolnej bryły widzenia dla rzutu równoległego w kanoniczną bryłę widzenia dla rzutu równoległego, co umożliwia to, żeby prymitywy wyjściowe były obcinane przez kononiczną bryłę widzenia dla rzutu równoległego.

6.6.2. Przypadek rzutu perspektywicznego

Teraz wyprowadzimy znormalizowaną macierz przekształcenia N_{per} dla rzutów perspektywicznych. N_{per} tak przekształca położenie we współrzędnych świata, żeby bryła widzenia stała się kanoniczną bryłą widzenia dla rzutu perspektywicznego – ostrosłupem ściętym o wierzchołku w początku układu współrzędnych określonym równaniem (6.13). Po zastosowaniu macierzy N_{per} wykonuje się obcinanie względem tej bryły kanonicznej i wyniki są rzutowane na rzutnię za pomocą M_{per} (wyprowadzonej w p. 6.5).

Ciąg przekształceń składających się na N_{per} jest następujący:

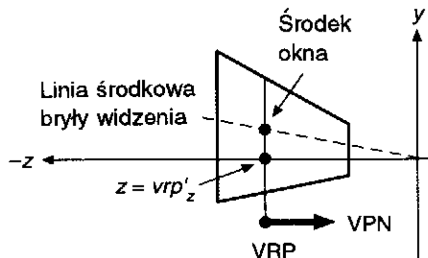
1. Przesunięcie punktu VRP do początku układu współrzędnych.
2. Taki obrót układu VRC, żeby oś n (VPN) pokryła się z osią z, oś u pokryła się z osią x i oś v pokryła się z osią y.
3. Takie przesunięcie, żeby środek rzutowania (COP), określony przez punkt PRP, stał się początkiem układu współrzędnych.
4. Takie pochylenie, żeby środkowa linia bryły widzenia pokryła się z osią z.
5. Takie skalowanie, żeby bryła widzenia stała się kanoniczną bryłą widzenia dla rzutu perspektywicznego – prawidłowym ostrosłupem ściętym, określonym przez sześć płaszczyzn z równania (6.13).



Rys. 6.39. Końcowy rzut perspektywiczny obciętego domu

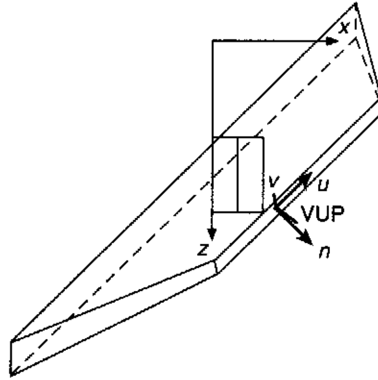
Ta sekwencja przekształceń została zastosowana do bryły widzenia dla rzutu perspektywicznego i do domu (rys. 6.41). Wynikowy rzut perspektywiczny pokazano na rys. 6.39.

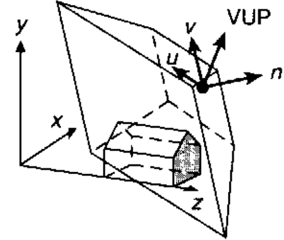
Kroki 1 i 2 są takie same jak dla rzutu równoległego: $R \cdot T(-VRP)$. Krok 3 polega na przesunięciu środka rzutu (COP) do początku układu współrzędnych, tak jak to jest wymagane dla kanonicznej bryły widzenia dla rzutu perspektywicznego. Punkt COP jest określony względem punktu VRP w układzie VRC przez punkt $PRP = (prp_u, prp_v, prp_n)$. Układ VRC został przekształcony we współrzędne świata w wyniku kroków 1 i 2 i specyfikacja dla punktu COP w układzie VRC jest teraz również we współrzędnych świata. Dlatego przesunięcie dla kroku 3, to po prostu $T(-PRP)$.



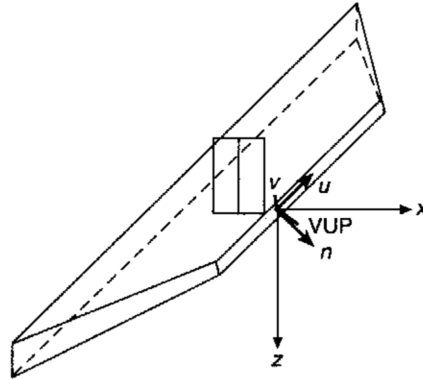
Rys. 6.40. Przekrój bryły widzenia po trzech krokach przekształcania

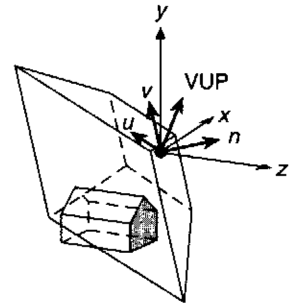
a)



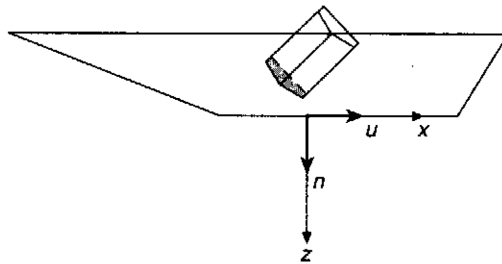


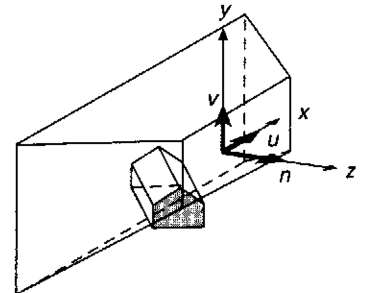
b)

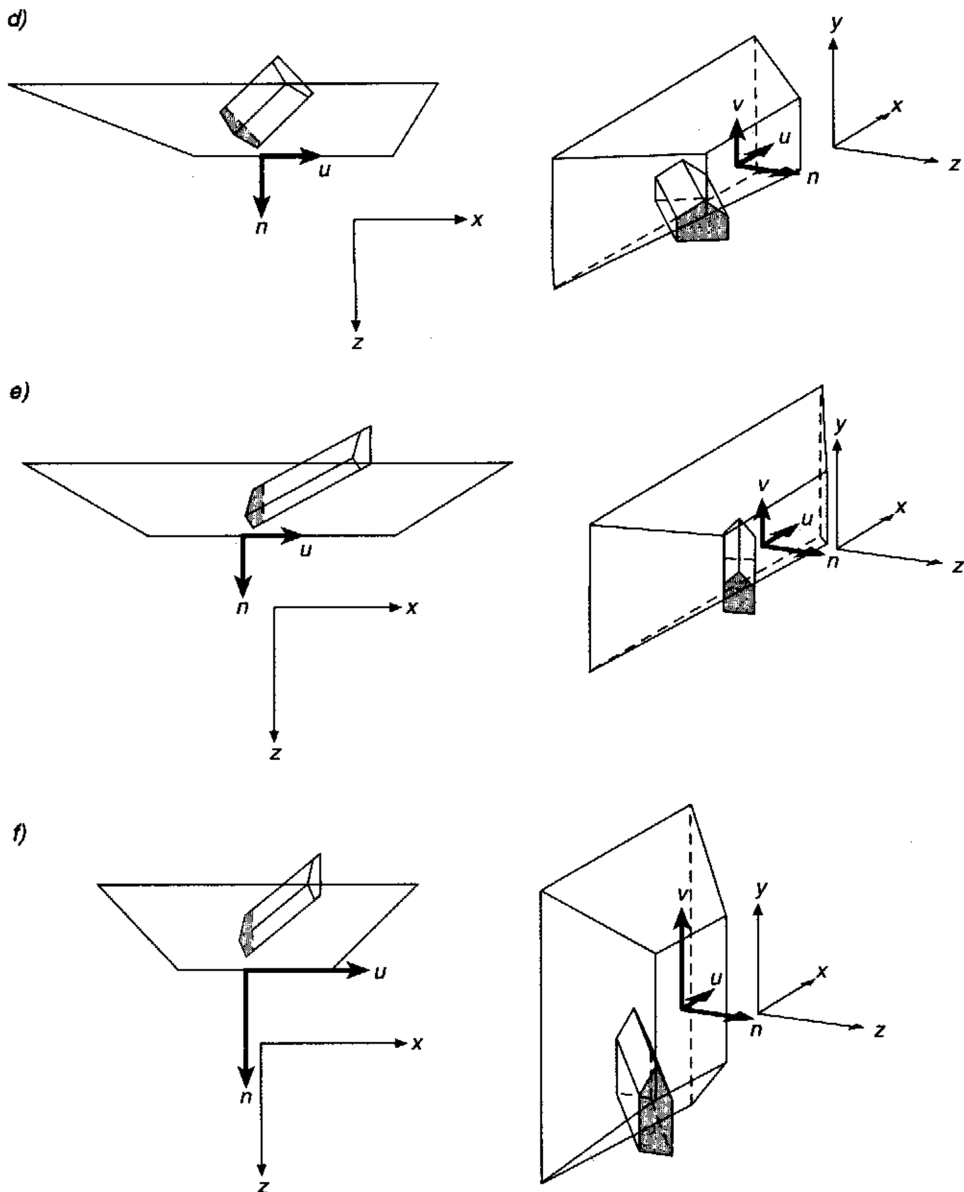




c)







Rys. 6.41. Kolejne kroki procesu rzutowania perspektywicznego. W każdym przypadku są pokazane rzuty równoległe górny i z punktu poza osią: a) początkowa sytuacja; b) VRP jest przesunięte do początku układu współrzędnych; c) układ współrzędnych (*u*, *v*, *n*) został tak obrócony, żeby pokrył się z układem (*x*, *y*, *z*); d) środek rzutowania (COP) został przesunięty do początku układu współrzędnych; e) bryła widzenia została tak pochylona, że kierunek rzutowania (DOP) jest równoległy do osi *z*; f) bryła widzenia została przeskalowana do postaci kanonicznej bryły widzenia dla rzutu perspektywicznego. Parametry rzutowania są następujące: VRP = (1,0, 1,275, 2,6), VPN = (1,0, 0,253, 1,0), VUP = (0,414, 1,0, 0,253), PRP = (1,6, 0,0, 1,075), Okno = (−1,325, 2,25, −0,575, 0,575), $F = 0$, $B = -1{,}2$. (Rysunki zostały wykonane za pomocą programu napisanego przez L. Lu z George Washington University.)

W celu obliczenia pochylenia dla kroku 4 popatrzmy na rys. 6.40, na którym pokazano boczny widok bryły widzenia po przekształceniach w krokach 1 do 3. Zauważmy, że linia środkowa bryły widzenia, która przechodzi przez początek układu współrzędnych i środek okna, nie pokrywa się z osią $-z$. Celem operacji pochylenia jest przekształcenie linii środkowej w oś $-z$. Linia środkowa bryły widzenia idzie z punktu PRP (który jest teraz w początku układu współrzędnych) do środka okna CW. Dlatego jest taka sama jak kierunek rzutowania dla rzutu równoległego – to znaczy CW – PRP. Wobec tego macierzą pochylenia jest SH_{par}, ta sama macierz co dla rzutu równoległego! Inny sposób rozumowania w tym przypadku polega na tym, że przesunięcie o $-$PRP w kroku 3, które przesuwa środek rzutowania do początku układu współrzędnych, przesuwa również CW o $-$PRP; tak więc po kroku 3 linia środkowa bryły widzenia przechodzi przez początek układu współrzędnych i CW – PRP.

Po wykonaniu operacji pochylenia okno (a tym samym bryła widzenia) jest symetryczne względem osi z. Krawędzie okna na rzutni są określone w następujący sposób:

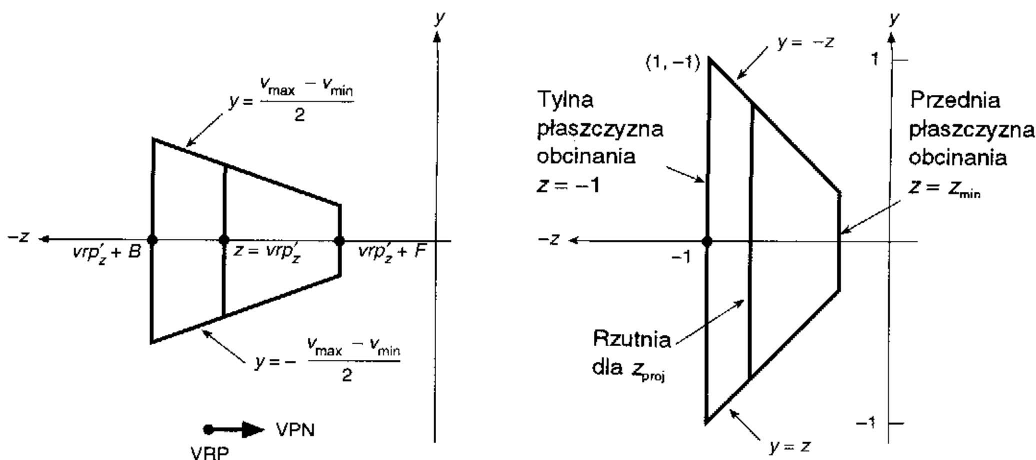
$$-\frac{u_{max}-u_{min}}{2} \leqslant x \leqslant \frac{u_{max}-u_{min}}{2}$$
$$-\frac{v_{max}-v_{min}}{2} \leqslant y \leqslant \frac{v_{max}-v_{min}}{2} \qquad (6.27)$$

Punkt VRP, który przed krokiem 3 był w początku układu współrzędnych, został teraz przesunięty w kroku 3 i pochylony w kroku 4. Określając punkt VRP′ jako VRP po przekształceniach w krokach 3 i 4 mamy

$$\mathrm{VRP}' = SH_{par} \cdot T(-\mathrm{PRP}) \cdot [0\ 0\ 0\ 1]^T \qquad (6.28)$$

Współrzędna z punktu VRP′ oznaczona jako vrp'_z jest równa $-prp_n$, ponieważ pochylenie SH_{par} (x, y) nie wpływa na współrzędne z.

W ostatnim kroku następuje skalowanie wzdłuż wszystkich trzech osi w celu utworzenia kanonicznej bryły widzenia zdefiniowanej równaniem (6.13) i pokazanej na rys. 6.42. Skalowanie najlepiej jest traktować jako proces składający się z dwóch etapów. W pierwszym skalujemy różnie w kierunkach x i y po to, żeby płaszczyzny ograniczające bryłę widzenia uzyskały nachylenie jednostkowe. Wykonujemy to przez takie skalowanie okna, żeby połowy jego wysokości i szerokości były równe $-vrp'_z$. Współczynniki skalowania x i y są równe odpowiednio $-2 \cdot vrp'_z/(u_{max}-u_{min})$ i $-2 \cdot vrp'_z/(v_{max}-v_{min})$. W drugim etapie skalujemy równomiernie wzdłuż wszystkich trzech osi (żeby zachować na-



Rys. 6.42. Przekrój bryły widzenia przed skalowaniem (a); po skalowaniu (b). W tym przykładzie F i B mają przeciwne znaki i płaszczyzny przednia i tylna są po przeciwnych stronach VRP

chylenia jednostkowe), tak żeby tylna płaszczyzna obcinania $z = vrp'_z + B$ pokryła się z płaszczyzną $z = -1$. Współczynnik skalowania dla tego etapu jest równy $-1/(vrp'_z + B)$. Współczynnik skalowania ma znak ujemny, co w połączeniu z ujemnym znakiem $vrp'_z + B$ daje dodatnią wartość współczynnika skalowania.

Łacząc te dwa etapy otrzymujemy przekształcenie skalowania perspektywicznego

$$S_{per} = S\left(\frac{2vrp'_z}{(u_{max} - u_{min})(vrp'_z + B)}, \frac{2vrp'_z}{(v_{max} - v_{min})(vrp'_z + B)}, \frac{-1}{vrp'_z + B}\right) \tag{6.29}$$

Zastosowanie skalowania do z zmienia pozycje płaszczyzny rzutowania i płaszczyzn obcinania; nowe pozycje są następujące [1]:

$$z_{proj} = -\frac{vrp'_z}{vrp'_z + B}, \quad z_{min} = -\frac{vrp'_z + F}{vrp'_z + B}, \quad z_{max} = -\frac{vrp'_z + B}{vrp'_z + B} = -1 \tag{6.30}$$

[1] z_{min} i z_{max} zostały tak oznaczone ze względu na zależność między wartościami bezwzględnymi, ponieważ algebraicznie z_{min} jest większe niż z_{max}.

W sumie, przekształcenie normalizujące rzutowania, które przekształca bryłę widzenia dla rzutu perspektywicznego w kanoniczną bryłę widzenia dla rzutu perspektywicznego, ma postać

$$N_{per} = S_{per} \cdot SH_{par} \cdot T(-\mathrm{PRP}) \cdot R \cdot T(-\mathrm{VRP}) \tag{6.31}$$

Przypomnijmy, że przekształcenie normalizujące rzutowania, które przekształca bryłę widzenia dla rzutu równoległego w kanoniczną bryłę widzenia dla rzutu równoległego, ma postać

$$N_{par} = S_{par} \cdot T_{par} \cdot SH_{par} \cdot R \cdot T(-\mathrm{VRP}) \tag{6.26}$$

Te przekształcenia zachodzą we współrzędnych jednorodnych. Przy jakich warunkach możemy teraz wrócić do przestrzeni 3D w celu wykonania obcinania? Odpowiedź jest następująca: jeżeli $W > 0$. Ten warunek jest łatwy do zrozumienia. Ujemna wartość W oznacza, że gdy dzielimy przez W, wówczas znaki z i Z będą przeciwne. Punkty z ujemną wartością Z będą miały dodatnie z i mogłyby być wyświetlone nawet, gdyby powinny zostać obcięte.

Kiedy możemy mieć pewność, że jest spełniony warunek $W > 0$? Obroty, przesunięcia, skalowanie i pochylanie (według definicji z rozdz. 5) zastosowane do punktów, odcinków i płaszczyzn zachowują $W > 0$; a faktycznie zachowują $W = 1$. Ponieważ ani N_{per}, ani N_{par} nie wpływają na współrzędne jednorodne przekształcanych punktów, dzielenie przez W zwykle nie będzie potrzebne przy odwrotnym odwzorowaniu do 3D i obcinanie względem odpowiedniej kanonicznej bryły widzenia może być wykonane. Po obcięciu przez kanoniczną bryłę widzenia dla rzutu perspektywicznego trzeba zastosować macierz M_{per} rzutowania perspektywicznego, która wymaga dzielenia.

Można otrzymać $W < 0$, gdy prymitywy wyjściowe zawierają krzywe i powierzchnie, które są reprezentowane jako funkcje we współrzędnych jednorodnych i są wyświetlane jako połączone segmenty odcinków. Jeżeli, na przykład, znak dla funkcji W zmienia się od jednego punktu na krzywej do drugiego, podczas gdy znak X nie zmienia się, to iloraz X/W będzie miał różne znaki dla tych dwóch punktów krzywej. Przykład takiego zachowania można znaleźć w wymiernych krzywych B-sklejanych omawianych w rozdz. 9. Ujemna wartość W może również powstać w wyniku stosowania niektórych innych przekształceń niż omawiane w rozdz. 5, np. „pozorne" cienie [BLIN88]. W następnym punkcie omówiono różne algorytmy obcinania w 3D. Następnie w p. 6.6.4 omówiono sposób obcinania, jeżeli nie można zapewnić spełnienia warunku $W > 0$.

6.6.3. Obcinanie w 3D przez kanoniczną bryłę widzenia

Kanoniczne bryły widzenia są to prostopadłościany $2 \times 2 \times 1$ dla rzutów równoległych i prawidłowe ostrosłupy ścięte dla rzutów perspektywicznych. Algorytmy obcinania Cohena-Sutherlanda i Cyrusa-Becka omówione w rozdz. 3 można bezpośrednio uogólnić na przypadek 3D.

W rozszerzeniu algorytmu 2D Cohena-Sutherlanda dla kanonicznej bryły widzenia dla rzutu równoległego wykorzystuje się 6-bitowy kod; bit przyjmuje wartość 1, gdy jest spełniony odpowiedni warunek:

bit 1 –	punkt jest powyżej bryły widzenia	$y > 1$
bit 2 –	punkt jest poniżej bryły widzenia	$y < -1$
bit 3 –	punkt jest z prawej strony bryły widzenia	$x > 1$
bit 4 –	punkt jest z lewej strony bryły widzenia	$x < -1$
bit 5 –	punkt jest z tyłu za bryłą widzenia	$z < -1$
bit 6 –	punkt jest przed bryłą widzenia	$z > 0$

Podobnie jak w przypadku 2D, odcinek jest bezpośrednio akceptowany, jeżeli jego oba końce mają kody zawierające same zera, i jest bezpośrednio odrzucany, jeżeli iloczyn logiczny kodów brany bit po bicie nie zawiera samych zer. W przeciwnym przypadku rozpoczyna się proces dzielenia odcinka. Może być konieczne obliczenie nawet sześciu przecięć, po jednym dla każdej ściany bryły widzenia.

Przy obliczaniu przecięć wykorzystuje się parametryczną reprezentację odcinka między punktami $P_0(x_0, y_0, z_0)$ i $P_1(x_1, y_1, z_1)$:

$$x = x_0 + t(x_1 - x_0) \tag{6.32}$$

$$y = y_0 + t(y_1 - y_0) \tag{6.33}$$

$$z = z_0 + t(z_1 - z_0) \qquad 0 \leqslant t \leqslant 1 \tag{6.34}$$

Gdy t zmienia się od 0 do 1, wówczas te trzy równania dają współrzędne wszystkich punktów na odcinku od P_0 do P_1.

W celu obliczenia przecięcia odcinka z płaszczyzną $y = 1$ bryły widzenia, zastępujemy zmienną y w równaniu (6.33) przez 1 i rozwiązujemy ze względu na t; otrzymujemy, że $t = (1 - y_0)/(y_1 - y_0)$. Jeżeli t jest poza przedziałem [0, 1], to przecięcie jest na prostej przechodzącej przez punkty P_0 i P_1, ale nie jest na odcinku prostej między P_0 i P_1 i dlatego nie jest interesujące. Jeżeli t jest w przedziale [0, 1], to tę wartość podstawiamy do równań dla x i z i znajdujemy współrzędne przecięcia:

$$x = x_0 + \frac{(1 - y_0)(x_1 - x_0)}{y_1 - y_0}, \quad z = z_0 + \frac{(1 + y_0)(z_1 - z_0)}{y_1 - y_0} \tag{6.35}$$

Kody są wykorzystywane w algorytmie po to, żeby wyeliminować test dla t w przedziale [0, 1].

W przypadku obcinania względem kanonicznej bryły widzenia dla rzutu perspektywicznego bity kodu są ustawiane na wartość 1 zgodnie z następującymi regułami:

bit 1	–	punkt jest powyżej bryły widzenia	$y > -z$
bit 2	–	punkt jest poniżej bryły widzenia	$y < z$
bit 3	–	punkt jest z prawej strony bryły widzenia	$x > -z$
bit 4	–	punkt jest z lewej strony bryły widzenia	$x < z$
bit 5	–	punkt jest z tyłu za bryłą widzenia	$z < -1$
bit 6	–	punkt jest przed bryłą widzenia	$z > z_{min}$

Obliczanie przecięć odcinków z pochylonymi płaszczyznami jest proste. Na płaszczyźnie $y = z$, dla której równanie (6.33) musi być równe równaniu (6.34), $y_0 + t(y_1 - y_0) = z_0 + t(z_1 - z_0)$. Wtedy

$$t = \frac{z_0 - y_0}{(y_1 - y_0) - (z_1 - z_0)} \tag{6.36}$$

Po podstawieniu t do równań (6.32) i (6.33) otrzymujemy:

$$x = x_0 + \frac{(x_1 - x_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)}, \quad y = y_0 + \frac{(y_1 - y_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)} \tag{6.37}$$

Wiemy, że $z = y$. Powód wybrania tej kanonicznej bryły widzenia jest teraz jasny: dzięki jednostkowym nachyleniom płaszczyzn obliczanie przecięć jest prostsze niż w przypadku dowolnych nachyleń.

Znane są inne algorytmy obcinania [CYRU78; LIAN84], w których jest wykorzystywany parametryczny opis odcinka; mogą być one efektywniejsze niż prosty algorytm Cohena-Sutherlanda. Porównaj rozdziały 6 i 19 pracy [FOLE90].

6.6.4. Obcinanie we współrzędnych jednorodnych

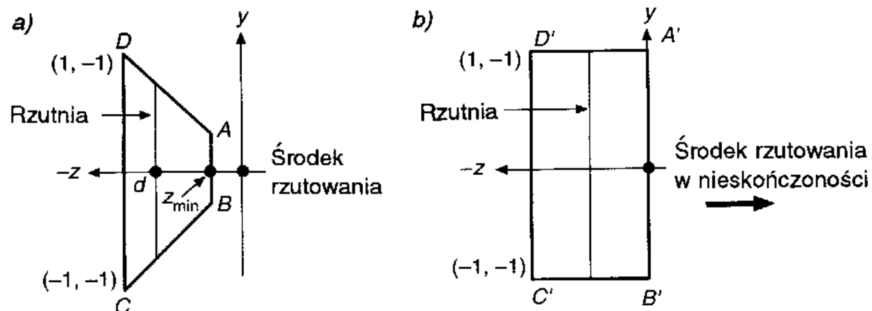
Są dwa powody obcinania we współrzędnych jednorodnych. Pierwszy jest związany z efektywnością; można przekształcić kanoniczną bryłę widzenia dla rzutu perspektywicznego w kanoniczną bryłę widzenia dla rzutu równoległego i zawsze można użyć jednej procedury obcinania zoptymalizowanej dla kanonicznej bryły widzenia dla rzutu równoległego. Ze względu jednak na zapewnienie poprawności wyniku, obcina-

nie musi być wykonywane we współrzędnych jednorodnych. Ta jednolita procedura obcinania jest typowo dostępna w sprzętowych realizacjach operacji rzutowania. Drugim powodem jest to, że punkty, które mogą się pojawić jako wynik nietypowych przekształceń jednorodnych i w przypadku stosowania wymiernych parametrycznych funkcji sklejanych (rozdz. 9), mogą mieć ujemną wartość W i mogą być obcięte prawidłowo we współrzędnych jednorodnych, ale nie w 3D.

W odniesieniu do obcinania można pokazać, że przekształcenie z kanonicznej bryły widzenia dla rzutu perspektywicznego w kanoniczną bryłę widzenia dla rzutu równoległego ma postać

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(1+z_{min}) & -z_{min}/(1+z_{min}) \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad z_{min} \neq -1 \qquad (6.38)$$

Z równania (6.30) mamy, że $z_{min} = -(vrp'_z + F)/(vrp'_z + B)$, i z równania (6.28), że $VRP' = SH_{par} \cdot T(-PRP) \cdot [0\ 0\ 0\ 1]^T$. Na rysunku 6.43 pokazano wyniki zastosowania macierzy M do kanonicznej bryły widzenia dla rzutu perspektywicznego.



Rys. 6.43. Boczne rzuty znormalizowanej bryły widzenia dla rzutu perspektywicznego przed (a) i po (b) zastosowaniu macierzy M

Macierz M można połączyć z normalizującym przekształceniem dla rzutu perspektywicznego N_{per}:

$$N'_{per} = M \cdot N_{per} = M \cdot S_{per} \cdot SH_{par} \cdot T(-PRP) \cdot R \cdot T(-VRP) \qquad (6.39)$$

Korzystając dla rzutów perspektywicznych z N'_{per} zamiast z N_{per} i z N_{par} dla rzutów równoległych, możemy obciąć względem kanonicznej bryły widzenia dla rzutu równoległego, a nie względem kanonicznej bryły widzenia dla rzutu perspektywicznego.

Bryła widzenia dla rzutu równoległego 3D jest określona przez $-1 \leqslant x \leqslant 1$, $-1 \leqslant y \leqslant 1$, $-1 \leqslant z \leqslant 0$. Odpowiednie nierówności we współrzędnych jednorodnych znajdujemy zastępując x przez X/W, y przez Y/W i z przez Z/W, co daje:

$$-1 \leqslant \frac{X}{W} \leqslant 1, \quad -1 \leqslant \frac{Y}{W} \leqslant 1, \quad -1 \leqslant \frac{Z}{W} \leqslant 0 \tag{6.40}$$

Odpowiednie równania płaszczyzn są następujące:

$$X = -W, \quad X = W, \quad Y = -W, \quad Z = -W, \quad Z = 0 \tag{6.41}$$

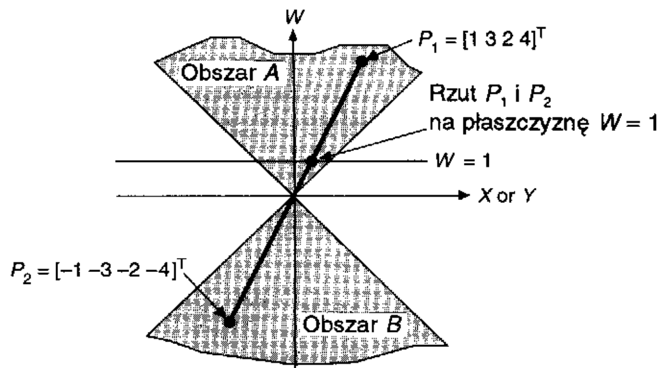
W celu wyjaśnienia, jak należy korzystać z tych ograniczeń i płaszczyzn, musimy niezależnie rozpatrzyć przypadki dla $W > 0$ i $W < 0$. W pierwszym przypadku możemy pomnożyć nierówności z równania (6.40) przez W bez zmiany kierunku nierówności. W drugim przypadku mnożenie zmienia te kierunki. Wynik można zapisać jako:

$$W > 0: \quad -W \leqslant X \leqslant W, \quad -W \leqslant Y \leqslant W, \quad -W \leqslant Z \leqslant 0 \tag{6.42}$$

$$W < 0: \quad -W \geqslant X \geqslant W, \quad -W \geqslant Y \geqslant W, \quad -W \geqslant Z \geqslant 0 \tag{6.43}$$

W przypadku obcinania zwykłych odcinków i punktów trzeba wykorzystać tylko obszary określone przez zależności (6.42), ponieważ przed zastosowaniem macierzy M wszystkie widoczne punkty mają $W > 0$ (normalnie $W = 1$).

Jednak, jak zobaczymy w rozdz. 9, czasami warto jest reprezentować punkty bezpośrednio we współrzędnych jednorodnych z dowolnymi współrzędnymi W. Możemy więc mieć $W < 0$, co oznacza, że obcinanie może następować przez obszary określone równaniami (6.42) i (6.43). Na rysunku 6.44 pokazano to w postaci obszarów A i B; widać również, dlaczego trzeba użyć obu obszarów.

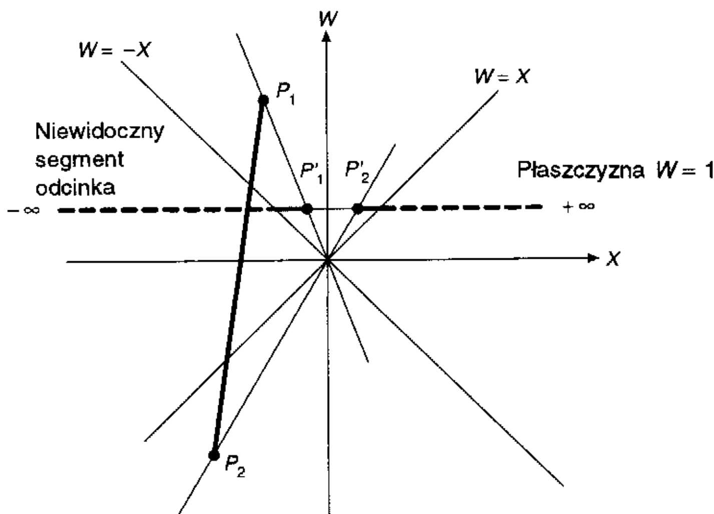


Rys. 6.44. Oba punkty P_1 i P_2 odwzorowują się na ten sam punkt na płaszczyźnie $W = 1$, tak jak wszystkie inne punkty na odcinku przechodzącym przez początek układu współrzędnych i te dwa punkty. Obcinanie we współrzędnych jednorodnych względem obszaru A niepoprawnie odrzuci punkt P_2

Punkt $P_1 = [1\ 3\ 2\ 4\ ]^T$ w obszarze A przekształca się w punkt 3D (1/4, 3/4, 2/4), który jest w kanonicznej bryle widzenia określony przez $-1 \leqslant x \leqslant 1$, $-1 \leqslant y \leqslant 1$, $-1 \leqslant z \leqslant 0$. Punkt $P_2 = -P_1 = [-1\ -3\ -2\ -4]^T$, który nie jest w obszarze A, ale jest w obszarze B, przekształca się w ten sam punkt 3D co P_1, a mianowicie (1/4, 3/4, 2/4). Gdyby obcinanie następowało tylko w obszarze A, wówczas P_2 byłby odrzucony nieprawidłowo. Taka możliwość powstaje, ponieważ współrzędne jednorodne punktów P_1 i P_2 różnią się o stały czynnik (-1); wiemy, że takie punkty jednorodne odpowiadają temu samemu punktowi 3D (na płaszczyźnie $W = 1$ w przestrzeni jednorodnej).

Są dwa sposoby rozwiązania tego problemu dla punktów w obszarze B. Jeden sposób polega na obcięciu wszystkich punktów dwukrotnie, po razie w każdym obszarze. Ale dwukrotne obcinanie jest kosztowne. Lepszym rozwiązaniem jest negowanie punktów takich jak P_2 z ujemną wartością W i dopiero potem obcinanie. Podobnie możemy obciąć poprawnie odcinek, którego dwa końce są w obszarze B z rys. 6.44 mnożąc oba punkty końcowe przez -1 po to, żeby umieścić punkty w obszarze A.

Inny problem powstaje z odcinkami takimi jak $P_1 P_2$ z rys. 6.45, których punkty końcowe mają przeciwne wartości W. Rzut odcinka na płaszczyznę $W = 1$ ma dwa segmenty: jeden, który rozciąga się do $+\infty$, i drugi, który rozciąga się do $-\infty$. Teraz rozwiązanie polega na podwójnym obcięciu, po razie względem każdego obszaru – możliwe jest,



Rys. 6.45. Rzut odcinka P_1P_2 tworzą dwa segmenty, jeden od P'_2 do $+\infty$, drugi od P'_1 do $-\infty$ (pokazane jako ciągłe linie w obszarze obcinania i jako przerywane linie poza obszarem obcinania). Odcinek musi być obcięty dwukrotnie – raz dla każdego obszaru

że każde obcięcie da w efekcie widoczny segment odcinka. Prosty sposób wykonania tego polega na obcięciu odcinka przez obszar A, zanegowaniu obu końców odcinka i ponownym obcięciu przez obszar A. Takie podejście zachowuje jeden z oryginalnych celów obcinania we współrzędnych jednorodnych: korzystanie z jednego obszaru obcinania. Czytelników zainteresowanych dalszymi szczegółami odsyłamy do pracy [BLIN78a].

Jeżeli dane jest równanie (6.41), to do obcinania można użyć algorytmu Cohena-Sutherlanda albo Cyrusa-Becka. Kod dla metody Cyrusa-Becka jest podany w pracy [LIAN84]. Jedyna różnica polega na tym, że obcinanie jest w 4D, a nie w 3D.

6.6.5. Odwzorowanie na pole wizualizacji

Prymitywy wyjściowe są obcinane w znormalizowanym układzie współrzędnych rzutowania, który jest również określany jako układ współrzędnych ekranowych 3D. Założymy dla celów tej dyskusji, że do obcinania została wykorzystana kanoniczna bryła widzenia dla rzutu równoległego. (Jeżeli to założenie nie jest poprawne, to rzut perspektywiczny M przekształca bryłę widzenia dla rzutu perspektywicznego w bryłę widzenia rzutu równoległego.) Dlatego współrzędne wszystkich prymitywów wyjściowych, które pozostają, są w bryle widzenia $-1 \leqslant x \leqslant 1$, $-1 \leqslant y \leqslant 1$, $-1 \leqslant z \leqslant 0$.

Programista korzystający z PHIGS określa pole wizualizacji 3D, na które ma być odwzorowana bryła widzenia. Pole wizualizacji jest zawarte w sześcianie jednostkowym $0 \leqslant x \leqslant 1$, $0 \leqslant y \leqslant 1$, $0 \leqslant z \leqslant 1$. Przednia ściana $z = 1$ sześcianu jednostkowego jest odwzorowywana na największy kwadrat, który może być wpisany w ekran. Zakładamy, że dolny lewy róg kwadratu jest w (0, 0). Na przykład na monitorze o rozdzielczości poziomej 1024 i pionowej 800 kwadrat składa się z pikseli P w punktach (P_x, P_y), przy czym $0 \leqslant P_x \leqslant 799$, $0 \leqslant P_y \leqslant 799$. Punkty z sześcianu jednostkowego wyświetlamy odrzucając współrzędną z. Dlatego punkt (0,5, 0,75, 0,46) powinien być wyświetlony w punkcie o współrzędnych urządzenia (400, 599). W przypadku określania widocznej powierzchni (rozdz. 13) współrzędna z każdego prymitywu wyjściowego jest używana do określenia, które prymitywy są widoczne, a które są zasłonięte przez prymitywy o większej wartości z.

Dla danego pola wizualizacji 3D w sześcianie jednostkowym, określonym równaniami $x_{v\ min} \leqslant x \leqslant x_{v\ max}$ itd., odwzorowanie z kanonicznej bryły widzenia dla rzutu równoległego może być traktowane jako proces składający się z trzech kroków. W pierwszym kroku kanoniczna bryła widzenia dla rzutu równoległego jest tak przesuwana, żeby jej

róg $(-1, -1, -1)$ znalazł się w początku układu współrzędnych. Wykorzystuje się tu przesunięcie $T(1, 1, 1)$. Następnie przesunięta bryła widzenia jest skalowana do wielkości pola widzenia 3D, ze współczynnikiem skalowania

$$S\left(\frac{x_{v\ max} - x_{v\ min}}{2}, \frac{y_{v\ max} - y_{v\ min}}{2}, \frac{z_{v\ max} - z_{v\ min}}{1}\right)$$

Wreszcie poprawnie przeskalowana bryła widzenia jest przesuwana do lewego dolnego rogu pola wizualizacji; wykorzystuje się tu przekształcenie $T(v_{v\ min}, y_{v\ min}, z_{v\ min})$. Stąd złożone przekształcenie kanonicznej bryły widzenia w pole widzenia 3D ma postać

$$M_{VV3DV} = T(x_{v\ min}, y_{v\ min}, z_{v\ min}) \cdot S\left(\frac{x_{v\ max} - x_{v\ min}}{2}, \frac{y_{v\ max} - y_{v\ min}}{2}, \frac{z_{v\ max} - z_{v\ min}}{1}\right) \cdot T(1, 1, 1) \quad (6.44)$$

Zauważmy, że przekształcenie to jest podobne, ale nie takie samo jak przekształcenie M_{WV} okno-pole wizualizacji, wyprowadzone w p. 5.5.

6.6.6. Podsumowanie problemów realizacyjnych

Są dwie powszechnie używane realizacje całego ciągu przekształceń. Pierwszą pokazano na rysunku 6.34 i omówiono w p. od 6.1 do 6.3. Jest ona używana wówczas, gdy prymitywy są zdefiniowane w 3D i przekształcenia stosowane do prymitywów wyjściowych nigdy nie tworzą ujemnych wartości W. W procedurze tej wyróżnia się następujące kroki:

1. Rozszerzenie współrzędnych 3D do współrzędnych jednorodnych.
2. Zastosowanie przekształcenia normalizującego N_{par} albo N_{per}.
3. Podzielenie przez W w celu odwzorowania z powrotem do 3D (czasami wiadomo, że $W = 1$ i dzielenie nie jest potrzebne).
4. Obcięcie w 3D przez kanoniczną bryłę widzenia dla rzutu równoległego albo dla rzutu perspektywicznego, zależnie od sytuacji.
5. Rozszerzenie współrzędnych 3D do współrzędnych jednorodnych.
6. Wykonanie rzutu równoległego za pomocą M_{ort}, równanie (6.11), albo wykonanie rzutu perspektywicznego za pomocą M_{per}, równanie (6.3) z $d = -1$.
7. Przesunięcie i skalowanie do współrzędnych urządzenia z wykorzystaniem równania (6.44).

8. Podzielenie przez W w celu przejścia ze współrzędnych jednorodnych na współrzędne 2D; dzielenie wpływa na rzut perspektywiczny.

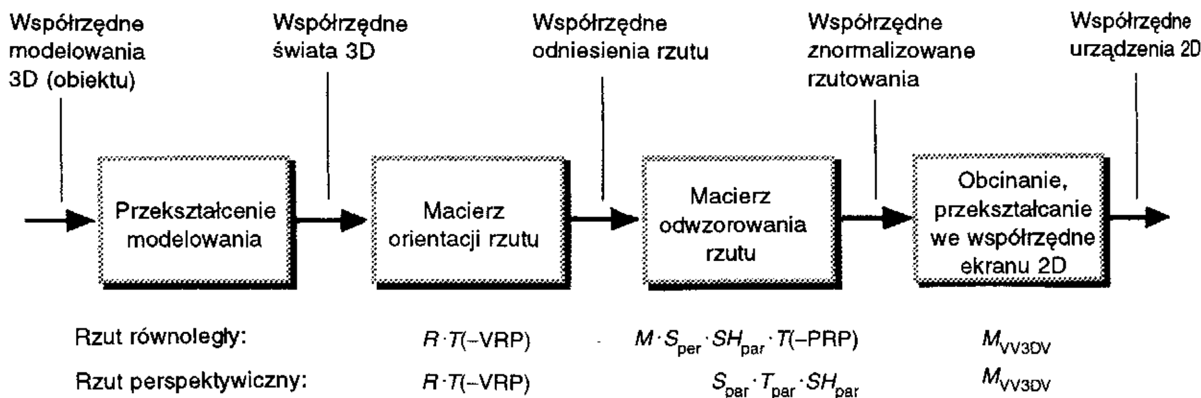
Kroki 6 i 7 są wykonywane jako jedno mnożenie przez macierz i odpowiada etapom 3 i 4 na rys. 6.34.

Drugi sposób realizacji operacji rzutowania jest potrzebny wówczas, gdy prymitywy wyjściowe są zdefiniowane we współrzędnych jednorodnych i mogą mieć $W < 0$, gdy przekształcenia zastosowane do prymitywów wyjściowych mogą tworzyć ujemne W albo gdy jest stosowany algorytm jednego obcinania. Jak pokazano w p. 6.6.4, występują tu następujące kroki:

1. Rozszerzenie współrzędnych 3D do współrzędnych jednorodnych.
2. Zastosowanie przekształcenia normalizującego N_{par} albo N'_{per} (które zawiera M, równanie (6.38)).
3. Jeżeli $W > 0$, obcięcie we współrzędnych jednorodnych względem bryły zdefiniowanej równaniami (6.42) i (6.43).
4. Przesunięcie i skalowanie do współrzędnych urządzenia za pomocą równania (6.44).
5. Podzielenie przez W w celu przejścia od współrzędnych jednorodnych do współrzędnych 2D; dzielenie wpływa na rzut perspektywiczny.

6.7. Układy współrzędnych

W rozdziałach 5 i 6 korzystano z kilku różnych układów współrzędnych. W tym punkcie podsumujemy informacje o tych układach i omówimy zależności między nimi. Podano również synonimy używane w różnych materiałach referencyjnych i pakietach graficznych. Na rysunku 6.46



Rys. 6.46. Układy współrzędnych i zależności między nimi. Macierze poniżej każdego stopnia wpływają na przekształcenia wykonywane w tym stopniu dla rzutów perspektywicznych i równoległych

pokazano kolejność używania układów współrzędnych, przy czym wykorzystano określenia stosowane w tej książce; w konkretnym pakiecie graficznym w rzeczywistości tylko niektóre układy współrzędnych rzeczywiście są wykorzystywane. Wybraliśmy nazwy różnych układów współrzędnych najczęściej stosowane; stąd niektóre z tych nazw nie są logicznie spójne z innymi. Zauważmy, że określenie przestrzeń jest czasami używane jako synonim układu.

Zaczynamy od układu, który na rys. 6.46 jest najdalej z lewej strony od urządzenia wyjściowego; obiekty są tu zdefiniowane w *układzie współrzędnych obiektu.* W standardzie PHIGS jest to układ *współrzędnych modelowania*; często jest również używane określenie *układ współrzędnych lokalnych.* Jak zobaczymy dalej w rozdz. 7, często występuje hierarchia układów współrzędnych modelowania.

Obiekty są przekształcane w układ współrzędnych świata – układ, w którym scena albo kompletny obiekt są reprezentowane w komputerze dzięki *przekształceniom modelowania.* Ten układ jest czasami określany jako *układ współrzędnych problemu* albo *układ współrzędnych zastosowania.*

W systemie PHIGS jest używany bazowy *układ współrzędnych rzutowania*, w którym jest definiowana bryła widzenia. Jest on również określany jako układ (u, v, n) albo układ (u, v, VPN). W systemie Core [GSPC79] był używany podobny, ale nie nazwany układ lewoskrętny. Układ lewoskrętny jest używany, ponieważ, jeżeli oko albo kamera znajdujące się w początku układu współrzędnych są skierowane w kierunku osi $+z$, to rosnące wartości z oddalają się od oka, oś x jest skierowana w prawo, a oś y ku górze.

W innych pakietach, np. RenderMan firmy Pixar [PIXA88], są formułowane ograniczenia na bazowy układ współrzędnych rzutowania, określające, że początek układu musi być w środku rzutowania i że rzutnia ma być prostopadła do osi z. Określamy to jako *układ współrzędnych oka*; w systemie RenderMan i w kilku innych systemach jest używane określenie *układ współrzędnych kamery.* W punkcie 6.6 pierwsze trzy kroki przekształcenia normalizującego dla rzutu perspektywicznego dokonywały konwersji z układu współrzędnych świata do układu współrzędnych oka. Układ współrzędnych oka jest czasami lewoskrętny.

Ze współrzędnych oka przechodzimy następnie do *układu współrzędnych znormalizowanego rzutowania* albo *współrzędnych ekranu 3D* – układu współrzędnych kanonicznej bryły widzenia dla rzutu równoległego (i kanonicznej bryły widzenia dla rzutu perspektywicznego po przekształceniu perspektywicznym). W systemie Core ten układ był określany jako *układ współrzędnych 3D znormalizowanego urządzenia.* Czasami układ jest określany jako *układ współrzędnych 3D urządzenia logicznego.* Określenie znormalizowany ogólnie oznacza, że wszystkie

wartości współrzędnych są albo w przedziale [0, 1], albo [−1, 1], natomiast określenie logiczny oznacza, że wartości współrzędnych są w pewnym innym wcześniej podanym zakresie, na przykład [0, 1023], który typowo odpowiada niektórym powszechnie dostępnym układom współrzędnych urządzenia.

Rzutowanie z 3D na 2D tworzy to, co nazywamy *układem współrzędnych 2D urządzenia*; są również stosowane nazwy *układ współrzędnych znormalizowanych urządzenia*, *układ współrzędnych obrazu* [SUTH74a] albo *układ współrzędnych ekranu* (RenderMan). Występują również takie terminy jak *współrzędne ekranu*, *współrzędne urządzenia*, *współrzędne urządzenia 2D*, *współrzędne urządzenia fizycznego* (w przeciwieństwie do wcześniej wymienionych współrzędnych urządzenia logicznego). RenderMan używa terminu *współrzędne rastra*.

Niestety nie ma jednolitości, jeśli chodzi o znaczenie przypisywane różnym nazwom. Na przykład określenie *układ współrzędnych ekranu* jest używane przez różnych autorów w odniesieniu do ostatnich trzech omówionych układów, obejmując zarówno układy współrzędnych 2D i 3D, jak i układy współrzędnych logiczne i fizyczne.

Zadania

6.1. Napisz program, który akceptuje parametry rzutowania, oblicza albo N_{par}, albo N_{per} i wyświetla dom o współrzędnych zdefiniowanych na rys. 6.18.

6.2. Zrealizuj algorytm obcinania 3D dla rzutów równoległego i perspektywicznego.

6.3. Pokaż, że dla rzutu perspektywicznego z $F = -\infty$ i $B = +\infty$ wynik obcinania w 3D i potem rzutowania do 2D jest taki sam jak wynik rzutowania do 2D i potem obcinania w 2D.

6.4. Pokaż, że jeżeli wszystkie obiekty są przed środkiem rzutowania i jeżeli $F = -\infty$ i $B = +\infty$, to wynik obcinania w 3D względem kanonicznej bryły widzenia dla rzutu perspektywicznego, po którym następuje rzut perspektywiczny, jest taki sam jak w przypadku, gdy najpierw wykona się rzut perspektywiczny do 2D, a potem wykona się obcinanie w 2D.

6.5. Sprawdź, że S_{per} (p. 6.6.2) przekształca bryłę widzenia z rys. 6.42a w bryłę z rys. 6.42b.

6.6. Napisz kod dla obcinania 3D względem sześcianu jednostkowego. Uogólnij kod tak, żeby możliwe było obcinanie przez dowolną bryłę prostokątną o ścianach prostopadłych do głównych osi. Czy ogólny kod jest bardziej czy mniej efektywny od kodu dla przypadku sześcianu jednostkowego? Wyjaśnij odpowiedź.

6.7. Napisz kod dla obcinania 3D względem kanonicznej bryły widzenia dla rzutu perspektywicznego. Potem uogólnij na bryłę widzenia zdefiniowaną przez:

$$-az_v \leqslant x_v \leqslant bz_v, \quad -cz_v \leqslant y_v \leqslant dz_v, \quad z_{min} \leqslant z_v \leqslant z_{max}$$

Te zależności reprezentują ogólną postać bryły widzenia po krokach od 1 do 4 perspektywicznego przekształcenia normalizującego. Który przypadek jest efektywniejszy? Uzasadnij odpowiedź.

6.8. Napisz kod dla obcinania 3D względem ogólnej bryły widzenia w postaci sześcianu o ścianach zdefiniowanych przez

$$A_i x + B_i y + C_i z + D_i = 0 \quad 1 \leqslant i \leqslant 6$$

Porównaj potrzebne nakłady obliczeniowe z nakładami potrzebnymi w następujących przypadkach:

a. Obcinanie względem dowolnej kanonicznej bryły widzenia.

b. Zastosowanie N_{par} i potem obcinanie względem sześcianu jednostkowego.

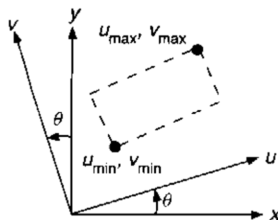
6.9. Weźmy odcinek określony we współrzędnych świata przez punkty $P_1(6, 10, 3)$ i $P_2(-3, -5, 2)$ oraz otwarty ostrosłup widzenia w obszarze $-z \leqslant x \leqslant z$, $-z \leqslant y \leqslant z$, który jest ograniczony przez płaszczyzny $z = +x$, $z = -x$, $z = +y$, $z = -y$. Płaszczyzna rzutowania jest w $z = 1$.

a. Obetnij odcinek w 3D (korzystając z równań parametrycznych odcinka), potem zrzutuj go na płaszczyznę rzutowania. Jakie są obcięte końce odcinka na płaszczyźnie?

b. Zrzutuj odcinek na płaszczyznę, potem obetnij odcinki korzystając z obliczeń 2D. Jakie są obcięte końce na płaszczyźnie?
(*Uwaga*: Jeżeli odpowiedzi na części a) i b) nie są identyczne, to należy spróbować jeszcze raz!)

6.10. Pokaż, co się stanie, jeżeli obiekt leżący za środkiem rzutowania jest rzutowany za pomocą M_{per} i potem obcinany. Odpowiedź powinna pokazać, dlaczego w ogólnym przypadku nie możemy rzutować i potem obcinać.

6.11. Rozważ operację rzutowania 2D z obróconym oknem. Znajdź przekształcenie normalizujące dla przekształcenia okna w kwadrat jednostkowy. Okno jest określone przez u_{min}, v_{min}, u_{max}, v_{max} w układzie współrzędnych VRC (rys. 6.47). Pokaż, że to przekształcenie jest takie samo



Rys. 6.47. Obrócone okno

jak przekształcenie dla ogólnego przypadku 3D N_{par}, gdy rzutnią jest płaszczyzna (x, y) i wektor VUP ma jako składową x równą $-\sin\theta$, i składową y równą $\cos\theta$ (to znaczy rzut równoległy wektora VUP na rzutnię pokrywa się z osią v).

6.12. Jaki jest efekt zastosowania M_{per} do punktów, których współrzędna z jest mniejsza niż 0?

6.13. Zaprojektuj i zrealizuj zbiór podprogramów do generowania macierzy przekształceń 4×4 dla dowolnej sekwencji podstawowych przekształceń R, S i T.

6.14. Narysuj drzewo decyzyjne do określania typu rzutu użytego przy tworzeniu obrazu. Wykorzystaj to drzewo do rysunków z tego rozdziału, które są rzutami z 3D.

6.15. Przyjęto, że kanoniczna bryła widzenia dla rzutu równoległego jest prostopadłościanem $2 \times 2 \times 1$. Załóż, że zamiast tego użyto sześcianu jednostkowego w dodatnim oktancie z jednym wierzchołkiem w początku układu współrzędnych.

a. Znajdź normalizację N'_{par} dla tej bryły widzenia.

b. Znajdź odpowiednią bryłę widzenia we współrzędnych jednorodnych.

6.16. Podaj parametry rzutowania dla rzutów domu: górnego, przedniego i bocznego z rys. 6.18 z punktem VRP w środku okna. Czy punkt PRP musi być różny dla każdego rzutu? Wyjaśnij odpowiedź.

6.17. Para stereo to dwa rzuty tej samej sceny wykonane z nieco różnych punktów obserwacji (rzutowania), ale z tym samym punktem VRP. Niech d będzie odległością między punktami obserwacji. Jeżeli punkty obserwacji utożsamić z oczami, to d jest odległością między oczami. Niech P będzie punktem leżącym po środku między oczami. Dla danych P, d, VRP, VPN i VUP wyprowadź wyrażenia dla dwóch punktów obserwacji.

7. Hierarchia obiektów i SPHIGS

Pakiet graficzny jest elementem pośredniczącym między programem użytkowym a sprzętem. Prymitywy wyjściowe i urządzenia interakcyjne, które są obsługiwane przez pakiet graficzny, mogą być zarówno bardzo proste, jak i bardzo złożone. W rozdziale 2 opisaliśmy dość prosty pakiet niskiego poziomu SRPG i wskazaliśmy na niektóre z jego ograniczeń. W tym rozdziale opiszemy pakiet bazujący na znacznie bogatszym i bardziej złożonym standardzie *PHIGS* (Programmer's Hierarchical Interactive Graphics System)[1]. *Standardowy pakiet graficzny*, np. PHIGS albo GKS (Graphical Kernel System), jest to implementacja specyfikacji przyjęta jako standard przez oficjalną narodową albo międzynarodową organizację standaryzacyjną; GKS i PHIGS zostały przyjęte przez ANSI (American National Standards Institute) i ISO (International Standards Organization). Głównym celem takich standardów jest promowanie przenośności programów użytkowych. Nieoficjalne standardy przemysłowe, omawiane w p. 7.11.6, okazały się równie ważne dla grafiki interakcyjnej. Należą tu OpenGL firmy Silicon Graphics [NEID93] i HOOPS™ firmy Ithaca Software [BASS90]. Chociaż dużo mówiono o różnicach między rozwiązaniami przyjętymi w PHIGS PLUS, OpenGL i HOOPS, są one znacznie bardziej podobne do siebie niż różne, przynajmniej w zakresie podstawowych możliwości, a także w interfejsie programu użytkowego (API). Większość tego, czego się

[1] Pojęcie *PHIGS* w tym rozdziale obejmuje również zbiór rozszerzeń do standardu PHIGS, określanego jako PHIGS PLUS, który zawiera zaawansowane prymitywy geometryczne, np. wielościany, krzywe i powierzchnie, jak również metody renderingu wykorzystujące oświetlenie i cieniowanie, omawiane w rozdz. 12–14.

nauczymy w tym rozdziale, będzie się odnosiła, z niewielkimi zmianami, do innych pakietów, ponieważ we wszystkich z nich jest hierarchia obiektów z przekształceniami.

Opisany tu pakiet jest określany jako *SPHIGS* (od Simple PHIGS; czytamy tę nazwę jako *es-figs*), ponieważ jest on podzbiorem standardu PHIGS. Zachowuje on większość możliwości standardu PHIGS i jego siłę, ale upraszcza i modyfikuje różne funkcje po to, żeby ułatwić jego stosowanie. SPHIGS zawiera również kilka ulepszeń zaadaptowanych z rozszerzenia PHIGS PLUS. Przy projektowaniu pakietu SPHIGS naszym celem było wprowadzanie koncepcji w możliwie najprostszy sposób, a nie prezentowanie pakietu, który jest dokładnie zgodny ze standardem PHIGS. W przypisach wskazujemy na istotne różnice między SPHIGS a PHIGS; funkcje SPHIGS są również dostępne w PHIGS, jeżeli nie zaznaczono, że jest inaczej.

Są trzy istotne różnice między pakietem SPHIGS a całkowitoliczbowymi pakietami takimi jak SRGP albo Xlib dla systemu X Window. Po pierwsze, ze względu na zastosowania inżynierskie i naukowe, w pakiecie SPHIGS wykorzystuje się układ współrzędnych zmiennopozycyjnych 3D i realizuje się sekwencję rzutowania omówioną w rozdz. 6.

Druga istotniejsza różnica polega na tym, że SPHIGS ma bazę danych dla struktur. *Struktura* jest to logiczna grupa prymitywów, atrybutów i innych informacji. Programista może modyfikować struktury w bazie danych za pomocą kilku poleceń edycyjnych; SPHIGS zapewnia to, że obraz na ekranie jest dokładną reprezentacją zawartości zapamiętanej bazy danych. Struktury zawierają nie tylko specyfikację prymitywów i atrybutów, ale również odwołania do podrzędnych struktur. Dlatego struktury mają niektóre właściwości procedur w językach programowania. W szczególności, tak jak hierarchia procedur jest indukowana przez procedury wywołujące podprocedury, tak hierarchia struktur jest indukowana przez struktury wywołujące podstruktury. Taka hierarchiczna kompozycja jest szczególnie użyteczna, gdy można sterować geometrią (pozycja, orientacja, wielkość) i wyglądem (barwa, styl, grubość itd.) dowolnego wywołania podstruktury.

Trzecia różnica polega na tym, że SPHIGS działa w abstrakcyjnym układzie współrzędnych świata 3D, a nie w przestrzeni ekranu 2D i dlatego nie umożliwia bezpośredniego manipulowania pikselami. Ze względu na te różnice SPHIGS i SRGP mają różne przeznaczenie; jak podkreśliliśmy w rozdz. 2, każde z tych rozwiązań ma swoje zalety – żaden pakiet graficzny nie zaspokoi wszystkich potrzeb.

Ze względu na zdolność do obsługiwania hierarchii struktur SPHIGS dobrze nadaje się do zastosowań bazujących na modelach zawierających hierarchię zespół-podzespół; SPHIGS może być traktowany jako prosta specjalizowana hierarchia modelowania. W punkcie 7.1

przyjrzymy się modelowaniu, zanim omówimy specyfikę modelowania geometrycznego za pomocą SPHIGS. W punktach od 7.2 do 7.9 pokażemy, jak tworzyć, wyświetlać i dokonywać edycji bazy danych w SPHIGS. W punkcie 7.10 omówimy interakcję, zwłaszcza korelację wskazywania.

7.1. Modelowanie geometryczne

W kursach nauk fizycznych i społecznych spotkaliśmy wiele przykładów modeli. Na przykład znamy prawdopodobnie model atomu Bohra, w którym elektrony krążą wokół jądra zawierającego neutrony i protony. Inne przykłady to wykładniczy nieograniczony model wzrostu w biologii oraz makro- i mikroekonomiczne modele, które mają na celu opisanie niektórych aspektów ekonomii. Model jest reprezentacją pewnych (niekoniecznie wszystkich) cech konkretnej albo abstrakcyjnej wielkości. Model jakiegoś obiektu ma umożliwić ludziom zobrazowanie i zrozumienie struktury lub zachowania się tego obiektu i dać wygodne narzędzie do eksperymentowania i przewidywania wpływu wejść lub zmian modelu. Ilościowe modele popularne w naukach fizycznych, społecznych i działalności inżynierskiej są zazwyczaj formułowane w postaci układu równań; użytkownik modelu może eksperymentować zmieniając wartości zmiennych niezależnych, współczynników i wykładników. Często modele upraszczają rzeczywistą strukturę albo zachowanie się modelowanego obiektu po to, żeby model był łatwiejszy do wizualizowania albo w przypadku modeli reprezentowanych w postaci układu równań po to, żeby model był łatwiejszy ze względów obliczeniowych.

W książce ograniczamy się do omówienia modeli wykorzystujących komputery – w szczególności do tych, które prowadzą do interpretacji graficznej. Grafika może być wykorzystana do tworzenia i edytowania modelu, do otrzymania wartości dla jego parametrów i do wizualizacji jego zachowania i struktury. Są różne modele i środki graficzne do jego tworzenia i wizualizacji; modele takie jak modele populacji nie muszą zawierać żadnych wewnętrznych aspektów graficznych. Grafika komputerowa jest używana w odniesieniu do wielu popularnych typów modeli, na przykład:

- *Modele typu organizacyjnego*; są to hierarchie reprezentujące instytucjonalne biurokracje i taksonomie, takie jak biblioteczne systemy klasyfikacji czy taksonomie biologiczne. Takie modele mają różne reprezentacje w postaci grafów skierowanych, na przykład schematy organizacyjne.
- *Modele jakościowe*; są to równania opisujące systemy ekonometryczne, finansowe, socjologiczne, demograficzne, klimatyczne, che-

miczne, fizyczne i matematyczne. Modele te są często reprezentowane w postaci grafów albo wykresów statystycznych.

- *Modele geometryczne*; są to zbiory elementów z dobrze zdefiniowaną geometrią i połączeniami między elementami; należą tu struktury inżynierskie i architektoniczne, cząsteczki i inne struktury chemiczne, struktury geograficzne i pojazdy. Te modele są zazwyczaj przedstawiane w postaci schematów blokowych albo za pomocą pseudorealistycznych „fotografii syntetycznych".

Modelowanie wspomagane komputerowo umożliwia twórcom nowych lekarstw modelowanie chemicznego zachowania się składników, które mogą być skuteczne w zwalczaniu określonych chorób, projektantom samolotów przewidywanie deformacji skrzydła w czasie lotu ponaddźwiękowego, nauczanie pilotów latania, przewidywanie przez ekspertów w zakresie reaktorów jądrowych wpływu różnych uszkodzeń w reaktorze i opracowywanie odpowiednich środków zaradczych, projektantom samochodów testowanie zachowania się kabiny pasażera w czasie katastrofy. W tych i w wielu innych przypadkach jest łatwiej, taniej, szybciej i bezpieczniej eksperymentować z modelem niż z obiektem rzeczywistym. A w rzeczywistości w wielu sytuacjach, takich jak szkolenie pilotów stacji kosmicznych i badanie bezpieczeństwa reaktora, modelowanie i symulacja są jedynymi dostępnymi metodami poznawania takich systemów. Z tych powodów modelowanie komputerowe zastępuje bardziej tradycyjne metody, np. testy w tunelach aerodynamicznych. Inżynierowie i naukowcy mogą wykonywać wiele eksperymentów za pomocą cyfrowych tuneli, mikroskopów, teleskopów itd. Takie numeryczne symulowanie i modelowanie staje się szybko nowym paradygmatem w nauce, zyskując pozycję obok tradycyjnych gałęzi teoretycznych i eksperymentów fizycznych. Oczywiste jest, że modelowanie i symulacja są tak dobre, jak dobry jest model i wejścia do niego – stwierdzenie *śmieci na wejściu, śmieci na wyjściu* odnosi się szczególnie dobrze do modelowania.

Modele niekoniecznie muszą zawierać dane geometryczne; takie abstrakcje jak modele organizacji nie są zorientowane przestrzennie. Niemniej większość takich modeli może być reprezentowana geometrycznie; na przykład model organizacji może być reprezentowany przez schemat organizacyjny albo wyniki badań klinicznych lekarstwa mogą być reprezentowane przez histogram. Nawet jeżeli model reprezentuje wewnętrznie geometryczny obiekt, to nie jest określona żadna ustalona reprezentacja geometryczna modelu albo jego widoku. Na przykład możemy wybrać, czy reprezentować robota jako zbiór wielościanów, czy za pomocą powierzchni krzywoliniowych, i możemy określić, jak robot ma być „fotografowany" – z którego punktu obserwacji, za pomocą jakiego rodzaju rzutów geometrycznych i z jakim poziomem reali-

zmu. Możemy również zdecydować się na pokazywanie obrazowe, albo struktury albo zachowania się modelu; na przykład możemy zechcieć obejrzeć zarówno projekt rozmieszczenia układu VLSI w strukturze krzemu, jak i jego zachowanie elektryczne i logiczne w funkcji wejść i czasu.

7.1.1. Modele geometryczne

Modele geometryczne albo graficzne opisują elementy o wewnętrznych właściwościach geometrycznych i dlatego nadają się w naturalny sposób do reprezentacji graficznej. Wśród elementów, jakie może reprezentować model geometryczny, można wyróżnić:

- Przestrzenne rozmieszczenie i kształt elementów (to jest *geometrię* obiektu) i inne atrybuty wpływające na wygląd elementów, np. barwa.
- Połączenia elementów (to jest strukturę albo *topologię* obiektu); zauważmy, że informacja o połączeniach może być przedstawiona abstrakcyjnie (na przykład w postaci macierzy sąsiedztwa dla sieci albo struktury drzewiastej dla hierarchii) albo może mieć własną wewnętrzną geometrię (wymiary kanałów w układzie scalonym).
- Uzależnione od zastosowania wartości danych i właściwości związane z elementami, np. charakterystyki elektryczne albo tekst opisu.

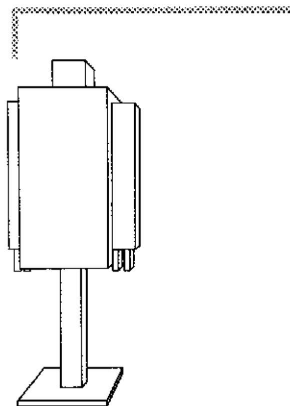
Z modelem mogą być związane algorytmy przetwarzania, np. analiza układów liniowych dla modeli układów dyskretnych, analiza elementów skończonych dla struktur mechanicznych i minimalizacja energii dla modeli molekularnych.

Między tym, co jest zapamiętane bezpośrednio w modelu, a tym, co musi być obliczone przed analizą albo wyświetlaniem musi być zachowany kompromis – klasyczny kompromis to zajętość pamięci-czas. Na przykład model sieci komputerowej mógłby pamiętać bezpośrednio linie połączeń albo mógłby je wyliczać za każdym razem, gdy jest potrzebny nowy obraz na podstawie macierzy połączeń za pomocą prostego algorytmu rozmieszczania grafu. Z modelem musi być zapamiętana dostateczna informacja dla umożliwienia analizy i wyświetlania, ale dokładny format i wybory metod kodowania zależą od zastosowania i kompromisu zajętość pamięci-czas.

7.1.2. Hierarchia modeli geometrycznych

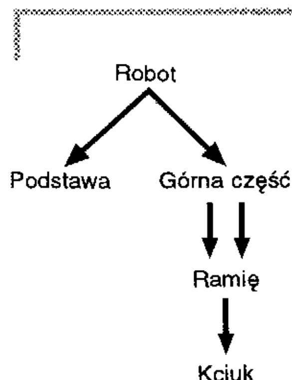
Modele geometryczne często mają strukturę hierarchiczną wynikającą z procesu konstrukcji typu wstępującego; z elementów są tworzone obiekty wyższego poziomu, te z kolei służą jako bloki do budowy obie-

któw jeszcze wyższego poziomu itd. Podobnie jak w systemach programowania, hierarchie rzadko są konstruowane dokładnie według metody wstępującej albo zstępującej; to, o co chodzi, to końcowa hierarchia, a nie sam proces jej konstrukcji. Hierarchia obiektów jest popularna, ponieważ niewiele obiektów jest monolitycznych (niepodzielnych); jeżeli raz zdekomponujemy obiekt na zbiór części, to już utworzyliśmy przynajmniej dwupoziomową hierarchię. W rzadko spotykanym przypadku, kiedy każdy obiekt występuje tylko raz w obiekcie wyższego rzędu, hierarchia może być przedstawiona w postaci drzewa, którego węzłami są obiekty, a relacje zawierania między obiektami są krawędziami. W bardziej typowym przypadku obiektów włączanych wiele razy hierarchia jest reprezentowana przez *skierowany graf acykliczny* (DAG). Jako prosty przykład hierarchii obiektów na rys. 7.1 pokazano rzut perspektywiczny prostego robota – *androida*; na rysunku 7.2a pokazano strukturę robota w postaci DAG. Zauważmy, że powielając niektóre obiekty możemy zmieniać graf DAG na drzewo (rys. 7.2b). Na zasadzie konwencji strzałki są pominięte jako nadmiarowe, ponieważ relacja porządkowania między węzłami wynika ze względnego położenia węzła w drzewie – jeżeli węzeł *A* jest nad węzłem *B*, to *A* zawiera *B*.



Rys. 7.1. Perspektywiczny rzut robota

Robot składa się z górnej części obracającej się na podstawie. Górna część składa się z głowy, która obraca się względem tułowia; tułów ma doczepione dwa identyczne ramiona, które mogą się niezależnie obracać wokół poziomej osi. Ramię składa się ze stałej części – dłoni i kciuka, który porusza się równolegle do dłoni w celu utworzenia prostego uchwytu. Wobec tego obiekt-kciuk jest raz wywoływany z ramieniem, a obiekt-ramię jest wywoływany dwa razy z górną częścią ciała. W tym rozdziale omawiamy tworzenie tego robota; jego kształt jest również prezentowany w postaci trzech rzutów prostokątnych w oknach 2–4 ekranu pokazanego na rys. 7.5b.

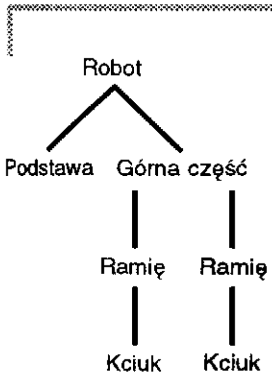


Rys. 7.2a. Reprezentacja hierarchii robota za pomocą skierowanego grafu acyklicznego

Chociaż obiekt w hierarchii jest złożony z geometrycznych prymitywów jak również podobiektów niższego poziomu, graf DAG i drzewo reprezentujące hierarchię robota pokazuje tylko odniesienia do podobiektów. Jest to analogiczne do diagramu hierarchii procedur powszechnie używanego do pokazania struktury wywołań programu w języku proceduralnym wysokiego poziomu. Trzeba zauważyć, że to projektant musi zdecydować o dokładnej hierarchii konstrukcji złożonego obiektu. Na przykład robot mógłby być modelowany za pomocą dwupoziomowej hierarchii, w której baza stanowiłaby korzeń, głowa i tułów byłyby prymitywami geometrycznymi (na przykład prostopadłościany) i dwóch odwołań do *atomowego* obiektu – ramienia złożonego tylko z prymitywów geometrycznych.

Wiele systemów, np. sieci komputerowe lub fabryki chemiczne, może być reprezentowanych za pomocą diagramów sieciowych, w których

obiekty nie tylko są włączane wiele razy, ale również są dowolnie łączone ze sobą. Takie sieci są również modelowane jako grafy, zawierające również cykle, i mimo to wciąż mające właściwości hierachii zawierania obiektów przy wielokrotnym występowaniu podsieci.



Rys. 7.2b. Reprezentacja hierarchii robota w postaci drzewa

W celu uproszczenia zadania budowania złożonych obiektów (i ich modeli) na ogół używamy jako podstawowych bloków elementów atomowych zależnych od stosowania. W 2D te elementy są często rysowane za pomocą szablonów o standardowych kształtach (używa się również nazw symbol i matryca). W programach rysowania te kształty z kolei są składane z prymitywów geometrycznych, takich jak odcinki, prostokąty, wielokąty, łuki eliptyczne i inne. W 3D podstawowymi blokami są walce, prostopadłościany, kule, ostrosłupy i powierzchnie obrotowe. Te kształty 3D mogą być definiowane w zależności od prymitywów geometrycznych niższego rzędu, np. wielokątów 3D; w tym przypadku gładkie powierzchnie krzywoliniowe muszą być aproksymowane za pomocą wielokątów z towarzyszącą utratą rozdzielczości. W zaawansowanych systemach modelowania, które operują bezpośrednio na dowolnych powierzchniach albo objętościach, kształty takie jak parametryczne powierzchnie wielomianowe i bryły takie jak walce, kule i stożki same są prymitywami geometrycznymi i są definiowane analitycznie, bez straty rozdzielczości – por. rozdz. 9 i 10. W tym rozdziale korzystamy z określenia *obiekt* w stosunku do tych elementów 2D i 3D, które są zdefiniowane w ich własnym układzie współrzędnych modelowania w zależności od prymitywów geometrycznych i obiektów niższego poziomu i które zazwyczaj mają nie tylko dane geometryczne, ale również odpowiednie dane związane z zastosowaniem. Obiekt jest to wobec tego (złożony) kształt i wszystkie jego dane.

Hierarchia jest tworzona z wielu względów:

- W celu konstruowania złożonych obiektów w modułowy sposób, typowo na zasadzie wywoływania elementów składowych, które różnią się atrybutami geometrii i wyglądu.
- W celu zwiększenia stopnia wykorzystania pamięci, ponieważ wystarcza tylko zapamiętanie odwołań do obiektów, które są ciągle używane, zamiast ich definiowania za każdym razem.
- W celu ułatwienia wprowadzania zmian, ponieważ zmiana w definicji jednego bloku podstawowego automatycznie przenosi się do wszystkich obiektów wyższego rzędu, które wykorzystują ten obiekt (ponieważ teraz odnoszą się do uaktualnionej wersji); przydatne jest tu podobieństwo między hierarchią obiektów i hierarchią procedur, ponieważ zmiana w ciele procedury również odbija się we wszystkich odwołaniach do tej procedury.

Program użytkowy może korzystać z różnych metod kodowania modeli hierarchicznych. Na przykład sieć albo relacyjna baza danych

może być użyta do zapamiętania informacji o obiektach i relacjach między obiektami. W niektórych modelach połączenia między modelami same są obiektami; muszą one być również reprezentowane przez rekordy danych modelu. Jeszcze inna alternatywa polega na użyciu bazy danych zorientowanej obiektowo [ZDON90]. W graficznych programach użytkowych do pamiętania informacji o modelowaniu obiektów geometrycznych są coraz częściej używane środowiska programowe zorientowane obiektowo, np. SmallTalk [GOLD83], MacApp [SCHM86] i C++ [STRO91].

Połączenia. W większości sieci obiekty są umieszczane w specjalnych miejscach (albo interakcyjnie przez użytkownika, albo automatycznie przez program użytkowy) i potem są łączone ze sobą. Połączenia mogą być abstrakcyjne i wobec tego mogą mieć dowolny kształt (na przykład w diagramach hierarchicznych albo sieciowych, np. wykresy schematu organizacyjnego albo wykresy harmonogramów projektów) albo mogą mieć swoją geometrię (na przykład struktury VLSI). Jeżeli połączenie jest abstrakcyjne, to możemy używać różnych standardowych konwencji do rozmieszczania hierarchicznych lub sieciowych diagramów i korzystać z różnych standardowych atrybutów, np. styl linii, szerokość linii albo barwa, do oznaczenia różnych typów relacji (na przykład odpowiedzialność zaznaczona linią przerywaną w schematach organizacji). Połączenia, dla których kształt jest istotny, np. kanały łączące tranzystory i inne elementy układu VLSI, są często ograniczane, jeśli chodzi o kierunek, na przykład do poziomego i pionowego (czasami używa się nazwy: schemat połączeń typu Manhattan) w celu uproszczenia wizualizacji i konstrukcji fizycznej.

Przekazywanie parametrów w hierarchii obiektów. Obiekty wywoływane jako bloki podstawowe muszą być umieszczane dokładnie w miejscu określanym przez obiekty macierzyste i często muszą być w związku z tym poddawane skalowaniu i reorientacji. Do przesuwania prymitywów w rozdz. 5 i do normalizowania bryły widzenia w rozdz. 6 używano macierzy współrzędnych jednorodnych; podobnie nie powinno budzić zdziwienia częste używanie w modelach hierarchicznych macierzy skalowania, obrotu i przesuwania w stosunku do podobiektów. Sutherland jako pierwszy używał tej możliwości do modelowania graficznego w systemie Sketchpad [SUTH63], używając terminu *master* do określenia obiektu i terminu *instance* do gemetrycznie przekształconego odwołania. Jak pokazano w p. 4.3.3, systemy graficzne wykorzystujące *hierarchiczną listę wyświetlania* (określaną również jako *strukturalny plik wyświetlania*) realizują hierarchię master-instance w sprzęcie, korzystając ze skoków do podprogramów i szybkiej jednostki arytmetyki zmiennopozycyjnej dla przekształceń. Ponieważ chcemy rozróżnić przekształcenia geometryczne wykorzystywane do normalizacji bryły widze-

nia od używanych w tworzeniu hierarchii obiektów, często określamy te ostatnie jako *przekształcenia modelowania*. Oczywiście matematycznie nie ma różnicy między przekształceniami modelującym i normalizującym.

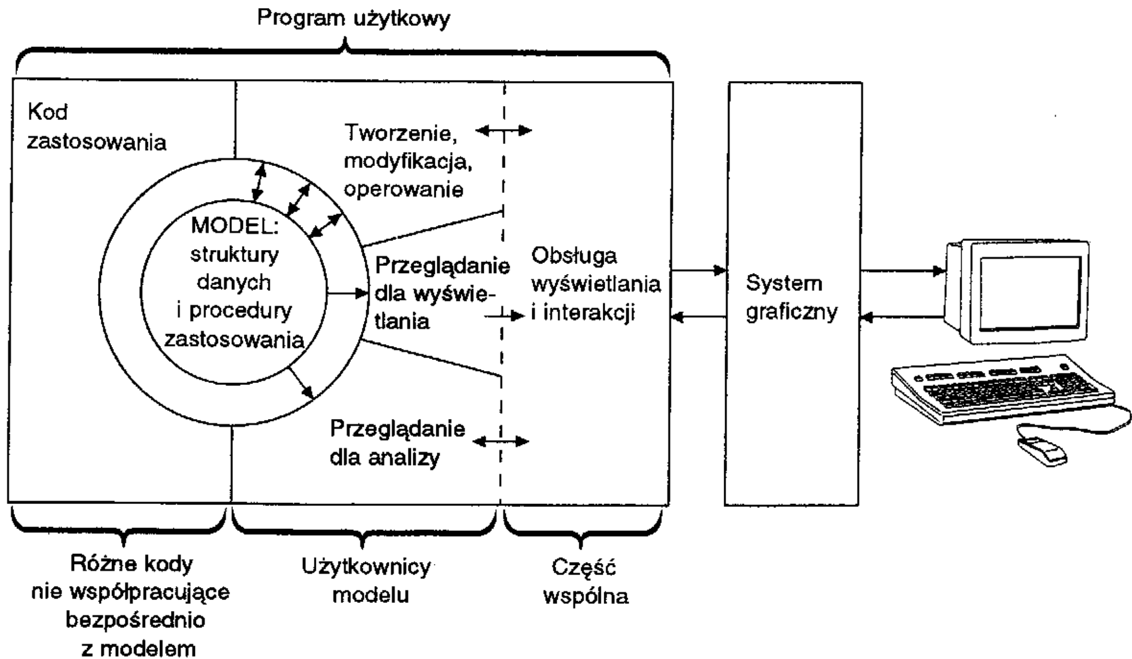
Przez analogię do hierarchii procedur, niekiedy mówimy o wywoływaniu przez obiekt macierzysty obiektu potomka w hierarchii i przekazywaniu mu parametrów geometrycznych odpowiadających jego skali, orientacji i pozycji w systemie współrzędnych ojca. Jak zobaczymy, pakiety graficzne wykorzystujące hierarchię obiektów, np. SPHIGS, mogą pamiętać, tworzyć i stosować macierze przekształceń do wierzchołków prymitywów, jak również do wierzchołków obiektów dzieci. Co więcej, atrybuty wpływające na wygląd również mogą być przekazywane do podległych obiektów. W p. 7.5.3 zobaczymy, że mechanizm przekazywania parametrów w SPHIGS nie jest tak ogólny jak w językach proceduralnych.

7.1.3. Zależność między modelem, programem użytkowym i systemem graficznym

Dotychczas mówiliśmy ogólnie o modelach i bardziej szczegółowo o modelach geometrycznych z hierarchią i o przekształceniach modelowania. Zanim omówimy SPHIGS, przyjrzyjmy się koncepcyjnemu modelowi grafiki pokazanemu pierwotnie na rys. 1.11 i rozwiniętemu na rys. 3.2, w celu pokazania wzajemnych zależności między modelem, programem użytkowym i systemem graficznym. W schemacie na rys. 7.3 programy użytkowe są podzielone na pięć podsystemów, oznaczonych od (a) do (e):

a. Budowanie, modyfikowanie i utrzymanie modelu na zasadzie dodawania, usuwania i zastępowania informacji.
b. Przeglądanie modelu w celu wybrania informacji do wyświetlania.
c. Przeglądanie modelu w celu wybrania informacji wykorzystywanej do analizy zachowania/parametrów modelu.
d. Wyświetlanie zarówno informacji (na przykład rendering modelu geometrycznego będącego wynikiem analizy), jak i narzędzi interfejsu użytkownika (na przykład menu, pola dialogowe).
e. Wykonanie różnych zadań programu użytkowego nie wpływających bezpośrednio na model lub wyświetlanie (na przykład porządkowanie).

Określenie *podsystem* nie oznacza dużych modułów kodu – kilka wywołań albo krótka procedura mogą wystarczyć do realizacji danego podsystemu. Co więcej, podsystem może być rozproszony w programie



Rys. 7.3. Model zastosowania i jego użytkownicy

użytkowym i nie musi być zebrany w oddzielnym module programowym. Dlatego na rys. 7.3 pokazano elementy logiczne, a nie elementy struktury programu; co więcej, ponieważ rysunek rozróżnia procedury, które budują, modyfikują albo wyświetlają model, nie zawsze jest jasne, czy wywoływać określoną część modelu, czy część kodu realizującego model. Można by na przykład spierać się, czy moduł analizy układu jest rzeczywiście częścią definicji modelu, ponieważ opisuje on zachowanie układu. Dla programisty, korzystającego z tradycyjnych języków programowania, np. Pascal lub C, rysunek 7.3 jest najlepiej zrozumiały, jeżeli przyjmie się, że model głównie zawiera dane. Ci, którzy znają języki programowania obiektowego, przyjmą mieszaninę danych i procedur jako coś naturalnego.

W wielu programach użytkowych, zwłaszcza przemysłowych, obowiązuje reguła 80/20: większa część programu (80%) zajmuje się modelowaniem obiektów, a tylko mała część (20%) zajmuje się tworzeniem ich obrazów. Innymi słowy, w wielu programach użytkowych, np. CAD, obrazowa reprezentacja modelu jest pomocą przy analizie, fizycznej konstrukcji, obróbce sterowanej numerycznie, albo przy innym rodzaju przetwarzania końcowego. Naturalnie jest również wiele zastosowań, dla których obraz jest istotą rzeczy – na przykład malowanie, rysowanie, wytwarzanie filmu i wideo, animacja scen dla symulatorów

lotu. Wszystkie te zastosowania, oprócz malowania, wymagają modelu, na podstawie którego jest wykonywany rendering obrazów. Mówiąc krótko, większość zastosowań grafiki wymaga modelowania (i symulacji) i jest wiele prawdy w stwierdzeniu „grafika jest modelowaniem"; rozdziały 9 i 10 są poświęcone temu ważnemu zagadnieniu.

7.2. Charakterystyka pakietów graficznych wykorzystujących tryb podtrzymywania

Przy omawianiu roli programu użytkowego, modelu zastosowania i pakietu graficznego skomentowaliśmy dokładnie możliwości pakietu graficznego i skutki modyfikacji modelu. SRGP opisany w rozdz. 2 działa w *trybie natychmiastowym* i nie przechowuje żadnych rekordów z przekazanymi do niego parametrami i atrybutami. Dlatego usunięcie i zmiana w obiekcie danego zastosowania wymaga usunięcia informacji na ekranie i albo selektywnej modyfikacji, albo regeneracji ekranu; w każdym z tych przypadków program użytkowy musi na nowo podać specyfikację prymitywów ze swojego modelu. PHIGS działa w *trybie podtrzymania*: przechowuje rekord wszystkich prymitywów i innych związanych informacji po to, żeby umożliwić kolejne edytowanie i automatyczne uaktualnianie ekranu, zwalniając z tego program użytkowy.

7.2.1. Główna pamięć struktury i jej zalety

PHIGS pamięta informacje w specjalnej bazie danych nazywanej *główną pamięcią struktury* (CSS). *Struktura* w PHIGS jest sekwencją *elementów* – prymitywów, atrybutów widoczności, macierzy przekształceń i odwołań do podrzędnych struktur – których celem jest definiowanie spójnego elementu geometrycznego. PHIGS pamięta specjalizowaną hierarchię modelowania, uzupełnioną o przekształcenia modelowania i inne atrybuty przekazywane jako parametry do podrzędnych struktur. Zauważmy podobieństwa hierarchii modelowania CSS do sprzętowej hierarchicznej listy wyświetlania, która pamięta hierarchię oryginał-kopia. W efekcie PHIGS można traktować jako specyfikację pakietu niezależnej od urządzenia hierarchicznej listy wyświetlania; dana realizacja jest oczywiście optymalizowana dla określonego urządzenia wyświetlającego, ale programista zajmujący się zastosowaniem nie musi się tym zajmować. Wiele realizacji PHIGS jest czysto programowych, ale najpopularniejszym rozwiązaniem jest wykonywanie operacji CSS programowo i wykorzystywanie kombinacji sprzętu i oprogramowania do wykonywania renderingu.

Podobnie jak to robi lista wyświetlania, CSS dubluje informację geometryczną zapamiętaną w bardziej uniwersalnym modelu albo bazie danych zastosowania w celu umożliwienia szybkiego przeglądania ekranu – to znaczy przeglądania wykorzystywanego do obliczenia nowego rzutu modelu. Podstawową zaletą CSS jest szybkie automatyczne regenerowanie ekranu wówczas, gdy program użytkowy uaktualnia CSS. Sama ta cecha warta jest powielenia danych geometrycznych w bazie danych programu użytkowego i w CSS, zwłaszcza jeżeli realizacja PHIGS wykorzystuje oddzielny procesor do wspomagania przeglądania po to, żeby odciążyć CPU wykonujące program użytkowy od przeglądania ekranu. PHIGS umożliwia również efektywne wykonywanie małych edycji, takich jak zmiana macierzy przekształceń.

Drugą zaletą CSS jest automatyczne wykonywanie korelacji wskazywania: pakiet określa obiekt i jego miejsce w hierarchii dla prymitywu wskazanego przez użytkownika (por. p. 7.10.2). Możliwość realizacji korelacji wskazywania to przykład popularnej metody przesuwania często potrzebnych działań do odpowiedniego pakietu graficznego.

Trzecią zaletą CSS jest to, że możliwości edycyjne, w połączeniu z możliwościami modelowania hierarchicznego, ułatwiają tworzenie różnych efektów dynamicznych – na przykład dynamiki ruchu – w których przekształcenia zmienne w czasie są wykorzystywane do skalowania, obracania i pozycjonowania podobiektów w kontekście obiektów macierzystych. Możemy na przykład stworzyć taki model naszego prostego robota, żeby każde złącze było reprezentowane przez obrót zastosowany do podstruktury (to znaczy ramię jest obrotową częścią podrzędną dla górnej części), i dynamicznie obracać ramię przez edytowanie macierzy obrotu.

7.2.2. Ograniczenia pakietów działających w trybie podtrzymania

Chociaż CSS (jako specjalizowana jednostka zbudowana przede wszystkim do wyświetlania i szybkiego uaktualniania przyrostowego) umożliwia wykonywanie pewnych popularnych operacji modelowania, ani nie jest konieczna, ani wystarczająca dla celów modelowania. Nie jest konieczna, ponieważ program użytkowy może wykonywać sam regenerację ekranu wówczas, gdy model ulega zmianie, może sam wykonywać korelację wskazywania (chociaż przy istotnym nakładzie pracy) i może realizować swoją własną hierarchię obiektów za pomocą procedur definiujących obiekty i akceptujących przekształcenia i inne parametry. CSS ogólnie nie wystarcza, ponieważ w większości zastosowań oddzielnie zbudowana i uaktualniana struktura danych programu użytkowego jest wciąż potrzebna do rejestrowania wszystkich odpowiednich danych

dla każdego obiektu programu użytkowego. Stąd jest powielanie wszystkich danych geometrycznych i te dwie reprezentacje muszą być właściwie zsynchronizowane. Z tych powodów niektóre pakiety graficzne wykorzystują współrzędne zmiennopozycyjne i uogólnione możliwości rzutowania 2D i 3D bez jakiejkolwiek struktury pamięci. Uzasadnieniem dla takich pakietów z trybem natychmiastowym jest to, że utrzymanie CSS często nie jest warte narzutów, ponieważ program użytkowy na ogół ma model wystarczający do regenerowania wyglądu ekranu.

W zastosowaniach, w których jest istotna zmiana strukturalna między kolejnymi obrazami, korzystanie pakietu z trybem podtrzymania nie opłaca się. Na przykład w analizie cyfrowego tunelu dla skrzydła samolotu, gdzie powierzchnia jest reprezentowana za pomocą siatki trójkątów, większość wierzchołków zmienia nieco swoją pozycję pod wpływem sił aerodynamicznych działających na skrzydło. Edycja strukturalnej bazy danych dla takiego przypadku nie ma sensu, ponieważ w każdym nowym obrazie większość danych jest zastępowana. Edytowanie strukturalnej bazy danych PHIGS nie jest zalecane, jeżeli liczba elementów, które mają być poddane edycji, jest mała w porównaniu z wielkością wyświetlanej sieci. Narzędzia do edycji dostępne w PHIGS są elementarne; na przykład jest łatwo zmienić macierz przekształcenia, ale żeby zmienić wierzchołek wielokąta, trzeba usunąć wielokąt i na nowo podać specyfikację zmienionej wersji. Typowo implementacje są raczej optymalizowane ze względu na przeglądanie ekranu, ponieważ jest to najpopularniejsza operacja, niż na masową edycję. Dalej, model programu użytkowego musi być uaktualniany w każdym przypadku i jest łatwiej i szybciej uaktualnić jedną bazę danych niż dwie.

Ze względu na te ograniczenia wiele realizacji standardu PHIGS oferuje możliwość pracy w trybie natychmiastowym lub nawet tryb hybrydowy, w którym prymitywy określone w trybie natychmiastowym są łączone z podtrzymywanymi prymitywami.

7.3. Struktury dla definiowania i wyświetlania

W poprzednim punkcie omówiono ogólne właściwości PHIGS i SPHIGS. W tym punkcie zaczynamy szczegółowy opis pakietu SPHIGS; dyskusja ta jest słuszna również i dla PHIGS. Operacje dozwolone w SPHIGS obejmują:

- Otwieranie (dla zapoczątkowania edycji) i zamykanie (dla zakończenia edycji).
- Usuwanie.
- Umieszczanie *elementów struktury* (trzema podstawowymi typami elementów strukturalnych są *prymitywy*, *atrybuty*, w tym te elemen-

ty, które określają przekształcenia modelowania, oraz elementy, które wywołują podstruktury). Element jest to rekord danych, który jest tworzony i umieszczany w bieżąco otwartej strukturze wówczas, gdy jest wywoływana procedura generatora elementu, i który pamięta parametry procedury.
- Usuwanie elementów struktury.
- Wysyłanie w celu wyświetlenia (przez analogię do wysyłania zdjęcia w biuletynie elektronicznym), zgodnie z operacją rzutowania, która określa, jak odwzorować współrzędne zmiennopozycyjne na układ współrzędnych ekranu.

7.3.1. Otwieranie i zamykanie struktur

W celu utworzenia struktury – na przykład zbioru prymitywów i atrybutów tworzących ramię robota z rys. 7.2 – bierzemy w nawiasy wywołania do funkcji generatora elementu z wywołaniami do

```
void SPH_openStructure ( int structureID );
void SPH_closeStructure();
```

Te funkcje wykonują na strukturach w zasadzie to samo co standardowe polecenia otwarcia i zamknięcia pliku dla plików dyskowych. Jednak w przeciwieństwie do plików dyskowych w danym czasie może być otwarta tylko jedna struktura i są w niej zapamiętywane wszystkie elementy określone w czasie jej otwierania. Po zamknięciu struktury mogą być ponownie otwierane dla celów edycji (por. p. 7.9).

Zwróćmy uwagę na dwie dodatkowe właściwości struktur. Po pierwsze, prymitywy i atrybuty mogą być określone jedynie jako elementy struktury. Nie ma reguł określających, ile elementów może być zapamiętanych w strukturze; struktura może być pusta albo może zawierać dowolnie dużą liczbę elementów, ograniczoną tylko przez pojemność pamięci. Oczywiście elementy tworzące strukturę powinny być w ogólnym przypadku logicznie spójnym zbiorem, który określa jeden obiekt.

Po drugie, identyfikatory struktur są liczbami całkowitymi. Ponieważ zwykle są one używane tylko przez program użytkowy, a nie przez użytkownika współpracującego interakcyjnie, nie muszą mieć bardziej ogólnej postaci łańcucha znaków, chociaż programista ma swobodę w określaniu stałych symbolicznych dla identyfikatora struktury. Całkowitoliczbowe identyfikatory umożliwiają również wygodne odwzorowanie między obiektami w strukturze danych zastosowania i odpowiadającej obiektowi strukturze identyfikatora.

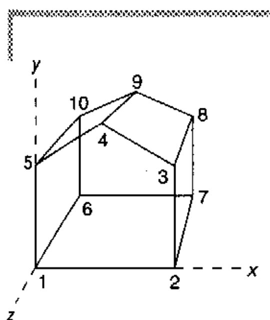
7.3.2. Specyfikowanie prymitywów wyjściowych i ich atrybutów

Funkcje, które generują elementy wyjściowych prymitywów, wyglądają jak ich odpowiedniki w SRGP; są jednak istotne różnice. Po pierwsze, punkty są określane za pomocą współrzędnych zmiennopozycyjnych x, y i z. Ponadto, te funkcje umieszczają elementy w bieżąco otwartej strukturze w CSS, a nie zmieniają bezpośrednio wyglądu ekranu – wyświetlanie struktur jest oddzielną operacją opisaną w p. 7.3.3. W tym rozdziale określenie *prymityw* jest używane jako skrót dla trzech związanych ze sobą wielkości: funkcji generowania elementu, takiej jak SPH_polyLine; elementu struktury generowanej przez tę funkcję (na przykład element polyLine); wyświetlanego obrazu tworzonego wówczas, gdy element prymitywu jest wykonywany w czasie przeglądania głównej pamięci struktury przy wyświetlaniu. W SPHIGS wykonanie elementu prymitywu polega na przekształceniu współrzędnych prymitywu za pomocą przekształceń modelujących i operacji rzutowania, włączając w to obcinanie do bryły widzenia i potem rasteryzację (to jest zamianę na piksele). Atrybuty są bardziej wyspecjalizowane niż w SRGP w tym sensie, że każdy typ prymitywu ma swoje własne atrybuty. Atrybuty takie jak barwa są wpisywane, co oznacza, że programista może na przykład usunąć bieżącą barwę odcinków i zachować bieżące barwy wielościanów i tekstu.

Prymitywy. SPHIGS ma mniej prymitywów wyjściowych niż SRGP, ponieważ odpowiedniki 3D niektórych prymitywów SRGP (na przykład elipsoid) są kosztowne obliczeniowo w realizacji, zwłaszcza w odniesieniu do przekształceń, obcinania i przeglądania wierszowego.

Jeśli chodzi o metody specyfikacji, to większość prymitywów SPHIGS jest identyczna jak ich odpowiedniki w SRGP (z wyjątkiem tego, że punkty są 3D):

```
void SPH_polyLine ( int vertexCount, pointList vertices );
void SPH_polyMarker ( int vertexCount, pointList vertices );
void SPH_fillArea ( int vertexCount, pointList vertices );
   /* Podobnie jak SRGP_polygon */
void SPH_text ( point origin, char *str );
   /* Nie w pełni 3D; zobacz p. 7.7.2 */
```



Rys. 7.4. Prosty dom zdefiniowany jako zbiór wierzchołków i ścian

Zauważmy, że SPHIGS nie weryfikuje, czy wypełniane obszary (albo ścianki opisane dalej) są płaskie; jeżeli nie są płaskie, to wyniki nie są zdefiniowane.

Rozważmy teraz definicję prostego domu pokazanego na rys. 7.4. Możemy opisać ten dom w SPHIGS specyfikując każdą ścianę jako wypełniony obszar, przy niepotrzebnym dwukrotnym koszcie specyfikacji i pamięci (w CSS) dla każdego wspólnego wierzchołka. To zdublowanie zwalnia również generowanie obrazu, ponieważ obliczenia

związane z operacją rzutowania byłyby wykonywane na jednym wierzchołku więcej niż raz. Znacznie efektywniejsze, jeśli chodzi o pamięć i czas przetwarzania, jest określanie ścian za pomocą pośrednich odwołań do wspólnych wierzchołków. Traktujemy więc wielościan jako zbiór ścian, przy czym każda ściana jest określona przez listę indeksów wierzchołków, a każdy indeks jest wskaźnikiem na listę wierzchołków. Możemy opisać specyfikację wielościanu korzystając z następującej notacji:

Polyhedron = {VertexList, FacetList}
VertexList = {V1, V2, V3, V4, V5, V6, V7, V8, V9, V10}
V1 = (x_1, y_1, z_1), V2 = (x_2, y_2, z_2), ...
FacetList = {front = {1, 2, 3, 4, 5}, right = {2, 7, 8, 3}, ... bottom = {...}}

SPHIGS umożliwia taką efektywną formę specyfikacji w odniesieniu do prymitywu wielościan. W terminologii SPHIGS wielościan jest zbiorem ścian, które mogą, ale nie muszą, tworzyć zamkniętą bryłę. W zamkniętym wielościanie takim jak dom wierzchołki na ogół są wspólne dla co najmniej trzech ścian i efektywność pośredniej metody specyfikowania jest wyjątkowo duża. Wygląd wielościanu zależy od tych samych atrybutów, które stosują się do wypełnianych obszarów.

Lista ścian jest prezentowana generatorowi elementu wielościan w postaci tablicy liczb całkowitych (typ „vertexIndexlist w SPHIGS) pamiętającej połączony zbiór opisów ścian. Każdy opis ściany jest sekwencją $(V+1)$ liczb całkowitych, przy czym V jest liczbą wierzchołków ściany. Pierwsze V liczb całkowitych to indeksy do listy wierzchołków; ostatnia liczba to (-1) i oznacza koniec specyfikacji ściany. Tak więc określimy ściany domu (za pomocą czwartego parametru funkcji, opisanego niżej) wysyłając tablicę: 1, 2, 3, 4, 5, −1, 2, 7, 8, 3, −1, ...

```
void SPH_polyhedron
    ( int vertexCount, int facetCount, pointList vertices, vertexIndexList facets);
```

Zauważmy, że algorytmy renderingu w SPHIGS wymagają, żeby rozróżnialne były dwie strony ściany (wewnętrzna i zewnętrzna). Dlatego, aby sprawdzić, czy strona ścianki jest zewnętrzna, wierzchołki muszą być określone w kierunku przeciwnym względem ruchu wskazówek zegara (reguła prawej ręki)[2].

[2] W SPHIGS wymaga się, żeby jedna strona każdej ściany była traktowana jako zewnętrzna, nawet jeżeli ściany wielościanu nie tworzą zamkniętego obiektu. Co więcej, wewnętrzna strona wielościanu nigdy nie jest widoczna.

Dla przykładu, następujący kod w języku C tworzy strukturę składającą się z jednego wielościanu modelującego dom z rys. 7.4:

```
SPH_openStructure (HOUSE_STRUCT);
   SPH_polyhedron (10, 7, houseVertexList, houseFacetDescriptions);
SPH_closeStructure();
```

W istocie, SPHIGS dopuszcza jedynie geometrię wielokątową. Bardziej zaawansowane prymitywy modelowania 3D będą omówione później – gładkie krzywe i powierzchnie definiowane wielomianowo w rozdz. 9 i bryły w rozdz. 10.

Atrybuty. Funkcje wymienione w programie 7.1 generują elementy – atrybuty. W czasie przeszukiwania ekranu wykonanie elementu atrybutu zmienia wartość atrybutu w sposób modalny: nowa wartość obowiązuje dopóty, dopóki nie zostanie bezpośrednio zmieniona. Atrybuty są związane z prymitywami w czasie przeglądania ekranu, tak jak to zostanie omówione w następnym punkcie i w p. 7.7.

Program 7.1 Funkcje generujące elementy – atrybuty

```
polyLine:
   void SPH_setLineStyle( style CONTINUOUS / DASHED / DOTTED / DOT_DASHED );
   void SPH_setLineWidthScaleFactor( double scaleFactor );
   void SPH_setLineColor( int colorIndex);

fill area and polyhedron:
   void SPH_setInteriorColor( int colorIndex );
   void SPH_setEdgeFlag( flag EDGE_VISIBLE / EDGE_INVISIBLE );
   void SPH_setEdgeStyle( style CONTINUOUS / DASHED / DOTTED / DOT_DASHED );
   void SPH_setEdgeWidthScaleFactor( double scaleFactor );
   void SPH_setEdgeColor( int colorIndex );

polyMarker:
   void SPH_setMarkerStyle( style MARKER_CIRCLE / MARKER_SQUARE / ... );
   void SPH_setMarkerSizeScaleFactor( double scaleFactor );
   void SPH_setMarkerColor( int colorIndex );

text:
   void SPH_setTextFont( int fontIndex );
   void SPH_setTextColor( int colorIndex );
```

Atrybuty wypełniania powierzchni różnią się od atrybutów wielokątów w SRGP. Prymitywy wypełniania obszaru i wielościany mają wnętrza i krawędzie, których atrybuty są specyfikowane niezależnie. Wnętrze ma tylko atrybut barwy, poczas gdy dla krawędzi określa się styl, szerokość i barwę. Co więcej, widoczność krawędzi można wyłączyć za pomocą atrybutu wskaźnika krawędzi, co jest wygodne w wielu trybach renderingu; omawiamy to w p. 7.8.

Szerokość odcinka albo krawędzi i wielkość znacznika są określone w sposób niegeometryczny: nie są one zdefiniowane za pomocą układu współrzędnych świata i dlatego nie są przedmiotem przekształceń geometrycznych. Dlatego przekształcenia modelujące i operacje rzutowania mogą zmienić długość widocznego odcinka, ale nie mogą zmienić jego szerokości. Podobnie długość kresek w stylu linii przerywanej jest niezależna od przekształceń stosowanych do odcinka. Jednak, inaczej niż w SRGP, piksele nie są używane jako jednostki miary, ponieważ ich wielkości zależą od urządzenia. Dla każdego urządzenia jest przypisana nominalna szerokość/wielkość i jednostka szerokości/wielkości będzie wyglądała z grubsza tak samo na każdym urządzeniu wyjściowym; zastosowanie SPHIGS określa (niecałkowitą) wielokrotność tej nominalnej szerokości/wielkości.

SPHIGS nie umożliwia korzystania ze wzorców z trzech powodów. Po pierwsze, SPHIGS rezerwuje wzorce do symulowania cieniowania w systemach dwupoziomowych. Po drugie, gładkie cieniowanie obszarów pokrytych wzorcami w systemie barwnym jest zbyt złożone obliczeniowo dla większości systemów wyświetlania. Po trzecie, typ wzorca geometrycznego określany w PHIGS jako hatching (*kreskowanie*) jest również zbyt czasochłonny nawet dla systemów wyświetlania ze sprzętowymi przekształceniami w czasie rzeczywistym.

7.3.3. Wysyłanie struktur w celu wyświetlenia

SPHIGS rejestruje nowo utworzoną strukturę w CSS, ale nie wyświetla jej dopóty, dopóki program użytkowy nie wyśle struktury przy określonych parametrach rzutowania[3]. SPHIGS wykonuje potem przeglądanie elementów struktury w CSS, wykonując każdy element w kolejności od pierwszego do ostatniego. Wykonanie elementu prymitywu wpływa na obraz ekranu (jeżeli część prymitywu jest w polu wizualizacji). Wykonanie elementu atrybutu (zarówno przekształcenia geometryczne, jak i atrybuty wyglądu) zmienia zbiór atrybutów zapamiętanych w wektorze stanu (stan atrybutu przeglądania), który jest stosowany do kolejnych prymitywów po ich napotkaniu w trybie modalnym. Dlatego atrybuty są stosowane do prymitywów w kolejności przeglądania.

[3] Ten sposób określania struktury ekranu jest największą różnicą między PHIGS a SPHIGS. W ogólniejszym mechanizmie PHIGS specyfikacja rzutu jest elementem struktury; dzięki temu rzut może być zmieniony w czasie przeglądania ekranu i może być poddany edycji tak jak każdy inny element. Wiele z obecnych realizacji PHIGS również wykorzystuje prostszy mechanizm wysyłania podobnie jak SPHIGS.

Następująca funkcja dodaje strukturę do listy wysłanych struktur utrzymywanych wewnętrznie przez SPHIGS:

```
void SPH_postRoot ( int structureID, int viewIndex );
```

Określenie *root* oznacza, że przy wysyłaniu struktury *S*, która odwołuje się do podstruktur, faktycznie wysyłamy hierarchiczny DAG, określany jako sieć struktury, której korzeniem jest *S*. Nawet jeżeli wysłana struktura nie wywołuje podstruktury, to jest określana jako korzeń; wszystkie wysłane struktury są korzeniami.

Parametr *viewIndex* wybiera pozycję tablicy rzutów (omawiamy ją w następnym punkcie); ta pozycja określa, jak współrzędne prymitywów struktury mają być odwzorowane na przestrzeń współrzędnych całkowitych ekranu.

Możemy usunąć obraz obiektu z ekranu usuwając struktury (albo elementy) z CSS (por. p. 7.9) albo korzystając z mniej drastycznej funkcji SPH_unpostRoot, która usuwa korzeń z listy wysłanych korzeni bez usuwania go z CSS:

```
void SPH_unpostRoot ( int structureID, int viewIndex );
```

7.3.4. Rzutowanie

Syntetyczna kamera. Jak zauważono w rozdz. 6, wygodnie jest myśleć o pakiecie graficznym jak o syntetycznej kamerze, która wykonuje zdjęcia świata 3D, w którym znajdują się obiekty zdefiniowane geometrycznie. Tworzenie struktury jest równoważne ustawianiu obiektu w studiu fotograficznym; wysyłanie struktury jest analogiczne do wykonania zdjęcia kamerą ustwioną wcześniej tak, żeby była skierowana na scenę, a potem wysłania zdjęcia sceny do biuletynu elektronicznego. Jak zobaczymy, za każdym razem, gdy coś się zmienia w scenie, nasza syntetyczna kamera automatycznie tworzy nowy uaktualniony obraz, który jest wyświetlany na miejscu poprzedniego. W celu stworzenia animacji pokazujemy wiele obrazów statycznych szybko jeden po drugim, tak jak to robi kamera filmowa.

Kontynuując metaforę zastanówmy się, jak jest tworzony obraz syntetyczny. Najpierw operator kamery musi umiejscowić i zorientować kamerę, a potem określić, jaka część sceny powinna być widoczna: na przykład czy kamera ma zrobić zdjęcie z bliska, pokazując część interesującego obiektu, czy widok całej sceny z dalszej odległości? Z kolei fotograf musi zdecydować, jak dużą zrobić odbitkę: czy ma to być odbitka wielkości portfela czy plakatu? Wreszcie trzeba określić miejsce, gdzie fotografia ma być umieszczona. W SPHIGS te kryteria są

reprezentowane w funkcji *view*, która zawiera specyfikację operacji rzutowania; pole wizualizacji dla tej operacji określa wielkość fotografii i jej pozycję. Nie wszystkie obiekty w bazie danych struktury muszą być fotografowane za pomocą kamery o takich samych nastawach. Można określić wielokrotne rzuty, tak jak to wkrótce zobaczymy.

Pole wizualizacji. Tak jak to powiedziano w poprzednim rozdziale, pole wizualizacji określa prostopadłościan w układzie znormalizowanych współrzędnych rzutowania NPC, na który jest odwzorowana zawartość bryły widzenia określonej w układzie współrzędnych odniesienia dla rzutowania (VRC). Układ VRC jest odwzorowany na układ współrzędnych całkowitoliczbowych fizycznego urządzenia w ustalony sposób; pole wizualizacji również określa, gdzie na ekranie ma się pojawić obraz. Układ NPC 3D jest odwzorowywany na współrzędne 2D ekranu w następujący sposób: sześcian jednostkowy NPC mający jeden wierzchołek w punkcie (0, 0, 0) i przeciwny wierzchołek w punkcie (1, 1, 1) jest odwzorowywany na największy kwadrat, jaki może być wpisany w ekran; współrzędna z jest po prostu pomijana. Na przykład w urządzeniu wyświetlającym o rozdzielczości 1024 punkty w poziomie i 800 punktów w pionie punkt $(0{,}5,\ 0{,}75,\ z)_{NPC}$ jest odwzorowywany na punkt $(511,\ 599)_{DC}$. Ze względu na przenoszalność program użytkowy nie powinien wykorzystywać przestrzeni NPC leżącej poza sześcianem jednostkowym; często jednak zalety pełnego wykorzystania ekranu o kształcie różnym od kwadratowego mogą być warte kosztów związanych z przenoszalnością.

Tablica rzutów. SPHIGS ma tablicę rzutów o liczbie pozycji zależnej od konkretnej realizacji. Każdy rzut składa się ze specyfikacji bryły widzenia i pola wizualizacji określanego jako *reprezentacja rzutu* oraz listy (początkowo pustej), która zawiera wysłane do niej korzenie. Pozycja 0 tablicy określa *domniemany rzut* o objętości opisanej na rys. 6.19b z płaszczyznami przednią i tylną w $z = 0$ i $z = -\infty$. Polem wizualizacji dla tego domniemanego rzutu jest jednostkowy sześcian NPC.

Reprezentacje rzutów dla wszystkich pozycji w tablicy (poza rzutem 0) mogą być edytowane przez program użytkowy za pomocą:

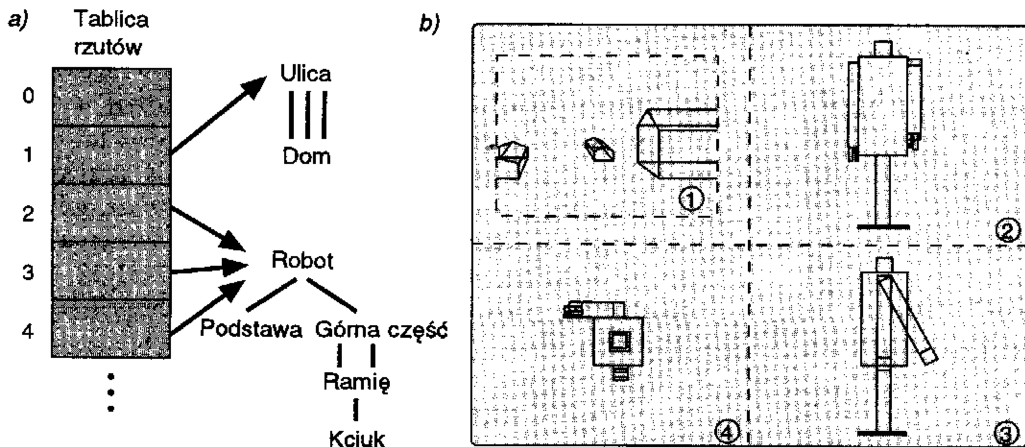
```
void SPH_setViewRepresentation(
    int viewIndex, matrix_4x4 voMatrix, matrix_4x4 vmMatrix,
    double NPCviewport_minX, double NPCviewport_maxX,
    double NPCviewport_minY, double NPCviewport_maxY,
    double NPCviewport_minZ, double NPCviewport_maxZ );
```

Dwie macierze współrzędnych jednorodnych 4×4 to macierz orientacji rzutu i macierz odwzorowania rzutu opisane w rozdz. 6. Są one tworzone przez funkcje pokazane w programie 7.2.

Program 7.2 Narzędzia do tworzenia macierzy przekształcania rzutu

```
/* Ustawienie układu współrzędnych odniesienia UNV */
void SPH_evaluateViewOrientationMatrix( point_3D viewRefPoint,
        vector_3D viewPlaneNormal, vector_3D viewUpVector,
        matrix_4x4 *voMatrix );

/* Ustawienie bryły widzenia i opisanie, jak jest ona odwzorowywana na przestrzeń NPC */
void SPH_evaluateViewMappingMatrix(
    /* Najpierw określamy granice rzutni */
        double umin, double umax, double vmin, double vmax,
        int projectionType,          /* PARALLEL / PERSPECTIVE */
        point_3D projectionReferencePoint,                    /* w VRC */
        double frontPlaneDistance, double backPlaneDistance,  /* płaszczyzny
                                                                 obcinające */
    /* Teraz określamy pole wizualizacji NPC */
        double NPCvp_minX, double NPCvp_maxX,
        double NPCvp_minY, double NPCvp_maxY,
        double NPCvp_minZ, double NPCvp_maxZ,
      matrix_4x4 *vmMatrix );
```

Rzuty wielokrotne. Indeks rzutu określony w czasie wysyłania odnosi się do specyficznego pola wizualizacji NPC opisującego, gdzie na ekranie (w biuletynie elektronicznym) ma się pojawić obraz struktury (zdjęcie). Tak jak można przypiąć kilka odbitek fotograficznych na tablicy, tak program użytkowy może podzielić ekran na kilka pól wizualizacji. Możliwość korzystania z wielu rzutów jest wygodna z wielu względów. Możemy równocześnie wyświetlić kilka różnych struktur w poszczególnych polach ekranu wysyłając do nich różne rzuty. Na rysunku 7.5a



Rys. 7.5. Wiele rzutów na ekranie: a) schemat tablicy rzutów; każda pozycja wskazuje na listę korzeni wysłanych do tego rzutu; b) wynikowy obraz. Pola wizualizacji są zaznaczone liniami przerywanymi i nie pokazują całego obrazu. Cyfry w kółeczkach pokazują pola wizualizacji i związane z nimi wskaźniki

przedstawiono schematyczną reprezentację tablicy rzutów, przy czym pokazano tylko wskazania do list sieci struktury wysyłanych do każdego rzutu. Można zauważyć, że jest jeden rzut pokazujący scenę ulicy; są również trzy oddzielne rzuty robota. Struktura robota była wysłana trzy razy, za każdym razem z innym indeksem rzutu. Na rysunku 7.5b pokazano wynikowy obraz ekranu. Przy wielokrotnych rzutach robota zmienia się specyfikacja pola wizualizacji i specyfikacja bryły widzenia.

Poprzedni scenariusz zakładał, że każdy rzut ma najwyżej jedną wysłaną strukturę; w rzeczywistości jednak do jednego rzutu widoku może być wysłana dowolna liczba korzeni. Dlatego możemy wyświetlić różne struktury korzeni w jednym jednorodnym obszarze wysyłając je razem. W tym przypadku nasza umowna kamera zrobi jedno złożone zdjęcie sceny, która zawiera wiele różnych obiektów.

Inną właściwością pól wizualizacji jest to, że – w przeciwieństwie do rzeczywistych zdjęć i okien programu zarządzania oknami – są one przezroczyste. W praktyce wiele zastosowań dzieli pole wizualizacji na rozłączne obszary po to, żeby uniknąć nakładania; jednak i nakładanie można czasami uznać za zaletę. Na przykład możemy złożyć dwa różne obrazy utworzone przez różne przekształcenia związane z rzutowaniem albo pokazywać obiekty zdefiniowane w różnych jednostkach miary; przy budowaniu powiększonego schematu części silnika moglibyśmy wstawić mały obrazek pokazujący całą maszynę (dla kontekstu) przesłaniający obraz zbliżenia. (Moglibyśmy to zrobić wybierając obszar zbliżenia po prostu jako tło.)

W celu zregenerowania sceny SPHIGS wyświetla wcześniej wysłane sieci przeszukując korzenie wysłane do każdego rzutu w tablicy rzutów, w rosnącym porządku indeksów rzutów, zaczynając od rzutu 0; obrazy obiektów wysłanych do rzutu N mają priorytet wyświetlania większy w stosunku do obiektów obrazu wysłanych do rzutu o indeksie mniejszym niż N i dlatego mogą je potencjalnie przykrywać. Oczywiście takie uporządkowanie jest ważne tylko wówczas, gdy pola wizualizacji rzeczywiście nakładają się. (Zauważmy, że ten trywialny system priorytetów widzenia jest mniej złożony niż w PHIGS, gdzie jest możliwe przypisanie przez program użytkowy bezpośrednich priorytetów rzutowania.)

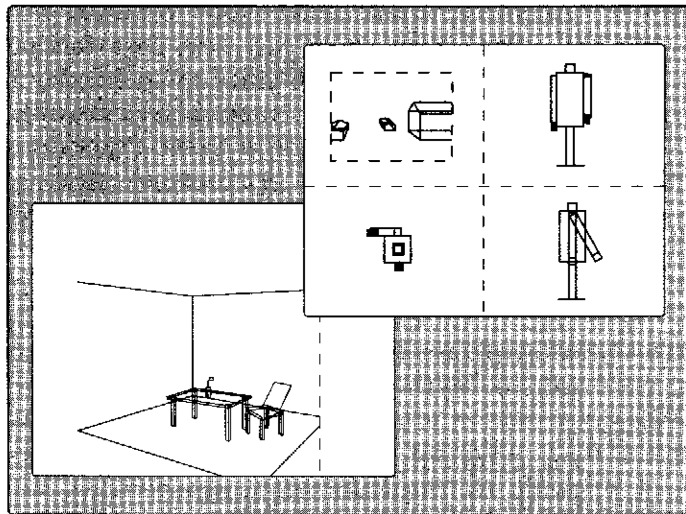
Zauważmy, że program użytkowy może wytwarzać wiele niezależnych układów współrzędnych WC i może korzystać z dowolnej liczby potrzebnych jednostek miar. Na przykład na rys. 7.5 struktura ulicy jest zdefiniowana w układzie WC, w którym każdy przyrost na osi reprezentuje 10 jardów, podczas gdy robot jest określony w całkowicie niezależnym układzie z jednostkami wyrażonymi w centymetrach. Chociaż każda struktura korzenia jest modelowana we własnym układzie współ-

rzędnych WC, dla urządzenia jest tylko jeden układ NPC, wspólny dla wszystkich wysłanych struktur – ta przestrzeń jest abstrakcją urządzenia wyświetlającego.

7.3.5. Zastosowania graficzne wykorzystujące ekran na zasadzie zarządzania oknami

Na początku lat siedemdziesiątych, gdy były projektowane pierwsze standardowe interakcyjne pakiety graficzne, w danej chwili był wykonywany tylko jeden program graficzny i zajmował cały ekran. Projekt standardu PHIGS zaczął się w końcu lat siedemdziesiątych, gdy ten tryb pracy wciąż dominował i programy zarządzania oknami nie były ogólnie dostępne. Wobec tego sześcian jednostkowy przestrzeni NPC był tradycyjnie odwzorowywany na ekran w całości.

Współczesne stacje graficzne z wielozadaniowymi systemami operacyjnymi umożliwiają równoczesne wykonywanie wielu graficznych programów użytkowych, na zasadzie podziału zasobów stacji, ekranu i urządzeń wejściowych pod kontrolą programu zarządzania oknami. W tym środowisku każdy program użytkowy jest przypisany do swojego własnego okna, które działa jak wirtualny ekran. Użytkownik może zmieniać wielkość okien i przesuwać je wywołując funkcje programu zarządzania oknami. Podstawową zaletą jest to, że każdy program użytkowy może działać tak, jak gdyby kontrolował cały ekran; program



Rys. 7.6. Dwa zastosowania SPHIGS, każde wykonywane w swoim oknie programu zarządzania oknami. Linie przerywane ilustrują granice pól wizualizacji

nie musi wiedzieć, że jego ekran jest tylko fragmentem rzeczywistego ekranu urządzenia wyświetlającego. Dlatego program użytkowy SPHIGS nie musi być modyfikowany dla środowiska programu zarządzania oknami; pakiet i program zarządzania oknami współpracują przy odwzorowywaniu przestrzeni NPC do przypisanego okna, a nie do całego ekranu.

Na rysunku 7.6 pokazano dwa programy użytkowe SPHIGS wykonywane równolegle na stacji graficznej. Ponieważ SPHIGS odwzorowuje przestrzeń NPC na największy kwadrat mieszczący się w oknie programu zarządzania oknami, niektóre części niekwadratowego okna nie są dostępne dla programu użytkowego SPHIGS, tak jak to pokazano w oknie SPHIGS ze sceną zawierającą stół i krzesło.

7.4. Przekształcenia modelowania

W punkcie 7.3.2 podano fragment kodu w języku C, który tworzył prostą strukturę modelującą dom. Dla uproszczenia umieściliśmy jeden wierzchołek domu w początku układu współrzędnych, a boki domu ustawiliśmy równolegle do osi układu współrzędnych i wybraliśmy wymiary będące wielokrotnościami całych jednostek. Będziemy mówili, że obiekt zdefiniowany w początku układu współrzędnych i ustawiony równolegle do osi układu jest obiektem znormalizowanym; nie tylko jest łatwiej zdefiniować (określić współrzędne wierzchołków) znormalizowany obiekt niż obiekt zajmujący dowolną pozycję w układzie współrzędnych, ale również łatwiej jest manipulować geometrią znormalizowanego obiektu w celu zmiany jego wielkości, orientacji albo położenia.

Powiedzmy, że chcemy, żeby nasz dom pojawił się w innym miejscu niż w pobliżu początku układu współrzędnych. Moglibyśmy oczywiście na nowo obliczyć same wierzchołki i utworzyć strukturę domu korzystając z tego samego kodu w języku C pokazanego w p. 7.3.2 (zmieniając jedynie współrzędne wierzchołków). Zamiast tego pokażemy metodę przekształcania normalizującego obiektu podstawowego, mającego na celu zmianę jego wymiarów albo położenia.

Jak pokazaliśmy w rozdz. 5, możemy dokonać przekształcenia takiego prymitywu jak wielokąt, mnożąc każdy wierzchołek reprezentowany przez wektor kolumnowy $[x\ y\ z\ 1]^T$ przez macierz przekształceń jednorodnych 4×4. Następujące funkcje tworzą takie macierze:

```
matrix_4x4 SPH_scale( double scaleX, double scaleY, double scaleZ );
matrix_4x4 SPH_rotateX( double angle );
matrix_4x4 SPH_rotateY( double angle );
matrix_4x4 SPH_rotateZ( double angle );
matrix_4x4 SPH_translate( double deltaX, double deltaY, double deltaZ );
```

Dla każdej z osi może być określony inny współczynnik skalowania i obiekt może być rozciągany albo ścieśniany niejednakowo. Dla obrotów kąt wyrażony w stopniach reprezentuje ruch w kierunku przeciwnym względem ruchu wskazówek zegara, wokół odpowiedniej osi głównej, dla obserwatora patrzącego wzdłuż osi od nieskończoności w kierunku początku układu współrzędnych.

Macierze mogą być używane do tworzenia elementu przekształcenia, który ma być umieszczony w strukturze. Następująca funkcja jest generatorem elementu:

```
void SPH_setLocalTransformation( matrix_4x4 matrix, int REPLACE / PRECONCATENATE
    / POSTCONCATENATE);
```

Użycie prefiksu *local* odnosi się tu do tego, jak SPHIGS wyświetla strukturę. Gdy SPHIGS przegląda strukturę, zapamiętuje macierz lokalną jako część informacji o stanie odnoszącym się tylko do przeglądanej struktury. Macierz lokalna jest przez domniemanie inicjalizowana jako macierz jednostkowa. Gdy zostaje napotkany element setLocalTransformation, macierz lokalna jest w jakiś sposób modyfikowana: albo jest zastępowana, albo zmieniana przez operację mnożenia, co zależy od parametru *mode*. Gdy w czasie przeglądania zostaje napotkany prymityw, każdy z jego wierzchołków jest przekształcany przez macierz lokalną, a potem przy wyświetlaniu jest poddawany przekształceniom rzutowania. (Jak zobaczymy dalej, hierarchia komplikuje tę regułę.)

Następujący kod tworzy strukturę zawierającą nasz dom w dowolnym położeniu i wysyła tę strukturę do wyświetlenia korzystając z domniemanego rzutu. Dom zachowuje swoje znormalizowane wymiary i orientację.

```
SPH_openStructure (HOUSE_STRUCT);
   SPH_setLocalTransformation (SPH_translate(...), REPLACE);
   SPH_polyhedron (...);      /* wierzchołki są tu znormalizowane jak poprzednio) */
SPH_closeStructure();
SPH_postRoot (HOUSE_STRUCT, 0);
```

Takie proste przekształcenia jak te są rzadkością. Na ogół chcemy nie tylko przekształcić obiekt, ale również wpłynąć na jego wielkość i orientację. Gdy trzeba kilku przekształceń prymitywów, program użytkowy buduje macierz lokalną składając kolejno poszczególne macierze przekształcenia dokładnie w takim porządku, w jakim mają być zastosowane. Ogólnie, znormalizowane obiekty składowe są skalowane, potem obracane i wreszcie przesuwane na odpowiednie pozycje; jak pokazano w rozdz. 5, taki porządek zapobiega niepożądanym przesunięciom albo pochylaniu.

Następujący kod tworzy i wysyła strukturę domu, który jest odsunięty od początku układu współrzędnych i jest obracany do położenia, w którym widzimy jego bok zamiast jego przodu:

```
SPH_openStructure (MOVED_HOUSE_STRUCT);
   SPH_setLocalTransformation (SPH_rotate(...), REPLACE);
   SPH_setLocalTransformation (SPH_translate(...), PRECONCATENATE);
   SPH_polyhedron (...);      /* wierzchołki są tu znormalizowane jak poprzednio) */
SPH_closeStructure();
SPH_postRoot (MOVED_HOUSE_STRUCT, 0);
```

Użycie trybu PRECONCATENATE dla macierzy przesunięcia zapewnia, że w wyniku wstępnego mnożenia uzyskuje się złożenie macierzy przesunięcia z macierzą obrotu i dlatego przesunięciu towarzyszy obrót. Wstępne mnożenie jest o wiele częściej wykorzystywane niż końcowe mnożenie, ponieważ odpowiada kolejności poszczególnych elementów przekształcenia. SPHIGS wykonuje skalowanie i obroty względem głównych osi, a więc jeżeli programista chce wykonać przekształcenie względem dowolnej osi, to musi najpierw wygenerować macierze potrzebne do odwzorowania tej osi na jedną z głównych osi, potem wykonać przekształcenie i z kolei odwrotne odwzorowanie, tak jak to pokazano w rozdz. 5.

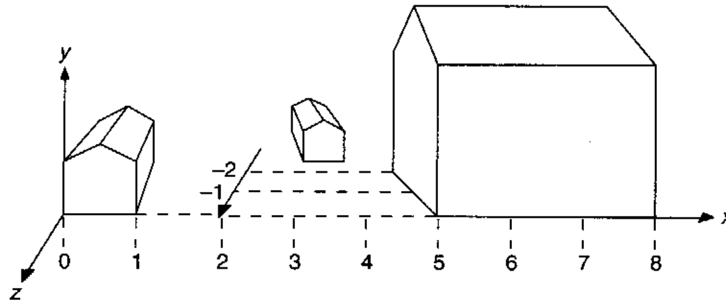
Składanie elementów przekształcenia jest wykonywane przez SPHIGS w *czasie przeglądania*; tak więc za każdym razem, gdy ekran jest regenerowany, trzeba wykonać złożenie. Alternatywna metoda określania ciągłej sekwencji przekształceń zwiększa wydajność procesu wyświetlania-przeszukiwania: zamiast dla każdego z nich tworzyć element struktury, składamy je sami w czasie specyfikowania i generujemy element przekształcenia. Następująca funkcja wykonuje mnożenie macierzy w czasie specyfikowania:

```
matrix_4x4 SPH_composeMatrix( matrix_4x4 mat1, matrix_4x4 mat2 );
```

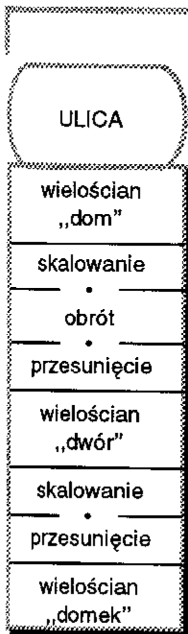
Wobec tego w poprzednim kodzie muszą być zastąpione dwa elementy setLocalTransformation przez:

```
SPH_setLocalTransformation (
      SPH_composeMatrix(SPH_translate(...), SPH_rotate(...)), REPLACE);
```

Wadą tej metody jest to, że teraz nie można robić dynamicznej zmiany wielkości, orientacji albo pozycji prymitywu przez selektywne „edytowanie" potrzebnego składnika sekwencji elementów setLocalTransformation; całe złożenie musi być ponownie obliczone i wyspecyfikowane. Prosta reguła mówi, że ze względu na efektywność trzeba



Rys. 7.7a. Modelowanie ulicy z trzema domami. Rzut perspektywiczny



Rys. 7.7b. Struktura modelu ulica-dom

używać składania w czasie specyfikowania dopóty, dopóki indywidualne przekształcenia nie będą musiały być uaktualniane selektywnie, w którym to przypadku powinny być specyfikowane indywidualnie.

Utwórzmy strukturę ulicy, która zawiera trzy kopie naszego prostego domu, tak jak go pokazano po raz pierwszy na rys. 7.4. Na rysunku 7.7a pokazano rzut perspektywiczny domu z lewej strony, dwór z prawej strony i domek w środku. Dodaliśmy linie przerywane równoległe do osi x i znaczniki dla osi x w celu wskazania względnych pozycji domów; użyliśmy trybu wyświetlania SPHIGS, który pokazuje drutowe krawędzie wielościanu z usuniętymi niewidocznymi krawędziami (por. p. 7.8). Dom z lewej strony rysunku jest kopią naszego znormalizowanego domu bez przekształceń; pozostałe dwie kopie różnią się wielkością, orientacją i pozycją.

Bezpośrednia metoda tworzenia takiej struktury polega na trzykrotnym określeniu wielościanu znormalizowanego domu i późniejszym wykonaniu potrzebnych elementów przekształceń, tak jak to pokazano na schemacie struktury z rys. 7.7b. Pokazujemy blok kolejnych elementów przekształceń jako pojedynczą jednostkę; pierwszy element wykorzystuje tryb REPLACE, a wszystkie inne tryb PRECONCATENATION, z symbolem mnożenia (·) separującym je w celu wskazania składania. Kod generujący strukturę pokazano w programie 7.3.

Program 7.3 Kod użyty do wygenerowania rys. 7.7a

```
SPH_openStructure(STREET_STRUCT);
/* Najpierw definicja domu w standardowej postaci */
SPH_polyhedron(...);

/* Dwór jest skalowany razy 2 w kierunku osi x, 3 w kierunku osi y, 1 w kierunku osi z,
   obrócony o 90° wokół osi y i potem przesunięty. Zauważmy, że w konsekwencji jego
   lewy bok stał się widoczny z przodu i leży w płaszczyźnie (x, y) */
SPH_setLocalTransformation(SPH_scale(2.0, 3.0, 1.0), REPLACE);
SPH_setLocalTransformation(SPH_rotateY(90.0), PRECONCATENATE);
SPH_setLocalTransformation(SPH_translate(8.0, 0.0, 0.0), PRECONCATENATE);
SPH_polyhedron(...);
```

```
    /* Domek jest skalowany ze współczynnikiem 0.75 i przesuwany względem z oraz x. */
    SPH_setLocalTransformation(SPH_scale(0.75, 0.75, 0.75), REPLACE);
    SPH_setLocalTransformation(SPH_translate(3.5, 0.0, -2.5), PRECONCATENATE);
    SPH_polyhedron(...);
SPH_closeStructure();
SPH_postRoot(STREET_STRUCT, 0);
```

Możemy wyeliminować redundancyjną specyfikację wielościanu znormalizowanego domu definiując funkcję C wykonującą wywołanie generujące wielościan domu, tak jak to pokazano w pseudokodzie z rys. 7.4. Ponieważ dom jest zdefiniowany przez jedno wywołanie wielościanu, w tym przykładzie efektywność tej metody nie jest oczywista; gdyby jednak nasz dom był bardziej złożony i wymagał ciągu specyfikacji atrybutów i prymitywów, ta metoda w oczywisty sposób wymagałaby mniej kodu w języku C. Co więcej, tę metodę cechuje inna zaleta, a mianowicie modularyzacja: możemy zmienić kształt albo styl domu dokonując edycji funkcji House bez konieczności edycji kodu, który tworzy ulicę.

Funkcję taką jak House, która generuje sekwencję elementów definiujących znormalizowany element podstawowy i która może być wykorzystywana wielokrotnie z dowolnymi przekształceniami, nazywamy *funkcją szablonu*. Funkcja szablonu jest wygodna dla programisty i reprezentuje dobry styl programowania. Zauważmy jednak, że chociaż funkcja House dodaje poziom funkcjonalnej hierarchii do programu w języku C, to nie jest tworzona struktura hierarchii – model ulicy wciąż jest płaski. Sieć struktury tworzona przez kod programu 7.4 jest nierozróżnialna od tworzonej przez kod z programu 7.3. Nie ma oszczędności, jeśli chodzi o liczbę elementów wytwarzanych dla struktury.

Program 7.4 Wykorzystanie funkcji szablonu do modelowania ulicy

```
void House
{
    SPH_polyhedron(...);
}

main()
{
    SPH_openStructure(STREET_STRUCT);
        House();                    /* Pierwszy dom */
        ustawienie macierzy lokalnego przekształcenia;
        House();                    /*Dwór */
        ustawienie macierzy lokalnego przekształcenia;
        House();                    */ Domek */
    SPH_closeStructure();
    SPH_postRoot(STREET_STRUCT, 0);
}
```

Jedna zmiana, jaką moglibyśmy zrobić w naszej funkcji szablonu, to przyjęcie, że macierz przekształceń jest parametrem, który mógłby być używany do specyfikowania elementu setLocalTransformation. Chociaż w niektórych przypadkach przekazywanie parametrów przekształcenia mogłoby być wygodne, metodzie brak ogólności cechującej naszą oryginalną metodę polegającą na możliwości specyfikowania dowolnej liczby przekształceń przed wywołaniem szablonu.

7.5. Hierarchiczne sieci struktur

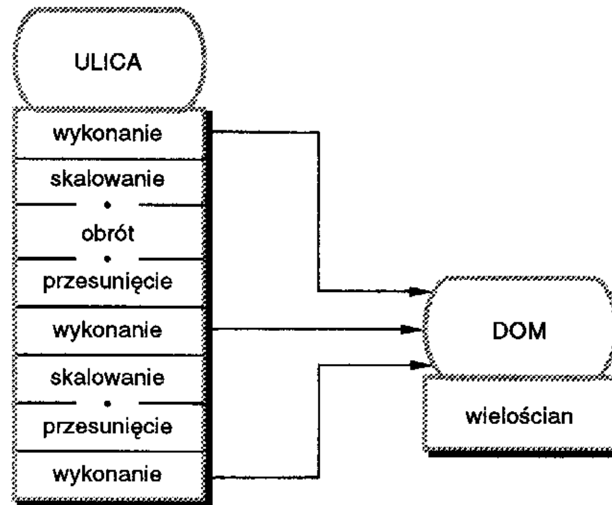
7.5.1. Hierarchia dwupoziomowa

Dotychczas zajmowaliśmy się trzema typami elementów struktur: prymitywami wyjściowymi, atrybutami wyglądu i przekształceniami. Teraz pokażemy, jak siła SPHIGS zależy w dużej mierze od struktury hierarchii, realizowanej za pomocą elementu, który wywołuje podstrukturę podczas przeglądania. Struktury hierarchii nie należy mylić z hierarchią procedury szablonu z poprzedniego punktu. Hierarchia procedury szablonu jest rozwiązywana w czasie specyfikacji, gdy ma miejsce edycja CSS i na bieżąco wytwarza elementy, a nie wywołania podstruktur. Dla kontrastu hierarchia struktury wynikająca z wywoływania podstruktur jest rozwiązywana, gdy CSS jest przeglądane przy wyświetlaniu – wykonanie elementu wywołania działa jako wywołanie podprogramu.

Element wykonania struktury, który wywołuje podstrukturę, jest tworzony przez

```
void SPH_executeStructure( int structureID );
```

Zastąpmy funkcję szablonu z poprzedniego punktu funkcją, która tworzy strukturę domu w CSS (por. program 7.5). Ta funkcja jest wywoływana dokładnie raz przez funkcję main i HOUSE_STRUCT nigdy nie jest wysyłane; wyświetlany wynik jest efektem wywołania jako podobiektu struktury opisującej ulicę. Zauważmy, ze jedyną różnicą w specyfikacji STREET_STRUCT jest dodanie wywołania do funkcji, która tworzy strukturę domu, i zastąpienia wywołania każdej funkcji szablonu przez generowanie elementu wykonania struktury. Chociaż wyświetlony obraz jest taki sam jak na rys. 7.7a, struktura sieci jest inna, tak jak to pokazano na rys. 7.8, na którym element wykonania struktury jest oznaczony strzałką.



Rys. 7.8. Sieć struktury pokazująca wywołanie podporządkowanej struktury

Program 7.5 Wykorzystanie struktury podporządkowanej do modelowania ulicy

```
void BuildStandardizedHouse
{
   SPH_openStructure(HOUSE_STRUCT);
   SPH_polyhedron(...);
   SPH_closeStructure;
}

main()
{
   BuildStandardizedHouse();
   SPH_openStructure(STREET_STRUCT);
      SPH_executeStructure(HOUSE_STRUCT);          /* Pierwszy dom */
      ustawienie macierzy lokalnego przekształcenia;
      SPH_executeStructure(HOUSE_STRUCT);          /* Dwór */
      ustawienie macierzy lokalnego przekształcenia;
      SPH_executeStructure(HOUSE_STRUCT);          /* Domek */
   SPH_closeStructure();
   SPH_postRoot(STREET_STRUCT);
}
```

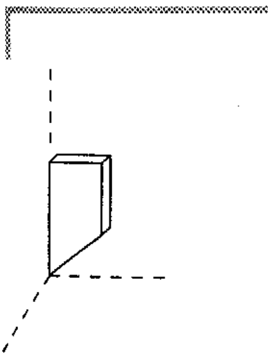
Wysłanie STREET_STRUCT informuje SPHIGS, że ma uaktualnić ekran przeglądając sieć struktury STREET_STRUCT; przeglądanie polega na chodzeniu po drzewie na zasadzie przeglądania zstępującego, dokładnie tak, jak wykonywana hierarchia funkcja/podprogram. W poprzednim przykładzie blok przeglądania inicjalizuje macierz

lokalną struktury ulicy jako macierz jednostkową, a potem wykonuje pierwsze odwołanie do podstruktury domu wykorzystując macierz lokalną struktury ulicy do każdego wierzchołka domu, tak jak gdyby wielościan domu był prymitywem w samej strukturze ulicy. Po powrocie z pierwszego odwołania jest ustawiana macierz lokalna jako potrzebne złożone przekształcenie i potem wykonuje się drugie odwołanie, stosując nową złożoną macierz do wierzchołków domu w celu utworzenia drugiej wersji domu. Po powrocie ponownie zmienia się lokalną macierz; nowa złożona macierz jest stosowana do wierzchołków domu w celu utworzenia trzeciej wersji domu.

O strukturze myślimy jak o niezależnym obiekcie, przy czym jego prymitywy są zdefiniowane w jego własnym zmiennopozycyjnym układzie współrzędnych modelowania (MCS); taki sposób rozumowania ułatwia budowanie znormalizowanych bloków podstawowych niskiego poziomu. Jak zauważyliśmy w p. 5.9, przekształcenie odwzorowuje wierzchołki w jednym układzie współrzędnych na inny; tutaj SPHIGS wykorzystuje lokalną macierz struktury *S* do przekształcenia prymitywów podstruktur do własnego układu MCS struktury *S*.

7.5.2. Prosta hierarchia trójpoziomowa

Prostym przykładem hierarchii trójpoziomowej jest rozbudowany dom z naszego przykładu ulicy. Nowy dom składa się z oryginalnego znormalizowanego domu (o nowej nazwie SIMPLE_HOUSE_STRUCT) i komina w odpowiedniej skali umieszczonego w odpowiedni sposób na górze domu. Moglibyśmy zweryfikować strukturę domu dodając ścianki komina bezpośrednio do oryginalnego wielościanu albo dodając drugi wielościan do struktury, ale wybieramy tutaj rozwiązanie prowadzące do struktury trójpoziomowej, polegające na zdekomponowaniu domu na dwa podobiekty. Zaletą tej modularyzacji jest to, że możemy zdefiniować komin w standardowy sposób (w początku układu współrzędnych, jednostkowa wielkość) w jego własnym układzie MCS (tak jak pokazano na rys. 7.9a) i potem za pomocą skalowania i przesunięcia umieścić go na dachu w układzie MCS domu. Gdybyśmy mieli zdefiniować komin tak, żeby pasował dokładnie do dachu i odwzorować go na układ MCS dachu bez skalowania, musielibyśmy wykonać masę obliczeń w celu bezpośredniego obliczenia wierzchołków. Korzystając natomiast z modułowości po prostu określamy znormalizowany komin tak, żeby jego dolna ściana miała takie samo nachylenie jak dach; przy zachowaniu tego warunku można stosować jednolite skalowanie i dowolne przesunięcia.

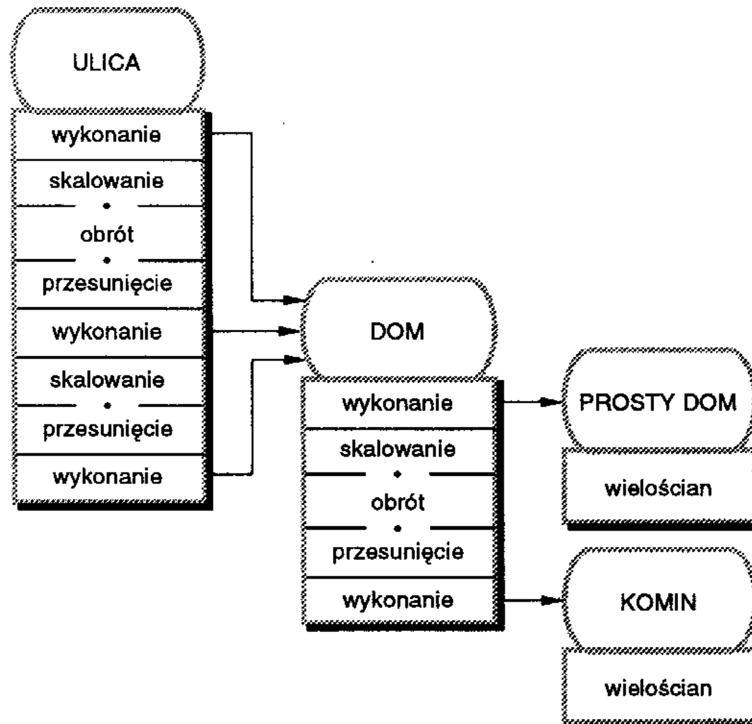


Rys. 7.9a. Standaryzowany komin dla hierarchii trójpoziomowej

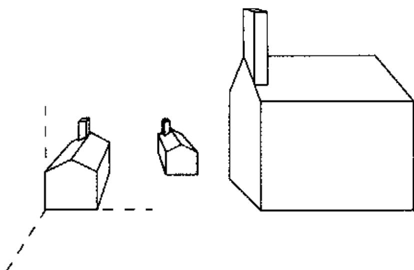
Zmodyfikowaną strukturę domu tworzymy w następujący sposób:

```
SPH_openStructure (HOUSE_STRUCT);
   SPH_executeStructure (SIMPLE_HOUSE_STRUCT);
   ustawienie lokalnej macierzy dla skalowania/przesunięcia znormalizowanego
   komina na dach prostego domu;
   SPH_executeStructure (CHIMNEY_STRUCT);
SPH_closeStructure();
```

Co się stanie, gdy ten dom – dwupoziomowy obiekt – zostanie poddany przekształceniom przez strukturę ulicy w celu otrzymania hierarchii trójpoziomowej pokazanej na rys. 7.9b. Ponieważ SPHIGS przekształca ojca na zasadzie przekształcenia końcowych elementów składowych i podstruktur, mamy pewność, że dwa prymitywy składowe (prosty dom i komin) są przekształcane razem jako jeden obiekt (rys.7.9c). Kluczową sprawą jest to, że specyfikacja struktury ulicy nie musi być w ogóle zmieniona. Dlatego projektant struktury ulicy nie musi się zajmować wewnętrznymi szczegółami konstrukcji domu albo jego późniejszymi edycjami – jest to czarna skrzynka.



Rys. 7.9b. Sieć struktury dla hierarchii trójpoziomowej



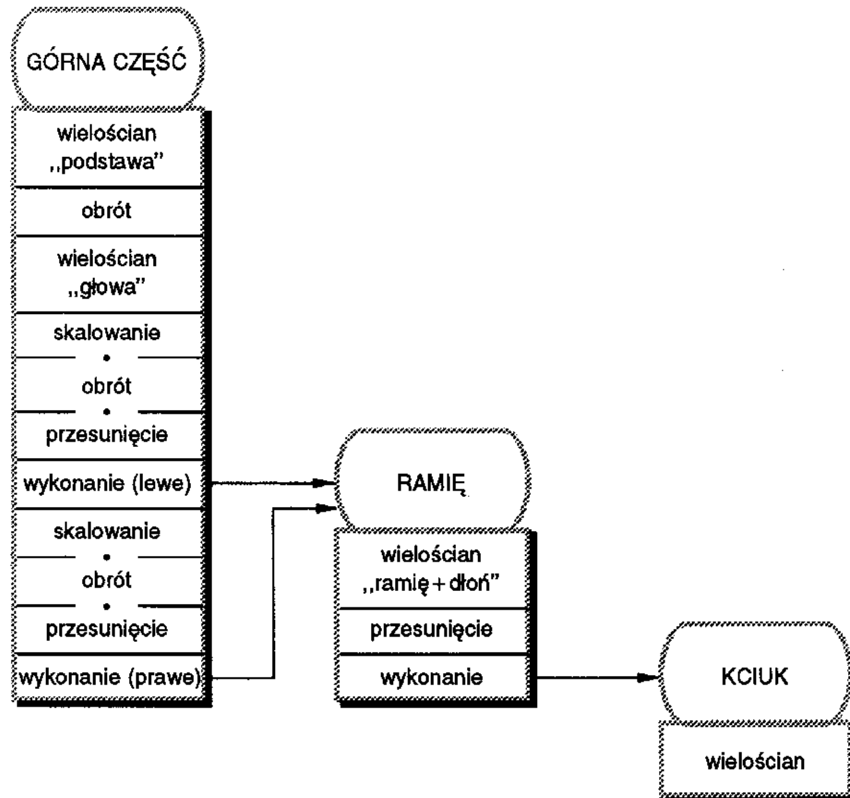
Rys. 7.9c. Obraz otrzymany w wyniku przekształcenia hierarchii trójpoziomowej

7.5.3. Konstruowanie robota metodą wstępującą

Zajmiemy się teraz bardziej interesującym przykładem – prostym robotem; wykorzystamy tu kluczowe idee modelowania za pomocą hierarchii struktur i kolejnych edycji przekształceń w celu uzyskania dynamiki ruchu. Złożony obiekt albo hierarchia systemu są zazwyczaj opisywane metodą zstępującą. Na przykład budynek wydziału informatyki jest złożony z pięter, na których są biura i laboratoria, które są złożone z mebli, sprzętu itd. Przypomnijmy, że nasz prosty robot jest złożony z tułowia, głowy i dwóch identycznych ramion, z których każde ma dłoń i przesuwający się kciuk.

Nawet jeżeli projektujemy metodą zstępującą, to często korzystamy z metody wstępującej, określając bloki elementarne do wykorzystania przy określaniu podzespołów wyższego poziomu itd., w celu utworzenia hierarchii podzespołów. Konstruując robota definiujemy podzespół kciuka dla ramienia robota, potem samo ramię i potem dołączamy dwie kopie podzespołu ramienia do tułowia itd., jak na rys. 7.2, na którym pokazano hierarchię symbolicznych części, i na dokładniejszym schemacie sieci struktury tułowia z rys. 7.10.

Przyjrzyjmy się dokładniej wstępującemu procesowi konstruowania, żeby zobaczyć, jaka jest geometria i jakie są wykorzystywane przekształcenia. Sensowne jest projektowanie ramienia i kciuka w tych samych jednostkach miary, tak żeby je łatwo można było do siebie dopasować. Określamy strukturę kciuka w znormalizowanej pozycji w jego własnym układzie MCS, w którym jest on skierowany wzdłuż osi y (rys. 7.11a). Struktura ramienia jest określana przy wykorzystaniu tej samej jednostki co w przypadku kciuka; ramię składa się z wielościanu ramię+dłoń (znormalizowana postać skierowana wzdłuż osi y, jak na rys. 7.11b) i przesuniętej struktury kciuka. Element przesuniecia poprzedzający odwołanie do kciuka jest odpowiedzialny za przesunięcie uchwytu ze znormalizowanej pozycji w początku układu współrzędnych na właściwe miejsce na przegubie ramienia, tak jak na rys. 7.11c.

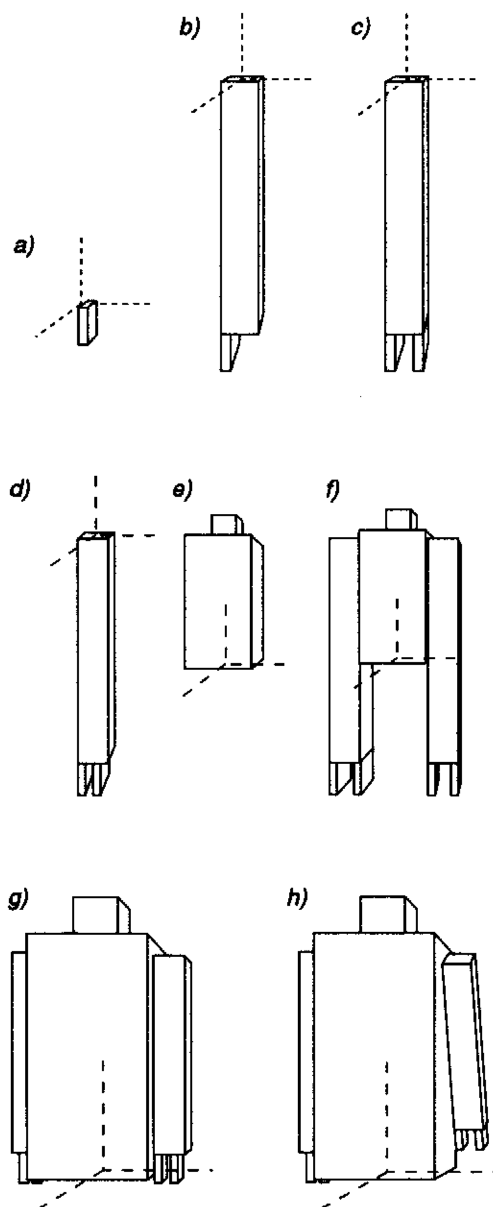


Rys. 7.10. Hierarchia struktury górnej części robota

Sieć ramię-kciuk jest hierarchią dwupoziomową podobną do przykładu ulica-dom. Dokonując edycji elementu przesunięcia w strukturze ramienia możemy przesuwać kciuk wzdłuż przegubu ramienia (rys. 7.11d)[4].

Następnie budujemy tułów. Ponieważ chcemy móc obracać głową, najpierw określamy wielościan tułowia, potem obrót i następnie wielościan głowy (rys. 7.11e). Nasz następny krok polega na dwukrotnym wywołaniu przez strukturę tułowia struktury ramienia. Jakie przekształcenia powinny poprzedzać te odwołania? Jeżeli naszym jedynym celem jest poprawne umieszczenie (przesunięcie) każdego ramienia w układzie MCS tułowia, to możemy otrzymać obraz taki jak na rys. 7.11f, na którym ramię i tułów były zaprojektowane w różnych skalach. Z tym

[4] Uważny czytelnik może się dziwić, jak kciuk może się przesuwać w tym modelu, ponieważ w istocie nie jest w żaden sposób doczepiony do dłoni ramienia. W rzeczywistości żaden z elementów naszego robota nie jest doczepiony do innego za pomocą obiektów reprezentujących złącza.



Rys. 7.11. Konstruowanie górnej części robota: a) uchwyt; b) ramię plus dłoń; c) kompletne ramię; d) kompletne ramię z przesuniętym uchwytem; e) tułów i głowa; f) górna część ze źle zwymiarowanymi ramionami; g) poprawione ramiona; h) obrócone lewe ramię

można sobie łatwo poradzić: przed przesunięciem możemy dodać skalowanie (rys. 7.11g). Jednak skalowanie i przesunięcie nie wystarczą, jeżeli zależy nam na ramionach, które mogą się obracać wokół osi, która

łączy oba ramiona (chodzi o ruch taki, jaki wykonuje ramię maszerującego żołnierza); z tego względu przed przesunięciem umieszczamy w strukturze element obrotu. Zakończyliśmy naszą definicję struktury tułowia; połączenie całego robota zostawiamy jako zadanie (zad. 7.1). Na rysunku 7.11h pokazano tułów wyglądający tak jak wówczas, gdy element obrotu lewego ramienia jest różny od zera.

Ponieważ każde odwołanie do ramienia jest poprzedzone niezależną sekwencją przekształceń, można niezależnie kontrolować ruch każdego ramienia. Jedno ramię może być ustawione swobodnie, drugie zaś waha się (rys. 7.11h). Różnica w przekształceniach rozróżnia lewe ramię od prawego. Zauważmy jednak, że ruchomy kciuk jest nie tylko po tej samej stronie, ale również w tej samej odległości od dłoni zarówno w lewym, jak i w prawym ramieniu, ponieważ jest częścią wewnętrznej definicji ramienia. (W rzeczywistości, gdy program użytkowy po prostu zmienia element przesunięcia w strukturze ramienia, wszystkie kciuki we wszystkich kopiach robota nagle poruszają się!). Musimy obrócić jedno z ramion o 180° wokół osi y po to, żeby ramiona były symetryczne. Struktura odwołania do ramienia może sterować wielkością ramienia, jego orientacją i pozycją jako całością i nie może zmienić wewnętrznej konstrukcji ramienia. Jak powiedzieliśmy wcześniej podstruktura jest w zasadzie czarną skrzynką; wywołujący musi wiedzieć, jak geometria jest określona przez podstrukturę, ale nie musi wiedzieć, jak podstruktura była utworzona, i nie może wpływać na żaden element wewnętrzny podstruktury.

W sumie, przy specyfikacji dowolnej struktury zajmujemy się tylko prymitywami i podstrukturami niskiego poziomu włączonymi w strukturę, przekształceniami modelowania, jakie powinny być użyte do pozycjonowania części składowych, i atrybutami, jakie powinny być użyte, żeby wpływać na ich wygląd. Nie musimy i nie możemy zajmować się wewnętrznymi elementami struktur niskiego poziomu. Co więcej, projektujemy elementy bez zwracania uwagi na to, jak będą używane przez odwołujące się struktury, ponieważ do uzyskania potrzebnych wymiarów, orientacji i pozycji mogą być wykorzystane przekształcenia. W praktyce jest pomocne normalizowanie elementu w jego lokalnym układzie MCS, tak żeby można było łatwo skalować i obracać wokół głównych osi.

Trzeba zwrócić uwagę na dwa dodatkowe punkty. Po pierwsze, programista nie musi projektować ściśle według metody zstępującej i implementować według czystej metody wstępującej; w strukturze hierarchii można użyć *ślepych podstruktur*. Ślepą podstrukturą może być pusta podstruktura; w istocie SPHIGS dopuszcza, żeby struktura wykonywała podstrukturę, która nie jest (jeszcze) utworzona; w takim przypadku SPHIGS automatycznie tworzy odpowiednią podstrukturę i inicjalizuje ją jako pustą. W niektórych przypadkach dobrze jest, żeby pusta podstruktura była prostym obiektem (na przykład prostopadłoś-

cianem) o w przybliżeniu tej samej wielkości co bardziej złożona wersja, która będzie określona później. Ta metoda umożliwia przeszukiwanie zstępujące złożonej hierarchii struktury, zanim wszystkie elementy będą w pełni określone. Po drugie, nie porównaliśmy hierarchii struktury z hierarchią funkcji-szblonu, co jest omawiane w p. 7.15 pracy [FOLE90]. Po prostu mówimy, że hierarchia struktury działa dobrze, jeżeli potrzebujemy wielu kopii podstawowych elementów i chcemy sterować (w ograniczonym zakresie) wyglądem i położeniem każdej kopii, a nie jej wewnętrzną definicją.

7.5.4. Interakcyjne programy modelowania

Interakcyjne programy konstruowania i modelowania 3D ułatwiają tworzenie hierarchii obiektów dzięki wyżej opisanemu procesowi łączenia wstępującego. Większość takich programów oferuje paletę ikon reprezentujących podstawowy zestaw elementów składowych dostępnych w programie. Jeżeli program rysujący jest przystosowany do określonego zastosowania, to dotyczy to również elementów podstawowych; w przeciwnym przypadku są to takie popularne elementy jak łamane, wielokąty, sześciany, prostopadłościany, walce i kule. Użytkownik może wybrać ikonę w celu utworzenia kopii odpowiedniego elementu składowego i może potem określić przekształcenia, jakie w stosunku do tej kopii należy wykonać. Taka specyfikacja jest na ogół wykonywana przez urządzenia wejściowe, np. myszka czy tarcze sterujące, które umożliwiają użytkownikowi eksperymentowanie i dobieranie wielkości, orientacji i pozycji dopóty, dopóki nie uzna on, że element wygląda dobrze. Ponieważ jest trudno sprawdzić zależności przestrzenne w rzutach 2D scen 3D, bardziej zaawansowane metody interakcyjne wykorzystują siatki 3D, skale numeryczne, potencjometry, różnego rodzaju suwaki itd. (por. p. 8.2.5). Niektóre programy konstrukcyjne umożliwiają użytkownikowi łączenie kopii podstawowych elementów graficznych w celu tworzenia elementu wyższego poziomu, który jest potem dodawany do palety dostępnych elementów, i które mogą być wykorzystywane przy tworzeniu obiektów wyższego poziomu.

7.6. Składanie macierzy przy przeglądaniu w celu wyświetlenia

Dotychczas dyskutowaliśmy, jak programista konstruuje model, korzystając z projektu typu zstępującego i (mniej lub więcej) implementacji wstępującej. Niezależnie od tego, jak model został skonstruowany,

SPHIGS wyświetla model wykonując przeglądanie zstępujące DAG-u o korzeniu w wysłanej strukturze. W czasie przeglądania SPHIGS przetwarza całą geometrię określoną przez wiele poziomów przekształceń i odwołań. Zauważmy, że w czasie przeglądania zstępującego, gdy struktura A korzenia wywołuje strukturę B, która z kolei wywołuje strukturę C, wówczas jest to równoznaczne ze stwierdzeniem, że struktura B była skonstruowana metodą wstępującą w wyniku przekształcania prymitywów określonych w układzie MCS struktury C w układ MCS struktury B i że A była potem skonstruowana podobnie w wyniku przekształcenia prymitywów struktury B (włączając w to dowolny określony przez wywołanie C) w układ MCS struktury A. Ostateczny wynik jest taki, że prymitywy C były przekształcone dwa razy: pierwszy z MCS_C w MCS_B i potem z MCS_B w MCS_A.

Korzystając z notacji podanej w p. 5.9 oznaczmy przez $M_{B \leftarrow C}$ wartość lokalnej macierzy dla struktury B, która w czasie wywołania C odwzorowuje wierzchołki w MCS_C na ich poprawnie przekształcone pozycje w MCS_B. Żeby odwzorować wierzchołek z MCS_C na MCS_B, piszemy $V^{(B)} = M_{B \leftarrow C} \cdot V^{(C)}$ (przy czym $V^{(H)}$ oznacza wektor reprezentujący wierzchołek, którego współrzędne są wyrażone w układzie współrzędnych H) i podobnie $V^{(A)} = M_{A \leftarrow B} \cdot V^{(B)}$. W celu odwzorowania procesu konstrukcji wstępującej program przeglądania musi kolejno stosować przekształcenia, które odwzorowują wierzchołki z C na B i potem z B na A:

$$V^{(A)} = M_{A \leftarrow B} \cdot V^{(B)} = M_{A \leftarrow B} \cdot (M_{B \leftarrow C} \cdot V^{(C)}) \tag{7.1}$$

Dzięki asocjacyjności macierzy $V^{(A)} = (M_{A \leftarrow B} \cdot M_{B \leftarrow C}) \cdot V^{(C)}$. Stąd program przeglądania po prostu składa dwie lokalne macierze i stosuje wynikową macierz do każdego z wierzchołków C.

Korzystając z notacji dla drzew przyjmijmy, że korzeń jest na poziomie 1 i że kolejni potomkowie są na poziomach 2, 3, 4, ... Wtedy na zasadzie indukcji dla dowolnej konstrukcji na poziomie j ($j > 4$), możemy przekształcić wierzchołek $V^{(j)}$ w układzie MCS struktury w wierzchołek $V^{(1)}$ w układzie współrzędnych korzenia w następujący sposób:

$$V^{(1)} = (M_{1 \leftarrow 2} \cdot M_{2 \leftarrow 3} \cdot \ldots \cdot M_{(j-1) \leftarrow j}) \cdot V^{(j)} \tag{7.2}$$

Ponieważ SPHIGS umożliwia przekształcanie prymitywów w lokalnym MCS za pomocą lokalnej macierzy, współrzędne wierzchołka $V^{(j)}$ otrzymamy po pomnożeniu współrzędnych prymitywu przez lokalną macierz

$$V^{(j)} = M^{(j)} \cdot V^{(\text{prim})} \tag{7.3}$$

W czasie przeglądania struktury lokalną macierz oznaczamy jako $M^{(j)}$, żeby pokazać, że ta macierz jest używana do przekształcania prymitywów w układ MCS struktury poziomu j. Jeżeli struktura z kolei wywołuje podprogram, to sposób wykorzystania macierzy zmienia się; jest ona wtedy wykorzystywana do przekształcenia wywołanej struktury na poziomie $j+1$ w układ MCS poziomu j; oznaczamy to pisząc $M_{j \leftarrow (j+1)}$. To nie oznacza, że zmienia się wartość macierzy – zmienia się tylko jej wykorzystanie.

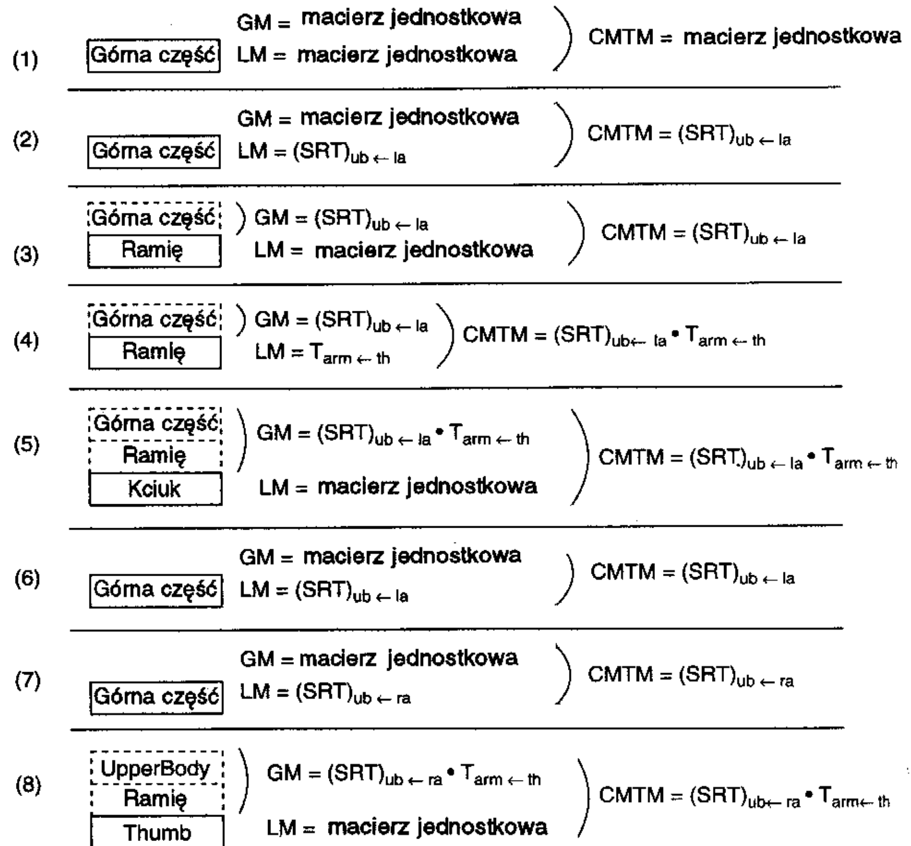
Łącząc równania (7.2) oraz (7.3) i korzystając z właściwości asocjacji otrzymujemy

$$V^{(1)} = (M_{1 \leftarrow 2} \cdot \ldots \cdot M_{(j-1) \leftarrow j} \cdot M^{(j)}) \cdot V^{(\text{prim})} \tag{7.4}$$

Wobec tego, aby dokonać przekształcenia prymitywu na poziomie j hierarchii w układ MCS korzenia (jest to przestrzeń współrzędnych świata), wszystko, co musimy zrobić, to zastosować złożenie bieżących wartości lokalnych macierzy dla każdej struktury od korzenia w dół do struktury, w której jest określony prymityw. To złożenie lokalnych macierzy – czynnik w nawiasach w równaniu (7.4) – jest nazywane *złożoną macierzą przekształcenia modelowania* (CMTM). Gdy stan przeglądania jest taki, że są wykonywane elementy struktury poziomu j, CMTM jest złożeniem j macierzy. Tylko ostatnia z tych macierzy ($M^{(j)}$) może być zmieniona przez strukturę, ponieważ struktura może modyfikować tylko swoją lokalną macierz. Tak więc, jeżeli struktura jest aktywna, pierwsze $j-1$ macierzy w liście CMTM jest stałych. Złożeniem tych $j-1$ macierzy jest *globalna macierz* (GM) – czynnik w nawiasach w równaniu (7.2) – dla wykonywanej struktury poziomu j. Dla SPHIGS jest dogodnie utrzymywać GM w czasie przeglądania struktury; gdy element setLocalTransformation modyfikuje lokalną macierz (LM), SPHIGS może łatwo obliczyć nowe CMTM na zasadzie złożenia końcowego nowej lokalnej macierzy z GM.

Możemy teraz podsumować algorytm przeglądania. SPHIGS wykonuje przeglądanie zstępujące, zapamiętując macierze CMTM, GM i LM tuż przed wywołaniem dowolnej struktury; potem inicjalizuje podstruktury GM i CMTM do dziedziczonej macierzy CMTM i jej LM na macierz jednostkową. Macierz CMTM jest stosowana do wierzchołków i jest uaktualniana w wyniku zmian w LM. Wreszcie gdy układ przeglądania wraca, odtwarza CMTM, GM i LM macierzystej struktury i kontynuuje przeglądanie. Ze względu na zapamiętywanie i odtwarzanie macierzy rodzice oddziałują na swoje potomstwo, ale nie odwrotnie.

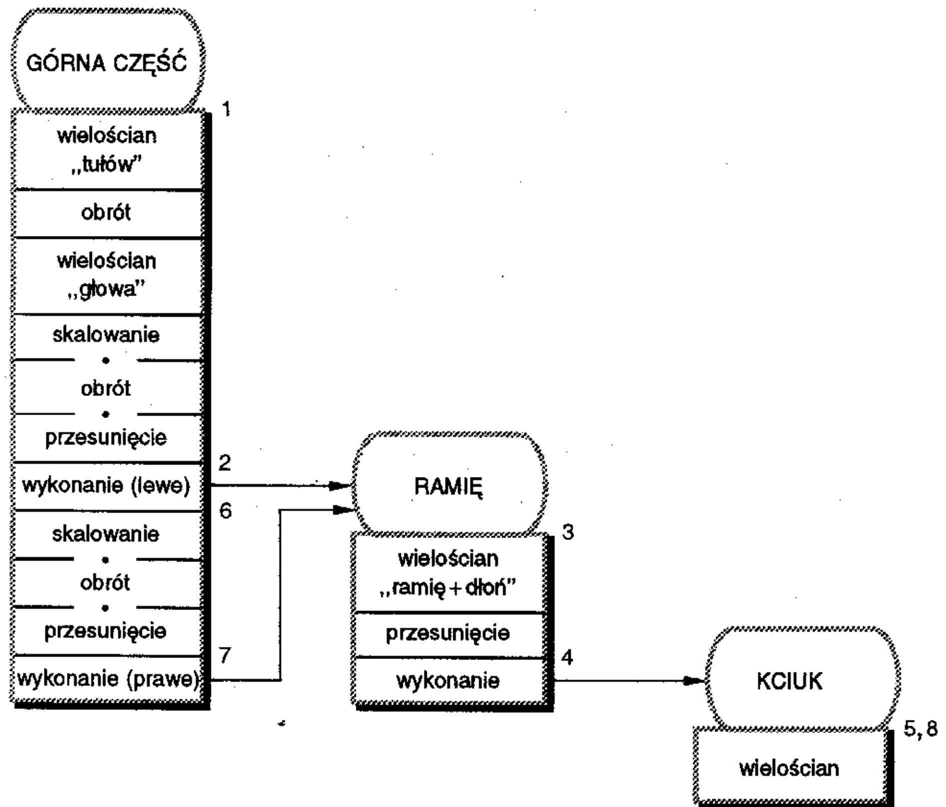
Przyjrzyjmy się, jak SPHIGS przegląda trójpoziomową hierarchię z rys. 7.10 (górna część – ramię-kciuk). Przyjęliśmy, że korzeniem jest



Rys. 7.12a. Stany stosu dla hierarchii trójpoziomowej

struktura UPPER_BODY. Na rysunku 7.12a pokazano sekwencję stanów przeglądania; pokazane stany odpowiadają punktom zaznaczonym numerami w schemacie sieci struktury z rys. 7.12b.

Stan przeglądania jest utrzymywany za pomocą stosu pokazanego na rys. 7.12a rosnącego w dół; bieżąca aktywna struktura jest w ramkach prostokątnych i jej przodek w ramce z linii przerywanej. Wartości trzech macierzy stanu dla bieżącej aktywnej struktury są pokazane z prawej strony schematu stosu. Łuki pokazują zakres przekształcenia: łuk GM pokazuje, ze przodek struktury ma udział w GM, a łuk CMTM pokazuje, że CMTM jest iloczynem GM i LM. Przypomnijmy, że w każdej grupie przekształceń pierwsze jest w trybie REPLACE, a reszta w trybie PRECONCATENATE. Dlatego pierwsza operacja obrotu w strukturze odnosi się tylko do głowy, ponieważ będzie ona zastąpiona (tryb REPLACE) przez pierwszą operację skalowania, która będzie dotyczyła lewego ramienia.



Rys. 7.12b. Opisana sieć struktury hierarchii trójpoziomowej

W punkcie 1 na rysunku 7.12b program przeglądania ma wykonać pierwszy element korzenia. Ponieważ korzeń nie ma przodków, jego GM jest macierzą jednostkową; podobnie LM jest macierzą jednostkową, tak jak w przypadku, gdy zaczyna się wykonanie struktury. W punkcie 2 macierz LM jest złożną macierzą dla trzech przekształceń (skalowanie (S), obrót (R), przesunięcie (T)). Teraz CMTM jest uaktualniana jako macierz będąca wynikiem mnożenia macierzy jednostkowej GM i złożonej macierzy SRT; teraz macierz CMTM może być użyta do przekształcenia podobiektu ramienia w układ MCS górnej części, tak żeby uzyskać lewe ramię za pomocą przekształceń $(STR)_{ub \leftarrow la}$. Następnie w punkcie 3 program przeglądania ma wykonać pierwszy element struktury ramienia. Macierz GM do wykonania ramienia jest, tak jak należałoby się spodziewać dla kopii poziomu 2, macierzystą macierzą LM w punkcie wywołania.

W punkcie 4 macierz LM ramienia jest przygotowana do umieszczenia kciuka w ramieniu ($T_{arm \leftarrow th}$) i macierz CMTM jest uaktualniona

jako iloczyn GM i LM. Ta macierz CMTM poziomu 2 staje się macierzą GM dla kopii kciuka poziomu 3 (punkt 5). Ponieważ macierz LM kciuka jest macierzą jednostkową, macierz CMTM kciuka daje pożądany efekt przekształcenia współrzędnych kciuka najpierw we współrzędne ramienia, a potem we współrzędne górnej części. W punkcie 6 program przeglądania wrócił z wywołań do kciuka i ramienia z powrotem do górnej części. Macierze dla struktury górnej części są takie jak przed wywołaniem, ponieważ podrzędne struktury nie mogą zmienić lokalnej macierzy. W punkcie 7 macierz LM górnej części jest zastąpiona nową macierzą złożoną dla prawego ramienia. Gdy zejdziemy do struktury kciuka dla prawego ramienia (punkt 8), macierz CMTM jest prawie identyczna jak w punkcie 5; jedyna różnica jest w macierzy poziomu 2, która przesuwa ramię do położenia w górnej części.

W celu animowania złożonego obiektu takiego jak robot, musimy myśleć tylko o tym, jak struktura potomka i przodka mają na siebie wpływać, i określać odpowiednie elementy przekształcenia dla każdego składnika, który może później być dynamicznie edytowany. Tak więc po to, żeby uzyskać obrót robota wokół jego osi, podnoszenie ramienia i otwarcie obu rąk, zmieniamy macierz obrotu w strukturze robota tak, żeby wpływała na górną część, macierz obrotu w strukturze górnej części tak, żeby wpływała na odpowiednie ramię, i macierz przesunięcia w strukturze ramienia tak, żeby wpływała na kciuk. Przekształcenia są wykonywane niezależnie na każdym poziomie hierarchii, ale ostateczny efekt jest łączny. Trudna część specyfikowania animacji polega na ustalaniu sekwencji przekształceń w celu uzyskania pożądanego wyniku wstecznie na podstawie znajomości tego wyniku (takiego jak „robot porusza się do północno-zachodniego rogu pokoju i podnosi blok ze stołu").

7.7. Obsługa atrybutów wyglądu w hierarchii

7.7.1. Reguły dziedziczenia

Stan przeglądanie atrybutów jest ustawiany przez elementy atrybutu w czasie przeglądania i podobnie jak w SRGP jest stosowany modalnie do wszystkich napotkanych atrybutów. Widzieliśmy, jak rodzice wpływają na dzieci przez przekształcenia geometryczne. Jakie reguły odnoszą się do atrybutów wyglądu? W naszym przykładzie ulicy wszystkie ulice mają domniemaną barwę. W celu nadania obiektowi określonej barwy (na przykład brązowej dla domu) możemy podać tę barwę jako

początkowy element w samej strukturze obiektu, ale wtedy barwa staje się elementem wewnętrznym obiektu i nie podlega zmianom w czasie przeglądania. Wolelibyśmy „przekazywać barwę jako parametr", tak żeby potomek mógł dziedziczyć w taki sam sposób, w jaki potomek dziedziczy CMTM jako swoje GM.

W SPHIGS każda podstruktura dziedziczy stan przeglądania, jaki jest w czasie odwołania do podstruktury i może potem modyfikować ten stan w razie potrzeby bez wpływania na swoich przodków. Innymi słowy, atrybuty i przekształcenia są raczej powiązane dynamicznie w czasie przeglądania niż statycznie w czasie specyfikowania. To dynamiczne powiązanie jest jedną z głównych cech SPHIGS, dzięki czemu można łatwo przystosowywać kopie podstruktury.

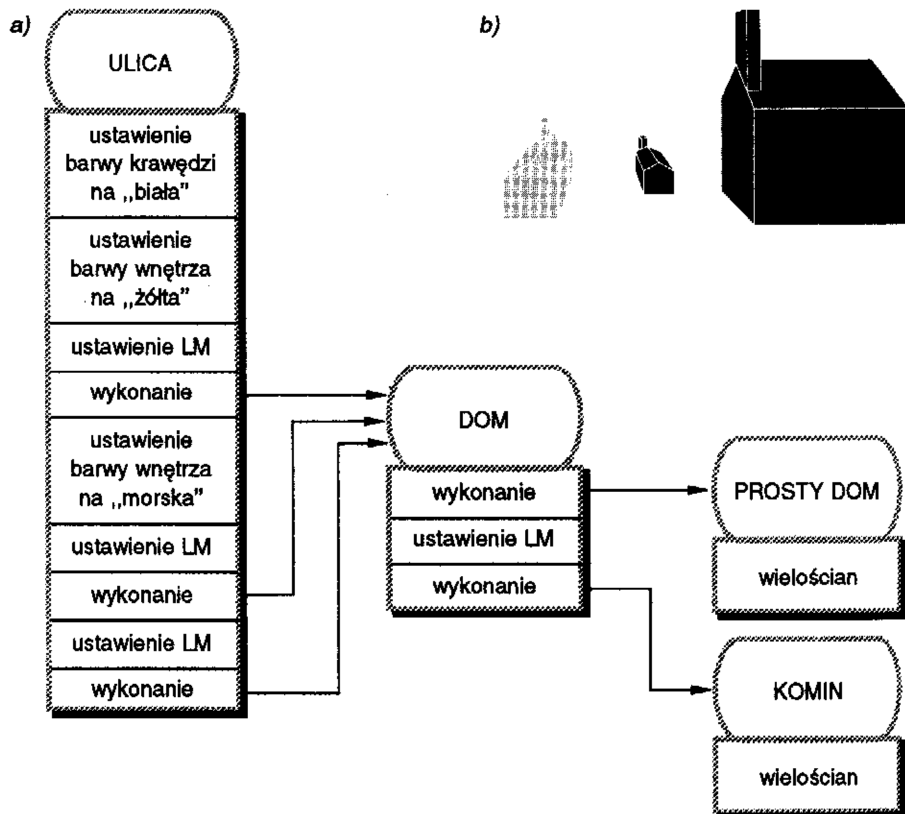
To, co podstruktura robi z dziedziczonym stanem, zależy od typu wykorzystanych danych. Widzieliśmy, że dla przekształceń geometrycznych podstruktura dziedziczy GM, ale nie może pokonać dziedziczenia, ponieważ może wpływać tylko na swoją własną macierz lokalną. Atrybuty są prostsze, ponieważ podstruktura dziedziczy atrybuty rodzica jako początkowe wartości swojego lokalnego stanu atrybutu, ale później może zmienić swój stan lokalny. Nie ma potrzeby rozróżniać między atrybutami globalnym i lokalnym, ponieważ nie istnieje pojęcie atrybutów złożonych. Zauważmy, że przy tym mechanizmie jest taki sam problem, z jakim zetknęliśmy się przy dziedziczeniu przekształceń – tak jak dwie kopie ramienia naszego robota nie mogą mieć różnych przekształceń dla kciuka, tak dwie kopie ramienia nie mogą mieć tej samej barwy dla stałej części i różnych barw dla kciuków.

W sieci struktury z rys. 7.13a struktura ulicy ustawia barwy dla podstruktury domu. Otrzymany obraz pokazano na rys. 7.13b, a kod generujący sieć pokazano jako program 7.6.

W ramach podstruktury atrybut może być wyzerowany po to, żeby zmienić wartość dziedziczenia. Następujący fragment kodu specyfikuje zrewidowaną strukturę domu, którego komin jest zawsze czerwony:

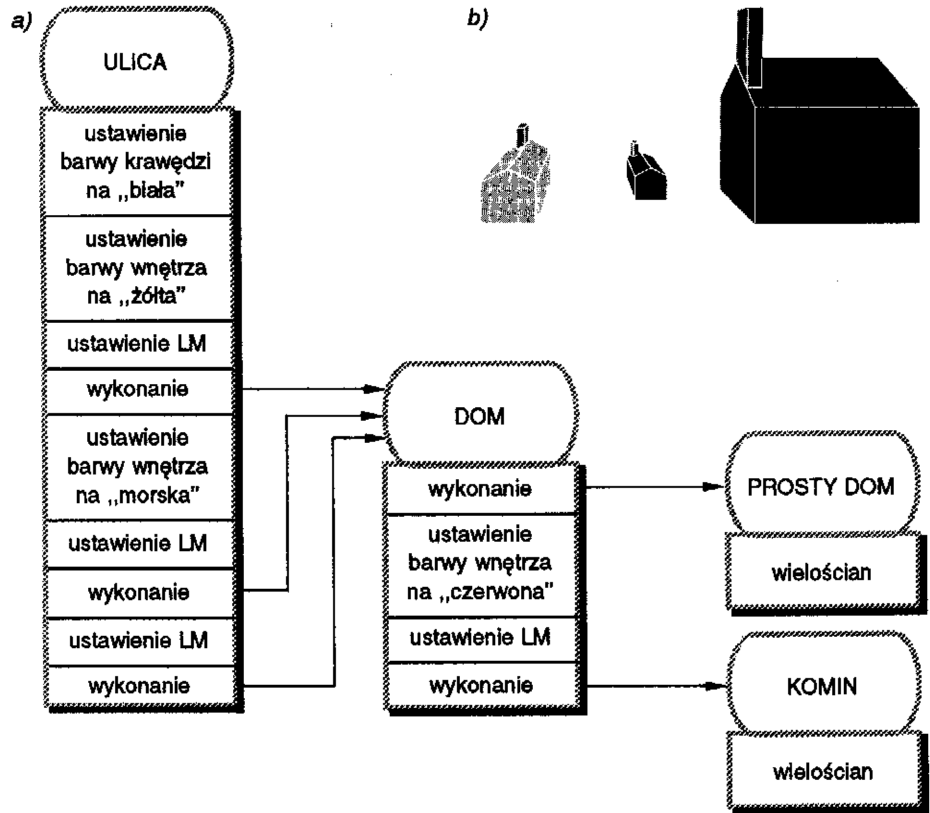
```
SPH_openStructure (HOUSE_STRUCT);
   SPH_executeStructure (SIMPLE_HOUSE_STRUCT);
   SPH_setInteriorColor (COLOR_RED);
   ustawienie przekształceń;
   SPH_executeStructure (CHIMNEY_STRUCT);
SPH_closeStructure();
```

Wykorzystajmy tę nową strukturę domu razem ze strukturą ulicy generowaną przez kod w programie 7.6. Na rysunku 7.14 pokazano sieć struktury i wynikowy obraz. Program przeglądania zaczyna się w STREET_STRUCT; atrybuty barwy wnętrza i krawędzi mają



Rys. 7.13. Wykorzystanie atrybutu dziedziczenia do modelowania ulicy z kolorowymi domami: a) struktura sieci; b) wynikowy obraz (Wewnętrzne barwy są symulowane za pomocą wzorów.)

wartości domniemane. Krawędź ma barwę białą i jest to wartość utrzymywana przez cały czas przeglądania i wyświetlania tej sieci. Pierwsze polecenie setInteriorColor powoduje dziedziczenie żółtej barwy przez pierwszą kopię HOUSE_STRUCT, która z kolei przekazuje barwę żółtą do SIMPLE_HOUSE_STRUCT, której wielościan będzie miał tę barwę. Gdy program przeglądania wraca z SIMPLE_HOUSE_STRUCT do HOUSE_STRUCT atrybut wewnętrznej barwy jest natychmiast zmieniany przez następny element na czerwony. Wobec tego odwołanie do CHIMNEY_STRUCT daje w wyniku czerwony komin z białymi karawędziami. Żadna z tych operacji nie wpływa oczywiście na atrybut grupy dla STREET_STRUCT; jeżeli program przeglądania wraca z HOUSE_STRUCT, to atrybut wewnętrznej barwy dla STREET_STRUCT wraca na żółty. Wtedy atrybut barwy wnętrza zmienia się na morski w celu przygotowania do narysowania dwóch domów o barwie morskiej.



Rys. 7.14. Podrzędna struktura zmienia dziedziczony atrybut: a) sieć struktury; b) wynikowy obraz

Program 7.6
Kod wykorzystany do narysowania rys. 7.13

```
SPH_openStructure (STREET_STRUCT);
   SPH_setEdgeColor (COLOR_WHITE);

   SPH_setInteriorColor (COLOR_YELLOW);
   ustawienie przekształceń;
   SPH_executeStructure (HOUSE_STRUCT);

   SPH_setInteriorColor(COLOR_NAVY);
   ustawienie przekształceń;
   SPH_executeStructure (HOUSE_STRUCT);

   ustawienie przekształceń;
   SPH_executeStructure (HOUSE_STRUCT);
SPH_closeStructure();
```

7.7.2. Atrybuty SPHIGS i tekst nie modyfikowane przez przekształcenia

W rzeczywistych implementacjach PHIGS tekst może podlegać przekształceniom, podobnie jak inne parametry. Wobec tego znaki tekstu na bokach podstawy wyświetlane w perspektywie są obracane i pokazywane z odpowiednim skrótem perspektywicznym, tak jak gdyby litery były zrobione z pojedynczych łamanych i wypełnionych obszarów. Podobnie kreski w liniach przerywanych powinny podlegać przekształceniom geometrycznym i skrótom perspektywicznym. Podobnie jednak jak atrybuty w SPHIGS są wielkościami niegeometrycznymi ze względu na efektywność, również tekst jest zaliczany do takiej klasy. Podobnie jak w SRGP krój pisma tekstu określa wielkość tekstu na ekranie i ciąg znaków nie może być nawet obrócony – obraz ciągu znaków tekstu jest zawsze równoległy do poziomego boku ekranu i nigdy nie podlega kompresji ani rozszerzaniu. W efekcie obroty i skalowanie wpływają na położenie punktu początkowego tekstu, nie wpływają natomiast na jego wielkość i orientację. Dlatego prymityw tekstu w SPHIGS jest użyteczny przede wszystkim do etykietowania.

7.8. Uaktualnianie ekranu i tryby renderingu

SPHIGS stale uaktualnia obraz ekranu w celu dopasowania bieżącego stanu CSS i tablicy rzutów. Następujące akcje mogą spowodować, że jakaś część obrazu ekranu będzie nieaktualna:

- Zmieniła się pozycja w tablicy rzutów.
- Struktura została zamknięta (po tym, jak została otwarta i poddana edycji).
- Struktura została usunięta.
- Struktura została wysłana albo nie wysłana.

Za każdym razem kiedy SPHIGS jest wywoływany w celu wykonania jednej z tych akcji, musi zregenerować obraz ekranu w celu wyświetlenia bieżącego stanu wszystkich wysłanych sieci. To, jak SPHIGS będzie generował obraz, jest określone przez tryb renderingu wybrany przez program użytkowy. Te tryby umożliwiają dokonanie wyboru między jakością a szybkością regeneracji: im lepsza jest jakość, tym dłużej trwa rendering obrazu. Tryb renderingu dla określonego rzutu jest ustawiany przez

```
void SPH_setRenderingMode( int viewIndex, int WIREFRAME / FLAT / LIT_FLAT /
    GOURAUD );
```

Teraz omówimy krótko tryby renderingu w SPHIGS; są one omawiane dokładniej w rozdz. od 12 do 14.

Tryb drutowy. Tryb WIREFRAME (drutowy) jest trybem najszybszym, ale daje najmniej realistyczną postać obrazu. Obiekty są rysowane tak, jak gdyby były wykonane z drutu i są pokazywane tylko ich krawędzie. Widoczne (w ramach bryły widzenia) części krawędzi wszystkich obiektów są pokazywane w całości, bez usuwania krawędzi niewidocznych. Prymitywy są rysowane w kolejności, w jakiej program przeglądania napotyka je w wysłanej sieci struktury w bazie danych; na tę kolejność ma wpływ priorytet wyświetlania określony przez indeks rzutu, tak jak to powiedziano w p. 7.3.4.

Wszystkie atrybuty krawędzi wpływają na wygląd ekranu w sposób wynikający z tego trybu; gdy wskaźnik krawędzi jest ustawiony na EDGE_INVISIBLE, wypełnione obszary i wielościany są w tym trybie całkowicie niewidoczne.

Tryby z cieniowaniem. W swoich innych trzech trybach renderingu SPHIGS wyświetla wypełnione obszary i wielościany w bardziej realistyczny sposób, rysując wypełnione obszary i ścianki jako wypełnione wielokąty. Dodanie cieniowanych obszarów w procesie renderingu zwiększa w istotny sposób złożoność, ponieważ istotne staje się przestrzenne uporządkowanie – części obiektów, które są niewidoczne (ponieważ są przesłonięte przez części bliższych obiektów), nie mogą być wyświetlane. Metody określania powierzchni widocznych (znane również jako usuwanie powierzchni niewidocznych) są omawiane w rozdz. 13.

W trzech metodach renderingu z cieniowaniem SPHIGS cieniuje wewnętrzne piksele widocznych części ścianek; jakość renderingu zależy od trybu. Przy cieniowaniu w trybie FLAT, używanym często na rysunkach w tym rozdziale, wszystkie ściany wielościanów są pokrywane bieżącą barwą wewnętrzną bez uwzględniania wpływu jakiegokolwiek źródła światła w scenie. Widoczne fragmenty krawędzi są pokazane (jeżeli wskaźnik krawędzi jest ustawiony na wartość EDGE_VISIBLE) tak, jak wyglądałyby w trybie WIREFRAME. Jeżeli barwa wnętrza jest taka sama jak barwa tła ekranu, to są widoczne tylko krawędzie – takie wykorzystanie trybu FLAT prowadzi do efektów jak na rys. 7.7a i 7.9c i symuluje model drutowy z usuniętymi krawędziami niewidocznymi.

Dwa tryby renderingu o najlepszej jakości tworzą obrazy oświetlone przez źródło światła[5]; modele oświetlenia i cieniowania są omawiane w rozdz. 14. Takie obrazy są cieniowane nierównomiernie; barwy pikseli zależą, ale nie wyłącznie, od atrybutu barwy wewnętrznej. W trybie

[5] Rozszerzenie PHIGS PLUS daje wiele możliwości sterowania renderingiem, włącznie z określeniem rozmieszczenia i barw wielu źródeł światła, właściwości materiałów obiektu wpływających na oddziaływanie ze światłem itd; por. rozdz. od 12 do 14.

LIT_FLAT wszystkie piksele na ścianie mają tę samą barwę określoną przez kąt, pod jakim światło pada na ścianę. Ponieważ każda ściana ma jednolitą barwę, widać, że cały obraz jest złożony ze ścian i zauważalny jest kontrast między ścianami mającymi wspólną krawędź. W trybie GOURAUD barwy pikseli są tak ustalane, żeby uzyskać efekt gładkiego cieniowania, który eliminuje wyraźny podział na ściany.

W trybie FLAT atrybut wskaźnika krawędzi powinien być ustawiony na EDGE_VISIBLE, ponieważ bez widocznych krawędzi obserwator może określić tylko granice sylwetki obiektu. Jednak w dwóch trybach o najlepszej jakości widzialność krawędzi zazwyczaj jest wyłączona, ponieważ cieniowanie pomaga użytkownikowi określić kształt obiektu.

7.9. Edycja struktury sieci dla efektów dynamicznych

Podobnie jak w przypadku każdej bazy danych musimy mieć możliwość tworzenia struktury bazy danych SPHIGS oraz zadawania pytań; musimy mieć również możliwość wygodnego wykonywania edycji. Program użytkowy dokonuje edycji struktury za pomocą funkcji opisanych w tym punkcie; jeżeli program użytkowy również obsługuje model dla danego zastosowania, to musi zapewnić, żeby edycja obu reprezentacji odbywała się równocześnie. *Dynamika ruchu* wymaga modyfikacji przekształceń rzutowania i modelowania; *dynamika uaktualniania* wymaga zmian lub zastąpień w strukturze. Programista może wybrać edytowanie wewnętrznej listy elementów struktury, jeżeli zmiany są niewielkie; w przypadku dużych edycji praktykuje się usuwanie i potem ponowne specyfikowanie struktury jako całości.

W pozostałej części tego punktu przedstawiamy metody edycji wewnątrzstrukturalnej; w podręczniku referencyjnym można znaleźć informacje o operacjach edycji, które wpływają na całe struktury (na przykład usuwanie, zerowanie), oraz bardziej szczegółowe opisy prezentowanych tu funkcji.

7.9.1. Dostęp do elementów za pomocą indeksów i etykiet

Podstawowe możliwości edycyjne zarówno SPHIGS, jak i PHIGS przypominają stare programy edycyjne zorientowane na wiersze, które używały numerów wierszy. Elementy w strukturze są numerowane od 1 do *N*; gdy element jest wstawiany lub usuwany, wówczas indeks związany z każdym elementem w tej samej strukturze o większej wartości indeksu jest zwiększany albo zmniejszany. Unikatowy element bieżący jest to

taki element, którego indeks jest zapamiętany w zmiennej stanu element-wskaźnik. Gdy struktura jest otwarta za pomocą wywołania SPH_openStructure, wówczas wskaźnik jest ustwiany na *N* (wskazując na ostatni element) albo na zero dla pustej struktury. Wskaźnik jest zwiększany, gdy nowy element jest umieszczany po bieżącym elemencie, i zmniejszany, gdy bieżący element jest usuwany. Wskaźnik może być również ustawiony bezpośrednio przez programistę korzystającego z poleceń pozycjonowania bezwzględnego i względnego:

```
void SPH_setElementPointer( int index );
void SPH_offsetElementPointer( int delta );
   /* „+" – do przodu, „–" – do tyłu */
```

Ponieważ indeks elementu zmienia się, gdy poprzedni element jest dodawny albo usuwany ze swojej macierzystej struktury, z korzystaniem z indeksów elementów do umieszczenia wskaźnika elementu wiąże się możliwość powstawania błędów. Dlatego SPHIGS umożliwia programowi użytkowemu umieszczanie w strukturze elementów orientacyjnych określanych etykietami. Etykiecie w czasie tworzenia jest nadawany identyfikator całkowitoliczbowy:

```
void SPH_label( int id );
```

Program użytkowy może przesunąć wskaźnik elementu za pomocą

```
void SPH_moveElementPointerToLabel( int id );
```

Wskaźnik jest wtedy przesuwany do przodu w poszukiwaniu określonej etykiety. Jeżeli zostanie osiągnięty koniec struktury, zanim zostanie znaleziona etykieta, to poszukiwanie kończy się niepowodzeniem; dlatego wskazane jest przesuwanie wskaźnika na początek struktury (indeks 0), zanim zacznie się poszukiwanie etykiety.

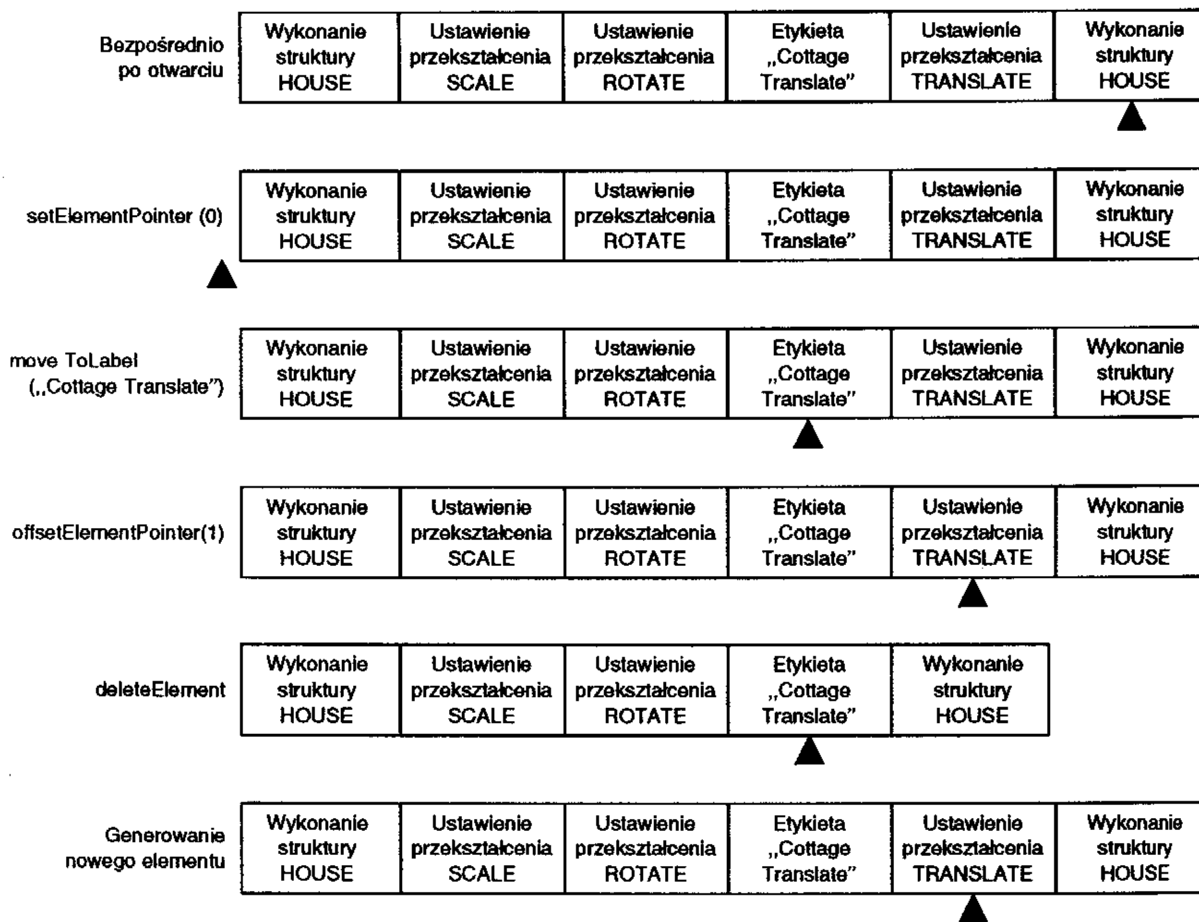
7.9.2. Operacje edycji wewnątrzstrukturalnej

Najpopularniejszą czynnością edycyjną jest wstawianie nowych elementów do struktury. Gdy jest wywoływana funkcja generująca element, wówczas nowy element jest umieszczany natychmiast za bieżącym elementem i wskaźnik elementu jest tak zwiększany, żeby wskazywał nowy element. Elementy są usuwane za pomocą następujących funkcji:

```
void SPH_deleteElement();
void SPH_deleteElementsInRange( int firstIndex int secondIndex );
void SPH_deleteElementsBetweenLabels( int firstLabel, int secondLabel );
```

We wszystkich przypadkach, po wykonaniu operacji usunięcia, wskaźnik elementu jest przesuwany do elementu bezpośrednio poprzedzającego element usunięty i wszystkie pozostałe elementy są przenumerowywane. Pierwsza funkcja usuwa bieżący element. Druga funkcja usuwa element leżący między dwoma określonymi elementami, łącznie z tymi elementami. Trzecia funkcja jest podobna, ale nie usuwa tych dwóch elementów z etykietami.

Zauważmy, że te wszystkie funkcje edycyjne wpływają na cały element albo na zbiór elementów; nie ma mechanizmów dla selektywnej edycji pól danych wewnątrz elementu. Stąd na przykład, gdy ma być uaktualniony jeden wierzchołek, programista musi ponownie podać specyfikację całego wielościanu.



Rys. 7.15. Kolejne stany struktury w czasie edycji. Czarny trójkąt pokazuje pozycję wskaźnika elementu (Ze względów ilustracyjnych skrócono syntaktykę wywołań.)

Przykład edycji. Przyjrzyjmy się modyfikacji naszego prostego przykładu ulicy. Ulica składa się teraz tylko z pierwszego domu i domku, przy czym ten pierwszy jest stały, a ten drugi może być przesuwany. Przed domkiem umieszczamy etykietę tak, żeby było możliwe dokonywanie edycji przekształcenia w celu przesunięcia domku.

W celu przesunięcia domku ponownie otwieramy strukturę ulicy, przesuwamy wskaźnik do etykiety, a następnie o ustaloną wartość do elementu przekształcenia, zastępujemy element przekształcenia i zamykamy strukturę. Ekran jest automatycznie uaktualniany po zamknięciu struktury w celu pokazania domku w nowym położeniu. Ten kod jest pokazany w programie 7.7, a sekwencja jego działania jest zilustrowana na rys. 7.15.

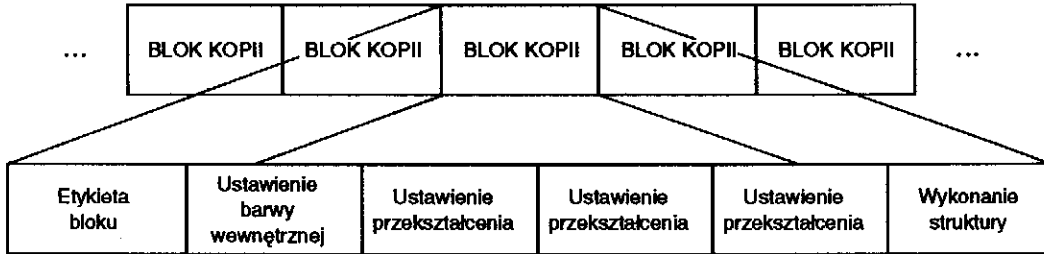
Program 7.7 Kod dla edycji struktury ulicy z rys. 7.15

```
SPH_openStructure(STREET_STRUCT);
  /* Po otwarciu struktury wskaźnik elementu jest początkowo na jej końcu. Najpierw
     musimy przesunąć wskaźnik na początek, tak żebyśmy mogli szukać etykiet. */
  SPH_setElementPointer(0);
  SPH_moveElementPointerToLabel(COTTAGE_TRANSLATION_LABEL);
  SPH_offsetElementPointer(1); /* Wskaźnik pokazuje teraz element przekształcenia */
  SPH_deleteElement(); /* Zastąpienie za pomocą kombinacji usunięcie/wstawienie */
  SPH_setLocalTransformation(newTranslationMatrix, PRECONCATENATE);
SPH_closeStructure();
```

7.9.3. Bloki kopii

Poprzedni przykład edycji sugeruje, że umieszczamy etykietę przed każdym elementem, który chcemy poddać edycji; jednak tworzenie tylu etykiet jest oczywiście zbyt pracochłonne. Jest kilka metod unikania takich trudności. Pierwsza polega na ujęciu grupy elementów w nawias z dwóch etykiet i potem używaniu etykiet do usuwania albo zastępowania całej grupy. Inna popularna metoda polega na grupowaniu zbioru elementów w stały format i wprowadzaniu grupy za pomocą jednej etykiety. W celu dokonania edycji dowolnego elementu z grupy przesuwamy wskaźnik elementu do etykiety, a potem przesuwamy wskaźnik o znaną wartość w ramach grupy. Ponieważ format jest ustalony, przesunięcie jest łatwą do określenia niewielką liczbą całkowitą.

Specjalny przypadek tej metody polega na zaprojektowaniu standardowego sposobu kopiowania podstruktur przez poprzedzenie elementu wykonanie-struktury wspólną listą elementów ustawianie-atrybutów. Typowy format takiej sekwencji elementów, określany jako *blok*



Rys. 7.16. Prosty format bloku kopii

kopii, pokazano na rys. 7.16; najpierw jest etykieta jednoznacznie identyfikująca cały blok, potem jest określenie barwy wnętrza, następnie są trzy podstawowe przekształcenia i wreszcie wywołanie struktury symbolu.

W celu określenia przesunięć możemy utworzyć zbiór stałych symbolicznych:

```
#define INTERIOR_COLOR_OFFSET    1
#define SCALE_OFFSET             2
#define ROTATION_OFFSET          3
#define TRANSLATION_OFFSET       4
```

Korzystanie ze stałego formatu dla bloku gwarantuje, że określony atrybut jest modyfikowany w ten sam sposób dla każdej kopii. W celu zmienienia przekształcenia obrotu dla określonej kopii korzystamy z następującego kodu:

```
SPH_openStructure (numer identyfikacyjny struktury, która ma być poddana edycji);
   SPH_setElementPointer (0);
   SPH_moveElementPointerToLabel (etykieta potrzebnego bloku kopii);
   SPH_offsetElementPointer (ROTATION_OFFSET);
   SPH_deleteElement();
   SPH_setLocalTransformation (newMatrix, mode);
SPH_closeStructure();
```

Inną interesującą cechą bloków kopii jest to, że można łatwo określić etykietę wprowadzającą każdy blok; jeżeli program użytkowy ma wewnętrzną bazę danych określającą wszystkie kopie obiektów, co jest popularnym rozwiązaniem, to etykieta może otrzymać unikatowy numer, który program użytkowy wykorzystuje do wewnętrznego określania kopii.

7.9.4. Sterowanie automatyczną regeneracją obrazu na ekranie

SPHIGS stale uaktualnia obraz ekranu w celu pokazania bieżącego statusu bazy danych pamięci struktury i jej tablicy rzutów. Czasami jednak chcemy wstrzymać tę regenerację albo w celu zwiększenia efek-

tywności, albo w celu uniknięcia prezentowania użytkownikowi ciągle zmieniającego się obrazu, co wprowadza zamieszanie i pokazuje nieistotne pośrednie kroki procesu edycji.

Jak widzieliśmy, SPHIGS sam ogranicza regenerację w czasie edycji struktury; niezależnie od ilości robionych zmian obraz jest regenerowany tylko po zamknięciu struktury. Takie grupowanie uaktualnień jest wykonywane ze względu na efektywność – każde usunięcie albo przekształcenie prymitywu może spowodować dowolne zniszczenia w obrazie na ekranie – uszkodzenie, które wymaga albo selektywnego naprawienia, albo wymusza ponowne przejrzenie wszystkich wysłanych sieci w jednym lub kilku rzutach. Oczywiście dla SPHIGS jest szybciej obliczyć skumulowany efekt pewnej liczby kolejnych edycji tylko raz przed regeneracją.

Podobna sytuacja powstaje, gdy kilka kolejnych zmian jest wykonywanych w odniesieniu do różnych struktur – na przykład, gdy struktura i jej podstruktury są usuwane za pomocą kolejnych wywołań do deleteStructure. W celu uniknięcia tego problemu program użytkowy może zatrzymać automatyczną regenerację przed wykonaniem ciągu zmian i potem ponownie ją uruchomić:

```
void SPH_setImplicitRegenerationMode( int ALLOWED / SUPPRESSED );
```

Nawet jeżeli jest zatrzymana bezpośrednia regeneracja, to program użytkowy może bezpośrednio zażądać regeneracji ekranu wywołując:

```
void SPH_regenerateScreen();
```

7.10. Interakcja

Moduły interakcji w SRGP i SPHIGS są zbudowane na bazie specyfikacji PHIGS i dlatego mają te same możliwości ustawiania trybów urządzeń i atrybutów i otrzymywania wartości. Klawiatura SPHIGS jest identyczna jak w SRGP, prócz tego że źródło echa jest określone w przestrzeni NPC z pominięciem współrzędnej z. Lokalizator SPHIGS ma dodatkowe pole dla współrzędnej z, a poza tym nie jest zmieniony. SPIGS dodaje również dwie nowe możliwości interakcji. Pierwsza to *korelacja wskazywania* rozszerzająca funkcje lokalizatora o określanie obiektu wskazanego przez użytkownika. Druga to urządzenie do wybierania, opisane w podręczniku referencyjnym, współpracujące z menu.

7.10.1. Lokalizatory

Lokalizator SPHIGS przesyła pozycję kursora we współrzędnych NPC, z $z_{NPC} = 0$. Przesyła również indeks rzutu o najwyższym priorytecie do pola wizualizacji, w którym znajduje się kursor.

```
typedef struct {
    point  position;       /* pozycja na ekranie [x,y,0]NPC */
    int    viewIndex;      /* Indeks rzutu o najwyższym priorytecie, w którego
                              polu wizualizacji znajduje się kursor */
    int    buttonOfMostRecentTransition;
    enum {
        UP, DOWN
    } buttonChord[MAX_BUTTON_COUNT];   /* Typowo 1...3 */
} locatorMeasure;
```

Gdy dwa pola wizualizacji nakładają się i pozycja kursora znajduje się na przecięciu ich brzegów, wówczas w drugim polu jest pokazane pole wizualizacji o najwyższym indeksie (w tablicy rzutów). Dlatego indeks rzutu jest używany do określenia priorytetu rzutu zarówno dla wejścia, jak i dla wyjścia. Pole indeksu rzutu jest użyteczne z kilku powodów. Rozważmy zastosowanie, które umożliwia użytkownikowi interakcyjne określenie granic pola wizualizacji, na przykład można przesuwać i zmieniać okna w programie zarządzania oknami. W odpowiedzi na zachętę do zmiany wielkości użytkownik może wskazać dowolne miejsce w polu wizualizacji. Program użytkowy może wtedy wykorzystać pole *viewIndex* do określenia, który rzut został wskazany, zamiast wykonywać test sprawdzający, czy punkt należy do prostokąta, w odniesieniu do granic pola wizualizacji. Indeks rzutu jest również używany w zastosowaniach, w których są tylko rzuty wyjściowe; takie zastosowania mogą sprawdzać zwrócony indeks rzutu w celu stwierdzenia, czy funkcja korelacji w ogóle musi być wywoływana.

7.10.2. Korelacja wskazywania

Ponieważ programista SPHIGS myśli raczej w kategoriach modelowanych obiektów niż pikseli tworzących jego obrazy, dla zastosowania jest istotna możliwość określania obiektu, którego obraz został wskazany. Dlatego pierwszym zastosowaniem lokalizatora jest dostarczenie punktu w układzie NPC na wejście funkcji korelacji wskazywania omawianej w tym punkcie. Jak widzieliśmy w przypadku SRGP, w świecie ograniczonym do płaszczyzny korelacja wskazywania jest bezpośrednim sposobem wykrywania trafień – prymitywów, których obrazy leżą dostatecznie blisko pozycji lokalizatora na to, żeby można uważać, że zostały wybrane przez użytkownika. Jeżeli jest więcej niż jedno trafienie, to, ze względu na nakładanie się prymitywów w pobliżu kursora, wybiera się ten ostatnio narysowany, ponieważ właśnie on leży na górze. Tak więc korelator wskazywania sprawdza prymitywy w odwrotnym porządku czasowym i wskazuje pierwsze trafienie. Wskazywanie obiektów w hierarchicznym

świecie 3D jest o wiele bardziej skomplikowane z przyczyn opisanych niżej; na szczęście SPHIGS uwalnia program użytkowy od tego zadania.

Wskazywanie w hierarchii. Rozważmy problemy wprowadzane przez hierarchię. Po pierwsze, jaka informacja powinna być przesłana przez narzędzie korelacji wskazywania w celu zidentyfikowania obiektu? Numer identyfikacyjny struktury nie wystarcza, ponieważ nie zapewnia rozróżnienia między licznymi kopiami struktury. Tylko pełna ścieżka – opis pełnego pochodzenia od korzenia do wskazanego prymitywu – zapewnia jednoznaczne określenie.

Po drugie, jeżeli jest wskazany określony prymityw, to o jakim poziomie hierarchii myślał użytkownik? Na przykład, jeżeli kursor jest umieszczony niedaleko jednego z kciuków robota, to czy użytkownik chce wybrać tułów, ramię, górną część czy całego robota? Czasami chodzi o bieżący prymityw, czasami o strukturę liścia lub jakikolwiek inny poziom aż do samego korzenia! Niektóre programy użytkowe rozwiązują ten problem dzięki wprowadzeniu mechanizmu sprzężenia zwrotnego umożliwiającego użytkownikowi przechodzenie przez poziomy od prymitywu do korzenia, w celu dokładnego określenia, który poziom jest potrzebny (por. zadanie 7.8).

Kryteria porównania. Jak jest zdefiniowana odległość od obiektu, gdy porównanie musi być rzeczywiście wykonane w 3D? Ponieważ lokalizator w zasadzie dostarcza wartość NPC 2D, nie ma bazy dla porównania współrzędnych *z* prymitywu i lokalizatora. Tak więc SPHIGS może porównywać pozycję kursora tylko w odniesieniu do obrazu prymitywu na ekranie, a nie w odniesieniu do położeń prymitywów w układzie WC. Jeżeli prymityw zostaje trafiony, to jest uważany za kandydata do korelacji. W trybie drutowym SPHIGS wskazuje pierwszego kandydata napotkanego w czasie przeglądania; uzasadnieniem dla tej strategii jest to, że w obrazie drutowym nie ma oczywistej informacji o głębokości i użytkownik nie może spodziewać się, że korelacja wskazywania weźmie pod uwagę relatywną głębokość. (Ubocznym efektem tej strategii jest to, że optymalizuje korelację wskazywania.) W trybach z renderingiem z cieniowaniem SPHIGS wskazuje kandydata, którego punkt trafienia (punkt NPC na znormalizowanej powierzchni prymitywu (3D NPC), który użytkownik wskazał bezpośrednio) jest najbliżej punktu obserwacji – czyli tego najbliższego z przodu.

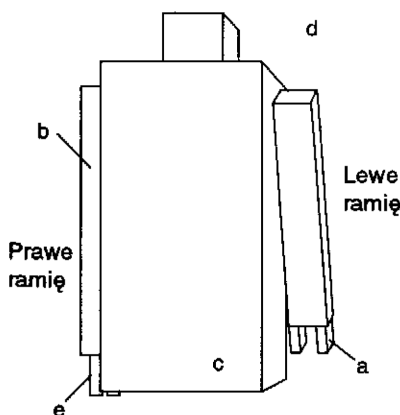
Funkcja korelacji wskazywania. W celu wykonania korelacji wskazywania program użytkowy wywołuje funkcję SPHIGS korelacji wskazywania z punktem NPC i indeksem rzutu, zazwyczaj otrzymanej w wyniku poprzedniej interakcji z lokalizatorem:

```
void SPH_pickCorrelate( point_3D position, int viewIndex,
    pickInformation *pickInfo);
```

Otrzymana zwrotnie informacja identyfikuje wskazany prymityw i jego przodków poprzez ścieżkę wskazania (*pick path*), tak jak to opisują typy danych C w programie 7.8.

Jeżeli w pobliżu pozycji kursora nie ma prymitywu, to zostaje przesłana wartość *pickLevel* (poziom wskazywania) 0 i pole *path* nie jest zdefiniowane. Jeżeli wartość pickLevel jest większa niż 0, to określa długość ścieżki od korzenia do wskazanego prymitywu – to znaczy głębokość prymitywu w sieci. W tym ostatnim przypadku pozycje od [1] do [*pickLevel*] tablicy *path* zwracają identyfikację elementów struktury występujących w ścieżce od korzenia do wskazanego prymitywu. Na najgłębszym poziomie (pozycja [*pickLevel*]) zidentyfikowany element jest tym prymitywem, który został wskazany; na wszystkich innych poziomach (pozycje od [*pickLevel*-1] do [1]) elementy są strukturami wykonania. Każda pozycja w *path* identyfikuje jeden element z rekordem, który zawiera identyfikator struktury zawierającej element, indeks elementu w tej strukturze, kod określający typ elementu i identyfikator wskazania elementu (omawiamy to dalej).

Na rysunku 7.17 wykorzystano strukturę sieci z rys. 7.10 dla górnej części robota i pokazano informację uzyskaną w wyniku kilku wskazań w wyświetlonym obrazie struktury.



Należy odwołać się do sieci struktury z rys. 7.10

(a) poziom=3
ścieżka[1] : struct UPPER _BODY, element 7
ścieżka[2] : struct ARM, element 3
ścieżka[3] : struct THUMB, element 1

(b) poziom=2
ścieżka[1] : struct UPPER_BODY, element 11
ścieżka[2] : struct ARM, element 1

(c) poziom=1
ścieżka[1] : struct UPPER_BODY, element 1

(d) poziom=0

(e) poziom=3
ścieżka[1] : struct UPPER_BODY, element 11
ścieżka[2] : struct ARM, element 3
ścieżka[3] : struct THUMB, element 1

Rys. 7.17. Przykład korelacji wskazywania

W jaki sposób ścieżka wskazania identyfikuje jednoznacznie każdą kopię struktury wywołanej arbitralnie wiele razy w ramach hierarchii? Na przykład, jak możemy rozróżnić wskazanie lewego kciuka robota od wskazania prawego kciuka? Ścieżki wskazania są identyczne z wyjątkiem poziomu korzenia, co widać na rys. 7.17 w punktach *a* i *e*.

Program 7.8
Typy pamiętania ścieżki wskazania

```
typedef struct {
   int    structureID;
   int    elementIndex;
   enum {
      POLYLINE, POLYHEDRON, EXECUTE_STRUCTURE
   } elementType;
   int    pickID;
} pickPathItem;

typedef pickPathItem pickPath[ MAX_HIERARCHY_LEVEL ];

typedef struct {
   int    pickLevel;
   pickPath path;
} pickInformation;
```

Identyfikator wskazania może dać korelację wskazania z większą rozdzielczością, niż to może zapewnić identyfikator struktury. Indeks elementu może być wykorzystany do identyfikowania poszczególnych elementów, podlega jednak zmianie w czasie edycji struktury. Dlatego korzystanie z identyfikatora wskazania jest łatwiejsze, ponieważ na identyfikator wskazania nie wpływają edycje innych elementów. Jego domniemaną wartością jest 0 i jest on ustawiony w strukturze w zależności od trybu. Element identyfikator wskazania generuje się za pomocą

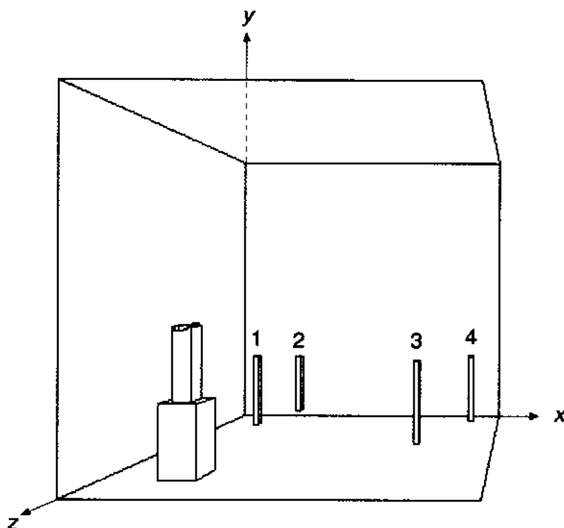
```
void SPH_setPickIdentifier( int id );
```

Element identyfikator wskazania jest pomijany w czasie przeglądania ekranu. Również identyfikator wskazania nie ma odniesienia co do dziedziczenia: początkowo jest ustawiony na 0, gdy SPHIGS zaczyna przeglądanie dowolnej struktury bez względu na to, czy jest to korzeń, czy podstruktura. Ze względu na te dwa aspekty identyfikatory wskazania nie zachowują się jak atrybuty. Powtarzające się prymitywy w strukturze mogą mieć albo niezależne identyfikatory, albo jeden wspólny; umożliwia to uzyskanie dowolnie dokładnej rozdzielczości korelacji wskazywania w strukturze, zależnie od potrzeb programu użytkowego. Chociaż etykiety i identyfikatory wskazania są różnymi mechanizmami – pierwszy jest wykorzystywany dla celów edycji, a drugi dla korelacji wskazywania – często są wykorzystywane razem. W szczególności gdy struktury są organizowane z wykorzystaniem metody kopii bloku opisanej w p. 7.9.2, element identyfikatora wskazania jest również częścią bloku i sam identyfikator wskazania jest typowo ustwiony na tę samą wartość całkowitą co identyfikator etykiety bloku.

Przykład 7.1 **Problem**: Wzbogać animację robota o interakcję z użytkownikiem. Załóżmy, że jest pewna liczba obiektów i pozwólmy użytkownikowi wybrać (za pomocą lokalizatora) obiekt, który robot powinien podnieść. Dla uproszczenia robot może być jednoręczny.

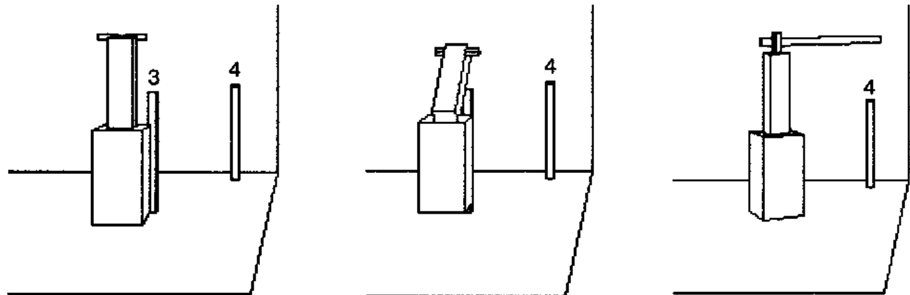
Odpowiedź: *Punkt obserwacji użytkownika.* Użytkownik widzi proste pomieszczenie z jednoręcznym robotem oczekującym w lewej przedniej stronie pomieszczenia. W głębi pomieszczenia są umieszczone pręty. Korzystając z myszki i naciskając lewy przycisk użytkownik może wybrać dowolny z tych obiektów i robot ma go usunąć. Użytkownik może przerwać w dowolnej chwili naciskając klawisz „q".

Rozmieszczenie obiektów. Obiekty, którymi robot może manipulować, rozmieszczamy w taki sposób, żeby prosta strategia wystarczyła na zbliżenie się, uchwycenie i usunięcie jednego z nich. Obiekty są równomiernie rozmieszczone w kierunku osi x i ograniczone, jeśli chodzi o głebokość obszaru pomieszczenia, w którym mogą być rozmieszczone (ograniczenie w kierunku osi z). Obiekty są zawieszone na odpowiedniej wysokości. Na rysunku pokazano typowe rozmieszczenie:



Przy takim rozmieszczeniu obiektów robot może poruszać się wzdłuż ścieżki równoległej do osi x, oczywiście przed obiektami dopóty, dopóki nie wyrówna się z jednym z nich (wyrównanie względem osi x). Następnie ruch odbywa się równolegle do osi z i wielkość ruchu w przód potrzebnego do umieszczenia robota w zasięgu obiektu jest łatwa do określenia. Z tej pozycji robot może uchwycić obiekt i podnieść go, a potem odwrócić się i wrócić drogą, którą przyszedł – do

przodu w kierunku równoległym do osi z. Dopuszczamy ruch robota w przód wzdłuż osi z na zewnątrz pomieszczenia (zakładamy, że jest tam korytarz), a potem w bok (równolegle do osi x). Zauważmy, że ten schemat działa dla wszystkich obiektów, przy czym jako parametry są potrzebne tylko współrzędne x i z. Następująca sekwencja rysunków pokazuje, jak robot zbliża się do wybranego pręta (*3*), zgina się, żeby uchwycić go, i przygotowuje się do opuszczenia pomieszczenia, żeby pozbyć się właśnie wybranego pręta.



Gdy robot znajdzie się poza ekranem, użytkownik nie wie, jaką czynność wykonuje robot. Dlatego chcemy, żeby robot jak najszybciej pozbył się obiektu i żeby tak zmienić jego kierunek poruszania, żeby wrócił na ekran. W tym celu po prostu odłączamy obiekt od hierarchii robota, zerujemy pozycje kciuka i każemy robotowi wrócić do jego obszaru oczekiwania.

Podamy teraz implementację algorytmu w kodzie języka C.

Program zapewniający użytkownikowi możliwość interakcyjnej kontroli nad animacją robota

```
/* definicja stałych dla programu */
/* definicja geometrii pomieszczenia, obiektów i robota */

main(argc, argv)
int   argc;
char **argv;
{

/* ustawienie początkowych rzutów, barw i parametrów SPHIGS */

   /* przygotowanie do obsługi zdarzenia wejściowego */
   SPH_setInputMode(KEYBOARD, EVENT);
   SPH_setKeyboardProcessingMode(RAW);

   SPH_setInputMode(LOCATOR, EVENT);
   SPH_setLocatorButtonMask(LEFT_BUTTON_MASK);

   ShowWorld(OUR_ROOM);    /* wyświetlenie początkowego rzutu pomieszczenia */
```

```
terminate = NO;
do {
   whichdev = SPH_waitEvent(INDEFINITE);
   switch (whichdev) {
      case KEYBOARD;
      SPH_getKeyboard(keymeasure, KEYMEASURE_SIZE);
      if (keymeasure[0] == 'Q' || keymeasure[0] == 'q')
         terminate = YES;
      break;
      case LOCATOR; {
         SPH_getLocator(&curr_locmeasure);
         /* po naciśnięciu przycisku sprawdzamy, co jest wskazane... */
         if (curr_locmeasure.button_chord[LEFT_BUTTON] == DOWN) {
            SPH_pickCorrelate(curr_locmeasure.position,
                              curr_locmeasure.view_index,
                              &pick_info);
            if (pick_info.pickLevel > 2) {   /* obiekt jest na drugim poziomie */
               if (pick_info.path[1].structureID == OBJECT_SET)
                  MakeAndRunPlan(pick_info.path[1].pickID);
            }
         }
         break;
      }              /* LOCATOR */
   }                 /* przełączenie */
}                    /* wykonanie */
   while (!terminate);
   SPH_end();
}

void   MakeAndRunPlan(int object_id)
{
   double  x, z;

   x = objects_info[object_id].x;
   z = objects_info[object_id].z;

   /* przesunięcie do wyrównania z obiektem według współrzędnej x: */
   MoveRobot(x, Z_WAIT);
   /* obrót w stronę obiektu i podejście: */
   SpinRobot(NORTH, CLOCKWISE);
   MoveRobot(x, z + DIST_OF_APPROACH);

   /* wzięcie obiektu... */
   LowerArm();
   Grab();
   Pickup(object_id);
   RaiseArm();
   /* obiekt wzięty. */

   /* zabranie obiektu */
```

```
    SpinRobot(EAST, COUNTERCLOCKWISE);
    SpinRobot(SOUTH, COUNTERCLOCKWISE);
    MoveRobot(x, Z_HALL);
    SpinRobot(EAST, CLOCKWISE);
    MoveRobot(X_OFFSCREEN, Z_HALL);

    /* odstawienie obiektu... */
    LowerArm();
    UnGrab();
    Letgo();
    RaiseArm();

    /* i powrót do obszaru oczekiwania */
    SpinRobot(WEST, CLOCKWISE);
    MoveRobot(X_WAIT, Z_HALL);
    SpinRobot(NORTH, COUNTERCLOCKWISE);
    MoveRobot(X_WAIT, Z_WAIT);
    SpinRobot(EAST, COUNTERCLOCKWISE);
}

void Pickup(int id)
/* Usunięcie odpowiedniego pręta z pomieszczenia i dodanie go do ramienia (dłoni)
   robota */
{
    matrix temp_matrix;                    /* macierz przekształcenia */
    SPH_openStructure(OBJECT_SET);
    SPH_setElementPointer(0);
    SPH_moveElementPointerToLabel(objects_info[id].label);
    SPH_offsetElementPointer(1);
    SPH_deleteElement();
    SPH_executeStructure(NULL_OBJECT);
    SPH_closeStructure();

    SPH_openStructure(OUR_ARM);
    SPH_setElementPointer(0);
    SPH_moveElementPointerToLabel(OUR_ARM_DRAW_ROD);
    SPH_offsetElementPointer(1);
    SPH_deleteElement();
    SPH_executeStructure(objects_info[id].struct_id);
    SPH_closeStructure();
    /* pokazanie ramki */
    SPH_regenerateScreen();
}
```

7.11. Zaawansowane problemy

Jest kilka problemów związanych z SPHIGS i standardem PHIGS, których nie można omówić szczegółowo w tej książce. Ograniczymy się do

podsumowania kilku z najważniejszych problemów, a jeśli chodzi o szczegóły, to odsyłamy do rozdz. 7 książki [FOLE90].

7.11.1. Dodatkowe funkcje wyjściowe

Paczki atrybutów. Standard PHIGS ma mechanizm do bezpośredniego ustawiania wartości atrybutu. Program użytkowy może w czasie swojej sekwencji inicjalizacji zapamiętać zbiór wartości atrybutów w *paczce atrybutów*. Prosty mechanizm dynamicznego zmieniania wyglądu obiektów polega na zmianie definicji paczek w tablicy paczek bez zmiany sieci struktury.

Zbiory nazw dla podświetlania i wygaszania. SPHIGS ma dwie tradycyjne metody sprzężenia zwrotnego, z których często korzystają programy użytkowe razem z funkcją wskazywania dostępną w SPHIGS: podświetlanie obiektów i wygaszanie ich tak, żeby stały się niewidoczne. Pierwsza z tych metod jest na ogół używana do realizacji sprzężenia zwrotnego, gdy użytkownik wskazuje obiekt; druga metoda umożliwia „odśmiecenie" ekranu i pokazanie tylko potrzebnych szczegółów. Podręcznik referencyjny SPHIGS opisuje elementy struktury, które wykonywane w czasie przeglądania dodają nazwy albo usuwają nazwy ze zbioru nazw.

Wymiana obrazów i metapliki. Chociaż ze względu na przenoszalność PHIGS i inne standardowe pakiety grafiki są niezależne od systemu i urządzenia, konkretne implementacje takich pakietów w określonym środowisku z dużym prawdopodobieństwem są ze względów na efektywność optymalizowane w sposób niezgodny z zasadami przenoszalności. Plik archiwalny PHIGS, zdefiniowany przez komitet standardów graficznych, jest przenoszalnym odzwierciedleniem stanu struktury bazy danych w danej chwili i umożliwia różnym implementacjom PHIGS wspólne korzystanie z modeli geometrycznych. Implementacje PHIGS mogą również zapisywać w metapliku informację o tym, co program użytkowy w danej chwili prezentuje na ekranie.

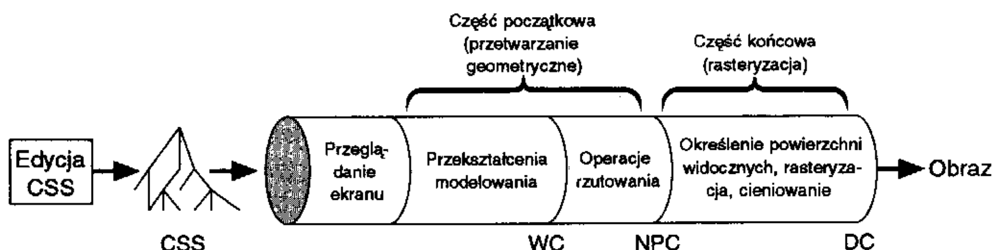
7.11.2. Problemy implementacyjne

W celu wyświetlenia struktury sieci SPHIGS „odwiedza" składniki elementów struktury korzystając z metody rekursywnego zstępowania, (przeglądania zstępującego) i dla każdego elementu wykonuje odpowiednie czynności, zależne od typu elementu. Ten proces wyświetlania, który odwzorowuje model na obraz na ekranie (albo trwałej kopii)

w kontekście PHIGS, jest określany jako *przeglądanie* ekranu, w ogólnym natomiast przypadku jest określany jako *rendering*; jego implementacja programowa i/lub sprzętowa jest określana jako *potok renderingu.*

Przeglądanie. Optymalizacja regeneracji mająca na celu przeglądanie jak najmniejszej części CSS jest dość trudna, ponieważ wpływ tej trywialnej operacji jest potencjalnie ogromny. Na przykład trudno jest określić, jaka część ekranu musi być zregenerowana na skutek wykonania edycji struktury.

Rendering. Na rysunku 7.18 pokazano koncepcyjny potok dla realizacji renderingu. Pierwszy krok to przeglądanie zstępujące. (Alternatywnie, jeżeli jest wykorzystywany pakiet graficzny działający w trybie natychmiastowym, to program użytkowy musi przejrzeć model programu użytkowego albo generować prymitywy i atrybuty w sposób proceduralny.) Każdy prymityw napotkany w czasie przeglądania jest przepuszczany przez pozostałą część potoku: najpierw są wykonywane przekształcenia modelowania (opisane w rozdz. 5) w celu odwzorowania prymitywu ze współrzędnych modelu na współrzędne świata. Następnie jest realizowana operacja rzutowania, mająca na celu przesunięcie i obcięcie prymitywu do kanonicznej bryły widzenia i potem odwzorowanie na prostopadłościan NPC (por. rozdz. 6). Ponieważ te procesy są niezależne od urządzenia wyświetlającego i są wykonywane dla wierzchołków we współrzędnych zmiennopozycyjnych, ta część potoku występująca bezpośrednio po przeglądaniu jest określana często jako podsystem przetwarzania geometrycznego.



Rys. 7.18. Potok renderingu w SPHIGS

Do końcowej części potoku, gdzie są tworzone piksele, docierają prymityw po przekształceniach i po obcięciu; proces przekształcania na piksele nazywa się *rasteryzacją*. Ten proces jest oczywiście bezpośredni dla trybu drutowego: współrzędne NPC można łatwo odwzorować (przez skalowanie i przesunięcie, przy pominięciu współrzędnej z) na całkowitoliczbowe współrzędne urządzenia; potem jest wywoływana z pakietu grafiki rastrowej odpowiednia funkcja rysowania odcinka w celu dokonania faktycznej rasteryzacji. Rendering z cieniowaniem

jest jednak złożoną operacją i składa się z trzech podprocesów: *określenia powierzchni widocznych* (określenia tych części prymitywu, które są widoczne dla syntetycznej kamery), *wykonania konwersji* (określenia pikseli pokrytych przez obraz prymitywu) i *cieniowania* (określenia, jaką barwę należy przypisać każdemu pokrytemu pikselowi). Dokładna kolejność wykonywania tych podprocesów zmienia się zależnie od trybu renderingu i metody implementacji. Procesy rasteryzacji dokładnie opisano w rozdz. od 12 do 14.

Optymalizacja przez sprawdzanie zasięgu. W przedstawionej strategii zawartość sieci jest przeglądana bezwarunkowo. Często jednak nie wszystkie obiekty sieci są widoczne, ponieważ w efekcie przekształceń modelowania i rzutowania wykonywanych w czasie przeglądania może się okazać, że duże części sieci leżą na zewnątrz bryły widzenia. W celu wykonania optymalizacji jest nam potrzebna prosta metoda obliczania granic dowolnie złożonego obiektu i efektywny sposób porównywania tych granic z polem wizualizacji NPC.

Animacja i podwójne buforowanie. Jednym z ubocznych efektów niektórych prostych implementacji SPHIGS jest to, że między ramkami animacji obserwator widzi wymazywanie ekranu i może (w mniejszym lub większym stopniu) widzieć, jak obiekty są rysowane w czasie, gdy program przeglądania wykonuje regenerację. Ten artefakt wizualny można zredukować korzystając z podwójnego buforowania: wykorzystanie kanw/map bitowych pozaekranowych do pamiętania następnej ramki w czasie, gdy bieżąca ramka jest rysowana; wtedy po zakończeniu regeneracji zawartość kanwy może być skopiowana na ekran.

Korelacja wskazywania. W korelacji wskazywania SPHIGS przegląda te sieci, które zostały wysłane do pola wizualizacji i w których leży określony punkt NPC. Przeglądanie jest prawie identyczne jak w czasie wyświetlania; są podtrzymywane macierze modelowania-przekształcania i jest wykonywana duża część potoku renderingu. Przeglądanie związane z korelacją wskazywania może być optymalizowane na wiele sposobów, włączając w to analityczne wykrywanie trafienia i wykrywanie trafienia metodą obcinania. Przykładem tego pierwszego rozwiązania jest funkcja PtInPolygon, która jest stosowana do testowania ze względu na trafienie w obszarach wypełnionych i ścianach w trybach renderingu z cieniowaniem. Jeden z popularnych algorytmów określania, czy pozycja kursora NPC leży wewnątrz wielokąta, wykorzystujący regułę parzystości i opisany w p. 2.1.3, wysyła promień z pozycji lokalizatora i określa, ile razy promień przecina wielokąt. Algorytm przegląda listę krawędzi i sprawdza przecięcia i specjalne przypadki (na przykład przecinanie wierzchołków, równoległość między krawędzią a promieniem). Algorytm konwersji wielokąta opisany w p. 3.5 dotyczy podobnego problemu i może być zaadaptowany do użycia jako funkcja PtInPolygon.

7.11.3. Optymalizacja wyświetlania modeli hierarchicznych

Elizja. Budynek możemy modelować jako hierarchię części mówiąc, że składa się z pięter, piętro składa się z biur itd.; w węzłach hierarchii nie ma prymitywów dopóty, dopóki nie dojdziemy do poziomu cegły, deski i płyt betonu, z których składa się wielościan. Chociaż taka reprezentacja może być wygodna dla celów konstrukcyjnych, nie jest użyteczna dla celów wyświetlania, gdzie czasami chcemy widzieć zgrubny, uproszczony obraz, który eliminuje zakłócające i niepotrzebne detale. Określenie *elizja* odnosi się do decyzji podejmowanej przez pogram przeglądania dla celów wyświetlania w celu powstrzymania przed schodzeniem do podstruktury. Elizja może być wykonana przez przycinanie, odrywanie oraz rozpatrywanie poziomu szczegółowości.

Wiązanie struktury. Niektóre implementacje PHIGS i PHIGS PLUS dopuszczają niestandardową formę wykonywania struktury, określaną jako *wiązanie*, która polega na tym, że pomija się kosztowne zapamiętywanie i odtwarzanie stanu ExecuteStructure. Optymalizacja polega na dodaniu operacji ReferStructure i zezwoleniu programiście na korzystanie z niej wówczas, gdy wywoływana struktura potomka nie ma żadnych elementów ustawiania atrybutów.

7.11.4. Ograniczenia modelowania hierarchicznego w PHIGS

Ograniczenia zwykłej hierarchii. Jak wspomniano w p. 1.3, niektóre zastosowania nie mają rzeczywistej struktury dla swoich danych (na przykład wykresy) dla danych pomiarowych albo mają najwyżej (częściowy) porządek w swoich danych (na przykład funkcja reprezentowana algebraicznie). Wiele innych zastosowań jest w bardziej naturalny sposób reprezentowanych jako sieci – to znaczy jako ogólne (skierowane) grafy (które mogą mieć podsieci hierarchiczne). Wsród tego są schematy układów i połączeń elektrycznych, sieci transportowe oraz komunikacyjne i schematy instalacji w fabrykach chemicznych. Innym przykładem modeli, dla których nie wystarcza prosta hierarchia, jest kostka Rubika – zbiór elementów, w których sieć i jakakolwiek hierarchia (powiedzmy warstwy, wiersze i kolumny) jest w zasadniczy sposób zmieniana po każdym przekształceniu.

Dla innych typów modeli jedna hierarchia nie wystarcza. Na przykład, uchwyt piórka w ploterze (x, y) jest poruszany przez oba ramiona poziome oraz pionowe i wobec tego „należy" do obu ramion. W skrócie, niezależnie od tego, czy model zastosowania cechuje czysta hierarchia, czysta sieć bez hierarchii, hierarchia w sieci z połączeniami krzyżo-

wymi, czy wielokrotne hierarchie, można wykorzystać SPHIGS do wyświetlania tego modelu, ale można nie chcieć albo nie móc wykorzystać struktury hierarchii w pełnej jej ogólności.

Ograniczenia „przekazywania parametrów" w SPHIGS. Traktowanie struktur jak czarnych pudełek jest dobre ze względu na modularność, ale – jak pokazano na przykładzie robota – może mieć ograniczenia. Na przykład, jak możemy zbudować robota z dwoma identycznymi ramionami i zlecić mu podnoszenie ze stołu elementu za pomocą prawego ramienia i odchodzenie z tym elementem? Struktura hierarchii nie zezwala, żeby kopie struktur różniły się parametrami przekształceń na różnych poziomach hierarchii, ponieważ hierarchia struktury nie ma ani ogólnego mechanizmu przekazywania parametrów hierarchii funkcji, ani ogólnych konstrukcji przepływu sterowania.

7.11.5. Alternatywne formy modelowania hierarchicznego

Hierarchia funkcji. W przedziale od czystej hierarchii danych do czystej hierarchii funkcji hierarchia struktury jest prawie przy końcu danych, ponieważ brak jest ogólnego przepływu sterowania. Dla kontrastu, funkcja szablonu (to jest funkcja określająca obiekt szablonu, składająca się z prymitywów albo wywołań do podrzędnych funkcji szablonu) może korzystać z parametrów i dowolnego przepływu sterowania.

Hierarchia danych. W przeciwieństwie do hierarchii funkcji, hierarchia danych dobrze nadaje się do tworzenia dynamicznego. Podobnie jak w hierarchii funkcji szablonu, może być wykorzystywana w połączeniu z pakietami graficznymi albo w trybie natychmiastowym albo podtrzymania.

Korzystanie z systemów baz danych. Ponieważ uniwersalna baza danych ma większą moc niż specjalizowana baza danych, powinniśmy wziąć pod uwagę wykorzystanie systemów standardowych baz danych w grafice komputerowej. Niestety takie bazy danych są projektowane do pracy z dużymi ilościami danych w pamięciach masowych i przy założeniu, żeby czas odpowiedzi był mierzony w skali czasu człowieka.

7.11.6. Inne (przemysłowe) standardy

W punkcie 7.1 mówiliśmy, że wiele cech omawianych w kontekście PHIGS jest dostępnych również i w konkurencyjnych rozwiązaniach takich jak OpenGL i HOOPS. W tym punkcie krótko powiemy, jakie są różnice między tymi pakietami, oraz pokażemy związki między PEX a PHIGS.

Najpierw musimy rozróżnić protokół grafiki 3D klient/serwer od proceduralnych API, takich jak PHIGS wykorzystywanych do implementacji zastosowań. API, czyli biblioteki lub pakiety poziomu zastosowania, wykorzystują protokoły niskiego poziomu klient/serwer do realizacji wymiany informacji na zasadzie potwierdzania potrzebnej do obliczeń rozproszonych. System X Window (X) jest od pewnego czasu standardem przemysłowym dla zastosowań grafiki 2D wybieranym do pracy w środowisku klient/serwer. Często procesy klienta i serwera są wykonywane fizycznie na różnych komputerach, połączonych siecią lokalną (LAN) albo siecią rozległą (WAN). W X proces serwera zarządza wyświetlaniem, a proces klienta zawiera kod programu użytkowego i pakiet graficzny; te dwa procesy komunikują się za pomocą protokołu komunikacji międzyprocesowej (IPC), który ma postać sekwencji poleceń, z których każde zawiera na ogół operację i jej parametry.

PEX (pierwotnie skrót od PHIGS Extensions to X; obecnie nie jest to rozwiązanie bazujące na PHIGS) jest to rozproszone graficzne rozszerzenie 3D do systemu X Window. PEX jest tylko wyjściem; wejście jest obsługiwane przez standardowy protokół wejściowy X.

PHIGS, HOOPS i PEXlib są najpopularniejszymi API, które rezydują na górze protokołu PEX. PEXlib, przez analogię do Xlib dla grafiki 2D, jest cienką nadbudową na górze protokołu PEX, która daje dostęp do wszystkich funkcji protokołu, ale nie daje wsparcia dla wysokiego stopnia abstrakcji. PHIGS i HOOPS ukrywają funkcjonalność niskiego poziomu, ale są mniej efektywne niż czyste wywołania PEXlib. OpenGL nie rezyduje na górze PEX, ale ma swój własny protokół dla grafiki rozproszonej.

Biblioteka OpenGL jest przystosowana tylko do renderingu, jest API niezależnym od dostawcy realizującym funkcje grafiki 2D i 3D, włączając w to modelowanie, przekształcenia, barwę, oświetlenie, gładkie cieniowanie, jak również zaawansowane funkcje, np. odwzorowanie tekstur (p. 14.3), NURBS (p. 9.2) i rozmycie ruchu (p. 12.4). Funkcje renderingu w OpenGL są nieco bardziej rozwinięte niż te w PHIGS PLUS, ponieważ były one oryginalnie wspierane przez sprzęt firmy Silicon Graphics. OpenGL, podobnie jak PEX, dopuszcza tryby natychmiastowy i podtrzymania. Nie zależy od systemu okien ani od systemu operacyjnego i została zintegrowana z systemem X Window w systemie UNIX.

HOOPS różni się od PHIGS w kilku istotnych aspektach. Ponieważ został on zaprojektowany i zaimplementowany przez jednego wytwórcę, wszystkie implementacje są ze sobą zgodne, co nie jest prawdą, jeśli chodzi o implementacje PHIGS – oficjalne standardy dopuszczają zbyt dużą elastyczność dla indywidualnych implementacji.

HOOPS daje pełniejsze wsparcie, jeśli chodzi o kroje pisma, dane obrazu i zaawansowany rendering, niż PHIGS. Dodatkowo do moż-

liwości renderingu dostępnych w PHIGS PLUS, HOOPS daje kilka form modeli oświetlenia globalnego, w tym metodę śledzenia promieni, metodę energetyczną i kombinowany renderer łączący śledzenie promieni z metodą energetyczną (por. rozdz. 14).

Podobnie jak PHIGS, HOOPS udostępnia hierarchię struktury, ale różnice są dosyć znaczne. Struktury PHIGS zawierają uporządkowane listy elementów graficznych i atrybutów. Wykonywanie edycji tych list wymaga od programisty dokładnej znajomości, jak, gdzie i kiedy atrybuty są ustawiane i modyfikowane. Dla kontrastu, wszystkie prymitywy w segmencie HOOPS mają ten sam stan atrybutu, który jest wyszczególniony w nagłówku segmentu. Upraszcza to zadanie programisty, tworzy bardziej modułowe, niezależne od uporządkowania modele i umożliwia implementacje o lepszych parametrach dzięki uproszczeniu operacji pomocniczych i eliminowaniu wielu przełączeń kontekstowych w potoku renderingu.

Podsumowanie

W rozdziale podano ogólne wprowadzenie do modeli geometrycznych, ze zwróceniem uwagi na modele hierarchiczne, które reprezentują zestawy części. Chociaż wiele typów danych i obiektów nie jest hierarchicznych, większość obiektów wytwarzanych przez człowieka przynajmniej częściowo jest. PHIGS i jego alternatywy (na przykład OpenGL, HOOPS i PEXlib) zapewniają wysoki poziom funkcjonalności kosztem istotnego wzrostu złożoności. Dają one możliwości hierarchicznego modelowania geometrycznego za pomocą wielokątów, wielościanów, krzywych oraz powierzchni i mogą dobrze działać w otoczeniu programów zarządzania oknami i obliczeń rozproszonych klient/serwer, jakie występują w systemie X Window.

SPHIGS, podzbiór PHIGS, jest zaprojektowany tak, żeby dostarczyć efektywną i naturalną reprezentację obiektów geometrycznych zapamiętanych w zasadzie jako hierarchie wielokątów i wielościanów. Ponieważ te pakiety pamiętają wewnętrzną bazę danych obiektów, programista może niewielkim wysiłkiem robić małe zmiany w bazie danych i pakiet automatycznie tworzy uaktualniony rzut. Tak więc program użytkowy tworzy i edytuje bazę danych, na ogół w odpowiedzi na wejście użytkownika, i pakiet jest odpowiedzialny za tworzenie określonych rzutów bazy danych. Te rzuty wykorzystują różne metody renderingu w celu uzyskania kompromisu jakość-szybkość. Pakiet zapewnia obsługę urządzeń wejściowych do lokalizowania i wskazywania, a także korelację wskazywania, dzięki czemu jest możliwy wybór obiektów na dowolnym poziomie hierarchii. Jako innej formy kontroli nad

wyglądem obiektów można użyć filtrów podświetlania i widzialności do selektywnego zezwalania i wzbraniania.

Ze względu na naturę struktur i ograniczone środki dla przeglądania i edycji, taki specjalizowany system najlepiej nadaje się do dynamiki ruchu i dynamiki uaktualniania światła, zwłaszcza jeżeli baza danych struktury może być utrzymywana w systemie wyświetlania tak zoptymalizowanym, żeby był urządzeniem zewnętrznym PHIGS. Jeżeli między dwoma kolejnymi obrazami trzeba uaktualnić dużą część bazy danych struktury albo jeżeli bazę danych zastosowania można przeglądać szybko i nie ma wąskiego gardła między komputerem a podsystemem wyświetlania, efektywniejsze jest używanie pakietu graficznego w trybie natychmiastowym, bez podtrzymywania informacji.

Hierarchia struktur leży między hierarchią czystych danych a hierarchią czystych funkcji. Umożliwia również prostą formę przekazywania parametru do podstruktur (geometrycznych albo atrybutów wyglądu), z wykorzystaniem mechanizmu stanu atrybut-przeglądanie. Ponieważ jednak brak jest ogólnych konstrukcji sterowania przepływem, mechanizm przekazywania parametrów jest ograniczony i struktury nie mogą selektywnie ustawiać różnych atrybutów w różnych kopiach podstruktury. Zamiast tego można użyć funkcji szablonów do ustawiania wielu kopii (hierarchicznych) struktur, które mają identyczną strukturę, ale różnią się atrybutami geometrii albo wyglądu podstruktur. Alternatywnie mogą być one wykorzystane do sterowania pakietem działającym w trybie natychmiastowym.

SPHIGS jest zorientowany na modele geometryczne budowane w zasadzie z wielokątów i wielościanów, zwłaszcza takich, w których występuje hierarchia; w rozdziałach 9 i 10 przyjrzymy się modelom geometrycznym, które mają bardziej złożone prymitywy i kombinacje prymitywów. Zanim przejdziemy do tych bardziej zaawansowanych zagadnień modelowania, zajmiemy się metodami i narzędziami interakcji i interfejsami użytkownika.

Zadania

7.1. a. Uzupełnij model robota z rys. 7.11 dodając podstawę, na której obraca się górna część, i utwórz prostą animację jego ruchu po pomieszczeniu.
b. Utwórz program użytkowy korzystając z SPHIGS tworzący animację, w której robot z jednym ramieniem zbliża się do stołu, na którym leży obiekt, podnosi ten obiekt i odchodzi z nim.

7.2. Ulepsz animację robota tak, żeby jednocześnie były widoczne trzy rzuty, w tym ogólny prostokątny widok i widok z *oka robota*, pokazujący, co widzi robot w czasie poruszania się.

7.3. Uaktualnij nasz program rekursywnego przeglądania tak, żeby zawierał informację o rozszerzeniu MC zapamiętaną dla każdej struktury. Załóż, że gdy struktura *S* po edycji zostaje zamknięta, jest ustawiane w rekordzie *S* pole boolowskie *extentObsolete*. Załóż również, że dostępne są funkcje, które dla danego prymitywu przesyłają rozszerzenie NPC prymitywu.

7.4. Zaproponuj algorytm analitycznego obliczania punktu trafienia wskazanego odcinka zakładając, że dane są końce odcinka NPC i odczyt z lokalizatora.

7.5. Zaproponuj algorytm analitycznego obliczania punktu trafienia w dany wypełniony obszar.

7.6. Zaproponuj, korzystając z pseudokodu, rekurencyjny program przeglądania dla korelatora wskazania, który jest przeznaczony tylko dla trybu drutowego.

7.7. Zaimplementuj funkcję PtInPolygon w celu wykorzystania w korelatorze wskazywania. Uwzględnij specjalne przypadki promieni, które przechodzą przez wierzchołki i są koincydentne z krawędziami. Dyskusję szczegółów tego problemu można znaleźć w pracach [PREP85] i [FORR85].

7.8. Zaprojektuj interfejs użytkownika dla wskazywania, który umożliwi użytkownikowi wskazanie potrzebnego poziomu hierarchii. Zaimplementuj i przetestuj swój interfejs na modelu robota – trzeba napisać program użytkowy, który umożliwi użytkownikowi podświetlenie części robota, od poszczególnych części do całych podsystemów.

8. Urządzenia wejściowe, metody i zadania interakcyjne

Uzyskanie dobrej jakości interfejsów użytkownika jest z wielu względów ostatnią przeszkodą w udostępnieniu komputerowych metod obliczania wielu użytkownikom, ponieważ koszty sprzętu i oprogramowania są obecnie dostatecznie niskie na to, żeby możliwości obliczeniowe trafiły do naszych biur i domów. Tak jak inżynieria oprogramowania stworzyła ostatnio struktury dla działalności, która kiedyś była działalnością ad hoc, tak również nowy obszar inżynierii interfejsu użytkownika generuje zasady dla interfejsów użytkownika i metodologię projektowania.

Jakość interfejsu użytkownika często określa, czy użytkownicy przyjmą, czy odrzucą system, czy projektanci systemu zostaną nagrodzeni, czy potępieni, czy system odniesie sukces, czy porażkę na rynku. Projektant interakcyjnego graficznego programu użytkowego musi być wyczulony na wymagania użytkowników stawiane interfejsom – muszą one być łatwe do nauczenia się, a mimo to bardzo silne.

Rozwiązanie interfejsu użytkownika z oknami, ikonami i rozwijanymi menu, wykorzystującymi silnie grafikę rastrową, jest popularne, ponieważ jest łatwe do nauczenia się i nie wymaga dużych umiejętności pisania na klawiaturze. Większość użytkowników takich systemów to nie są programiści, którzy nie darzą wielką sympatią staromodnych, trudnych do opanowania, zorientowanych klawiaturowo interfejsów języków poleceń, do których wielu programistów jest przyzwyczajonych. Proces projektowania, testowania i implementacji interfejsu użytkownika jest złożony; wskazówki i metodologie można znaleźć w pracach [FOLE90; SHNE86; MAYH90].

W tym rozdziale skupimy się na urządzeniach wejściowych, technologiach interakcji i zadaniach interakcyjnych. Są to podstawowe

elementy składowe, z których są budowane interfejsy użytkownika. Urządzenia wejściowe są to rozwiązania sprzętowe, za pomocą których użytkownik wprowadza informacje do systemu komputerowego. W rozdziale 4 omówiliśmy wiele takich urządzeń. W tym rozdziale wprowadzamy dodatkowe urządzenia i omawiamy powody, dla których jedno urządzenie może być lepsze od drugiego. W punkcie 8.1.6 opisujemy urządzenia wejściowe zorientowane specjalnie na interakcję 3D. W dalszym ciągu korzystamy z logicznych urządzeń: lokalizatora, klawiatury, urządzenia do wybierania, urządzenia do wprowadzania wartości i urządzenia do wskazywania, używanych przez SRGP, SPHIGS i inne podprogramy pakietów graficznych zależnych od urządzenia. Omawiamy również podstawowe elementy interfejsu użytkownika: metody interakcyjne i zadania interakcyjne. *Metody interakcyjne* są to sposoby wykorzystania urządzeń wejściowych do wprowadzenia informacji do komputera; *zadania interakcyjne* klasyfikują podstawowe typy informacji wprowadzanej metodami interakcyjnymi. Metody interakcyjne są to podstawowe elementy, z których korzysta interfejs użytkownika.

Zadanie interakcyjne polega na wprowadzeniu jednostki informacji przez użytkownika. Cztery podstawowe zadania interakcyjne to: *pozycjonowanie*, *tekst*, *wybór* i *określenie ilościowe*. Jednostką informacji wprowadzaną w zadaniu pozycjonowania jest oczywiście pozycja. Podobnie zadanie związane z tekstem daje ciąg znaków; zadanie wyboru identyfikuje obiekt; zadanie określania ilościowego daje w rezultacie wartość liczbową. Dla danego zadania interakcyjnego można użyć wielu różnych metod interakcyjnych. Na przykład zadanie wyboru może być wykonane za pomocą myszki wskazującej element w menu, za pomocą klawiatury do wprowadzenia nazwy wyboru, przez naciśnięcie klucza funkcyjnego albo za pomocą urządzenia do rozpoznawania mowy. Podobnie jedno urządzenie może być użyte do różnych zadań: myszka jest często używana zarówno do pozycjonowania, jak i do wybierania.

Zadania interakcyjne różnią się od logicznych urządzeń wejściowych omówionych we wcześniejszych rozdziałach. Zadania interakcyjne są określone tym, co użytkownik wykonuje, podczas gdy logiczne urządzenia wejściowe klasyfikują, jak zadanie jest wykonywane przez program użytkowy i pakiet graficzny. Problem interakcji jest związany z użytkownikiem, a logiczne urządzenia wejściowe są ważne dla programisty i pakietu graficznego.

Wiele z zagadnień omawianych w tym rozdziale jest omówionych dokładniej w innych opracowaniach; warto zajrzeć do książek autorstwa: Baeckera i Buxona [BAEC87], Hutchinsa, Hollana i Normana [HUTC86], Mayhewa [MAYH90], Normana [NORM88], Rubensteina

i Hersha [RUBE84], Shneidermana [SHNE86] i Foleya [FOLE90]; książki referencyjnej napisanej przez Salvendy'ego [SALV87]; pracy przeglądowej napisanej przez Foleya, Wallace'a i Chana [FOLE84].

8.1. Sprzęt dla interakcji

Wprowadzimy tu kilka urządzeń przeznaczonych dla zastosowań interakcyjnych nie omówionych w p. 4.5, pokażemy, jak one działają, i omówimy zalety i wady różnych urządzeń. Prezentacja jest oparta na klasyfikacji urządzeń logicznych z p. 4.5 i może być traktowana jako bardziej szczegółowa kontynuacja tego punktu.

Zalety i wady różnych urządzeń dla celów interakcji można omawiać na trzech poziomach: urządzenia, zadania i dialogu (to jest sekwencji kilku zadań interakcji). *Poziom urządzenia* koncentruje się na charakterystyce sprzętu i nie zajmuje się aspektami wykorzystania urządzenia pod kontrolą programów. Na przykład na poziomie urządzenia możemy zauważyć, że jeden kształt myszki może być wygodniejszy niż inny i że tabliczka zajmuje więcej miejsca niż drążek sterowniczy.

Na *poziomie zadania* możemy porównywać metody interakcyjne wykorzystujące różne urządzenia do tego samego zadania. Możemy przyjąć, że doświadczeni użytkownicy często szybciej wprowadzają polecenia za pomocą klawiszy funkcyjnych albo klawiatury niż za pomocą wyboru z menu albo że ci użytkownicy mogą szybciej wskazywać wyświetlone obiekty za pomocą myszki niż za pomocą drążka sterowniczego albo klawiszy sterujących kursorem.

Na *poziomie dialogu* rozpatrujemy nie tyle poszczególne zadania interakcyjne co sekwencje takich zadań. Ruchy ręki między urządzeniami zajmują czas: chociaż zadanie pozycjonowania jest na ogół wykonywane szybciej za pomocą myszki niż za pomocą klawiszy sterujących kursorem, korzystanie z kluczy sterujących kursorem może być szybsze niż z myszki, jeżeli ręce użytkownika są już nad klawiaturą i będą nadal nad klawiaturą przy następnych zadaniach po zmianie pozycji kursora.

Istotnymi zagadnieniami na poziomie urządzenia, omawianymi w tym punkcie, są rozmiary urządzenia (zajmowana powierzchnia robocza), zmęczenie operatora i rozdzielczość urządzenia. Inne istotne cechy urządzenia, np. koszt, niezawodność i łatwość eksploatacji, zmieniają się zbyt szybko z postępem technologii, żeby warto je było tutaj omawiać.

8.1.1. Lokalizatory

Lokalizatory można klasyfikować ze względu na trzy niezależne cechy: bezwzględne/względne, bezpośrednie/pośrednie i dyskretne/ciągłe.

Urządzenia bezwzględne, np. tabliczka albo ekran dotykowy, mają ramkę odniesienia albo początek układu współrzędnych i podają pozycję względem tego początku. *Urządzenia względne*, np. myszka, kula śledząca i drążek sterowania prędkością, nie mają bezwzględnego początku i podają tylko zmiany względem poprzedniego położenia. Urządzenie względne może być użyte do określenia dowolnie dużej zmiany położenia: użytkownik może przesunąć myszkę po stole, podnieść ją i umieścić z powrotem w punkcie początkowym i znowu ją poruszyć. Tabliczka może być zaprogramowana tak, żeby zachowywała się jak urządzenie względne: pierwsza pozycja współrzędnych (x, y) odczytana po przejściu pióra od stanu *daleko* do stanu *blisko* (to jest blisko tabliczki) jest odejmowana od wszystkich kolejno odczytywanych współrzędnych, tak żeby uzyskać tylko zmianę kierunku x albo y, która jest dodawana do poprzedniej pozycji (x, y). Ten proces jest kontynuowany dopóty, dopóki pióro znowu nie znajdzie się w stanie *daleko*.

Urządzenia względne nie mogą być użyte bezpośrednio do wprowadzania rysunków w postaci cyfrowej, a urządzenia bezwzględne mogą. Zaletą urządzeń względnych jest to, że program użytkowy może umieścić kursor gdziekolwiek na ekranie.

Za pomocą *urządzenia bezpośredniego* – takiego jak ekran dotykowy – użytkownik wskazuje bezpośrednio na ekranie palcem albo czymś co zastępuje palec; za pomocą *urządzenia pośredniego* – takiego jak tabliczka, myszka albo drążek – użytkownik przesuwa kursor po ekranie korzystając z urządzenia nie będącego na ekranie. W tym ostatnim przypadku trzeba opanować nową formę koordynacji oko-ręka; rozprzestrzenienie się gier komputerowych w domach i w salonach gier stworzyło środowisko, w którym wielu przypadkowych użytkowników opanowało już te umiejętności. Bezpośrednie wskazywanie może jednak spowodować zmęczenie, zwłaszcza wśród przypadkowych użytkowników.

Urządzenie ciągłe to takie, w którym ciągły ruch ręki może utworzyć ciągły ruch kursora. Tabliczki, drążki i myszki to przykłady urządzeń ciągłych; klawisze sterujące kursorem są *urządzeniami dyskretnymi*. Urządzenia ciągłe na ogół umożliwiają bardziej naturalny, łatwiejszy i szybszy ruch kursora niż urządzenia dyskretne. Większość urządzeń ciągłych umożliwia również łatwiejszy ruch w dowolnym kierunku, niż to ma miejsce w przypadku klawiszy sterowania kursorem.

Szybkość pozycjonowania kursora za pomocą urządzenia ciągłego zależy od *stosunku sterowanie/wyświetlanie*, określanego często jako stosunek C/D [CHAP72]; jest to stosunek między ruchem ręki (sterowanie) a ruchem kursora (wyświetlanie). Duża wartość stosunku jest korzystna ze względu na dokładne pozycjonowanie, ale utrudnia szyb-

kie ruchy; mała wartość stosunku jest korzystna ze względu na szybkość, ale nie na dokładność. Na szczęście dla urządzenia z pozycjonowaniem względnym, ten stosunek nie musi być stały i może być zmieniany adaptacyjnie w funkcji szybkości sterowania ruchem. Szybkie ruchy oznaczają, że użytkownik robi duży ruch ręką i jest używana mała wartość stosunku; gdy szybkość maleje, stosunek C/D jest zwiększany. Takie zmienianie stosunku C/D może być tak ustawione, żeby użytkownik mógł używać myszki do dokładnego ustawiania kursora na 15-calowym ekranie bez zmiany położenia przegubu! Podobna technika jest wykorzystywana w pośrednich urządzeniach dyskretnych (klawisze sterowania kursorem): odległość, o jaką przesuwa się kursor w jednostce czasu, jest zwiększana w funkcji czasu przytrzymania wciśniętego klawisza.

W przypadku urządzeń bezpośrednich jest trudno uzyskać precyzyjne pozycjonowanie, jeżeli ramię nie jest podparte i przedłużone w kierunku ekranu. Spróbujmy napisać swoje imię na tablicy w tej pozie i porównajmy wynik ze zwykłym podpisem. Ten problem można złagodzić, jeżeli ekran jest pochylony prawie do poziomu. Urządzenia pośrednie umożliwiają takie pochylenie ręki, żeby mogła się opierać i żeby można było efektywnie wykorzystać precyzyjne sterowanie palcami. Nie wszystkie jednak urządzenia ciągłe pośrednie są jednakowo dobre do rysowania. Spróbujmy napisać swoje imię za pomocą drążka sterowniczego, myszki i pióra tabliczki. Korzystanie z pióra jest szybsze i daje najlepsze rezultaty.

8.1.2. Klawiatury

Z dobrze znanymi klawiaturami typu QWERTY mamy do czynienia od lat. Te klawiatury, o ironio, były pierwotnie zaprojektowane z myślą o zmniejszeniu szybkości pisania, tak żeby uniknąć problemu blokowania się czcionek. Badania wykazały, że nowsza klawiatura Dvořáka [DWOR43], która umieszcza samogłoski i inne często występujące znaki pod spoczynkowymi pozycjami palców, jest nieco szybsza niż klawiatura QWERTY [GREE87]. Klawiatura ta nie rozpowszechniła się. Czasami, gdy użytkownikami nie są zawodowe maszynistki, są używane klawiatury z układem alfabetycznym. Coraz więcej użytkowników korzysta z klawiatury QWERTY i kilka eksperymentów pokazało, że klawiatura alfabetyczna nie ma istotnej przewagi nad klawiaturą QWERTY [HIRS70; MICH71].

Inne problemy dotyczące klawiatur, związane raczej z projektem oprogramowania niż sprzętu, są związane z umożliwieniem użytkownikowi wprowadzania często używanych znaków przestankowych i ko-

rekcyjnych bez konieczności równoczesnego naciskania klawiszy sterujących i przypisywania czynności (takich jak kasowanie), które mogą sprawiać kłopoty przy omyłkowym użyciu, klawiszom odległym od innych często używanych klawiszy.

8.1.3. Urządzenia do wprowadzania wartości

W niektórych urządzeniach do wprowadzania wartości występuje ograniczenie, na przykład przy sterowaniu gałką siły głosu w radioodbiorniku – można ją obracać dopóty, dopóki nie napotka się oporu. Takie urządzenia z ograniczeniem wprowadzają wartości bezwzględne. Potencjometry obrotowe bez ograniczenia kąta obrotu mogą być obracane nieskończoną liczbę razy w dowolnym kierunku. Dla danej wartości początkowej nieograniczony potencjometr może być wykorzystany do wprowadzania wartości bezwzględnych; w przeciwnym przypadku uzyskiwane wartości są traktowane jako wartości względne. Przy zrealizowaniu swego rodzaju echa użytkownik może określić, jaka wartość w danym momencie jest wprowadzana: względna czy bezwzględna. W przypadku korzystania przy wprowadzaniu wartości z potencjometrów suwakowych albo obrotowych można mówić o wartości stosunku C/D omawianego wcześniej w odniesieniu do urządzeń pozycjonujących.

8.1.4. Urządzenia wybierające

Typowym przykładem urządzeń do wybierania są klawisze funkcyjne. Ich położenie ma wpływ na ich użyteczność: klawisze umieszczone na dole monitora są trudniejsze w użyciu niż klawisze umieszczone na klawiaturze albo na specjalnym urządzeniu znajdującyn się w pobliżu. W zastosowaniach, w których ręce użytkownika są zajęte, a często trzeba zwierać klucz, można wykorzystać przełącznik nożny.

8.1.5. Inne urządzenia

Omówimy teraz kilka mniej popularnych, a w wielu przypadkach eksperymentalnych, interakcyjnych urządzeń 2D. Urządzenia do rozpoznawania głosu, które są użyteczne z tego względu, że nie angażują rąk, które mogą być użyte do innych czynności, wykorzystują podejście polegające na rozpoznawaniu wzorów fal głosowych powstających w czasie wymawiania wyrazu. Przebiegi te są zwykle rozdzielane na kilka

różnych zakresów częstotliwości i zmiana w czasie amplitudy przebiegu w każdym z pasm stanowi bazę badania zgodności wzorów. Przy tym badaniu zgodności mogą jednak pojawić się błędy i jest istotne, żeby program użytkowy wykorzystujący urządzenie rozpoznawania głosu zapewniał możliwość wygodnego wykonywania korekcji.

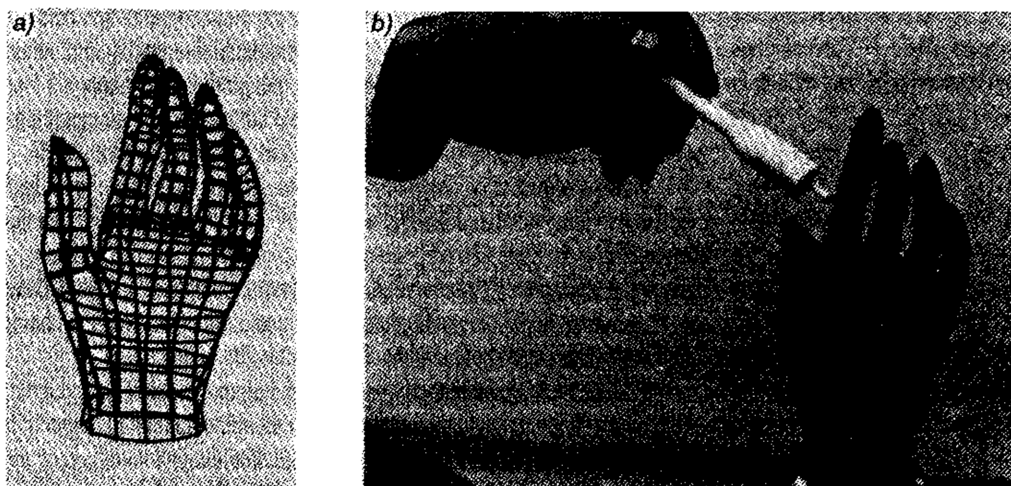
Urządzenia do rozpoznawania głosu różnią się między sobą takimi cechami jak: konieczność uczenia rozpoznawania przebiegów pochodzących od określonej osoby mówiącej, możliwością rozpoznawania ciągłej mowy w przeciwieństwie do rozpoznawania poszczególnych wyrazów lub zdań. Urządzenia uzależnione od osoby mówiącej mają słowniki zawierające cyfry i do 1000 słów.

Tabliczka została poddana różnym modyfikacjom. Wiele lat temu Herot i Negroponte korzystali z eksperymentalnego pióra czułego na nacisk [HERO76]: przy silnym naciśnięciu i wolnym rysowaniu przyjmowano, że użytkownik rysuje odcinek i równocześnie zastanawia się nad tym – w takim przypadku odcinek był rejestrowany dokładnie tak, jak był rysowany; przy niewielkim nacisku i dużej szybkości przyjmowano, że odcinek jest rysowany szybko i był rejestrowany odcinek łączący punkty końcowe. W tabliczce dostępnej od niedawna w handlu [WACO93] wykorzystuje się pióro czułe na siłę nacisku. Otrzymana w wyniku możliwość dysponowania trzema stopniami swobody może być twórczo wykorzystana w różny sposób.

8.1.6. Urządzenia interakcyjne

Niektóre urządzenia interakcyjne 2D zostały już tak zmodyfikowane, że są urządzeniami 3D. Drążki mogą się obracać i jest możliwe uzyskanie trzeciego wymiaru (por. rys. 4.15). Kule śledzące obok obrotów wokół dwóch osi poziomych mogą być czułe na obrót wokół osi pionowej. W obu przypadkach nie ma jednak bezpośredniej zależności między ruchem ręki poruszającej urządzenie a odpowiednim ruchem w przestrzeni 3D.

Kilka urządzeń może rejestrować ruchy ręki w 3D. Na przykład w urządzeniu 3SPACE firmy Polhemus czujnik położenia i orientacji 3D wykorzystuje sprzężenie elektromagnetyczne między trzema antenami nadajników a trzema antenami odbiorników. Cewki anteny nadajnika, które są ustawione pod kątami prostymi względem siebie, tak że tworzą układ współrzędnych kartezjańskich, są pobudzane kolejno. Odbiornik ma trzy podobnie zorientowane anteny; za każdym razem, gdy zostaje pobudzona cewka nadajnika, jest indukowany prąd w każdej z cewek odbiornika. Natężenie prądu zależy od odległości między odbiornikiem a nadajnikiem oraz od względnej orientacji cewek nadajnika

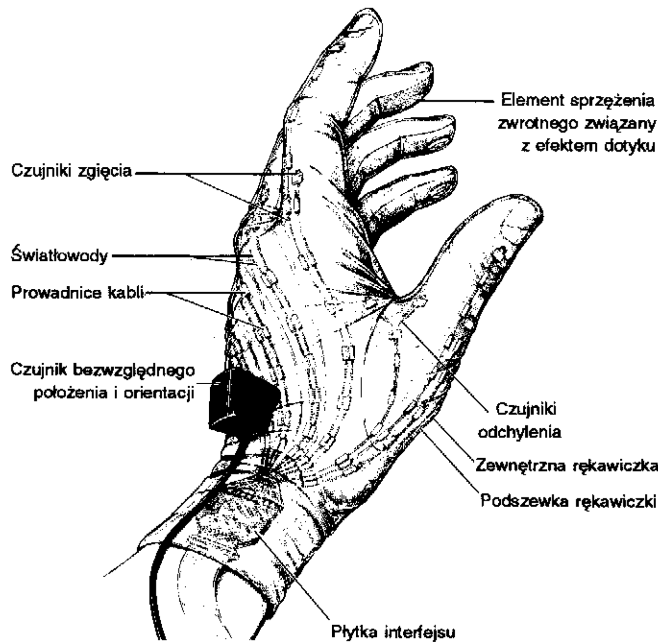


Rys. 8.1. Czujnik położenia 3D firmy Polhemus wykorzystywany do wprowadzenia informacji o obiekcie 3D w postaci cyfrowej (a); wynikowy model szkieletowy (b) (Urządzenie 3SPACE, za zgodą firmy Polhemus, Inc., Colchester, VT.)

i odbiornika. Na rysunku 8.1 pokazano takie urządzenie wykorzystywane w jednym z najczęstszych zastosowań: do wprowadzania danych o obiekcie 3D w postaci cyfrowej.

Rękawiczka typu DataGlove rejestruje położenie ręki i jej orientację oraz ruchy palców. Jak pokazano na rys. 8.2, jest to rękawiczka pokryta małymi lekkimi czujnikami. Każdy czujnik składa się z krótkiego odcinka światłowodu z diodą świecącą (LED) z jednej strony i fototranzystorem z drugiej strony. Powierzchnia kabla jest nierówna w obszarze, gdzie ma on być czuły na zginanie. Gdy kabel zostaje zgięty, traci się pewną ilość światła z diody i fototranzystor odbiera mniejszą ilość światła. Dodatkowo czujniki położenia i orientacji firmy Polhemus rejestrują ruchy ręki. Mając założoną rękawiczkę typu DataGlove użytkownik może chwytać obiekty, przesuwać i obracać je oraz puszczać – w ten sposób jest możliwe naturalne realizowanie interakcji w 3D [ZIMM87]. Koncepcję tę ilustruje fot. 6.

Duży wysiłek skierowano na tworzenie tak zwanej *sztucznej rzeczywistości*; chodzi tu o całkowicie komputerowo generowane środowisko o realistycznym wyglądzie, zachowaniu i metodach interakcji [FOLE87]. W jednej z wersji użytkownik nosi hełm z wmontowanym stereoskopowym urządzeniem wyświetlającym, w celu pokazywania prawidłowych widoków dla lewego i prawego oka, i z czujnikiem Polhemus na głowie umożliwiającym śledzenie pozycji i orientacji głowy po to, żeby można było odpowiednio zmieniać obraz stereo; rękawiczki DataGlove umożliwiają interakcję 3D; mikrofon jest używany do wydawania poleceń głosem. Cały zestaw urządzeń pokazano na fot. 7.



Rys. 8.2. Rękawiczka DataGlove firmy VPL; pokazano światłowody używane do śledzenia ruchów palca oraz czujnik położenia i orientacji firmy Polhemus (Z J. Foley: Interfaces for Advanced Computing, Copyright @ 1987, Scientific American. Wszystkie prawa zastrzeżone.)

Do rejestrowania położenia 3D można zastosować kilka innych technologii. W jednej z nich, wykorzystującej czujniki optyczne, diody LED są montowane na użytkowniku (albo w jednym punkcie, na przykład na końcu palca, albo na całym ciele w celu mierzenia ruchów ciała). Czujniki światła są umieszczane wysoko w rogach małego na wpół zacienionego pomieszczenia, w którym pracuje użytkownik, i kolejno każda dioda jest podświetlana. Czujniki mogą określić płaszczyznę, w której leży dioda; położenie diody jest określane na przecięciu trzech płaszczyzn. (Zwykle jest używany czwarty czujnik na wypadek, gdyby któryś z czujników nie widział diody.) Diody mogą być zastąpione małymi reflektorami umieszczonymi na końcach palców i w innych miejscach ciała; czujniki wychwytują wtedy odbite światło zamiast światła emitowanego przez diody LED.

Krueger [KRUE83] opracował czujnik do rejestrowania ruchów ręki i palców w 2D. Kamera telewizyjna rejestruje ruchy ręki; do określenia obrysu ręki i palców wykorzystuje się metody przetwarzania obrazów zwiększające kontrast i wykrywające krawędzie. Różne położenia palca mogą być interpretowane jako polecenia i użytkownik może chwytać obiekty i manipulować nimi, tak jak na fot. 8. Ta metoda może być rozszerzona na 3D dzięki użyciu kilku kamer.

8.2. Podstawowe zadania interakcyjne

Za pomocą podstawowych zadań interakcyjnych system wprowadza jednostkę informacji, która ma znaczenie w kontekście zastosowania. Jak wielka jest ta jednostka? Na przykład, czy przesunięcie urządzenia pozycjonującego o niewielką odległość wprowadzi jednostkę informacji? Tak, jeżeli nowe położenie jest istotne ze względu na jakąś funkcję programu użytkowego, taką jak zmiana położenia obiektu albo określenie końca odcinka. Nie, jeżeli zmiana położenia jest jedną z sekwencji zmian położenia, gdy użytkownik przesuwa kursor w celu umieszczenia go nad odpowiednią pozycją menu: tutaj jednostką informacji jest wybór menu.

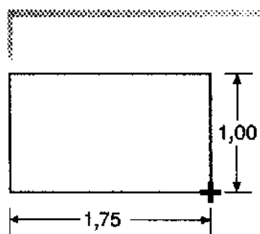
Podstawowe zadania interakcyjne (BIT) są niepodzielne; to znaczy, że gdyby były podzielone na mniejsze jednostki informacji, wówczas te mniejsze jednostki nie miałyby znaczenia dla programu użytkowego. W tym punkcie omawiamy podstawowe zadania interakcyjne. W punkcie 8.3 zajmiemy się złożonymi zadaniami interakcyjnymi (CIT), które są zestawami podstawowych zadań interakcyjnych tutaj opisanych. Gdyby myśleć o podstawowych zadaniach jak o atomach, złożone zadania byłyby cząsteczkami.

Pełny zestaw zadań dla grafiki interakcyjnej obejmuje pozycjonowanie, wybieranie, wprowadzanie tekstu i wprowadzanie wielkości numerycznych. Każde z tych zadań opisano w tym punkcie i dla każdego z nich omówiono kilka z wielu metod interakcyjnych. Jest jednak zbyt wiele metod interakcyjnych, żeby dać tu ich wyczerpujący wykaz i nie możemy założyć, że nie pojawią się nowe metody. Tam, gdzie jest to możliwe, omawiamy zalety i wady poszczególnych metod; należy pamiętać, że pewne metody interakcyjne mogą być dobre w pewnych sytuacjach i złe w innych.

8.2.1. Zadanie interakcyjnego pozycjonowania

Zadanie pozycjonowania polega na określeniu położenia (x, y) albo (x, y, z) dla programu użytkowego. Zwyczajowe metody interakcyjne wykonywania tego zadania wykorzystują albo przesuwanie kursora na ekranie do odpowiedniego miejsca i potem naciśnięcie przycisku, albo wpisanie współrzędnych odpowiedniego miejsca za pomocą rzeczywistej albo symulowanej klawiatury. Urządzenie pozycjonujące może być bezpośrednie albo pośrednie, ciągłe albo dyskretne, bezwzględne albo względne. Dodatkowo polecenie przesuwania kursora może być napisane bezpośrednio na klawiaturze, jako Up (do góry), Left (w lewo) itp.

albo to samo polecenie może być wypowiedziane do jednostki rozpoznawania głosu. Co więcej, metody te mogą być wykorzystywane łącznie – myszka sterująca kursorem może być wykorzystana do zgrubnego pozycjonowania, a klawisze ze strzałkami mogą być użyte do precyzyjnego przesuwania kursora po ekranie w celu precyzyjnego ustalenia położenia.



Rys. 8.3. Numeryczne sprzężenie zwrotne dotyczące wielkości konstruowanego obiektu. Wysokość i szerokość są zmieniane w ślad za ruchami kursora (+) i użytkownik może dobrać odpowiednie dla niego wymiary

Są dwa typy zadań pozycjonownia: przestrzenne i lingwistyczne. W zadaniu pozycjonowania *przestrzennego* użytkownik wie, gdzie jest pożądana pozycja w przestrzennej relacji do sąsiedniego elementu, tak jak przy rysowaniu odcinka między dwoma prostokątami albo centrowaniu obiektu między dwoma innymi. W zadaniu pozycjonowania *lingwistycznego* użytkownik zna wartości numeryczne współrzędnych (x, y) pozycji. W pierwszym przypadku użytkownikowi jest potrzebne sprzężenie zwrotne pokazujące bieżącą pozycję na ekranie; w drugim przypadku są potrzebne współrzędne pozycji. Przy źle rozwiązanym sprzężeniu zwrotnym użytkownik musi sam przechodzić z jednej postaci na drugą. Obie postacie sprzężenia można uzyskać wyświetlając zarówno kursor, jak i jego współrzędne numeryczne (rys. 8.3).

8.2.2. Wybór interakcyjny – zbiór wyborów o zmiennej wielkości

Wybór polega na wybraniu elementu ze zbioru wyborów. Typowe zbiory wyborów to polecenia, wartości atrybutów, klasy obiektów i kopie obiektów. Na przykład menu stylów linii w typowym programie malarskim jest to zbiór wartości atrybutów, a menu typów obiektów (odcinek, okrąg, prostokąt, tekst itd.) w takich programach jest zbiorem klas obiektów. Niektóre metody interakcyjne mogą być użyte do wybrania z dowolnego z tych czterech typów zbiorów wyboru; inne są mniej ogólne. Na przykład wskazywanie na wizualną reprezentację elementu zbioru może służyć do wybrania go niezależnie od tego, jaki to typ zbioru. Chociaż jednak klawisze funkcyjne często działają dobrze przy wybieraniu ze zbioru poleceń, klas obiektów albo atrybutów, trudno jest przypisać oddzielny klawisz każdej kopii na rysunku, ponieważ wielkość zbioru wyboru jest zmienna, często jest duża (większa niż liczba dostępnych klawiszy funkcyjnych) i zmienia się szybko w miarę, jak użytkownik tworzy i usuwa obiekty.

Korzystamy z określeń *zbiór wyborów o* (względnie) *stałej wielkości* i *zbiór wyborów o zmiennej wielkości*. Pierwsze określenie charakteryzuje zbiory wyboru poleceń, atrybutów i klas obiektów, drugie – zbiory wyboru kopii obiektu. Modyfikator względnie rozpoznaje, że któryś z tych zbiorów może się zmienić wówczas, gdy zostaną określone nowe polecenia, atrybuty albo klasy obiektów (takie jak symbole w systemie ryso-

wania). Ale wielkość zbioru nie zmienia się często i zazwyczaj niewiele. Z drugiej strony zbiory wyborów o zmiennych wielkościach mogą stać się duże i mogą się często zmieniać.

W tym punkcie omawiamy metody, które szczególnie dobrze nadają się do potencjalnie dużych zbiorów wyborów o zmiennych wielkościach; obejmuje to nazywanie i wskazywanie. W punkcie 8.2.3 omawiamy metody wyboru nadające się szczególnie dobrze do zbiorów wyboru o (względnie) stałej wielkości. Takie zbiory na ogół są małe; wyjątkiem są duże (ale o względnie stałych wymiarach) zbiory poleceń występujące w złożonych zastosowaniach. Omawiane metody obejmują pisanie albo wymawianie nazwy, skrótu albo innego kodu, który reprezentuje element zbioru; naciśnięcie klawisza funkcyjnego związanego z elementem zbioru (można to traktować jako identyczne z napisaniem jednego znaku na klawiaturze); wskazanie w menu wizualnej reprezentacji (tekstowej albo graficznej) elementu zbioru; przeglądanie zbioru dopóty, dopóki nie zostanie wyświetlony potrzebny element; wykonywanie charakterystycznego ruchu za pomocą ciągłego urządzenia pozycjonującego.

Wybieranie obiektu przez nazwę. Użytkownik może napisać nazwę wybieranego obiektu. Pomysł jest prosty, ale co zrobić, jeżeli użytkownik nie zna nazwy obiektu, co może się zdarzyć, jeżeli są wyświetlane setki obiektów, albo jeżeli użytkownik nie ma potrzeby poznania wszystkich nazw? Niemniej ta metoda jest czasami użyteczna. Po pierwsze, jeżeli użytkownik prawdopodobnie zna nazwy różnych obiektów, tak jak dowódca floty powinien znać nazwy okrętów wchodzących w skład floty, to odwoływanie się do nich przez nazwę jest rozsądne i może być szybsze niż wskazywanie, zwłaszcza gdyby użytkownik musiał przewijać ekran w celu znalezienia potrzebnego obiektu. Po drugie, jeżeli ekran jest tak załadowany, że wybieranie na zasadzie wskazywania jest trudne i jeżeli powiększanie nie jest możliwe (być może ze względu na to, że nie ma wspomagania sprzętowego dla powiększania, a programowe powiększanie jest za wolne), to nazywanie może być jedynym rozwiązaniem. Jeżeli przepełnienie ekranu może być problemem, to użyteczne może być polecenie włączania i wyłączania nazw obiektu.

Pisanie umożliwia wykonywanie wielokrotnego wyboru na zasadzie znaków wieloznacznych albo znaków obojętnych, jeżeli elementy zbioru danych są nazwane w sensowny sposób. Wybór przez nazwę jest najwłaściwszy dla doświadczonych i stałych użytkowników, a nie dla przypadkowych użytkowników rzadko korzystających z programu.

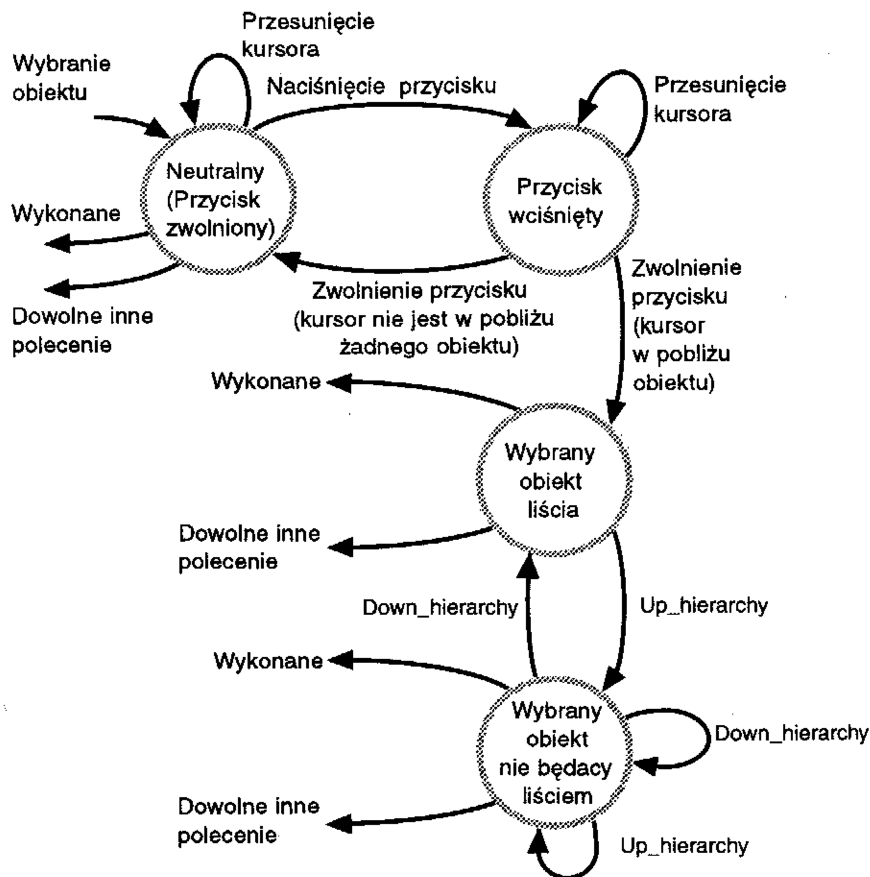
Jeżeli konieczne jest nazywanie metodą pisania, to użyteczną postacią sprzężenia zwrotnego jest wyświetlanie, natychmiast po każdym naciśnięciu klawisza, listy (lub częściowej listy, jeżeli pełna lista jest zbyt długa) nazw w zbiorze wyboru pasujących do dotychczas napisanej sekwencji znaków. Jeżeli użytkownik podał kilka pierwszych znaków, to

obraz może podpowiadać pisownię nazwy. Jak tylko zostanie napisana niedwuznacznie zgodna nazwa, na liście może zostać podświetlona poprawna nazwa. Alternatywnie nazwa może zostać automatycznie skompletowana po napisaniu części jednoznacznie ją identyfikującej. Ta metoda, określana jako *autouzupełnianie*, jest czasami zaskakująca dla nowych użytkowników, należy więc korzystać z niej ostrożnie. Inna strategia wpisywania nazw polega na korygowaniu tego, co zostało napisane (czasami używa się tu określenia *Do What I Mean* albo skrótu DWIM – zrób to o czym myślę). Jeżeli napisana nazwa nie zgadza się z żadną nazwą znaną systemowi, to można użytkownikowi jako alternatywę zaproponować inne nazwy zbliżone do nazwy napisanej. Określenie stopnia bliskości może być tak proste jak poszukiwanie błędów tylko na jednej pozycji albo też może obejmować błędy na wielu pozycjach, albo brak niektórych znaków.

W przypadku urządzenia do rozpoznawania głosu użytkownik może wymawiać nazwę, skrót albo kod. Wejście z wykorzystaniem głosu umożliwia łatwe odróżnienie poleceń od danych: polecenia są wprowadzane głosem, a dane – za pomocą klawiatury albo innych środków. W środowisku klawiatury ta cecha umożliwia wyeliminowanie potrzeby stosowania specjalnych znaków lub trybów dla rozróżnienia danych i poleceń.

Wybieranie obiektów metodą wskazywania. Do wybierania obiektów może być wykorzystana dowolna metoda wskazywania wymieniona we wstępie do p. 8.2; może to polegać na tym, że najpierw się wskazuje, a potem potwierdza (na ogół przez naciśnięcie przycisku), że potrzebny obiekt został wskazany. Ale co zrobić, gdy obiekt ma wiele poziomów hierarchii, tak jak na przykład robot z rozdz. 7? Jeżeli kursor jest nad dłonią robota, to nie wiadomo, czy użytkownik wskazuje dłoń, ramię czy całego robota. Polecenia takie jak Select_robot i Select_arm mogą być wykorzystane do określenia poziomu hierarchii. Jeżeli jednak poziom, na którym pracuje użytkownik, zmienia się rzadko, to użytkownik będzie mógł pracować szybciej z niezależnym poleceniem, np. Set_selection_level, użytym do zmiany poziomu hierarchii.

Inne podejście jest potrzebne, jeżeli liczba poziomów hierarchii nie jest znana projektantowi systemu i potencjalnie jest duża (tak jak w systemach rysowania, gdzie symbole są tworzone z prymitywów graficznych i innych symboli). Są potrzebne przynajmniej dwa polecenia użytkownika: Up_hierarchy i Down_hierarchy. Gdy użytkownik coś wybiera, wówczas system podświetla widziany obiekt najniższego poziomu. Jeżeli jest to to, co jest potrzebne użytkownikowi, to może on przejść dalej. Jeżeli nie, to użytkownik wydaje pierwsze polecenie: Up_hierarchy. Zostaje podświetlony cały obiekt pierwszego poziomu,



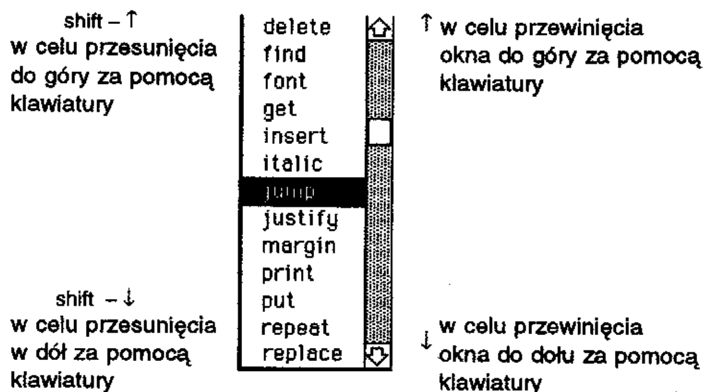
Rys. 8.4. Diagram stanów dla metody wyboru obiektu przy dowolnej liczbie poziomów hierarchii. Polecenia Up i Down służą do przechodzenia w górę i w dół hierarchii. W stanie „wybrany obiekt liścia" polecenie Down_hierarchy nie jest dostępne. Użytkownik wybiera obiekt wskazując go kursorem i naciskając, a potem zwalniając przycisk

którego częścią jest wykryty obiekt. Jeżeli nie jest to to, czego oczekiwał użytkownik, idzie on znowu do góry i zostaje podświetlona większa część obrazu. Jeżeli użytkownik przejdzie zbyt wysoko w hierarchii, to zmienia się kierunek za pomocą polecenia Down_hierarchy. Dodatkowo, jeżeli zejdzie się zbyt głęboko w hierarchii, to przydatne może być polecenie Return_to_lowest_level; podobnie może być przydatny schemat hierarchii w innym oknie pokazujący, gdzie w hierarchii jest umieszczony bieżący wybór. Diagram stanów z rys. 8.4 pokazuje jedno z podejść do wyboru hierarchicznego. Alternatywnie, jedno polecenie, powiedzmy Move_up_hierarchy, umożliwia przeskoczenie z powrotem do oryginalnie wybranego węzła liścia po tym, gdy zostanie osiągnięty węzeł korzenia.

8.2.3. Wybór interakcyjny – zbiór wyborów o względnie stałej wielkości

Wybór z menu to jedna z najlepszych metod wybierania ze zbioru o względnie stałej wielkości. Tutaj omawiamy kilka kluczowych aspektów projektowania menu.

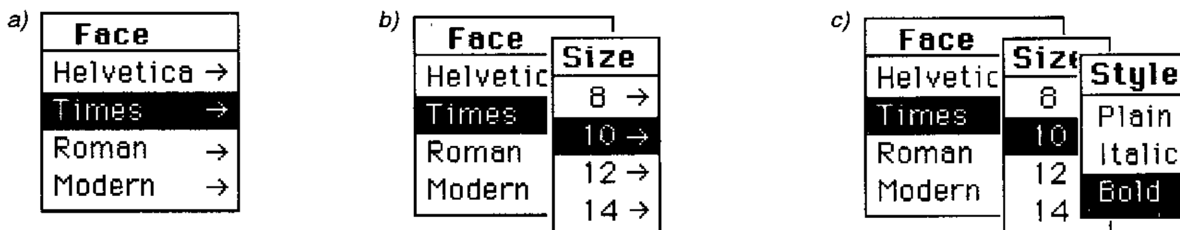
Projekt jednopoziomowy a poziom hierarchiczny. Jedna z podstawowych decyzji w projektowaniu menu powstaje wówczas, gdy zbiór wyborów jest zbyt duży na to, żeby mógł być wyświetlony w całości. Takie menu może być podzielone na logicznie uporządkowaną hierarchię albo przedstawione jako liniowa sekwencja wyborów, która może być przeglądana stronami albo przewijana na ekranie. Pasek przewijania, używany w wielu systemach zarządzania oknami, umożliwia przedstawianie w zwarty sposób wszystkich poleceń przewijania albo stronicowania. Używając klawiatury można korzystać z szybszego rozwiązania; na przykład klawisze ze strzałkami mogą być używane do przewijania okna, a klawisz shift może być razem z klawiszami ze strzałkami wykorzystany do przesuwania wskaźnika wyboru w obrębie widocznego okna (rys. 8.5).



Rys. 8.5. Menu z oknem przewijania. Użytkownik steruje przewijaniem wybierając strzałki wskazujące do góry albo w dół lub przesuwając prostokąt w pasku przewijania

Przy menu hierarchicznym użytkownik najpierw wybiera ze zbioru na górze hierarchii, co daje dostęp do drugiego wyboru. Proces jest powtarzany dopóty, dopóki nie zostanie wybrany węzeł liścia (to znaczy element samego zbioru wyborów) hierarchii drzewa. Podobnie jak przy hierarchicznym wyborze obiektu, trzeba zapewnić mechanizm naprowadzania, tak żeby użytkownik mógł wrócić w górę hierarchii, jeżeli zostanie wybrane niewłaściwe poddrzewo. Potrzebne jest również wizualne sprzężenie zwrotne po to, żeby dać użytkownikowi pewne wrażenie położenia w hierarchii.

Menu hierarchiczne mogą być prezentowane różnymi sposobami. Oczywiście kolejne poziomy hierarchii mogą być pokazywane w miarę wykonywania kolejnych wyborów, ale to nie daje użytkownikowi wielkiego pojęcia o położeniu w hierarchii. *Hierarchia kaskadowa* (rys. 8.6) jest bardziej atrakcyjna. Trzeba odsłonić odpowiednią część każdego menu, tak żeby była widoczna cała ścieżka wyboru; są również potrzebne pewne środki do wskazywania, czy element menu jest węzłem liściem czy nazwą menu niższego poziomu (na rysunku funkcję tę spełnia strzałka skierowana w prawo). Inne rozwiązanie polega na tym, że pokazuje się nazwę każdego dokonanego dotychczas wyboru przy schodzeniu w dół hierarchii plus wszystkie wybory dostępne na bieżącym poziomie.



Rys. 8.6. Chwilowe menu hierarchiczne: a) pierwsze menu pojawia się tam, gdzie jest kursor, w odpowiedzi na wciśnięcie przycisku; kursor może być przesunięty do góry albo w dół w celu wybrania odpowiedniego kroju pisma; b) kursor jest przesuwany w prawo w celu wyświetlenia drugiego menu; c) proces jest powtarzany dla trzeciego menu

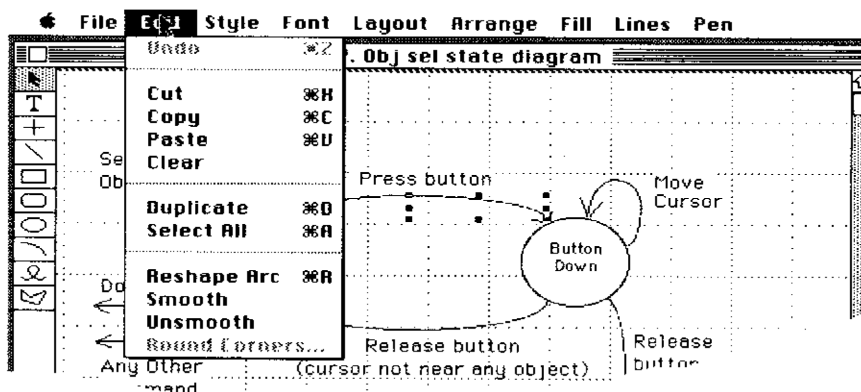
Jeśli projektujemy menu hierarchiczne, to zawsze pojawia się problem kompromisu między głębokością a szerokością. Snowberry [SNOW83] stwierdził doświadczalnie, że czas wyboru i dokładność poprawiają się, gdy korzysta się z szerszego menu o mniejszej liczbie poziomów wyboru. Podobne wyniki podali Landauer oraz Nachbar [LAND85] i inni badacze. Niemniej te wyniki niekoniecznie dają się uogólnić na hierarchie menu, którym brak naturalnej zrozumiałej struktury.

Wybór z menu hierarchicznego zawsze wymaga użycia klawiatury albo klawiszy funkcyjnych do przyspieszenia wyboru przez bardziej doświadczonych użytkowników. Jest to łatwe, jeżeli każdy węzeł w drzewie ma jednoznaczną nazwę, tak że użytkownik może bezpośrednio wprowadzać nazwę, a system menu zapewnia pomoc na wypadek, gdyby użytkownika zawiodła pamięć. Jeżeli nazwy są jednoznaczne tylko w obrębie każdego poziomu hierarchii, to użytkownik musi pisać pełną nazwę ścieżki do potrzebnego węzła liścia.

Rozmieszczenie menu. Menu pokazane na ekranie może być statyczne i widoczne stale albo może pokazywać się dynamicznie na żądanie (menu typu odrywanego, pojawiającego się, chwilowego, rozwijanego albo wyciąganego).

Menu chwilowe pojawia się na ekranie wtedy, kiedy ma być dokonany wybór albo w odpowiedzi na bezpośrednią akcję użytkownika (na ogół naciśnięcie przycisku myszki albo przycisku pióra tabliczki), albo automatycznie, ponieważ następny krok dialogu wymaga wyboru z menu. Menu pojawia się zwykle w miejscu, gdzie jest kursor, które jest w centrum uwagi użytkownika i dzięki temu zachowuje się ciągłość wzrokową. Atrakcyjną cechą menu chwilowego jest początkowe podświetlenie ostatnio dokonanego wyboru ze zbioru wyborów, jeżeli jest większe prawdopodobieństwo wybrania po raz drugi tego samego elementu co ostatnio; w takich sytuacjach tak ustawia się menu, żeby kursor wskazywał tę pozycję.

Menu chwilowe i inne pojawiające się menu umożliwiają oszczędzanie cennej powierzchni ekranu – jednej z cenniejszych rzeczy dla projektanta interfejsu użytkownika. Ich użycie jest ułatwione dzięki operacji RasterOp, tak jak to omówiono w rozdz. 2.

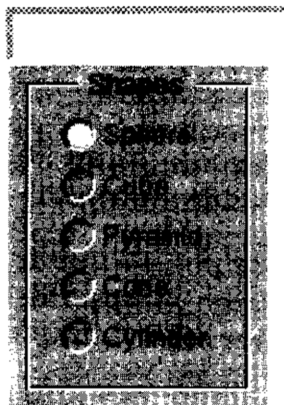


Rys. 8.7. Rozwijane menu w komputerze Macintosh. Ostatnia pozycja menu jest szara a nie czarna, co sygnalizuje, że nie jest ona obecnie dostępna dla wyboru (obecnie wybrany obiekt – łuk – nie ma rogów, które można by zaokrąglać). Polecenie Undo jest również szare, ponieważ poprzednio wykonana operacja nie może być cofnięta. Skróty oznaczają przyspieszające klawisze dla doświadczonych użytkowników. (Prawa autorskie 1988 Claris Corporation. Wszystkie prawa zastrzeżone.)

W przeciwieństwie do menu chwilowych menu rozwijane jest zaczepione w pasku menu znajdującym się na górze ekranu. Wszystkie popularne interfejsy graficzne użytkownika – Apple Macintosh, Microsoft Windows, OPEN LOOK i Motif – używają menu rozwijanych. Menu stosowane w komputerach Macintosh, pokazane na rys. 8.7, również ilustruje wykorzystanie klawiszy przyspieszających i czułości kontekstowej.

Bieżący wybór. Jeżeli w systemie wykorzystano koncepcję bieżąco wybranego elementu ze zbioru wyborów, to wybór z menu wiąże się z podświetleniem tego elementu. W niektórych przypadkach system

zapewnia początkowe domniemane ustawienie, które jest używane dopóty, dopóki użytkownik nie zmieni tego. Bieżąco wybrany element może być pokazany na wiele sposobów. Jeden ze sposobów to metoda interakcji za pomocą przycisków radiowych, stosowaną na wzór przycisków wybierających w odbiornikach samochodowych (rys. 8.8). I znowu, niektóre z menu chwilowych podświetlają ostatnio wybrany element i umieszczają go tam, gdzie jest kursor, przy założeniu, że użytkownik z większym prawdopodobieństwem wybierze ten element niż któryś inny.



Rys. 8.8. Metoda przycisków radiowych do wybierania ze zbioru wzajemnie wykluczających się wyborów. (Za zgodą NeXT, Inc. © 1989 NeXT, Inc.)

Wielkość i kształt elementów menu. Dokładność wskazywania i szybkość zależą od wielkości każdego indywidualnego menu. Większe elementy wybiera się szybciej, tak jak to wynika z prawa Fittsa [FITT54; CARD83]; mniejsze elementy zajmują mniej miejsca i umożliwiają wyświetlenie większej liczby pozycji menu na ustalonej powierzchni, ale powodują większą liczbę błędów w czasie wyboru. Jest więc konflikt między korzystaniem z małego menu w celu oszczędzania powierzchni ekranu a wykorzystywaniem większego menu w celu zmniejszenia czasu wyboru i zmniejszenia liczby błędów.

Rozpoznawanie wzorca. W metodzie wyboru wykorzystującej rozpoznawanie wzorca użytkownik wykonuje sekwencję ruchów za pomocą urządzenia z ciągłym pozycjonowaniem, takiego jak myszka albo tabliczka. Układ rozpoznawania wzorców automatycznie porównuje sekwencję ze zbiorem określonych wzorców, z których każdy odpowiada elementowi zbioru wyborów. Potencjalnymi kandydatami dla takiego podejścia są znaki korektorskie oznaczające usunięcie, duże litery, przesunięcie itd. [WOLF87].

Ostatnie postępy w zakresie algorytmów rozpoznawania znaków doprowadziły do pojawienia się systemów operacyjnych opartych na piórze i komputerach notatnikowych, np. Newton firmy Apple. Wzorce są wprowadzane za pomocą tabliczki i są rozpoznawane i interpretowane jako polecenia, liczby i litery.

Klucze funkcyjne. Elementy zbioru wyboru mogą być powiązane z klawiszami funkcyjnymi. (Możemy myśleć o wejściach związanych z naciśnięciem jednego klawisza zwykłej klawiatury jak o klawiszach funkcyjnych.) Niestety nigdy nie będzie dostępnych dostatecznie dużo klawiszy! Klawisze mogą być używane w trybie wyboru hierarchicznego i ich znaczenie może być zmienione za pomocą akordów, powiedzmy przez naciśnięcie klawiszy shift i sterujących razem z klawiszem funkcyjnym. Na przykład program Word firmy Microsoft na komputerze Macintosh wykorzystuje „shift-option->)” w celu zwiększenia wielkości punktu symetrycznie „shift-option-<” do zmniejszenia wielkości punktu; „shift-option-I” zamienia zwykły tekst na drukowany kursywą i drukowany kursywą na zwykły; podobnie jest z kombinacją „shift-option-U” w stosunku do tekstu podkreślonego.

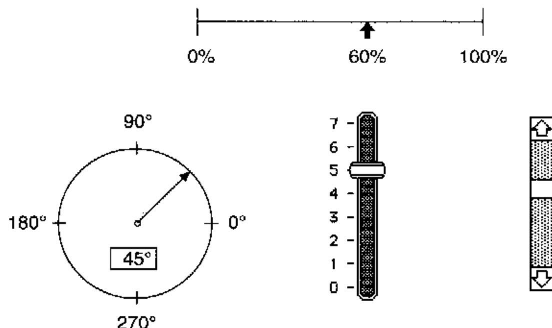
8.2.4. Wybór interakcyjny metodą wprowadzania tekstu

Wybór interakcyjny metodą wprowadzania tekstu powoduje wprowadzenie ciągu znaków, którym program użytkowy nie przypisuje żadnego specjalnego znaczenia. Tak więc napisanie nazwy polecenia nie jest zadaniem wprowadzania tekstu. Dla kontrastu napisanie opisu do grafu i napisanie tekstu w procesorze tekstu są zadaniami wprowadzania tekstu. Oczywiście najpowszechniejszą metodą wprowadzania tekstu jest korzystanie z klawiatury QWERTY.

8.2.5. Wybór interakcyjny metodą wprowadzania wartości

Wybór interakcyjny metodą wprowadzania wartości polega na określeniu wartości numerycznej między pewnymi wartościami minimalną i maksymalną. Typowe metody interakcyjne polegają na napisaniu wartości, ustawieniu tarczy na określonej wartości i wykorzystaniu licznika zliczającego w przód i do tyłu. Podobnie jak w zadaniu pozycjonowania, zadanie to może być albo lingwistyczne, albo przestrzenne. W przypadku zadania lingwistycznego użytkownik zna wartość, która ma być wprowadzona; gdy jest to przypadek przestrzenny, użytkownik dąży do zwiększenia albo zmniejszenia wartości o pewną wielkość, mając – być może – przybliżony pogląd na to, jaka powinna być wartość końcowa. W pierwszym przypadku metoda interakcji w oczywisty sposób musi zapewniać numeryczne sprzężenie zwrotne dla wybieranej wartości (jeden ze sposobów wykonania tego polega na napisaniu przez użytkownika bieżącej wartości); w drugim przypadku ważniejsze jest zapewnienie ogólnego wrażenia o przybliżonym ustawieniu wartości. Zazwyczaj wykonuje się to za pomocą metody sprzężenia zorientowanej przestrzennie, takiej jak na przykład wyświetlenie tarczy albo wskaźnika, na którym jest pokazana bieżąca (i być może poprzednia) wartość.

Wartości można wprowadzać korzystając z potencjometru. Decyzja o tym, czy powinno się używać potencjometru obrotowego czy liniowego, powinna brać pod uwagę to, czy wzrokowe sprzężenie zmiany wartości jest typu obrotowego (na przykład obracająca się wskazówka zegara), czy liniowego (na przykład wskaźnik rosnącej temperatury). Bieżąca pozycja jednego albo grupy potencjometrów suwakowych jest znacznie łatwiejsza do szybkiego zinterpretowania niż w przypadku potencjometrów obrotowych, nawet jeżeli pokrętła mają wskaźniki. Potencjometry obrotowe są jednak łatwiejsze do regulacji. Mając do dyspozycji potencjometry liniowe i obrotowe użytkownik może łatwiej przypisać odpowiednie znaczenia. Ważne jest, żeby używać konsekwentnie kierunków: na ogół obrót w kierunku ruchu wskazówek zegara czy ruch do góry odpowiadają zwiększaniu wartości.



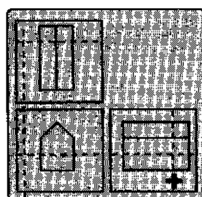
Rys. 8.9. Kilka tarcz, które użytkownik może wykorzystać do wprowadzania wartości metodą przesuwania wskaźnika sterującego. Sprzężenie zwrotne zapewnia wskaźnik i w dwóch przypadkach wartości numeryczne. (Vertical sliders © Apple Computer, Inc.)

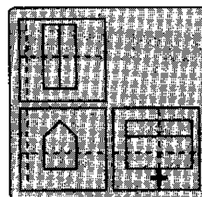
W przypadku skali ciągłej użytkownik wskazuje na bieżącą wartość na wyświetlonym wskaźniku albo na skali, naciska przycisk wyboru, przesuwa wskaźnik wzdłuż skali do pożądanej wartości i potem zwalnia naciśnięty przycisk. Typowo do wskazania wartości wybranej na skali używa się wskaźnika; można również wprowadzić numeryczne echo. Na rysunku 8.9 pokazano kilka takich tarcz i związane z nimi sprzężenia zwrotne.

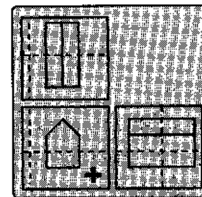
8.2.6. Zadania interakcyjne 3D

Dwa spośród czterech zadań interakcyjnych opisanych wcześniej dla zastosowań 2D komplikują się w 3D: pozycjonowanie i wybieranie. W pierwszej części tego punktu opisano metody pozycjonowania i wybierania, które są ze sobą ściśle związane. W tym punkcie wprowadzimy również dodatkowe zadanie interakcyjne 3D: obrót (w sensie orientowania obiektu w przestrzeni 3D). Zasadniczą przyczyną komplikacji jest trudność postrzegania w 3D relacji głębokości kursora albo obiektu względem innych wyświetlanych obiektów. To kontrastuje silnie z interakcją 2D, gdzie użytkownik może łatwo zauważyć, czy kursor jest ponad, obok czy na obiekcie. Druga komplikacja jest związana z tym, że powszechnie dostępne urządzenia interakcyjne, np. myszka i tabliczka, są tylko urządzeniami 2D, a istnieje potrzeba odwzorowania ruchów tych urządzeń 2D w 3D.

Wyświetlanie par stereo, odpowiadających rzutom dla lewego i prawego oka, jest użyteczne dla zrozumienia ogólnych zależności związanych z głębokością, ma jednak ograniczoną dokładność jako precyzyjna metoda lokalizacyjna. Metody prezentowania par stereo są omawiane w rozdz. 12 i w pracy [HODG85]. Inne sposoby pokazania zależności związanych z głębokością są omawiane w rozdz. 12 do 14.


Nacisnąć przycisk, gdy kursor 2D jest na przerywanej linii kursora 3D


Przeciągnąć kursor 3D; wszystkie rzuty są odpowiednio uaktualniane

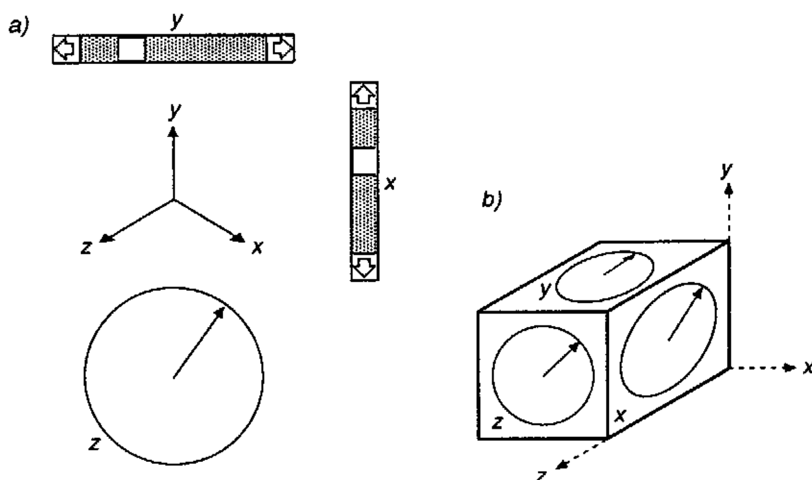

Zwolnić przycisk; kursor 2D nie steruje już kursorem 3D

Rys. 8.10. Metoda pozycjonowania 3D wykorzystująca trzy rzuty tej samej sceny (dom). Do wybrania jednej z przerywanych linii kursora 3D jest wykorzystywany kursor 2D (+)

Na rysunku 8.10 pokazano typowy sposób pozycjonowania 3D. Kursor 2D, będący pod kontrolą, powiedzmy myszki, porusza się swobodnie między tymi trzema rzutami. Użytkownik może wybrać dowolną z linii przerywanych kursora 3D i może przeciągnąć linię korzystając z sekwencji naciśnięcie przycisku-przesunięcie-zwolnienie przycisku. Jeżeli przycisk zostanie naciśnięty wtedy, kiedy kursor jest w pobliżu przecięcia dwóch przerywanych linii kursora, to obie zostają wybrane i są przesuwane za pomocą myszki. Chociaż ta metoda może się wydawać restrykcyjna w tym sensie, że zmusza użytkownika do pracy jednocześnie w jednym albo w dwóch wymiarach, niekiedy jest wygodniej zdekomponować zadanie manipulacji w 3D na prostsze zadania w przestrzeniach niższego stopnia. Wybieranie, podobnie jak lokalizowanie, jest ułatwione wówczas, gdy są pokazane różne rzuty: obiekty, które nakładają się i dlatego są trudne do rozróżnienia w jednym rzucie, mogą nie nakładać się w innym rzucie.

Podobnie jak przy lokalizowaniu i wybieraniu, dla obrotów w 3D istotne są relacje głębokości, odwzorowanie urządzeń interakcyjnych 2D na 3D i zapewnienie zgodności pobudzenie-odpowiedź (zgodność S-R)[1]. Łatwa do implementacji metoda obrotu wykorzystuje tarcze obrotowe albo wskaźniki, które sterują obrotem wokół trzech osi. Zgodność S-R sugeruje, że trzy osie powinny być w układzie współrzędnych ekranu – x na prawo, y do góry i z przed (albo za) ekranem [BRIT78]. Oczywiście środek obrotu musi być albo określony bezpośrednio w niezależnym kroku, albo musi być określony pośrednio (na ogół początek układu współrzędnych, początek obiektu albo środek obiektu). Wykonanie obrotu wokół osi x i y ekranu jest wyjątkowo proste, tak jak pokazano to

[1] Zasada, która mówi, że odpowiedzi systemu na akcję użytkownika muszą być w tym samym kierunku i o tej samej orientacji i że wielkość odpowiedzi powinna być proporcjonalna do akcji.



Rys. 8.11. Dwa rozwiązania dla obrotów 3D: a) dwa potencjometry wpływające na obrót wokół osi ekranu *x* i *y* i tarcza dla obrotu wokół osi ekranu *z*; układ współrzędnych reprezentuje układ współrzędnych świata i pokazuje, jak współrzędne są powiązane ze współrzędnymi ekranu; b) trzy tarcze do sterowania obrotem wokół trzech osi; umieszczenie tarcz na sześcianie daje silną zgodność pobudzenie-odpowiedź

na rys. 8.11a. System współrzędnych (x, y, z) związany z suwakami jest obracany w ślad za przesuwaniem suwaka. Rozwiązanie z dwiema osiami można łatwo uogólnić na rozwiązanie z trzema osiami dodając tarczę dla obrotu wokół osi z (preferowana jest tarcza w stosunku do suwaka, ze względu na zgodność S-R). Jeszcze większą zgodność uzyskuje się dla kombinacji: tarcze na ścianach sześcianu (rys. 8.11b); tutaj wyraźnie widać, która oś jest sterowana którą tarczą. Zamiast tarcz można użyć kuli śledzącej 3D.

Często trzeba łączyć zadania interakcyjne 3D. I tak przy obrocie jest potrzebne zadanie wyboru obiektu, który ma być obracany, zadanie pozycjonowania do określenia środka obrotu i zadanie orientowania do wykonania samego obrotu. Określanie rzutu 3D może być traktowane jako połączenie zadań pozycjonowania (gdzie jest oko), orientacji (jak oko jest zorientowane) i skalowania (pole wizualizacji albo jak duża część rzutni jest odwzorowywana na pole wizualizacji). Możemy utworzyć takie zadanie łącząc kilka metod już przez nas omówionych albo tworząc możliwość latania, kiedy użytkownik lata wyimaginowanym samolotem po świecie 3D. Kontroli podlegają na ogół kąty pochylenia, przechylenia i azymutu oraz prędkość w celu umożliwienia przyspieszania albo zwalniania. Przy koncepcji latania użytkownikowi jest potrzebny ogólny plan sytuacyjny, taki jak widok 2D pokazujący położenie samolotu oraz kierunek lotu.

8.3. Złożone zadania interakcyjne

Złożone zadania interakcyjne (CIT) są tworzone na bazie podstawowych zadań interakcyjnych (BIT) opisanych w poprzednim punkcie i są kombinacjami zadań podstawowych połączonych w całość. Są trzy podstawowe formy składania zadań: pola dialogowe wykorzystywane do określania wielu jednostek informacji; konstrukcje używane do tworzenia obiektów wymagających dwóch lub więcej pozycji; manipulacje wykorzystywane do zmiany kształtów istniejących obiektów geometrycznych.

8.3.1. Pola dialogowe

Często musimy wybrać kilka elementów ze zbioru wyborów. Na przykład takie atrybuty tekstu jak pochylenie, pogrubienie, podkreślenie, wypełnienie wnętrza albo duże litery wzajemnie nie wykluczają się i użytkownik może chcieć jednocześnie wybrać dwa z nich lub więcej. Ponadto może być kilka zbiorów odpowiednich atrybutów, np. czcionka albo krój pisma. Niektóre z rozwiązań wykorzystujących menu wygodne przy wybieraniu jednego elementu nie są zadowalające przy wielokrotnych wyborach. Na przykład menu rozwijane albo chwilowe na ogół znikają po dokonaniu wyboru, co stwarza konieczność ponownej aktywacji w celu wykonania drugiego wyboru.

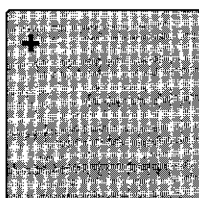
Ten problem można obejść korzystając z pól dialogowych – formy menu, która pozostaje widoczna dopóty, dopóki nie zostanie bezpośrednio usunięta przez użytkownika. Dodatkowo pola dialogowe umożliwiają wybór z więcej niż jednego zbioru wyborów i mogą również zawierać pola do wprowadzania tekstu i wartości. Wybory dokonane w polu dialogowym mogą być natychmiast skorygowane. Gdy cała informacja zostanie wprowadzona do pola dialogowego, wówczas pole na ogół jest usuwane bezpośrednio za pomocą polecenia. Atrybuty i inne wartości określone w polu dialogowym mogą być zastosowane natychmiast, co umożliwia użytkownikowi wstępne obejrzenie wpływu zmiany kroju pisma albo stylu linii.

8.3.2. Metody konstrukcyjne

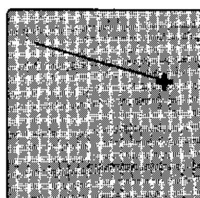
Jeden ze sposobów tworzenia odcinka polega na tym, że użytkownik wskazuje jeden koniec, a potem drugi; po określeniu drugiego końca jest rysowany odcinek między tymi dwoma punktami. Korzystając z ta-

kiej metody użytkownik nie może jednak łatwo wypróbować różnych położeń odcinka przed ostatecznym wyborem położenia, ponieważ odcinek tak naprawdę nie jest rysowany dopóty, dopóki nie zostanie podany drugi punkt końcowy. Przy takim sposobie interakcji użytkownik musi wydawać polecenie za każdym razem, gdy chce zmienić położenie drugiego końca.

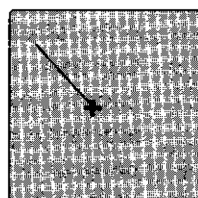
Znacznie lepszym rozwiązaniem jest *metoda gumki* omówiona w rozdz. 2. Gdy użytkownik naciska przycisk (często przycisk w końcówce pióra tabliczki albo przycisk myszki), kursor ustala punkt początkowy odcinka (zazwyczaj, chociaż nie jest to konieczne, za pomocą urządzenia do ciągłego wskazywania położenia). W ślad za ruchami kursora porusza się koniec odcinka; po zwolnieniu przycisku koniec odcinka jest zamrażany. Na rysunku 8.12 pokazano sekwencję rysowania odcinka metodą gumki. Stan, kiedy działa gumka, jest aktywny tylko wówczas, gdy przycisk jest wciśnięty. W tym właśnie stanie ruchy kursora powodują zmiany bieżącego położenia odcinka.



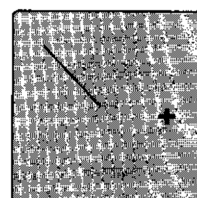
Nacisnąć przycisk; w miejscu wskazywanym przez kursor zostaje zaczepiona gumka



Odcinek jest rysowany między początkowym położeniem kursora a jego bieżącym położeniem



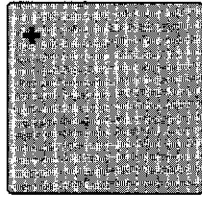
Zwolnić przycisk; kończy się działanie gumki i odcinek zostaje zamrożony



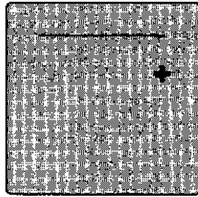
Kursor już nie ma wpływu na odcinek

Rys. 8.12. Rysowanie odcinka metodą gumki

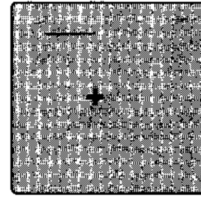
Metoda rysowania odcinków na zasadzie gumki stanowi podstawę grupy metod interakcyjnych. Metoda *rysowania prostokąta* polega na tym, że zaczepia się jeden wierzchołek prostokąta w wyniku naciśnięcia przycisku, a potem przeciwny róg jest dynamicznie łączony z kursorem dopóty, dopóki nie nastąpi zwolnienie przycisku. Diagram stanów dla tej metody różni się od diagramu rysowania odcinka tylko tym, że w wyniku dynamicznego sprzężenia zwrotnego uzyskuje się prostokąt, a nie odcinek. Przy *rysowaniu okręgu* powstaje okrąg o środku w punkcie początkowej pozycji kursora i który przechodzi przez bieżącą pozycję kursora albo który jest w środku kwadratu określonego przez przeciwne wierzchołki. Wszystkie te metody mają wspólną sekwencję akcji użytkownika: naciśnięcie przycisku, przesunięcie lokalizatora i obejrzenie uzyskanego obrazu w wyniku sprzężenia zwrotnego, zwolnienie przycisku.



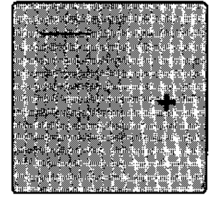
Nacisnąć przycisk; w miejscu wskazywanym przez kursor zostaje zaczepiona gumka



Odcinek jest rysowany między początkowym położeniem kursora a współrzędną x bieżącego położenia kursora



Zwolnić przycisk; kończy się działanie gumki i odcinek zostaje zamrożony



Kursor już nie ma wpływu na odcinek

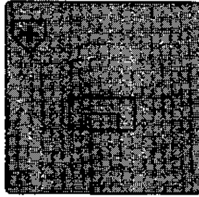
Rys. 8.13. Rysowanie odcinka metodą gumki przy ograniczeniu do położenia poziomego

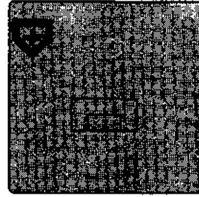
W dowolnej z tych metod można stosować różnego typu ograniczenia odnośnie do położenia kursora. Na przykład na rys. 8.13 pokazano sekwencję odcinków rysowanych za pomocą takich samych położeń kursora jak na rys. 8.12, ale z efektem ograniczania co do poziomu. W ten sposób można również rysować odcinki pionowe lub o innym nachyleniu. Łatwo można tworzyć łamane składające się tylko z odcinków poziomych i pionowych, tak jak w obwodach drukowanych, układach VLSI i niektórych rodzajach map; kąty proste są wprowadzane albo w wyniku polecenia użytkownika, albo automatycznie, gdy użytkownik zmienia kierunek. Ta idea może być uogólniona na dowolne kształty, takie jak koła, elipsy i inne krzywe; określa się początek krzywej, a następnie ruchy kursora określają, jaką część krzywej należy wyświetlić. Ogólnie położenie kursora jest wykorzystywane jako wejście do funkcji ograniczania, której wyjście jest potem wykorzystane do wyświetlenia odpowiedniego fragmentu obiektu.

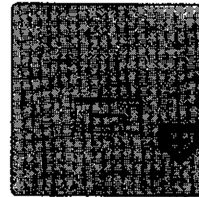
8.3.3. Manipulacje dynamiczne

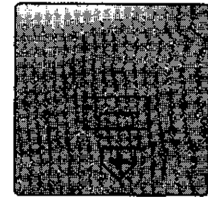
Nie wystarczy utworzyć odcinki, prostokąty itd. W wielu sytuacjach użytkownik musi móc modyfikować poprzednio utworzone obiekty geometryczne.

Przeciąganie powoduje przesunięcie wybranego symbolu z jednej pozycji na inną pod kontrolą kursora (rys. 8.14). Proces przeciągania na ogół rozpoczyna naciśnięcie przycisku (czasami naciśnięcie przycisku jest również stosowane do wybrania symbolu wskazywanego przez kursor, który ma być przesunięty); następnie zwolnienie przycisku zamraża położenie symbolu i dalsze ruchy kursora nie mają już wpływu. Ta sekwencja naciśnięcie przycisku-przeciągnięcie-zwolnienie przycisku jest często określana jako *wybierz i przeciągnij*.

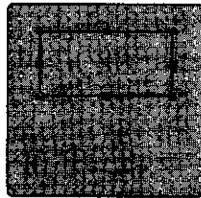

Ustawić kursor nad symbolem, który ma być przesunięty i nacisnąć przycisk

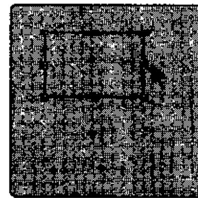

Symbol zostaje podświetlony w celu potwierdzenia wyboru

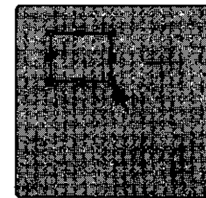

Kilka pośrednich ruchów kursora


Zwolnienie przycisku; symbol zostaje zablokowany

Rys. 8.14. Przeciąganie symbolu do nowego położenia


Wybranie prostokąta za pomocą kursora powoduje pojawienie się uchwytów


Przycisk działający na ten uchwyt przesuwa tylko prawą stronę prostokąta

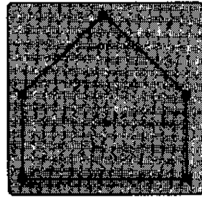

Przycisk działający na ten uchwyt przesuwa tylko wierzchołek prostokąta

Rys. 8.15. Zastosowanie uchwytów do zmieniania kształtu obiektów

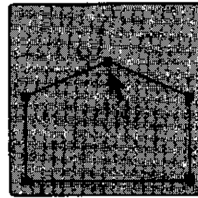
Przy skalowaniu obiektu wygodna jest koncepcja *uchwytów*. Na rysunku 8.15 pokazano obiekt z ośmioma uchwytami, które są wyświetlone jako małe kwadraciki w rogach i na bokach wyimaginowanego prostokąta otaczającego obiekt. Użytkownik wybiera jeden z uchwytów i przeciąga go w celu zmiany skali obiektu. Jeżeli uchwyt jest w narożniku, to przeciwległy narożnik jest nieruchomy. Jeżeli uchwyt jest w środku boku, to przeciwny bok jest nieruchomy.

Jeżeli tę metodę włączy się do kompletnego interfejsu użytkownika, to uchwyty pojawiają się tylko wówczas, gdy obiekt zostanie wybrany w celu wykonania pewnych operacji. Uchwyty służą równocześnie jako jednoznaczny wskaźnik wzrokowy dla zaznaczenia, że obiekt został wybrany; inne ewentualne wskaźniki (na przykład grubość linii, linia przerywana albo zmieniona jasność) mogą być wykorzystane w samym rysunku.

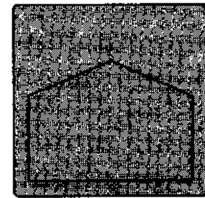
Operacje przesuwania, obrotów i skalowania wpływają na cały obiekt. A co zrobić, jeżeli chcemy przesuwać poszczególne punkty, np. wierzchołki wielokąta? Wierzchołki mogłyby być nazwane i użytkownik



Wielokąt został wybrany w celu modyfikacji wierzchołków; w każdym wierzchołku pojawił się uchwyt



Naciśnięcie-przesunięcie-zwolnienie powoduje ruch wierzchołka



Wielokąt już nie jest wybrany; uchwyty zostały usunięte

Rys. 8.16. Uchwyty wykorzystane do zmiany położenia wierzchołków wielokąta

mógłby wprowadzić nazwę wierzchołka i jego nowe współrzędne (x, y). Ale bardziej atrakcyjna jest ta sama strategia wskaż i przesuń, która była użyta do przesuwania całego obiektu. W tym przypadku użytkownik wskazuje wierzchołek, wybiera go i przesuwa do nowego położenia. Wierzchołki sąsiadujące z wybranym pozostają połączone za pomocą odcinków typu gumka. W celu ułatwienia wyboru odcinka możemy spowodować migotanie wierzchołka wówczas, gdy kursor jest niedaleko, albo możemy na każdy wierzchołek nałożyć uchwyt, tak jak na rys. 8.16. Podobnie, użytkownik może przesunąć krawędź wielokąta, wybierając ją i przesuwając, przy czym krawędź zachowuje swoje oryginalne nachylenie. Dla krzywych i powierzchni gładkich również można wykorzystać uchwyty w celu umożliwienia użytkownikowi manipulowania punktami, które określają kształt, co omówimy w rozdz. 9.

8.4. Narzędzia wspomagające metody interakcji

Wygląd i skuteczność interfejsu użytkownik-komputer jest określona głównie przez zestaw wykorzystanych metod interakcyjnych. Przypomnijmy, że metody interakcyjne implementują część sprzętową projektu interfejsu użytkownik-komputer. Zaprojektowanie i zrealizowanie dobrego zestawu metod interakcyjnych jest czasochłonne: narzędzia wspomagające metody interakcyjne, np. biblioteki podprogramów, są mechanizmami umożliwiającymi realizowanie metod interakcyjnych dostępnych dla programisty związanego z określonym zastosowaniem. To podejście, które zapewnia dobry wygląd i skuteczność programom użytkowym, jest często wykorzystywane przez inżynierów oprogramowania.

Narzędzia wspomagające oprogramowanie metod interakcyjnych mogą być wykorzystywane nie tylko przez programy użytkowe, ale również przez rezydentne programy zarządzania oknami, które są po prostu innymi programami. Korzystanie z tych samych narzędzi w różnych miejscach jest ważnym i często stosowanym sposobem zapewnienia odpowiedniego wyglądu i skuteczności, który gwarantuje jednolitość wielu zastosowań i samego środowiska okien. Na przykład styl menu używany do wybierania operacji związanych z oknami powinien być taki sam jak styl wykorzystywany przez program użytkowy.

Narzędzia mogą być implementowane nad systemem zarządzania oknami [FOLE90]. Jeżeli nie ma systemu okien, to narzędzia mogą być zrealizowane bezpośrednio nad pakietem podprogramów graficznych. Ponieważ jednak do tych narzędzi należą: menu, pola dialogowe, paski przesuwne i inne, które mogą być wygodnie zrealizowane w systemie okien, na ogół jako podłoże jest wykorzystywany system okien. Do powszechnie używanych zestawów narzędzi należą: zestaw dla komputera Macintosh [APPLE85], OSF/Motif [OPEN89] i InterViews [LINT89] dla zastosowań z systemem X Window oraz kilka zestawów wykorzystujących OPEN LOOK [SUN89]. Na fotografii 9 pokazano interfejs OSF/Motif, a na fot. 10 – interfejs OPEN LOOK.

Podsumowanie

Przedstawiliśmy kilka najważniejszych koncepcji związanych z interfejsami użytkownika: urządzenia wejściowe, metody interakcyjne i zadania interakcyjne. Jest jednak znacznie więcej aspektów związanych z interfejsami użytkownika, których nie omówiliśmy. Wśród nich są zalety i wady różnych styli dialogowych – takie jak to co widzisz, to jest to, co dostaniesz (WYSIWYG), języki poleceń i manipulacje bezpośrednie – oraz zagadnienia związane z systemami zarządzania oknami, które wpływają na interfejs użytkownika. W pracy [FOLE90] omówiono dokładnie te zagadnienia.

Zadania

8.1. Przyjrzyj się wykorzystywanemu interfejsowi użytkownika. Wymień wszystkie używane zadania interakcyjne. Zakwalifikuj każde z nich do jednej z czterech grupy podstawowych (BIT) z p. 8.2. Jeżeli nie można się dopasować do tego schematu, to należy wykonać dalszą dekompozycję.

8.2. Rozszerz diagram stanów z rys. 8.4 tak, żeby uwzględnić polecenie „powrót do najniższego poziomu", które sprowadza wybór z powrotem do najniższego poziomu hierarchii, tak że to, co zostało wybrane jako pierwsze, zostaje wybrane ponownie.

8.3. Zrealizuj pakiet menu na kolorowym ekranie rastrowym, który ma tabelę barw taką, że menu jest wyświetlane w silnym, jasnym, ale częściowo przezroczystym kolorze i wszystkie kolory pod menu są zamieniane na stonowane kolory szare.

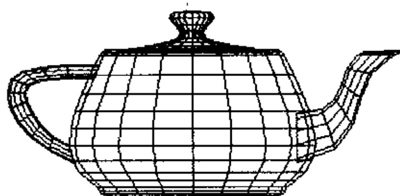
8.4. Zrealizuj dowolną z metod interakcyjnych 3D omówionych w tym rozdziale.

8.5. Narysuj diagram stanów dla sterowania chwilowymi menu hierarchicznymi. Narysuj diagram stanów dla sterowania panelem hierarchicznych menu.

9. Reprezentowanie krzywych i powierzchni

Klasyczny czajniczek, pokazany na rys. 9.1, jest chyba najbardziej znaną ikoną grafiki komputerowej. Od 1975 r., gdy jego model został opracowany przez Martina Newella [CROW87], był używany przez dziesiątki badaczy jako struktura do demonstrowania najnowszych metod wytwarzania realistycznych powierzchni i tekstur. Modelowanie eleganckiego czajniczka wymagało określenia jego kształtu jako zbioru elementów gładkiej powierzchni, znanych jako *płaty bikubiczne*. Gładkie krzywe i powierzchnie muszą być generowane w wielu zastosowaniach grafiki komputerowej. Wiele obiektów świata rzeczywistego to z natury obiekty gładkie i wiele programów grafiki komputerowej jest związanych z modelowaniem świata rzeczywistego. Gładkie krzywe i powierzchnie występują w projektowaniu wspomaganym komputerowo (CAD), w dobrej jakości czcionkach, wykresach i rysunkach artystycznych. W animacji ścieżka kamery albo obiektu jest prawie zawsze gładka; podobnie często gładka musi być ścieżka przechodząca przez przestrzeń barw (rozdz. 12 i 11).

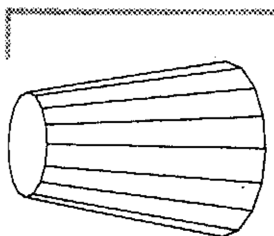
Konieczność reprezentowania krzywych i powierzchni powstaje w dwóch przypadkach: w modelowaniu istniejących obiektów (samochód, twarz albo góra) i w modelowaniu obiektu nie istniejącego.



Rys. 9.1. Klasyczny czajniczek – model składający się ze zbioru gładkich powierzchni krzywoliniowych

W pierwszym przypadku może nie istnieć matematyczny opis obiektu. Oczywiście możemy użyć jako modelu współrzędnych nieskończenie wielu punktów obiektu, ale to podejście nie jest możliwe dla komputera o skończonej pamięci. Znacznie częściej jedynie aproksymujemy obiekt za pomocą kawałków płaszczyzn, powierzchni kuli i innych kształtów, które jest łatwo opisać matematycznie, i wymagamy, żeby punkty naszego modelu były blisko odpowiednich punktów obiektu.

W drugim przypadku, gdy nie istnieje obiekt, który modelujemy, użytkownik tworzy obiekt w procesie modelowania; dlatego obiekt przybliża dokładnie swoją reprezentację, ponieważ stanowi jedynie jej urzeczywistnienie. W celu utworzenia obiektu użytkownik może rzeźbić obiekt interakcyjnie, opisać go matematycznie albo dać przybliżony opis do wypełnienia przez jakiś program. W CAD komputerowa reprezentacja jest później używana do generowania fizycznej realizacji abstrakcyjnie zaprojektowanego obiektu.

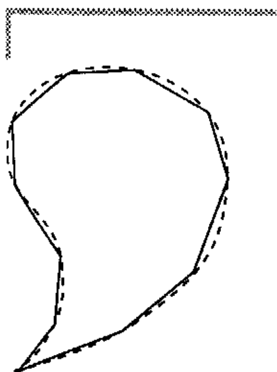


Rys. 9.2. Obiekt 3D reprezentowany za pomocą wielokątów

Ten rozdział wprowadza do ogólnej dziedziny *modelowania powierzchni*. Dziedzina jest szeroka i tu prezentujemy tylko trzy najpopularniejsze reprezentacje powierzchni 3D: siatki wielokątów, powierzchnie parametryczne i powierzchnie drugiego stopnia. Omawiamy również krzywe parametryczne zarówno dlatego, że same w sobie są interesujące, jak i dlatego, że powierzchnie parametryczne są uogólnieniem krzywych.

Modelowanie brył wprowadzone w następnym rozdziale jest reprezentacją objętości całkowicie otoczonych przez powierzchnie takie, jak powierzchnia sześcianu, samolotu albo budynku. Omawiane w tym rozdziale reprezentacje powierzchni mogą być wykorzystane w modelowaniu brył do określania powierzchni otaczających objętość.

Siatka wielokątów jest to zbiór połączonych płaskich powierzchni ograniczonych przez łamane zamknięte. Siatki wielokątów mogą stanowić naturalną reprezentację otwartych skrzynek, pomieszczeń i otoczenia budynków, podobnie jak objętości ograniczonych przez płaskie powierzchnie. Siatki wielokątów mogą być użyte, chociaż nie tak łatwo, do reprezentowania obiektów o powierzchniach krzywoliniowych (rys. 9.2); taka reprezentacja stanowi jednak tylko przybliżenie. Na rysunku 9.3 pokazano przekrój kształtu krzywoliniowego i siatki wielokątowej reprezentującej ten kształt. Możemy dowolnie zmniejszać błędy aproksymacji wykorzystując większą liczbę wielokątów do utworzenia dokładniejszej liniowej aproksymacji, jednak takie podejście zwiększa wymagania co do pamięci i czas wykonywania algorytmów przetwarzania tej reprezentacji. Co więcej, jeżeli obraz zostanie powiększony, to proste krawędzie znowu staną się widoczne.



Rys. 9.3. Przekrój kształtu krzywoliniowego (linia przerywana) i jego reprezentacji wielokątowej (linia ciągła)

Wielomianowe krzywe parametryczne definiują punkty na krzywej 3D za pomocą trzech wielomianów z parametrem t, odpowiednio dla x, y i z. Współczynniki wielomianów są tak dobierane, żeby krzywa

przebiegała wzdłuż pożądanej ścieżki. Chociaż można wykorzystać różne stopnie wielomianów, przedstawiamy tylko najpopularniejszy przypadek: wielomiany trzeciego stopnia (występują w nich trzecie potęgi parametru). W odniesieniu do takich krzywych często będzie używane określenie *krzywa trzeciego stopnia.*

Parametryczne wielomianowe płaty powierzchni z dwiema zmiennymi określają współrzędne punktów na powierzchni krzywoliniowej za pomocą trzech wielomianów dwóch zmiennych, po jednym dla x, y i z. Brzegi płatów są parametrycznymi krzywymi wielomianowymi. W celu otrzymania aproksymacji powierzchni krzywoliniowej o zadanej dokładności jest potrzebna znacznie mniejsza liczba płatów powierzchni wielomianowych dwóch zmiennych niż płatów wielokątowych. Algorytmy potrzebne w przypadku korzystania z wielomianów z dwiema zmiennymi są jednak bardziej złożone niż w przypadku korzystania z wielokątów. Podobnie jak w przypadku krzywych można korzystać z wielomianów różnych stopni, niemniej tutaj ograniczymy się do omówienia najpopularniejszego przypadku wielomianów trzeciego stopnia w odniesieniu do obu parametrów. Powierzchnie takie są nazywane odpowiednio *powierzchniami bikubicznymi.*

Powierzchnie drugiego stopnia są to powierzchnie zdefiniowane bezpośrednio równaniami $f(x, y, z) = 0$, przy czym f jest wielomianem drugiego stopnia zmiennych x, y i z. Powierzchnie drugiego stopnia są wygodną reprezentacją kuli, elipsoidy i walca.

W rozdziale 10 poświęconym modelowaniu brył te reprezentacje są wykorzystane w systemach do reprezentowania nie tylko powierzchni, ale również ograniczonych (pełnych) objętości. Reprezentacje powierzchni opisane w tym rozdziale są używane czasami w różnych kombinacjach do ograniczania objętości 3D.

9.1. Siatki wielokątowe

Siatka wielokątowa jest zbiorem krawędzi, wierzchołków i wielokątów tak połączonych, że każda krawędź jest wspólna przynajmniej dla dwóch wielokątów. Krawędź łączy dwa wierzchołki, a wielokąt jest zamkniętą sekwencją krawędzi. Krawędź może być wspólna dla dwóch sąsiednich wielokątów, wierzchołek jest wspólny dla przynajmniej dwóch krawędzi i każda krawędź jest częścią jakiegoś wielokąta. Siatka wielokątowa może być reprezentowana na kilka sposobów, z których każdy ma swoje zalety i wady. Zadaniem programisty piszącego program użytkowy jest wybranie najlepszej reprezentacji. W jednym programie można wykorzystać kilka reprezentacji: jedną dla pamięci ze-

wnętrznej, inną dla potrzeb wewnętrznych i jeszcze inną, za pomocą której użytkownik interakcyjnie tworzy siatkę.

W celu porównania różnych reprezentacji można użyć dwóch podstawowych kryteriów: miejsce w pamięci i czas. Typowe operacje związane z siatką wielokątową umożliwiają znalezienie wszystkich krawędzi związanych z węzłem, wielokątów mających wspólną krawędź albo wierzchołek, wierzchołków połączonych krawędzią, krawędzi wielokąta, wyświetlenie siatki i zidentyfikowanie błędów reprezentacji (na przykład brakująca krawędź, wierzchołek albo wielokąt). Ogólnie, im bardziej bezpośrednio są reprezentowane relacje między wielokątami, wierzchołkami i krawędziami, tym operacje są szybsze i tym więcej miejsca w pamięci ta reprezentacja potrzebuje. Woo [WOO85] analizował złożoność czasową dziewięciu podstawowych operacji dostępu i dziewięć podstawowych operacji uaktualniania struktury danych siatki wielokątowej.

W punktach 9.1.1 i 9.1.2 są omawiane różne problemy związane z siatkami wielokątowymi: reprezentowanie siatek wielokątowych, zapewnienie poprawności danej reprezentacji, obliczenie współczynników płaszczyzny wielokąta.

9.1.1. Reprezentowanie siatek wielokątowych

W tym punkcie omawiamy trzy reprezentacje siatek wielokątowych: bezpośrednią, wskaźników na listę wierzchołków i wskaźników na listę krawędzi. W *reprezentacji bezpośredniej* każdy wielokąt jest opisany przez listę współrzędnych wierzchołków:

$$P = ((x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n))$$

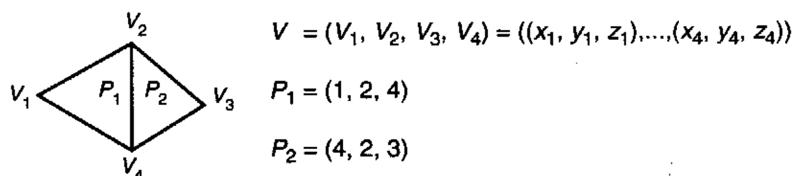
Wierzchołki są zapamiętane w kolejności, w jakiej napotkalibyśmy je posuwając się wokół wielokąta. Między kolejnymi wierzchołkami na liście i między pierwszym a ostatnim wierzchołkiem są krawędzie. Dla jednego wielokąta taka reprezentacja jest oszczędna, jeśli chodzi o zajętość pamięci; jednak dla siatki wielokątowej traci się dużo miejsca, ponieważ współrzędne wspólnych wierzchołków są powielane. Jeszcze większym problemem jest to, że nie ma bezpośredniej reprezentacji dla wspólnych krawędzi i wierzchołków. Na przykład, w celu interakcyjnego przesunięcia wierzchołka ze wszystkimi jego krawędziami, musimy znaleźć wszystkie wielokąty wspólne dla wierzchołka. Takie przeszukanie wymaga porównania trójek współrzędnych jednego wielokąta z trójkami współrzędnych innych wielokątów. Najefektywniejszą metodą wykonania tego byłoby posortowanie wszystkich N trójek współrzędnych, ale ten proces jest w najlepszym przypadku o złożoności

$N \log_2 N$, a nawet wtedy istnieje niebezpieczeństwo, że ten sam wierzchołek mógłby, ze względu na niedokładności obliczeń, mieć nieco inne wartości współrzędnych w każdym wielokącie i poprawnego dopasowania można by nigdy nie znaleźć.

Przy takiej reprezentacji wyświetlanie siatki albo jako wypełnionych wielokątów, albo jako obwodów wielokątów wymaga przekształcenia każdego wierzchołka i obcięcia każdej krawędzi każdego wielokąta. W czasie rysowania krawędzi każda wspólna krawędź jest rysowana dwukrotnie, co stwarza problemy dla ploterów piórowych, naświetlarek filmów i monitorów wektorowych, ze względu na ponowne kreślenie po już wykreślonym kształcie. Problem może również powstać na monitorach rastrowych wtedy, kiedy krawędzie są rysowane w przeciwnych kierunkach i mogą w efekcie zostać wyświetlone dodatkowe piksele.

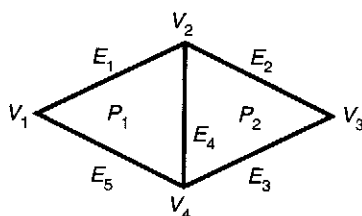
Wielokąty zdefiniowane za pomocą *wskaźników na listę wierzchołków* (jest to metoda wykorzystywana w SPHIGS) mają każdy wierzchołek siatki wielokątowej zapamiętany tylko raz na liście wierzchołków $V = ((x_1, y_1, z_1), \ldots, (x_n, y_n, z_n))$. Wielokąt jest zdefiniowany jako lista wskaźników na listę wierzchołków. Wielokąt zawierający wierzchołki 3, 5, 7 i 10 na liście wierzchołków byłby więc reprezentowany jako $P = (3, 5, 7, 10)$.

Przykład takiej reprezentacji pokazano na rys. 9.4. Ma ona kilka zalet w porównaniu z bezpośrednią reprezentacją wielokątową. Ponieważ każdy wierzchołek jest zapamiętany tylko raz, oszczędza się dużą ilość miejsca. Ponadto łatwo można zmienić współrzędne wierzchołków. Wciąż jest jednak trudno znaleźć wielokąty o wspólnej krawędzi i krawędzie wspólne dla wielokątów są wciąż rysowane dwukrotnie wówczas, gdy są wyświetlane wszystkie wielokąty. Możemy te dwa problemy wyeliminować, jeżeli krawędzie będą reprezentowane bezpośrednio, tak jak w następnej metodzie.



Rys. 9.4. Siatki wielokątowe zdefiniowane za pomocą wskaźników na listę wierzchołków

Jeżeli wielokąty definiujemy przez *wskaźniki na listę krawędzi*, to znowu mamy listę wierzchołków V, ale wielokąt jest reprezentowany jako lista wskaźników z tym, że nie na listę wierzchołków, a na listę krawędzi, w której każda krawędź występuje tylko raz. Z kolei każda



$$V = (V_1, V_2, V_3, V_4) = ((x_1, y_1, z_1), \ldots, (x_4, y_4, z_4))$$
$$E_1 = (V_1, V_2, P_1, \lambda)$$
$$E_2 = (V_2, V_3, P_2, \lambda)$$
$$E_3 = (V_3, V_4, P_2, \lambda)$$
$$E_4 = (V_4, V_2, P_1, P_2)$$
$$E_5 = (V_4, V_1, P_1, \lambda)$$
$$P_1 = (E_1, E_4, E_5)$$
$$P_2 = (E_2, E_3, E_4)$$

Rys. 9.5. Siatka wielokątowa zdefiniowana za pomocą listy krawędzi dla każdego wielokąta (λ reprezentuje zero)

krawędź z listy krawędzi wskazuje dwa wierzchołki z listy wierzchołków definiujące krawędź, a także jeden albo dwa wielokąty, do których należy krawędź. Dlatego opisujemy wielokąt jako $P = (E_1, \ldots, E_n)$ i krawędź jako $E = (V_1, V_2, P_1, P_2)$. Gdy krawędź należy tylko do jednego wielokąta, wówczas P_1 albo P_2 jest zerem. Na rysunku 9.5 pokazano przykład takiej reprezentacji.

Kontury wielokątów pokazujemy wyświetlając wszystkie krawędzie, a nie wszystkie wielokąty; w ten sposób omija się nadmiarowe obcinanie, przekształcenia i konwersje. Również łatwo wyświetla się wypełnione wielokąty. Czasami, na przykład przy opisie struktury 3D przypominającej plaster miodu, niektóre krawędzie są wspólne dla trzech wielokątów. W takich przypadkach opis krawędzi może być tak rozszerzony, żeby zawierał dowolną liczbę wielokątów: $E = (V_1, V_2, P_1, P_2, \ldots, P_n)$.

W żadnej z trzech reprezentacji (to znaczy bezpośrednich wielokątów, wskaźników na listę wierzchołków, wskaźników na listę krawędzi) nie jest łatwo określić, które krawędzie łączą się z wierzchołkiem: trzeba sprawdzić wszystkie krawędzie. Oczywiście można dodać bezpośrednią informację, która umożliwi określanie takich zależności. Na przykład reprezentacja używana przez Baumgarta [BAUM75] rozszerza opis krawędzi tak, żeby zawierał wskaźniki na krawędzie następne w opisie każdej ze ścian. Opis wierzchołka zawiera wskaźnik na (dowolną) krawędź zawierającą ten wierzchołek.

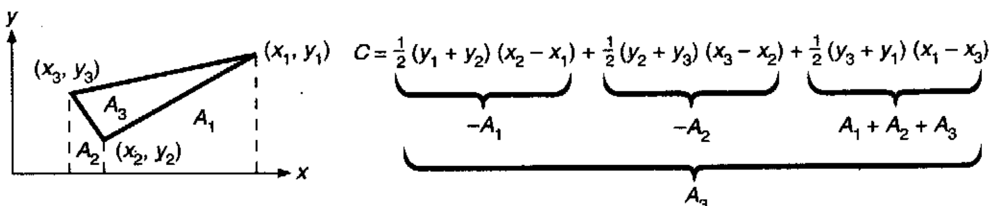
9.1.2. Równania płaszczyzny

Gdy pracujemy z wielokątami albo siatkami wielokątowymi, wówczas często potrzebna jest nam znajomość równań płaszczyzny, na której leży wielokąt. Oczywiście w niektórych przypadkach równanie wynika pośrednio z interakcyjnej metody konstrukcji używanej do określania

wielokąta. Jeżeli równanie nie jest znane, to możemy wykorzystać współrzędne trzech wierzchołków do znalezienia płaszczyzny. Przypomnijmy równanie płaszczyzny

$$Ax + By + Cz + D = 0 \tag{9.1}$$

Współczynniki A, B i C definiują normalną do płaszczyzny $[A\ B\ C]$. Dla danych trzech punktów P_1, P_2, P_3 na płaszczyźnie możemy policzyć normalną do płaszczyzny jako iloczyn wektorowy $P_1P_2 \times P_1P_3$ (albo $P_2P_3 \times P_2P_1$ itd.). Jeżeli ten iloczyn jest równy zeru, to te trzy punkty są współliniowe i nie określają płaszczyzny. Zamiast tego można użyć innych wierzchołków. Dla niezerowego iloczynu wektorowego możemy znaleźć D podstawiając normalną do $[A\ B\ C]$ i dowolny z trzech punktów do równania (9.1).



Rys. 9.6. Obliczanie pola C trójkąta za pomocą równania (9.2)

Jeżeli mamy więcej niż trzy wierzchołki, to mogą one nie leżeć na jednej płaszczyźnie albo ze względów numerycznych, albo ze względu na metodę, jaką były generowane wielokąty. Wtedy od metody iloczynu wektorowego lepsza jest inna metoda znajdowania współczynników A, B, C płaszczyzny, która przechodzi blisko wszystkich wierzchołków. Można wykazać, że A, B i C są proporcjonalne do obszarów ze znakiem rzutów wielokąta odpowiednio na płaszczyzny (y, z), (z, x) i (x, y). Na przykład, jeżeli wielokąt jest równoległy do płaszczyzny (x, y), to $A = B = 0$; tak jak należałoby się spodziewać: rzuty wielokąta na płaszczyzny (x, z) i (z, x) mają zerową powierzchnię. Ta metoda jest lepsza niż metoda iloczynu wektorowego, ponieważ pola rzutów są funkcją współrzędnych wszystkich wierzchołków i wobec tego nie są czułe na wybór kilku wierzchołków, które mogłyby być współliniowe. Na przykład, pole (i stąd współczynnik) C wielokąta zrzutowanego na płaszczyznę (x, y) na rys. 9.6 jest polem trapezoidu A_3 minus pola A_1 i A_2. Ogólnie

$$C = \frac{1}{2} \sum_{i=1}^{n} (y_i + y_{i \oplus 1})(x_{i \oplus 1} - x_i) \tag{9.2}$$

przy czym operator $\oplus$ jest zwykłym dodawaniem z wyjątkiem przypadku, gdy $n \oplus 1 = 1$. Pola dla A i B są dane za pomocą podobnej formuły, z wyjątkiem tego, że przy B pojawia się znak minus (por. przykład 9.1).

Równanie (9.2) zawiera sumę pól wszystkich trapezoidów utworzonych przez kolejne krawędzie wielokątów. Jeżeli $x_{i \oplus 1} < x_i$, to pole wnosi ujemny wkład do sumy. Znak sumy jest również użyteczny: jeżeli wierzchołki są ponumerowane w kierunku zgodnym z ruchem wskazówek zegara (przy rzutowaniu na płaszczyznę), to znak jest dodatni; w przeciwnym przypadku jest ujemny.

Gdy raz określimy równanie płaszczyzny korzystając ze wszystkich wierzchołków, wówczas możemy oszacować, jak bardzo wielokąt nie jest płaski, obliczając odległość płaszczyzny od każdego wierzchołka. Odległość d dla wierzchołka w (x, y, z) wynosi

$$d = \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}} \tag{9.3}$$

Ta odległość jest albo dodatnia, albo ujemna, zależnie od tego, po której stronie płaszczyzny jest umieszczony punkt. Jeżeli wierzchołek jest na płaszczyźnie, to $d = 0$. Oczywiście, w celu określenia jedynie po której stronie płaszczyzny jest punkt, istotny jest tylko znak d i dzielenie przez pierwiastek kwadratowy nie jest konieczne.

Równanie płaszczyzny nie jest unikatowe; każdy stały niezerowy współczynnik k zmienia równanie, ale nie płaszczyznę. Często jest wygodnie pamiętać współczynniki ze znormalizowaną normalną; możemy zrobić to wybierając

$$k = \frac{1}{\sqrt{A^2 + B^2 + C^2}} \tag{9.4}$$

jest to odwrotność długości normalnej. Teraz odległość można łatwiej policzyć za pomocą równania (9.3), ponieważ mianownik jest równy 1.

Przykład 9.1 **Problem**: Napisz funkcję, która oblicza współczynniki równania płaszczyzny, jeżeli danych jest n wierzchołków wielokąta, który jest w przybliżeniu płaski. Załóż, że wierzchołki wielokąta są ponumerowane w kierunku przeciwnym względem ruchu wskazówek zegara, jeśli patrzymy w kierunku płaszczyzny od dodatniej strony płaszczyzny. Wierzchołki i liczba wierzchołków są argumentami przekazywanymi do funkcji.

Odpowiedź: Korzystając z równania (9.2) i podobnych równań określających A i B otrzymujemy następujący program:

Ta funkcja oblicza współczynniki równania płaszczyzny

```
FindPlaneCoefficients(float x[], float y[], float z[], int num_verts,
                      float *a, float *b, float *c, float *d)
{
   float A, B, C, D;
   int   i, j;

   A = B = C = 0.0;
   for (i = 0; i < num_verts; i++) {
      j = (i + 1) % num_verts;
      A += (z[i] + z[j]) * (y[j] – y[i]);
      B += – (x[i] + x[j]) * (z[j] – z[i]);
      C += (y[i] + y[j]) * (x[j] – x[i]);
   }
   A /= 2.0; B /= 2.0; C /= 2.0;
   D = –(A * x[0] + B * y[0] + C * z[0]);

   *a = A;
   *b = B;
   *c = C;
   *d = D;
}
```

9.2. Parametryczne krzywe trzeciego stopnia

Łamane i wielokąty są aproksymacjami pierwszego stopnia kawałkami liniowymi, odpowiednio dla krzywych i powierzchni. Z wyjątkiem przypadku, gdy aproksymowane krzywe i powierzchnie są również kawałkami liniowe, w celu uzyskania odpowiedniej dokładności trzeba tworzyć i pamiętać duże ilości współrzędnych punktów końcowych. Interakcyjne manipulowanie danymi w celu aproksymowania kształtu jest męczące, ponieważ trzeba precyzyjnie umieścić wiele punktów.

W tym punkcie przedstawiono bardziej zwartą i łatwiejszą do manipulowania reprezentację krzywych kawałkami gładkich; w punkcie 9.3 wywód matematyczny jest uogólniony na powierzchnie. Ogólne podejście polega na tym, żeby używać funkcji wyższego stopnia niż funkcje liniowe. Funkcje nadal tylko aproksymują pożądane kształty, ale potrzebują mniej pamięci i są łatwiejsze przy pracy interakcyjnej niż funkcje liniowe.

Aproksymacje wyższego stopnia mogą bazować na jednej z trzech metod. Po pierwsze, możemy wyrazić y i z jako *bezpośrednie* funkcje x, tak że $y = f(x)$ i $z = g(x)$. Z takim podejściem są związane pewne trudności: 1) nie można uzyskać wielu wartości y dla jednej wartości x

i krzywe takie jak okręgi i elipsy muszą być reprezentowane przez kilka segmentów; 2) taka definicja nie jest inwariantna ze względu na obroty (opisanie obróconej wersji krzywej wymaga dużego nakładu pracy i ogólnie może wymagać dzielenia segmentu krzywej na wiele innych); 3) opisanie krzywych za pomocą pionowych stycznych jest trudne, ponieważ trudno jest reprezentować nachylenie nieskończenie duże.

Po drugie, możemy zdecydować się na modelowanie krzywych jako rozwiązań równań *uwikłanych* o postaci $f(x, y, z) = 0$; ta metoda jest najeżona niebezpieczeństwami. Dane równanie może mieć więcej rozwiązań, niż chcemy. Na przykład przy modelowaniu okręgu moglibyśmy z powodzeniem użyć równania $x^2 + y^2 = 1$. Ale jak modelować połowę okręgu? Musimy dodać takie ograniczenia jak $x \geqslant 0$, które nie mogą być zawarte w równaniu uwikłanym. Ponadto jeżeli dwa segmenty krzywej określonej w sposób uwikłany są połączone razem, to może być trudno określić, czy kierunki ich stycznych zgadzają się w punkcie połączenia. Ciągłość stycznej jest krytyczna w wielu zastosowaniach.

Te dwa matematyczne sposoby umożliwiają szybkie określenie, czy punkt leży na krzywej, albo po której stronie krzywej, tak jak to było robione w rozdz. 3. Normalne do krzywej są również łatwe do obliczenia. Postać uwikłaną omówimy dokładniej w p. 9.4.

Parametryczna reprezentacja krzywych $x = x(t)$, $y = y(t)$, $z = z(t)$ omija problemy związane z reprezentacją funkcyjną albo uwikłaną i oferuje wiele innych zalet, które zostaną przedstawione w dalszej części rozdziału. Krzywe parametryczne zastępują korzystanie z nachyleń geometrycznych (które mogą być nieskończenie duże) parametrycznymi wektorami stycznymi (które, jak zobaczymy, nigdy nie są nieskończone). Tutaj krzywa jest aproksymowana krzywą kawałkami wielomianową zamiast krzywą kawałkami liniową używaną w p. 9.1. Każdy segment Q całej krzywej jest określony trzema funkcjami x, y i z, które są wielomianami trzeciego stopnia parametru t.

Wielomiany trzeciego stopnia są używane najczęściej, ponieważ wielomiany niższego stopnia są zbyt mało elastyczne, jeśli chodzi o sterowanie kształtem krzywej, a wielomiany wyższego stopnia wprowadzają niepożądane oscylacje, a ponadto wymagają większej liczby obliczeń. Żadna reprezentacja niższego stopnia nie zapewnia tego, żeby segment krzywej interpolował (przechodził przez) dwa określone punkty końcowe z określonymi pochodnymi na każdym końcu. Dla danego wielomianu trzeciego stopnia cztery znane wielkości są wykorzystywane do znalezienia jego czterech nie znanych współczynników. Tymi czterema znanymi wielkościami mogą być dwa punkty końcowe i pochodne w punktach końcowych. Dla odcinka pochodne na każdym końcu są określone przez sam odcinek i nie mogą być kontrolowane niezależnie.

W przypadku wielomianów drugiego stopnia – i stąd trzech współczynników – można określić dwa punkty końcowe i jeden dodatkowy warunek, np. nachylenie albo dodatkowy punkt.

Krzywe parametryczne trzeciego stopnia są również krzywymi najniższego stopnia, które nie leżą w jednej płaszczyźnie w 3D. Możemy się o tym przekonać, jeżeli zauważymy, że trzy współczynniki wielomianu drugiego stopnia mogą być w pełni określone przez trzy punkty i że trzy punkty definiują płaszczyznę, w której leży wielomian.

Krzywe wyższego stopnia wymagają większej liczby warunków do określenia współczynników i mogą oscylować w sposób trudny do kontrolowania. Mimo tych trudności krzywe wyższych stopni są używane w zastosowaniach – np. projektowanie samochodów czy samolotów – w których trzeba kontrolować pochodne wyższego rzędu po to, żeby uzyskiwać powierzchnie aerodynamiczne. W matematyce wyprowadzane krzywe parametryczne i powierzchnie są często określane dla dowolnego stopnia n. W tym rozdziale przyjmujemy $n = 3$.

9.2.1. Podstawowe charakterystyki

Wielomiany trzeciego stopnia, które określają segment krzywej $Q(t) = [x(t)\ y(t)\ z(t)]^T$ mają postać:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$z(t) = a_z t^3 + a_z t^2 + c_z t + d_z, \quad 0 \leqslant t \leqslant 1 \tag{9.5}$$

Aby operować na skończonych segmentach krzywej, ograniczamy, bez straty ogólności, parametr t do przedziału [0, 1].

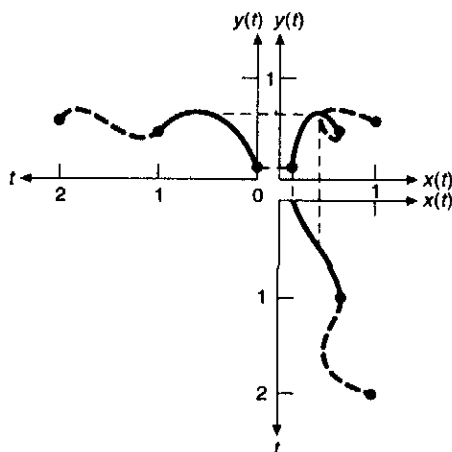
Dla $T = [t^3\ t^2\ t\ 1]^T$ i macierzy współczynników trzech wielomianów o postaci

$$C = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix} \tag{9.6}$$

możemy przepisać równanie (9.5) jako

$$Q(t) = [x(t)\ y(t)\ z(t)]^T = C \cdot T \tag{9.7}$$

Jest to zwarty sposób zapisu równania (9.5).



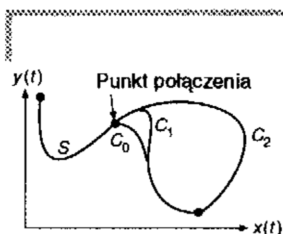
Rys. 9.7. Dwa połączone segmenty krzywej 2D i definiujące ją wielomiany. Linie przerywane między wykresem (x,y) i wykresami $x(t)$ i $y(t)$ pokazują odpowiedniość między punktami na krzywej (x,y) i definiującymi ją wielomianami trzeciego stopnia. Wykresy $x(t)$ i $y(t)$ dla drugiego segmentu zostały przesunięte tak, żeby zaczynały się dla $t = 1$, a nie dla $t = 0$ po to, żeby pokazać ciągłość krzywych w punkcie połączenia

Na rysunku 9.7 pokazano dwa połączone segmenty krzywej parametrycznej trzeciego stopnia i ich wielomiany; rysunek ilustruje również zdolność opisu parametrycznego łatwego reprezentowania wielokrotnych wartości y dla jednej wartości x za pomocą wielomianów, które same są jednowartościowe. (Ten rysunek, podobnie jak i inne w tym rozdziale, pokazuje krzywą 2D reprezentowaną przez $[x(t)\ y(t)]^T$.)

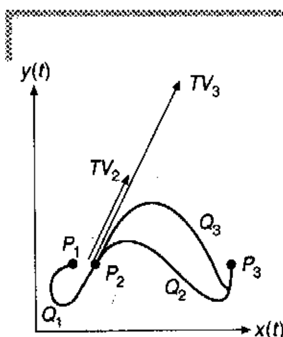
Ciągłość między segmentami krzywej. Pochodna $Q(t)$ jest parametrycznym wektorem stycznym krzywej. Stosując tę definicję do równania (9.7) mamy

$$\frac{d}{dt}Q(t) = Q'(t) = \left[\frac{d}{dt}x(t) \quad \frac{d}{dt}y(t) \quad \frac{d}{dt}z(t)\right]^T = \frac{d}{dt}C \cdot T =$$
$$= C \cdot [3t^2\ 2t\ 1\ 0]^T =$$
$$= [3a_x t^2 + 2b_x t + c_x \quad 3a_y t^2 + 2b_y t + c_y \quad 3a_z t^2 + 2b_z t + c_z]^T \tag{9.8}$$

Jeżeli dwa segmenty krzywej łączą się ze sobą, to krzywa ma ciągłość geometryczną G^0. Jeżeli kierunki (ale niekoniecznie długości) dwóch wektorów stycznych segmentów są równe w punkcie połączenia, to krzywa ma ciągłość geometryczną G^1. Ciągłość G^1 między segmentami krzywej jest często wymagana w projektowaniu wspomaganym komputerem. Ciągłość G^1 oznacza, że geometryczne nachylenia seg-



Rys. 9.8. Segment S krzywej połączony z segmentami C_0, C_1 i C_2 o stopniach ciągłości parametrycznej odpowiednio 0, 1 i 2. Wzrokowe rozróżnienie między C_1 a C_2 jest trudne w punkcie połączenia, oczywiste zaś w pewnej odległości od połączenia



Rys. 9.9. Segmenty Q_1, Q_2 i Q_3 łączą się w punkcie P_2. Segmenty Q_1 i Q_2 mają równe wektory styczne i dlatego oba mają w punkcie P_2 zarówno ciągłość G^1, jak i C^1. Wektory styczne Q_1 i Q_3 mają ten sam kierunek, natomiast Q_3 ma dwa razy większą amplitudę i wobec tego w punkcie P_2 jest tylko ciągłość G^1. Dłuższy wektor styczny Q_3 oznacza, że krzywa jest bardziej przeciągnięta w kierunku wektora stycznego przed zagięciem w kierunku P_3. Wektor TV_2 jest wektorem stycznym dla Q_2, a TV_3 dla Q_3

mentów są równe w punkcie połączenia. Na to, żeby dwa wektory styczne TV_1 i TV_2 miały ten sam kierunek, trzeba, żeby jeden był skalarną wielokrotnością drugiego: $TV_1 = k \cdot TV_2$ przy $k > 0$ [BARS88].

Jeżeli wektory styczne dwóch segmentów krzywej trzeciego stopnia są równe (to znaczy, ich kierunki i długości są równe) w punkcie połączenia, to krzywa ma ciągłość pierwszego stopnia ze względu na parametr t albo *ciągłość parametryczną* i mówi się, że ma ciągłość C^1. Jeżeli kierunki i długości $\frac{d^n}{dt^n}[Q(t)]$ do n-tej pochodnej są równe w punkcie wspólnym, mówi się, że krzywa ma *ciągłość* C^n. Na rysunku 9.8 pokazano krzywe z trzema różnymi stopniami ciągłości. Zauważmy, że segment krzywej parametrycznej jest sam ciągły wszędzie; interesuje nas tu ciągłość w punktach połączenia.

Wektor styczny $Q'(t)$ reprezentuje *prędkość* punktu na krzywej względem parametru t. Podobnie druga pochodna $Q(t)$ jest *przyspieszeniem*. Załóżmy, że kamera porusza się wzdłuż krzywej parametrycznej trzeciego stopnia ze stałym kwantem czasu i że rejestrujemy obraz po każdym kroku; wektor styczny daje prędkość kamery wzdłuż krzywej. Na to, żeby uniknąć gwałtownych ruchów w otrzymanej animacji, prędkość kamery i przyspieszenie w punktach połączenia powinny być ciągłe. Skutkiem tej ciągłości przyspieszenia w punkcie połączenia jest to, że na rys. 9.8 krzywa C_2 biegnie dalej w prawo niż krzywa C_1, przed dojściem do punktu końcowego.

Ogólnie ciągłość parametryczna C^1 implikuje ciągłość geometryczną G^1, ale odwrotne stwierdzenie nie zawsze jest prawdziwe. To znaczy, ogólnie ciągłość G^1 jest mniej restrykcyjna niż ciągłość C^1, a więc krzywa może mieć nieciągłość G^1, ale niekoniecznie musi mieć ciągłość C^1. Niemniej punkty połączenia z ciągłością G^1 wyglądają tak samo gładko jak te z ciągłością C^1, co widać na rys. 9.9.

Wykres krzywej parametrycznej jest istotnie różny od wykresu zwykłej funkcji, na którym zmienna niezależna jest rysowana na osi x, a zmienna zależna na osi y. Na wykresach krzywych parametrycznych zmienna niezależna w ogóle nie jest rysowana. Dlatego patrząc tylko na krzywą nie możemy określić wektora stycznego do krzywej. Można określić kierunek wektora stycznego, ale nie jego wielkość. Możemy się o tym przekonać rozumując w następujący sposób: jeżeli $\gamma(t)$, $0 \leqslant t \leqslant 1$, jest krzywą parametryczną, to jej wektorem stycznym w chwili 0 jest $\gamma'(0)$. Jeżeli przyjmiemy $\eta(t) = \gamma(2t)$, $0 \leqslant t \leqslant 1/2$, to wykresy parametryczne γ i η są identyczne. A z drugiej strony $\eta'(0) = 2\gamma'(0)$. Dlatego dwie krzywe, które mają identyczne wykresy, mogą mieć różne wektory styczne. Ten fakt jest podstawą definicji *ciągłości geometrycznej*: na to, żeby dwie krzywe łączyły się gładko, trzeba jedynie, żeby kierunki ich wektorów stycznych zgadzały się; nie wymagamy, żeby się zgadzały ich wielkości.

Związek z ograniczeniami. Segment krzywej $Q(t)$ jest określony przez ograniczenia: punkty końcowe, wektory styczne i ciągłość między segmentami krzywej. Każdy wielomian trzeciego stopnia z równania (9.5) ma cztery współczynniki, a więc są potrzebne cztery ograniczenia; umożliwia to ułożenie czterech równań z czterema niewiadomymi i następnie rozwiązanie tych równań. Trzy główne typy krzywych omawiane w tym punkcie to *krzywe Hermite'a*, określone przez dwa punkty końcowe i dwa wektory styczne w tych punktach końcowych; *krzywe Béziera*, określone przez dwa punkty końcowe i dwa inne punkty, które mają wpływ na wektory styczne w punktach końcowych; różnego rodzaju *krzywe sklejane*, które są określone przez cztery punkty kontrolne. Krzywe sklejane mają ciągłość C^1 i C^2 w punktach połączenia i przechodzą blisko swoich punktów sterujących, ale w ogólnym przypadku nie interpolują punktów. Przykładami krzywych sklejanych są jednorodne krzywe B-sklejane i niejednorodne krzywe B-sklejane.

W celu pokazania, jak współczynniki równania (9.5) mogą zależeć od czterech ograniczeń, przypomnijmy, że krzywa parametryczna trzeciego stopnia jest określona przez $Q(t) = C \cdot T$. Przepiszmy macierz współczynników jako $C = G \cdot M$, przy czym M jest *macierzą bazową* 4×4, a G jest czteroelementową macierzą ograniczeń geometrycznych, określaną jako *macierz geometrii*. Ograniczenia geometryczne to po prostu warunki takie jak punkty końcowe albo wektory styczne, które określają krzywą. G_x będzie oznaczało wektor wierszowy elementów x macierzy geometrii. G_y i G_z mają podobne definicje. G albo M, albo zarówno G, jak i M są różne dla różnych typów krzywych.

Elementy G i M są stałe, a więc iloczyn $G \cdot M \cdot T$ to po prostu trzy wielomiany trzeciego stopnia zmiennej t. Rozwijając iloczyn $Q(t) = G \cdot M \cdot T$ otrzymujemy

$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = [G_1 \; G_2 \; G_3 \; G_4] \begin{bmatrix} m_{11} & m_{21} & m_{31} & m_{41} \\ m_{12} & m_{22} & m_{32} & m_{42} \\ m_{13} & m_{23} & m_{33} & m_{43} \\ m_{14} & m_{24} & m_{34} & m_{44} \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \tag{9.9}$$

Równanie to możemy czytać w następujacy sposób: punkt $Q(t)$ jest sumą ważoną kolumn macierzy geometrii G, z których każda reprezentuje punkt albo wektor w przestrzeni 3D.

Wymnażając tylko $x(t) = G_x \cdot M \cdot T$ otrzymujemy

$$\begin{aligned} x(t) = {} & (t^3 m_{11} + t^2 m_{21} + t m_{31} + m_{41}) g_{1x} + \\ & + (t^3 m_{12} + t^2 m_{22} + t m_{32} + m_{42}) g_{2x} + \\ & + (t^3 m_{13} + t^2 m_{23} + t m_{33} + m_{43}) g_{3x} + \\ & + (t^3 m_{14} + t^2 m_{24} + t m_{34} + m_{44}) g_{4x} \end{aligned} \tag{9.10}$$

Równanie (9.10) podkreśla, że krzywa jest sumą ważoną elementów macierzy geometrii. Wagi są wielomianami trzeciego stopnia od t i są nazywane *funkcjami bazowymi*. Funkcje bazowe są określone jako $B = M \cdot T$. Zauważmy podobieństwo do aproksymacji kawałkami liniowej, dla której są potrzebne tylko dwa ograniczenia geometryczne (punkty końcowe odcinka), i każdy segment krzywej jest odcinkiem określonym przez punkty końcowe G_1 i G_2:

$$\begin{aligned} x(t) &= g_{1x}(1-t) + g_{2x}(t) \\ y(t) &= g_{1y}(1-t) + g_{2y}(t) \\ z(t) &= g_{1z}(1-t) + g_{2z}(t) \end{aligned} \tag{9.11}$$

Krzywe parametryczne trzeciego stopnia są po prostu uogólnieniem aproksymacji odcinkami. Krzywa trzeciego stopnia $Q(t)$ jest kombinacją czterech kolumn macierzy geometrii, podobnie jak segment typu odcinek jest kombinacją dwóch wektorów kolumnowych.

W celu pokazania, jak obliczać macierz bazową M, zajmiemy się teraz specyficznymi postaciami krzywych parametrycznych trzeciego stopnia.

9.2.2. Krzywe Hermite'a

Postać Hermite'a (od nazwiska matematyka) segmentu krzywej wielomianowej trzeciego stopnia jest określona przez ograniczenia dotyczące punktów końcowych P_1 i P_4 i wektorów stycznych w punktach końcowych R_1 i R_4. (Są używane indeksy 1 i 4, a nie 1 i 2 ze względu na zgodność z późniejszym punktem, gdzie do określania krzywej będą używane punkty pośrednie P_2 i P_3, zamiast wektorów stycznych.)

W celu znalezienia *macierzy bazowej Hermite'a* M_H, która wiąże *wektor geometrii Hermite'a* G_H ze współczynnikami wielomianu, piszemy cztery równania, po jednym dla każdego ograniczenia, z czterema nie znanymi współczynnikami wielomianu i następnie rozwiązujemy te równania.

Określając G_{H_x}, x-ową składową macierzy geometrii Hermite'a, jako

$$G_{H_x} = [P_{1_x}\ P_{4_x}\ R_{1_x}\ R_{4_x}] \tag{9.12}$$

i przepisując $x(t)$ z równań (9.5) i (9.9) jako

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x = C_x \cdot T = \\ &= G_{H_x} \cdot M_H \cdot T = G_{H_x} \cdot M_H [t^3\ t^2\ t\ 1] \end{aligned} \tag{9.13}$$

ograniczenia na $x(0)$ i $x(1)$ znajduje się w wyniku bezpośredniego podstawienia do równania (9.13) jako:

$$x(0) = P_{1_x} = G_{H_x} \cdot M_H [0\ 0\ 0\ 1]^T \tag{9.14}$$

$$x(1) = P_{4_x} = G_{H_x} \cdot M_H [1\ 1\ 1\ 1]^T \tag{9.15}$$

Podobnie jak w ogólnym przypadku różniczkowaliśmy równanie (9.7) w celu znalezienia równania (9.8), różniczkujemy teraz równanie (9.13) i otrzymujemy $x'(t) = G_{H_x} \cdot M_H [3t^2\ 2t\ 1\ 0]^T$. A więc równania dla ograniczeń typu wektory styczne mogą być napisane jako:

$$x'(0) = R_{1_x} = G_{H_x} \cdot M_H [0\ 0\ 1\ 0]^T \tag{9.16}$$

$$x'(1) = R_{4_x} = G_{H_x} \cdot M_H [3\ 2\ 1\ 0]^T \tag{9.17}$$

Cztery ograniczenia z równań (9.14)-(9.17) mogą być przepisane w postaci macierzowej

$$[P_{1_x}\ P_{4_x}\ R_{1_x}\ R_{4_x}] = G_{H_x} = G_{H_x} \cdot M_H \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{9.18}$$

Na to, żeby to równanie (i odpowiednie wyrażenia określające y i z) było spełnione, M_H musi być odwrotnością macierzy 4×4 w równaniu (9.18)

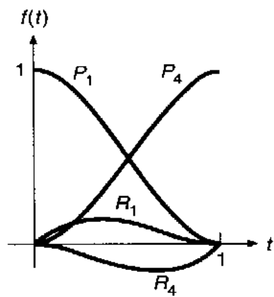
$$M_H = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \tag{9.19}$$

M_H może być teraz użyta w równaniu $x(t) = G_{H_x} \cdot M_H \cdot T$ w celu znalezienia $x(t)$ w zależności od wektora geometrii G_{H_x}. Podobnie $y(t) = G_{H_y} \cdot M_H \cdot T$ i $z(t) = G_{H_z} \cdot M_H \cdot T$ i możemy napisać

$$Q(t) = [x(t)\ y(t)\ z(t)]^T = G_H \cdot M_H \cdot T \tag{9.20}$$

przy czym G_H jest macierzą

$$[P_1\ P_4\ R_1\ R_4]$$

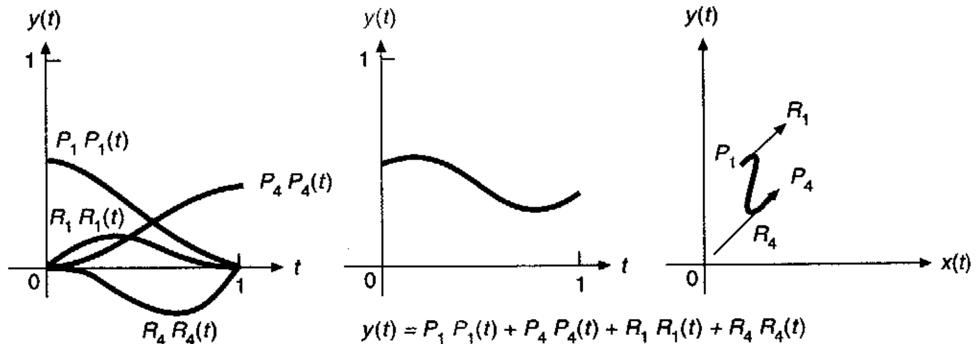


Rys. 9.10. Funkcje bazowe Hermite'a oznaczone tak jak elementy wektora geometrii, dla których są wagami

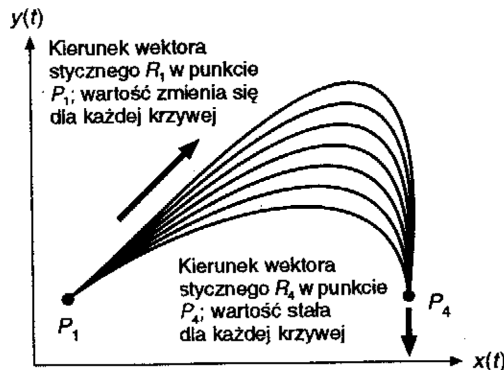
Rozszerzenie iloczynu $M_H \cdot T$ na $Q(t) = G_H \cdot M_H \cdot T$ daje *funkcje bazowe Hermite'a* B_H jako wielomiany wagowe dla każdego elementu macierzy geometrii

$$Q(t) = G_H \cdot M_H \cdot T = G_H \cdot B_H = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + \\ + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4 \tag{9.21}$$

Na rysunku 9.10 pokazano cztery funkcje bazowe. Zauważmy, że dla $t = 0$ tylko funkcja P_1 jest niezerowa: w punkcie $t = 0$ tylko P_1 wpływa na krzywą. Gdy tylko t stanie się większe od zera, wpływ zaczynają mieć R_1, P_4 i R_4. Na rysunku 9.11 pokazano cztery funkcje wagowe dla składowej y określonego wektora geometrii, ich sumę $y(t)$ i krzywą $Q(t)$.

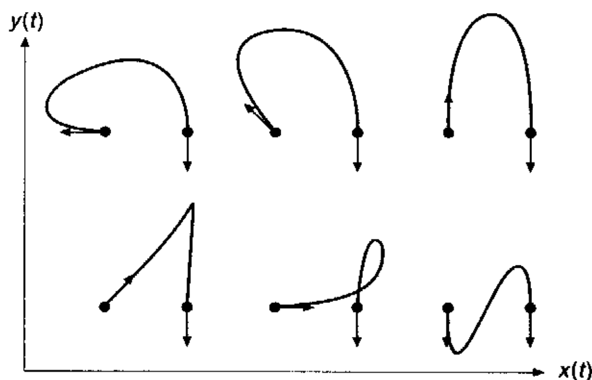


Rys. 9.11. Krzywa Hermite'a pokazująca cztery elementy wektora geometrii ważone przez funkcje bazowe (cztery krzywe z lewej strony), ich sumę $y(t)$ i samą krzywą 2D (z prawej strony). $x(t)$ jest zdefiniowane przez podobną sumę ważoną

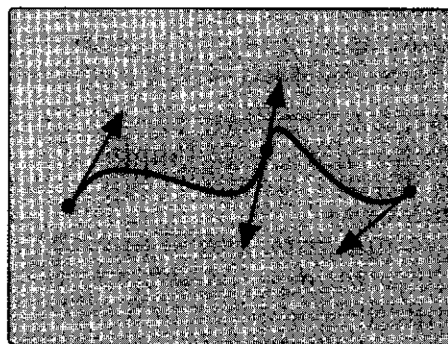


Rys. 9.12. Rodzina krzywych parametrycznych Hermite'a trzeciego stopnia. Dla każdej krzywej zmienia się tylko R_1, wektor styczny w P_1; jego wartość rośnie dla wyższych krzywych

Na rysunku 9.12 pokazano ciąg krzywych Hermite'a. Jedyną różnicą między nimi jest długość wektora stycznego R_1: kierunki wektorów stycznych są ustalone. Im dłuższe są wektory, tym mają większy wpływ na krzywą. Na rysunku 9.13 pokazano inny zestaw krzywych Hermite'a z wektorami stycznymi o stałej długości, ale o różnych kierunkach. W graficznym systemie interakcyjnym punkty końcowe i wektory styczne krzywej są zmieniane przez użytkownika interakcyjnie, co umożliwia kształtowanie krzywej. Przykładowy sposób implementacji takiej interakcji pokazano na rys. 9.14.



Rys. 9.13. Rodzina krzywych parametrycznych trzeciego stopnia Hermite'a. Zmienia się tylko kierunek wektora stycznego w lewym punkcie początkowym; wszystkie wektory styczne mają tę samą wartość. Mniejsza wartość wyeliminowałaby pętlę w jednej krzywej



Rys. 9.14. Wyświetlono dwa segmenty krzywej Hermite'a trzeciego stopnia z elementami sterującymi w celu ułatwienia interakcyjnej pracy. Użytkownik może zmieniać punkty końcowe przeciągając je oraz może zmieniać wektory styczne przeciągając strzałki. Wektory styczne w punkcie połączenia muszą być współliniowe (w celu zapewnienia ciągłości C^1): użytkownik zazwyczaj ma możność wydania polecenia wymuszającego ciągłość C^0, C^1, G^1 albo brak ciągłości. Wektory styczne na końcu $t = 1$ każdej krzywej są rysowane w odwrotnym kierunku od używanego w matematycznym sformułowaniu krzywej Hermite'a – chodzi o przejrzystość i wygodę w interakcji

Rysowanie krzywych parametrycznych. Krzywe Hermite'a i inne podobne krzywe parametryczne trzeciego stopnia są łatwe do wyświetlania. Obliczamy równania (9.5) dla n kolejnych wartości t z krokiem δ. Odpowiedni kod jest podany w postaci programu 9.1. Obliczenia w {...} w pętli **for** wymagają 11 mnożeń i 10 dodawań dla punktu 3D. Reguła Hornera faktoryzacji wielomianów

$$f(t) = at^3 + bt^2 + ct + d = ((at + b)t + c)t + d \tag{9.22}$$

zmniejszyłaby nakład obliczeń do 9 mnożeń i 10 dodawań dla punktu 3D. W bardziej efektywnym sposobie wyświetlania korzysta się z metody różnic omówionej w pracy [FOLE90].

Program 9.1
Program wyświetlania krzywej parametrycznej trzeciego stopnia

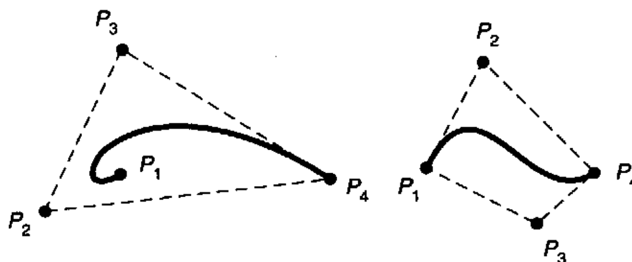
```
typedef float CoefficientArray[4];
void DrawCurve(CoefficientArray *cx, CoefficientArray *cy,
               CoefficientArray *cz, int n)
  /* cx, cy i cz są współczynnikami dla x(t), y(t) i z(t) */
  /* na przykład C_x = G_x · M itd. */
  /* n – liczba kroków */
{
  float x, y, z, delta, t, t2, t3;
  int   i;

  MoveAbs3( cx[3], cy[3], cz[3] );
  delta = 1.0 / n;
  for (i = 1; i <= n; i++) {
    t = i * delta;
    t2 = t * t;
    t3 = t2 * t;
    x = cx[0] * t3 + cx[1] * t2 + cx[2] * t + cx[3];
    y = cy[0] * t3 + cy[1] * t2 + cy[2] * t + cy[3];
    z = cz[0] * t3 + cz[1] * t2 + cz[2] * t + cz[3];
    DrawAbs3( x, y, z );
  }
}
```

Ponieważ krzywe trzeciego stopnia są kombinacjami liniowymi (sumami ważonymi) czterech elementów wektora geometrii, tak jak w równaniu (9.10), możemy przekształcać krzywe na zasadzie przekształcenia wektora geometrii i wykorzystania go potem do generowania przekształconej krzywej, co jest równoważne stwierdzeniu, że krzywe są niezmiennicze względem obrotu, przesunięcia i skalowania. Taka strategia jest bardziej efektywna niż generowanie krzywej jako ciągu krótkich

odcinków i następnie przekształcanie każdego odcinka indywidualnie. Krzywe nie są niezmiennicze względem rzutu perspektywicznego, co będzie omawiane w p. 9.2.6.

9.2.3. Krzywe Béziera

Postać Béziera [BEZI70; BEZI74] segmentu krzywej wielomianowej trzeciego stopnia zawdzięcza swą nazwę Pierre'owi Bézierowi, który opracował ją z myślą o wykorzystaniu przy projektowaniu samochodów w firmie Renault. W przypadku tych krzywych wektory styczne w punktach końcowych są określone bezpośrednio przez dwa punkty



Rys. 9.15. Dwie krzywe Béziera i ich punkty kontrolne. Zauważmy, że wypukły wielokąt utworzony przez punkty kontrolne, pokazany linią przerywaną, nie musi przechodzić przez wszystkie punkty kontrolne

pośrednie, które nie należą do krzywej (rys. 9.15). Wektory styczne początkowy i końcowy są określone przez wektory P_1P_2 i P_3P_4 i są związane z R_1 i R_2 zależnościami:

$$R_1 = Q'(0) = 3(P_2 - P_1), \qquad R_4 = Q'(1) = 3(P_4 - P_3) \qquad (9.23)$$

Krzywa Béziera interpoluje dwa końcowe punkty kontrolne i aproksymuje dwa pozostałe. W celu wyjaśnienia, dlaczego w równaniu (9.23) jest stała 3 proponujemy zajrzeć do zadania 9.9. *Macierz geometrii Béziera* G_B składająca się z czterech punktów wygląda następująco:

$$G_B = [P_1\ P_2\ P_3\ P_4] \qquad (9.24)$$

Macierz M_{HB}, która określa relację $G_H = G_B \cdot M_{BH}$ między macierzą geometrii Hermite'a G_H i macierzą geometrii Béziera G_B, jest to macierz 4×4 występująca w następującym równaniu:

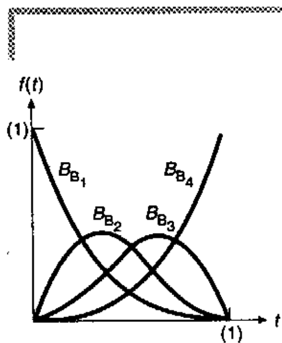
$$G_H = [P_1\ P_4\ R_1\ R_4] = [P_1\ P_2\ P_3\ P_4] \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix} =$$

$$= G_B \cdot M_{HB} \qquad (9.25)$$

W celu znalezienia *macierzy bazowej Béziera* M_B, korzystamy z równania (9.20) dla postaci Hermite'a, podstawiamy $G_H = M_{HB} \cdot G_B$ i definiujemy $M_B = M_H \cdot M_{HB}$:

$$Q(t) = G_H \cdot M_H \cdot T = (G_B \cdot M_{HB}) \cdot M_H \cdot T = G_B \cdot (M_{HB} \cdot M_H) \cdot T =$$

$$= G_B \cdot M_B \cdot T \qquad (9.26)$$



Rys. 9.16. Wielomiany Bernsteina, które są funkcjami wagowymi krzywych Béziera. Dla $t = 0$ tylko B_{B_1} jest różne od zera i krzywa interpoluje P_1; podobnie dla $t = 1$ tylko B_{B_4} jest różne od zera i krzywa interpoluje P_4

Wykonując mnożenie $M_B = M_{HB} \cdot M_H$ otrzymujemy

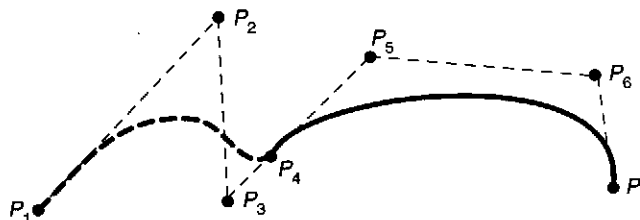
$$M_B = M_{HB} \cdot M_H = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad (9.27)$$

i iloczyn $Q(t) = G_B \cdot M_B \cdot T$ jest równy

$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4 \qquad (9.28)$$

Cztery wielomiany $B_B = M_B \cdot T$, które są wagami w równaniu (9.28), są nazywane *wielomianami Bernsteina* i pokazane na rys. 9.16.

Łączenie segmentów krzywej. Na rysunku 9.17 pokazano dwa segmenty krzywej Béziera ze wspólnym punktem końcowym. Ciągłość G^1 jest zachowana w punkcie końcowym wówczas, gdy $P_3 - P_4 = k(P_4 - P_5)$, $k > 0$. Oznacza to, że trzy punkty P_3, P_4 i P_5 muszą być różne i współliniowe. W bardziej restrykcyjnym przypadku, gdy $k = 1$, obok ciągłości G^1 jest ciągłość C^1.



Rys. 9.17. Dwie krzywe Béziera łączące się w punkcie P_4. Punkty P_3, P_4 i P_5 są współliniowe

Jeżeli segment lewy oznaczymy przez x^l, a prawy przez x^r, to warunki dla ciągłości C^0 i C^1 w punkcie połączenia są następujące:

$$x^l(1) = x^r(0), \qquad \frac{d}{dt}\, x^l(1) = \frac{d}{dt}\, x^r(0) \tag{9.29}$$

skąd dla zmiennej x otrzymujemy:

$$x^l(1) = x^r(0) = P_{4_x}, \qquad \frac{d}{dt}\, x^l(1) = 3(P_{4_x} - P_{3_x}),$$

$$\frac{d}{dt}\, x^r(0) = 3(P_{5_x} - P_{4_x}) \tag{9.30}$$

Jak zwykle, takie same warunki są słuszne dla zmiennych y i z. A więc ciągłość C^0 i C^1 mamy, gdy $P_4 - P_3 = P_5 - P_4$.

Znaczenie wielokąta rozpiętego opisanego na zbiorze punktów. Analizując cztery wielomiany B_B z równania (9.28) i rysunek 9.16 możemy zauważyć, że ich suma jest wszędzie równa jedności i każdy wielomian dla $0 \leqslant t < 1$ jest nieujemny. Stąd $Q(t)$ jest średnią ważoną czterech punktów kontrolnych. Ten warunek oznacza, że każdy segment krzywej, który jest po prostu sumą czterech punktów kontrolnych wziętych z wagami określonymi przez wielomiany, jest całkowicie zawarty w *wielokącie rozpiętym* opisanym na czterech punktach kontrolnych (wielokącie rozpiętym): ten wielokąt można przez analogię porównać z gumką rozpiętą na czterech punktach sterujących (rys. 9.15). Dla krzywych 3D odpowiednikiem wielokąta rozpiętego opisanego na zbiorze punktów jest wielościan rozpięty utworzony przez punkty kontrolne: przez analogię można myśleć o wielościanie utworzonym przez gumkę rozpiętą na zbiorze punktów.

Ta właściwość wielokąta rozpiętego opisanego na zbiorze punktów jest słuszna dla wszystkich krzywych trzeciego stopnia określonych przez sumy ważone punktów kontrolnych, jeżeli funkcje bazowe są nieujemne i suma jest równa jedności. Ogólnie, średnia ważona n punktów znajduje się wewnątrz wielokąta rozpiętego na zbiorze n punktów; to może być intuicyjnie zrozumiałe dla $n = 2$ i $n = 3$ i można to uogólnić. Inną konsekwencją faktu, że suma czterech wielomianów jest równa jedności, jest to, że możemy znaleźć wartość czwartego wielomianu dla dowolnej wartości t odejmując pierwsze trzy od jedności – fakt ten może być wykorzystany do zmniejszenia czasu obliczeń.

Właściwość wielokąta rozpiętego jest również użyteczna przy obcinaniu segmentów krzywej: zamiast obcinać każdy krótki kawałek segmentu krzywej w celu określenia jego widoczności, najpierw stosujemy

algorytm obcinania wielokąta, na przykład algorytm Sutherlanda-Hodgmana omówiony w rozdz. 3 – do obcięcia rozpiętego wielokąta albo opisanego na nim prostokąta przez obszar obcinający. Jeżeli wielokąt rozpięty (opisany prostokąt) jest całkowicie w obszarze obcinania, to również cały segment krzywej znajduje się w tym obszarze. Jeżeli wielokąt rozpięty (opisany prostokąt) znajduje się całkowicie na zewnątrz obszaru obcinania, to również cały segment krzywej jest na zewnątrz. Jedynie jeżeli wielokąt rozpięty (opisany prostokąt) przecina obszar obcinający, trzeba sprawdzić sam segment.

Przykład 9.2 **Problem**: Napisz program, korzystając z SRGP, który umożliwia użytkownikowi określanie czterech punktów kontrolnych dla krzywej Béziera 2D i potem narysowanie krzywej korzystając z podejścia przedstawionego w programie 9.1. Należy zapewnić możliwość specyfikowania dowolnej liczby krzywych Béziera, zerowanie okna SRGP oraz kończenie programu.

Odpowiedź: Realizujemy funkcję DrawCurve, korzystając z równania (9.28), które wiąże krzywą $Q(t)$ z czterema punktami kontrolnymi. Ogólnie ze względu na przejrzystość ta realizacja nie jest efektywna. Korzystamy jednak z funkcji SRGP_polyLine, która zapewnia najefektywniejszy sposób rysowania krzywej. Reszta realizacji jest zgodna z modelem z programu 9.1.

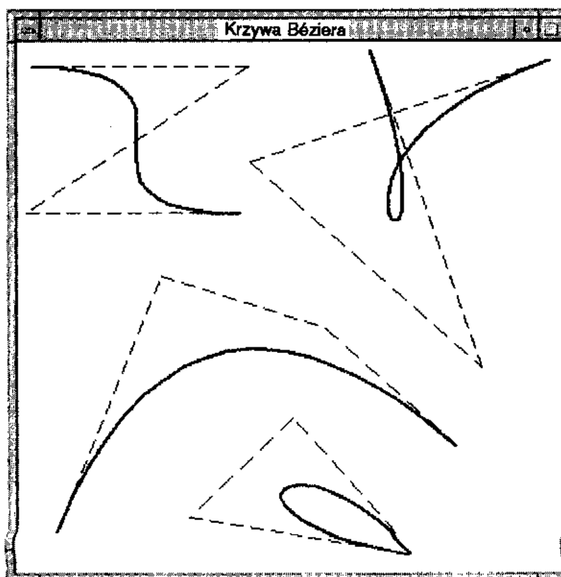
Wielkość okna i liczba używanych kroków do aproksymacji krzywej zostały przyjęte arbitralnie (odpowiednio 400 i 20). Jest wiele możliwych sposobów realizacji interakcyjnej części programu; tutaj została wybrana kombinacja lokalizatora i klawiatury. Prawy przycisk lokalizatora jest używany do specyfikowania początku nowej sekwencji punktów kontrolnych, a lewy przycisk – do określenia pozostałych trzech punktów. Echo w postaci linii typu gumka ułatwia rozmieszczanie punktów. Krzywa Béziera jest rysowana natychmiast po wprowadzeniu ostatniego punktu.

Okno jest zerowane po naciśnięciu przez użytkownika klawisza „c"; naciśnięcie klawisza „q" kończy program. Typowy zestaw krzywych tworzonych przez program pokazano na towarzyszącym rysunku.

Program interakcyjnego rysowania krzywej Béziera

```
#include "srgp.h"
#include <stdio.h>

#define KEYMEASURE_SIZE    80
#define WINDOW_SIZE        400
#define NUM_STEPS          20
```

```
void DrawCurve(point *ControlPoints, int n)
{
   int   i;
   float t, delta;
   point CurvePoints[n];

   CurvePoints[0].x = ControlPoints[0].x;    /* Krzywe Béziera interpolują pierwszy
   CurvePoints[0].y = ControlPoints[0].y;       i ostatni punkt kontrolny */
   delta = 1.0 / n;          /* Krzywa ma być aproksymowana przez n punktów */
                                  /* t zmienia się od 0.0 do 1.0 */
   for (i = 1; i <= n; i++) {
      t = i * delta;
      CurvePoints[i].x = ControlPoints[0].x * (1.0 – t) * (1.0 – t) * (1.0 – t)
      + ControlPoints[1].x * 3.0 * t * (1.0 – t) * (1.0 – t)
      + ControlPoints[2].x * 3.0 * t * t * (1.0 – t)
      + ControlPoints[3].x * t * t * t;

   CurvePoints[i].y = ControlPoints[0].y * (1.0 – t) * (1.0 – t) * (1.0 – t)
      + ControlPoints[1].y * 3.0 * t * (1.0 – t) * (1.0 – t)
      + ControlPoints[2].y * 3.0 * t * t * (1.0 – t)
      + ControlPoints[3].y * t * t * t;
   }
   SRGP_polyLine(n + 1, CurvePoints);      /* Rysowanie kompletnej krzywej */
}
```



Typowy wynik działania programu rysowania krzywych Béziera

```
main()
{
   locator_measure locMeasure, pastlocMeasure;
```

```
    char   keyMeasure[KEYMEASURE_SIZE];
    int    device;
    int    numCtl;
    boolean terminate;
    rectangle screen;
    point  ControlPoints[4];

    SRGP_begin("Bezier Curves", WINDOW_SIZE, WINDOW_SIZE, 1, FALSE);
    SRGP_setLocatorEchoType(CURSOR);
    SRGP_setLocatorButtonMask(LEFT_BUTTON_MASK|RIGHT_BUTTON_MASK);
    pastlocMeasure.position = SRGP_defPoint(-1, -1);    /* Początkowa pozycja
                                                           w dowolnym miejscu */
    SRGP_setLocatorMeasure(pastlocMeasure.position);  /* dowolne ulokowanie */
    SRGP_setKeyboardProcessingMode(RAW);
    SRGP_setInputMode(LOCATOR, EVENT);   /* Zarówno lokalizator (myszka),
    SRGP_setInputMode(KEYBOARD, EVENT);  /* jak i klawiatura są aktywne */
    screen = SRGP_defRectangle(0, 0, WINDOW_SIZE - 1, WINDOW_SIZE - 1);

/* Main event loop */
    terminate = FALSE;
    do {
        device = SRGP_waitEvent(INDEFINITE);
        switch (device) {
        case KEYBOARD:{
                SRGP_getKeyboard(keyMeasure, KEYMEASURE_SIZE);
                switch (keyMeasure[0]) {
                case 'q':                          /* Wyjście z programu */
                    terminate = TRUE;
                    break;
                case 'c':                          /* Zerowanie okna */
                    SRGP_setColor(0);
                    SRGP_fillRectangle(screen);
                    SRGP_setColor(1);
                    break;
                }
                break;
            }                                      /* Klawiatura */
        case LOCATOR:{
                SRGP_getLocator(&locMeasure);
                switch (locMeasure.buttonOfMostRecentTransition) {
                case LEFT_BUTTON: /* Definiowanie pozostałych punktów kontrolnych */
                    if ((locMeasure.buttonChord[LEFT_BUTTON] == DOWN) &&
                        pastlocMeasure.position.x > 0) {
                        SRGP_setLocatorEchoRubberAnchor(locMeasure.position);
                        SRGP_line(pastlocMeasure.position, locMeasure.position);
                        pastlocMeasure = locMeasure;
                        ControlPoints[numCtl] = locMeasure.position;
                        numCtl++;
                        if (numCtl == 4) {
```

```
                    SRGP_setLineStyle(CONTINUOUS);   /* Dla rysowanej krzywej */
                    SRGP_setLineWidth(2);
                    DrawCurve(ControlPoints, NUM_STEPS);
                    pastlocMeasure.position.x = -1;
                    SRGP_setLocatorEchoType(CURSOR);
                }
                break;
            case RIGHT_BUTTON:    /* Start nowego zbioru punktów kontrolnych */
                SRGP_setLocatorEchoRubberAnchor(locMeasure.position);
                pastlocMeasure = locMeasure;
                SRGP_setLocatorEchoType(RUBBER_LINE);
                SRGP_setLineStyle(DASHED);   /* Dla rysowanego wielokąta */
                SRGP_setLineWidth(1);
                ControlPoints[0] = locMeasure.position;
                numCtl = 1;
                break;
            }
          }           /* Obsługa przycisku */
        }             /* Lokalizator */
      }               /* Urządzenie */
    } while (!terminate);
    SRGP_end();
}
```

9.2.4. Jednorodne nieułamkowe krzywe B-sklejane

Angielska nazwa *krzywych B-spline* wzięła się z nazwy używanej przez kreślarzy w odniesieniu do długiej elastycznej taśmy metalowej używanej do rysowania powierzchni samolotów, samochodów i statków. „Obciążniki" dołączane do taśmy umożliwiały wyginanie taśmy w różnych kierunkach. Takie metalowe taśmy, jeżeli nie były nadmiernie wygięte, miały ciągłość drugiego rodzaju. Matematyczny odpowiednik takich taśm – krzywa sklejana trzeciego stopnia – jest to wielomian trzeciego stopnia z ciągłością C^0, C^1 i C^2, która interpoluje (przechodzi przez) punkty kontrolne. Ten wielomian ma jeden stopień ciągłości więcej niż postacie Hermite'a i Béziera. Dlatego krzywe sklejane charakteryzuje lepsza gładkość niż poprzednie postacie.

Współczynniki wielomianów dla naturalnych krzywych trzeciego stopnia zależą jednak od wszystkich n punktów kontrolnych; ich obliczenie wiąże się z odwróceniem macierzy $n + 1 \times n + 1$ [BART87]. Ma to dwie wady: przesuwanie dowolnego punktu kontrolnego wpływa na całą krzywą i czas obliczeń potrzebny do odwrócenia macierzy może być w kolizji z szybkim interakcyjnym zmienianiem kształtu krzywej.

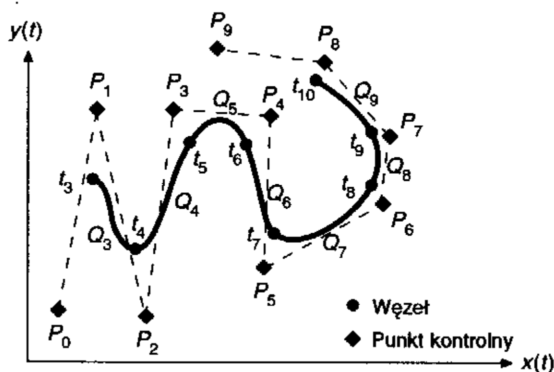
Krzywe B-sklejane omawiane w tym punkcie składają się z segmentów krzywej, których współczynniki wielomianów zależą tylko od kilku

punktów kontrolnych. Takie zachowanie jest określane jako *sterowanie lokalne*. Stąd przesuwanie punktu kontrolnego wpływa tylko na niewielką część krzywej. Ponadto czas potrzebny na obliczenie współczynników jest istotnie skrócony. Krzywe B-sklejane mają taką samą ciągłość jak naturalne krzywe sklejane, ale nie interpolują punktów kontrolnych.

W dalszej dyskusji zmieniamy nieco notację, ponieważ musimy omawiać całą krzywą składającą się z kilku segmentów, a nie jej poszczególne segmenty. Segment krzywej nie musi przechodzić przez jej punkty kontrolne i dwa warunki ciągłości segmentu wynikają z sąsiednich segmentów. Takie zachowanie wynika z istnienia wspólnych punktów kontrolnych dla segmentów i lepiej jest opisać proces w zależności od wszystkich segmentów jednocześnie.

Krzywe B-sklejane trzeciego stopnia aproksymują ciąg $m + 1$ punktów kontrolnych $P_0, P_1, ..., P_m$, przy czym $m \geqslant 3$, krzywą składającą się z $m - 2$ segmentów krzywych wielomianowych trzeciego stopnia Q_3, $Q_4, ..., Q_m$. Chociaż takie krzywe trzeciego stopnia mogłyby być określane każda w swoim własnym przedziale $0 \leqslant t < 1$, możemy tak dobierać ten parametr (przez podstawienie o postaci $t = t + k$), żeby zakresy parametru dla różnych segmentów krzywej były sekwencyjne. Mówimy, że zakres parametru, dla którego Q_i jest określone, jest przedziałem $t_i \leqslant t < t_{i+1}$, dla $3 \leqslant i \leqslant m$. W szczególnym przypadku dla $m = 3$ jest jeden segment krzywej Q_3 określony w przedziale $t_3 \leqslant t < t_4$ przez cztery punkty kontrolne P_0 do P_3.

Dla każdego $i \geqslant 4$, jest punkt połączenia albo *węzeł* między Q_{i-1} i Q_i, który jest zdefiniowany dla parametru o wartości t_i; wartość parametru w tym punkcie jest określana jako *wartość węzłowa*. Punkty początkowy i końcowy w t_3 i t_{m+1} są również nazywane węzłami i w sumie jest $m - 1$ węzłów. Na rysunku 9.18 pokazano krzywą B-sklejaną



Rys. 9.18. Krzywa B-sklejana z segmentami krzywej $Q_3 ... Q_9$. Tę krzywą i wiele innych w tym rozdziale utworzono za pomocą programu napisanego przez Carlesa Castellsaquè'a

2D z zaznaczonymi węzłami. Łatwo można utworzyć krzywą zamkniętą: punkty sterujące P_0, P_1, P_2 powtarza się na końcu sekwencji – P_0, P_1, ..., P_m, P_0, P_1, P_2.

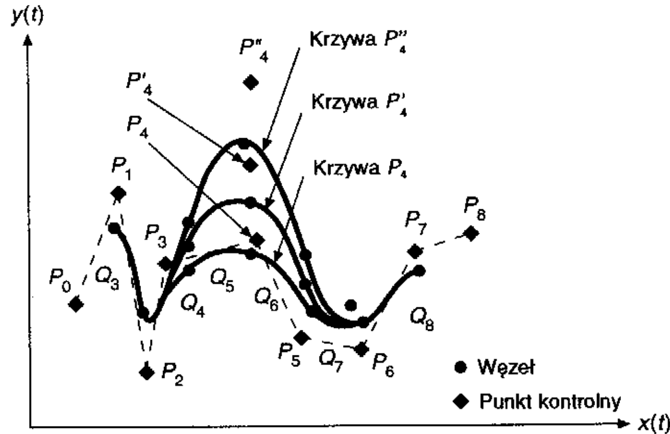
Określenie *jednorodna* oznacza, że węzły są w jednakowych odstępach, jeśli chodzi o parametr t. Bez utraty ogólności możemy założyć, że $t_3 = 0$ i przedział $t_{i+1} - t_i = 1$. Niejednorodne nieułamkowe krzywe B-sklejane są omawiane w p. 9.2.5. (Koncepcja węzła jest wprowadzana w tym punkcie po to, żeby móc wprowadzić krzywe sklejane niejednorodne.) Określenie nieułamkowe jest używane w celu odróżnienia tych krzywych sklejanych od ułamkowych krzywych wielomianowych trzeciego stopnia, omawianych w p. 9.2.6, przy czym $x(t)$, $y(t)$ i $z(t)$ są określone jako stosunki dwóch wielomianów trzeciego stopnia. „B'' jest związane z bazą, ponieważ krzywe sklejane mogą być reprezentowane jako sumy ważone wielomianowych funkcji bazowych, w odróżnieniu od naturalnych krzywych sklejanych, dla których właściwość sumy ważonej nie jest prawdziwa.

Każdy z $m - 2$ segmentów krzywej B-sklejanej jest określony przez cztery z $m + 1$ punktów kontrolnych. W szczególności segment Q_i jest określony przez punkty P_{i-3}, P_{i-2}, P_{i-1} i P_i. Stąd macierz geometrii G_{Bs_i} krzywej B-sklejanej dla segmentu Q_i jest

$$G_{Bs_i} = [P_{i-3}\ P_{i-2}\ P_{i-1}\ P_i], \qquad 3 \leqslant i \leqslant m \tag{9.31}$$

Pierwszy segment Q_3 krzywej jest określony przez punkty P_0 do P_3 dla zakresu parametru t od $t_3 = 0$ do $t_4 = 1$; Q_4 jest określony przez punkty od P_1 do P_4 dla zakresu od $t_4 = 1$ do $t_5 = 2$ i ostatni segment krzywej Q_m jest określony przez punkty P_{m-3}, P_{m-2}, P_{m-1} i P_m dla zakresu od $t_m = m - 3$ do $t_{m+1} = m - 2$. Ogólnie segment Q_i krzywej zaczyna się gdzieś w okolicy punktu P_{i-2} i kończy gdzieś blisko punktu P_{i-1}. Zobaczymy, że funkcje bazowe dla krzywej B-sklejanej są wszędzie nieujemne i sumują się do jedności, tak że segment Q_i jest ograniczony do wielokąta rozpiętego na czterech punktach kontrolnych.

Każdy segment krzywej jest określony przez cztery punkty kontrolne i każdy punkt kontrolny (prócz tych na początku i końcu sekwencji P_0, P_1, ..., P_m) wpływa na cztery segmenty krzywej. Przesuwanie punktu kontrolnego w danym kierunku powoduje przesunięcie czterech segmentów krzywej, na które ten punkt wpływa, w tym samym kierunku; punkt ten nie ma żadnego wpływu na pozostałe segmenty krzywej (rys. 9.19). Takie zachowanie jest związane z właściwością lokalnego sterowania krzywymi B-sklejanymi i innymi krzywymi sklejanymi omawianymi w tym rozdziale.



Rys. 9.19. Krzywa B-sklejana z punktem P_4 znajdującym się w różnych miejscach

Jeżeli zdefiniujemy T_i jako wektor kolumnowy $[(t-t_i)^3 \; (t-t_i)^2 \; (t-t_i) \; 1]^T$, to segment i krzywej B-sklejanej można opisać następująco:

$$Q_i(t) = G_{Bs_i} \cdot M_{Bs} \cdot T_i, \qquad t_i \leqslant t < t_{i+1} \tag{9.32}$$

Całą krzywą generujemy korzystając z równania (9.23) dla $3 \leqslant i \leqslant m$.

Macierz bazowa krzywej B-sklejanej M_{Bs} wiąże geometryczne ograniczenia G_{Bs} z funkcjami bazowymi i współczynnikami wielomianu:

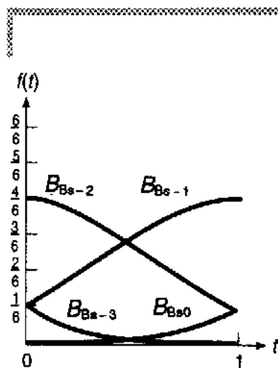
$$M_{Bs} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \tag{9.33}$$

Wyprowadzenie macierzy można znaleźć w pracy [BART87].

Funkcje bazowe B_{Bs} krzywej B-sklejanej są określone przez iloczyn $M_{Bs} \cdot T_i$, analogicznie do poprzednich sformułowań Béziera i Hermite'a. Zauważmy, że funkcje bazowe dla każdego segmentu krzywej są dokładnie takie same, ponieważ dla każdego segmentu i wartości $t - t_i$ zmieniają się od 0 dla $t = t_i$ do 1 dla $t = t_{i+1}$. Jeżeli zastąpimy $t - t_i$ przez t i przedział $[t_i, t_{i+1}]$ przez [0, 1] otrzymamy

$$\begin{aligned} B_{Bs} &= M_{Bs} \cdot T = [B_{Bs_{-3}} B_{Bs_{-2}} B_{Bs_{-1}} B_{Bs_0}]^T = \\ &= \frac{1}{6}[-t^3+3t^2-3t+1 \quad 3t^3-6t^2+4 \quad -3t^3+3t^2+3t+1 \quad t^3]^T = \\ &= \frac{1}{6}[(1-t)^3 \quad 3t^3-6t^2+4 \quad -3t^3+3t^2+3t+1 \quad t^3]^T, \qquad 0 \leqslant t < 1 \end{aligned} \tag{9.34}$$

Na rysunku 9.20 pokazano funkcje bazowe B_{Bs} krzywej B-sklejanej. Ponieważ te cztery funkcje sumują się do jedności i są nieujemne, dla każdego segmentu krzywej B-sklejanej obowiązuje właściwość wielokąta rozpiętego. W pracy [BART87] wyjaśniono relację między tymi funkcjami bazowymi i funkcjami bazowymi wielomianów Bernsteina.



Rys. 9.20. Cztery funkcje bazowe krzywej B-sklejanej z równania (9.34). Dla $t = 0$ i $t = 1$ trzy funkcje są różne od zera

Rozszerzając równanie (9.23) i ponownie zastępując $t - t_i$ przez t przy drugim znaku równości otrzymujemy

$$Q_i(t - t_i) = G_{\mathrm{Bs}_i} \cdot M_{\mathrm{Bs}} \cdot T_i = G_{\mathrm{Bs}_i} \cdot M_{\mathrm{Bs}} \cdot T = G_{\mathrm{Bs}_i} \cdot B_{\mathrm{Bs}} =$$

$$= P_{i-3} \cdot B_{\mathrm{Bs}_{-3}} + P_{i-2} \cdot B_{\mathrm{Bs}_{-2}} + P_{i-1} \cdot B_{\mathrm{Bs}_{-1}} + P_i \cdot B_{\mathrm{Bs}_0} =$$

$$= \frac{(1-t)^3}{6} P_{i-3} + \frac{3t^3 - 6t^2 + 4}{6} P_{i-2} +$$

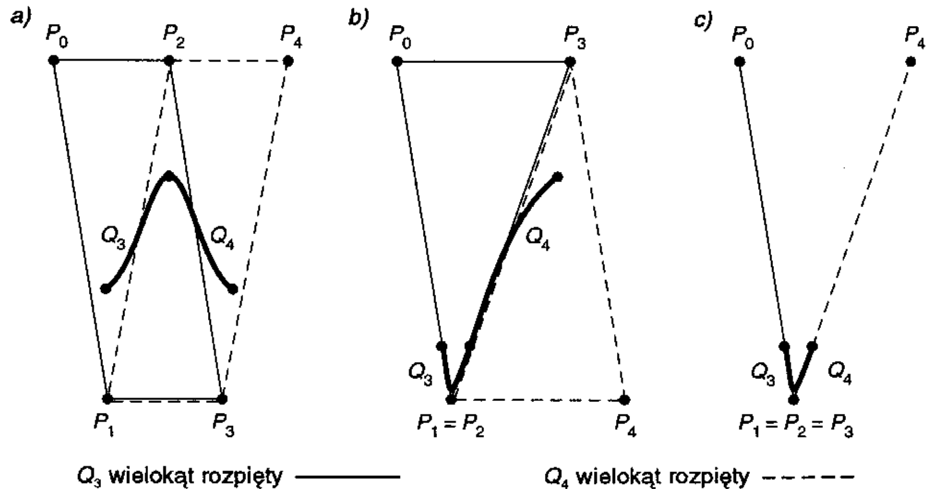
$$+ \frac{-3t^3 + 3t^2 + 3t + 1}{6} P_{i-1} + \frac{t^3}{6} P_i, \qquad 0 \leqslant t < 1 \qquad (9.35)$$

Można łatwo wykazać, że Q_i i Q_{i+1} mają ciągłość C^0, C^1 i C^2 w miejscu połączenia. Dodatkowa ciągłość występująca dla krzywych B-sklejanych jest atrakcyjna, ale okupuje się ją mniejszą kontrolą nad przebiegiem krzywej. Możemy wymuszać, żeby krzywa interpolowała określone punkty przez powielanie punktów kontrolnych; jest to użyteczne zarówno w punktach końcowych, jak i w punktach pośrednich na krzywej. Na przykład, jeżeli $P_{i-2} = P_{i-1}$, to krzywa jest wypychana w kierunku tego punktu, ponieważ segment Q_i krzywej jest określony przez trzy różne punkty, a punkt $P_{i-2} = P_{i-1}$ jest ważony dwukrotnie w równaniu (9.35) – raz przez $B_{\mathrm{Bs}_{-2}}$ i raz przez $B_{\mathrm{Bs}_{-1}}$.

Jeżeli punkt kontrolny jest używany trzy razy – na przykład, jeżeli $P_{i-2} = P_{i-1} = P_i$ – to równanie (9.35) przyjmuje postać

$$Q_i(t) = P_{i-3} \cdot B_{\mathrm{Bs}_{-3}} + P_i(B_{\mathrm{Bs}_{-2}} + B_{\mathrm{Bs}_{-1}} + B_{\mathrm{Bs}_0}) \qquad (9.36)$$

Q_i jest oczywiście odcinkiem. Następnie punkt P_{i-2} jest interpolowany przez odcinek dla $t = 1$, przy czym trzy wagi zastosowane do P_i sumują się do 1, ale P_{i-3} na ogół nie jest interpolowany dla $t = 0$. O tym zachowaniu można również myśleć w następujący sposób: wielokąt rozpięty dla Q_i jest teraz określony przez dwa różne punkty, a więc Q_i musi być odcinkiem. Na rysunku 9.21 pokazano wpływ różnych punktów kontrolnych na wnętrze krzywej B-sklejanej.



Rys. 9.21. Wpływ wielokrotnych punktów kontrolnych na jednorodną krzywą B-sklejaną. Na rysunku a) nie ma wielokrotnych punktów kontrolnych. Wypukłe wielokąty rozpięte dwóch krzywych zachodzą na siebie; punkt połączenia między Q_3 i Q_4 jest w obszarze wspólnym dla obu wielokątów. Na rysunku b) jest podwójny punkt kontrolny i wypukłe wielokąty rozpięte mają wspólną krawędź P_2P_3; punkt połączenia musi leżeć na tej krawędzi. Na rysunku c) jest potrójny punkt kontrolny i oba wypukłe wielokąty są odcinkami, dla których ten punkt potrójny jest wspólny; stąd punkt wspólny jest również punktem potrójnym. Ponieważ wielokąty wypukłe są odcinkami, oba segmenty krzywej również muszą być odcinkami. W punkcie połączenia jest ciągłość C^2 i tylko ciągłość G^0

Inna metoda interpolowania punktów końcowych (z wierzchołkami pozornymi) jest omawiana w pracach [BARS83, BART87]. W następnym punkcie zobaczymy, że niejednorodne krzywe B-sklejane umożliwiają w sposób bardziej naturalny interpolowanie punktów końcowych i punktów wewnętrznych niż w przypadku jednorodnych krzywych B-sklejanych.

9.2.5. Niejednorodne nieułamkowe krzywe B-sklejane

Niejednorodne nieułamkowe krzywe B-sklejane różnią się od jednorodnych nieułamkowych krzywych B-sklejanych omówionych w p. 9.2.4 tym, że przedział parametru między kolejnymi wartościami węzłowymi nie musi być równomierny. Niejednorodna sekwencja wartości węzłowych oznacza, że funkcje tworzące nie są takie same dla każdego przedziału i zmieniają się między poszczególnymi segmentami krzywej.

Takie krzywe mają kilka zalet w porównaniu z jednorodnymi krzywymi B-sklejanymi. Po pierwsze, ciągłość w wybranych węzłach może być zredukowana z C^2 do C^1 albo do C^0, albo w ogóle nie ma ciągłości. Jeżeli ciągłość zostanie zredukowana do C^0, to krzywa interpoluje

pierwszy punkt kontrolny, ale bez niepożądanego efektu występującego w jednorodnych krzywych B-sklejanych, polegającego na tym, że segmenty z każdej strony interpolowanego punktu kontrolnego są odcinkami. Można również łatwo dokładnie interpolować punkty początkowy i końcowy bez równoczesnego wprowadzania segmentów liniowych. Jak pokazano w pracy [FOLE90], można dodać dodatkowy węzeł i punkt kontrolny do niejednorodnej krzywej B-sklejanej tak, żeby można było łatwo zmieniać kształt wynikowej krzywej; takiej modyfikacji nie można zrobić w jednorodnych krzywych B-sklejanych.

Większa ogólność niejednorodnych krzywych B-sklejanych wymaga nieco innej notacji niż w przypadku jednorodnych krzywych B-sklejanych. Tak jak poprzednio krzywa sklejana jest krzywą kawałkami ciągłą złożoną z wielomianów trzeciego stopnia aproksymującą punkty kontrolne od P_0 do P_m. *Sekwencja wartości węzłowych* jest nierosnącą sekwencją wartości węzłowych od t_0 do t_{m+4} (to znaczy jest o cztery więcej węzłów niż punktów kontrolnych). Ponieważ najmniejsza liczba punktów kontrolnych wynosi cztery, najmniejsza sekwencja węzłów ma osiem wartości węzłowych i krzywa jest określona w przedziale parametrów od t_3 do t_4.

Jedynym ograniczeniem co do sekwencji węzłów jest to, że musi być niemalejąca i dlatego kolejne wartości węzłów mogą być równe. Gdy tak się stanie, wartość parametru jest nazywana *węzłem wielokrotnym*, a liczba identycznych wartości parametru – *wielokrotnością węzła* (jeden węzeł ma wielokrotność 1). Na przykład w sekwencji węzłów (0,0,0,0,1,1,2,3,4,4,5,5,5,5) węzeł 0 ma wielokrotność 4; węzeł 1 ma wielokrotność 2; węzły 2 i 3 mają wielokrotności 1; węzeł 4 ma wielokrotność 2, a węzeł 5 ma wielokrotność 4.

Segment Q_i krzywej jest określony przez punkty kontrolne P_{i-3}, P_{i-2}, P_{i-1}, P_i i przez funkcje bazowe $B_{i-3,4}(t)$, $B_{i-2,4}(t)$, $B_{i-1,4}(t)$, $B_{i,4}(t)$ jako suma ważona

$$Q_i(t) = P_{i-3} \cdot B_{i-3,4}(t) + P_{i-2} \cdot B_{i-2,4}(t) + P_{i-1} \cdot B_{i-1,4}(t) + \\ + P_i \cdot B_{i,4}(t) \qquad 3 \leqslant i \leqslant m, \qquad t_i \leqslant t < t_{i+1} \tag{9.37}$$

Krzywa nie jest określona na zewnątrz przedziału od t_3 do t_{m+1}. Gdy $t_i = t_{i+1}$ (węzeł wielokrotny) segment Q_i jest jednym punktem. Możliwość redukowania segmentu do punktu jest dodatkową cechą świadczącą o elastyczności niejednorodnych krzywych B-sklejanych.

Nie ma jednego zbioru funkcji bazowych, tak jak to było dla innych typów krzywych sklejanych. Funkcje zależą od przedziałów między wartościami węzłów i są określone rekursywnie w zależności od funkcji bazowych niższego rzędu. $B_{i,j}(t)$ jest funkcją bazową j-tego rzędu dla wagowego punktu kontrolnego P_i. Ponieważ zajmujemy się krzy-

wymi B-sklejanymi czwartego rzędu (to jest trzeciego stopnia), rekursywna definicja kończy się na $B_{i,4}(t)$ i może być łatwo przedstawiona w „rozwiniętej" postaci. Rekurencja dla krzywej B-sklejanej wygląda następująco:

$$B_{i,1}(t) = \begin{cases} 1, & t_i \leqslant t < t_{i+1} \\ 0, & \text{w przeciwnym przypadku} \end{cases}$$

$$B_{i,2}(t) = \frac{t - t_i}{t_{i+1} - t_i} B_{i,1}(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} B_{i+1,1}(t)$$

$$B_{i,3}(t) = \frac{t - t_i}{t_{i+2} - t_i} B_{i,2}(t) + \frac{t_{i+3} - t}{t_{i+3} - t_{i+1}} B_{i+1,2}(t)$$

$$B_{i,4}(t) = \frac{t - t_i}{t_{i+3} - t_i} B_{i,3}(t) + \frac{t_{i+4} - t}{t_{i+4} - t_{i+1}} B_{i+1,3}(t) \qquad (9.38)$$

Można wykazać, że funkcje bazowe są nieujemne i sumują się do jedności, zatem segmenty niejednorodnej krzywej B-sklejanej leżą wewnątrz wielokątów rozpiętych na ich czterech punktach kontrolnych. Dla węzłów o wielokrotności większej od jedności mianowniki mogą być zerami ze względu na równość kolejnych wartości węzłowych: dzielenie przez zero z definicji ma dawać zero.

Zwiększenie wielokrotności węzła ma dwa skutki. Po pierwsze, krzywa sklejana oszacowana dla dowolnej wartości węzła t_i automatycznie daje punkt wewnątrz wypukłego wielokąta rozpiętego na punktach P_{i-3}, P_{i-2} i P_{i-1}. Jeżeli t_i oraz t_{i+1} są równe, to muszą leżeć w wielokącie rozpiętym na P_{i-3}, P_{i-2} i P_{i-1} i w wielokącie rozpiętym na punktach P_{i-2}, P_{i-1} i P_i. Stąd muszą leżeć na odcinku segmentu między P_{i-2} i P_{i-1}. Podobnie, jeżeli $t_i = t_{i+1} = t_{i+2}$, to ten węzeł musi leżeć w P_{i-1}. Jeżeli $t_i = t_{i+1} = t_{i+2} = t_{i+3}$, to węzeł musi leżeć zarówno w P_{i-1}, jak i w P_i – krzywa zostaje przerwana. Po drugie, wielokrotne węzły redukują ciągłość parametryczną: od ciągłości C^2 do C^1 dla jednego dodatkowego węzła (wielokrotność 2); od ciągłości C^1 do C^0 dla dwóch dodatkowych węzłów (wielokrotność 3); od ciągłości C^0 do braku ciągłości dla trzech dodatkowych węzłów (wielokrotność 4).

Interakcyjne tworzenie niejednorodnych krzywych sklejanych na ogół wiąże się ze wskazywaniem punktów kontrolnych; punkty wielokrotne wybiera się wskazując kilka razy ten sam punkt. Inny sposób polega na bezpośrednim wskazywaniu krzywej za pomocą myszki z wieloma przyciskami; podwójne naciśnięcie przycisku może oznaczać podwójny punkt sterujący; podwójne naciśnięcie innego przycisku może oznaczać podwójny węzeł.

9.2.6. Niejednorodne ułamkowe segmenty wielomianowej krzywej trzeciego stopnia

Ogólnie segmenty ułamkowej krzywej trzeciego stopnia są stosunkami wielomianów:

$$x(t) = \frac{X(t)}{W(t)}, \qquad y(t) = \frac{Y(t)}{W(t)}, \qquad z(t) = \frac{Z(t)}{W(t)} \tag{9.39}$$

przy czym $X(t)$, $Y(t)$, $Z(t)$ i $W(t)$ są krzywymi wielomianowymi trzeciego stopnia, których punkty kontrolne są określone we współrzędnych jednorodnych. Możemy również myśleć, że krzywa istnieje w przestrzeni jednorodnej jako $Q(t) = [X(t)\ Y(t)\ Z(t)\ W(t)]^T$. Jak zwykle przejście od przestrzeni jednorodnej do przestrzeni trójwymiarowej wiąże się z dzieleniem przez $W(t)$. Możemy przekształcić każdą nieułamkową krzywą w krzywą ułamkową dodając $W(t) = 1$ jako czwarty element. Ogólnie wielomianami w krzywej ułamkowej mogą być wielomiany typu Béziera, Hermite'a albo inne. Jeżeli są to krzywe B-sklejane, to mamy niejednorodne ułamkowe krzywe B-sklejane, określane jako krzywe *NURBS* [FORR80].

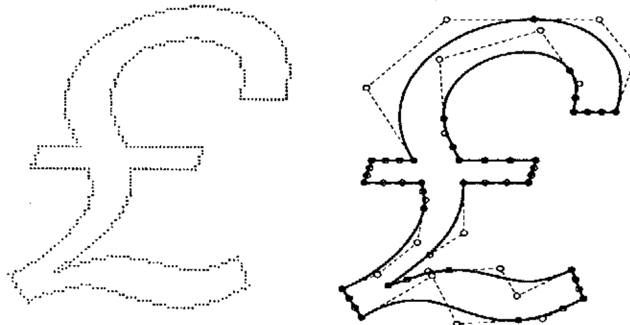
Krzywe ułamkowe są użyteczne z dwóch względów. Pierwszą i najważniejszą przyczyną jest to, że są one niezmiennicze względem przekształceń obrotu, skalowania, przesunięcia i perspektywy punktów kontrolnych (krzywe nieułamkowe są niezmiennicze tylko względem obrotu, skalowania i przesunięcia). Dlatego przekształcenie perspektywiczne trzeba stosować tylko do punktów kontrolnych, które potem mogą być wykorzystane do generowania przekształceń perspektywicznych oryginalnej krzywej. Alternatywą dla konwersji krzywej nieułamkowej na krzywą ułamkową przed przekształceniem perspektywicznym jest wygenerowanie najpierw punktów na samej krzywej, a potem zastosowanie przekształcenia perspektywicznego do każdego punktu – jest to znacznie mniej efektywny proces. Podobnie można zauważyć, że przekształcenie perspektywiczne kuli to nie to samo co kula, której środek i promień są przekształconymi środkiem i promieniem oryginalnej kuli.

Drugą zaletą krzywych ułamkowych sklejanych jest to, że mogą one, w przeciwieństwie do krzywych nieułamkowych, dokładnie definiować dowolny przekrój stożka. Za pomocą krzywych nieułamkowych możemy jedynie aproksymować stożek, korzystając z wielu punktów kontrolnych w pobliżu stożka. Ta druga właściwość jest użyteczna w tych zastosowaniach, zwłaszcza CAD-owskich, w których są potrzebne ogólne krzywe i powierzchnie oraz krzywe stożkowe. Oba rodzaje krzywych można zdefiniować za pomocą krzywych NURBS.

Dalszą dyskusję na temat krzywych stożkowych i NURBS można znaleźć w pracach [FAUX79; BÖHM84; TILL83].

9.2.7. Dopasowywanie krzywych do zbioru punktów

Inżynier albo artysta często ma nieelektroniczną reprezentację złożonego kształtu, który może zostać przedstawiony w postaci cyfrowej za pomocą zbioru dyskretnych punktów. Na przykład może być dostępna tylko kopia papierowa kształtu. W celu wykonania dodatkowych manipulacji w odniesieniu do kształtu możemy chcieć dopasować gładką krzywą albo ciąg krzywych do (zazwyczaj) nieprecyzyjnej reprezentacji cyfrowej. Były opublikowane różne metody dopasowania krzywej; mają one różne zalety i wady. Schneider [SCHN90] opracował metodę aproksymowania krzywych przedstawionych w postaci cyfrowej za pomocą segmentów Béziera. Zaletami w porównaniu z poprzednimi podejściami



Rys. 9.22. Znak przedstawiony w postaci cyfrowej; pokazano oryginalny kształt, dopasowaną krzywą i punkty kontrolne Béziera (Za zgodą Academic Press, Inc.)

są geometryczna ciągłość, stabilność i łatwość realizacji. Kompletną realizację algorytmu w języku C podano w pracy [SCHN90]. Na rysunku 9.22 pokazano przykład metody zastosowanej do kształtu przedstawionego w postaci cyfrowej.

9.2.8. Porównanie krzywych trzeciego stopnia

Różne rodzaje krzywych parametrycznych trzeciego stopnia można porównywać za pomocą rozmaitych kryteriów, np. łatwość manipulacji interakcyjnych, stopień ciągłości w punktach połączenia, ogólność i szybkość obliczeń przy wykorzystaniu tych reprezentacji. Oczywiście nie jest konieczne wybranie jednej reprezentacji, ponieważ można prze-

chodzić z jednej reprezentacji na drugą, tak jak to pokazano w pracy [FOLE90]. Na przykład niejednorodne ułamkowe krzywe B-sklejane mogą być używane jako wewnętrzna reprezentacja, użytkownik może natomiast interakcyjnie manipulować punktami kontrolnymi Béziera albo Hermite'a i wektorami stycznych. Niektóre interakcyjne edytory graficzne udostępniają użytkownikowi krzywe Hermite'a, podczas gdy wewnętrznie korzystają z reprezentacji Béziera występującej w PostScripcie [ADOB85]. Ogólnie użytkownik interakcyjnego systemu CAD może mieć do dyspozycji krzywe Hermite'a, Béziera, jednorodne krzywe B-sklejane i niejednorodne krzywe B-sklejane. Jest prawdopodobne, że wewnętrzna reprezentacja będzie wykorzystywała niejednorodne ułamkowe krzywe B-sklejane ze względu na ich największą ogólność.

W tablicy 9.1 porównano większość rodzajów krzywych omówionych w tym rozdziale. Łatwość interakcyjnych manipulacji nie jest pokazana w tablicy bezpośrednio, ponieważ ten atrybut zależy silnie od zastosowania. Liczba parametrów sterujących segmentu krzywej obejmuje cztery ograniczenia geometryczne plus inne parametry. Łatwość uzyskania ciągłości odnosi się do takich ograniczeń jak wymuszanie współliniowości punktów kontrolnych w celu uzyskania ciągłości G^1. Ponieważ ciągłość C^n jest bardziej restrykcyjna niż G^n, dowolna forma, która może osiągnąć C^n, może również z definicji osiągnąć przynajmniej G^n.

Tablica 9.1. Porównanie czterech postaci krzywych parametrycznych trzeciego stopnia

	Hermite	Bézier	Jednorodne B-sklejane	Niejednorodne B-sklejane
Wielokąt rozpięty na punktach kontrolnych	N/A	Tak	Tak	Tak
Interpolacja niektórych punktów kontrolnych	Tak	Tak	Nie	Nie
Interpolacja wszystkich punktów kontrolnych	Tak	Nie	Nie	Nie
Łatwość podziału	Dobra	Najlepsza	Średnia	Wysoka
Ciągłości właściwe dla reprezentacji	C^0G^0	C^0G^0	C^2G^2	C^2G^2
Łatwo osiągalne ciągłości	C^1G^1	C^1G^1	C^2G^2*	C^2G^2*
Liczba parametrów sterujących segmentem krzywej	4	4	4	5

* Poza specjalnym przypadkiem omówionym w p. 9.2.

Jeżeli jest wymagana tylko ciągłość geometryczna, jak to ma często miejsce w zastosowaniach CAD, wybór jest ograniczony do różnych rodzajów krzywych sklejanych, które umożliwiają osiągnięcie ciągłości zarówno G^1, jak i G^2. Spośród trzech typów krzywych sklejanych w tablicy najbardziej ograniczające są jednorodne krzywe B-sklejane. Wielokrotne węzły dostępne w niejednorodnych krzywych B-sklejanych dają użytkownikowi dużą możliwość sterowania kształtem. Oczywiście ważny jest również dobry interfejs użytkownika, który umożliwia łatwe korzystanie z tej cechy.

Regułą jest stwarzanie użytkownikowi możliwości interakcyjnego przeciągania punktów kontrolnych albo wektorów stycznych z równoczesnym ciągłym uaktualnianiem krzywej sklejanej. Na rysunku 9.19 pokazano taką sekwencję krzywych B-sklejanych. Jedną z wad krzywych B-sklejanych w niektórych zastosowaniach jest to, że punkty sterujące nie leżą na krzywej. Można jednak nie wyświetlać punktów kontrolnych, lecz umożliwić użytkownikowi w zamian korzystanie z węzłów (które muszą być zaznaczone tak, żeby można je było wybrać).

9.3. Parametryczne powierzchnie bikubiczne

Parametryczne powierzchnie bikubiczne są uogólnieniem parametrycznych krzywych trzeciego stopnia. Przypomnijmy ogólną postać krzywej parametrycznej trzeciego stopnia $Q(t) = G \cdot M \cdot T$, przy czym G, macierz geometrii, jest stała. Najpierw ze względu na wygodę notacyjną zastąpmy t przez s; stąd $Q(s) = G \cdot M \cdot T$. Jeżeli teraz przyjmiemy, że punkty w G zmieniają się w 3D wzdłuż pewnej ścieżki z parametrem t, to otrzymamy

$$Q(s,t) = [G_1(t)\ G_2(t)\ G_3(t)\ G_4(t)] \cdot M \cdot S \tag{9.40}$$

Teraz dla ustalonego t_1, $Q(s,t_1)$ jest krzywą, ponieważ $G(t_1)$ jest stałe. Jeżeli t przyjmie jakąś nową wartość, powiedzmy t_2, przy czym $t_2 - t_1$ jest bardzo małe, to $Q(s,t)$ jest nieco inną krzywą. Powtarzając ten proces dla dowolnie wielu innych wartości t_2 między 0 a 1, definiujemy rodzinę krzywych leżących dowolnie blisko siebie. Zbiór wszystkich takich krzywych określa powierzchnię. Jeżeli $G_i(t)$ same są krzywymi trzeciego stopnia (kubikami), to o powierzchni mówimy, że jest *parametryczną powierzchnią bikubiczną*.

Jeżeli $G_i(t)$ są krzywymi trzeciego stopnia, to każda z nich może być reprezentowana jako $G_i(t) = \mathbf{G}_i \cdot M \cdot T$, przy czym $\mathbf{G}_i = [g_{i1}\ g_{i2}\ g_{i3}\ g_{i4}]$ (używamy oznaczeń $\mathbf{G}$ i g dla odróżnienia od G występującego w opisie krzywej). Stąd g_{i1} jest pierwszym elementem macierzy geometrii dla krzywej $G_i(t)$ itd.

Dokonajmy teraz transpozycji równania $G_i(t) = G_i \cdot M \cdot T$, korzystając z równości $(A \cdot B \cdot C)^T = C^T \cdot B^T \cdot A^T$. W wyniku otrzymujemy $G_i(t) = T^T \cdot M^T \cdot G_i^T = T^T \cdot M^T \cdot [g_{i1}\ g_{i2}\ g_{i3}\ g_{i4}]^T$. Jeżeli teraz podstawimy ten wynik do równania (9.40) dla każdego z czterech punktów, to otrzymamy

$$Q(s, t) = T^T \cdot M^T \cdot \begin{bmatrix} g_{11} & g_{21} & g_{31} & g_{41} \\ g_{12} & g_{22} & g_{32} & g_{42} \\ g_{13} & g_{23} & g_{33} & g_{43} \\ g_{14} & g_{24} & g_{34} & g_{44} \end{bmatrix} \cdot M \cdot S \qquad (9.41)$$

albo

$$Q(s, t) = T^T \cdot M^T \cdot G \cdot M \cdot S, \quad 0 \leqslant s, t \leqslant 1 \qquad (9.42)$$

W rozdzielonym zapisie dla x, y, z otrzymujemy:

$$\begin{aligned} x(s, t) &= T^T \cdot M^T \cdot G_x \cdot M \cdot S \\ y(s, t) &= T^T \cdot M^T \cdot G_y \cdot M \cdot S \\ z(s, t) &= T^T \cdot M^T \cdot G_z \cdot M \cdot S \end{aligned} \qquad (9.43)$$

Mając tę ogólną postać przejdziemy teraz do omówienia specjalnych sposobów określania powierzchni korzystając z różnych macierzy geometrii.

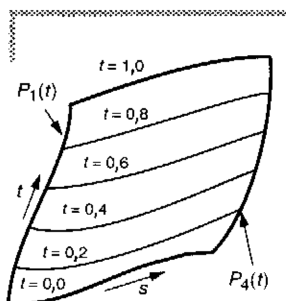
9.3.1. Powierzchnie Hermite'a

Powierzchnie Hermite'a są w pełni określone przez macierz geometrii G_H 4×4. Wyprowadzenie G_H jest podobne jak w przypadku równania (9.42). Pokażemy tutaj wyprowadzenie dla $x(s, t)$. Najpierw zastępujemy t przez s w równaniu (9.13) i otrzymujemy $x(s) = G_{H_x} \cdot M_H \cdot S$. Przepisując to wyrażenie z uwzględnieniem faktu, że macierz geometrii Hermite'a G_{H_x} nie jest stała, lecz jest funkcją t otrzymujemy

$$x(s, t) = G_{H_x}(t) \cdot M_H \cdot S = [P_{1_x}(t)\ P_{4_x}(t)\ R_{1_x}(t)\ R_{4_x}(t)] \cdot M_H \cdot S \qquad (9.44)$$

Funkcje $P_{1_x}(t)$ i $P_{4_x}(t)$ określają składowe x punktów początkowego i końcowego dla krzywej w zależności od parametru s. Podobnie $R_{1_x}(t)$ i $R_{4_x}(t)$ są wektorami stycznymi w tych punktach. Dla dowolnej

wartości t są dwa określone punkty końcowe i wektory styczne. Na rysunku 9.23 pokazano krzywe $P_1(t)$ i $P_4(t)$ oraz krzywe trzeciego stopnia zmienne w funkcji s i określone dla t = 0,0, 0,2, 0,4, 0,6 0,8 i 1,0. Płat powierzchni jest w istocie sześcienną interpolacją między $P_1(t) = Q(0, t)$ a $P_4(t) = Q(1,t)$ albo alternatywnie między $Q(s, 0)$ a $Q(s, 1)$.



Rys. 9.23. Linie stałej wartości parametru na powierzchni bikubicznej: $P_1(t)$ jest dla $s = 0$ i $P_4(t)$ jest dla $s = 1$

W specjalnym przypadku, kiedy cztery linie interpolujące $Q(0, t)$, $Q(1, t)$, $Q(s, 0)$ i $Q(s, 1)$ są liniami prostymi, w wyniku otrzymuje się *powierzchnię prostoliniową*. Jeżeli linie interpolujące są również koplanarne, to powierzchnia jest płaskim czworokątem.

Kontynuując wyprowadzenie załóżmy, że każda krzywa $P_{1_x}(t)$, $P_{4_x}(t)$, $R_{1_x}(t)$ i $R_{4_x}(t)$ reprezentuje postać Hermite'a:

$$P_{1_x}(t) = [g_{11}\ g_{12}\ g_{13}\ g_{14}]_x \cdot M_H \cdot T, \quad P_{4_x}(t) = [g_{21}\ g_{22}\ g_{23}\ g_{24}]_x \cdot M_H \cdot T$$
$$R_{1_x}(t) = [g_{31}\ g_{32}\ g_{33}\ g_{34}]_x \cdot M_H \cdot T, \quad R_{4_x}(t) = [g_{41}\ g_{42}\ g_{43}\ g_{44}]_x \cdot M_H \cdot T \tag{9.45}$$

Te cztery krzywe trzeciego stopnia mogą być zapisane w postaci jednego równania

$$[P_{1_x}(t)\ P_{4_x}(t)\ R_{1_x}(t)\ R_{4_x}(t)]^T = G_{H_x} \cdot M_H \cdot T \tag{9.46}$$

przy czym

$$G_{H_x} = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}_x \tag{9.47}$$

Transponowanie obu stron równania (9.46) daje

$$[P_{1_x}(t)\ P_{4_x}(t)\ R_{1_x}(t)\ R_{4_x}(t)] = T^T \cdot M_H^T \cdot \begin{bmatrix} g_{11} & g_{21} & g_{31} & g_{41} \\ g_{12} & g_{22} & g_{32} & g_{42} \\ g_{13} & g_{23} & g_{33} & g_{43} \\ g_{14} & g_{24} & g_{34} & g_{44} \end{bmatrix}_x =$$
$$= T^T \cdot M_H^T \cdot G_{H_x} \tag{9.48}$$

Podstawienie równania (9.48) do równania (9.44) daje

$$x(s, t) = T^T \cdot M_H^T \cdot G_{H_x} \cdot M_H \cdot S \tag{9.49}$$

podobnie

$$y(s, t) = T^T \cdot M_H^T \cdot G_{H_y} \cdot M_H \cdot S, \quad z(s, t) = T^T \cdot M_H^T \cdot G_{H_z} \cdot M_H \cdot S \quad (4.50)$$

Trzy macierze 4 × 4 G_{H_x}, G_{H_y} i G_{H_z} odgrywają taką samą rolę dla powierzchni Hermite'a jaką G_H dla krzywych.

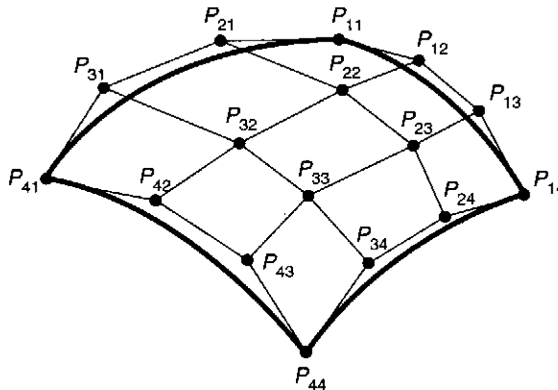
Powierzchnia bikubiczna Hermite'a umożliwia uzyskanie ciągłości C^1 i G^1 przy przejściu z jednego płatu do drugiego, podobnie jak krzywa trzeciego stopnia Hermite'a umożliwia uzyskanie ciągłości przy przejściu z jednego segmentu krzywej do drugiego. Szczegóły można znaleźć w rozdziale 11 pracy [FOLE90].

9.3.2. Powierzchnie Béziera

Bikubiczną postać powierzchni Béziera można wyprowadzić dokładnie w taki sam sposób jak postać Hermite'a. W wyniku otrzymujemy

$$\begin{aligned} x(s, t) &= T^T \cdot M_B^T \cdot G_{B_x} \cdot M_B \cdot S \\ y(s, t) &= T^T \cdot M_B^T \cdot G_{B_y} \cdot M_B \cdot S \\ z(s, t) &= T^T \cdot M_B^T \cdot G_{B_z} \cdot M_B \cdot S \end{aligned} \quad (9.51)$$

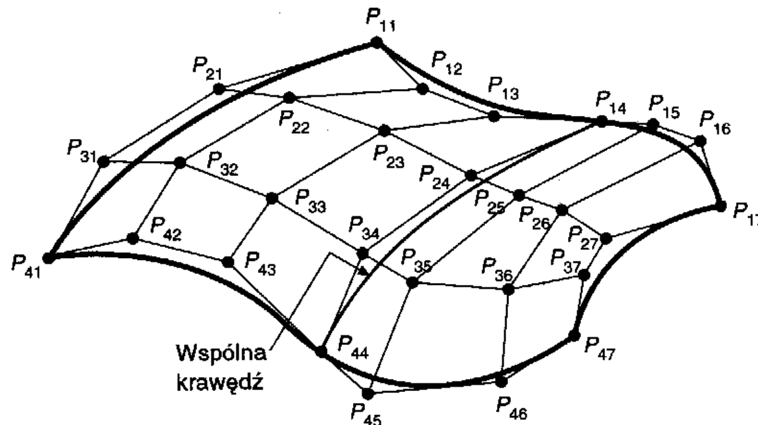
Macierz geometrii G Béziera składa się z 16 punktów kontrolnych (rys. 9.24). Powierzchnie Béziera są atrakcyjne przy projektowaniu interakcyjnym z tego samego powodu co w przypadku krzywych Béziera: niektóre z punktów kontrolnych interpolują powierzchnię, dając wy-



Rys. 9.24. Szesnaście punktów sterujących dla bikubicznego płata Béziera

godne precyzyjne sterowanie, podczas gdy wektory styczne mogą być również sterowane bezpośrednio. Gdy powierzchnie Béziera są wykorzystywane jako wewnętrzna reprezentacja, wówczas atrakcyjna jest ich właściwość związana z wielokątem rozpiętym.

Ciągłość C^0 i C^1 na granicach płatów uzyskujemy przez zrównanie czterech wspólnych punktów kontrolnych. Ciągłość G^1 powstaje wówczas, gdy dwa zbiory czterech punktów kontrolnych po każdej stronie krawędzi są współliniowe z punktami na krawędzi. Na rysunku 9.25 następujący zbiór punktów kontrolnych jest współliniowy i określa



Rys. 9.25. Dwa płaty Béziera połączone wzdłuż krawędzi P_{14}, P_{24}, P_{34} i P_{44}

cztery segmenty, których długości mają ten sam współczynnik k: (P_{13}, P_{14}, P_{15}), (P_{23}, P_{24}, P_{25}), (P_{33}, P_{34}, P_{35}) i (P_{43}, P_{44}, P_{45}). Czajniczek pokazany na rys. 9.1 był modelowany za pomocą 32 płatów Béziera, wszystkich połączonych w taki sposób, żeby zapewnić ciągłość G^1.

9.3.3. Powierzchnie B-sklejane

Płaty B-sklejane są reprezentowane w postaci:

$$x(s, t) = T^{\mathrm{T}} \cdot M_{\mathrm{Bs}}^{\mathrm{T}} \cdot G_{\mathrm{Bs}_x} \cdot M_{\mathrm{Bs}} \cdot S$$

$$y(s, t) = T^{\mathrm{T}} \cdot M_{\mathrm{Bs}}^{\mathrm{T}} \cdot G_{\mathrm{Bs}_y} \cdot M_{\mathrm{Bs}} \cdot S$$

$$z(s, t) = T^{\mathrm{T}} \cdot M_{\mathrm{Bs}}^{\mathrm{T}} \cdot G_{\mathrm{Bs}_z} \cdot M_{\mathrm{Bs}} \cdot S \qquad (9.52)$$

W przypadku krzywych B-sklejanych ciągłość C^2 na brzegach jest zapewniona automatycznie; nie jest konieczne specjalne uporządkowanie

punktów, z wyjątkiem tego, że należy unikać dublowania punktów kontrolnych, które tworzą nieciągłości.

Bikubiczne niejednorodne i ułamkowe powierzchnie B-sklejane i inne powierzchnie ułamkowe są podobne do swoich odpowiedników kubicznych. Wszystkie metody wyświetlania przenoszą się bezpośrednio na przypadek bikubiczny.

9.3.4. Normalne do powierzchni

Normalną do powierzchni bikubicznej można łatwo znaleźć. Jest ona potrzebna przy cieniowaniu (rozdz. 14), przy wykrywaniu interferencji w robotyce, przy obliczaniu przesunięć dla maszyn sterowanych numerycznie. Z równań (9.42) wektor styczny s do powierzchni $Q(s, t)$ jest równy

$$\frac{\partial}{\partial s}Q(s, t) = \frac{\partial}{\partial s}(T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot S) = T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot \frac{\partial}{\partial s}(S) = T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot [3s^2 \; 2s \; 1 \; 0]^{\mathrm{T}} \tag{9.53}$$

a wektor styczny t jest równy

$$\frac{\partial}{\partial t}Q(s, t) = \frac{\partial}{\partial t}(T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot S) = \frac{\partial}{\partial t}(T^{\mathrm{T}}) \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot S = [3t^2 \; 2t \; 1 \; 0]^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot S \tag{9.54}$$

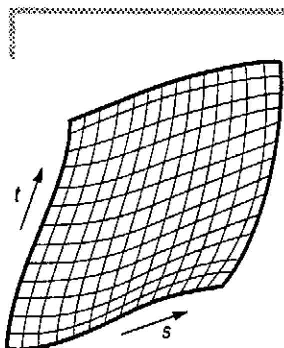
Oba wektory styczne są równoległe do powierzchni w punkcie (s, t) i ich iloczyn wektorowy jest wobec tego prostopadły do powierzchni. Zauważmy, że jeżeli oba wektory styczne są zerami, to iloczyn wektorowy jest zerem i nie ma normalnej do powierzchni. Przypomnijmy, że wektor styczny może zmaleć do zera w punktach połączenia, w których jest ciągłość C^1, a nie ma ciągłości G^1.

Każdy z wektorów stycznych jest oczywiście trójką, ponieważ równanie (9.42) reprezentuje składowe x, y, z powierzchni bikubicznej. Wprowadzając oznaczenie x_s dla składowej x wektora stycznego s, y_s dla składowej y i z_s dla składowej z, normalną można zapisać w następujący sposób:

$$\frac{\partial}{\partial s}Q(s, t) \times \frac{\partial}{\partial t}Q(s, t) = [y_s z_t - y_t z_s \;\; z_s x_t - z_t x_s \;\; x_s y_t - x_t y_s] \tag{9.55}$$

Normalna do powierzchni jest opisana wielomianem piątego stopnia o dwóch zmiennych i jej wyznaczenie wymaga wielu obliczeń. W pracy [SCHW82] podano bikubiczną aproksymację, która jest wystarczająca, jeżeli płat jest względnie gładki.

9.3.5. Wyświetlanie powierzchni bikubicznych



Rys. 9.26. Płat powierzchni wyświetlony jako zbiór krzywych o stałym *s* i stałym *t*

Tak jak krzywe, powierzchnie mogą być wyświetlane na zasadzie iteracyjnego obliczania bikubicznych wielokątów. Do wyświetlania bikubicznych płatów takich jak na rys. 9.26 najlepsze jest obliczanie iteracyjne. Każda z krzywych dla stałego s i stałego t na powierzchni sama jest krzywą sześcienną, więc wyświetlanie każdej z krzywych jest bezpośrednie, tak jak w programie 9.2.

Bezpośrednie iteracyjne obliczenia są dla powierzchni jeszcze kosztowniejsze niż dla krzywych, ponieważ równania powierzchni muszą być obliczane około $2/\delta^2$ razy. Dla $\delta = 0{,}1$ ta wartość wynosi 200; dla $\delta = 0{,}01$ wynosi 20 000. Przy tych liczbach alternatywna metoda obliczania metodą różnic jest jeszcze bardziej atrakcyjna niż dla krzywych. Ta metoda oraz inne użyteczne sposoby wyświetlania powierzchni bikubicznych są przedstawione w pracach [FOLE90; FORR79].

Program 9.2 Wyświetlanie bikubicznego płata jako siatki. Funkcje $X(s,t)$, $Y(s,t)$ i $Z(s,t)$ przy obliczaniu powierzchni wykorzystują współczynniki macierzy współczynników

```
typedef float Coeffs[4][4][3];

void DrawSurface( Coeffs *coefficients, int ns, int nt, int n )
   /* zmienne współczynniki są współczynnikami dla Q(s,t) */
   /* ns i nt są to liczby krzywych o stałym s i t, które trzeba narysować */
{
   float del, dels, delt, s, t;
   int i, j;
   /* Inicjalizacja */
   del = 1.0 / n; /* Krok wykorzystywany przy rysowaniu każdej krzywej */
   dels = 1.0 / (ns – 1); /* Krok dla zmiennej s przy przejściu do następnej krzywej
                           o stałym t*/
   delt = 1.0 / (nt – 1); /* Krok dla zmiennej t przy przejściu do następnej krzywej
                           o stałym s */

   /* Rysowanie ns krzywych o stałym s dla s=0.0, dels, 2dels, ...1.0 */
   for ( i = 0; i < ns; i++ ) {
      s = i * dels;
      /* Rysowanie krzywej o stałym s dla t zmieniającego się od 0.0 do 1.0 */
      /* X,Y i Z są funkcjami do obliczenia powierzchni bikubicznej dla danego s i t */
      MoveAbs3(X(s, 0.0), Y(s, 0.0), Z(s, 0.0));
      for ( j = 0; j < n; j++ ) {
         t = j * del;
         /* dla każdej krzywej używa się n kroków, gdy t zmienia się od 0.0 do 1.0 */
         LineAbs3(X(s, t), Y(s, t), Z(s, t));
      }
   }

   /* Rysowanie nt krzywych o stałym t dla t=0.0, delt, 2delt, ...1.0 */
```

```
    for ( i = 0; i < nt; i++ ) {
        t = i * delt;
        /* Rysowanie krzywej o stałym t dla s zmieniającego się od 0.0 do 1.0 */
        MoveAbs3(X(0.0, t), Y(0.0, t), Z(0.0, t));
        for ( j = 0; j < n; j++ ) {
            sπ = j * del;
            /* dla każdej krzywej używa się n kroków, gdy s zmienia się od 0.0 do 1.0 */
            LineAbs3(X(s, t), Y(s, t), Z(s, t));
        }
    }
}
```

Funkcje $X(s, t)$, $Y(s, t)$ i $Z(s, t)$ można łatwo znaleźć dla określonego typu krzywej. Dla przykładu rozważymy krzywą Béziera. Jeżeli w równaniu (9.51) określającym $x(s, t)$ wymnożymy macierze $T^{\mathrm{T}} \cdot M_{\mathrm{B}}^{\mathrm{T}}$ i $M_{\mathrm{B}} \cdot S$ to, otrzymamy

$$x(s, t) = [(1-t)^3 \; 3t(1-t)^2 \; 3t^2(1-t) \; t^3] \cdot \boldsymbol{G}_{\mathrm{B}_x} \cdot \begin{bmatrix} (1-s)^3 \\ 3s(1-s)^2 \\ 3s^2(1-s) \\ s^3 \end{bmatrix} \quad (9.56)$$

Przypomnijmy, że $\boldsymbol{G}_{\mathrm{B}_x}$ jest macierzą składowej x punktów kontrolnych, które pokazano na rys. 9.24, i może być zapisana jako

$$\boldsymbol{G}_{\mathrm{B}_x} = \begin{bmatrix} \boldsymbol{P}_{11} & \boldsymbol{P}_{21} & \boldsymbol{P}_{31} & \boldsymbol{P}_{41} \\ \boldsymbol{P}_{12} & \boldsymbol{P}_{22} & \boldsymbol{P}_{32} & \boldsymbol{P}_{42} \\ \boldsymbol{P}_{13} & \boldsymbol{P}_{23} & \boldsymbol{P}_{33} & \boldsymbol{P}_{43} \\ \boldsymbol{P}_{14} & \boldsymbol{P}_{24} & \boldsymbol{P}_{34} & \boldsymbol{P}_{44} \end{bmatrix}$$

Wreszcie, całkowicie rozwinięta postać równania (9.56) może zostać zapisana jako

$$\begin{aligned} x(s, t) &= \\ &= (1-s)^3 (\boldsymbol{P}_{11_x}(1-t)^3 + 3\boldsymbol{P}_{12_x}(1-t)^2 t + 3\boldsymbol{P}_{13_x}(1-t)t^2 + \boldsymbol{P}_{14_x}t^3) + \\ &+ 3(1-s)^2 s(\boldsymbol{P}_{21_x}(1-t)^3 + 3\boldsymbol{P}_{22_x}(1-t)^2 t + 3\boldsymbol{P}_{23_x}(1-t)t^2 + \boldsymbol{P}_{24_x}t^3) + \\ &+ 3(1-s)s^2(\boldsymbol{P}_{31_x}(1-t)^3 + 3\boldsymbol{P}_{32_x}(1-t)^2 t + 3\boldsymbol{P}_{33_x}(1-t)t^2 + \boldsymbol{P}_{34_x}t^3) + \\ &+ s^3(\boldsymbol{P}_{41_x}(1-t)^3 + 3\boldsymbol{P}_{42_x}(1-t)^2 t + 3\boldsymbol{P}_{43_x}(1-t)t^2 + \boldsymbol{P}_{44_x}t^3) \end{aligned}$$

Równania dla $y(s, t)$ i $z(s, t)$ można wyprowadzić w podobny sposób. Funkcje $X(s, t)$, $Y(s, t)$ i $Z(s, t)$, które są potrzebne dla programu 2.2, mogą być zakodowane bezpośrednio z wyrażeń $x(s, t)$, $y(s, t)$

i $z(s, t)$. Funkcje do rysowania innych typów powierzchni można otrzymać podobnie łatwo jak dla powierzchni Béziera.

9.4. Powierzchnie drugiego stopnia

Uwikłana postać równania

$$f(x, y, z) = ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + \\ + 2gx + 2hy + 2jz + k = 0 \tag{9.57}$$

definiuje rodzinę powierzchni drugiego stopnia. Na przykład, jeżeli $a = b = c = -k = 1$ i pozostałe współczynniki są równe zeru, to jest zdefiniowana kula jednostkowa o środku w początku układu współrzędnych. Jeżeli współczynniki od a do f są zerami, to jest zdefiniowana płaszczyzna. Powierzchnie drugiego stopnia są szczególnie użyteczne w takich zastosowaniach jak modelowanie molekularne [PORT79; MAX79] i są również wbudowane w systemach modelowania brył. Przypomnijmy również, że ułamkowe krzywe trzeciego stopnia mogą reprezentować przekroje stożka; podobnie ułamkowe powierzchnie bikubiczne mogą reprezentować powierzchnie drugiego stopnia. Dlatego uwikłane równanie drugiego stopnia jest alternatywą dla powierzchni ułamkowych, jeżeli mają być reprezentowane tylko powierzchnie drugiego stopnia. Do innych przyczyn dla korzystania z powierzchni drugiego stopnia można zaliczyć łatwość:

- obliczania normalnej do powierzchni,
- testowania, czy punkt jest na powierzchni (wystarczy podstawić punkt do równania (9.57), policzyć i sprawdzić, czy wynik jest w odległości nie większej niż ε od zera),
- obliczania z dla danych x i y (ważne w algorytmach usuwania powierzchni niewidocznych – por. rozdz. 13),
- obliczania przecięcia jednej powierzchni z drugą.

Alternatywny zapis równania (9.57) jest następujący:

$$P^{\mathrm{T}} \cdot Q \cdot P = 0 \tag{9.58}$$

z

$$Q = \begin{bmatrix} a & d & f & g \\ d & b & e & h \\ f & e & c & j \\ g & h & j & k \end{bmatrix} \quad \text{oraz} \quad P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{9.59}$$

Powierzchnia reprezentowana przez Q może być łatwo przesuwana i skalowana. Dla macierzy przekształceń M 4×4 o postaci podanej w rozdz. 5 przekształcona powierzchnia drugiego stopnia Q' jest dana zależnością

$$Q' = (M^{-1})^{\mathrm{T}} \cdot Q \cdot M^{-1} \tag{9.60}$$

Normalną do uwikłanej powierzchni zdefiniowanej przez $f(x, y, z) = 0$ jest wektor $[df/dx \ df/dy \ df/dz]$. Tę normalną do powierzchni jest znacznie łatwiej obliczyć niż normalną do powierzchni drugiego stopnia omówioną w p. 9.3.4.

9.5. Specjalizowane metody modelowania

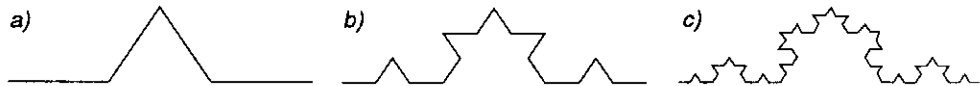
W tym rozdziale skoncentrujemy się na modelach geometrycznych; zajmiemy się nimi również w rozdz. 10. W świecie złożonym tylko z prostych obiektów geometrycznych te modele całkowicie wystarczyłyby. Jednak wiele zjawisk naturalnych nie da się efektywnie reprezentować za pomocą modeli geometrycznych, przynajmniej nie w dużej skali. Na przykład mgła składa się z maleńkich kropli wody, ale użycie modelu, w którym każda kropla musi być umieszczona osobno, nie wchodzi w rachubę. Co więcej, taki model kropli wody nie reprezentuje dokładnie naszej percepcji mgły; mgłę postrzegamy jako rozmycie w powietrzu naprzeciwko nas, a nie jako miliony kropli. Nasza wzrokowa percepcja mgły jest oparta na tym, jak mgła zmienia światło docierające do naszych oczu, a nie na kształcie i rozmieszczeniu poszczególnych kropli. Dlatego w celu efektywnego modelowania percepcyjnego efektu mgły jest nam potrzebny inny model. W ten sam sposób kształt liścia drzewa może być modelowany za pomocą wielokątów, a jego łodyga może być modelowana za pomocą rurki utworzonej przez powierzchnię sklejaną, natomiast dokładne umieszczenia każdego konaru, gałęzi, gałązki i liścia drzewa byłoby niesłychanie czasochłonne i kłopotliwe.

Wyczerpującą dyskusję na temat zaawansowanych metod modelowania można znaleźć w rozdz. 20 pracy [FOLE90]; tutaj omówimy dwie specjalizowane metody, które są zaskakująco łatwe do realizacji i które tworzą zaskakująco realistyczne obrazy.

9.5.1. Modele fraktalne

Fraktalom poświęcono ostatnio wiele uwagi [VOSS87; MAND82; PEIT86]. Otrzymywane za ich pomocą obrazy są efektowne; opracowano wiele różnych metod generowania fraktali. Określenie *fraktal* zostało

uogólnione przez środowisko grafiki komputerowej tak, że obejmuje obiekty wykraczające poza oryginalną definicję Mandelbrota. Oznacza obecnie wszystko, co ma faktyczną miarę dokładnego albo statystycznego samopodobieństwa; w tym znaczeniu korzystamy z tego pojęcia tutaj, chociaż dokładna matematyczna definicja wymaga statystycznego samopodobieństwa dla wszystkich rozdzielczości. Dlatego tylko fraktale generowane przez nieskończone procesy rekursywne są prawdziwymi obiektami fraktalnymi. Fraktale generowane przez skończone procesy mogą jednak nie wnosić widocznych zmian szczegółowych po wykonaniu kilku kroków i są odpowiednimi aproksymacjami ideału. To, co rozumiemy przez *samopodobieństwo*, jest najlepiej pokazane na przykładzie śnieżynki von Kocha. Zaczynając od odcinka z wybrzuszeniem po środku (rys. 9.27) zastępujemy każdy segment odcinka dokładnie taką samą figurą jak oryginalna. Ten proces powtarza się; każdy segment w części b) rysunku jest zastąpiony przez kształt dokładnie taki sam jak cała figura. (Nie ma znaczenia, czy zastępuje się przez kształt pokazany w części a), czy przez kształt pokazany na rysunku b); jeżeli wykorzysta się kształt z rysunku a), to wynik jest taki sam po 2^n krokach jak po n krokach, w których każdy segment krzywej bieżącej figury jest zastępowany przez cały bieżący kształt danego kroku.) Jeżeli ten proces powtarza się nieskończenie wiele razy, to mówimy, że wynik jest *samopodobny*: cały obiekt jest podobny (to znaczy może być przesuwany, obracany i skalowany) do swojej części.



Rys. 9.27. Konstrukcja śnieżynki von Kocha: każdy segment z rysunku a) jest zastępowany dokładną kopią całej figury, zmniejszoną 3 razy. Ten sam proces jest stosowany do segmentów z rysunku b) do generowania obrazu z rysunku c)

Z tą notacją samopodobieństwa jest związana notacja *wymiaru fraktalnego*. W celu zdefiniowania wymiaru fraktalnego przypomnijmy kilka właściwości obiektów o znanych wymiarach. Odcinek ma wymiar 1D; jeżeli podzielimy odcinek na N równych części, to każda z nich wygląda jak oryginalny odcinek zmniejszony ze współczynnikiem $N = N^{1/1}$. Kwadrat ma wymiar 2D; jeżeli podzielimy go na N części, to każda część wygląda jak oryginał zmniejszony ze współczynnikiem $\sqrt{N} = N^{1/2}$. (Na przykład kwadrat dzieli się dobrze na dziewięć podkwadratów; każdy wygląda jak oryginał zmniejszony ze współczynnikiem 1/3.) A co ze śnieżynką von Kocha? Jeżeli podzielimy ją na cztery części (każda część jest związana z oryginalnymi czterema segmentami z rys. 9.27a), to każda otrzymana część wygląda jak oryginał zmniejszo-

ny ze współczynnikiem 3. Chcielibyśmy powiedzieć, że ma wymiar d, przy czym $4^{1/d} = 3$. Dlatego d musi być równe lg(4)/lg(3) = 1,26.... Jest to definicja wymiaru fraktalnego.

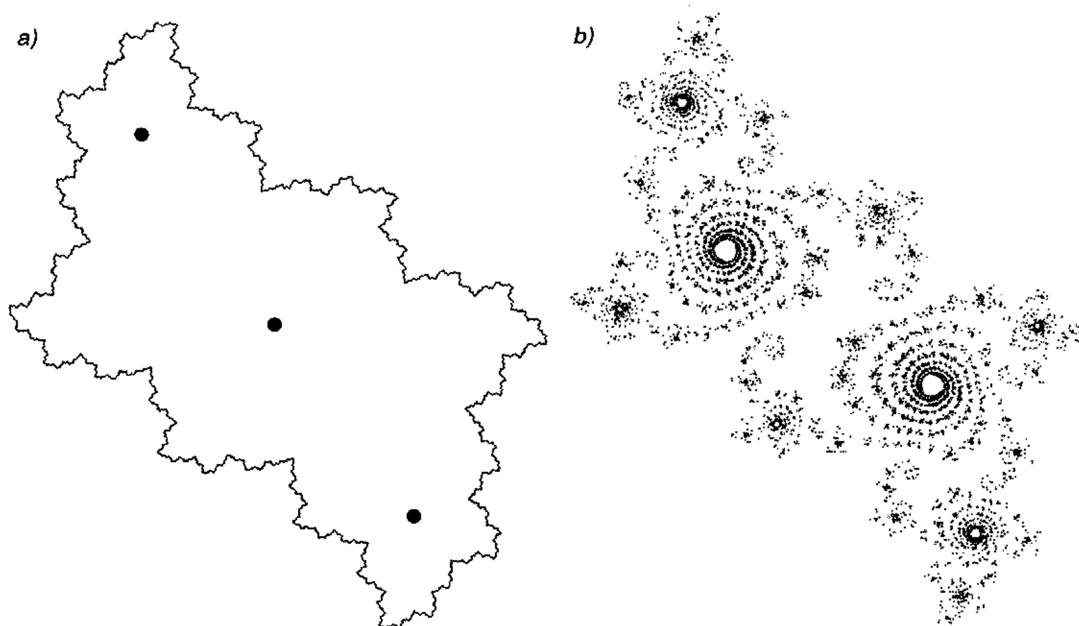
Najbardziej znanymi obiektami fraktalnymi są zbiór Julia-Fatou i zbiór Mandelbrota. Te obiekty są generowane za pomocą reguły $x \rightarrow x^2 + c$ (jest również wiele innych reguł – ta jest najprostsza i najbardziej znana). Tutaj x jest *liczbą zespoloną*[1], $x = a + bi$. Jeżeli moduł liczby zespolonej jest < 1, to przy podnoszeniu do kwadratu wartość modułu dąży do zera. Jeżeli moduł jest > 1, to przy kolejnym podnoszeniu do kwadratu moduł wzrasta coraz bardziej. W przypadku liczb z modułem 1 po podniesieniu do kwadratu moduł wciąż jest równy 1. Dlatego niektóre liczby zespolone przy kolejnym podnoszeniu do kwadratu dążą do zera, niektóre do nieskończoności, a inne ani nie dążą do zera, ani nie dążą do nieskończoności – ta ostatnia grupa tworzy granicę między liczbami przyciąganymi do zera a liczbami przyciąganymi do nieskończoności.

Załóżmy, że wielokrotnie zastosujemy odwzorowanie $x \rightarrow x^2 + c$ do każdej liczby zespolonej x o pewnej niezerowej wartości c, takiej jak na przykład $c = -0{,}12375 + 0{,}056805i$; niektóre liczby zespolone będą przyciągane do nieskończoności, niektóre do liczb skończonych, a niektóre nie będą zmierzały w żadnym z tych kierunków. Rysując zbiór punktów, które nie zmierzają w żadnym z tych kierunków, otrzymamy zbiór Julia-Fatou pokazany na rys. 9.28a.

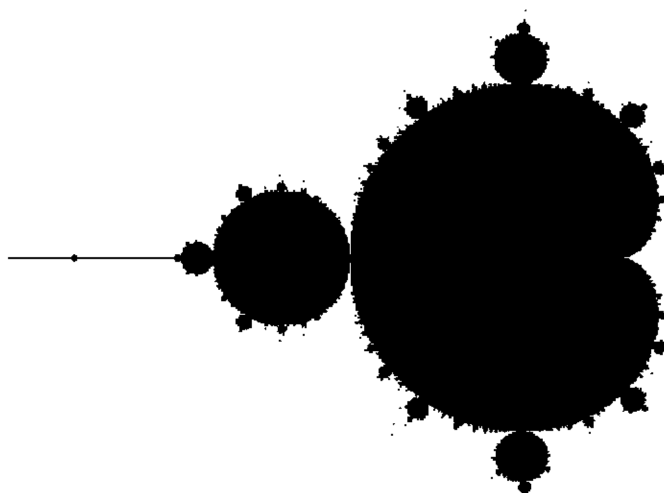
Zauważmy, że obszar na rys. 9.28b nie jest tak dobrze połączony jak ten z rys. 9.28a. W części b) niektóre punkty zmierzają w kierunku trzech pokazanych czarnych punktów, niektóre do nieskończoności, a niektóre nie zmierzają w żadnym z tych kierunków. Te ostatnie punkty to są punkty, które są narysowane jako obrys kształtu w części b). Kształt zbioru Julia-Fatou ewidentnie zależy od wartości liczby c. Jeżeli obliczymy zbiór Julia dla wszystkich możliwych wartości c i zaznaczymy punkt c na czarno, gdy zbiór Julia-Fatou jest spójny (to znaczy złożony z jednego kawałka, nie podzielonego na rozłączne „wyspy"), albo na biało, gdy zbiór nie jest spójny, to otrzymamy obiekt z rys. 9.29 znany jako *zbiór Mandelbrota*. Zauważmy, że zbiór Mandelbrota jest samopodobny w tym sensie, że wokół brzegu dużego dysku w zbiorze jest kilka mniejszych zbiorów, z których każdy wygląda bardzo podobnie jak ten duży po zmniejszeniu.

---

[1] Jeżeli liczby zespolone nie są znane, to wystarczy symbol i traktować jako specjalny symbol i poznać definicje dodawania i mnożenia liczb zespolonych. Jeżeli $z = c + di$ jest drugą liczbą zespoloną, to $x + z$ jest określone jako $(a + c) + (b + d)i$, a xz – jako $(ac - bd) + (ad + bc)i$. Liczby zespolone możemy reprezentować jako punkty na płaszczyźnie identyfikując punkt (a, b) z liczbą zespoloną $(a + bi)$. Moduł liczby $a + bi$ jest liczbą rzeczywistą $(a^2 + b^2)^{1/2}$, co daje „wielkość" liczby zespolonej.



Rys. 9.28. Zbiór Julia-Fatou: a) $c = -0{,}12375 + 0{,}056805i$; b) $c = -0{,}012 + 0{,}74i$

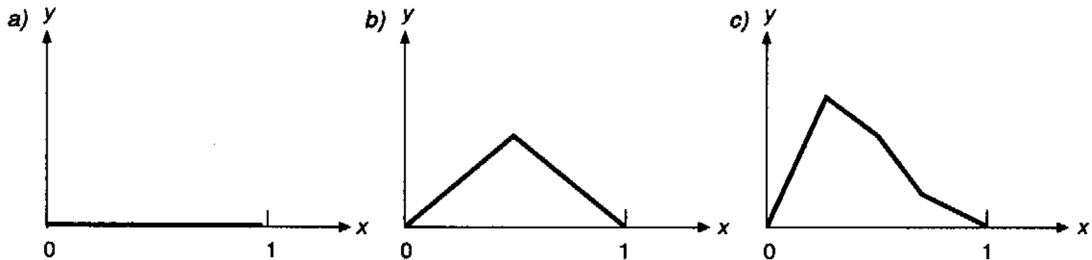


Rys. 9.29. Zbiór Mandelbrota. Każdy punkt c na płaszczyźnie zespolonej otrzymuje barwę czarną, jeżeli zbiór Julia dla procesu $x \to x^2 + c$ jest spójny

Na szczęście jest łatwiejszy sposób generowania przybliżeń zbioru Mandelbrota: dla każdej wartości c weźmy liczbę zespoloną $0 = 0 + 0i$ i zastosujmy proces $x \to x^2 + c$ pewną skończoną liczbę razy (na przykład 1000). Jeżeli po wielu tych iteracjach punkt jest na zewnątrz dysku

zdefiniowanego przez moduł < 100, to c otrzymuje barwę białą; w przeciwnym przypadku przypisujemy barwę czarną. Gdy liczba iteracji i promień dysku rosną, wówczas otrzymany obraz stanowi lepszą aproksymację zbioru. Peitgen i Richter [PEIT86] podali bezpośrednie wskazówki co do generowania wielu efektownych obrazów zbiorów Mandelbrota i Julia-Fatou.

Te wyniki są wyjątkowo sugestywne dla modelowania form naturalnych, ponieważ wiele obiektów naturalnych wydaje się posiadać cechę samopodobieństwa. Góry mają szczyty, mniejsze szczyty, skały, skałki, które wyglądają podobnie; drzewa mają konary, gałęzie i gałązki, które również wyglądają podobnie; wybrzeże morskie ma zatoki, zatoczki, ujścia rzek, strumyczków i kanałów drenujących, które też wyglądają podobnie. Dlatego modelowanie samopodobieństwa w pewnej skali jest sposobem na generowanie ciekawie wyglądających modeli zjawisk naturalnych. Skala, przy której samopodobieństwo załamuje się, nie jest tu szczególnie istotna, ponieważ celem jest raczej modelowanie niż matematyka. Dlatego, jeżeli obiekt został wygenerowany rekursywnie w wystarczającej liczbie kroków, tak że dalsze zmiany są już na poziomie podpikselowym, to nie ma potrzeby kontynuowanie procesu.

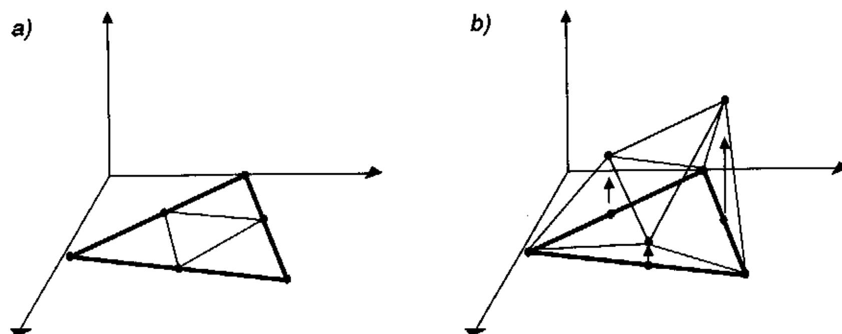


Rys. 9.30. Odcinek na osi x (a); punkt środkowy odcinka został przesunięty w kierunku y o wielkość losową (b); wynik kolejnej iteracji (c)

Fournier, Fussell i Carpenter [FOUR82] opracowali mechanizm generowania klasy gór fraktalnych z wykorzystaniem rekursywnego podziału. Jest to łatwiej wytłumaczyć w 1D. Załóżmy, że startujemy z odcinkiem leżącym na osi x (rys. 9.30a). Jeżeli teraz podzielimy odcinek na połowy i przesuniemy punkt środkowy o pewną odległość w kierunku y, to otrzymamy kształt pokazany na rys. 9.30b. Kontynuując podział każdego segmentu, obliczamy nową wartość dla punktu środkowego segmentu z (x_i, y_i) do (x_{i+1}, y_{i+1}) w następujący sposób: $x_{new} = 1/2(x_i + x_{i+1})$, $y_{new} = 1/2(y_i + y_{i+1}) + P(x_{i+1} - x_i)R(x_{new})$, przy czym $P()$ jest funkcją określającą zakres zakłócenia w funkcji wielkości

zakłócanego odcinka, a $R()$ jest liczbą losową[2] między 0 i 1 wybraną na bazie x_{new} (rys. 9.30c). Jeżeli $P(s) = s$, to pierwszy punkt nie może być przesunięty o więcej niż 1, każdy z następnych dwóch punktów (które są już najwyżej na wysokości 1/2) nie może być przesunięty o więcej niż 1/2 itd. Dlatego wszystkie otrzymane punkty trafiają do kwadratu jednostkowego. Dla $P(s) = s^a$ kształt wyniku zależy od wartości a; mniejsze wartości a dają większe zakłócenia i odwrotnie. Oczywiście inne funkcje, np. $P(s) = 2^{-s}$, mogą być również użyte.

Fournier, Fussell i Carpenter wykorzystali ten proces do modyfikowania kształtów 2D w następujący sposób. Zaczynali od trójkąta, zaznaczali punkt środkowy na każdym boku i łączyli trzy punkty środkowe (rys. 9.31a). Współrzędna y każdego punktu środkowego była z kolei modyfikowana w wyżej opisany sposób i wynikowy zbiór czterech trójkątów wyglądał jak na rys. 9.31b. Ten proces powtarzany iteracyjnie wytwarzał góry wyglądające realistycznie, co widać na przykład na fot. 11 (chociaż przy powiększeniu można zauważyć bardzo regularną strukturę).



Rys. 9.31. Podział trójkąta na cztery mniejsze trójkąty (a); punkty środkowe oryginalnego trójkąta są zakłócone w kierunku y tak, że powstał kształt z rysunku (b)

Zauważmy, że możemy zacząć z kombinacją trójkątów o pewnym kształcie i zastosować omawiany proces w celu wygenerowania szczegółów. Ta możliwość jest szczególnie istotna w niektórych zastosowaniach modelowania, w których rozmieszczenie obiektów w scenie może być stochastyczne na niskim poziomie, natomiast uporządkowane na wysokim poziomie: roślinność w dekoracyjnym ogrodzie może być generowana za pomocą mechanizmu stochastycznego, ale jej rozmieszczenie

[2] $R()$ jest w istocie *zmienną losową*, funkcją, która dla liczb rzeczywistych generuje liczby losowe z przedziału od 0 do 1. Jeżeli zostanie to zrealizowane za pomocą generatora liczb pseudolosowych, to zaletą takiego rozwiązania jest powtarzalność fraktali: możemy je ponownie wygenerować, jeżeli stworzymy takie same warunki początkowe.

w ogrodzie musi podlegać pewnym ścisłym regułom. Z drugiej strony to, że struktury wysokiego poziomu początkowego uporządkowania trójkątów są zachowane przy iteracyjnym dzieleniu, może nie odpowiadać pewnym zastosowaniom (w szczególności, tak generowany fraktal nie ma wszystkich statystycznych samopodobieństw obecnych we fraktalach bazujących na ruchach Browna [MAND82]). Ponadto, ponieważ położenie każdego wierzchołka jest regulowane tylko raz i potem jest stacjonarne, powstaje tendencja do rozbudowywania powierzchni wzdłuż boków między pierwotnymi trójkątami, a to może wyglądać nienaturalnie.

Rendering fraktali może być trudny. Jeżeli będzie wykorzystywany *z*-bufor, to wyświetlenie całego obiektu zajmie dużo czasu, ponieważ jest uwikłana ogromna liczba wielokątów. Przy renderingu typu przeglądanie wierszy kosztowne jest sortowanie wszystkich wielokątów i bierze się pod uwagę tylko te, które przecinają przeglądany wiersz. Śledzenie promieni w przypadku fraktali jest wyjątkowo trudne, ponieważ każdy promień musi być sprawdzony ze względu na przecięcie z każdym z możliwych wielokątów. Kajiya [KAJI83] podał sposób stosowania metody śledzenia promieni do obiektów fraktalnych klasy opisanej w pracy [FOUR82], a Bouville [BOUV85] ulepszył ten algorytm wprowadzając lepsze ograniczenie objętości zajmowanej przez obiekty.

9.5.2. Modele wykorzystujące gramatyki

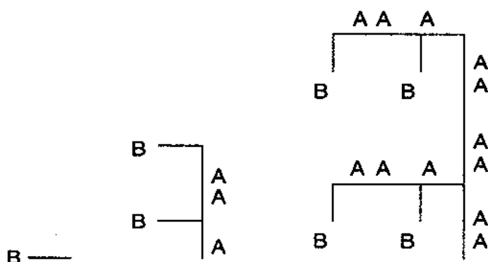
Smith [SMIT84] przedstawił metodę opisu struktury określonych roślin, opracowaną pierwotnie przez Lindenmayera [LIND68], wykorzystującą języki równoległych gramatyk grafowych (*L-gramatyki*), które Smith nazwał *graftalami*. Te języki są opisane przez gramatyki składające się ze zbioru produkcji, z których wszystkie są stosowane od razu. Lindenmayer rozszerzył te języki o nawiasy, tak że alfabet zawiera dwa specjalne symbole „[" i „]". Typowym przykładem jest gramatyka z alfabetem {A,B,[,]} i dwiema regułami produkcji:

1. A → AA
2. B → A[B]AA[B]

Zaczynając od aksjomatu A pierwsze kilka generacji to A,AA,AAAA itd; startując od aksjomatu B pierwsze kilka generacji to:

0. B
1. A[B]AA[B]
2. AA[A[B]AA[B]]AAAA[A[B]AA[B]]

itd. Jeżeli powiemy, że słowo w języku reprezentuje sekwencję segmentów w strukturze grafu i że części w nawiasach reprezentują części,



Rys. 9.32. Drzewo reprezentujące pierwsze trzy słowa języka. Wszystkie gałęzie są narysowane na lewo od bieżącej głównej osi

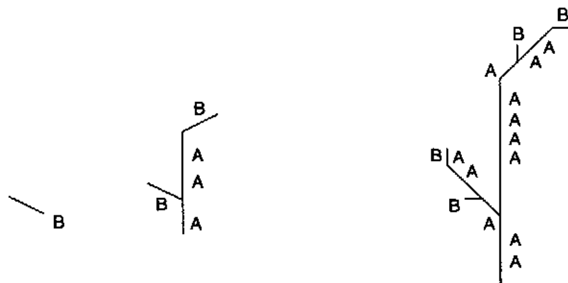
które odgałęziają się od poprzedzającego je symbolu, to figury związane z tymi trzema poziomami wyglądają tak jak na rys. 9.32.

Ten zbiór obrazów ma przyjemną strukturę z rozgałęzieniami, ale lepiej byłoby, gdyby drzewo było bardziej zrównoważone. Jeżeli dodamy do języka symbole nawiasów „(" i „)" i zmienimy drugą produkcję na A[B]AA(b), to druga generacja przyjmie postać

2. AA[A[B]AA(B)]AAAA(A[B]AA(B))

Jeżeli powiemy, że nawiasy kwadratowe oznaczają lewą gałąź, a nawiasy okrągłe prawą gałąź, to otrzymamy obrazy jak na rys. 9.33. Kontynuując generowanie w tym języku otrzymamy struktury grafowe reprezentujące bardzo złożone wzory. Te struktury grafowe mają rodzaj samopodobieństwa w tym sensie, że wzór opisany przez wyraz n-tej generacji jest zawarty (wielokrotnie w tym przypadku) w $(n + 1)$-ym słowie generacji.

Generowanie obiektu z takiego słowa jest procesem oddzielnym od generowania samego słowa. Na rysunku kolejne segmenty drzewa mają coraz mniejsze długości, a wszystkie kąty przy rozgałęzianiu mają po 45° oraz gałęzie odchodzą w lewo albo w prawo. Wybierając zmieniające się kąty rozgałęzienia dla gałęzi na różnych głębokościach i zmieniając



Rys. 9.33. Trzy reprezentacje pierwszych trzech słów, ale w języku z dwustronnym rozgałęzianiem. Przechodząc do kolejnych generacji skracamy każdy segment drzewa

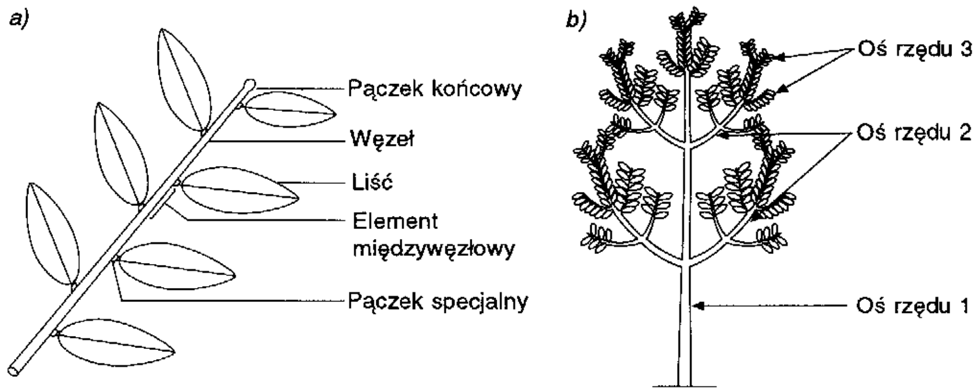
grubości linii (a nawet walców) reprezentujących segmenty, uzyskuje się różne wyniki; rysując „kwiatek" albo „liść" w każdym węźle końcowym drzewa ulepszymy obraz. Sama gramatyka nie ma wrodzonej treści geometrycznej, więc korzystanie z modeli bazujących na gramatykach wymaga zarówno gramatycznej, jak i geometrycznej reprezentacji języka.

Taki sposób wzbogacania języka i interpretacji słów w języku (to jest obrazów generowanych ze słów) był stosowany przez kilku badaczy [REFF88; PRUS88]. Gramatyki zostały tak wzbogacone, że umożliwiały śledzenie „wieku" litery w słowie tak, że stare i młode litery są różne (ta rejestracja wieku może być wykonana za pomocą reguł o postaci $A \to B$, $B \to C$, $C \to D$, ..., $Q \to QG[Q]$ i że interesujące przekształcenie nie pojawia się dopóty, dopóki roślina nie osiągnie odpowiedniego „wieku"). Wiele pracy włożono w opracowanie gramatyk, które reprezentują biologię roślin w czasie rozwoju.

W pewnym momencie jednak gramatyka staje się nieporęczna jako narzędzie do opisu roślin: dodanych zostaje zbyt wiele cech do gramatyki albo do interpretacji jej słów. W modelu Reffye'a [REFF88] symulacja wzrostu roślin jest kontrolowana za pomocą niewielkiego zbioru parametrów, które są opisane w kategoriach biologicznych i które mogą być dodane do algorytmu. Produkcje gramatyki są stosowane raczej losowo niż deterministycznie.

W tym modelu rozpoczynamy jak poprzednio od jednej łodygi. Na końcu tej łodygi jest *pączek*, który może podlegać jednemu z kilku przekształceń: może umrzeć, może zakwitnąć i umrzeć, może być uśpiony przez pewien czas albo może się stać elementem międzywęzłowym – segmentem rośliny między pąkami. Proces stawania się elementem międzywęzłowym ma trzy etapy: oryginalny pączek może generować jeden lub kilka *specjalnych pączków* (pączki z jednej strony połączenia między węzłami wewnętrznymi) – taki proces jest określany jako *rozgałęzianie*; dodawany jest element międzywęzłowy; i koniec nowego odcinka międzywęzłowego staje się *pączkiem wierzchołkowym* (pączek na końcu sekwencji elementów międzywęzłowych). Na rysunku 9.34a pokazano przykład przejścia od pączka do elementu międzywęzłowego.

Każdy z pączków w otrzymanym obiekcie może dalej podlegać podobnym przekształceniom. Jeżeli powiemy, że początkowy segment drzewa jest rzędu 1, to możemy zdefiniować porządek wszystkich innych elementów międzywęzłowych indukcyjnie: odcinki międzywęzłowe generowane z węzłowego pączka elementu międzywęzłowego rzędu i są również rzędu i; te generowane ze specjalnych pączków elementu międzywęzłowego rzędu i są rzędu $(i + 1)$. Dlatego cały pień drzewa jest rzędu 1, konary są rzędu 2, gałęzie tych konarów są rzędu 3 itd. Na rysunku 9.34b pokazano bardziej złożoną roślinę i rzędy różnych odcinków międzywęzłowych w roślinie.



Rys. 9.34. Przykłady wzrostu roślin: a) pączek na końcu segmentu może się stać elementem międzywęzłowym; wtedy tworzy się nowy pączek (specjalny pączek), nowy segment (element międzywęzłowy) i nowy pączek na końcu (pączek węzłowy); b) bardziej złożona roślina; rzędy są związane z różnymi elementami międzywęzłowymi

Konwersja opisu na aktualny obraz drzewa wymaga modelu dla kształtów jego różnych elementów: element międzywęzłowy rzędu 1 może być dużym stożkiem, a element międzywęzłowy rzędu 7 może być na przykład małym zielonym odcinkiem. Jedynym wymaganiem jest to, że przy każdym specjalnym pączku musi być liść (chociaż liść może kiedyś odpaść).

Wreszcie w celu symulowania wzrostu rośliny w tym modelu musimy znać następującą informację biologiczną: bieżący wiek modelu, szybkość wzrostu elementów międzwęzłowych poszczególnych rzędów i prawdopodobieństwa zamierania, przerwy, rozgałęziania i reiteracji jako funkcji wieku, rozmiaru i rzędu. Jest również potrzebna pewna informacja geometryczna: kształt każdego odcinka międzywęzłowego (jako funkcja rzędu i wieku), kąty rozgałęziania dla każdego rzędu i wieku i orientacja każdej osi (to, czy każda sekwencja elementów międzywęzłowych rzędu tworzy linię prostą, czy zakrzywia się w kierunku poziomym albo pionowym). W celu narysowania obrazu rośliny potrzebujemy jeszcze dalszych informacji: barwa i tekstura każdego z rysowanych elementów – elementy międzywęzłowe różnych rzędów, liście w różnym wieku i kwiaty w różnym wieku. Za pomocą modeli wykorzystujących gramatyki można narysować bardzo przekonujące modele drzewa; por. fot. 12.

Podsumowanie

W tym rozdziale zaledwie zostały poruszone ważne idee związane z reprezentacją krzywych i powierzchni, niemniej podano wystarczająco dużo informacji, żeby było możliwe zrealizowanie systemu interakcyj-

nego z wykorzystaniem tych reprezentacji. Teoretyczne ujęcie materiału można znaleźć w książkach [BART87; DEBO78, FAUX79; MORT85].

Siatki wielokątowe, które są kawałkami liniowe, dobrze się nadają do reprezentowania obiektów o płaskich ścianach, rzadko natomiast są wystarczające dla obiektów o powierzchniach krzywoliniowych. Kawałkami ciągłe parametryczne krzywe trzeciego stopnia są powszechnie używane w grafice komputerowej i w CAD do reprezentowania obiektów o powierzchniach krzywoliniowych, ponieważ:

- umożliwiają występowanie kilku wartości dla jednej wartości x i y,
- reprezentują nieskończone nachylenia,
- zapewniają lokalne sterowanie, co polega na tym, że zmiana punktów kontrolnych wpływa tylko na lokalny kawałek krzywej,
- umożliwiają albo interpolowanie, albo aproksymowanie punktów kontrolnych, zależnie od wymagań zastosowania,
- są efektywne obliczeniowo,
- można je łatwo przekształcać na zasadzie przekształcania punktów kontrolnych.

Chociaż omówiliśmy tylko powierzchnie trzeciego stopnia, można również używać powierzchni wyższego albo niższego stopnia. We wcześniej wymienionych podręcznikach podano wyprowadzenia krzywych i powierzchni parametrycznych dla ogólnego przypadku stopnia n.

Omówiliśmy również pokrótce niektóre metody modelowania zjawisk naturalnych, w szczególności podejścia fraktalne i wykorzystujące gramatyki.

Zadania

9.1. Znajdź macierz geometrii i macierz bazową dla parametrycznej reprezentacji odcinka danego równaniem (9.11).

9.2. Pokaż, że dla krzywej 2D $[x(t)\ y(t)]^T$ ciągłość G^1 oznacza, że nachylenie geometryczne dy/dx jest równe w punktach połączenia między segmentami.

9.3. Niech $\gamma(t) = (t, t^2)$ dla $0 \leqslant t \leqslant 1$ i niech $\eta(t) = (2t + 1, t^3 + 4t + 1)$ dla $0 \leqslant t \leqslant 1$. Zauważmy, że $\gamma(1) = (1, 1) = \eta(0)$, a więc γ i η łączą się z ciągłością C^0.

a. Narysuj $\eta(t)$ i $\gamma(t)$ dla $0 \leqslant t \leqslant 1$.

b. Określ, czy $\eta(t)$ i $\gamma(t)$ spotykają się z ciągłością C^1 w punkcie połączenia. (Trzeba będzie obliczyć wektory $d\gamma/dt(1)$ i $d\eta/dt(0)$ w celu sprawdzenia odpowiedzi.)

c. Określ, czy $\eta(t)$ i $\gamma(t)$ spotykają się w punkcie połączenia z ciągłością G^1. (Trzeba będzie sprawdzić stosunki z części (b) w celu sprawdzenia odpowiedzi.)

9.4. Rozważ ścieżki

$$\gamma(t) = (t^2 - 2t + 1,\ t^3 - 2t^2 + t) \text{ oraz } \eta(t) = (t^2 + 1,\ t^3)$$

obie zdefiniowane w przedziale $0 \leqslant t \leqslant 1$. Krzywe łączą się, ponieważ $\gamma(1) = (1, 0) = \eta(0)$. Pokaż, że w punkcie połączenia jest ciągłość C^1, ale nie ma ciągłości G^1. Narysuj obie krzywe w funkcji t w celu dokładnego wykazania, dlaczego tak jest.

9.5. Pokaż, że obie krzywe $\gamma(t) = (t^2 - 2t,\ t)$ i $\eta(t) = (t^2 + 1,\ t + 1)$ mają w punkcie połączenia $\gamma(1) = \eta(0)$ ciągłości C^1 i G^1.

9.6. Przeanalizuj wpływ na krzywą B-sklejaną sekwencji czterech współliniowych punktów kontrolnych.

9.7. Napisz program akceptujący dowolną macierz geometrii, macierz bazową, listę punktów kontrolnych i rysujący odpowiednią krzywą.

9.8. Znajdź warunki, przy jakich dwie połączone krzywe Hermite'a mają ciągłość C^1.

9.9. Załóż, że równania wiążące geometrię Hermite'a z geometrią Béziera mają postać $R_1 = \beta(P_2 - P_1)$, $R_4 = \beta(P_4 - P_3)$. Weź pod uwagę cztery równo oddalone punkty kontrolne Béziera $P_1 = (0,\ 0)$, $P_2 = (1,\ 0)$, $P_3 = (2, 0)$, $P_4 = (3, 0)$. Pokaż, że na to, żeby dla krzywej parametrycznej $Q(t)$ była stała szybkość od P_1 do P_4, współczynnik β musi być równy 3.

9.10. Wyjaśnij, dlaczego równanie (9.35) dla jednorodnej krzywej B-sklejanej jest zapisane jako $Q_i(t - t_i)$, a równanie (9.37) dla niejednorodnej krzywej B-sklejanej ma postać $Q_i(t)$.

9.11. Dla danej niejednorodnej krzywej B-sklejanej 2D i wartości $(x,\ y)$ na krzywej napisz program znajdowania odpowiedniej wartości t. Pamiętaj o uwzględnieniu możliwości, że dla danej wartości x (albo y) może być wiele wartości y (albo x).

9.12. Zastosuj metodologię użytą do wyprowadzenia równań (9.49) i (9.50) dla powierzchni Hermite'a do wyprowadzenia równania (9.51) dla powierzchni Béziera.

9.13. Niech $t_0 = 0,\ t_1 = 1,\ t_2 = 3,\ t_3 = 4,\ t_4 = 5$. Korzystając z tych wartości oblicz $B_{0,4}$ i każdą z funkcji użytych w tej definicji. Następnie narysuj te funkcje w przedziale $-3 \leqslant t \leqslant 8$.

9.14. Opracuj program, podobnie jak w przykładzie 9.2, do wyświetlania płatów powierzchni Béziera, korzystając ze schematu programu 9.2. Program powinien oferować opcję wyświetlania punktów kontrolnych dla określonego płatu tak, żeby użytkownik mógł wybrać dowolny z nich (korzystając z lokalizatora) i przesunąć go na nowe miejsce. Płat powinien być wtedy przerysowany w celu odzwierciedlenia nowych ograniczeń geometrycznych. Wejście lokalizatora jest 2D; jak można wykorzystać go do manipulowania punktami kontrolnymi 3D? Można określić własną geometrię płata albo korzystać z istniejących danych, takich jak dla czajniczka z rys. 9.1. Pełne dane dla czajniczka można znaleźć w pracy [CROW87].

10. Modelowanie brył

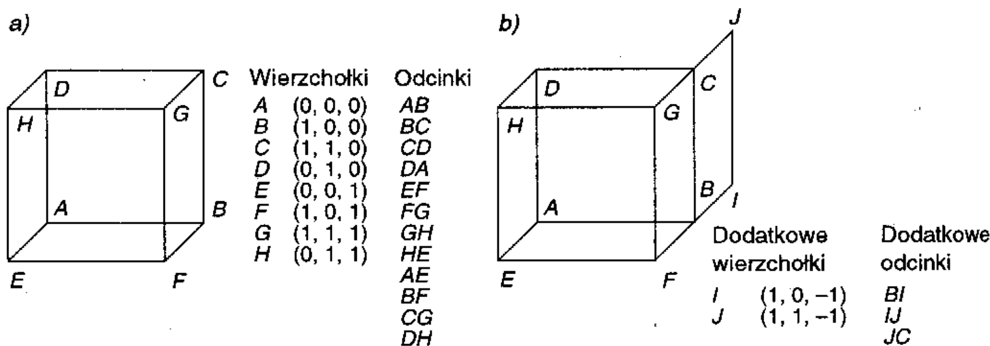
Reprezentacje omówione w rozdz. 9 umożliwiają opisywanie krzywych i powierzchni w 2D i 3D. Tak jak zbiór odcinków i krzywych 2D nie musi opisywać brzegu zamkniętego obszaru, tak zbiór płaszczyzn i powierzchni 3D niekoniecznie musi ograniczać zamkniętą objętość. W wielu jednak zastosowaniach ważne jest rozróżnienie między częścią wewnętrzną, zewnętrzną i powierzchnią obiektu 3D i ważna jest możliwość wyznaczania właściwości obiektu, które zależą od tego rozróżnienia. Na przykład w programach CAD/CAM, jeżeli bryłę można modelować w sposób, który odpowiednio odzwierciedla jej geometrię, to można wykonać wiele różnych operacji, zanim obiekt zostanie wytworzony. Możemy na przykład chcieć określić, czy obiekty stykają się ze sobą, czy ramię robota będzie w swoim środowisku uderzało w obiekty albo czy narzędzie tnące będzie zdejmowało tylko ten materiał, który należy odrzucić. Przy symulacji mechanizmów fizycznych, takich jak przekładnia zębata, może być ważne określenie takich parametrów jak objętość czy środek masy. Analiza elementów skończonych jest stosowana do modeli brył w celu obliczenia odpowiedzi na takie czynniki, jak ciśnienie i temperatura. Odpowiednia reprezentacja brył może nawet umożliwić automatyczne generowanie instrukcji dla maszyn sterowanych numerycznie w celu utworzenia obiektu albo szybkie wykonywanie prototypów za pomocą takich metod jak stereolitografia – proces, który wykorzystuje promień lasera do formowania utwardzonych obiektów. Niektóre metody graficzne, np. modelowanie refrakcyjnej przezroczystości, zależą od możliwości określenia, gdzie strumień światła wchodzi i wychodzi z bryły. Są to przykłady *modelowania brył*. Potrzeba modelowania obiektów jako brył doprowadziła do opracowania

różnych specjalizowanych metod ich reprezentowania. Ten rozdział zawiera krótkie wprowadzenie do tych reprezentacji.

10.1. Reprezentowanie brył

Zdolność reprezentacji do kodowania obiektów wyglądających jak bryły nie oznacza automatycznie, że ta reprezentacja jest odpowiednia do reprezentowania brył. Dotychczas reprezentowaliśmy obiekty jako zbiory odcinków, krzywych, wielokątów i powierzchni. Czy odcinki z rys. 10.1 definiują sześcian? Jeżeli przyjmiemy, że każdy zbiór czterech odcinków z każdej strony obiektu ogranicza kwadratową ścianę, to na rysunku jest sześcian. W tej reprezentacji nie ma jednak niczego, co by sugerowało taką interpretację. Na przykład, ten sam zbiór odcinków byłby użyty do narysowania figury, gdyby brakowało jednej albo wszystkich ścian. Co by było, gdybyśmy zdecydowali się, że każda płaska pętla z połączonych odcinków na rysunku z definicji określa wielokątową ścianę. Wtedy rysunek 10.1 składałby się ze wszystkich ścian z rys. 10.1a plus dodatkowa „wisząca" ściana, tworząc obiekt, który nie ogranicza objętości. Jak zobaczymy w p. 10.5, są potrzebne pewne dodatkowe ograniczenia, jeżeli chcemy zapewnić to, żeby tego rodzaju reprezentacja modelowała tylko bryły.

Requicha [REQU80] podaje listę właściwości potrzebnych do reprezentowania brył. *Domena* reprezentacji powinna być na tyle duża, żeby umożliwić reprezentowanie użytecznego zbioru obiektów fizycznych. Idealnie reprezentacja powinna być niedwuznaczna: nie powinno być wątpliwości co do tego, co jest reprezentowane, i dana reprezentacja powinna odpowiadać tylko jednej bryle, a nie tak jak w przypadku obiektu z rys. 10.1a. Niedwuznaczna reprezentacja powinna być rów-



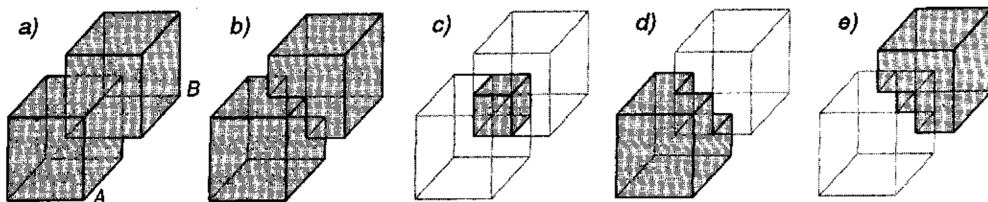
Rys. 10.1. Model drutowy sześcianu złożony z 12 odcinków (a); model szkieletowy sześcianu z dodatkową ścianą (b)

nież *kompletna*. Reprezentacja jest unikatowa, jeżeli może być użyta do kodowania dowolnej bryły tylko w jeden sposób. Jeżeli reprezentacja może zapewnić unikatowość, to łatwe jest wykonywanie takich operacji jak sprawdzanie równości dwóch obiektów. Dokładna reprezentacja umożliwia reprezentowanie obiektu bez aproksymacji. Podobnie jak systemy graficzne, które mogą rysować tylko odcinki, zmuszają nas do tworzenia przybliżeń dla gładkich krzywych, niektóre reprezentacje modelowania brył opisują wiele obiektów w sposób przybliżony. Idealnie schemat reprezentacji powinien uniemożliwiać tworzenie nieprawdziwej reprezentacji (to znaczy takiej, która nie odpowiada bryle), takiej jak na rys. 10.1b. Ponadto tworzenie poprawnej reprezentacji powinno być łatwe, na ogół za pomocą interakcyjnego systemu modelowania brył. Chcielibyśmy, żeby obiekty zachowywały *domknięcie* przy wykonywaniu obrotów, przesunięć i innych operacji. Tak więc wykonywanie tych operacji na poprawnych bryłach powinno dawać tylko poprawne bryły. Reprezentacja powinna być zwarta ze względu na oszczędność miejsca, co z kolei może zaoszczędzić czas przesyłania w systemie rozproszonym. Wreszcie reprezentacja powinna umożliwić korzystanie z efektywnych algorytmów obliczania odpowiednich właściwości fizycznych i, co najważniejsze dla nas, tworzenie obrazów.

Opracowanie reprezentacji, która miałaby te wszystkie cechy, jest rzeczywiście trudne i często są potrzebne kompromisy. Omawiając główne reprezentacje używane dzisiaj, zwrócimy uwagę na te elementy, które umożliwią zrozumienie, jak te reprezentacje mogą pasować do programów graficznych. Więcej szczegółów z naciskiem na aspekty modelowania brył można znaleźć w pracach [REQU80; MORT85; MANT88].

10.2. Regularyzowane operacje boolowskie

Niezależnie od tego, w jaki sposób reprezentujemy obiekty, chcielibyśmy wykonywać na nich operacje umożliwiające tworzenie nowych brył. Jedną z najbardziej intuicyjnych i popularnych metod wykonywania operacji na obiektach jest wykorzystanie *zbioru operacji boolowskich*, takich jak suma, różnica i przecięcie, których działanie ilustruje rys. 10.2. Te opera-

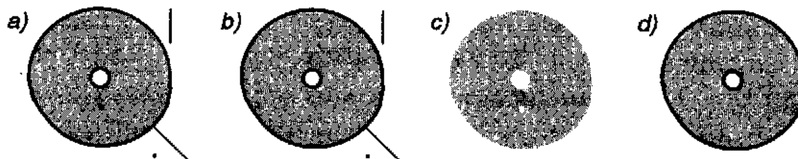


Rys. 10.2. Operacje boolowskie: a) obiekty A i B; b) $A \cup B$; c) $A \cap B$; d) $A - B$; e) $B - A$

cje są odpowiednikami 3D znanych operacji boolowskich 2D. Jednak stosując zwykły zbiór operacji boolowskich do dwóch brył niekoniecznie otrzymamy bryłę. Na przykład zwykłe przecięcie dwóch sześcianów, które mają wspólny tylko jeden wierzchołek, daje punkt.

Zamiast korzystać ze zwykłego zbioru operatorów boolowskich będziemy korzystali z *regularyzowanych operatorów boolowskich* [REQU77], oznaczanych jako $\cup^*$, $\cap^*$, $-^*$ i tak zdefiniowanych, że wykonanie takich operacji na bryłach da zawsze bryły. Na przykład regularyzowane boolowskie przecięcie dwóch sześcianów, które mają tylko jeden wierzchołek wspólny, daje obiekt pusty.

W celu wyjaśnienia, na czym polega różnica między zwykłymi a regularyzowanymi operatorami, rozważmy dowolny obiekt zdefiniowany jako zbiór punktów podzielonych na punkty wewnętrzne i punkty brzegowe (rys. 10.3a). *Punkty brzegowe* są to te punkty, których odległość od obiektu i od uzupełnienia obiektu jest równa zeru. Punkty brzegowe nie muszą być częścią obiektu. *Zbiór domknięty* zawiera wszystkie jego punkty brzegowe, a *zbiór otwarty* ich nie zawiera. Suma zbioru i zbioru jego punktów brzegowych jest określana jako *domknięcie* zbioru (rys. 10.3b), który sam jest zbiorem domkniętym. *Brzeg* zbioru domkniętego jest zbiorem jego punktów brzegowych, a *wnętrze*, pokazane na rys. 10.3c, składa się ze wszystkich innych punktów zbioru i dlatego jest uzupełnieniem brzegu w stosunku do obiektu. *Regularyzacja* zbioru jest definiowana jako domknięcie punktów wewnętrznych zbioru.



Rys. 10.3. Regularyzacja obiektu: a) obiekt jest zdefiniowany przez punkty wewnętrzne, pokazane jako jasnoszare, i punkty brzegowe; punkty brzegowe, które są częścią obiektu są pokazane jako czarne; pozostałe punkty brzegowe są pokazane jako ciemnoszare; obiekt ma doczepione i niepołączone punkty i odcinki i we wnętrzu jest punkt brzegowy, który nie jest częścią obiektu; b) domknięcie obiektu; wszystkie punkty brzegowe są częścią obiektu; punkt brzegowy zanurzony we wnętrzu części a) jest teraz częścią wnętrza; c) wnętrze obiektu; doczepione i niepołączone punkty i odcinki zostały wyeliminowane; d) regularyzacja obiektu polega na domknięciu jego wnętrza

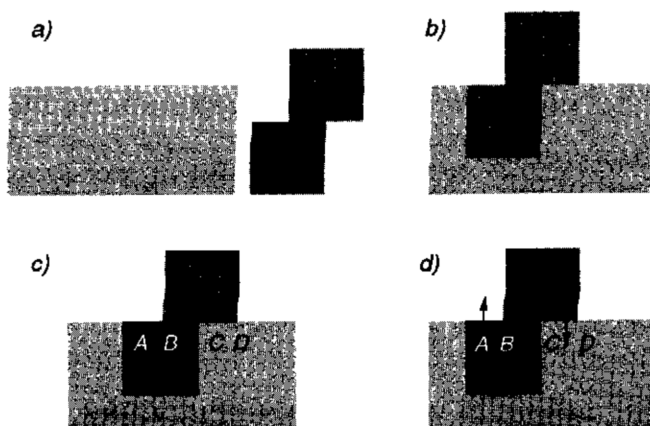
Na rysunku 10.3d pokazano domknięcie obiektu z rys. 10.3c i tym samym regularyzację obiektu z rys. 10.3a. Zbiór, który jest równy swojej własnej regularyzacji, jest określany jako *zbiór regularny*. Zauważmy, że zbiór regularny nie może zawierać punktu brzegowego, który nie sąsiaduje z jakimś punktem wewnętrznym; dlatego nie może on mieć „doczepionych'' punktów brzegowych, odcinków albo powierzchni. Może-

my zdefiniować każdy regularyzowany operator boolowski w zależności od odpowiedniego zwykłego operatora boolowskiego jako

$$A\ op^*B = \text{domknięcie}(\text{wnętrze}(A\ op\ B)) \tag{10.1}$$

przy czym *op* jest jedną z operacji $\cup$, $\cap$ albo $-$. Regularyzowane operatory boolowskie zastosowane do zbiorów regularnych tworzą tylko zbiory regularne.

Porównamy teraz działanie zwykłych i regularyzowanych operacji boolowskich w zastosowaniu do zbiorów regularnych. Weźmy pod *uwagę dwa obiekty z rys. 10.4a umieszczone jak na rys. 10.4b. Zwykłe* boolowskie przecięcie dwóch obiektów zawiera przecięcie wnętrza i brzegu każdego obiektu z wnętrzem i brzegiem innego obiektu (rys. 10.4c). Regularyzowane przecięcie boolowskie dwóch obiektów, pokazane na rys. 10.4d, zawiera przecięcie ich wnętrz i przecięcie wnętrza każdego obiektu z brzegiem drugiego obiektu, ale nie zawiera podzbioru przecięć ich brzegów. Kryterium użyte do zdefiniowania tego podzbioru określa, czym regularyzowane przecięcie boolowskie różni się od zwykłego przecięcia boolowskiego, do którego należą wszystkie części przecięcia brzegów.



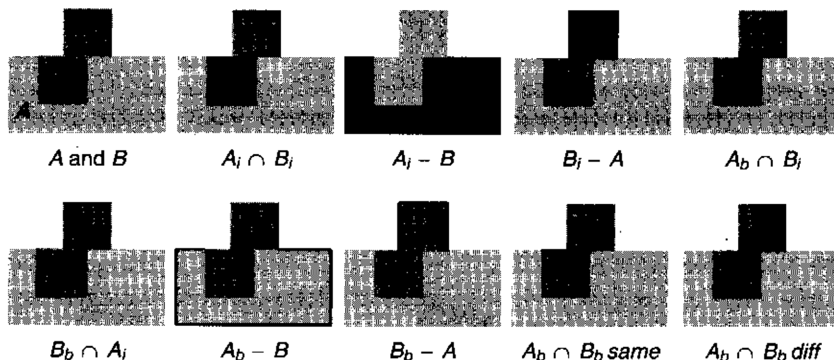
Rys. 10.4. Przecięcie boolowskie: a) przekroje dwóch obiektów; b) położenia obiektów przed przecięciem; c) zwykłe przecięcie boolowskie daje doczepioną ścianę, pokazaną jako linia *CD* na przekroju; d) regularyzowane przecięcie boolowskie zawiera kawałek wspólnego brzegu w wynikowym brzegu, jeżeli oba obiekty leżą po tej samej jego stronie (*AB*), i nie zawiera go, jeżeli obiekty leżą po przeciwnych stronach (*CD*). Przecięcia brzeg-wnętrze są zawsze włączone (*BC*)

Intuicyjnie kawałek przecięcia brzeg-brzeg jest włączony do regularyzowanego przecięcia boolowskiego wtedy i tylko wtedy, gdy wnętrza obu obiektów leżą po tej samej stronie tego kawałka wspólnego brzegu. Ponieważ punkty wewnętrzne obu obiektów, które bezpośrednio sąsia-

dują z tą częścią brzegu, należą do przecięcia, kawałek brzegu musi również być włączony po to, żeby zachować domknięcie. Rozważmy przypadek kawałka wspólnego brzegu, który leży na koplanarnych ścianach dwóch wielościanów. Określenie, czy wnętrza leżą po tej samej stronie wspólnego brzegu, jest proste, jeżeli oba obiekty są tak zdefiniowane, że normalne do ich powierzchni są skierowane na zewnątrz (albo do wewnątrz). Wnętrza są po tej samej stronie, jeżeli normalne są skierowane w tym samym kierunku. Dlatego segment *AB* na rys. 10.4d jest włączony. Pamiętajmy, że te części brzegu jednego obiektu, które przecinają się z wnętrzem innego obiektu, takie jak odcinek *BC*, są zawsze włączone.

Zastanówmy się, co się stanie, jeżeli wnętrza obiektów leżą po przeciwnych stronach wspólnego brzegu, jak w przypadku segmentu *CD*. W takich przypadkach żaden z punktów wewnętrznych sąsiadujący z brzegiem nie jest włączany do przecięcia. Dlatego kawałek wspólnego brzegu nie sąsiaduje z żadnym punktem wewnętrznym wynikowego obiektu i nie jest włączony do regularyzowanego przecięcia. To dodatkowe ograniczenie dla włączania kawałków wspólnego brzegu zapewnia regularność wynikowego zbioru. Normalna do powierzchni każdej ściany wynikowego brzegu obiektu jest normalną do powierzchni którejkolwiek powierzchni, która wniosła udział do tej części brzegu. (Jak zobaczymy w rozdz. 14, normalne do powierzchni są ważne przy cieniowaniu obiektów.) Po określeniu, które ściany leżą na brzegu, włączamy krawędź albo wierzchołek przecięcia brzeg-brzeg do brzegu przecięcia, jeżeli krawędź albo wierzchołek sąsiadują z jedną z tych ścian.

Wyniki działania każdego regularyzowanego operatora można zdefiniować w zależności od zwykłych operatorów zastosowanych do brzegów i wnętrz obiektów. Tablica 10.1 pokazuje, jak regularyzowane operatory są zdefiniowane dla dowolnych obiektów *A* i *B*; na rysunku 10.5 pokazano wyniki wykonywania operacji. A_b i A_i są odpowiednio brzegiem i wnętrzem *A*. $A_b \cap B_b$ *same* jest tą częścią brzegu wspólną dla



Rys. 10.5. Zwykłe operacje boolowskie na podzbiorach dwóch obiektów

A i B, dla której A_i i B_i leżą po tej samej stronie. Ten wynik jest dla przypadku jakiegoś punktu b na wspólnym brzegu, jeżeli przynajmniej jeden punkt i sąsiadujący z nim należy zarówno do A_i, jak i do B_i. $A_b \cap B_b$ *diff* jest tą częścią brzegu wspólną dla A i B, dla której A_i i B_i leżą po przeciwnych stronach. Jest to prawdą dla b, jeżeli nie sąsiaduje on z żadnym takim punktem i. Każdy regularyzowany operator jest zdefiniowany przez sumę zbiorów związanych z tymi wierszami, które mają znak (•) w kolumnie operatora.

Tablica 10.1. Regularyzowane operacje boolowskie

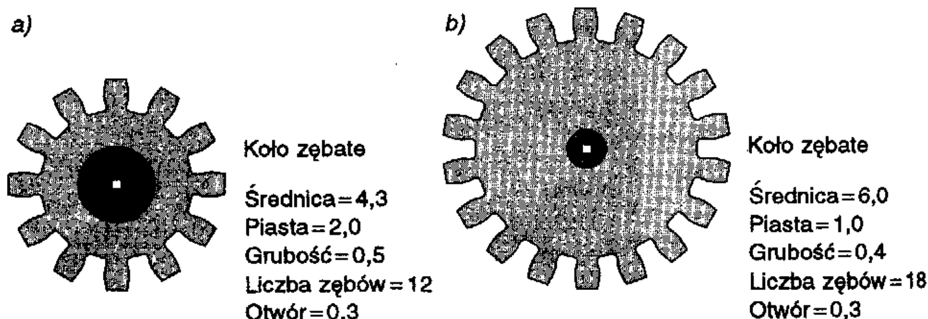
Zbiór	$A \cup^* B$	$A \cap^* B$	$A -^* B$
$A_i \cap B_i$	•	•	
$A_i - B$	•		•
$B_i - A$	•		
$A_b \cap B_i$		•	
$B_b \cap A_i$		•	•
$A_b - B$	•		•
$B_b - A$	•		
$A_b \cap B_b$ *same*	•	•	
$A_b \cap B_b$ *diff*			•

Zauważmy, że we wszystkich przypadkach każdy kawałek wynikowego brzegu obiektu jest na brzegu jednego lub kilku oryginalnych obiektów. Jeżeli obliczamy $A \cup {}^*B$ albo $A \cap {}^*B$, to normalna do powierzchni ściany wyniku jest odziedziczona po normalnej do powierzchni odpowiedniej ściany jednego lub obu oryginalnych obiektów. Jednak w przypadku $A - {}^*B$ normalna do powierzchni każdej ściany wyniku, *dla której B zostało wykorzystane do usunięcia części A, musi* wskazywać w przeciwnym kierunku niż normalna do powierzchni B tej ściany. Odpowiada to kawałkowi brzegu $A_b \cap B_b$ *diff* i $B_b \cap A_i$. Alternatywnie $A - {}^*B$ może być przepisane jako $A \cap {}^*\overline{B}$. Możemy otrzymać $\overline{B}$ (uzupełnienie B) uzupełniając wnętrze B i zmieniając kierunek normalnej jego brzegu.

Regularyzowane operatory boolowskie są używane w większości reprezentacji, które będziemy omawiali, jako metoda interfejsu użytkownika przy budowaniu złożonych obiektów z prostych obiektów. *Operatory te są również włączone bezpośrednio do jednej z reprezentacji*, a mianowicie do konstrukcyjnej geometrii brył. W następnych punktach opiszemy różne sposoby niedwuznacznego reprezentowania brył.

10.3. Kopiowanie prymitywów

Przy *kopiowaniu prymitywów* system modelujący definiuje zbiór prymitywów 3D, odpowiednich dla danego zastosowania. Te prymitywy typowo są parametryzowane nie tylko ze względu na przekształcenia z rozdz. 7, ale również ze względu na inne właściwości. Na przykład prymitywem może być regularny ostrosłup, którego liczba ścian bocznych, spotykających się w wierzchołku, jest określana przez użytkownika. Kopie prymitywów są podobne do parametryzowanych obiektów, takich jak menu z rozdz. 2, przy czym obiektami są bryły. O parametryzowanym obiekcie można myśleć jak o definiowaniu rodziny części, której członkowie różnią się kilkoma parametrami – jest to ważna koncepcja dla metod CAD znana jako *technologia grup*. Kopiowanie prymitywów jest często używane w relatywnie złożonych obiektach, np. koła zębate albo śruby, które są trudne do określenia na zasadzie kombinacji boolowskich prostszych obiektów, a są łatwe do opisania przez kilka parametrów wysokiego poziomu. Na przykład koło zębate może być parametryzowane za pomocą średnicy i liczby zębów (rys. 10.6).

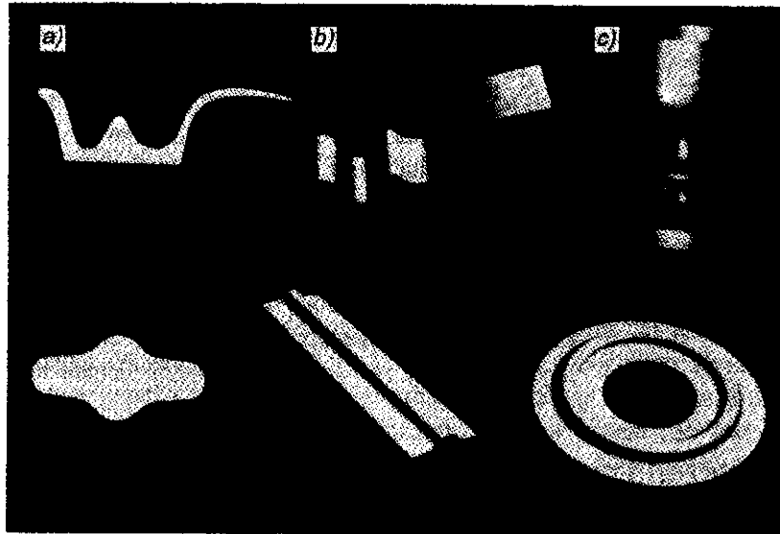


Rys. 10.6. Dwa koła zębate zdefiniowane metodą kopiowania prymitywów

Chociaż możemy budować hierarchię kopiowanych prymitywów, każda kopia w węźle-liściu wciąż jest obiektem definiowanym niezależnie. Przy pisaniu prymitywów nie robi się żadnych ustaleń, jeśli chodzi o łączenie obiektów w obiekty wyższego rzędu, korzystając na przykład z regularyzowanych operacji boolowskich. Dlatego jedynym sposobem tworzenia nowego rodzaju obiektu jest napisanie definiującego go kodu. Podobnie dla każdego prymitywu muszą być indywidualnie pisane programy rysowania obiektów albo określania ich ogólnych właściwości.

10.4. Reprezentacje z przesuwaniem

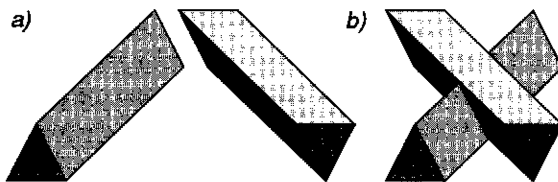
Przesuwanie obiektu wzdłuż trajektorii w przestrzeni definiuje nowy obiekt. Najprostszy sposób przesunięcia polega na zdefiniowaniu obszaru 2D i przesunięciu go wzdłuż liniowej ścieżki prostopadłej do płaszczyzny obszaru w celu utworzenia bryły. Jest to naturalny sposób reprezentowania obiektów wykonywanych na zasadzie wytłaczania metalu albo plastyku przez matrycę o odpowiednim kształcie. W takich prostych przypadkach objętość każdej uzyskanej bryły jest iloczynem pola wzoru i długości przesunięcia. Proste rozszerzenie polega na skalowaniu wzoru w czasie przesuwania w celu wytworzenia zwężających się obiektów albo przesuwaniu przekroju wzdłuż ścieżki, która nie jest prostopadła do powierzchni wzoru. *Przesunięcia obrotowe* są definiowane jako obracanie wzoru wokół osi. Na rysunku 10.7 pokazano dwa obiekty oraz bryły uzyskane na zasadzie prostego przesuwania albo obracania.



Rys. 10.7. Przesunięcia (a) wzorów 2D są używane do definiowania (b) brył przesuwanych, (c) brył obrotowych. (Utworzone za pomocą systemu modelowania Alpha_1. Za zgodą University of Utah.)

Przesuwanymi obiektami nie muszą być wzory 2D. Przesuwanie brył jest użyteczne przy modelowaniu obszarów wyciętych za pomocą noża obrabiarki albo robota przesuwającego się po określonej ścieżce. Przesunięcia, w których wielkość, kształt albo orientacja wzoru zmieniają się w trakcie przesuwania i w których trajektoria może być dowolna, są określane jako *przesunięcia ogólne*. Przesunięcia ogólne przekrojów 2D są znane w wizji komputerowej jako *walce uogólnione* [BINF71]

i są na ogół modelowane jako parametryzowane przekroje 2D przesuwane pod kątami prostymi wzdłuż dowolnej krzywej. Przesunięcia ogólne są szczególnie trudne do efektywnego modelowania. Na przykład trajektoria i wzór mogą prowadzić do samoprzecięcia obiektu, co komplikuje obliczania bryły. Ponadto przesunięcia ogólne nie zawsze generują bryły. Na przykład przesuwanie wzoru 2D w jego własnej płaszczyźnie generuje inną płaszczyznę 2D.

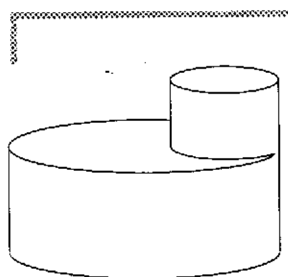


Rys. 10.8. Dwie proste bryły uzyskane w wyniku przesunięcia obiektów 2D (trójkąty) (a); połączenie brył z rysunku a) nie jest prostą bryłą otrzymaną za pomocą przesunięcia wzoru 2D (b)

Na ogół trudno jest stosować regularyzowane operacje boolowskie dla zbiorów do brył opisanych metodą przesuwania wzoru bez wcześniejszej konwersji na jakąś inną reprezentację. Nawet proste przesunięcia nie są domknięte dla regularyzowanych operacji boolowskich. Na przykład połączenie dwóch prostych przesunięć na ogół nie jest zwykłym przesunięciem, tak jak pokazano na rys. 10.8. Mimo jednak problemów związanych z domknięciami i obliczeniami przesunięcia są naturalną i intuicyjną metodą konstruowania różnych obiektów. Z tego powodu wiele systemów modelowania brył umożliwia użytkownikom konstruowanie obiektów metodą przesuwania wzoru, ale pamięta obiekty w jednej z innych reprezentacji, które będziemy omawiali dalej.

10.5. Reprezentacje brzegowe

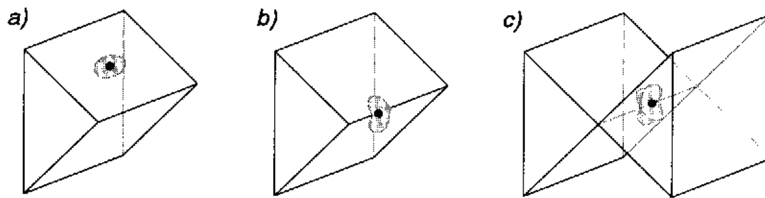
Reprezentacja brzegowa (znana również pod nazwą *b-rep*) przypomina naiwną reprezentację, jaką omawialiśmy w p. 10.1 – opisuje ona obiekt za pomocą powierzchni ograniczających, wierzchołków, krawędzi i ścian. Niektóre reprezentacje brzegowe są ograniczone do płaskich wielokątowych powierzchni i mogą wymagać, żeby ściany były wypukłymi wielokątami albo trójkątami. Określenie, co tworzy ścianę, może być szczególnie trudne, jeżeli dopuści się powierzchnie krzywoliniowe (rys. 10.9). Powierzchnie krzywoliniowe są często aproksymowane wielokątami. Powierzchnie krzywoliniowe mogą być również reprezentowane jako płaty powierzchni, jeżeli algorytmy, które są wykorzystywane do obliczeń przy tej reprezentacji, dopuszczają krzywe przecięć, któ-



Rys. 10.9. Ile ścian ma ten obiekt?

re w ogólnym przypadku będą wyższego stopnia niż oryginalne powierzchnie. Reprezentacje brzegowe wyrosły z prostych reprezentacji omawianych we wcześniejszych rozdziałach i są używane w wielu współczesnych systemach modelowania. Ze względu na zalety tych metod w grafice opracowano wiele efektywnych metod tworzenia gładkich, cieniowanych obrazów budowanych z wielokątów; wiele z tych metod omówiono w rozdz. 14.

Wiele systemów b-rep dopuszcza jedynie bryły, których brzegi są *dwuwymiarowymi rozmaitościami*. Z definicji każdy punkt na brzegu dwuwymiarowej rozmaitości ma pewne dowolnie małe sąsiedztwo punktów, które mogą być traktowane topologicznie tak samo jak dysk na płaszczyźnie. Oznacza to, że jest ciągła odpowiedniość jeden do jeden między sąsiedztwem punktów i dysku, tak jak na rysunku 10.10a i b.



Rys. 10.10. Przy dwuwymiarowej rozmaitości każdy punkt, pokazany jako czarna kropka, ma sąsiedztwo otaczających punktów, które jest topologicznym dyskiem, zaznaczonym na rysunkach a) i b) na szaro; jeżeli obiekt nie jest dwuwymiarową rozmaitością, to ma punkty, które nie mają sąsiedztwa, będącego topologicznym dyskiem (c)

Na przykład, jeżeli więcej niż dwie ściany mają wspólną krawędź (rys. 10.10c), to dowolne sąsiedztwo punktu na tej krawędzi zawiera punkty z każdej z tych ścian. Jest intuicyjnie oczywiste, że nie ma ciągłej odpowiedniości jeden do jeden między tym sąsiedztwem a dyskiem na płaszczyźnie, chociaż dowód matematyczny wcale nie jest trywialny. Dlatego powierzchnia na rys. 10.10c nie jest 2-zwrotna. Chociaż niektóre współczesne systemy nie mają tego ograniczenia, zawężamy naszą dyskusję reprezentacji brzegowych do 2-zwrotnych, chyba że bezpośrednio stwierdzimy co innego.

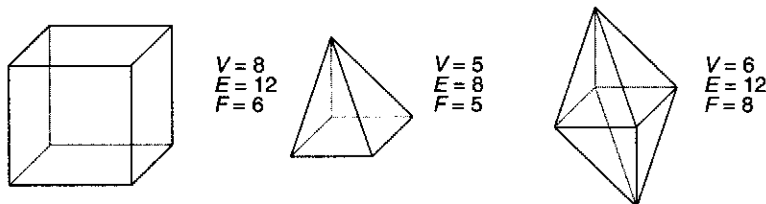
10.5.1. Wielościan i reguła Eulera

Wielościan jest bryłą ograniczoną przez zbiór wielokątów, których każda krawędź należy do parzystej liczby wielokątów (dokładnie dwóch wielokątów w przypadku dwuwymiarowych rozmaitości) i które spełniają niektóre dodatkowe ograniczenia (omówione niżej). *Wielościan*

prosty jest to taki wielościan, który może być przekształcony w kulę – to znaczy, wielościan, który w przeciwieństwie do torusa nie ma dziur. Reprezentacja brzegowa wielościanu prostego spełnia regułę Eulera, która wyraża niezmienniczą zależność między liczbą wierzchołków, krawędzi i ścian zwykłego wielościanu

$$V - E + F = 2 \tag{10.2}$$

przy czym V jest liczbą wierzchołków, E – liczbą krawędzi, a F – liczbą ścian. Na rysunku 10.11 pokazano wielościany proste i odpowiednie liczby wierzchołków, krawędzi i ścian. Zauważmy, że reguła obowiązuje również, jeżeli dopuści się krawędzie krzywoliniowe i nieplanarne ściany. Reguła Eulera formułuje konieczne, ale niewystarczające warunki na to, żeby obiekt był wielościanem prostym. Można zbudować obiekty, które spełniają tę regułę, ale nie ograniczają objętości – wystarczy doczepić jedną lub więcej ścian doczepionych albo krawędzi do poprawnej bryły, tak jak na rys. 10.1b. Aby zagwarantować, żeby obiekt był bryłą, są potrzebne dodatkowe ograniczenia: każda krawędź musi łączyć dwa wierzchołki i należeć dokładnie do dwóch ścian, przynajmniej trzy krawędzie muszą się spotykać w każdym wierzchołku i ściany nie mogą się przenikać.

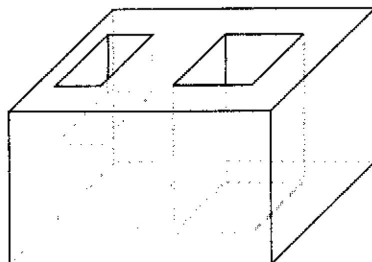


Rys. 10.11. Kilka prostych wielościanów z wartościami V, E i F. W każdym przypadku $V - E + F = 2$

Uogólnioną regułę Eulera stosuje się do dwuwymiarowych rozmaitości, które mają ściany z dziurami

$$V - E + F - H = 2(C - G) \tag{10.3}$$

przy czym H jest liczbą dziur w ścianach, G – liczbą dziur, które przechodzą przez obiekt, a C – liczbą niezależnych elementów (części) obiektu (rys. 10.12). Jeżeli obiekt ma jeden element, to jego G jest określane jako jego *genus*; jeżeli jest wiele elementów, to G jest sumą klas jego elementów. Jak poprzednio potrzebne są również dodatkowe kryteria zagwarantowania, że obiekt jest bryłą.



$$V - E + F - H = 2(C - G)$$
24 36 15 3 1 1

Rys. 10.12. Wielościan klasyfikowany według równania (10.3) z dwiema dziurami w górnej ścianie i jedną w dolnej ścianie

Baumgart wprowadził pojęcie zbioru *operatorów Eulera*, które działają na obiektach spełniających regułę Eulera w celu przekształcania obiektów w nowe obiekty, dla których również reguła obowiązuje, przez dodawanie i usuwanie wierzchołków, krawędzi i ścian [BAUM74]. Braid, Hillyard i Stroud [BRAI78] pokazują, jak niewielka liczba operatorów Eulera wystarcza do przekształcania obiektów przy założeniu, że nie wymaga się, by pośrednie obiekty były prawdziwymi bryłami; Mäntylä [MÄNT88] dowodzi, że wszystkie poprawne reprezentacje brzegowe można skonstruować za pomocą skończonej sekwencji operatorów Eulera. Inne operatory, które nie wpływają na liczbę wierzchołków, krawędzi albo ścian, mogą być zdefiniowane jako deformujące obiekt przez przesunięcie istniejących wierzchołków, krawędzi albo ścian.

Być może, najprostszą możliwą reprezentacją brzegową jest lista ścian wielokątów, z których każdy jest reprezentowany przez listę współrzędnych wierzchołków. W celu reprezentowania kierunku, w którym jest skierowana każda ściana, numerujemy wierzchołki w porządku zgodnym z ruchem wskazówek zegara, patrząc z zewnątrz bryły. W celu uniknięcia powielania współrzędnych wspólnych dla ścian możemy reprezentować każdy wierzchołek ściany za pomocą indeksu do listy współrzędnych. W tej reprezentacji krawędzie są wyobrażane pośrednio przez pary sąsiednich wierzchołków na liście wierzchołków wielokąta. Krawędzie mogą być również reprezentowane bezpośrednio jako pary wierzchołków, przy czym każda ściana może być teraz zdefiniowana jako lista indeksów do listy krawędzi. Te reprezentacje były omówione bardziej szczegółowo w p. 9.1.1.

10.5.2. Operacje boolowskie

Reprezentacje brzegowe mogą być łączone za pomocą regularyzowanych operatorów boolowskich w celu tworzenia nowych reprezentacji brzegowych [REQU85]. Sarraga [SARR83] i Miller [MILL87] omawia-

ją algorytmy, które określają przecięcia między powierzchniami drugiego stopnia. Algorytmy łączenia wielościanów są omawiane w pracach [TURN84; REQU85; PUTN86; LAID86], a Thibault i Naylor [THIB87] opisują metodę wykorzystującą reprezentację brył za pomocą drzew binarnych dzielących przestrzeń, omawianą w p. 10.6.4.

Jedno z podejść [LAID86] polega na przejrzeniu wielokątów obu obiektów, podzieleniu ich, jeśli jest to konieczne po to, żeby zapewnić, że przecięcie wierzchołka, krawędzi albo ściany jednego obiektu z dowolnym wierzchołkiem, krawędzią albo ścianą innego obiektu jest wierzchołkiem, krawędzią albo ścianą obu. Następnie wielokąty każdego obiektu są klasyfikowane względem drugiego obiektu w celu określenia czy leżą wewnątrz, na zewnątrz, czy na brzegu. Odwołując się do tabl. 10.1 zauważmy, że, ponieważ mamy do czynienia z reprezentacją brzegową, jesteśmy zainteresowani tylko ostatnimi sześcioma wierszami, z których każdy reprezentuje jakąś część jednego albo obu brzegów A_b i B_b oryginalnych obiektów. Po podziale każdy wielokąt jednego obiektu jest albo całkowicie wewnątrz innego obiektu ($A_b \cap B_i$, albo $B_b \cap A_i$), całkowicie na zewnątrz drugiego obiektu ($A_b - B$ albo $B_b - A$) albo częścią wspólnego brzegu ($A_b \cap B_b$ *same* albo $A_b \cap B_b$ *diff*).

Wielokąt może być klasyfikowany metodą śledzenia promieni omawianą w p. 13.4.1. Konstruujemy wektor w kierunku normalnej do powierzchni wielokąta i z punktu należącego do wnętrza wielokąta, a potem znajdujemy najbliższy wielokąt innego obiektu, który przecina wektor. Jeżeli żaden wielokąt nie jest przecinany, to oryginalny wielokąt jest na zewnątrz drugiego obiektu. Jeżeli najbliższy przecinany wielokąt jest koplanarny z oryginalnym wielokątem, to przecięcie to jest przecięciem typu brzeg-brzeg i porównanie normalnych do wielokątów pokazuje, jakiego rodzaju jest to przecięcie ($A_b \cap B_b$ *same* albo $A_b \cap B_b$ *diff*). W przeciwnym przypadku sprawdza się iloczyn skalarny normalnych do dwóch wielokątów. Dodatnia wartość iloczynu wskazuje, że oryginalny wielokąt jest wewnątrz innego obiektu, a ujemny iloczyn skalarny wskazuje, że oryginalny wielokąt jest na zewnątrz. Zerowa wartość iloczynu skalarnego występuje wówczas, gdy wektor jest w płaszczyźnie przeciętego wielokąta; w tym przypadku wektor jest nieco zakłócany i ponownie szuka się przecięcia z wielokątami innych obiektów.

Informacja o sąsiedztwie wierzchołków może być wykorzystana do tego, żeby uniknąć narzutu związanego z klasyfikowaniem w ten sposób każdego wielokąta. Jeżeli wielokąt jest sąsiedni (to znaczy ma wspólne wierzchołki) z klasyfikowanym wielokątem i nie spotyka powierzchni innego obiektu, to wielokątowi jest przypisywana taka sama klasyfikacja. Wszystkie wierzchołki na wspólnym brzegu między wierzchołkami mogą być oznaczane w czasie wstępnej fazy podziału wielokątów. To,

czy wielokąt spotyka powierzchnię innego obiektu, można określić sprawdzając, czy ma wierzchołki brzegowe.

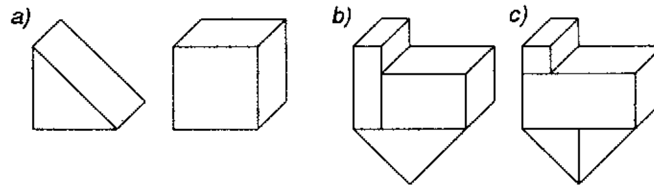
Każde zaklasyfikowanie wielokąta określa, czy jest on zachowywany czy odrzucany w operacji tworzenia złożonego obiektu, jak to opisano w p. 10.2. Na przykład przy tworzeniu sumy operacja odrzuca każdy wielokąt należący do jednego obiektu, który jest wewnątrz drugiego obiektu. Operacja zachowuje każdy wielokąt z każdego obiektu, który nie jest we wnętrzu drugiego, z wyjątkiem wielokątów koplanarnych. Wielokąty koplanarne są odrzucane, jeżeli mają przeciwne normalne do powierzchni, a tylko jeden z pary jest zachowywany, jeżeli kierunki normalnych są takie same. Decyzja o tym, który wielokąt zachować, jest ważna, jeżeli obiekty są wykonane z różnych materiałów. Chociaż $A \cup {}^*B$ ma to samo geometryczne znaczenie co $B \cup {}^*A$, może być w tym przypadku istotna różnica wizualna i można tak zdefiniować operację, żeby w przypadku wielokątów koplanarnych faworyzowała jeden ze swoich argumentów.

10.6. Reprezentacje z podziałem przestrzennym

W reprezentacjach z *podziałem przestrzennym* bryła jest dekomponowana na zbiór sąsiadujących nie przecinających się brył, które są prostszymi prymitywami od oryginalnej bryły, choć niekoniecznie tego samego typu. Prymitywy mogą być różnego typu, wielkości, mieć różne położenie, sposób parametryzacji i orientację, podobnie jak różne klocki w zestawie zabawek dla dzieci. To, jak dalece obiekt zostanie zdekomponowany, zależy od tego, jak proste muszą być prymitywy, żeby można było łatwo wykonywać interesujące operacje.

10.6.1. Dekompozycja na komórki

Jedna z najogólniejszych postaci podziału przestrzennego jest określana jako *dekompozycja na komórki*. Każdy system dekompozycji na komórki definiuje zbiór prostych komórek, które na ogół są parametryzowane i często są zakrzywione. Dekompozycja na komórki różni się od kopiowania prymitywów tym, że możemy składać bardziej złożone obiekty z prostych prymitywów w sposób wstępujący na zasadzie ich „sklejania". Operacja *sklejania* może być traktowana jako ograniczona postać łączenia, w której obiekty nie mogą się przecinać. Dalsze ograniczenia co do sklejania komórek często wymagają, żeby dwie komórki miały wspólny punkt, krawędź albo ścianę. Chociaż reprezentacja typu dekompozycja na komórki jest niedwuznaczna, niekoniecznie jest jedyna

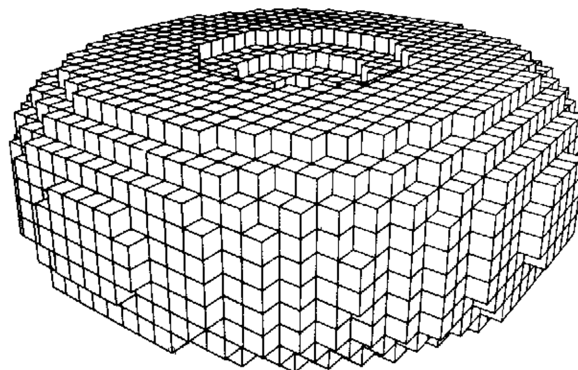


Rys. 10.13. Komórki pokazane na rysunku a) mogą być przekształcane w celu skonstruowania obiektu w różny sposób tak jak na rysunkach b) i c). Nawet jeden typ komórki wystarcza, aby spowodować dwuznaczności

(rys. 10.13). Dekompozycja na komórki jest również trudna do testowania, ponieważ każda para komórek musi potencjalnie być testowana ze względu na przecięcie. Niemniej dekompozycja na komórki jest ważną reprezentacją przy wykonywaniu analizy metodą elementów skończonych.

10.6.2. Reprezentacja wokselowa

Reprezentacja wokselowa jest szczególnym przypadkiem dekompozycji na komórki, w której bryła jest dekomponowana na identyczne komórki uporządkowane według stałej regularnej siatki. Te komórki często są nazywane *wokselami*, przez analogię do pikseli. Na rysunku 10.14 pokazano obiekt reprezentowany metodą wokselową. Najczęściej spotykanym rodzajem komórki jest sześcian. Przy reprezentacji obiektu metodą wokselową interesuje nas tylko obecność lub nieobecność jednej komórki w każdym miejscu siatki. Wyznaczając reprezentację obiektu musimy tylko zdecydować, które komórki są zajęte, a które nie. Obiekt może więc być kodowany za pomocą jednoznacznej listy zajętych komórek.



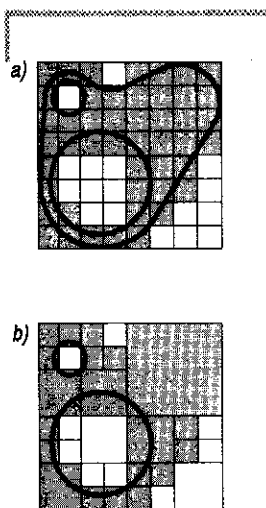
Rys. 10.14. Torus reprezentowany metodą wokselową [Z A.H.J. Christiansen, SIGGRAPH'80 Conference Proceedings, *Computer Graphics* (14)3, July 1980. Za zgodą Association for Computing Machinery, Inc.]

Łatwo jest sprawdzić, czy komórka jest wewnątrz, czy na zewnątrz bryły, i określić, czy dwa obiekty są sąsiednie. Metoda wokselowa jest często wykorzystywana w zastosowaniach biomedycznych do reprezentowania danych przestrzennych otrzymanych z takich źródeł jak komputerowa tomografia CAT.

Przy wszystkich zaletach metoda wokselowa ma kilka oczywistych wad, podobnie jak reprezentacja kształtu 2D za pomocą mapy bitowej o głębokości 1 bitu. Nie ma miejsca na pojęcie *częściowej* zajętości. Dlatego wiele brył może być tylko aproksymowanych; przykładem jest torus z rys. 10.14. Jeżeli komórki są sześcianami, to jedynymi obiektami, które mogą być reprezentowane dokładnie, są obiekty o ścianach równoległych do ścian sześcianu, i których wierzchołki pokrywają się z wierzchołkami siatki. Podobnie jak piksele w przypadku mapy bitowej, komórki mogą być w zasadzie tak małe, jak to jest potrzebne do zwiększenia dokładności reprezentacji. Istotnym jednak ograniczeniem staje się zajętość pamięci, ponieważ do reprezentowania obiektu o rozdzielczości n wokseli w każdym kierunku trzeba n^3 komórek.

10.6.3. Drzewa ósemkowe

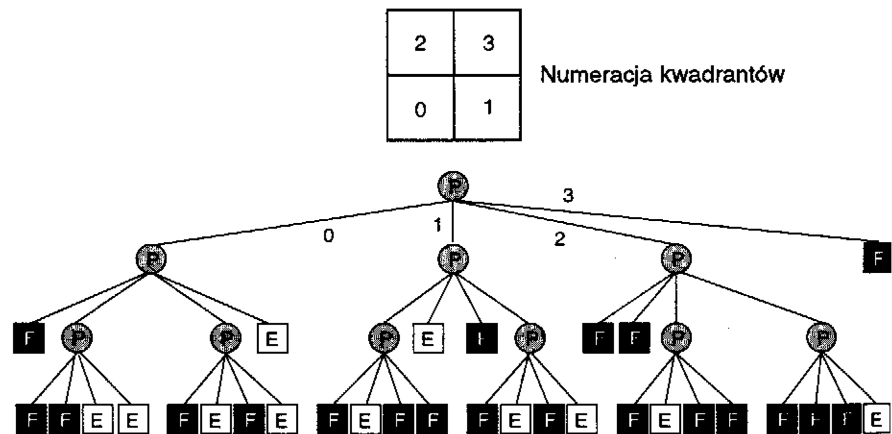
Drzewa ósemkowe to hierarchiczny wariant metody wokselowej zaprojektowany dla zastosowań, w których jest duże zapotrzebowanie na pamięć. Z kolei drzewa ósemkowe wywodzą się z *drzew czwórkowych*, formatu 2D wykorzystywanego do kodowania obrazów. Jak powiedziano w pracy przeglądowej Sameta [SAME84], obie reprezentacje zostały opracowane niezależnie przez różnych badaczy: drzewa czwórkowe na przełomie lat sześćdziesiątych i siedemdziesiątych [np. WARN69; KLIN71], a drzewa ósemkowe na przełomie lat siedemdziesiątych i osiemdziesiątych [np. HUNT78; REDD78; JACK80; MEAG80; MEAG82].



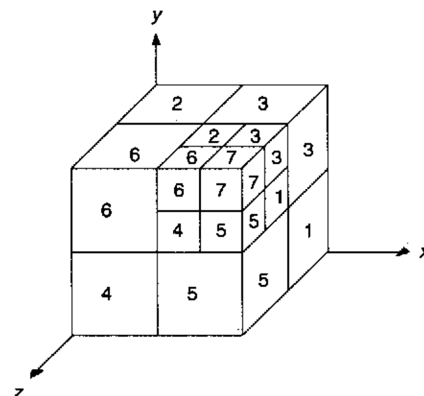
Rys. 10.15. Obiekt reprezentowany metodą wokselową na płaszczyźnie (a); metodą drzewa czwórkowego (b)

Podstawową ideą reprezentacji czwórkowych i ósemkowych jest siła podziału binarnego wynikająca z zasady dziel i zwyciężaj. Drzewo czwórkowe otrzymuje się w wyniku kolejnego podziału płaszczyzny 2D w obu kierunkach na kwadranty (rys. 10.15). Gdy drzewo czwórkowe jest wykorzystywane do reprezentowania obszaru na płaszczyźnie, wówczas każdy kwadrant może być pełny, częściowo zapełniony albo pusty (albo odpowiednio czarny, szary i biały) zależnie od tego, jaka część kwadrantu jest pokryta przez obszar. Kwadrant częściowo pokryty jest dzielony dalej rekursywnie na podkwadranty. Podział kontynuuje się dopóty, dopóki wszystkie kwadranty nie będą jednorodne (pełne albo puste), albo dopóty, dopóki nie zostanie osiągnięta założona głębokość podziału. Gdy cztery siostrzane kwadranty są pełne lub puste, wówczas są usuwane i ich częściowo pełny przodek jest zastępowany pełnym albo pustym węzłem.

(Zamiast tego można użyć podejścia wstępującego w celu uniknięcia procesu usuwania i mieszania [SAME90b].) Na rysunku 10.15 na poziomie odcięcia (końcowym) każdy częściowo zapełniony węzeł jest klasyfikowany jak pełny. Kolejny podział może być reprezentowany jako drzewo z częściowo zapełnionymi kwadrantami w węzłach wewnętrznych i pełnymi albo pustymi kwadrantami w liściach (rys. 10.16). Ta idea może być porównana z algorytmem podziału obszaru Warnocka omawianym w p. 13.5.2. Jeżeli osłabi się kryteria klasyfikowania węzłów jako jednorodnych, a dopuści klasyfikowanie węzłów, które są powyżej albo poniżej pewnego progu jako pełne albo puste, to reprezentacja staje się bardziej zwarta, ale mniej dokładna. Drzewo ósemkowe jest podobne do czwórkowego, z tą różnicą, że w przypadku drzewa ósemkowego trzy wymiary są rekursywnie dzielone na oktanty (rys. 10.17).



Rys. 10.16. Struktura drzewa czwórkowego dla obiektu z rys. 10.15; F – całkowite pokrycie, P – częściowe pokrycie, E – brak pokrycia



Rys. 10.17. Numeracja oktantów. Oktant 0 nie jest widoczny

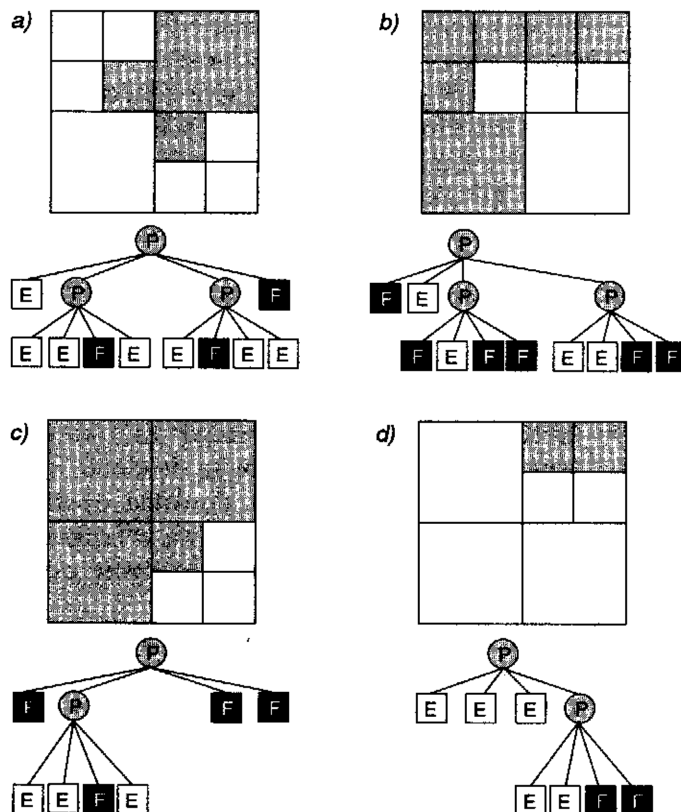
Kwadranty są często numerowane od 0 do 3, a oktanty od 0 do 7. Ponieważ nie został przyjęty żaden standardowy sposób numerowania, są używane również nazwy mnemoniczne. Kwadranty są określane według kierunków stron świata jak na kompasie, względem wspólnego punktu środkowego tych kwadrantów, odpowiednio: NW, NE, SW i SE. Oktanty są nazywane podobnie i rozróżnia się oktanty lewe (L) i prawe (R), górne (U) i dolne (D), przednie (F) i tylne (B), co prowadzi do oznaczeń: LUF, LUB, LDF, LDB, RUF, RUB, RDF i RDB.

Z wyjątkiem kilku przypadków, można wykazać, że liczba węzłów w reprezentacji czwórkowej albo ósemkowej obiektu jest proporcjonalna odpowiednio do obwodu albo powierzchni [HUNT78; MEAG80]. Ta zależność jest słuszna, bowiem podział węzłów wynika tylko z potrzeby reprezentowania brzegu kodowanego obiektu. Jedynymi wewnętrznymi węzłami, które są dzielone, są te, przez które przechodzi fragment brzegu. Dlatego dowolna operacja wykonywana na jednej z tych struktur, która jest liniowa w funkcji liczby węzłów, które zawiera, jest również wykonywana w czasie proporcjonalnym do wielkości obwodu albo powierzchni.

Operacje boolowskie i przekształcenia. Wiele pracy włożono w opracowanie efektywnych algorytmów pamiętania i przetwarzania drzew czwórkowych i ósemkowych [SAME84; SAME90a; SAME90b]. Na przykład operacje boolowskie są łatwe zarówno dla drzew czwórkowych, jak i ósemkowych. W celu obliczenia sumy albo przecięcia U dwóch drzew S i T przeszukujemy oba drzewa równolegle w kierunku zstępującym.

Na rysunku 10.18 pokazano operacje dla drzew czwórkowych; uogólnienie na drzewa ósemkowe jest oczywiste. Sprawdza się każdą parę odpowiadających sobie węzłów. Rozpatrzmy przypadek sumy. Jeżeli któryś z węzłów pary jest czarny, to odpowiedni węzeł czarny jest dodawany do U. Jeżeli jeden z węzłów pary jest biały, to w U jest tworzony odpowiedni węzeł o wartości drugiego węzła z pary. Jeżeli oba węzły z pary są szare, to do U jest dodawany węzeł szary i algorytm jest rekursywnie stosowany do potomków pary. W tym ostatnim przypadku trzeba sprawdzić potomków nowego węzła w U po zastosowaniu do nich algorytmu. Jeżeli są one całe czarne, to są usuwane i ich przodek w U jest zamieniany z szarego na czarny.

Wykonywanie prostych przekształceń na drzewach czwórkowych i ósemkowych jest łatwe. Na przykład obrót wokół osi o wielokrotność kąta 90° jest wykonywany na zasadzie rekursywnego obracania potomków na każdym poziomie. Skalowanie z potęgą 2 i odbicia są również proste. Przesunięcia są nieco bardziej złożone, podobnie jak ogólne przekształcenia. Ponadto, tak jak w reprezentacji wokselowej, przy ogólnych przekształceniach występuje problem aliasingu.



Rys. 10.18. Wykonywanie operacji boolowskich na drzewach czwórkowych: a) obiekt S i jego drzewo czwórkowe; b) obiekt T i jego drzewo czwórkowe; c) $S \cup T$; d) $S \cap T$

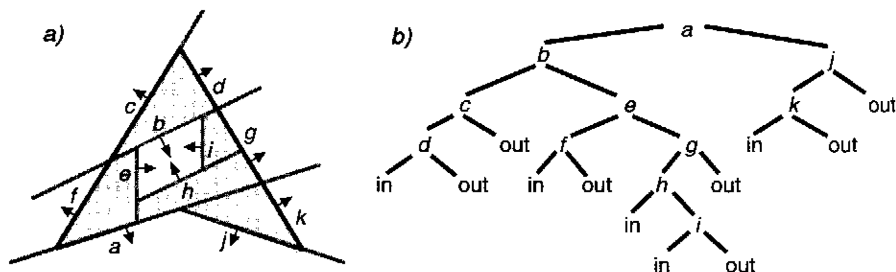
Znajdowanie sąsiadów. Ważną operacją na drzewach czwórkowych i ósemkowych jest znajdowanie sąsiedniego węzła dla danego węzła, (wspólna ściana, krawędź albo wierzchołek) o tej samej albo innej wielkości. Węzeł w drzewie czwórkowym ma sąsiadów w 8 możliwych kierunkach. Jego sąsiedzi N, S, E, W są sąsiadami ze wspólną krawędzią, a sąsiedzi NW, NE, SW i SE mają wspólny wierzchołek. Węzeł drzewa ósemkowego ma sąsiadów w 26 możliwych kierunkach: 6 sąsiadów ze wspólną ścianą, 12 sąsiadów ze wspólną krawędzią i 8 sąsiadów ze wspólnym wierzchołkiem.

Samet [SAME89a] opisuje sposób znajdowania sąsiada węzła w określonym kierunku. Przy stosowaniu metody zaczynamy od oryginalnego węzła i wchodzimy do góry drzewa czwórkowego albo ósemkowego dopóty, dopóki nie zostanie znaleziony pierwszy wspólny przodek oryginalnego węzła i sąsiada. Wtedy kontynuujemy ruch w dół w celu znalezienia żądanego sąsiada. Trzeba tu rozwiązać efektywnie

dwa problemy: znalezienie wspólnego przodka i określenie, który z potomków jest sąsiadem. Najprostszy przypadek polega na znalezieniu sąsiada węzła w drzewie ósemkowym w kierunku *d* jednej z jego ścian: L, R, U, D, F albo B. Jeżeli wspinamy się po drzewie zaczynając od oryginalnego węzła, to wspólny przodek jest pierwszym węzłem, który jest nieosiągalny od potomka po stronie węzła. Na przykład, jeżeli prowadzimy poszukiwania sąsiada dla L, to pierwszym wspólnym przodkiem jest pierwszy węzeł, który nie jest osiągalny od potomka LUF, LUB, LDF albo LDB. Jest to prawdą, ponieważ węzeł, który został osiągnięty przy starcie z jednego z tych potomków, nie może mieć żadnego potomka, który jest (L) lewym sąsiadem oryginalnego węzła. Po znalezieniu wspólnego przodka schodzi się jego poddrzewem w zwierciadlany sposób w stosunku do ścieżki od oryginalnego węzła do przodka, odbitej wokół wspólnej granicy. Jeżeli sąsiad jest większy od oryginalnego węzła, to przeszukuje się tylko część *odbitej ścieżki.*

10.6.4. Drzewa BSP

Drzewa ósemkowe rekursywnie dzielą przestrzeń za pomocą płaszczyzn, które są zawsze wzajemnie prostopadłe i które dokonują podziału na pół w każdym z trzech wymiarów na każdym poziomie drzewa. *Drzewa binarnego podziału przestrzeni* (BSP) rekurencyjnie dzielą przestrzeń na pary podprzestrzeni, przy czym każda para jest dzielona płaszczyzną o dowolnej orientacji i położeniu. Taka struktura drzewa binarnego była pierwotnie wykorzystana w grafice do określania widocznych powierzchni (p. 13.5). Później Thibault i Naylor [THIB87] stosowali drzewa BSP do reprezentowania dowolnego wielościanu. Każdy węzeł drzewa BSP jest związany z płaszczyzną i ma dwa wskaźniki na potomków, po jednym dla każdej strony płaszczyzny. Zakładając, że normalne wskazują obiekt z zewnętrz, lewy potomek jest za albo na płaszczyźnie, a prawy potomek jest przed albo na zewnątrz płaszczyzny. Jeżeli dalej podzielimy półprzestrzeń po jednej stronie płaszczyzny, to jej potomek jest korzeniem poddrzewa; jeżeli półprzestrzeń jest homogeniczna, to jej potomek jest liściem reprezentującym obszar znajdujący się albo całkowicie wewnątrz wielościanu albo całkowicie na zewnątrz wielościanu. Te homogeniczne obszary są nazywane *wewnętrznymi* albo *zewnętrznymi komórkami.* W celu uwzględnienia ograniczonej precyzji numerycznej wykonywania operacji, z każdym węzłem jest związana *grubość* odpowiedniej płaszczyzny. Każdy punkt leżący w granicach tej tolerancji jest traktowany jako leżący na płaszczyźnie.



Rys. 10.19. Reprezentacja drzewa BSP w 2D: a) niewypukły wielokąt ograniczony czarną linią; linie definiujące półpłaszczyzny są ciemnoszare, a komórki wewnętrzne są jasnoszare; b) drzewo BSP

Koncepcja podziału charakterystyczna dla drzew BSP, podobnie jak w przypadku drzew ósemkowych i czwórkowych nie, zależy od liczby wymiarów. Na rysunku 10.19a pokazano niewypukły wielokąt 2D ograniczony czarnymi liniami. Komórki wewnętrzne są jasnoszare, a linie określające półprzestrzenie są ciemnoszare; normalne są skierowane na zewnątrz. Odpowiednie drzewo BSP pokazano na rys. 10.19b. W 2D obszary wewnętrzne i zewnętrzne tworzą pokrycie powierzchni wielokątami wypukłymi; w 3D obszary wewnętrzne i zewnętrzne tworzą pokrycie przestrzeni wypukłymi wielościanami. Dlatego drzewo BSP może reprezentować dowolną bryłę niewypukłą z dziurami jako sumę wypukłych obszarów wewnętrznych.

Rozważmy zadanie określenia, czy punkt leży wewnątrz, na zewnątrz, czy na bryle – jest to problem znany jako problem *klasyfikacji punktu* [TILO80]. Drzewo BSP może być użyte do klasyfikowania punktu na zasadzie filtrowania tego punktu w dół drzewa, zaczynając w korzeniu. W każdym węźle punkt jest podstawiany do równania płaszczyzny węzła i rekursywnie przekazywany do lewego potomka, jeżeli leży poza (na) płaszczyzną, albo do prawego potomka, jeżeli leży przed (na zewnątrz) płaszczyzną. Jeżeli węzeł jest liściem, to punkt otrzymuje wartość liścia, *out* albo *in*. Jeżeli punkt leży na płaszczyźnie węzła, to jest przekazywany do obu potomków i są porównywane sposoby zaklasyfikowania. Jeżeli są takie same, to punkt otrzymuje tę wartość; jeżeli są różne, to punkt leży na brzegu między obszarami *out* oraz *in* i jest klasyfikowany jako *in*. To podejście może być rozszerzone na klasyfikowanie odcinków i wielokątów. Przeciwnie jednak niż w przypadku punktu, odcinek albo wielokąt mogą leżeć częściowo po obu stronach płaszczyzny. Dlatego w każdym węźle, którego płaszczyzna przecina odcinek albo wielokąt, muszą być one podzielone (obcięte) na części, które są przed płaszczyzną, za płaszczyzną albo na płaszczyźnie, i te części są klasyfikowane oddzielnie.

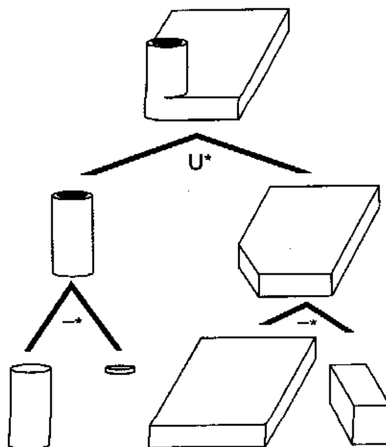
Thibault i Naylor opisują algorytm budowania drzewa BSP na podstawie reprezentacji b-rep, wykonywania operacji boolowskich w celu połączenia drzewa BSP z reprezentacją b-rep i określania

tych części, które leżą na brzegu drzewa BSP [THIB87]. Algorytmy działają na drzewach BSP, których węzły są powiązane z listą wielokątów na płaszczyźnie węzła. Wielokąty umieszcza się w drzewie za pomocą odmiany algorytmu budowania drzewa BSP naszkicowanego w p. 13.5.

Chociaż drzewa BSP dają elegancką i prostą reprezentację, wielokąty są dzielone w czasie konstrukcji drzewa i w czasie wykonywania operacji boolowskich, wskutek czego notacja staje się mniej zwarta niż w przypadku innych reprezentacji. Wykorzystując jednak niezależność drzew BSP od liczby wymiarów można opracować zamkniętą algebrę Boole'a dla drzew BSP 3D, która rekurencyjnie zależy od reprezentacji wielokątów jako drzew 2D, krawędzi jako drzew 1D i punktów jako drzew 0D [NAYL90].

10.7. Konstruktywna geometria brył

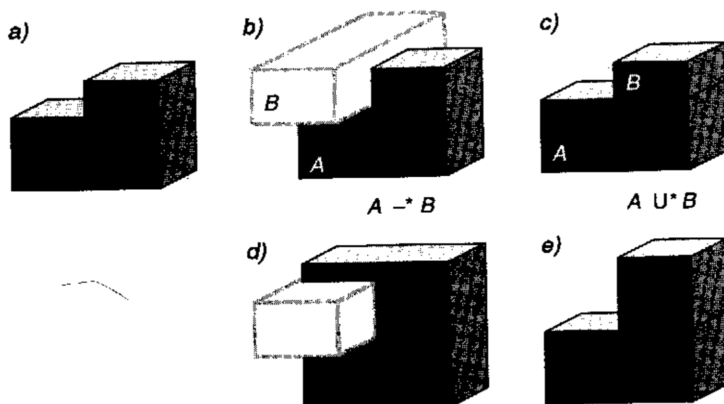
W *konstruktywnej geometrii brył* (CSG) łączy się proste prymitywy za pomocą regularyzowanych operatorów boolowskich, które są bezpośrednio włączone do reprezentacji. Obiekt jest pamiętany jako *drzewo z operatorami w węzłach wewnętrznych i prostymi prymitywami* w liściach (rys. 10.20). Niektóre węzły reprezentują operatory boolowskie, inne zaś wykonują przesunięcia, obroty i skalowanie, podobnie jak przy strukturach hierarchicznych z rozdz. 7. Ponieważ operatory boolowskie na ogół nie są przemienne, krawędzie drzewa są *uporządkowane*.



Rys. 10.20. Reprezentacja CSG obiektu i odpowiednie drzewo

W celu określenia właściwości fizycznych albo wykonania obrazów musimy móc łączyć właściwości liści, aby otrzymać właściwości korzenia. Ogólna strategia przetwarzania polega na przesuwaniu się po drzewie metodą zstępującą, jak w rozdz. 7, w celu łączenia węzłów i liści. Złożoność tego zadania zależy od reprezentacji, w której są zapamiętane obiekty, i od tego, czy w danej chwili musi być otrzymana pełna reprezentacja złożonego obiektu. Na przykład algorytmy regularyzowanych operatorów boolowskich dla reprezentacji brzegowej, omówione w p. 10.5, łączą reprezentacje brzegowe dwóch węzłów, tworząc trzecią reprezentację brzegową i są trudne w implementacji. Znacznie prostszy algorytm CSG omówiony w rozdz. 15 pracy [FOLE90] tworzy jednak obraz na zasadzie przetwarzania reprezentacji liści bez bezpośredniego ich łączenia.

W niektórych implementacjach prymitywy są prostymi bryłami, takimi jak sześciany albo kule, dzięki czemu jest pewność, że wszystkie regularyzowane kombinacje są również bryłami. W innych systemach do prymitywów należą półpłaszczyzny, które same nie są ograniczonymi bryłami. Na przykład sześcian może być określony jako przecięcie sześciu półprzestrzeni, a walec jako nieskończony walec, który jest obcięty na górze i na dole przez półprzestrzenie. Korzystanie z półprzestrzeni wprowadza problem testowania poprawności, ponieważ nie wszystkie kombinacje tworzą bryły. Półprzestrzenie są jednak użyteczne przy takich operacjach jak obcinanie obiektu przez płaszczyznę, operację, która w innym przypadku musiałaby być wykonana przy wykorzystaniu ściany innej bryły. Bez półprzestrzeni mielibyśmy dodatkowy narzut, ponieważ regularyzowane operacje boolowskie muszą być wykonywane z całym obiektem wykonującym obcinanie, nawet jeżeli jest potrzebna tylko jedna ściana obcinająca.



Rys. 10.21. Obiekt pokazany na rysunku a) może być zdefiniowany za pomocą różnych operacji CSG, jak na rysunkach b) i c). Wyciągnięcie górnej ściany w rysunkach b) i c) do góry daje różne obiekty pokazane na rysunkach d) i e)

O reprezentacjach komórkowej i wokselowej można myśleć jako o specjalnych przypadkach reprezentacji CSG, w której jedynym operatorem jest operator sklejania: połączenie dwóch obiektów, które mogą *się stykać, ale muszą mieć rozłączne wnętrza* (to znaczy obiekty muszą mieć zerowe regularyzowane przecięcie boolowskie).

CSG nie zapewnia unikatowości reprezentacji. Ta cecha może być częściowo myląca w systemach, które umożliwiają użytkownikowi manipulowanie obiektami liści za pomocą różnych operatorów. Stosując tę samą operację do dwóch początkowo takich samych obiektów otrzymuje się dwa różne wyniki (rys. 10.21). Niemniej, dzięki możliwości dokonywania edycji przez usuwanie, dodawanie, zastępowanie i modyfikowanie poddrzew, w połączeniu ze względnie zwartym sposobem pamiętania modeli, reprezentacja CSG stała się dominującą reprezentacją modelowania brył.

10.8. Porównanie reprezentacji

Omówiliśmy pięć głównych reprezentacji: kopiowanie prymitywów, przesuwanie, reprezentację brzegową (b-rep), podział przestrzeni (w tym komórkową, wokselową, drzewa ósemkowe i drzewa BSP) oraz CSG. Porównajmy je korzystając z kryteriów wprowadzonych w p. 10.1.

- *Dokładność.* Metody podziału przestrzeni i wielokątowa metoda b-rep dają tylko przybliżenia dla wielu obiektów. W niektórych zastosowaniach, np. znajdowanie ścieżki dla robota, to ograniczenie nie jest istotną wadą, jeżeli przybliżenie jest obliczane z dostateczną dokładnością (często dość niewielką). Jednak z punktu widzenia praktycznego rozdzielczość potrzebna do utworzenia wizualnie przyjemnej grafiki albo do obliczenia z dostateczną dokładnością wzajemnego oddziaływania obiektów może być zbyt duża. Metody gładkiego cieniowania omawiane w rozdz. 14 nie zajmują się artefaktami wizualnymi powodowanymi przez krawędzie wielokątowe. Dlatego w systemach wykorzystujących grafikę dobrej jakości często stosuje się reprezentację CSG z prymitywami, które nie są wielościanami, i b-rep z powierzchniami krzywoliniowymi. Metoda kopiowania prymitywów może również tworzyć obrazy dobrej jakości, ale nie pozwala na łączenie dwóch prostych obiektów za pomocą operatorów boolowskich.
- *Dziedzina.* Dziedzina obiektów, które mogą być reprezentowane metodami kopiowania prymitywów i przesuwania, jest ograniczona. Dla porównania, podejścia typu podział przestrzeni mogą reprezentować dowolną bryłę, chociaż często jedynie w przybliżeniu. Jeżeli dopuści się obok wielokątów ograniczonych odcinkami inne rodzaje

ścian i krawędzi, to reprezentacja b-rep może być użyta do reprezentowania szerokiej klasy obiektów. Wiele jednak systemów b-rep ma ograniczenie tylko do prostych powierzchni i topologii. Na przykład mogą one umożliwiać kodowanie tylko kombinacji powierzchni drugiego stopnia, które są 2-zwrotne.

- *Unikatowość.* Tylko drzewa ósemkowe i metoda wokselowa gwarantują unikatowość reprezentacji: jest tylko jeden sposób reprezentowania obiektu o określonej wielkości i położeniu. W przypadku drzew ósemkowych trzeba wykonać pewne przetwarzanie, żeby uzyskać pewność, że reprezentacja jest w pełni zredukowana (to znaczy, że żaden szary węzeł nie ma samych czarnych albo samych białych potomków). Kopiowanie elementów w ogóle nie gwarantuje unikatowości; na przykład kula może być reprezentowana albo jako prymityw kulisty, albo elipsoidalny. Jeżeli jednak starannie wybierze się zestaw prymitywów, to unikatowość można zapewnić.
- *Poprawność.* Spośród wszystkich reprezentacji najtrudniejszą ze względu na zapewnienie poprawności jest metoda b-rep. Nie tylko struktury danych wierzchołków, krawędzi i ścian mogą być niezgodne, ale również ściany albo krawędzie mogą się przecinać. Dla kontrastu, każde drzewo BSP reprezentuje poprawny zbiór przestrzenny, ale niekoniecznie ograniczoną bryłę. W celu stwierdzenia poprawności drzewa CSG (które jest zawsze ograniczone, jeżeli prymitywy są ograniczone) albo drzewa ósemkowego trzeba wykonać tylko kilka lokalnych sprawdzeń syntaktycznych, a żadne sprawdzenia nie są potrzebne dla reprezentacji wokselowej.
- *Domknięcie.* Prymitywy tworzone za pomocą kopiowania prymitywów nie mogą być w ogóle łączone, a proste przesuwanie nie jest domknięte ze względu na operacje boolowskie. Dlatego na ogół żadna z tych operacji nie jest używana jako wewnętrzna reprezentacja w systemach modelowania. Chociaż określona reprezentacja brzegowa może nie mieć domknięcia ze względu na operacje boolowskie (na przykład brak możliwości reprezentowania innych niż 2-zwrotne), takich przypadków można na ogół uniknąć.
- *Zwartość i efektywność.* Reprezentacje są często klasyfikowane ze względu na to, czy tworzą modele przetworzone czy nieprzetworzone. Modele nieprzetworzone zawierają informacje, które muszą być przetwarzane (albo obliczane) w celu wykonania podstawowych operacji, np. określenie brzegu obiektu. Z punktu widzenia możliwości użycia operacji boolowskich CSG tworzy modele nieprzetworzone w tym sensie, że za każdym razem, gdy trzeba wykonać operację, musimy poruszać się po drzewie obliczając wyrażenia. W konsekwencji zaletami CSG są jej zwartość i zdolność do szybkiego rejestrowania operacji boolowskich i zmian przekształceń oraz do

szybkiego wycofywania się, ponieważ wiąże się to tylko z budową węzła drzewa. Drzewa ósemkowe i BSP mogą być również traktowane jako modele nieprzetworzone, podobnie jak sekwencje operatorów Eulera, które tworzą reprezentację b-rep. Reprezentacje b-rep i wokselowa są często traktowane jako przekształcone modele, jeżeli zostały już wykonane wszystkie operacje boolowskie potrzebne do utworzenia obiektu. Zauważmy, że używanie tych pojęć jest względne; na przykład, jeżeli operacja, która ma być wykonana, polega na określeniu, czy punkt jest wewnątrz obiektu, więcej pracy trzeba wykonać obliczając b-rep niż obliczając równoważne drzewo CSG.

Jak to omówiono w rozdz. 13, istnieje kilka efektywnych algorytmów generowania obrazów obiektów zakodowanych metodą b-rep. Chociaż reprezentacja wokselowa i drzew ósemkowych może dać tylko zgrubne przybliżenie dla wielu obiektów, algorytmy wykorzystywane do manipulowania takimi obiektami są na ogół prostsze niż ich odpowiedniki dla innych reprezentacji. Powinny więc być *wykorzystywane w systemach modelowania brył wspieranych sprzętowo*, przeznaczonych dla zastosowań, w których zwiększona szybkość, z jaką mogą na nich być wykonywane operacje boolowskie, równoważy niezbyt dużą dokładność otrzymywanych obrazów.

W niektórych systemach wykorzystuje się kilka reprezentacji, ponieważ niektóre operacje są bardziej efektywne w jednych reprezentacjach niż w innych. Na przykład GMSOLID [BOYS82] korzysta z CSG ze względu na zwartość i z b-rep ze względu na szybkość wyszukiwania odpowiednich danych nie określonych bezpośrednio w CSG, jak na przykład rodzaje sąsiedztwa. Chociaż reprezentacja CSG w GMSOLID zawsze odzwierciedla bieżący stan modelowanego obiektu, jego reprezentacja b-rep jest uaktualniana tylko wówczas, gdy są wykonywane operacje wymagające tej reprezentacji. Obok systemów, które korzystają z dwóch pełnych i niezależnych reprezentacji, wyprowadzając jedną z drugiej w miarę potrzeby, są również systemy hybrydowe, które schodzą do pewnego poziomu szczegółowości w jednej reprezentacji, a potem przełączają się do drugiej, przy czym nigdy nie dublują informacji. Niektóre z podniesionych przez nas problemów wielokrotnej reprezentacji albo hybrydowej reprezentacji są omawiane w pracy [MILI89].

Jak pokazano w p. 10.1, reprezentacje drutowe zawierające tylko informacje o wierzchołkach i krawędziach, bez odniesienia do ścian, są wewnętrznie niejednoznaczne. Markowsky i Wesley opracowali algorytm znajdowania wszystkich wielościanów, które mogą być reprezentowane za pomocą danego modelu drutowego [MARK80], oraz towarzyszący algorytm, który generuje wszystkie wielościany, które mogłyby wygenerować dany rzut 2D [WESL81].

10.9. Interfejsy użytkownika dla systemu modelowania brył

Opracowanie interfejsu użytkownika dla systemu modelowania brył stanowi świetną okazję do wykorzystania w praktyce metod projektowania interfejsów omawianych w rozdz. 8. Do wykorzystania w interfejsie graficznym nadaje się wiele metod, łącznie z bezpośrednim wykorzystaniem regularyzowanych operatorów boolowskich, wyciąganiem i operatorami Eulera. W systemach CSG użytkownik może mieć możliwość dokonywania edycji obiektu na zasadzie modyfikowania albo zastępowania jednej z brył znajdujących się w liściach albo poddrzewach. Można określić operacje zlewania i ścinania krawędzi w celu złagodzenia przejścia od jednej powierzchni do drugiej. Interfejsy użytkownika w systemach, które zyskały akceptację na rynku, najczęściej są niezależne od przyjętej wewnętrznej reprezentacji. Wyjątkiem jest kopiowanie prymitywów, ponieważ w tym przypadku chodzi o zachęcenie użytkownika do myślenia o obiektach w kategoriach specjalizowanych parametrów.

W rozdziale 9 zwróciliśmy uwagę na to, że jest wiele sposobów opisania tej samej krzywej. Na przykład interfejs użytkownika w systemie rysowania krzywych może umożliwiać użytkownikowi wprowadzanie krzywych na zasadzie sterowania wektorami stycznymi Hermite'a albo określania kontrolnych punktów Béziera, krzywe natomiast są pamiętane tylko w postaci punktów kontrolnych Béziera. Podobnie system modelowania brył może umożliwiać użytkownikowi tworzenie obiektów w kilku różnych reprezentacjach, a pamiętanie ich w jeszcze innej postaci. Podobnie jak przy reprezentowaniu krzywej, każda reprezentacja wejściowa może mieć kilka istotnych zalet, dzięki którym jej wybór może być oczywisty przy tworzeniu obiektu.

Precyzja, z jaką muszą być określane obiekty, często dyktuje, jakich trzeba używać środków, żeby pomiary były wykonywane dokładnie – na przykład za pomocą lokalizatora albo przez wprowadzanie wartości numerycznych. Ponieważ położenie jednego obiektu często zależy od położenia innych obiektów, interfejsy dają możliwość ograniczania jednego obiektu przez inny. Zbliżona metoda polega na daniu użytkownikowi możliwości definiowania linii siatki w celu ograniczenia możliwych położeń obiektu.

Niektóre z podstawowych problemów projektowania interfejsu modelowania brył wynikają z potrzeby manipulowania i wyświetlania obiektów 3D metodami typowymi dla urządzeń interakcyjnych i wyświetlających 2D. Te ogólne problemy omówiono w rozdz. 8. Wiele systemów rozwiązuje niektóre z tych problemów za pomocą wyświetlania kilku okien, dzięki czemu użytkownik może oglądać obiekty równocześnie z różnych pozycji.

Podsumowanie

Modelowanie brył jest ważne w zastosowaniach CAD/CAM i graficznych. Chociaż istnieje wiele użytecznych algorytmów i systemów, które umożliwiają operowanie dotychczas opisanymi obiektami, wiele problemów jest dalej nie rozwiązanych. Jednym z najważniejszych jest zagadnienie odporności na błędy. Systemy modelowania brył na ogół mają problemy z niestabilnościami numerycznymi. Powszechnie używane algorytmy wymagają większej dokładności do przechowywania pośrednich wyników zmiennopozycyjnych, niż zezwala na to sprzęt. Na przykład algorytmy operacji boolowskich mogą zawodzić, jeżeli zastosuje się je do dwóch obiektów, z których jeden jest lekko przekształconą wersją drugiego.

Potrzebne są reprezentacje dla obiektów niesztywnych, elastycznych i łączonych. Wiele obiektów nie może być określonych z pełną dokładnością; ich kształty są określane parametrami, których wartości muszą leżeć w określonym przedziale. Obiekty określone z pewną tolerancją odpowiadają rzeczywistym obiektom wykonywanym przez takie maszyny jak tokarki czy prasy [REQU84]. Są opracowywane nowe reprezentacje dla kodowania obiektów z tolerancjami [GOSS88].

Wspólną dla wszystkich projektowanych obiektów jest koncepcja *cech indywidualnych* takich jak otwory czy wyżłobienia, które są projektowane dla specjalnych celów. Jeden z bieżących obszarów badań polega na badaniu automatycznego rozpoznawania tych cech i wnioskowaniu o intencji projektanta, co każda z nich powinna realizować [PRAT84]. Ten proces umożliwi sprawdzanie projektu pod względem poprawności zrealizowania cech zgodnie z założeniami. Na przykład, jeżeli pewna cecha jest zaprojektowana z myślą o zwiększeniu odporności części na ciśnienie, to można automatycznie sprawdzić jej zdolność do wykonywania tej funkcji. Można również sprawdzać obiekty w działaniu w celu sprawdzenia zachowania się cech.

Zadania

10.1. Zdefiniuj wyniki wykonywania operacji $\cup^*$ i $-^*$ dla dwóch wielościanów w taki sam sposób, w jaki w p. 10.2 został zdefiniowany wynik wykonania operacji $\cap^*$. Wyjaśnij, dlaczego wynikowy obiekt jest ograniczony do zbioru regularnego i określ, jak jest określana normalna do każdej ściany obiektu.

10.2. Rozważ zadanie określania, czy dopuszczalna bryła jest obiektem pustym (który nie ma objętości). Jakie trudności wiążą się z wykonaniem tego testu dla każdej z omówionych reprezentacji?

10.3. Rozważ system, w którym obiekty są reprezentowane za pomocą przesuwania przekroju i na których mogą być wykonywane regularyzowane operatory boolowskie. Jakie ograniczenia muszą być narzucone w stosunku do obiektów, żeby zapewnić domknięcie?

10.4. Wyjaśnij, dlaczego implementacje operacji boolowskich na drzewach czwórkowych albo ósemkowych nie muszą brać pod uwagę rozróżnienia między zwykłymi a regularyzowanymi operacjami opisanymi w p. 10.2.

10.5. Chociaż implikacje geometryczne stosowania regularyzowanych operatorów boolowskich są niedwuznaczne, jest mniej oczywiste, jak należy traktować właściwości obiektu. Na przykład, jakie właściwości należałoby przypisać przecięciu dwóch obiektów wykonanych z różnych materiałów? W modelowaniu rzeczywistych obiektów ten problem jest mniej istotny, ale w sztucznym świecie grafiki można przecinać ze sobą dwa dowolne materiały. Jakie rozwiązanie byłoby sensowne?

10.6. Wyjaśnij, jak drzewo czwórkowe albo ósemkowe mogłoby być użyte do przyspieszenia wskazywania 2D albo 3D w pakiecie graficznym.

10.7. Opisz, jak wykonać klasyfikację punktu metodami kopiowania prymitywów, b-rep, wokselową i CSG.

11. Światła achromatyczne i barwne

Student zajmujący się nowoczesną grafiką komputerową musi rozumieć teorię światła i barwy. Nawet rozsądne użycie kilku poziomów szarości może istotnie polepszyć wygląd wyświetlanego obiektu. Dzięki użyciu barw można było uzyskać odpowiednie wrażenia przy obserwacji obrazów pokazanych we wkładce ze zdjęciami barwnymi. Barwa jest ogromnie złożonym pojęciem – koncepcje związane z barwą mają swoje korzenie w fizyce, fizjologii, psychologii, sztuce i projektowaniu graficznym. Ten rozdział zawiera wprowadzenie w dziedzinę barwy w sposób najbardziej odpowiedni dla grafiki komputerowej.

Barwa obiektu zależy nie tylko od samego obiektu, ale również od źródła światła oświetlającego obiekt, barwy otoczenia i systemu wzrokowego człowieka. Dalej, niektóre obiekty odbijają światło (ściany, biurko, papier), a inne głównie przepuszczają światło (celofan, szkło). Jeżeli powierzchnia, która odbija wyłącznie czyste niebieskie światło, zostanie oświetlona czystym światłem czerwonym, to będzie wyglądała jak czarna. Podobnie barwa zielona oglądana przez czerwone okulary, które przepuszczają tylko czystą barwę czerwoną, będzie wyglądała jak czarna. Odkładamy rozważania niektórych z tych zagadnień i zaczynamy naszą dyskusję od wrażeń achromatycznych – to jest takich, które są opisywane jako czarne, szare albo białe.

11.1. Światło achromatyczne

Światło achromatyczne (dosłownie – brak barwy) widzimy w czarno-białym odbiorniku telewizyjnym albo monitorze komputerowym. Obserwując światło achromatyczne nie odczuwamy żadnych wrażeń zwią-

zanych z barwami czerwoną, zieloną, żółtą itd. Cecha ilościowa światła jest jedynym atrybutem światła achromatycznego. Może ona być rozważana w kategoriach energii fizycznej i wtedy są używane określenia *natężenie światła* i *luminancja* albo w kategoriach odbieranego natężenia w sensie psychofizjologicznym i w tym przypadku używa się określenia *jaskrawość*. Jak pokażemy dalej, te dwie koncepcje są związane, ale nie tożsame. Użyteczne jest nadanie wartości liczbowych różnym poziomom natężenia; przypisujemy 0 barwie czarnej, 1 barwie białej, a wartości między 0 i 1 odcieniom szarości.

Telewizja czarno-biała może wyświetlać pojedynczy piksel o różnej jasności. Drukarki wierszowe, plotery piórowe i plotery elektrostatyczne wytwarzają tylko dwa poziomy: biały (albo jasnoszary) papieru i czarny (albo ciemnoszary) atramentu albo toneru osadzanego na papierze. Niektóre metody, omawiane w dalszych punktach, umożliwiają wytwarzanie na takich, z natury dwupoziomowych urządzeniach dodatkowych poziomów szarości.

11.1.1. Wybór natężenia

Załóżmy, że chcemy wyświetlać 256 różnych poziomów natężenia. Wybieramy tę liczbę, ponieważ jasność każdego piksela w wielu obrazach jest reprezentowana przez dane 8-bitowe. Z jakich 256 poziomów natężenia powinniśmy korzystać? Z pewnością nie chcemy mieć 128 poziomów w zakresie 0 do 0,1 i pozostałych 128 w zakresie 0,9 do 1, ponieważ przejście od 0,1 do 0,9 z pewnością sprawiałoby wrażenie nieciągłości. Początkowo moglibyśmy rozmieścić poziomy równomiernie w przedziale 0 do 1, ale taki wybór pomija ważną charakterystykę oka, które jest raczej czułe na stosunki poziomów natężenia światła niż na bezwzględne wartości natężenia. To znaczy, odbieramy różnicę natężenia 0,10 i 0,11 tak samo jak różnicę natężenia 0,50 i 0,55. (Tę nieliniowość można łatwo zaobserwować: weźmy trzy żarówki 50, 100 i 150 W i włączajmy je kolejno; zauważymy, że efekt zmiany natężenia przy przejściu od 50 do 100 W jest znacznie większy niż przy przejściu od 100 do 150 W. Na skali jasności (to jest odbieranego natężenia) różnice między natężeniem światła 0,10 i 0,11 oraz 0,50 i 0,55 są sobie równe. Dlatego, jeśli chcemy uzyskać jednakowe kroki na skali jasności, to poziomy natężenia powinny być raczej w odstępach logarytmicznych niż liniowych.

W celu znalezienia 256 wartości natężenia światła zaczynając od najmniejszego możliwego natężenia I_0 i kończąc na maksymalnym natężeniu 1,0, przy czym każde z nich jest r razy większe od poprzedzającego, korzystamy z następujących równań:

$$I_0 = I_0,\ I_1 = rI_0,\ I_2 = rI_1 = r^2I_0,\ I_3 = rI_2 = r^3I_0,\ \ldots,\ I_{255} = r^{255}I_0 = 1 \tag{11.1}$$

Stąd

$$r = (1/I_0)^{1/255},\ I_j = r^jI_0 = (1/I_0)^{j/255}I_0 = I_0^{(255-j)/255} \quad \text{dla } 0 \leqslant j \leqslant 255 \tag{11.2}$$

i w ogólnym przypadku dla $n+1$ natężeń

$$r = (1/I_0)^{1/n},\ I_j = I_0^{(n-j)/n} \quad \text{dla } 0 \leqslant j \leqslant n \tag{11.3}$$

W przypadku tylko czterech wartości natężenia ($n = 3$) i $I_0 = 1/8$ (nierealistycznie duża wartość wybrana jedynie dla celów ilustracyjnych) równanie (11.3) określa, że $r = 2$, co daje wartości natężenia 1/8, 1/4, 1/2 i 1.

Minimalna osiągalna wartość I_0 na monitorze kineskopowym jest gdzieś w przedziale 1/200 do 1/40 maksymalnego natężenia 1,0. A więc typowa wartość I_0 jest w przedziale 0,005 do 0,025. Minimum nie jest równe 0 ze względu na odbicia światła od luminoforu w monitorze kineskopowym. Stosunek maksymalnego do minimalnego natężenia jest określany jako *zakres dynamiczny*. Dokładną wartość dla określonego kineskopu możemy znaleźć wyświetlając biały kwadrat na czarnym polu i mierząc natężenie emitowanego światła za pomocą fotometru. Pomiar taki wykonujemy w całkowicie zaciemnionym pomieszczeniu, tak żeby odbite światło otoczenia nie wpływało na wartość natężenia. Dla $I_0 = 0{,}02$ odpowiadającego zakresowi dynamicznemu 50, z równania (11.2) otrzymujemy $r = 1{,}015495...$ i pierwsze kilka i dwie ostatnie wartości natężeń dla zestawu 256 poziomów są następujące: 0,0200, 0,0203, 0,0206, 0,0209, 0,0213, 0,0216, ..., 0,9848, 1,0000.

Poprawne wyświetlenie poziomów natężenia określonych równaniem (11.1) na ekranie monitora kineskopowego jest trudnym zadaniem, a zarejestrowanie ich na filmie jest jeszcze trudniejsze, ze względu na nieliniowości kineskopu i filmu. Te trudności można obejść korzystając z metody określanej jako *gamma korekcja*, która wymaga załadowania do tablicy pośredniej monitora rastrowego skompensowanych wartości. Szczegóły procedury są podane w pracy [FOLE90].

Oczywiste jest pytanie: „ile poziomów natężenia wystarczy?" Przez „wystarczy" rozumiemy liczbę poziomów potrzebnych do takiego reprodukowania obrazów z ciągłymi przejściami barw od białej do czarnej, żeby reprodukcja sprawiała wrażenie ciągłej. Takie wrażenie uzyskuje się, gdy współczynnik r wynosi najwyżej 1,01 albo jest mniejszy (poniżej tej wartości oko nie może rozróżnić między natężeniami I_j

i I_{j+1}) [WYSZ82, str. 569]. Odpowiednią liczbę poziomów natężenia – wartość n – znajdujemy z przyrównania r do 1,01 w równaniu (11.3)

$$r = (1/I_0)^{1/n} \quad \text{lub} \quad 1{,}01 = (1/I_0)^{1/n} \tag{11.4}$$

Wyznaczając wartość n otrzymujemy

$$n = \log_{1{,}01}(1/I_0) \tag{11.5}$$

przy czym $1/I_0$ jest zakresem dynamicznym urządzenia.

W tablicy 11.1 pokazano zakresy dynamiczne $1/I_0$ dla kilku metod generacji obrazu i odpowiednie wartości n – liczby poziomów natężenia potrzebnych do utrzymania $r = 1{,}01$ i równocześnie do pełnego wykorzystania zakresu dynamicznego. Są to teoretyczne wartości przy założeniu perfekcyjnego procesu reprodukcji. W praktyce niewielkie rozmazanie będące skutkiem rozlewania się atramentu i pewnej ilości przypadkowego szumu w reprodukcji istotnie zmniejsza n dla nośników druku. Na przykład na rys. 11.1 pokazano fotografię ze wszystkimi odcieniami; na rysunku 11.2 przedstawiono reprodukcję tej samej fotografii z 4 i 32 poziomami natężenia. Przy czterech poziomach przejścia albo kontury między jednym poziomem natężenia a następnym są wyraźnie widoczne, ponieważ stosunek r między kolejnymi poziomami jest istotnie większy niż idealna wartość 1,01. Przy 32 poziomach natężenia kontury są zaledwie widoczne i dla tego konkretnego obrazu powinny zniknąć dla 64 poziomów. Ta obserwacja sugeruje, że 64 poziomy natężenia to absolutne minimum potrzebne do drukowania ciągłych obrazów czarno-białych na papierze, na jakim jest wydrukowana ta książka. Dla bardzo dobrze wyregulowanego monitora kineskopowego w idealnie czarnym pomieszczeniu uzyskanie wyższego zakresu dynamicznego wymaga użycia wielu więcej poziomów.

Rys. 11.1. Fotografia ze wszystkimi odcieniami

Rys. 11.2. Wpływ poziomów natężenia na reprodukcję obrazu: a) fotografia ze wszystkimi odcieniami reprodukowana z czterema poziomami natężenia; b) fotografia ze wszystkimi odcieniami reprodukowana z 32 poziomami natężenia. (Za zgodą Alana Paetha z University of Waterloo Computer Graphics Lab.)

Tablica 11.1. Zakres dynamiczny (I/I_0) i liczba potrzebnych poziomów natężenia $n = \log_{1,01}(I/I_0)$ dla różnych nośników wyświetlania

Nośnik wyświetlania	Typowy zakres dynamiczny	Liczba poziomów natężenia n
Kineskop	50-200	400-530
Odbitki fotograficzne	100	465
Slajdy	1000	700
Powlekany papier z drukiem czarno-białym	100	465
Powlekany papier z drukiem barwnym	50	400
Papier gazetowy z drukiem czarno-białym	10	234

11.1.2. Aproksymacja półtonowa

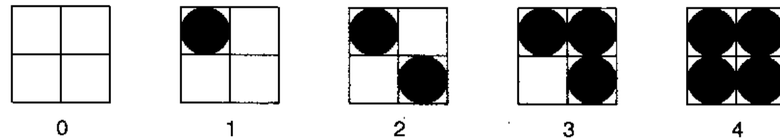
Wiele urządzeń wyświetlających i urządzeń tworzących trwałą kopię jest urządzeniami dwupoziomowymi – wytwarzają one dokładnie dwa poziomy natężenia – a nawet rastrowe urządzenia wyświetlające z dokładnością 2 albo 3 bitów na piksel tworzą mniej poziomów natężenia, niż chcielibyśmy. Jak możemy rozszerzyć liczbę dostępnych poziomów?

Odpowiedź tkwi w całkowaniu przestrzennym wykonywanym przez oko. Jeżeli obserwujemy małą powierzchnię z dostatecznie dużej odległości, to nasze oczy uśredniają małe szczegóły w niewielkim obszarze i rejestrują tylko całkowitą jasność obszaru.

Zjawisko to jest wykorzystywane przy drukowaniu fotografii czarno-białych w gazetach, magazynach i książkach metodą tak zwanych *półtonów* (określaną również w grafice komputerowej jako *metoda mikrowzorów*). Każda mała jednostka rozdzielczości jest pokrywana kółkiem czarnej farby o średnicy proporcjonalnej do zaczernienia $1 - I$ (przy czym I = jasność) obszaru w oryginalnej fotografii. Na rysunku 11.3 pokazano fragment wzoru półtonowego w znacznym powiększeniu. Zauważmy, że wzór tworzy kąt 45° z poziomem, określany jako *kąt rastra*. Półtony w gazetach wykorzystują 60 do 80 punktów na cal o zmiennej wielkości i o zmiennym kształcie [ULIC87], a półtony w magazynach i w książkach wykorzystują 110 do 200 punktów na cal.

Rys. 11.3. Metoda półtonów skutecznie zwiększa liczbę poziomów natężenia dostępnych dla nośników o niewielkim zakresie dynamicznym. Na tym powiększonym wzorze półtonowym możemy zauważyć, że wielkości plamek zmieniają się odwrotnie proporcjonalnie do jasności oryginalnej fotografii (por. rys. 11.1) (Za zgodą Alana Paetha z University of Waterloo Computer Graphics Lab.)

Wyjściowe urządzenia graficzne mogą aproksymować koła o zmiennej powierzchni dla reprodukcji półtonowych. Na przykład obszar 2 × 2 piksele monitora dwupoziomowego może być wykorzystany do tworzenia pięciu różnych poziomów natężenia kosztem zmniejszenia o połowę rozdzielczości przestrzennej wzdłuż każdej osi. Wzory pokazane na rys. 11.4 mogą być użyte do wypełnienia obszarów 2 × 2 włączonymi pikselami, których liczba n jest proporcjonalna do wymaganej jasności. Rysunek 11.5 pokazuje twarz przedstawioną w postaci cyfrowej jako matryca 351 × 351 i wyświetloną za pomocą wzoru 2 × 2.



Rys. 11.4. Pięć poziomów natężenia aproksymowanych za pomocą czterech mikrowzorów 2 × 2

Rys. 11.5. Fotografia ze wszystkimi odcieniami, zapisana w postaci cyfrowej z rozdzielczością 351 × 351 i wyświetlona z wykorzystaniem wzorów 2 × 2 z rys. 11.4. (Za zgodą Alana Paetha z University of Waterloo Computer Graphics Lab.)

Grupa $n \times n$ dwupoziomowych pikseli może dać $n^2 + 1$ poziomów natężenia. Ogólnie występuje kompromis między rozdzielczością przestrzenną a rozdzielczością poziomów natężenia. Użycie wzoru 3×3 zmniejsza rozdzielczość przestrzenną 3 razy wzdłuż każdej osi, ale daje 10 poziomów natężenia. Oczywiście możliwość wyboru jest ograniczona przez naszą ostrość widzenia (ok. 1 minuta kątowa przy normalnym oświetleniu), odległość, z której obraz jest oglądany i rozdzielczość (w punktach na cal) urządzenia graficznego.

Aproksymacja półtonowa nie jest ograniczona do wyświetlania dwupoziomowego. Rozważmy ekran z 2 bitami na piksel i stąd z 4 poziomami natężenia. Jeżeli użyjemy wzoru 2×2, to będziemy mieli do dyspozycji 4 piksele, z których każdy może przyjąć trzy wartości plus czerń; a więc mamy do dyspozycji $4 \times 3 + 1 = 13$ poziomów natężenia.

W prezentowanych dotychczas metodach zakładano, że matryca pokazywanego obrazu jest mniejsza niż matryca pikseli urządzenia wyświetlającego i stąd dla przedstawienia jednego piksela obrazu można było użyć kilku pikseli ekranu. A co będzie, jeżeli matryca urządzenia wyświetlającego i ekranu są takie same? Jedno z rozwiązań polega na użyciu metody *dyfuzji błędów* opracowanej przez Floyda i Steinberga

Rys. 11.6. Fotografia ze wszystkimi odcieniami reprodukowana metodą dyfuzji błędu Floyda-Steinberga. (Za zgodą Alana Paetha z University of Waterloo Computer Graphics Lab.)

[FLOY75]. Efekty wizualne stosowania metody dyfuzji błędów są często zadowalające. Błąd (to znaczy różnica między dokładną wartością piksela a aproksymowaną wartością faktycznie wyświetlaną) jest dodawany do wartości czterech pikseli matrycy obrazu znajdujących się z prawej strony i poniżej rozpatrywanego piksela: 7/16 błędu do piksela z prawej strony, 3/16 do piksela poniżej i z lewej strony, 5/16 do piksela bezpośrednio poniżej i 1/16 do piksela poniżej i z prawej strony. Ta strategia daje efekt rozprzestrzeniania albo dyfuzji błędu na kilka pikseli w matrycy obrazu. Rysunek 11.6 został utworzony na tej zasadzie.

Obraz *S* ma być wyświetlony w postaci macierzy jasności *I*; zmodyfikowane wartości *S* i wyświetlane wartości *I* są obliczane dla pikseli w kolejności wynikającej z przeglądania wierszy od najwyższego do najniższego:

```
K = Approximate(S[x][y]);   /* Aproksymacja S do najbliższej wyświetlanej jasności */
I[x][y] = K;                /* Rysowanie piksela w (x, y) */
error = S[x][y] – K;        /* Błąd; musi być reprezentowany zmienną typu float */

/* Krok 1: dodanie 7/16 błędu do piksela z prawej strony (x+1, y) */
S[x + 1][y] += 7 * error/16;

/* Krok 2: dodanie 3/16 błędu do piksela poniżej i z lewej strony */
S[x – 1][y – 1] += 3 * error/16;

/* Krok 3: dodanie 5/16 błędu do piksela poniżej */
S[x][y – 1] += 5 * error/16;

/* Krok 4: dodanie 1/16 błędu do piksela poniżej i z prawej strony */
S[x + 1][y – 1] += error/16;
```

W celu uniknięcia wizualnych artefaktów w wyświetlanym obrazie musimy zapewnić, żeby suma poszczególnych wartości błędów była równa dokładnie wartości *error*; nie dopuszcza się tu zaokrąglania. Możemy spełnić ten warunek obliczając błąd w kroku 4 jako *error* minus wartości błędów z trzech pierwszych kroków. Funkcja *Approximate* wyznacza wartości wyświetlanych poziomów natężenia najbliższe faktycznym wartościom piksela. Dla ekranów dwupoziomowych wartość *S* jest po prostu zaokrąglana do 0 albo 1.

Jeszcze lepsze rezultaty możemy otrzymać przeglądając wiersze od lewej strony do prawej i odwrotnie; przy przeglądaniu od prawej strony do lewej zostają odwrócone kierunki dla błędów w krokach 1, 2 i 4. Dokładne omówienie tych i innych metod dyfuzji błędów można znaleźć w pracy [ULIC87]. Inne podejścia są omawiane w pracy [KNUT87].

11.2. Barwa

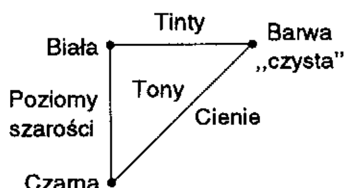
Wrażenia wzrokowe wywoływane przez światło barwne są znacznie bogatsze niż w przypadku światła achromatycznego. Przy omawianiu percepcji barwy zazwyczaj korzysta się z trzech pojęć określanych jako odcień barwy, nasycenie i jasność. Określenie *odcień* barwy dotyczy takich pojęć jak czerwony, zielony, purpurowy, żółty itd. *Nasycenie* określa, jak daleko dana barwa jest od poziomu szarości o tym samym poziomie natężenia. Barwa czerwona jest wysoce nasycona; barwa różowa jest względnie nienasycona; barwa ciemnoniebieska jest silnie nasycona; barwa niebieska jest względnie nienasycona. Pastele są względnie nienasycone; barwy nienasycone zawierają więcej światła białego niż żywe barwy nasycone. *Jasność* oznacza achromatyczny opis odbieranej jasności przy obserwacji obiektu odbijającego światło. Czwarte określenie *jaskrawość* jest używane zamiast *jasności* w odniesieniu do odbieranej jasności obiektów świecących (to jest emitujących, a nie odbijających światło) takich jak żarówka, słońce czy lampa kineskopowa.

Jeżeli w grafice komputerowej chcemy korzystać z barw w sposób precyzyjny, to musimy umieć je określić i mierzyć. Dla światła odbijanego możemy wykonać takie zadania porównując wzrokowo próbkę nieznanej barwy ze zbiorem standardowych próbek. Nieznana i wzorcowe barwy muszą być oglądane przy oświetleniu standardowym źródłem światła, ponieważ odbierana barwa powierzchni zależy zarówno od powierzchni, jak i od oświetlenia, przy jakim ta powierzchnia jest oglądana. Powszechnie używany system uporządkowania barw Munsella zawiera zbiory opublikowanych standardowych barw [MUNS76] zorganizowanych w przestrzeni trzech parametrów: odcienia barwy, wartości (określiliśmy to wyżej jako jasność) i chromy (nasycenie). Każda barwa ma nazwę i jest tak umieszczona, żeby znajdowała się w jednakowej

percepcyjnej odległości (określonej w wyniku obserwacji przez wielu obserwatorów) od swoich sąsiadów w przestrzeni barw. Kell [KELL76] wyczerpująco omawia standardowe próbki, wykresy ilustrujące przestrzeń Munsella i tablice nazw barw.

W drukarstwie i wśród profesjonalnych grafików barwy są na ogół określane na zasadzie zgodności z drukowanymi próbkami, takimi jak zestaw PANTONE MATCHING SYSTEM® [PANT91].

Artyści przy określaniu barw korzystają często z pojęć: tinty, cienie i tony, silnie nasyconych albo czystych barwników. *Tinta* powstaje w wyniku dodania białego barwnika do czystego barwnika, co zmniejsza nasycenie. *Cień* powstaje w wyniku dodania czarnego barwnika do czystego barwnika, co zmniejsza jasność. *Ton* powstaje w wyniku doda-



Rys. 11.7. Tinty, tony i cienie

nia zarówno czarnego, jak i białego barwnika do czystego pigmentu. Wszystkie te operacje powodują zmianę nasycenia i jasności barwy. Mieszanie tylko czarnego i białego pigmentu tworzy barwy szare. Na rysunku 11.7 pokazano zależności między tintami, tonami i cieniami. Procentowy udział pigmentu, który trzeba zmieszać w celu uzyskania potrzebnej barwy, może być używany do określenia barwy. System uporządkowania barw Ostwalda [OSTW31] jest podobny do modelu artystów wykorzystujących tinty, tony i cienie.

11.2.1. Psychofizyka

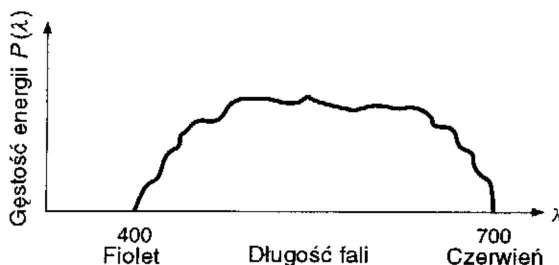
Metody Munsela oraz mieszania barwników stosowane przez artystów są metodami subiektywnymi: zależą od osądu obserwatora, oświetlenia, wielkości próbki, barwy otoczenia oraz całkowitej jasności otoczenia. Potrzebna jest zatem obiektywna ilościowa metoda określania barw; w tym celu sięgniemy do gałęzi fizyki – *kolorymetrii*. Ważnymi pojęciami w kolorymetrii są dominująca długość fali, czystość pobudzenia i luminancja.

Dominująca długość fali jest to długość fali światła, które „widzimy" jako światło barwne i odpowiada percepcyjnemu pojęciu odcienia barwy; *czystość pobudzenia* odpowiada nasyceniu barwy; *luminancja* jest wielkością związaną z natężeniem światła. Czystość pobudzenia światła barwnego jest stosunkiem czystego światła o dominującej dłu-

gości fali do światła białego, potrzebnych do określenia barwy. Całkowicie czysta barwa jest nasycona w 100% i dlatego nie zawiera światła białego; mieszaniny barwy czystej i światła białego mają nasycenia między 0 a 100%. Światło białe i poziomy szarości mają nasycenie 0% i nie zawierają żadnej barwy o dominującej długości fali. Wzajemnymi odpowiednikami określeń percepcyjnych i kolorymetrycznych są:

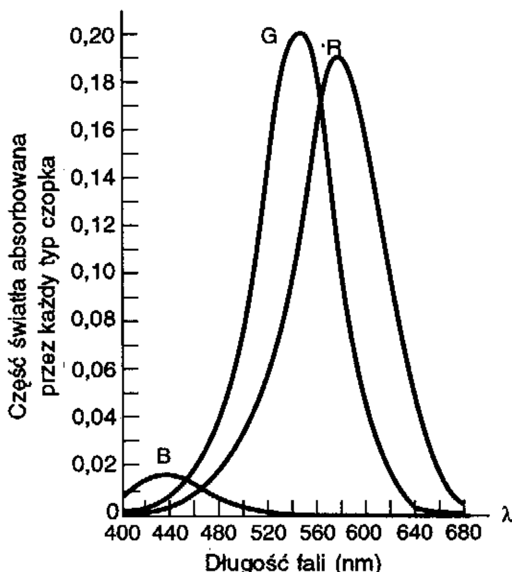
Określenia percepcyjne	*Kolorymetria*
Odcień barwy	Dominująca długość fali
Nasycenie	Czystość pobudzenia
Jasność (obiekty odbijające)	Luminancja
Jaskrawość (obiekty świecące)	Luminancja

W zasadzie światło jest energią elektromagnetyczną zawartą w przedziale fal o długości 400 do 700 nm, która jest postrzegana jako barwy od fioletu przez indygo, niebieski, zielony, żółty i pomarańczowy do czerwonego. Ilość energii dla każdej długości fali jest reprezentowana przez widmowy rozkład energii $P(\lambda)$ (rys. 11.8). Rozkład reprezentuje nieskończenie wiele liczb, po jednej dla każdej długości fali w widzialnym widmie (w praktyce rozkład jest reprezentowany przez dużą liczbę próbek w widmie, zmierzonych spektroradiometrem). Na szczęście możemy opisać efekt wizualny dowolnego rozkładu widmowego w bardziej zwarty sposób za pomocą trójki [dominująca długość fali, czystość pobudzenia, luminancja]. Skutkiem tego jest to, że wiele różnych widmowych rozkładów energii tworzy tę samą barwę: „wyglądają" one tak samo. Dlatego zależność między rozkładem widmowym a barwami jest *typu wiele do jednego.*



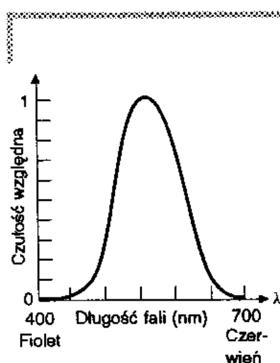
Rys. 11.8. Typowy rozkład energii widmowej $P(\lambda)$ światła

Jak ta dyskusja ma się do plamek luminoforu czerwonego, zielonego i niebieskiego na ekranie kineskopu? I jaki jest związek z *trójpobudzeniową teorią* percepcji barwy, która opiera się na hipotezie, że siatkówka ma trzy rodzaje czujników barwnych (tak zwanych czopków) ze szczytową czułością na światło czerwone, zielone i niebieskie.



Rys. 11.9. Funkcja odpowiedzi w przedziale fal dla każdego z trzech typów czopków siatkówki

Eksperymenty bazujące na tej hipotezie prowadzą do charakterystyk czułości receptorów (rys. 11.9). Wartość szczytowa odpowiedzi dla receptorów barwy niebieskiej wynosi ok. 440 nm; dla zielonej ok. 545 nm; dla czerwonej ok. 580 nm. (Określenia barwa czerwona i zielona są tu nieco mylące, ponieważ długości fal 545 i 580 nm faktycznie są w zakresie barw żółtych.) Krzywe sugerują, że czułość oka na światło niebieskie jest znacznie słabsza niż czułość na światło czerwone albo zielone.

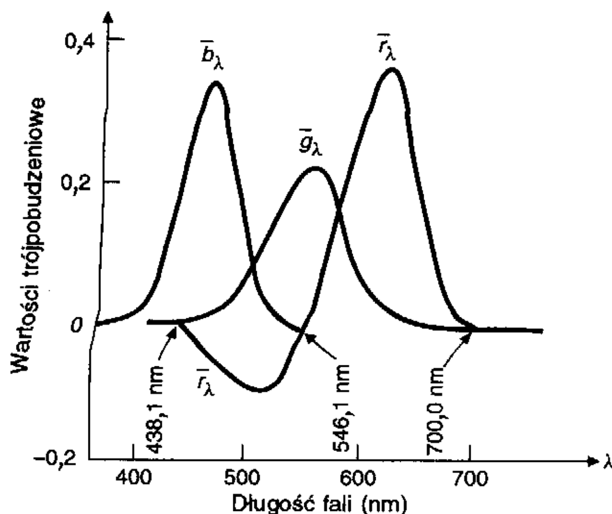


Rys. 11.10. Funkcja odpowiedzi oka na światło o stałej luminancji

Na rysunku 11.10 pokazano czułość oka na światło monochromatyczne o tej samej mocy dla każdej długości fali: nasza szczytowa czułość jest dla światła żółtozielonego o długości fali ok. 550 nm. Eksperyment potwierdza, że ta krzywa jest po prostu sumą trzech krzywych pokazanych na rys. 11.9.

Teoria trójpobudzeniowa jest intuicyjnie atrakcyjna, ponieważ z grubsza odpowiada wyobrażeniu, że barwa może być określona przez dodatnio ważoną sumę barw czerwonej, zielonej i niebieskiej (tak zwane barwy podstawowe). To wyobrażenie jest prawie prawdziwe: trzy składowe trójchromatyczne widmowe z rys. 11.11 pokazują, ile światła czerwonego, zielonego i niebieskiego jest potrzebne dla przeciętnego obserwatora do zrównania ze światłem o barwie widmowej i o stałej mocy, dla wszystkich wartości dominującej długości fali w widmie widzialnym.

Ujemna wartość na rys. 11.11 oznacza, że nie możemy dopasować danej barwy na zasadzie dodawania barw podstawowych. Jeżeli jednak jedna z barw podstawowych zostanie dodana do próbki barwy, to



Rys. 11.11. Funkcje dopasowania barw pokazujące, jaka ilość każdej z trzech barw podstawowych jest potrzebna do dopasowania wszystkich długości fali w widmie widzialnym

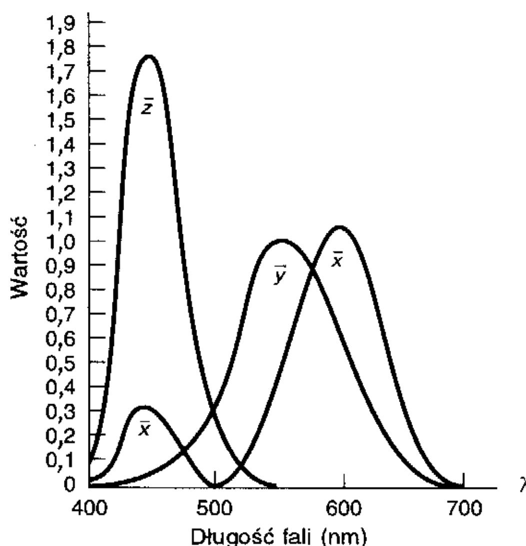
taką próbkę można zrównać z barwą otrzymaną przez zmieszanie dwóch pozostałych barw podstawowych. Dlatego ujemne wartości z rys. 11.11 wskazują, że barwa podstawowa została dodana do dopasowywanej barwy. Istnienie wartości ujemnych nie oznacza, że pojęcie mieszania barw czerwonej, zielonej i niebieskiej w celu uzyskania innych barw jest nieprzydatne; przeciwnie można uzyskać ogromny zakres barw, korzystając z dodatnich wartości barw czerwonej, niebieskiej i zielonej. W przeciwnym przypadku kineskop kolorowy nie mógłby działać! Oznacza to jednak, że niektórych barw nie można uzyskać na zasadzie mieszania RGB i dlatego nie mogą być pokazane na zwykłym kineskopie.

Oko ludzkie może rozróżnić setki tysięcy różnych barw w przestrzeni barw, gdy różne barwy są oceniane na zasadzie porównania przez różnych obserwatorów, którzy stwierdzają, czy dwie barwy są takie same czy różne. Jeżeli barwy różnią się tylko odcieniem, to długość fali między ledwie zauważalnymi różnicami barw zmienia się od ponad 10 nm na krańcach widma do mniej niż 2 nm w okolicach 480 nm (niebieska) i 580 nm (żółta) [BEDF58]. Jednak poza krańcami widma *próg rozróżniania odcieni barwy jest rzędu 4 nm. W sumie można rozróżnić około 128* w pełni nasyconych odcieni barwy.

Oko jest mniej czułe na zmiany odcienia przy mniej nasyconej barwie, natomiast czułość na zmiany nasycenia dla ustalonego odcienia barwy i jasności jest większa na krańcach widma widzialnego, gdzie istnieje około 23 rozróżnialnych kroków.

11.2.2. Wykres chromatyczności CIE

Zrównanie, a tym samym definiowanie światła barwnego za pomocą mieszaniny trzech ustalonych barw podstawowych jest atrakcyjnym podejściem do określenia barwy, ale konieczność korzystania z wag ujemnych występujących na rys. 11.11 jest niewygodna. W 1931 r. Międzynarodowa Komisja Oświetleniowa (CIE) zdefiniowała trzy standardowe barwy podstawowe określane jako **X**, **Y** i **Z** z myślą o zastąpieniu barw czerwonej, zielonej i niebieskiej w procesie dopasowania. Odpowiednie trzy składowe trójchromatyczne widmowe $\bar{x}_\lambda$, $\bar{y}_\lambda$ i $\bar{z}_\lambda$ pokazano na rys. 11.12. Te barwy podstawowe mogą być użyte do określenia wszyst-



Rys. 11.12. Funkcje zrównania barw $\bar{x}_\lambda$, $\bar{y}_\lambda$ i $\bar{z}_\lambda$, dla barw podstawowych **XYZ** w modelu CIE z 1931 r.

kich barw widzialnych przy korzystaniu tylko z dodatnich wag. Z założenia barwa podstawowa **Y** została tak zdefiniowana, żeby składowa trójchromatyczna widmowa $\bar{y}_\lambda$ dokładnie zgadzała się z funkcją czułości oka na światło o stałej luminancji z rys. 11.10. Zauważmy, że $\bar{x}_\lambda$, $\bar{y}_\lambda$ i $\bar{z}_\lambda$ nie są rozkładami widmowymi barw **X**, **Y** i **Z**, tak jak krzywe z rys. 11.11 nie są rozkładami widmowymi dla barw czerwonej, zielonej i niebieskiej.

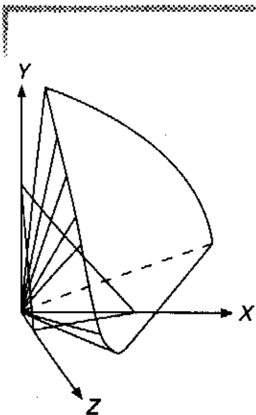
Wielkości składowych podstawowych odpowiadających barwie o rozkładzie widmowym energii $P(\lambda)$ są następujące:

$$X = k\int P(\lambda)\bar{x}_\lambda\, d\lambda,\quad Y = k\int P(\lambda)\bar{y}_\lambda\, d\lambda,\quad Z = k\int P(\lambda)\bar{z}_\lambda\, d\lambda \qquad (11.6)$$

Dla obiektów świecących, takich jak kineskop, k wynosi 680 lm/W. Dla obiektów odbijających k jest zazwyczaj tak wybierane, żeby

dla jasnej bieli wartość Y wynosiła 100; wtedy inne wartości Y będą w przedziale 0 do 100.

Na rysunku 11.13 pokazano stożkową bryłę, która w przestrzeni XYZ zawiera barwy widzialne. Bryła ma wierzchołek w początku układu współrzędnych i zawiera się w dodatnim oktancie i brzeg jej podstawy jest gładką krzywą.



Rys. 11.13. Stożek barw widzialnych w przestrzeni barw CIE, pokazany za pomocą linii wychodzących promieniście ze środka układu współrzędnych. Pokazano płaszczyznę $X + Y + Z = 1$. (Za zgodą Gary *Meyer, Program of Computer* Graphics, Cornell University, 1978)

Niech (X, Y, Z) będą wagami zastosowanymi do barw podstawowych CIE w celu opisania barwy **C** na podstawie równania (11.6). Wtedy $\mathbf{C} = X\mathbf{X} + Y\mathbf{Y} + Z\mathbf{Z}$. Określamy wartości współrzędnych trójchromatycznych (które zależą tylko od dominującej długości fali i nasycenia i są niezależne od strumienia energii świetlnej) wprowadzając normalizujący współczynnik $X + Y + Z$:

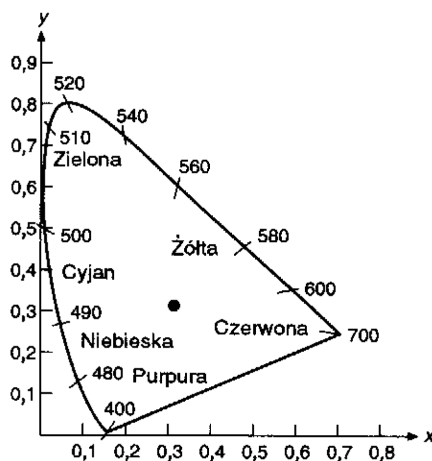
$$x = \frac{X}{(X + Y + Z)}, \quad y = \frac{Y}{(X + Y + Z)}, \quad z = \frac{Z}{(X + Y + Z)} \tag{11.7}$$

Zauważmy, że $x + y + z = 1$, co oznacza, że x, y i z znajdują się na płaszczyźnie $(X + Y + Z = 1)$ z rys. 11.13. Na fotografii 14 pokazano płaszczyznę $X + Y + Z = 1$ jako część płaszczyzny XYZ CIE i rzut *prostokątny płaszczyzny oraz rzut płaszczyzny na płaszczyznę (X, Y)*. Ten ostatni rzut jest właśnie wykresem chromatyczności CIE.

Jeżeli określimy x i y, to z można wyznaczyć jako $z = 1 - x - y$. Nie możemy jednak odtworzyć wartości X, Y i Z z wartości x i y. W tym celu musimy mieć trochę więcej informacji, na ogół wartość Y, która niesie informację o luminancji. Dla danych wartości (x, y, Y) przekształcenie w odpowiednie wartości (X, Y, Z) jest następujące:

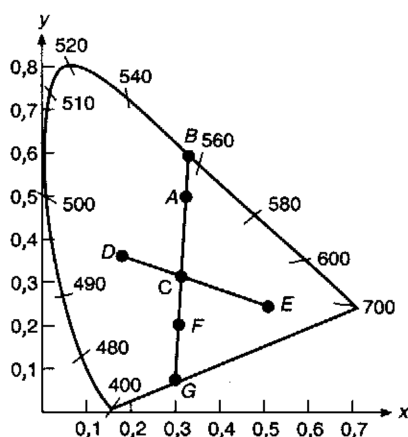
$$X = \frac{x}{y} Y, \quad Y = Y, \quad Z = \frac{1 - x - y}{y} Y \tag{11.8}$$

Wartości współrzędnych trójchromatycznych zależą tylko od dominującej długości fali i nasycenia i nie zależą od strumienia światła. Wykreślając wartości x i y dla wszystkich barw widzialnych otrzymujemy wykres chromatyczności pokazany na rys. 11.14, który jest rzutem na płaszczyznę (X, Y) płaszczyzny $(X + Y + Z = 1)$ z rys. 11.13. Wnętrze i brzeg obszaru o kształcie podkówki reprezentują wszystkie barwy widzialne. (Wszystkie postrzegane barwy o tym samym odcieniu nasycenia, ale o różnych luminancjach są odwzorowywane na ten sam punkt w tym obszarze.) 100-procentowo czyste barwy widmowe leżą na zakrzywionej części brzegu obszaru. Standardowe światło białe, w zamyśle przybliżające światło słoneczne, jest formalnie określone jako *iluminant C* oznaczony kropką w środku obszaru. Leży ona blisko punktu, dla którego $x = y = z = 1/3$ (ale nie w tym punkcie). Iluminant C był zdefiniowany przez określenie rozkładu widmowego mocy, bliskiego do światła dziennego dla skorelowanej temperatury barwowej 674 K.



Rys. 11.14. Wykres chromatyczności CIE. Długości fal na wykresie są podane w nanometrach. Kropka oznacza położenie iluminantu C

Użyteczność diagramu chromatyczności CIE jest wieloraka. Umożliwia on mierzenie dominującej długości fali i czystości pobudzenia dowolnej barwy na zasadzie opisania barwy jako mieszaniny trzech składowych podstawowych CIE. Załóżmy teraz, że dana barwa jest reprezentowana przez punkt *A* na rys. 11.15. Gdy dodamy dwie barwy do siebie, wówczas nowa barwa leży na odcinku łączącym dwie dodawane barwy. O barwie *A* można myśleć jak o mieszaninie „standardowego" światła białego (iluminant C) i czystego światła o barwie widmowej w punkcie *B*. A więc *B* definiuje dominującą długość fali. Stosunek



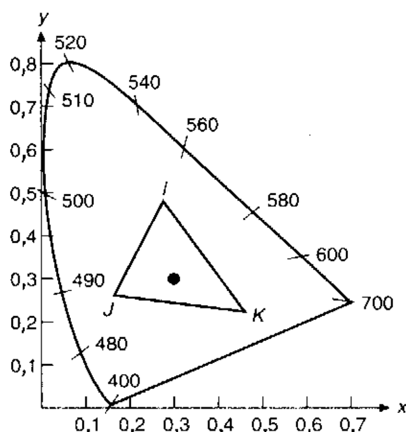
Rys. 11.15. Barwy na wykresie chromatyczności. Dominującą długością fali dla barwy *A* jest długość fali dla barwy *B*. Barwy *D* i *E* są komplementarne. Dominująca długość fali barwy *F* jest zdefiniowana jako uzupełnienie dominującej długości fali dla barwy *A*

długości AC do długości BC wyrażony w procentach jest czystością pobudzenia barwy A. Im A jest bliżej C, tym A zawiera więcej światła białego i jest mniej czysta.

Wykres chromatyczności eliminuje luminancję, więc nie opisuje wrażeń barwnych, które są zależne od luminancji. Na przykład, nie pojawia się barwa brązowa, która jest barwą pomarańczowoczerwoną na wykresie chromatyczności o bardzo małej luminancji względem otaczającej powierzchni. Trzeba więc pamiętać, że wykres chromatyczności to nie jest pełna paleta barw. Jest nieskończenie wiele płaszczyzn w przestrzeni (X, Y, Z), z których każda może być zrzutowana na wykres chromatyczności i każda w tym procesie traci informację o luminancji. Barwy występujące na każdej takiej płaszczyźnie są różne.

Barwy dopełniające są to takie barwy, które po zmieszaniu dają światło białe (na przykład barwy D i E na rys. 11.15). Niektóre barwy (takie jak F na rys. 11.15) nie mogą być zdefiniowane przez dominującą długość fali i dlatego są określane jako *niespektralne*. W tym przypadku mówi się, że dominująca długość fali jest dopełnieniem dla długości fali, dla której odcinek przechodzący przez F i C przecina część podkówki krzywej w punkcie B i jest oznaczana przez „c" (tutaj ok. 555 nm c). Czystość pobudzenia jest zdefiniowana jako stosunek długości (tutaj CF do CG). Barwy, które muszą być określone przez dopełniającą długość fali, to purpury i magenty; występują one w dolnej części wykresu chromatyczności.

Inne zastosowanie wykresu chromatyczności CIE, to definiowanie *gamy barw* albo zakresu barw, które pokazują efekt dodawania barw. Możemy dodać dwie dowolne barwy do siebie, na przykład I i J na



Rys. 11.16. Mieszanie barw. Możemy utworzyć wszystkie barwy na odcinku IJ mieszając barwy I i J; możemy utworzyć wszystkie barwy w trójkącie IJK mieszając barwy I, J i K

rys. 11.16 w celu uzyskania dowolnej barwy leżącej na odcinku łączącym te barwy, zmieniając względne ilości dwóch dodawanych barw. Trzecia barwa K (rys. 11.16) może być użyta z różnymi mieszaninami barw I i J w celu uzyskania gamy wszystkich barw w trójkącie IJK, znowu na zasadzie mieszania względnych ilości. Kształt wykresu pokazuje, dlaczego nie można mieszając addytywnie barw czerwonej, niebieskiej i zielonej uzyskać wszystkich barw: żaden trójkąt, którego wierzchołki leżą wewnątrz widzialnego obszaru, nie może w pełni pokryć całego widzialnego obszaru.

Wykres chromatyczności jest również używany do porównywania gam barw dostępnych na różnych monitorach kolorowych i urządzeniach drukujących. Na fotografii 15 pokazano gamy dla kolorowego monitora telewizyjnego, filmu i druku. Porównanie niedużej wielkości gamy dla druku z gamą dla monitora sugeruje, że jeżeli obrazy oglądane pierwotnie na monitorze mają być wiernie reprodukowane w druku, to dla monitora trzeba użyć zredukowanej gamy barw. W przeciwnym przypadku dokładna reprodukcja nie będzie możliwa. Jeżeli jednak celem jest wykonanie dobrze wyglądającej reprodukcji, która nie musi być dokładną reprodukcją, to niewielkie różnice w gamie barw są mniej istotne. Dyskusję na temat kompresji gamy barw można znaleźć w pracy [HALL89].

Po tym wprowadzeniu podstawowych wiadomości o barwie wracamy do zagadnienia barwy w grafice komputerowej.

11.3. Modele barw dla grafiki rastrowej

Model barw jest to określony trójwymiarowy system współrzędnych barw wraz z widzialnym podzbiorem, w którym leżą wszystkie barwy z określonej gamy barw. Na przykład model barw RGB jest sześcianem jednostkowym będącym podzbiorem trójwymiarowego układu współrzędnych kartezjańskich.

Model barw ma umożliwić wygodny wybór barw wewnątrz jakiejś gamy barw. Przede wszystkim jesteśmy zainteresowani gamą barw dla kolorowych monitorów kineskopowych, określaną przez barwy podstawowe RGB (czerwona, zielona i niebieska) jak na fot. 15. Jak widać na tej fotografii, gama barw jest podzbiorem wszystkich barw widzialnych. Dlatego model barw nie może być użyty do specyfikowania wszystkich barw widzialnych.

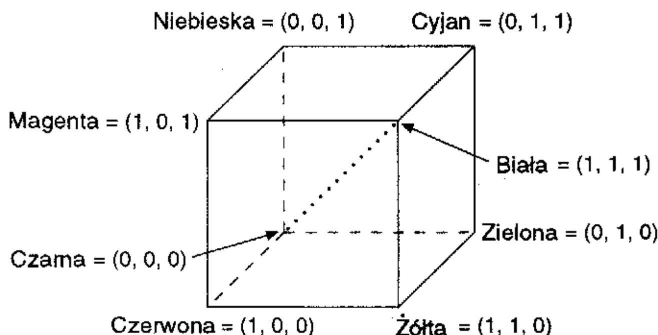
Trzema modelami barw związanymi ze sprzętem są: model RGB wykorzystywany w monitorach kineskopowych; model YIQ wykorzystywany w systemach telewizji kolorowej; model CMY (cyjan, magenta, żółty) wykorzystywany w niektórych urządzeniach drukujących. Niestety, żaden z tych modeli nie jest szczególnie łatwy w użytkowaniu, ponie-

waż nie są one związane z intuicyjnymi pojęciami odcienia barwy, nasycenia i jaskrawości. Dlatego opracowano inną klasę modeli z intencją zapewnienia łatwości użytkowania. Kilka takich modeli opisano w pracach [GSPC79; JOBL78; MEYE80; SMIT78]. Omówimy tylko jeden z nich, a mianowicie model HSV (czasami określany jako HSB).

Dla każdego modelu są określone sposoby reprezentacji w innym modelu. Pokażemy, jak można dokonać konwersji między RGB oraz HSV i CMY oraz między RGB i YIQ. Dodatkowe algorytmy konwersji można znaleźć w pracy [FOLE90].

11.3.1. Model barw RGB

Model barw RGB stosowany w kolorowych monitorach kineskopowych i w barwnej grafice rastrowej wykorzystuje układ współrzędnych kartezjańskich. Barwy podstawowe R, G, B są mieszane *addytywnie*; oznacza to, że indywidualne udziały każdej barwy podstawowej są sumowane razem w celu uzyskania wyniku, tak jak to sugeruje fot. 16. Przedmiotem zainteresowania jest sześcian jednostkowy pokazany na rys. 11.17. Główna przekątna sześcianu, z równym udziałem każdej barwy podstawowej, reprezentuje poziomy szarości: barwa czarna ma współrzędne (0, 0, 0), a biała (1, 1, 1).



Rys. 11.17. Sześcian RGB. Odcienie szarości są na głównej przekątnej (linia kropkowana)

Gama barw pokryta przez model RGB jest zdefiniowana przez współrzędne trójchromatyczne luminoforów kineskopu. Dwa kineskopy z różnymi luminoforami będą miały różne gamy barw. W celu dokonania konwersji barw określonych w gamie jednego kineskopu na gamę innego kineskopu korzystamy z przekształceń z przestrzeni barw RGB każdego monitora w przestrzeń barw (X, Y, Z). Szczegóły podano w pracy [FOLE90].

11.3.2. Model barw CMY

Barwy cyjan, magenta i żółta są barwami dopełniającymi: odpowiednio dla barw czerwonej, zielonej i niebieskiej. Barwy filtrów używanych w celu odjęcia barwy od światła białego są określane jako podstawowe *barwy subtraktywne*. Podzbiór układu współrzędnych kartezjańskich dla modelu CMY jest taki sam jak dla modelu RGB z wyjątkiem tego, że barwa biała (pełne światło) znajduje się w początku układu współrzędnych (a nie barwa czarna – brak światła). Barwy są określane przez to, co zostało usunięte albo odjęte od światła białego, a nie przez to co zostało dodane do czerni.

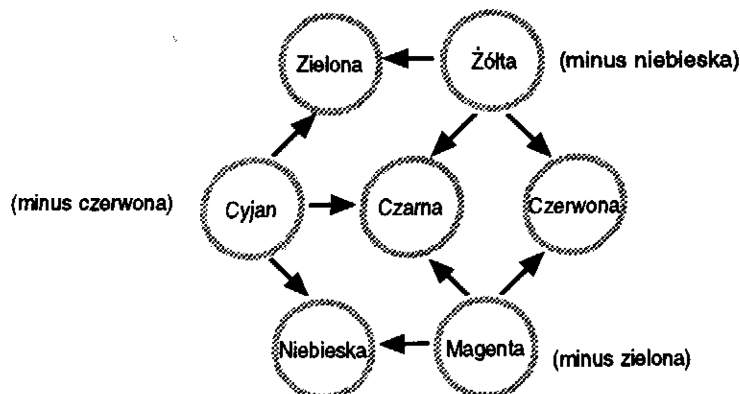
Znajomość modelu CMY jest ważna wówczas, gdy mamy do czynienia z urządzeniami tworzącymi trwałą kopię, które nanoszą barwne pigmenty na papier, np. jak plotery elektrostatyczne albo atramentowe. Gdy powierzchnia jest pokryta atramentem o barwie cyjan, wówczas czerwone światło nie jest odbijane od powierzchni. Cyjan odejmuje barwę czerwoną od odbijanego światła białego, które samo jest sumą barw czerwonej, zielonej i niebieskiej. Dlatego w kategoriach mieszania addytywnego cyjan jest barwą otrzymaną w wyniku odjęcia barwy czerwonej od białej – jest to barwa niebieska plus zielona. Podobnie magenta absorbuje barwę zieloną i jest sumą barw czerwonej i niebieskiej; barwa żółta absorbuje barwę niebieską, jest więc określona przez sumę barw czerwonej i zielonej. Powierzchnia pokryta cyjanem i barwą żółtą absorbuje barwy czerwoną oraz niebieską i jest odbijana tylko barwa zielona z białego światła oświetlającego. Powierzchnia pokryta barwami cyjan, żółta i magenta absorbuje barwy czerwoną, zieloną oraz niebieską i wobec tego jest barwy czarnej. Te zależności są ujęte w postaci diagramu na rys. 11.18; można je obejrzeć również na fot. 17. Są one reprezentowane za pomocą następujących równań:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{11.9}$$

Jednostkowy wektor kolumnowy jest reprezentacją barwy białej dla modelu RGB i czarnej dla modelu CMY.

Konwersja z modelu RGB na CMY ma postać

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix} \tag{11.10}$$



Rys. 11.18. Subtraktywne barwy podstawowe (cyjan, magenta, żółta) i ich mieszaniny. Na przykład cyjan i żółta dają zieleń

Te bezpośrednie przekształcenia mogą być użyte do konwersji ośmiu barw, które można uzyskać za pomocą dwójkowych kombinacji barw czerwonej, zielonej i niebieskiej na osiem barw możliwych do uzyskania dla dwójkowych kombinacji barw cyjan, magenta i żółta. Ta konwersja może być stosowana przy korzystaniu z kolorowych drukarek atramentowych i kserograficznych.

Inny model barw CMYK używa jako czwartej barwy barwy czarnej (oznaczanej literą K). Model CMYK jest używany w czterobarwnych urządzeniach drukarskich wykorzystujących proces czterobarwny i w niektórych urządzeniach tworzących trwałe kopie. Dla danej specyfikacji CMY barwa czarna jest używana zamiast równych wartości C, M, Y zgodnie z następującymi zależnościami:

$$\begin{aligned} K &= \min(C, M, Y) \\ C &= C - K \\ M &= M - K \\ Y &= Y - K \end{aligned} \tag{11.11}$$

Problem jest omawiany szerzej w pracy [STON88].

11.3.3. Model barw YIQ

Model YIQ jest używany w Stanach Zjednoczonych w komercyjnej telewizji kolorowej. YIQ jest wynikiem przekodowania RGB ze względu na efektywność transmisji i ze względu na zgodność z telewizją czarno-białą. Przekodowany sygnał jest transmitowany zgodnie ze standardem NTSC [PRIT77].

W modelu YIQ składowa Y nie jest żółta, lecz oznacza luminancję; jest zdefiniowana tak, żeby odpowiadała składowej podstawowej Y w modelu CIE. W telewizji czarno-białej jest pokazywana tylko składowa Y telewizyjnego sygnału barwnego: barwa jest zakodowana w I oraz Q. Model YIQ wykorzystuje układ współrzędnych kartezjańskich 3D, przy czym widzialny podzbiór jest wypukłym wielościanem, który odwzorowuje się na sześcian RGB.

Odwzorowanie RGB-YIQ jest zdefiniowane w następujący sposób:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}596 & -0{,}275 & -0{,}321 \\ 0{,}212 & -0{,}528 & 0{,}311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{11.12}$$

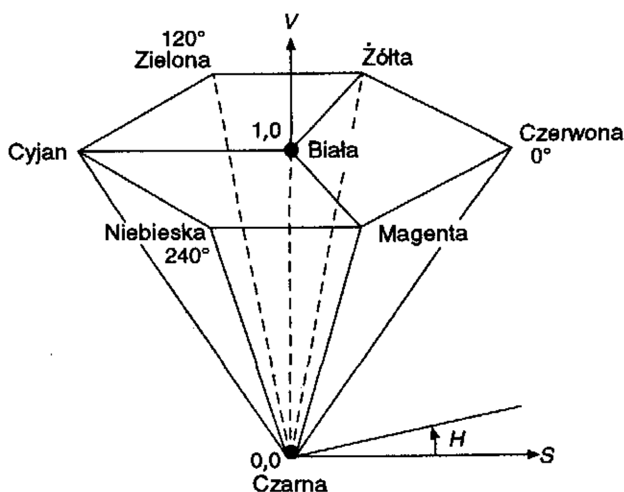
Wielkości w pierwszym wierszu odzwierciedlają znaczny udział czerwieni i zieleni i względnie mały udział niebieskiego w jaskrawości. Macierz odwrotna do macierzy RGB-YIQ jest wykorzystywana do konwersji YIQ-RGB.

Określanie barw w modelu YIQ rozwiązuje potencjalny problem dotyczący materiału, który został przygotowany dla telewizji: dwie różne barwy pokazane obok siebie na monitorze kolorowym będą wydawały się różne, ale po konwersji na YIQ i wyświetleniu na monitorze monochromatycznym mogą wyglądać identycznie. Możemy unikać tego problemu określając te dwie barwy z różnymi wartościami Y w przestrzeni modelu YIQ (to jest regulując tylko wartości Y w celu ich rozróżnienia).

W modelu YIQ są wykorzystane dwie użyteczne właściwości systemu wzrokowego. Po pierwsze, system jest bardziej czuły na zmiany luminancji niż na zmiany odcienia barwy albo nasycenia; to znaczy, nasza zdolność do przestrzennego dyskryminowania informacji barwnej jest słabsza niż nasza zdolność do przestrzennego dyskryminowania informacji monochromatycznej. Ta obserwacja sugeruje, że do reprezentowania wartości Y powinna być używana większa liczba bitów pasma niż do reprezentowania I oraz Q, tak żeby uzyskać większą rozdzielczość dla Y. Po drugie, obiekty, które pokrywają wyjątkowo małą część pola wizualizacji, wytwarzają ograniczone wrażenia barwne, które mogą być wystarczająco określane za pomocą jednego, a nie dwóch parametrów barwy. Ten fakt sugeruje, że albo I, albo Q może mieć mniejsze pasmo niż ten drugi parametr. W systemie NTSC przy kodowaniu sygnału YIQ na sygnał emitowany wykorzystuje się tę właściwość w celu maksymalizowania ilości transmitowanej informacji w ustalonym paśmie: składowej Y przypisuje się 4 MHz, składowej I 1,5 MHz i składowej Q 0,6 MHz. Dalszą dyskusję na temat modelu YIQ można znaleźć w pracach [SMIT78; PRIT77].

11.3.4. Model barw HSV

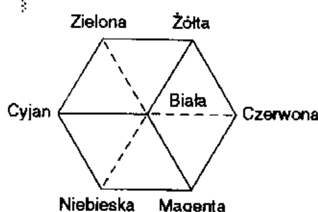
Modele RGB, CMY, YIQ są ukierunkowane sprzętowo. Model HSV (odcień barwy, nasycenie i wartość) Smitha [SMIT78] (określany również jako model HSB, przy czym *B* oznacza jaskrawość) jest zorientowany na użytkownika i wykorzystuje intuicyjne wrażenia modelu artysty, a więc tinty, tony i cienie. Układ współrzędnych jest układem cylindrycznym, a podzbiór przestrzeni, w którym jest zdefiniowany model, stanowi ostrosłup sześciokątny (rys. 11.19). Podstawa ostrosłupa ma wartość $V = 1$ i są zawarte na niej względnie jasne barwy. Nie wszystkie jednak barwy w płaszczyźnie $V = 1$ mają percepcyjnie taką samą jasność.



Rys. 11.19. Ostrosłup modelu barw HSV. Płaszczyzna $V = 1$ zawiera w odpowiednich regionach płaszczyzny $R = 1$, $G = 1$ i $B = 1$ z modelu RGB

Odcień barwy *H* jest mierzony za pomocą kąta wokół osi pionowej – barwie czerwonej odpowiada kąt 0°, barwie zielonej kąt 120° itd. (rys. 11.19). Barwy dopełniające w ostrosłupie HSV znajdują się naprzeciwko siebie w odległości 180°. Wartość *S* jest ułamkiem zmieniającym się od 0 na osi ostrosłupa do 1 na jego bokach. Nasycenie jest mierzone względem gamy barw reprezentowanej przez model, która jest oczywiście podzbiorem całego wykresu chromatyczności CIE. Stąd 100% nasycenia w modelu jest to mniej niż 100% czystości pobudzenia.

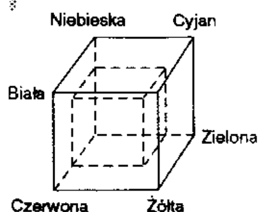
Wierzchołek ostrosłupa znajduje się w początku układu współrzędnych, a największa wartość parametru *V* wynosi 1. W wierzchołku znajduje się barwa czarna, a wartość współrzędnej *V* wynosi 0. W tym



Rys. 11.20. Sześcian modelu RGB widziany wzdłuż głównej przekątnej. Widzialne krawędzie sześcianu są narysowane linią ciągłą; krawędzie niewidoczne – linią przerywaną

punkcie wartości H i S są nieistotne. W punkcie $S = 0$, $V = 1$ jest barwa biała. Pośrednie wartości V dla $S = 0$ (na osi ostrosłupa) odpowiadają poziomom szarości. Dla $S = 0$ wartość H jest nieistotna (zgodnie z konwencją mówi się, że jest niezdefiniowana). Gdy S nie jest zerem, wartość H staje się istotna. Na przykład czysta barwa czerwona ma współrzędne $H = 0$, $S = 1$, $V = 1$. Każda barwa, dla której $V = 1$, $S = 1$, jest podobna do czystego pigmentu stosowanego przez artystę jako punkt wyjścia przy mieszaniu barw. Dodanie białego pigmentu odpowiada zmniejszeniu S (bez zmiany V). Cienie tworzymy utrzymując $S = 1$ i zmniejszając V. Tony tworzymy zmniejszając S i V. Oczywiście zmiana H odpowiada wybraniu początkowego czystego pigmentu. A więc H, S i V odpowiadają koncepcji modelu barw artysty i nie są dokładnie takie same jak pojęcia wprowadzone w p. 11.2.

Podstawa ostrosłupa HSV odpowiada rzutowi, jaki się obserwuje patrząc wzdłuż głównej przekątnej sześcianu barw RGB od strony barwy białej w kierunku barwy czarnej (rys. 11.20). Sześcian RGB ma podsześciany (rys. 11.21). Każdy podsześcian obserwowany wzdłuż głównej przekątnej wygląda jak sześciokąt z rys. 11.20, z tą różnicą, że jest mniejszy. Każda płaszczyzna stałego V w przestrzeni HSV odpowiada takiemu widokowi podsześcianu w przestrzeni RGB. Główna przekątna przestrzeni RGB staje się osią V przestrzeni HSV. Dlatego intuicyjnie możemy zauważyć odpowiedniość między RGB i HSV. Algorytmy z programów 11.1 i 11.2 definiują dokładną odpowiedniość podając zasady konwersji między tymi modelami.



Rys. 11.21. Sześcian RGB i podsześcian

Program 11.1 Algorytm konwersji z przestrzeni RGB do przestrzeni HSV

```
void  RGB_To_HSV(float r, float g, float b, float *h, float *s, float *v)
{
/* Dane: r,g,b, każda w przedziale [0,1].
   Szukane: h w [0,360], s i v w [0,1] z wyjątkiem s=0, kiedy H=UNDEFINED, co
   jest pewną stałą zdefiniowaną za pomocą wartości spoza przedziału [0,360] */
    float max, min, delta;

    max = MAX(r, g, b);
    min = MIN(r, g, b);
    *v = max;                          /* jest to wartość v */
    /* Obliczanie wartości nasycenia s */
    if (max != 0.0)
        *s = (max - min) / max;        /* s jest nasyceniem */
    else
        *s = 0.0;                      /* Nasycenie jest równe 0, jeżeli wartości
                                          czerwona, zielona i niebieska są zerami */

    if (*s == 0.0) {
        *h = UNDEFINED;
```

```
      return;
   }
   /* Przypadek chromatyczny: Nasycenie nie jest równe 0, więc określa się odcień
     barwy */
   delta = max – min;
   if (r == max)  /* Wynikowa barwa jest między magentą a cyjanem */
      *h = (g – b) / delta; /* Wynikowa barwa jest między cyjanem a magentą */
   else if (g == max)
      *h = 2.0 + (b – r) / delta; /* Wynikowa barwa jest między cyjanem a żółtym */
   else if (b == max)
      *h = 4.0 + (r – g) / delta;
   *h *= 60.0;                    /* Zamiana odcienia barwy na stopnie */
   if (*h < 0.0)
      *h += 360.0;      /* Trzeba sprawdzić, czy odcień barwy nie jest ujemny */
}
```

Program 11.2 Algorytm konwersji z przestrzeni barw HSV do przestrzeni RGB

```
void  HSV_To_RGB(float *r, float *g, float *b, float h, float s, float v)
{
/* Dane: h w [0,360] albo UNDEFINED, s i v w [0,1].
   Szukane: r, g, b, każde w [0,1] */
   float f, p, q, t;
   int   i;

   if (s == 0.0) {              /* Barwa leży na osi czarno-białej */
      if (h != UNDEFINED)  /* Barwa achromatyczna: nie ma nasycenia */
         Error();          /* Zgodnie z konwencją, gdy s = 0 i h ma wartość to jest błąd */
         return;
      }
      *r = v;
      *g = v;
      *b = v;
         return;
      }
                                /* Barwa chromatyczna: s ≠ 0, a więc jest odcień barwy */
      if (h == 360.0)           /* 360° jest równoważne 0° */
         h = 0.0;
      h /= 60.0;                /* h jest teraz w [0,6] */
      i = floor(h);             /* floor zwraca największą wartość całkowitą ≤ h */
      f = h – i;                /* f jest częścią ułamkową h */
      p = v * (1 – s);
      q = v * (1 – s * f);
      t = v * (1 – s * (1 – f));

      switch (i) {
      case 0:
         *r = v;
         *g = t;
         *b = p;
```

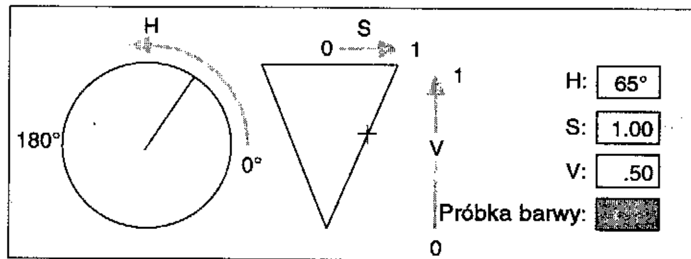
```
            break;

        case 1:
            *r = q;
            *g = v;
            *b = p;
            break;

        case 2:
            *r = p;
            *g = v;
            *b = t;
            break;

        case 3:
            *r = p;
            *g = q;
            *b = v;
            break;

        case 4:
            *r = t;
            *g = p;
            *b = v;
            break;

        case 5:
            *r = v;
            *g = p;
            *b = q;
            break;
        }
    }
```

11.3.5. Interakcyjny wybór barwy

Wiele programów użytkowych umożliwia określenie barwy powierzchni, linii, tekstu itd. Jeżeli jest dostępny tylko niewielki zestaw barw, to właściwym rozwiązaniem jest wybór za pomocą menu spośród dostępnych barw. Ale co zrobić, jeżeli zestaw barw jest większy, niż może być wyświelony w rozsądnym menu?

Podstawowe możliwości, to stosowanie nazw albo określanie wartości liczbowych współrzędnych barwy w przestrzeni barw (albo na zasadzie pisania wartości, albo wybierania za pomocą suwaków z podziałką); można również zapewnić bezpośrednią współpracę z wizualną reprezentacją przestrzeni barw. Podawanie nazw jest w zasadzie nie-

efektywne, ponieważ jest niejednoznaczne i subiektywne (jasnogranatowy z dodatkiem zielonego) i jest również antytezą interakcji graficznej. W pracy [BERK82] jest opisany system CNS dość dobrze zdefiniowanych nazw barw, w którym wykorzystuje się takie określenia jak zielonkawożółty, zielonożółty i żółtawozielony w celu rozróżnienia trzech odcieni między zielonym a żółtym. W eksperymencie użytkownicy CNS określali barwy bardziej precyzyjnie niż użytkownicy, którzy wprowadzali wartości liczbowe współrzędnych w przestrzeni RGB albo HSV.



Rys. 11.22. Wygodny sposób specyfikowania barw w przestrzeni HSV. Nasycenie i wartość są pokazane za pomocą kursora w polu trójkąta, a odcień barwy za pomocą promienia w kole. Użytkownik może obracać promień i poruszać kursorem, co w efekcie powoduje zmianę odczytu numerycznego. Alternatywnie użytkownik może wpisać nową wartość, co powoduje zmianę wskaźników. Można dodać suwaki z podziałkami dla *H, S, V*, co zapewniłoby użytkownikowi możliwość dokładnego sterowania pojedynczym parametrem na raz, bez konieczności wpisywania wartości

Współrzędne można określić za pomocą suwaków z podziałkami, korzystając z dowolnego modelu barw. Jeżeli użytkownik rozumie, jak każdy parametr wpływa na barwę, to ta metoda działa dobrze. Prawdopodobnie najlepszą metodą interakcyjnego określania wartości liczbowych współrzędnych jest zapewnienie użytkownikowi współpracy interakcyjnej bezpośrednio z reprezentacją przestrzeni barw (rys. 11.22). W kole (reprezentującym płaszczyznę $V = 1$) można obracać promień i określać, który przekrój bryły HSV jest wyświetlany w trójkącie. W trójkącie może poruszać się kursor i określać nasycenie i wartość. W trakcie przesuwania promienia albo kursora zmienia się wyświetlana wartość liczbowa. Gdy użytkownik wpisuje nowe wartości bezpośrednio do pola, wówczas promień i kursor zmieniają położenie. W polu barwy jest pokazana bieżąca barwa. Ludzka percepcja barwy zależy jednak od otaczających barw i od wielkości barwnych obszarów; dlatego barwa oglądana w polu wyświetlania wybranej barwy będzie prawdopodobnie różniła się od barwy odbieranej w końcowym obrazie. Jest więc ważne, żeby użytkownik widział również rzeczywisty obraz w czasie doboru barw.

11.3.6. Interpolacja w przestrzeni barw

Interpolacja barw jest potrzebna przynajmniej w trzech sytuacjach: przy cieniowaniu Gourauda (p. 14.2.4), przy eliminacji zakłóceń (p. 3.14) i przy łączeniu dwóch obrazów ze sobą, na przykład w sekwencji: zanikanie jednego obrazu i pojawianie się następnego. Wynik interpolacji zależy od modelu barw, w którym następuje interpolacja; musimy więc zwracać uwagę na wybór odpowiedniego modelu.

Jeżeli konwersja z jednego modelu do drugiego przekształca odcinek (reprezentujący ścieżkę interpolacji) w jednym modelu w odcinek w innym modelu, to wyniki interpolacji liniowej w obu modelach będą takie same. Taka sytuacja występuje dla modeli RGB, CMY, YIQ, CIE i XYZ, które są powiązane prostymi przekształceniami afinicznymi. Odcinek w modelu RGB w ogólnym przypadku nie przekształca się jednak w odcinek w modelu HSV. Na fotografii 19 pokazano wyniki liniowej interpolacji między tymi samymi dwiema barwami w przestrzeniach barw HSV, RGB i YIQ. Rozważmy interpolację między barwami czerwoną i zieloną. W RGB barwa czerwona = (1, 0, 0), a zielona = (0, 1, 0). Wynik interpolacji (dla wygody z wagami 0,5) jest (0,5, 0,5, 0). Stosując algorytm RGB_To_HSV (program 11.1) do tego wyniku otrzymujemy (60°, 1, 0,5). Teraz reprezentując barwy czerwoną i zieloną w HSV mamy (0°, 1, 1) i (120°, 1, 1). Interpolując z równymi wagami w HSV otrzymujemy (60°, 1, 1); wartość ta różni się o 0,5 w stosunku do takiej samej interpolacji w RGB.

Jako drugi przykład rozważmy interpolację barw czerwonej i cyjan w modelach RGB i HSV. W RGB zaczynamy od (1, 0, 0) i (0, 1, 1) i w wyniku interpolacji otrzymujemy (0,5, 0,5, 0,5), co w HSV jest reprezentowane przez (UNDEFINED, 0, 0,5). W HSV barwy czerwona i cyjan są (0°, 1, 1) i (180°, 1, 1). W wyniku interpolacji otrzymujemy (90°, 1, 1); został wprowadzony nowy odcień barwy o maksymalnej wartości i nasyceniu, podczas gdy poprawny wynik połączenia jednakowych ilości barw dopełniających określa barwę szarą. Tutaj ponownie interpolacja, a potem przekształcenie dają różne wyniki niż przy przekształceniu i późniejszej interpolacji.

Przy cieniowaniu Gourauda może być wykorzystany dowolny model, ponieważ dwie interpolowane barwy są zazwyczaj tak blisko siebie, że ścieżki interpolacji między barwami są również blisko siebie. Gdy dwa obrazy są ze sobą łączone – tak jak przy sekwencji stopniowego wygaszania i stopniowego pojawiania się drugiego obrazu albo przy eliminacji zakłóceń – barwy mogą być różne i właściwym modelem jest model addytywny, np. RGB. Jeżeli jednak celem jest interpolacja między dwiema barwami o ustalonym odcieniu (albo nasyceniu) i utrzyma-

nie stałej barwy (albo nasycenia) dla wszystkich interpolowanych barw, to lepszy jest model HSV.

11.4. Wykorzystanie barwy w grafice komputerowej

Barwę wykorzystujemy z różnych powodów: w celu uzyskania wrażeń estetycznych, w celu uzyskania tonalności albo nastroju, w celu uzyskania wrażenia realizmu, w celu identyfikacji powiązanych ze sobą obszarów i w celu kodowania. Przy zachowaniu odpowiedniej ostrożności barwa może być efektywnie wykorzystana do każdego z tych celów. Ponadto użytkownicy mają tendencję do wykorzystywania barwy nawet wówczas, gdy nie ma jakościowych przesłanek, że dzięki temu uzyska się jakąś poprawę.

Przy niestarannym użyciu barwy obraz może stać się mniej użyteczny albo mniej atrakcyjny niż w przypadku odpowiedniej prezentacji monochromatycznej. W pewnym eksperymencie wprowadzenie nieistotnej barwy zredukowało wydajność użytkownika do około jednej trzeciej tego, co było przed zastosowaniem barwy [KREB79]. Barwa powinna być stosowana ostrożnie. Każde dekoracyjne użycie barwy powinno być podporządkowane celom funkcjonalnym, tak żeby barwa nie powodowała zmiany interpretacji pierwotnego znaczenia. Dlatego użycie barwy, podobnie jak i wszystkie inne aspekty interfejsu użytkownik-komputer, musi być testowane przez rzeczywistych użytkowników, tak żeby można było zidentyfikować problem i zaradzić mu. Oczywiście ludzie mogą mieć różne preferencje i w praktyce często korzysta się z wartości domniemanych wybranych na zasadzie reguł używania barwy z możliwością zmiany barwy przez użytkownika. Ostrożne podejście do wyboru barwy polega na zaprojektowaniu najpierw obrazu monochromatycznego po to, żeby mieć pewność, że użycie barwy jest czysto redundancyjne. Ta opcja umożliwia uniknięcie problemów dla użytkowników, z wadami widzenia barw, a także oznacza, że program użytkowy może być wykorzystywany na monitorze monochromatycznym. W systemach zarządzania okien barwy są wybierane bardzo ostrożnie *(fot. 9 i 10).* Barwa nie jest używana jako unikatowy kod stanu przycisku, wybranej pozycji menu itd.

Napisano wiele książek na temat stosowania barwy w celach estetycznych, w tym pracę [BIRR61]; podamy tutaj jedynie kilka prostych reguł, które umożliwią uzyskanie harmonii barw. Podstawową regułą dla estetyki barw jest wybieranie barw zgodnie z jakąś metodą, na ogół wzdłuż gładkiej ścieżki w modelu barw, albo ograniczanie się do barw leżących na płaszczyznach lub ostrosłupach sześciokątnych w przestrzeni barw. Ta wskazówka mogłaby oznaczać stosowanie barw o stałej

jasności albo wartości. Powinny być wybierane barwy znajdujące się w percepcyjnie równych odległościach, co nie jest tym samym co jednakowe przyrosty współrzędnej i może być trudne do implementacji. Przypomnijmy, że dwie liniowe interpolacje (takie jak cieniowanie Gourauda) między dwiema barwami dają różne wyniki w różnych przestrzeniach barw (por. zadanie 11.6 i fot. 19).

Przypadkowy wybór różnych odcieni barw i nasyceń zazwyczaj daje krzykliwy efekt. Alvy Ray Smith wykonał nieformalny eksperyment, w którym siatka 16 × 16 została wypełniona losowo generowanymi barwami. Nie było zaskoczeniem, że siatka wyglądała nieatrakcyjnie. Sortowanie 256 barw według wartości *H*, *S* i *V* i ponowne wyświetlanie ich na siatce w nowym porządku poprawiło w istotny sposób wygląd siatki.

Bardziej specyficzna odmiana tej reguły sugeruje, że jeżeli wykres zawiera tylko kilka barw, to jako tło powinna być użyta jedna z barw dopełniających. W obrazie wykorzystującym wiele różnych barw jako tło powinna być używana barwa neutralna (szara), ponieważ dobrze harmonizuje i nie rzuca się w oczy. Jeżeli dwie sąsiednie barwy niezbyt harmonizują ze sobą, to można użyć cienkiej czarnej linii separującej je. Takie użycie obrzeży jest również bardzo efektywne dla achromatycznego kanału wzrokowego, ponieważ czarna obwódka ułatwia wykrywanie kształtów. Niektóre z tych reguł są zakodowane w systemie ACE (A Color Expert); jest to system ekspertowy dla wyboru barw dla interfejsów użytkownika [MEIE88]. Na ogół dobrze jest minimalizować liczbę używanych różnych barw (z wyjątkiem cieniowania obrazów realistycznych).

Barwa może być wykorzystana do kodowania informacji (fot. 20). Trzeba jednak pamiętać o pewnych przestrogach. Po pierwsze, barwa może wnieść niezamierzone znaczenie. Wyświetlanie dochodów firmy A na czerwono, a dochodów firmy B na zielono może sugerować, że firma A jest w kłopotach finansowych, ze względu na nasze skojarzenia czerwieni z deficytem finansowym. Jasne nasycone barwy rzucają się bardziej w oczy niż przyciemnione, blade barwy mogą niepotrzebnie coś uwypuklać. Dwa elementy obrazu, które mają tę samą barwę, mogą być widziane jako powiązane przez tę samą barwę, nawet jeżeli nie są.

Ten ostatni problem powstaje często wówczas, gdy barwa jest używana zarówno do grupowania pozycji menu, jak i do rozróżniania elementów obrazu, takich jak różne warstwy układów drukowanych albo struktury VLSI; na przykład użytkownicy mają tendencję do łączenia elementów wyświetlonych na zielono z pozycjami menu o takiej samej barwie. Ta tendencja jest jednym z powodów, dla których korzystanie z barwy w elementach interfejsu użytkownika – takich jak menu, pola

dialogowe i granice okien – powinno być ograniczone. (Inna przyczyna jest związana z dążeniem do pozostawienia samemu programowi użytkowemu możliwie największej liczby barw.)

Różne reguły stosowania barw wynikają raczej z psychologicznych niż estetycznych względów. Na przykład, ponieważ oko jest bardziej czułe na przestrzenne zmiany natężenia niż na zmiany barwy, linie, tekst i inne drobne szczegóły powinny odróżniać się od tła nie tyle barwą co jasnością (postrzeganym natężeniem) – zwłaszcza dla barw zawierających niebieski, ponieważ stosunkowo mało czopków jest czułych na barwę niebieską. Dlatego krawędź między dwoma barwnymi obszarami o jednakowej jasności, które różnią się tylko ilością barwy niebieskiej, będzie rozmyta. Czopki czułe na barwę niebieską są rozmieszczone na większej powierzchni siatkówki niż czopki czułe na barwy czerwoną i zieloną i peryferyjne widzenie barw jest bardziej ostre w odniesieniu do barwy niebieskiej – dlatego wiele „kogutów" na samochodach policyjnych jest teraz niebieskich, a nie czerwonych.

Barwy niebieska i czarna niewiele różnią się jasnością i dlatego stanowią wyjątkowo złą kombinację. Podobnie barwy żółta i biała są dość trudne do rozróżnienia, ponieważ obie są jasne. Na fotografii 10 pokazano efektywny sposób użycia barwy żółtej do uwypuklenia czarnego tekstu na białym tle. Barwa żółta dobrze kontrastuje z czarnym tekstem i dlatego jest dobrze widoczna. Ponadto uwypuklenie za pomocą barwy żółtej nie jest tak przytłaczające jak uwypuklenie za pomocą czarnego z białym tekstem, z czym się często można spotkać w monitorach monochromatycznych.

Biały tekst na niebieskim tle daje dobry kontrast, który jest mniej nieprzyjemny niż biały na czarnym. Najlepiej jest unikać barw czerwonej i zielonej o małym nasyceniu i luminancji, ponieważ są to barwy mylnie interpretowane przez daltonistów – nie rozróżniających barwy czerwonej od zielonej; jest to najczęstsza postać upośledzenia ze względu na percepcję barwy. Mayer i Greenberg opisują efektywne metody wyboru barw dla daltonistów [MEYE88].

Ludzkie oko nie jest w stanie rozróżnić barwy bardzo małych obiektów, o czym już raz wspomnieliśmy w związku z modelem barw YIQ NTSC, a więc barwy nie powinny być wykorzystywane do małych obiektów. W szczególności rozpoznawanie barw obiektów mniejszych niż 20 do 40 minut kątowych jest związane z dużym ryzykiem błędu [BISH60; HAEU76]. Obiekt o wysokości 0,25 cm widziany z odległości 60 cm (typowa odległość widzenia) odpowiada prawie takiemu łukowi i około 7 pikselom wysokości na monitorze o 1024 liniach o pionowej wysokości 36 cm. Jest oczywiste, że barwę jednego piksela jest trudno rozpoznać (por. zadanie 11.10).

Postrzegana barwa obszaru zależy od barwy otaczającego obszaru; ten efekt jest szczególnie problematyczny, jeżeli barwy są używane do kodowania informacji. Efekt zmniejsza się, jeżeli otaczająca powierzchnia ma barwę szarą albo względnie nienasyconą.

Barwa obszaru może wpływać na jego postrzeganą wielkość. Cleveland i McGill odkryli, że czerwony kwadrat jest postrzegany jako większy niż zielony kwadrat o tej samej wielkości [CLEV83]. Ten efekt może równie dobrze spowodować zwracanie przez użytkownika większej uwagi na obszar czerwony niż na zielony.

Jeżeli użytkownik przez kilka sekund wpatruje się w duży obszar o wysoce nasyconej barwie, a potem popatrzy gdzie indziej, to będzie w dalszym ciągu widział obraz tego dużego obszaru. Ten efekt wprawia w zakłopotanie i powoduje napięcie oka. Dlatego stosowanie dużych obszarów o nasyconych barwach jest nierozsądne. Ponadto, duże obszary o różnych barwach mogą sprawiać wrażenie, że znajdują się w różnych odległościach od obserwatora, ponieważ współczynnik załamania światła zależy od długości fali. Oko zmienia swoją ogniskową, gdy kierunek patrzenia zmienia się z jednego obszaru barwnego na drugi, a ta zmiana ogniskowej daje wrażenie zmiany odległości. Barwy czerwona i niebieska, które są na przeciwnych krańcach widma, mają najsilniejszy efekt różnicy odległości, przy czym czerwony obszar wydaje się być bliżej, a niebieski dalej. A więc równoczesne wykorzystywanie barwy niebieskiej dla bliskich obiektów i czerwonej dla dalekich nie jest racjonalne; odwrotna sytuacja jest poprawna.

Przy tych wszystkich niebezpieczeństwach i pułapkach nie można się dziwić temu, że jedna z naszych pierwszych reguł mówiła o ostrożności przy korzystaniu z barw.

Podsumowanie

Znaczenie barwy w grafice komputerowej będzie stale rosło, ponieważ monitory kolorowe i kolorowe urządzenia drukujące stają się normą w wielu zastosowaniach. W tym rozdziale wprowadziliśmy tylko te koncepcje związane z barwą, które są najistotniejsze dla grafiki komputerowej; dalsze informacje można znaleźć w bogatej literaturze poświęconej barwie, na przykład w pracach [BILL81; BOYN79; GREG66; HUNT87; JUDD75; WYSZ82]. Jeżeli chodzi o bardziej artystyczne i estetyczne spojrzenie na wykorzystanie barwy w grafice komputerowej, to można sięgnąć do takich pozycji jak [FROM84; MARC82; MEIE88; MURC85]. Trudne problemy precyzyjnego kalibrowania monitorów oraz dopasowywania barw pojawiających się na monitorach z barwami w druku są omawiane w pracach [COWA83; STON88].

Zadania

11.1. Znajdź równanie umożliwiające wyznaczenie liczby poziomów natężenia, jakie można uzyskać za pomocą wzorów $m \times n$ pikseli, jeżeli każdy piksel jest reprezentowany przez k bitów.

11.2. Napisz algorytm wyświetlania tablicy pikseli na dwupoziomowym urządzeniu wyjściowym. Wejściami do algorytmu są tablica $m \times m$ poziomów jasności pikseli, k bitów na piksel, macierz $n \times n$ sekwencji wzrostu [FOLE90, str. 569]. Załóżmy, że urządzenie wyjściowe ma rozdzielczość $m \cdot n \times m \cdot n$.

11.3. Napisz algorytm wyświetlania wypełnionego wielokąta na monitorze dwupoziomowym, korzystając ze wzoru wypełniania $n \times n$.

11.4. Jeżeli pewne wzory są wykorzystywane do wypełniania wyświetlanego wielokąta na rastrowym monitorze z wyświetlaniem międzyliniowym, to wszystkie bity włączone trafiają albo na nieparzyste, albo na parzyste wiersze, co wprowadza pewne migotanie. Zmodyfikuj tak algorytm z zadania 11.3, żeby dokonać takiego przestawienia wierszy wzoru $n \times n$, żeby kolejne wykorzystane kopie wzoru wykorzystywały na przemian nieparzyste i parzyste wiersze. Na rysunku 11.23 pokazano wynik uzyskany przy wykorzystaniu poziomu natężenia 1 z rys. 11.4 z i bez tej zmiany.

Rys. 11.23. Wyniki uzyskane, gdy poziom natężenia 1 z rys. 11.4 został wykorzystany: a) ze zmianą (podświetlone piksele są w obu wierszach); b) bez zmiany (wszystkie podświetlone piksele są w tym samym wierszu)

11.5. Narysuj miejsce geometryczne punktów o stałych wartościach luminancji 0,25, 0,50 i 0,75, zdefiniowanych przez $Y = 0{,}299R + 0{,}587G + 0{,}114B$, w sześcianie RGB oraz w ostrosłupie HSV.

11.6. W zależności od R, G i B wyraź I z YIQ oraz V z HSV. Zauważmy, że I i V to nie to samo.

11.7. Omów projekt monitora rastrowego, w którym wykorzystuje się do określania barwy model HSV, a nie RGB.

11.8. Przepisz algorytm konwersji HSV do RGB tak, żeby zwiększyć jego efektywność. Zastąp polecenia podstawienia dla p, q i t przez $vs = v * s$; $vsf = vs * f$; $p = v - vs$; $q = v - vsf$; $t = p + vsf$. Załóż również, że R, G i B są w przedziale [0, 255] i sprawdź, ile obliczeń można zamienić na całkowitoliczbowe.

11.9. Napisz program, który wyświetla obok siebie dwie siatki 16 × 16. Wypełnij każdą siatkę barwami. Lewa siatka będzie miała 256 barw losowo wybranych z przestrzeni barw HSV (utwórz je korzystając z generatora

liczb losowych do wybrania jednej z 10 równooddalonych wartości dla każdej z wielkości *H*, *S* i *V*). Prawa siatka zawiera te same 256 barw, posortowanych ze względu na *H*, *S* i *V*. Porównaj wyniki uzyskane przy wybieraniu jako pierwszego klucza wyboru *H*, *S* i *V*.

11.10. Napisz program wyświetlania na szarym tle małych kwadratów o barwach pomarańczowej, czerwonej, zielonej, niebieskiej, cyjan, magenta i żółtej. Każdy kwadrat jest oddzielony od pozostałych i zajmuje $n \times n$ pikseli, przy czym n jest zmienną wejściową. Jak duże musi być n, żeby użytkownik mógł jednoznacznie określić barwę każdego kwadratu z odległości 60 i 120 cm? Jaka powinna być zależność między dwiema wartościami n? Jaki wpływ, jeżeli w ogóle, mają różne barwy podłoża na ten wynik?

11.11. Oblicz liczbę bitów dokładności w pośredniej tablicy potrzebnych do zapamiętania 256 różnych poziomów natężenia dla dynamicznych zakresów natężenia 50, 100 i 200.

11.12. Napisz program interpolacji liniowej między dwiema barwami w RGB i HSV. Na wejście można podawać dwie dowolne barwy określone w jednym z tych modeli.

12. Dążenie do wizualnego realizmu

W poprzednich rozdziałach omawialiśmy metody graficzne wykorzystujące proste prymitywy 2D i 3D. Tworzone obrazy, np. szkieletowe domy z rozdz. 6, reprezentowały obiekty, które w rzeczywistości są znacznie bardziej złożone, zarówno jeśli chodzi o strukturę, jak i o wygląd. W tym rozdziale wprowadzamy ważne zastosowanie grafiki, a mianowicie tworzenie realistycznych obrazów scen 3D.

Co to jest obraz *realistyczny*? To, w jakim sensie o obrazie – namalowanym, sfotografowanym czy wygenerowanym komputerowo – można powiedzieć, że jest realistyczny, jest przedmiotem raczej akademickiej debaty [HAGE86]. Korzystamy z tego pojęcia w szerokim znaczeniu w odniesieniu do obrazu, który zawiera wiele efektów oddziaływania światła z rzeczywistymi obiektami fizycznymi. Dlatego traktujemy obrazy realistyczne jako kontinuum i swobodnie mówimy o obrazach i o metodach wykorzystanych do ich utworzenia jako o bardziej lub mniej realistycznych. Na jednym końcu kontinuum są przykłady tego, o czym mówimy jako o *realizmie fotograficznym* (albo o *fotorealizmie*). Te obrazy mają na celu syntezę gamy natężeń światła, które byłoby odwzorowane na filmie aparatu fotograficznego skierowanego na obrazowany obiekt. Gdy zbliżamy się do drugiego końca kontinuum, napotykamy obrazy, które zawierają sukcesywnie coraz mniej efektów wizualnych, które będziemy omawiali.

Powinniśmy pamiętać o tym, że bardziej realistyczny obraz to niekoniecznie obraz najbardziej pożądany albo użyteczny. Jeżeli ostatecznym celem obrazu jest przeniesienie informacji, to obraz wolny od złożonych cieni i odbić może być lepszy niż efekt dążenia do realizmu fotograficznego. W wielu zastosowaniach metod omawianych w następnych rozdziałach rzeczywistość jest świadomie zmieniana ze względu na

efekty estetyczne albo w celu spełnienia naiwnych oczekiwań obserwatora. Metody takie są stosowane z tych samych powodów, dla których w filmach fantastyczno-naukowych słychać dźwięki broni w przestrzeni kosmicznej – co jest niemożliwe w próżni. Na przykład przy obrazowaniu Uranu na fotografii 23 Blinn zapalił dodatkowe światło po zaciemnionej stronie planety i podciągnął kontrast po to, żeby równocześnie widoczne były jej wszystkie cechy – w przeciwnym przypadku zaciemniona strona planety powinna być czarna. Traktując swobodnie fizykę można uzyskać atrakcyjne, zapadające w pamięć i użyteczne obrazy!

Tworzenie obrazów realistycznych wiąże się z kilkoma etapami, które są dokładnie omawiane w następnych rozdziałach. Chociaż o tych etapach często się myśli jak o tworzeniu koncepcyjnego potoku, to – jak zobaczmy – kolejność wykonywania operacji może się zmieniać, zależnie od użytych algorytmów. W pierwszym etapie są generowane modele obiektów z wykorzystaniem metod omówionych w rozdz. 9 i 10. Następnie wybieramy rzutowanie (tak jak to pokazano w rozdz. 6) i warunki oświetlenia. Z kolei za pomocą algorytmów omawianych w rozdz. 13 są określane te powierzchnie, które są widoczne dla obserwatora. Barwa przypisywana każdemu pikselowi rzutu widocznej powierzchni jest funkcją światła odbitego i przepuszczanego przez obiekty i jest określana za pomocą metod omówionych w rozdz. 14. Otrzymany obraz może być z kolei łączony z obrazami wygenerowanymi wcześniej (na przykład w celu ponownego użycia złożonego tła) z wykorzystaniem metod składania obrazów. Wreszcie, jeżeli tworzymy animowaną sekwencję, to musimy określić zależne od czasu zmiany modelu, oświetlenia i rzuty. Proces tworzenia obrazów danych w postaci modeli jest często określany jako *rendering*. Określenie *rasteryzacja* jest używane w odniesieniu do tych kroków, które wiążą się z określeniem wartości pikseli na podstawie opisu wejściowych prymitywów geometrycznych.

W tym rozdziale realistyczny rendering jest przedstawiony z różnych punktów widzenia. Po pierwsze, przyjrzymy się kilku zastosowaniom, w których realistyczne obrazy były wykorzystywane. Potem omówimy, z grubsza w kolejności historycznej, kilka metod, które umożliwiają tworzenie sukcesywnie coraz bardziej realistycznych obrazów. Każda metoda jest ilustrowana obrazem standardowej sceny z zastosowaniem do niej nowych metod. Wreszcie kończymy sugestiami, jak należy podejść do kolejnych rozdziałów.

12.1. Dlaczego realizm?

Tworzenie realistycznych obrazów jest ważnym celem w takich dziedzinach jak symulacja, projektowanie, rozrywka i reklama, badania i nauczanie oraz zarządzanie i sterowanie.

Systemy symulacyjne tworzą obrazy, które nie tylko są realistyczne, ale również zmieniają się dynamicznie. Na przykład symulator lotu pokazuje widok, jaki powinien być widoczny z kokpitu lecącego samolotu. W celu uzyskania efektu ruchu system generuje i wyświetla nowy, nieco inny widok wiele razy na sekundę. Symulatory były i są używane do szkolenia pilotów statków kosmicznych, samolotów i łodzi, a ostatnio również kierowców samochodów.

Projektanci obiektów 3D, np. samochodów, samolotów i budowli, chcą oglądać wstępne projekty. Tworzenie realistycznych obrazów generowanych komputerowo jest często łatwiejsze, tańsze i bardziej efektywne, jeśli chodzi o oglądanie wstępnych rezultatów, niż budowanie modeli oraz prototypów, i umożliwia również rozpatrywanie dodatkowych alternatywnych projektów. Jeżeli sama praca projektowa jest również wykonywana za pomocą komputera, to opis cyfrowy obiektu może być dostępny dla potrzeb tworzenia obrazów. W idealnym przypadku projektant może również współpracować interakcyjnie z wyświetlanym obrazem w celu modyfikowania projektu. Dla potrzeb przemysłu motoryzacyjnego zostały opracowane systemy projektowania, które umożliwiają określanie, jak będzie wyglądał samochód przy różnych warunkach oświetlenia. Realistyczna grafika jest często łączona z programami, które analizują różne aspekty projektowanego obiektu, takie jak jego właściwości jako całości albo odpowiedź na naprężenia.

Obrazy generowane komputerowo są powszechnie używane w świecie rozrywki, zarówno w tradycyjnych filmach animowanych, jak i realistycznych i surrealistycznych obrazach dla logo, ogłoszeń i filmów z gatunku fantastyczno-naukowych. Filmy generowane komputerowo mogą naśladować tradycyjną animację, ale mogą również przewyższać ręczne metody dzięki wprowadzeniu bardziej złożonego ruchu i bogatszych albo bardziej realistycznych obrazów. Niektóre złożone obrazy realistyczne mogą być wytwarzane mniejszym kosztem niż filmowanie fizycznych modeli obiektów. Były wygenerowane obrazy, które w innym przypadku byłyby wyjątkowo trudne albo niemożliwe do zainscenizowania za pomocą rzeczywistych modeli. Specjalizowany sprzęt i oprogramowanie utworzone do zastosowań w rozrywce obejmuje wyrafinowane systemy malarskie i systemy czasu rzeczywistego do generowania specjalnych efektów i łączenia obrazów. W miarę postępu technologii domowe i uliczne gry wideo generują coraz bardziej realistyczne obrazy.

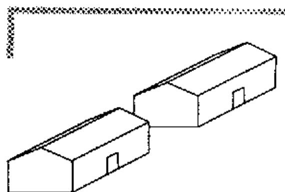
Realistyczne obrazy stają się podstawowym narzędziem w badaniach i edukacji. Szczególnie ważnym przykładem jest wykorzystanie grafiki w modelowaniu molekularnym (fot. 22). Jest interesujące, jak jest tu rozwinięta koncepcja realizmu; realistyczne obrazy nie prezentują „rzeczywistych" atomów, a raczej stylizowane przestrzenne modele składające się z kulek i patyków, które umożliwiają budowanie większych

struktur, niż to jest możliwe za pomocą fizycznych modeli, i które umożliwiają realizację specjalnych efektów, takich jak animowane drgające wiązania i zmiany barw reprezentujące reakcje. W skali makroskopowej filmy wyprodukowane w JPL pokazują misję sondy kosmicznej NASA, pokazanej na fot. 23.

Inne zastosowanie realistycznych obrazów wiąże się z zarządzaniem i sterowaniem, kiedy użytkownik musi być informowany o złożonych procesach reprezentowanych przez obraz i musi nimi sterować. W przeciwieństwie do symulacji, gdzie dąży się do tego, żeby odzwierciedlić to, co użytkownik faktycznie widzi i czuje w symulowanej sytuacji, zastosowania związane z zarządzaniem i sterowaniem często tworzą obrazy symboliczne, które podkreślają niektóre dane, a tłumią inne, aby móc podjąć decyzję.

12.2. Podstawowe trudności

Podstawowa trudność związana z uzyskaniem pełnego realizmu wzrokowego wiąże się ze złożonością rzeczywistego świata. Zaobserwujmy bogactwo swojego otoczenia. Jest wiele powierzchni pokrytych teksturą, występują subtelne gradacje barw, cieni, odbić i pewne nieregularności w otaczających obiektach. Pomyślmy o wzorach na pofałdowanych ubraniach, o teksturze skóry, rozczochranych włosach, śladach na podłodze i złuszczonej farbie na ścianie. To wszystko tworzy rzeczywiste doświadczenia wzrokowe. Koszty obliczeniowe związane z symulacją takich efektów mogą być wysokie: niektóre obrazy pokazane na zdjęciach wymagają wielu minut, a nawet godzin obliczeń na komputerach dużej mocy.

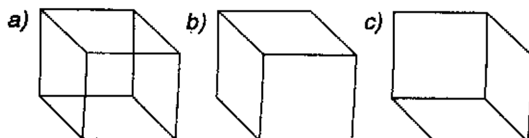


Rys. 12.1. Rysunek odcinkowy dwóch domów

Łatwiejszym celem w klasie problemów związanych z realizmem jest dostarczenie wystarczającej informacji, która umożliwi użytkownikowi zrozumienie przestrzennych zależności między kilkoma obiektami. Taki cel można uzyskać przy niewielkim koszcie i jest powszechnym wymaganiem w systemach CAD i w wielu innych zastosowaniach. Chociaż wysoce realistyczne obrazy przekazują przestrzenne zależności 3D, zazwyczaj przekazują o wiele więcej informacji. Na przykład na rys. 12.1 pokazano prosty obraz wystarczający do przekonania nas, że jeden budynek jest częściowo ukryty za drugim. Nie ma potrzeby pokazywać dachów budynków pokrytych gontem, ścian wykonanych z cegły albo cieni rzucanych przez budynki. W rzeczywistości w niektórych kontekstach takie dodatkowe szczegóły mogą tylko odwracać uwagę obserwatora od najważniejszej wyświetlanej informacji.

Jedną z trudności związanych z obrazowaniem zależności przestrzennych jest to, że większość urządzeń wyświetlających jest typu 2D. Dlatego obiekty 3D muszą być rzutowane na płaszczyznę z towarzyszą-

cą temu istotną utratą informacji – może to czasami prowadzić do niejednoznaczności w interpretacji obrazu. Niektóre z metod wprowadzonych w tym rozdziale mogą być użyte do ponownego dodania informacji występujących normalnie w naszym środowisku wizualnym, tak żeby mechanizmy percepcji odległości człowieka mogły rozwiązać poprawnie pozostałe niejednoznaczności.

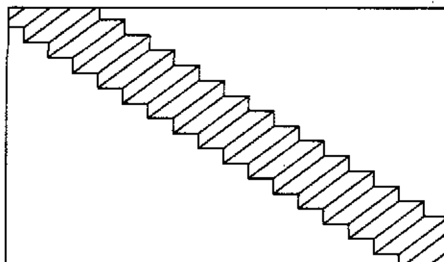


Rys. 12.2. Sześcian Neckera. Czy sześcian z rysunku a) jest zorientowany jak sześcian z rysunku b), czy jak sześcian z rysunku c)?

Weźmy pod uwagę złudzenie związane z sześcianem Neckera z rys. 12.2a – rzutem 2D sześcianu; nie wiemy, czy reprezentuje on sześcian z rysunku b) czy z rysunku c). Obserwator może mieć wątpliwości, ponieważ rys. 12.2a nie zawiera wystarczającej informacji wizualnej dla jednoznacznej interpretacji.

Im więcej obserwator wie o wyświetlanym obiekcie, tym łatwiej może tworzyć to, co Gregory nazywa *hipotezą obiektu* [GREG70]. Na rysunku 12.3 pokazano efekt schodów Schrödera – patrzymy w dół schodów, czy patrzymy na nie z dołu? Prawdopodobnie wybierzemy tę pierwszą interpretację, ponieważ częściej obserwujemy schody pod naszymi stopami niż nad naszymi głowami i dlatego więcej wiemy o schodach oglądanych z góry. Przy pewnej wyobraźni możemy jednak przyjąć alternatywną interpretację rysunku. Po przyjrzeniu się większość obserwatorów popatrzy odwrotnie i schody znowu będą wydawały się jak oglądane z góry. Oczywiście dodatkowy kontekst, na przykład osoba stojąca na schodach, umożliwi rozwiązanie tej niejednoznaczności.

W następnych punktach omawiamy kilka etapów ze ścieżki prowadzącej do realistycznych obrazów. Ścieżka jest raczej zbiorem przeple-



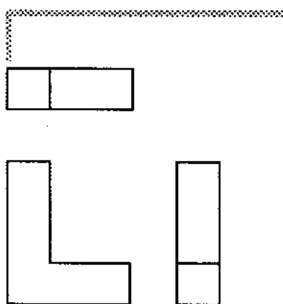
Rys. 12.3. Schody Schrödera. Czy schody są widziane z góry czy z dołu?

cionych prób niż jedną prostą drogą, ale ze względu na prostotę dokonaliśmy jej linearyzacji, dając czysto opisowe wprowadzenie do dokładnego omówienia w kolejnych rozdziałach. Najpierw wymieniamy metody odpowiednie dla statycznych rysunków konstruowanych z odcinków. Te metody koncentrują się na sposobach prezentowania przestrzennych zależności 3D między różnymi obiektami na ekranie 2D. Następnie pojawiają się metody cieniowania obrazów, możliwe do zrealizowania dzięki sprzętowi grafiki rastrowej, które koncentrują się na interakcji obiektów ze światłem. Następnie omawiamy problemy zwiększonej złożoności modelu i dynamiki, odnoszące się zarówno do obrazów odcinkowych, jak i cieniowanych. Wreszcie omawiamy możliwości generowania prawdziwych obrazów 3D, rozwój sprzętu wyświetlającego i przyszłe miejsce generowania obrazów w kontekście pełnej interaktywnej syntezy środowiskowej.

12.3. Metody renderingu dla rysunków odcinkowych

W tym punkcie skupimy uwagę na jednym z elementów prowadzących do realizmu: na pokazaniu przestrzennych zależności głebokości na płaszczyźnie. Ten cel osiąga się korzystając z płaskich rzutów geometrycznych zdefiniowanych w rozdz. 6.

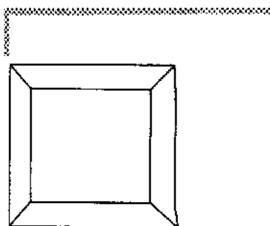
12.3.1. Wielokrotne rzuty prostokątne



Rys. 12.4. Rzuty prostokątne: przedni, górny i boczny bryły o kształcie litery „L"

Najłatwiejszy sposób rzutowania polega na wykonaniu rzutów prostokątnych, takich jak rzuty pionowy i poziomy, w których rzutnia jest prostopadła do głównej osi rzutowania. Ponieważ gubi się informację o głębokości, na ogół oba rzuty pokazuje się razem, tak jak w przypadku rzutów górnego, przedniego i bocznego bryły w kształcie litery „L" na rys. 12.4. Ten konkretny rysunek nie jest trudny do zrozumienia; zrozumienie złożonych rysunków produkowanych części na podstawie zestawu takich rzutów może jednak wymagać wielu godzin studiowania. Szkolenie oraz doświadczenie oczywiście zwiększają naszą zdolność interpretacji i znajomość typów reprezentowanych obiektów przyspiesza formułowanie poprawnych hipotez co do obiektu. Tak skomplikowane sceny jak ta z fotografii 24 często sprawiają kłopoty, gdy są podane tylko trzy takie rzuty. Chociaż jeden punkt może być jednoznacznie zlokalizowany na podstawie trzech wzajemnie prostopadłych rzutów, wiele punktów i odcinków może zasłaniać się wzajemnie przy takim rzutowaniu.

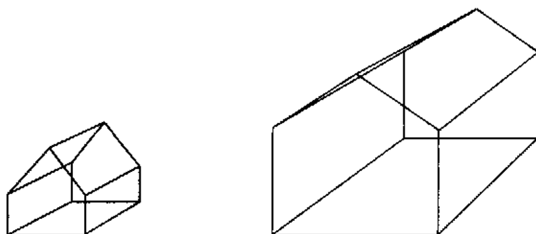
12.3.2. Rzuty perspektywiczne



Rys. 12.5. Rzut perspektywiczny sześcianu

W rzucie perspektywicznym wielkość obiektu zmienia się odwrotnie proporcjonalnie do jego odległości od obserwatora. Rzut perspektywiczny sześcianu (rys. 12.5) pokazuje takie skalowanie. Wciąż jednak istnieje niejednoznaczność; rzut równie dobrze mógłby pokazywać ramę obrazu albo równoległy rzut ostrosłupa ściętego, albo rzut perspektywiczny prostopadłościanu. Jeżeli postawimy hipotezę, że jest to ostrosłup ścięty, to mniejszy kwadrat reprezentuje ścianę bliższą obserwatorowi; jeżeli natomiast przyjmiemy hipotezę o sześcianie albo prostopadłościanie, to mniejszy kwadrat reprezentuje ścianę bardziej odległą od obserwatora.

Nasza interpretacja rzutów perspektywicznych wynika często z założenia, że mniejszy obiekt znajduje się w większej odległości. Na rysunku 12.6 założylibyśmy prawdopodobnie, że większy dom jest bliższy obserwatorowi. Jednak dom, który wydaje się większy (na przykład



Rys. 12.6. Rzut perspektywiczny dwóch domów

dworek), może faktycznie być dalej niż inny, który sprawia wrażenie mniejszego (na przykład mała willa), przynajmniej dopóty, dopóki nie ma innych wskazówek na przykład w postaci drzew czy okien. Jeżeli obserwator wie, że rzutowane obiekty mają wiele linii równoległych, to perspektywa ułatwia wyczucie odległości, ponieważ linie równoległe sprawiają wrażenie zbiegania się w punktach zbieżności. Ta zbieżność w rzeczywistości może być silniejszą wskazówką, jeśli chodzi o informację o głębokości, niż efekt malejących rozmiarów. Na fotografii 25 pokazano rzut perspektywiczny naszej standardowej sceny.

12.3.3. Wskazówki na temat głębokości (odległości)

Głębokość (odległość) obiektu może być reprezentowana za pomocą jasności obrazu: części obrazu, które mają sprawiać wrażenie bardziej odległych od obserwatora, są wyświetlane z mniejszą jasnością. Ten

efekt jest określany jako zmiana jasności w funkcji odległości. Wykorzystuje się tu fakt, że odleglejsze obiekty wydają się przyciemnione w porównaniu z bliższymi obiektami, zwłaszcza jeżeli są oglądane przez mgłę. Takie efekty mogą być wystarczająco przekonywujące na to, żeby artyści korzystali ze zmiany jasności (tekstur, ostrości i barwy) w celu pokazania odległości jako *perspektywy powietrznej*. Dlatego efekt zmiany jasności w funkcji odległości może być traktowany jako uproszczona wersja efektów tłumienia atmosferycznego.

W monitorach wektorowych efekt zmiany jasności w funkcji odległości jest implementowany na zasadzie interpolacji jasności strumienia wzdłuż wektora w funkcji współrzędnej z początku i końca wektora. W systemach grafiki barwnej na ogół uogólnia się tę metodę tak, żeby umożliwić interpolację między barwą prymitywu a określoną przez użytkownika barwą dla odległego punktu, którą najczęściej jest barwa tła. W celu ograniczenia efektu do pewnego zakresu głębokości, PHIGS PLUS umożliwia użytkownikowi wybranie przedniej i tylnej płaszczyzny, między którymi ma nastąpić realizacja efektu zmiany jasności w funkcji odległości. Współczynnik skali związany z każdą płaszczyzną wskazuje na proporcje oryginalnej barwy i barwy wynikającej ze zmiany jasności w funkcji odległości, jakie mają być użyte przed przednią ścianą i za tylną ścianą. Barwa punktów między płaszczyznami jest interpolowana liniowo między tymi dwiema wartościami. Rozdzielczość oka w odniesieniu do jasności jest mniejsza niż rozdzielczość przestrzenna, a więc zmiana jasności w funkcji odległości nie jest użyteczna, jeśli chodzi o dokładne wskazywanie niewielkich różnic odległości. Jest jednak bardzo efektywna we wskazywaniu dużych różnic albo jako wyolbrzymiona wskazówka przy pokazywaniu niewielkich różnic.

12.3.4. Obcinanie w funkcji odległości

Dalszą informację można uzyskać dzięki obcinaniu w funkcji odległości. Tylna ściana obcinająca przecina wyświetlane obiekty. Wtedy obserwator widzi obiekty częściowo obcięte przez płaszczyznę obcinającą. Można również użyć przedniej płaszczyzny obcinającej. Dopuszczając dynamiczne zmiany położenia jednej albo obu płaszczyzn, system może przekazać obserwatorowi więcej informacji o głębokości. Obcinanie za pomocą tylnej płaszczyzny może być traktowane jako szczególny przypadek zmiany jasności w funkcji odległości; przy zwykłej zmianie jasności mamy do czynienia z gładką funkcją odległości z; przy obcinaniu w funkcji odległości jest to funkcja skokowa. Metodą związaną z obcinaniem w funkcji odległości jest uwypuklanie wszystkich punktów obiektu przeciętych przez pewną płaszczyznę. Ta metoda jest szczegól-

nie efektywna, gdy płaszczyzna przecinająca przesuwa się dynamicznie wzdłuż obiektu; była nawet użyta do ilustrowania zmiany odległości wzdłuż czwartego wymiaru [BANC77].

12.3.5. Tekstura

Do obiektu można zastosować proste tekstury wektorowe, np. kreskowanie ukośne. Takie tekstury wypełniają kształt obiektu i lepiej go określają. Pokrycie teksturą jednej z kilku identycznych ścian może ułatwić interpretację potencjalnie niejednoznacznych rzutów. Korzystanie z tekstur jest szczególnie użyteczne w rzutach perspektywicznych, ponieważ wprowadza się więcej linii, których zbieżność i skrócenie perspektywiczne mogą dostarczyć użytecznych wskazówek co do głębokości.

12.3.6. Barwa

Barwa może być wykorzystana symbolicznie do rozróżnienia jednego obiektu od drugiego dzięki przypisaniu różnych barw każdemu obiektowi w scenie. Barwa może być również użyta w rysunkach odcinkowych w celu wprowadzenia dodatkowych informacji. Na przykład barwa każdego wektora może być określona na zasadzie interpolowania barw, które kodują temperatury na końcach wektora.

12.3.7. Określanie linii widocznych

Ostatnią omawianą przez nas metodą związaną z rysunkami odcinkowymi jest *określanie linii widocznych* albo *usuwanie linii niewidocznych,* co powoduje wyświetlanie tylko widocznych, to znaczy nie przesłoniętych, odcinków albo części odcinków. Tylko powierzchnie ograniczone przez krawędzie (odcinki) mogą przesłaniać inne odcinki. Dlatego obiekty, które mogą zasłaniać inne, muszą być modelowane albo jako zbiory powierzchni, albo jako bryły.

Na fotografii 26 zilustrowano problem usuwania linii niewidocznych (również z wykorzystaniem barwy, jak to opisano w p. 12.3.6). Ponieważ rzuty z usuniętymi liniami niewidocznymi ukrywają wszystkie wewnętrzne struktury nieprzezroczystych obiektów, nie jest to najbardziej efektywna metoda pokazywania zależności związanych z głębokością. Rzuty z usuniętymi liniami niewidocznymi niosą mniej informacji o głębokości niż rzuty złożeniowe i przekroje. Rozsądnym kompromisem może być pokazywanie linii niewidocznych za pomocą linii przerywanych.

12.4. Metody renderingu dla obrazów cieniowanych

Metody omówione w p. 12.3 mogą być użyte do tworzenia rysunków odcinkowych na monitorach wektorowych i monitorach rastrowych. Metody wprowadzone w tym punkcie wykorzystują zdolność urządzeń rastrowych do wyświetlania obszarów cieniowanych. Gdy obraz jest tworzony dla wyświetlacza rastrowego, wówczas powstają problemy związane ze względnie grubą siatką pikseli, na której muszą być reprodukowane gładkie kontury i cieniowanie. W najprostszych sposobach renderingu w przypadku cieniowanych obrazów występuje problem zakłóceń, z którym spotkaliśmy się w p. 3.14. Ze względu na podstawową rolę, jaką odgrywają metody odkłócania w tworzeniu obrazów dobrej jakości, wszystkie obrazy z tego punktu zostały utworzone z wykorzystaniem metod odkłócania.

12.4.1. Określanie powierzchni widocznych

Przez analogię do określania linii widocznych, określanie powierzchni widocznych albo usuwanie powierzchni niewidocznych oznacza wyświetlanie tylko tych części powierzchni, które są widoczne dla obserwatora. Jak widzieliśmy, proste rysunki odcinkowe mogą być często zrozumiałe bez określania widoczności odcinków. Jeżeli jest tylko kilka odcinków, to te z przodu nie muszą istotnie zakłócać widoku tych znajdujących się z tyłu. W grafice rastrowej, jeżeli powierzchnie są wyświetlane jako nieprzezroczyste obszary, to, żeby obraz miał sens, konieczne jest określanie powierzchni widocznych. Na fotografii 27 pokazano przykład, na którym wszystkie ściany obiektu są malowane tą samą barwą.

12.4.2. Oświetlanie i cieniowanie

Istotnym problemem związanym z fotografią 27 jest to, że każdy obiekt pojawia się jako płaska sylwetka. Naszym następnym krokiem w celu uzyskania realizmu jest więc pocieniowanie widocznych powierzchni. Ostatecznie wygląd każdej powierzchni powinien zależeć od rodzajów źródeł światła oświetlających obiekt, jego właściwości (barwa, tekstura, odbicia) oraz jego położenia i orientacji względem źródeł światła, obserwatora i innych powierzchni.

W wielu rzeczywistych środowiskach wizualnych istotna ilość światła otoczenia dociera ze wszystkich kierunków. Światło otoczenia jest najłatwiejszym do modelowania rodzajem źródła światła, ponieważ

w prostym modelu oświetlenia zakłada się wytwarzanie stałego oświetlenia wszystkich powierzchni, niezależnie od ich pozycji i orientacji. Korzystanie jedynie ze światła otoczenia prowadzi do nierealistycznych obrazów, ponieważ niewiele rzeczywistych środowisk jest oświetlanych wyłącznie jednolitym światłem otoczenia. Fotografia 27 jest przykładem obrazu cieniowanego w ten sposób.

Źródło punktowe, którego promienie rozchodzą się z jednego punktu, może aproksymować małą żarzącą się żarówkę. Źródło kierunkowe, dla którego wszystkie promienie przychodzą z tego samego kierunku, może być użyte do reprezentowania odległego słońca na zasadzie reprezentowania go jako źródła punktowego znajdującego się nieskończenie daleko. Modelowanie takich źródeł wymaga dodatkowej pracy, ponieważ ich efekt zależy od orientacji powierzchni. Jeżeli powierzchnia jest prostopadła do padających promieni świetlnych, to jest jasno oświetlona; im bardziej powierzchnia jest pochylona w stosunku do promieni świetlnych, tym słabsze jest jej oświetlenie. Ta zmiana oświetlenia jest oczywiście istotną wskazówką co do struktury 3D obiektu. Wreszcie jeszcze bardziej złożone do modelowania jest źródło rozproszone albo rozszerzone, np. zestaw jarzeniówek, którego powierzchnia emituje światło; przyczyną jest to, że światło dociera nie z jednego kierunku i nie z jednego punktu. Fotografia 28 pokazuje efekt oświetlenia sceny za pomocą światła otoczenia i źródła punktowego i cieniowania każdego wielokąta oddzielnie.

12.4.3. Cieniowanie interpolacyjne

Cieniowanie interpolacyjne jest to metoda, w której informacja o cieniowaniu jest obliczana dla każdego wierzchołka wielokąta i interpolowana wewnątrz wielokąta w celu określenia barwy każdego piksela. Ta metoda jest szczególnie efektywna wówczas, gdy opis obiektu wielokątowego ma przybliżać powierzchnię krzywoliniową. W tym przypadku informacja o cieniowaniu obliczana w każdym wierzchołku (jego barwa) może być wyznaczana z wykorzystaniem danych o bieżącej orientacji powierzchni w tym punkcie i jest wykorzystywana do wszystkich wielokątów, do których należy ten wierzchołek. Interpolowanie między tymi wartościami wzdłuż wielokąta aproksymuje gładkie zmiany barwy, które występują wzdłuż powierzchni krzywoliniowej.

Cieniowanie interpolacyjne może być korzystne również w odniesieniu do obiektów, co do których zakłada się, że są raczej wielokątowe niż krzywoliniowe, ponieważ informacje o barwie obliczone dla każdego wierzchołka wielokąta mogą się różnić, chociaż na ogół znacznie mniej niż w przypadku obiektów krzywoliniowych. Gdy informacja o barwie jest obliczana dla prawdziwego obiektu wielokątowego, war-

tość określana dla wierzchołka wielokąta jest wykorzystywana tylko dla tego wielokąta, a nie dla innych, w których ten wierzchołek również występuje. Na fotografii 29 pokazano cieniowanie Gourauda, będące odmianą cieniowania interpolacyjnego omawianego w p. 14.2.

12.4.4. Właściwości materiału

Dalszy krok w kierunku realizmu można zrobić biorąc pod uwagę przy określaniu cieniowania właściwości materiału, z którego jest wykonany obiekt. Niektóre materiały są matowe i rozpraszają światło niemal równomiernie we wszystkich kierunkach, tak jak w przypadku kawałka kredy; inne są błyszczące i odbijają światło tylko w określonych kierunkach względem obserwatora i źródła światła, jak w przypadku lustra. Na fotografii 31 pokazano naszą scenę w przypadku, gdy niektóre obiekty są modelowane jako błyszczące. Zostało tu wykorzystane cieniowanie Phonga, które jest dokładniejszą metodą interpolowania cieniowania (p. 14.2).

12.4.5. Modelowanie powierzchni krzywoliniowych

Chociaż cieniowanie interpolacyjne znacznie poprawia wygląd obrazu, geometria obiektu jest wciąż wielokątowa. Na fotografii 32 znajdują się obiekty, do modelowania których wykorzystano powierzchnie krzywoliniowe. Pełna informacja o cieniowaniu jest obliczana dla każdego piksela w obrazie.

12.4.6. Ulepszone oświetlanie i cieniowanie

Jedną z najważniejszych przyczyn „nierealistycznego" wyglądu większości obrazów generowanych metodami grafiki komputerowej jest brak dokładnego modelowania wielu sposobów oddziaływania światła z obiektami. Na fotografii 33 wykorzystano lepsze modele oświetlenia. W punkcie 14.1.7 omówiono sposoby zmierzające do projektowania bardziej efektywnych i fizycznie poprawnych modeli oświetlenia.

12.4.7. Tekstura

Tekstura obiektu nie tylko niesie dodatkowe wskazówki o głębokości, tak jak to omówiono w p. 12.3.5, ale może również naśladować szczegóły powierzchni rzeczywistych obiektów. Na fotografii 35 pokazano wie-

le sposobów, za pomocą których może być symulowana tekstura, zaczynając od zmiany barwy powierzchni (tak jak to zrobiono w przypadku kuli pokrytej wzorem), a kończąc na faktycznym deformowaniu geometrii powierzchni (tak jak to zrobiono w przypadku torusa i stożka).

12.4.8. Cienie

Dalszy element realizmu możemy wprowadzić reprodukując cienie rzucane przez obiekty jeden na drugi. Zauważmy, że jest to pierwsza metoda, w której wygląd powierzchni widocznych obiektu zależy od innych obiektów. Fotografia 35 pokazuje cienie rzucane przez lampę na tylną ścianę. Cienie zwiększają realizm i dostarczają dodatkowych informacji o głębokości: jeżeli obiekt *A* rzuca cień na powierzchnię *B*, to wiemy, że *A* jest między *B* i bezpośrednim albo odbitym źródłem światła. Punktowe źródło światła rzuca ostre cienie, ponieważ z dowolnego punktu jest ono albo całkowicie widoczne, albo całkowicie niewidoczne. Rozszerzone źródło światła rzuca miękkie cienie, ponieważ jest gładkie przejście od tych punktów, z których jest widoczne całe źródło światła, przez te punkty, z których jest widoczna tylko część źródła, do tych, z których go w ogóle nie widać.

12.4.9. Przezroczystość i odbicia

Dotychczas zajmowaliśmy się tylko powierzchniami nieprzezroczystymi. Przy tworzeniu obrazów mogą być użyteczne również powierzchnie przezroczyste. Proste modele przezroczystości nie uwzględniają załamania światła przy przejściu przez przezroczystą bryłę. Brak załamania może być jednak zdecydowaną zaletą, jeżeli przezroczystość jest używana nie tyle do symulowania rzeczywistości, ile do ujawnienia wewnętrznej geometrii obiektu. W bardziej złożonych modelach występuje załamanie, rozpraszająca przezroczystość i tłumienie światła w funkcji odległości. Podobnie model odbicia światła może symulować ostre odbicia zwierciadlane innego obiektu albo rozpraszane odbicia mniej dokładnie wypolerowanej powierzchni. Na fotografii 36 pokazano efekt odbicia na podłodze i czajniczku; na fotografii 41 pokazano efekt przezroczystości.

Podobnie jak przy modelowaniu cieni, modelowanie przezroczystości albo odbicia wymaga znajomości innych powierzchni prócz tej, która jest cieniowana. Rozpraszająca przezroczystość jest pierwszym efektem spośród omawianych, który wymaga, żeby obiekty były modelowane jako bryły, a nie jako powierzchnie! Musimy coś wiedzieć o materiałach, przez które przechodzi promień światła, i o odległości, jaką przebywa w celu poprawnego modelowania jego załamania.

12.4.10. Ulepszone modele kamery

We wszystkich pokazanych dotychczas obrazach wykorzystywano model kamery z obiektywem otworkowym i nieskończenie szybką migawką: wszystkie obiekty są ostre i reprezentują świat w jednej chwili. Sposób widzenia przez nas świata (i przez kamerę) można modelować dokładniej. Na przykład modelując właściwości skupiające soczewek możemy tworzyć obrazy z fotografii 37, które pokazuje *głębię ostrości*: niektóre części obiektu są widoczne ostro, inne nie. Szczegóły można znaleźć w pracy [POTM82]. Inne metody umożliwiają korzystanie ze specjalnych efektów, np. rybie oko. Brak efektów głębi ostrości jest częściową przyczyną surrealistycznego wyglądu wielu wczesnych obrazów generowanych komputerowo.

Na obrazie zdjętym za pomocą statycznej albo ruchomej kamery obiekty ruchome wyglądają inaczej niż stacjonarne. Ponieważ migawka jest otwarta przez skończony czas, widoczne części ruchomych obiektów są na płaszczyźnie filmu rozmyte. Ten efekt określany jako *rozmycie ruchu* może być symulowany w sposób przekonywający [KORE83]. Efekt rozmycia ruchu nie tylko symuluje ruch na zdjęciu, ale jest niesłychanie ważny w animacji dobrej jakości; problem ten jest opisany w rozdz. 21 książki [FOLE90].

12.5. Ulepszone modele obiektów

Niezależnie od używanej technologii renderingu, dążenie do realizmu skoncentrowało się, między innymi, na sposobach budowania lepszych modeli zarówno statycznych, jak i dynamicznych. Niektórzy badacze opracowali modele dla specjalnych rodzajów obiektów, takich jak gazy, fale, góry i drzewa; przykłady pokazano na fot. 11, 12, 13. Metody tworzenia takich obiektów wykorzystują techniki fraktalne, gramatyki i systemy cząstek. Inni badacze skoncentrowali się na zaawansowanych metodach modelowania z wykorzystaniem funkcji sklejanych, na modelach proceduralnych, na renderingu objętościowym, na modelowaniu fizycznym i na modelowaniu ludzi. Innym ważnym zagadnieniem jest automatyzowanie rozmieszczania dużej liczby obiektów, np. drzew w lesie, co byłoby trudne do zrealizowania ręcznego. Witkin i Kass [WITK88] opisują metodę automatycznego rozmieszczania, jaką wykorzystali do animowania modelu *Luxo Jr.* Niektóre z metod automatycznego rozmieszczania są omawiane w p. 9.5; dokładne omówienie można znaleźć w rozdz. 20 książki [FOLE90].

12.6. Dynamika i animacja

12.6.1. Znaczenie ruchu

Przez pojęcie *dynamiki* rozumiemy zmiany, które występują w sekwencji obrazów, włączając w to zmiany pozycji, wielkości, właściwości materiałów, oświetlenia i parametrów obserwacji – w istocie zmiany dowolnej części sceny albo wykorzystanych metod. Korzyści z dynamiki mogą być sprawdzone niezależnie od postępu w kierunku bardziej realistycznych obrazów statycznych.

Być może, najpopularniejszym rodzajem dynamiki jest dynamika ruchu, poczynając od prostych przekształceń wykonywanych pod kontrolą użytkownika do złożonych animacji. Ruch jest ważnym elementem grafiki komputerowej od początku jej rozwoju. Początkowo, gdy sprzęt grafiki komputerowej był wolny, możliwość ruchu była jednym z silniejszych argumentów komercyjnych systemów grafiki wektorowej. Jeżeli szybko wyświetli się ciąg rzutów tego samego obiektu, przy czym każdy jest z nieco innego punktu obserwacji wokół obiektu, to uzyskuje się wrażenie obrotu obiektu. Integrując informację z rzutów, obserwator tworzy hipotezę o obiekcie.

Na przykład rzut perspektywiczny obracającego się sześcianu dostarcza różnego rodzaju informacji. Jest tu ciąg różnych rzutów, z których każdy niezależnie jest użyteczny. Ta informacja jest uzupełniana przez efekt ruchu, w którym maksymalna prędkość liniowa punktów w pobliżu środka obrotu jest mniejsza niż prędkość punktu odległego od środka obrotu. Ta różnica może pomóc wyjaśnić względną odległość punktu od środka obrotu. Zmienia również wielkości różnych części sześcianu wraz ze zmianą odległości, przy rzucie perspektywicznym daje dodatkowe wskazówki co do głębokości. Ruch staje się jeszcze ważniejszy, gdy znajdzie się pod interakcyjną kontrolą obserwatora. Dokonując selektywnego obracania obiektu, użytkownik może szybciej utworzyć hipotezę na temat obiektu.

W przeciwieństwie do stosowania prostych przekształceń w celu poznania złożonych modeli zaskakująco proste modele wyglądają bardzo przekonująco, jeżeli poruszają się w realistyczny sposób. Na przykład zaledwie kilka punktów umieszczonych w kluczowych punktach modelu człowieka, przy starannym poruszaniu, może dać przekonujące złudzenie ruchu osoby. Same punkty nie wyglądają jak osoba, informują natomiast obserwatora o jej obecności. Wiadomo również, że ruchome obiekty mogą być odtwarzane za pomocą mniejszej liczby szczegółów, niż to jest potrzebne do reprezentowania obiektów statycznych, ponieważ obserwator ma znacznie większe trudności w wyławianiu

szczegółów, jeżeli obiekt jest w ruchu. Na przykład telewidzowie są często zaskoczeni stwierdzając, jak źle i ziarniście wygląda jedna ramka telewizyjna.

12.6.2. Animacja

Animować to w dosłownym znaczeniu ożywiać. Chociaż często myśli się o animacji jak o synonimie ruchu, pojęcie to obejmuje wszystkie zmiany, które dają efekt wizualny. Obejmuje więc ono zmiany pozycji w czasie (dynamika ruchu), kształtu, barwy, przezroczystości, struktury i tekstury obiektu (dynamika uaktualniania) i zmiany oświetlenia, położenia kamery, jej orientacji i ustawienia ostrości, a nawet zmiany metody renderingu.

Animacja jest używana powszechnie w przemyśle rozrywkowym, a także w edukacji, zastosowaniach przemysłowych, np. sterowanie systemami, i w hełmach z monitorami, symulatorach lotu i w badaniach naukowych. Naukowe zastosowania grafiki komputerowej, a zwłaszcza zaś animacji są określane jako *wizualizacja naukowa*. Jednak wizualizacja to coś więcej niż tylko zastosowanie grafiki w nauce i inżynierii; obejmuje również takie dziedziny jak przetwarzanie sygnałów, geometria obliczeniowa i teoria baz danych. Często animacje w wizualizacji naukowej są generowane na podstawie symulacji zjawisk fizycznych. Wynikami symulacji mogą być ogromne zbiory danych 2D i 3D (na przykład w przypadku symulacji przepływu cieczy); te dane są zamieniane na obrazy, które potem tworzą animację. Symulacja może jednak generować położenia i miejsca obiektów fizycznych, które muszą być potem poddane jakiegoś rodzaju renderingowi w celu utworzenia animacji. Tak jest na przykład w symulacjach chemicznych, gdzie położenie i orientacja różnych atomów w reakcji mogą być generowane za pomocą symulacji, ale animacja może pokazać widoki składające się z kul i odcinków każdej cząsteczki albo może pokazać nakładające się gładko pocieniowane kule reprezentujące poszczególne atomy. W niektórych przypadkach program symulacji może zawierać w sobie język animacji i procesy symulacji oraz animacji mogą przebiegać równocześnie.

Jeżeli jakiś element animacji zmienia się zbyt szybko w stosunku do liczby animowanych ramek wyświetlanych w ciągu sekundy, to pojawia się efekt *zakłócania czasowego*. Klasyczny przykład to koła wagonu, które obracają się do tyłu albo gwałtowny ruch obiektu, który przesuwa się przez duże pole widzenia w krótkim czasie. Taśma wideo jest pokazywana z szybkością 30 ramek na sekundę, a film z szybkością 24 klatek na sekundę i w obu przypadkach uzyskuje się dobre wyniki dla wielu zastosowań. Oczywiście po to, żeby móc skorzystać z zalet takich szybkości, musimy tworzyć nowy obraz dla każdej ramki wideo albo filmu.

Jeżeli zamiast tego animator rejestruje każdy obraz na dwóch ramkach taśmy wideo, to efektywnie będzie 15 ramek na sekundę i ruch będzie poszarpany.

Tradycyjna animacja (to znaczy niekomputerowa) jest sama w sobie dyscypliną i tutaj omówimy tylko kilka z jej aspektów. Zajmiemy się natomiast podsumowaniem podstawowych koncepcji animacji komputerowej. Zaczniemy od omówienia konwencjonalnej animacji i metod stosowania komputerów do tworzenia tej tradycyjnej animacji. Następnie przejdziemy do animacji wytwarzanej w zasadzie przez komputer. Ponieważ w większości jest to animacja 3D, wiele z metod tradycyjnej animacji 2D już bezpośrednio nie da się zastosować. Ponadto sterowanie przebiegiem animacji jest trudniejsze, gdy animator nie rysuje sam obrazów animacji; często jest trudniej opisać, jak coś zrobić, niż samemu to wykonać. Dlatego po omówieniu języków animacji zajmiemy się kilkoma metodami sterowania animacją. Na końcu omówimy kilka ogólnych zasad animacji i problemów charakterystycznych dla animacji.

Animacja tradycyjna i wspomagana komputerem. Animacja tradycyjna jest tworzona w dość stałej sekwencji: jest pisany scenariusz animacji (albo przynajmniej szkicowany), a następnie jest projektowana sekwencja zmian. Tą sekwencją jest zarys animacji – sekwencja szkiców wysokiego poziomu pokazująca strukturę i ideę animacji. Następnie jest rejestrowana ścieżka dźwiękowa (jeżeli występuje), jest tworzony dokładny plan (z rysunkami dla każdej sceny w animacji) i jest czytana ścieżka dźwiękowa – to znaczy są rejestrowane kolejne chwile, w których pojawiają się istotne dźwięki. Następnie dokonuje się korelacji szczegółowego planu i ścieżki dźwiękowej. Następnie są rysowane określone *ramki kluczowe* – są to ramki, w których animowane obiekty są w swoich ekstremalnych albo charakterystycznych pozycjach; z ramek tych można wywnioskować o pośrednich położeniach. Następnie są wstawiane *ramki pośrednie* i jest tworzony próbny film. Ramki z tego próbnego filmu są przenoszone na *klisze* (arkusze filmu na podłożu acetylocelulozowym) albo na zasadzie ręcznego kopiowania tuszem, albo w wyniku fotokopiowania bezpośrednio na te klisze. Z kolei są one kolorowane albo zamalowywane i łączone w odpowiednią sekwencję; następnie są filmowane. Ze względu na korzystanie z ramek kluczowych i z ramek pośrednich ten typ animacji jest określany jako *animacja z ramkami kluczowymi*. Ta nazwa jest również stosowana w systemach komputerowych, które naśladują ten proces.

Wiele etapów animacji tradycyjnej wydaje się idealnie nadawać do wspomagania przez komputer, zwłaszcza do wyznaczania ramek pośrednich i kolorowania, które może być wykonywane metodą wypełniania wnętrza ze znanym punktem początkowym [SMIT79]. Zanim jednak komputer będzie mógł być użyty, rysunki muszą zostać przedstawione w postaci cyfrowej. Można to zrobić za pomocą skanowania

optycznego, śledzenia oryginalnych rysunków za pomocą tabliczki z odpowiednim wskaźnikiem albo wykorzystania programu rysującego do utworzenia oryginalnych obrazów. Może zaistnieć potrzeba wykonywania przetwarzania końcowego (na przykład filtrowania) w celu usunięcia zakłóceń, które mogły powstać w czasie procesu wprowadzania (zwłaszcza przy optycznym skanowaniu) i w celu pewnego wygładzenia konturów.

Języki animacji. Do opisywania animacji opracowano wiele różnych języków poczynając od specjalizowanych rozwiązań, a kończąc na pakietach proceduralnych do wykorzystania z konwencjonalnymi językami. [FOLE90, rozdz. 21.] Niektóre języki animacji są łączone z językami modelowania i opisywanie obiektów w animacji oraz animowanie obiektów wykonuje się w tym samym czasie.

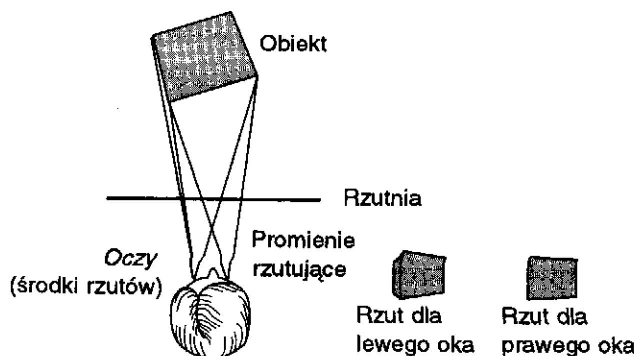
Metody sterowania animacją. Sterowanie animacją jest w pewnym sensie niezależne od języka używanego do jej opisu – większość mechanizmów sterujących może być adaptowana do wykorzystania z różnymi rodzajami języków. Wśród mechanizmów sterujących animacją są takie, które umożliwiają animatorowi bezpośrednie opisywanie położenia i atrybutów każdego obiektu w scenie za pomocą przesunięć, obrotów i innych operatorów zmieniających położenie i atrybuty, jak i takie z wysoce zautomatyzowanym sterowaniem zapewnianym przez systemy z bazą wiedzy, które na podstawie opisu animacji na wysokim poziomie ("niech postać wyjdzie z pokoju") generują bezpośrednie sterowanie, które wywołuje zmiany potrzebne do uzyskania animacji. Niektóre z bardziej zaawansowanych metod sterowania animacją opisano w książce [FOLE90].

Podstawowe zasady animacji. Tradycyjna postać przemysłowej animacji powstała w studio Walta Disneya między 1925 rokiem a późnymi latami trzydziestymi. Na początku animacja nie wymagała wiele więcej niż narysowania sekwencji statycznych obrazów, które razem tworzyły obrazy animowane. W miarę tworzenia metod animacji powstały pewne podstawowe zasady, które stały się zasadniczymi regułami animacji postaci i są wciąż używane [LAYB79; LASS87]. Wiele z zasad filmowej animacji postaci równie dobrze stosuje się do realistycznej animacji 3D. Te zasady, razem z ich zastosowaniem w animacji postaci 3D, są omówione w pracy [LASS87]. Trzeba jednak zauważyć, że te reguły nie są obligatoryjne. Tak jak wiele z nowoczesnej sztuki odbiega od tradycyjnych reguł rysowania, tak wielu współczesnych animatorów odeszło od tradycyjnych reguł animacji, często ze wspaniałymi wynikami (por. na przykład prace [LEAF74; LAEAF77]). Wsród reguł są takie jak: *zgniataj i rozciągaj*, która pokazuje fizyczne właściwości obiektu dzięki zniekształcaniu jego kształtu; ruchy typu *zwolnij i przyspiesz w celu uzyskania gładkich przejść; i właściwa inscenizacja* albo wybieranie rzutu, który pokazuje większość informacji o zdarzeniach występujących w czasie animacji.

Specyficzne problemy animacji. Tak jak przejście od grafiki 2D do grafiki 3D wprowadziło wiele nowych problemów i możliwości, zmiana z 3D na 4D (jest dodany wymiar czasu) również stwarza problemy. Jednym z tych problemów są *zakłócenia czasowe*. Podobnie jak problemy zakłóceń w grafice 2D i 3D są częściowo rozwiązywne przez zwiększenie rozdzielczości ekranu, problemy zakłóceń czasowych w animacji mogą być częściowo rozwiązane przez zwiększenie rozdzielczości czasowej. Oczywiście innym aspektem rozwiązania 2D są metody antyzakłóceniowe; odpowiednikiem takiego rozwiązania w 3D są czasowe metody antyzakłóceniowe.

12.7. Stereopsja

Wszystkie dotychczas omówione metody prezentują ten sam obraz dla każdego oka obserwatora. Wykonajmy teraz eksperyment: popatrzmy najpierw na stół jednym okiem a potem drugim. Oba widoki nieco różnią się od siebie, co wynika stąd, że nasze oczy są oddalone od siebie o kilka centymetrów (rys. 12.7). Ta różnica wynikająca z rozstawienia oczu daje istotną informację na temat głębokości określaną jako *stereopsja* albo *stereowizja*. Nasz mózg łączy oba obrazy w jeden interpretowany jako obraz 3D. Te dwa obrazy są określane jako *para stereo*; pary stereo były używane w fotoplastykonach, które były popularne na początku wieku i są używane dzisiaj w popularnych kalejdoskopach. Na fotografii 22 pokazano parę obrazów stereo cząsteczki. Można połączyć te dwa obrazy w jeden obraz 3D patrząc na nie w taki sposób, żeby każde oko widziało tylko jeden obraz; można to zrobić na przykład umieszczając sztywny kawałek papieru między obrazami i prostopadle do kartki. Niektórzy potrafią zauważyć efekt nawet bez konieczności używania kawałka papieru, a pewna liczba osób nie potrafi w ogóle zauważyć tego efektu.



Rys. 12.7. Widzenie dwuoczne

Istnieją również inne metody kierowania różnych obrazów do każdego oka, włączając w to okulary z polaryzującymi filtrami oraz holografię. Niektóre z tych metod umożliwiają uzyskanie prawdziwych obrazów 3D, które zajmują miejsce w przestrzeni, a nie tylko są rzutowane na jedną płaszczyznę. Takie wyświetlacze dostarczają dodatkowych wskazówek na temat głębokości: bliższe obiekty rzeczywiście są bliższe, tak jak w otaczającym nas świecie, i oczy obserwatora ogniskują się różnie na różnych obiektach, zależnie od bliskości każdego obiektu. Matematyka związana z rzutami stero jest opisana w zadaniu 6.17.

12.8. Ulepszone wyświetlacze

Prócz ulepszeń w programach wykorzystywanych do projektowania i renderingu obiektów iluzję rzeczywistości zwiększyły również same urządzenia wyświetlające. Historia grafiki komputerowej jest po części związana ze stałym ulepszaniem jakości wizualnej osiąganej przez urządzenia wyświetlające. Ciągle jednak gamy barw nowoczesnych monitorów i ich dynamiczny zakres jasności są małym podzbiorem tego, co możemy widzieć. Jeszcze czeka nas daleka droga, zanim obraz na naszym wyświetlaczu będzie porównywalny, jeśli chodzi o szczegółowość i kontrast z dobrze wydrukowaną profesjonalną fotografią! Ograniczona rozdzielczość wyświetlacza uniemożliwia reprodukowanie bardzo małych szczegółów. Zakłócenia takie jak: widoczne wzory luminoforu, blask od ekranu, zakłócenia geometryczne i efekt stroboskopowy migotania z częstotliwością wyświetlania ramek stale przypominają o tym, że obserwujemy ekran. Stosunkowo mała wielkość ekranu w porównaniu z naszym polem widzenia również przypomina nam, że ekran jest raczej oknem na świat niż samym światem.

12.9. Interakcja z innymi zmysłami

Być może, ostatnim krokiem w kierunku realizmu jest integracja realistycznych obrazów z informacją przekazywaną do innych naszych zmysłów. Grafika komputerowa ma długą historię programów zależnych od różnych urządzeń wejściowych, które umożliwiały interakcję z użytkownikiem. Symulatory lotu są przykładem sprzężenia grafiki z realistycznym dźwiękiem i ruchem samolotu, wszystko oferowane w makiecie kokpitu w celu stworzenia całego środowiska. Noszenie hełmu, który monitoruje ruch głowy, umożliwia wprowadzenie jeszcze innej wskazówki na temat głębokości 3D określanej jako *paralaksa ruchu głowy*:

gdy użytkownik porusza głową z boku na bok, być może w celu obejrzenia ukrytych obiektów, obraz zmienia się jak w rzeczywistości. Inne bieżące prace koncentrują się na badaniu *sztucznych światów*, takich jak wnętrze cząsteczek albo budynków, które jeszcze nie zostały skonstruowane [CHUN89].

W wielu współczesnych salonach gier występują samochody albo samoloty, którymi użytkownik może jechać albo lecieć przesuwając się w funkcji czasu w ramach symulacji, która obejmuje syntetyzowane albo przedstawione w postaci cyfrowej obrazy, dźwięk i sprzężenie zwrotne. Takie zastosowanie dodatkowego wyjścia i wejścia wskazuje na to, jakie możliwości w przyszłości będą miały systemy, które zapewnią całkowite zanurzenie dla wszystkich zmysłów, w tym słuchu, dotyku, smaku i węchu.

Podsumowanie

W rozdziale przedstawiliśmy wprowadzenie, na wysokim poziomie, do metod używanych przy tworzeniu realistycznych obrazów. W następnych rozdziałach omówimy szczegółowo, jak można te metody implementować. Są cztery kluczowe pytania, o których trzeba pamiętać czytając o algorytmach przedstawionych w dalszych rozdziałach:

1. Czy algorytm jest ogólny czy specjalizowany? Niektóre metody działają lepiej tylko w określonych okolicznościach; inne są projektowane jako bardziej ogólne. Na przykład w niektórych algorytmach zakłada się, że wszystkie obiekty są wypukłymi wielościanami i ich szybkość oraz względna prostota wynikają z tego założenia.
2. Jakie są parametry czasowo-przestrzenne algorytmu? Jak na algorytm wpływają takie czynniki, jak wielkość, złożoność bazy danych albo rozdzielczość, z jaką jest wykonywany rendering obrazu?
3. Jak przekonywające są generowane efekty? Na przykład, czy załamanie jest modelowane poprawnie, czy wygląda dobrze tylko w określonych specjalnych przypadkach, czy też w ogóle nie daje się modelować? Czy można dodać efekty takie jak cienie albo odbicia zwierciadlane? Jak przekonywająco będą wyglądały? Zmniejszenie dokładności renderingu może dać istotne polepszenie, jeśli chodzi o zajętość pamięci i wymagania czasowe.
4. Czy algorytm jest poprawny ze względu na tworzenie obrazu? Filozofia kryjąca się za wieloma obrazami w następnych rozdziałach może być podsumowana przez kredo: „jeżeli wygląda dobrze, to należy tak zrobić!" Ta dyrektywa może być interpretowana dwojako. Prosty albo szybki algorytm może być użyty, jeżeli daje atrakcyjne efekty,

nawet jeżeli nie można znaleźć uzasadnienia w prawach fizyki. Wyjątkowo kosztowny algorytm może być użyty, jeżeli jest to jedyny znany sposób dla renderingu pewnych efektów.

Zadania

12.1. Załóż, że jest do dyspozycji system graficzny, który mógłby narysować każdą z fotografii, do której było odwołanie w tym rozdziale, w czasie rzeczywistym. Weź pod uwagę niektóre ze znanych obszarów zastosowań. Dla każdego z nich wypisz te efekty, które byłyby najbardziej użyteczne i te które byłyby mniej użyteczne.

12.2. Pokaż, że nie można wywnioskować kierunku obrotu z rzutów prostokątnych monochromatycznego, obracającego się sześcianu drutowego. Wyjaśnij, jak dodatkowe metody mogą pomóc w określaniu kierunku obrotu bez zmiany rzutu.

13. Wyznaczanie powierzchni widocznych

Dla danego zbioru obiektów i specyfikacji parametrów widzenia chcemy określić, które linie albo powierzchnie obiektów są widoczne albo ze środka rzutowania (dla rzutu perspektywicznego), albo wzdłuż kierunku rzutowania (dla rzutu równoległego), tak żeby było możliwe wyświetlenie tylko linii albo powierzchni widocznych. Ten proces jest znany jako wyznaczanie *linii* albo *powierzchni widocznych* albo też eliminowanie *linii* albo *powierzchni niewidocznych*. Przy wyznaczaniu linii widocznych zakłada się, że linie są krawędziami powierzchni nieprzezroczystych, które mogą zasłaniać krawędzie innych powierzchni znajdujących się dalej od obserwatora. Dlatego mówimy o ogólnym procesie jako o *wyznaczaniu powierzchni widocznych.*

Chociaż ta podstawowa zasada jest prosta, jej realizacja wymaga znacznej mocy obliczeniowej i w konsekwencji wymaga dużego czasu obliczeniowego na konwencjonalnych maszynach. Te wymagania doprowadziły do opracowania licznych algorytmów wyznaczania powierzchni widocznych, spośród których wiele opisujemy w tym rozdziale. Ponadto zaprojektowano wiele specjalizowanych architektur związanych z rozwiązywaniem tego problemu; niektóre z nich są omawiane w rozdz. 18 książki [FOLE90]. Potrzebę poświęcenia takiej uwagi temu problemowi można zauważyć analizując dwa podstawowe podejścia do rozwiązania tego problemu. W obu przypadkach możemy traktować obiekt jako składający się z jednego albo więcej wielokątów (albo bardziej złożonych powierzchni).

W pierwszym podejściu określa się, który z n obiektów jest widoczny w każdym pikselu obrazu. Pseudokod dla tego podejścia wygląda następująco:

```
for (każdy piksel obrazu) {
    wyznaczyć obiekt najbliższy obserwatora, który jest napotkany przez promień
        rzutowania przechodzący przez piksel;
    narysować piksel o odpowiedniej barwie;
}
```

Bezpośredni sposób wykonania tego dla jednego piksela wymaga sprawdzenia wszystkich *n* obiektów w celu ustalenia, który leży najbliżej obserwatora wzdłuż promienia rzutowania przechodzącego przez piksel. Dla *p* pikseli nakład jest proporcjonalny do *np*, przy czym *p* jest większe niż 1 milion dla monitora o dużej rozdzielczości.

Drugie podejście polega na porównywaniu obiektów bezpośrednio każdy z każdym i eliminowaniu tych obiektów albo ich części, które nie są widoczne. Można to przedstawić za pomocą pseudokodu:

```
for (każdy obiekt) {
    wyznaczyć te części obiektu, których rzut nie jest zasłonięty przez inne części
        tego lub innych obiektów;
    narysować te części z wykorzystaniem odpowiedniej barwy;
}
```

Możemy to wykonać w sposób bezpośredni, porównując każdy z *n* obiektów ze sobą i z innymi obiektami i odrzucać niewidoczne części. Nakład obliczeniowy jest tu proporcjonalny do n^2. Chociaż to drugie podejście mogłoby się wydawać lepsze dla $n < p$, jak zobaczymy, jego poszczególne kroki są na ogół bardziej złożone, zajmują więcej czasu i w efekcie metoda często jest wolniejsza i trudniejsza w implementacji.

Te dwa prototypowe podejścia będziemy określali odpowiednio jako *algorytm z precyzją obrazową* i *algorytm z precyzją obiektową*. Algorytmy z precyzją obrazową są na ogół wykonywane z dokładnością, jaką ma urządzenie wyświetlające, i określają widoczność w każdym pikselu. Algorytmy z precyzją obiektową są wykonywane z dokładnością, z jaką jest zdefiniowany każdy obiekt, i określają widoczność każdego obiektu[1]. Ponieważ obliczenia z precyzją obiektową są wykonywane bez względu

[1] Pojęcia *przestrzeń obrazu* i *przestrzeń obiektu*, spopularyzowane przez Sutherlanda, Sproulla i Schumackera [SUTH74a], często są używane dla tego samego rozróżnienia. Niestety w grafice komputerowej te określenia były również używane zupełnie inaczej. Na przykład przestrzeń obrazu była używana w odniesieniu do obiektów po przekształceniu perspektywicznym [CATM75] albo po rzutowaniu na powierzchnię widoczną [GILO78], ale wciąż z ich oryginalną precyzją. W celu uniknięcia nieporozumień opowiadamy się za nieco zmodyfikowanymi określeniami. Odnosimy się bezpośrednio do przekształcenia perspektywicznego albo do rzutowania wówczas, gdy to jest właściwe, i rezerwujemy określenie precyzja obrazowa i precyzja obiektowa w celu pokazania dokładności, z jaką są wykonywane obliczenia. Na przykład przecinanie rzutów dwóch obiektów na rzutni jest operacją z precyzją obiektową, jeżeli jest utrzymana precyzja definicji oryginalnego obiektu.

na rozdzielczość określonego wyświetlacza, musi po nich następować krok, w którym obiekty są faktycznie wyświetlane z wymaganą rozdzielczością. Tylko ten ostatni krok wyświetlania musi być powtarzany, jeżeli trzeba zmienić na przykład wielkość skończonego obrazu po to, żeby pokryć inną liczbę pikseli na monitorze rastrowym. Wynika to stąd, że geometria każdego rzutu obiektu widocznego jest reprezentowana z pełną rozdzielczością bazy danych obiektu. Dla kontrastu rozważmy powiększanie obrazu utworzonego przez algorytm z precyzją obrazu. Ponieważ obliczenia związane z powierzchnią widoczną były wykonane początkowo z małą rozdzielczością, muszą być wykonane ponownie, jeżeli chcemy pokazać dalsze szczegóły. Dlatego algorytmy z precyzją obrazową są narażone na powstawanie zakłóceń przy wyznaczaniu widoczności, podczas gdy algorytmy z precyzją obiektową nie.

Algorytmy z precyzją obiektową były pierwotnie opracowane dla wektorowych systemów graficznych. Na tych urządzeniach usuwanie linii niewidocznych było wykonywane prawie naturalnie dzięki zamienianiu początkowej listy odcinków na listę, z której były usunięte odcinki całkowicie zasłonięte przez inne powierzchnie, a częściowo zasłonięte odcinki były obcinane do jednego lub kilku widocznych segmentów. Całe przetwarzanie było wykonywane z precyzją oryginalnej listy i prowadziło do listy o takim samym formacie. Dla kontrastu, algorytmy z precyzją obrazową były najpierw napisane dla urządzeń rastrowych z myślą o wykorzystaniu małej liczby pikseli, dla których trzeba było wykonywać obliczenia związane z widocznością. Był to zrozumiały podział. Wyświetlacze wektorowe miały duże przestrzenie adresowe (4096 × 4096 nawet we wczesnych systemach) i ostre ograniczenia, jeśli chodzi o liczbę odcinków i obiektów, które mogły być wyświetlane. Wyświetlacze rastrowe miały ograniczoną przestrzeń adresową (256 × 256 we wczesnych systemach) i zdolność do wyświetlania potencjalnie nieograniczonej liczby obiektów. W późniejszych algorytmach często łączono obliczenia z precyzją obiektową z obliczeniami z precyzją obrazową, przy czym obliczenia z precyzją obiektową były wybierane ze względu na dokładność, a obliczenia z precyzją obrazową ze względu na szybkość.

W tym rozdziale najpierw omówimy kilka problemów związanych w efektywnością ogólnych algorytmów wyznaczania powierzchni widocznych. Następnie przedstawimy różne podejścia do wyznaczania powierzchni widocznych.

13.1. Metody dla efektywnych algorytmów wyznaczania powierzchni widocznych

Przy formułowaniu typowych algorytmów z precyzją obrazową i z precyzją obiektową może być potrzebnych kilka potencjalnie kosztownych

operacji. Do tych operacji należy określenie dla promienia rzutowania i obiektu, albo dwóch rzutów obiektu, czy się przecinają i gdzie się przecinają. Następnie dla każdego zbioru przecięć trzeba wyznaczyć obiekt, który jest najbliżej obserwatora. W celu zminimalizowania czasu potrzebnego do utworzenia obrazu musimy tak organizować algorytmy wyznaczania powierzchni widocznych, żeby kosztowne operacje były wykonywane tak efektywnie i rzadko, jak to jest tylko możliwe. W następnych punktach opisano kilka ogólnych sposobów wykonania tego zadania.

13.1.1. Spójność

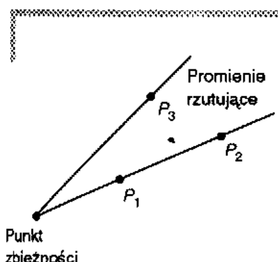
Sutherland, Sproull i Schumacker [SUTH74a] pokazali, jak algorytmy wyznaczania powierzchni widocznych mogą wykorzystywać *spójność* – stopień, w jakim fragmenty sceny albo ich rzuty wykazują lokalne podobieństwa. Sceny na ogół zawierają obiekty, których właściwości zmieniają się gładko od jednej części do drugiej. W rzeczywistości jest to najrzadziej występująca nieciągłość właściwości (takich jak głębokość, barwa i tekstura) i efektów, które one wytworzą w obrazach, umożliwiających rozróżnienie obiektów. Spójność wykorzystujemy wówczas, gdy ponownie korzystamy z obliczeń wykonywanych dla jednej części otoczenia albo obrazu w stosunku do innych bliskich części, albo bez zmian albo z przyrostowymi zmianami, które są bardziej efektywne do wykonania niż obliczenie informacji na nowo. Zidentyfikowano wiele różnych rodzajów spójności [SUTH74a]; wymieniamy je niżej i będziemy się do nich odwoływali później:

- **Spójność obiektów.** Jeżeli jeden obiekt jest całkowicie odseparowany od drugiego, to może być potrzebne wykonanie porównań tylko między tymi dwoma obiektami, a nie między ich ścianami składowymi albo krawędziami. Na przykład, jeżeli wszystkie części obiektu A są dalej od obserwatora niż wszystkie części obiektu B, to żadna ze ścian A nie musi być porównywana ze ścianami B w celu sprawdzenia, czy zasłaniają one ściany B.
- **Spójność ściany.** Właściwości powierzchni na ogół zmnieniają się gładko na powierzchni, dzięki czemu obliczenia dla jednej części ściany można modyfikować przyrostowo w celu zastosowania do sąsiednich części. W niektórych modelach może istnieć gwarancja, że ściany nie przenikają się.
- **Spójność krawędzi.** Widoczność krawędzi może zmieniać się tylko tam, gdzie przecina widoczną krawędź z tyłu albo przecina widoczną ścianę.

- **Implikowana spójność krawędzi.** Jeżeli jedna płaska ściana przecina drugą, to ich linia przecięcia (implikowana krawędź) może być określona na podstawie dwóch punktów przecięcia.
- **Spójność liniowa.** Zbiór widocznych odcinków obiektu określony dla jednej linii obrazu na ogół niewiele się różni od zbioru dla poprzedniej linii.
- **Spójność powierzchni.** Grupa sąsiednich pikseli jest często pokryta przez tę samą widoczną ścianę. Specjalnym rodzajem spójności powierzchni jest *spójność segmentowa*, która odnosi się do widoczności ściany w obrębie segmentu sąsiednich pikseli w linii.
- **Spójność głębokości.** Sąsiednie części tej samej powierzchni mają na ogół podobną odległość od obserwatora (głębokość), podczas gdy różne powierzchnie w tym samym miejscu ekranu mają na ogół znacznie różniące się głębokości. Po obliczeniu głębokości jednego punktu na powierzchni, głębokość punktów pozostałej części powierzchni może być często określona za pomocą prostego równania różnicowego.
- **Spójność ramek.** Jest prawdopodobne, że obrazy tej samej sceny w dwóch kolejnych chwilach są bardzo podobne, mimo niewielkich zmian obiektów i punktu obserwacji. Obliczenia wykonane dla jednego obrazu mogą być użyte dla kolejnego obrazu w sekwencji.

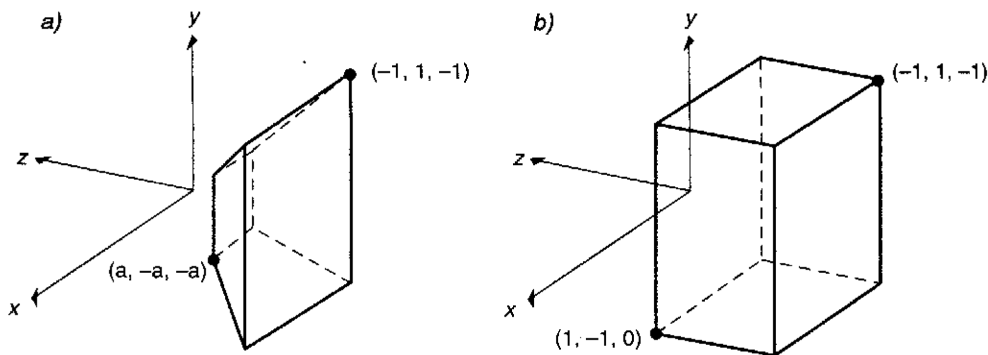
13.1.2. Przekształcenie perspektywiczne

Określenie powierzchni widocznych musi nastąpić oczywiście w przestrzeni 3D przed rzutowaniem na płaszczyznę 2D, które niszczy informację o głębokości potrzebną dla porównań głębokości. Niezależnie od rodzaju wybranego rzutu podstawowe porównanie głębokości w punkcie może być na ogół sprowadzone do następującego pytania: czy jeden z dwóch punktów $P_1 = (x_1, y_1, z_1)$ i $P_2(x_2, y_2, z_2)$ zasłania drugi (rys. 13.1)? Albo czy P_1 i P_2 leżą na tym samym promieniu rzutującym? Jeżeli tak, to porównuje się z_1 i z_2 w celu określenia, który z punktów jest bliżej obserwatora. Jeżeli nie, to żaden punkt nie zasłania drugiego.

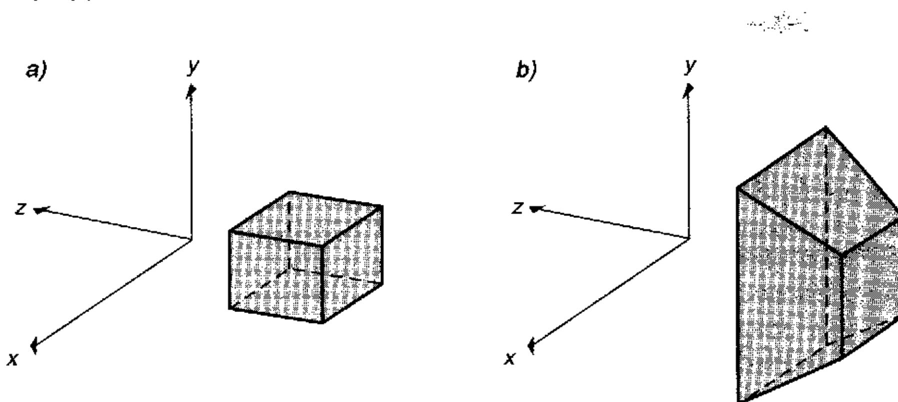


Rys. 13.1. Jeżeli dwa punkty P_1 i P_2 są na tym samym promieniu, to bliższy punkt zasłania dalszy; w przeciwnym przypadku nie (na przykład P_1 nie zasłania P_3)

Porównania głębokości są na ogół wykonywane po zastosowaniu przekształcenia normalizującego (rozdz. 6), tak że promienie rzutujące są równoległe do osi z w rzucie równoległym albo wychodzą z punktu zbieżności dla rzutu perspektywicznego. W rzucie równoległym punkty są na tym samym promieniu rzutowania, jeżeli $x_1 = x_2$ i $y_1 = y_2$. Dla rzutu perspektywicznego musimy, niestety, wykonać cztery dzielenia w celu sprawdzenia, czy $x_1/z_1 = x_2/z_2$ i $y_1/z_1 = y_2/z_2$; jeśli tak, to punkty leżą na tym samym promieniu rzutowania (rys. 13.1). Co więcej, jeżeli P_1 będzie porównywane dalej z P_3, to będą potrzebne jeszcze dwa dzielenia.



Rys. 13.2. Znormalizowana bryła widzenia perspektywicznego przed (a) i po przekształceniu perspektywicznym (b)



Rys. 13.3. Sześcian przed (a) i po przekształceniu perspektywicznym (b)

Niepotrzebnych dzieleń można uniknąć dokonując najpierw przekształcenia obiektu 3D w układ współrzędnych ekranu 3D, tak żeby rzut równoległy przekształconego obiektu był taki sam jak rzut perspektywiczny nieprzekształconego obiektu. Wtedy test sprawdzający przesłanianie punktów jest taki sam jak w rzucie równoległym. To przekształcenie perspektywiczne zniekształca obiekty i przesuwa środek rzutowania do nieskończoności na osi z i promienie rzutowania stają się równoległe. Na rysunku 13.2 pokazano skutek tego przekształcenia w bryłę rzutu perspektywicznego. Na rysunku 13.3 pokazano zniekształcenie, jakiemu ulega sześcian przy tym przekształceniu.

Istotą tego przekształcenia jest to, że zachowuje względną głębokość, odcinki i płaszczyzny i równocześnie wykonuje perspektywiczny skrót. Dzielenie prowadzące do tego skrótu jest wykonywane tylko raz dla punktu, a nie za każdym razem, gdy dwa punkty są porównywane. Macierz

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{min}} & \frac{-z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad z_{min} \neq -1 \qquad (13.1)$$

przekształca znormalizowaną bryłę widzenia perspektywicznego w prostopadłościan ograniczony przez:

$$-1 \leqslant x \leqslant 1, \quad -1 \leqslant y \leqslant 1, \quad -1 \leqslant z \leqslant 0 \qquad (13.2)$$

Przed zastosowaniem macierzy M można wykonać obcinanie względem znormalizowanej bryły widzenia, czyli ostrosłupa ściętego, ale wtedy wyniki obcinania musza być pomnożone przez M. Atrakcyjniejsze rozwiązanie polega na włączeniu M do normalizującego przekształcenia perspektywicznego N_{per} z rozdz. 6, tak że potrzebne jest tylko jedno mnożenie przez macierz, a potem obcinanie we współrzędnych jednorodnych przed dzieleniem. Jeżeli wyniki mnożenia oznaczymy jako (X, Y, Z, W), to dla $W > 0$ granice obcinania będą następujące:

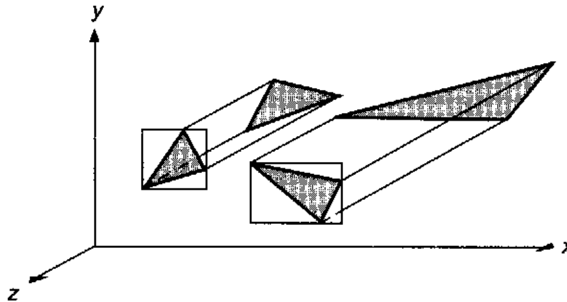
$$-W \leqslant X \leqslant W, \quad -W \leqslant Y \leqslant W, \quad -W \leqslant Z \leqslant 0 \qquad (13.3)$$

Te ograniczenia wynikają z równania (13.2), w którym zastąpi się x, y i z odpowiednio przez X/W, Y/W i Z/W po to, żeby uwzględnić fakt, że x, y, i z w równaniu (13.2) wynikają z dzielenia przez W. Po obcinaniu dzielimy przez W po to, żeby otrzymać (x_p, y_p, z_p). Zauważmy, że macierz M została zapisana przy założeniu, że bryła widzenia jest w ujemnej półprzestrzeni z. Ze względu jednak na wygodę notacyjną w naszych przykładach będą często używane raczej malejące dodatnie wartości z niż malejące ujemne wartości z po to, żeby pokazać rosnącą odległość od obserwatora. W wielu systemach graficznych zamiast prawoskrętnego układu współrzędnych korzysta się z lewoskrętnego układu współrzędnych widzenia, w którym rosnące dodatnie wartości z odpowiadają zwiększającej się odległości od obserwatora.

Możemy teraz wyznaczyć powierzchnie widoczne bez obawy przed komplikacjami sugerowanymi przez rys. 13.1. Oczywiście w przypadku rzutu równoległego przekształcenie perspektywiczne M nie jest konieczne, ponieważ przekształcenie normalizujące N_{par} dla rzutu równoległego powoduje, że promienie rzutowania są równoległe do osi z.

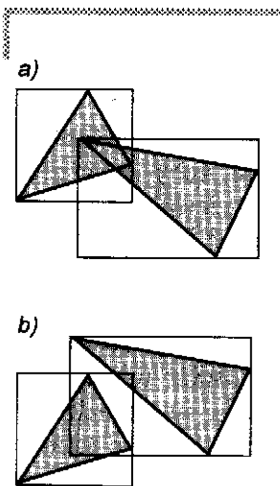
13.1.3. Prostokąty i bryły ograniczające

Prostokąty ograniczające wprowadzone w rozdz. 3 w celu uniknięcia niepotrzebnego obcinania są również często używane, aby uniknąć niepotrzebnych porównań obiektów albo ich rzutów. Na rysunku 13.4 pokazano dwa obiekty (w tym przypadku wielokąty 3D), ich rzuty i prostokąty opisane na nich o bokach równoległych do osi. Zakłada się, że



Rys. 13.4. Dwa obiekty i ich rzuty na płaszczyznę (x, y) i prostokąty opisane na tych rzutach

obiekty mają być poddane przekształceniu perspektywicznemu za pomocą macierzy M z p. 13.1.2. Dla wielokątów rzut prostokątny na płaszczyznę (x, y) jest wykonywany w sposób trywialny na zasadzie pominięcia współrzędnych z wierzchołków. Na rysunku 13.4 prostokąty ograniczające nie nachodzą na siebie i nie trzeba wykonywać testowania rzutów ze względu na przecinanie się ich ze sobą. Jeżeli prostokąty ograniczające przecinają się, to może wystąpić jedna z dwóch sytuacji pokazanych na rys. 13.5: albo rzuty przecinają się jak na rys. a), albo nie przecinają się jak na rys. b). W obu przypadkach trzeba wykonać więcej porównań w celu sprawdzenia, czy rzuty się przecinają. W sytuacji z rysunku b) wynik porównań pokaże, że dwa rzuty nie przecinają się; oznacza to, że fakt przecinania się prostokątów ograniczających był fałszywym alarmem. Testowanie prostokątów ograniczających daje w efekcie to samo co w przypadku testów odrzucania przy obcinaniu.

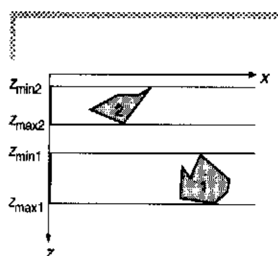


Rys. 13.5. Prostokąty ograniczające rzuty obiektów: a) prostokąty ograniczające i rzuty nachodzą na siebie; b) prostokąty ograniczające nachodzą na siebie, natomiast rzuty nie

Prostokąty ograniczające mogą być używane, jak w rozdziale 7, do otaczania samych obiektów, a nie ich rzutów: w takim przypadku miejsce prostokąta zajmuje bryła określana jako *bryła ograniczająca*. Można także koncepcję protokąta ograniczającego przenieść do jednego wymiaru w celu określenia, czy na przykład dwa obiekty nachodzą na siebie we współrzędnej z. Na rysunku 13.6 pokazano wykorzystanie koncepcji w takim przypadku; tutaj obszar ograniczający jest nieskończoną objętością ograniczającą wartości minimalną i maksymalną z dla

każdego obiektu. Nie ma nakładania się obiektów we współrzędnej z, jeżeli:

$$z_{\max 2} < z_{\min 1} \text{ lub } z_{\max 1} < z_{\min 2} \tag{13.4}$$



Rys. 13.6. Wykorzystanie obszarów ograniczających 1D do sprawdzania, czy obiekty przecinają się

Porównywanie wartości ograniczeń minimalnej i maksymalnej w jednym lub więcej wymiarach jest znane jako *testowanie minmax*. Przy porównywaniu typu minmax obszarów ograniczających najbardziej skomplikowną częścią pracy jest znalezienie samych obszarów ograniczających. Dla wielokątów (albo dla innych obiektów, które są całkowicie zawarte w wypukłym wielokącie opisanym na zbiorze definiujących punktów) obszar ograniczający można obliczyć przeszukując iteracyjnie listę współrzędnych punktów i rejestrując największe i najmniejsze wartości dla każdej współrzędnej.

Obszary ograniczające są używane nie tylko do porównywania dwóch obiektów albo ich rzutów, ale również do określania, czy promień rzutowania przecina obiekt. Wiąże się to z obliczaniem przecięcia punktu z rzutem 2D albo wektora z obiektem 3D (p. 13.4).

Chociaż dotychczas omawialiśmy tylko obszary ograniczające typu minmax, możliwe jest użycie innych brył ograniczających. Jakiej bryły użyć najlepiej? Nie może być zaskoczeniem, że odpowiedź zależy zarówno od kosztu wykonywania testów na samej bryle ograniczającej, jak i od tego, jak dobrze bryła zabezpiecza otaczany obiekt przed testami, które nie dają przecięcia. Weghorst, Hooper i Greenberg [WEGH84] traktują wybór bryły ograniczającej jako problem minimalizacji całkowitej funkcji kosztu T testu przecięcia dla obiektu. Można to wyrazić jako

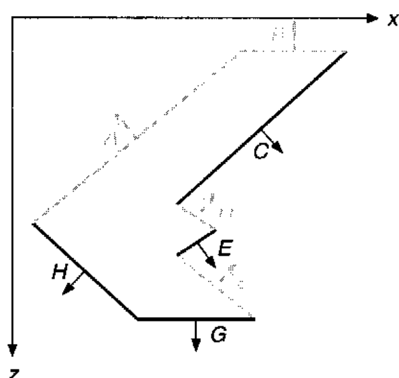
$$T = bB + oO \tag{13.5}$$

przy czym b jest liczbą określającą, ile razy bryła ograniczająca jest testowana ze względu na przecięcie, B jest kosztem wykonania testu przecięcia z bryłą ograniczającą, o jest liczbą określającą, ile razy obiekt jest testowany ze względu na przecięcie (ile razy rzeczywiście jest przecinana bryła ograniczająca), a O jest kosztem wykonania testu przecięcia obiektu.

Ponieważ test przecięcia obiektu jest wykonywany tylko wówczas, gdy rzeczywiście jest przecinana bryła ograniczająca, to $o \leqslant b$. Chociaż O i b są stałymi dla określonego obiektu i jest wykonywany zbiór testów, to B i o zmieniają się zależnie od kształtu i wielkości bryły ograniczającej. Im ciaśniejsza jest bryła ograniczająca, co minimalizuje o, tym na ogół jest większe B. Efektywność bryły ograniczającej może również zależeć od orientacji obiektu albo rodzaju obiektów, z którymi obiekt będzie przecinany.

13.1.4. Wybieranie tylnych ścian

Jeżeli obiekt jest aproksymowany przez wielościan, to jego ściany wielokątowe całkowicie otaczają jego wnętrze. Załóżmy, że wszystkie wielokąty zostały tak zdefiniowane, że ich normalne są skierowane na zewnątrz wielościanu. Jeżeli żadna część wnętrza wielościanu nie jest widoczna przez przednią płaszczyznę obcinającą, to te wielokąty, których normalne są skierowane do obserwatora, leżą w tej części wielościanu, której widoczność jest całkowicie zablokowana przez bliższe wielokąty (rys. 13.7). Takie *wielokąty niewidoczne* można wyeliminować z dalszego procesu za pomocą metody *wybierania tylnych ścian*. Przez analogię te wielokąty, które nie są tylnymi, są określane jako *przednie ściany*.



Rys. 13.7. Wyznaczanie ścian skierowanych do tyłu. Eliminowane są wielokąty skierowane do tyłu (*A, B, D, F*) i są zaznaczone jako szare, wielokąty natomiast skierowane do przodu (*C, E, G, H*) są zachowane

We współrzędnych oka tylny wielokąt może być zidentyfikowany przez nieujemny iloczyn skalarny normalnej powierzchni i wektora od punktu zbieżności do dowolnego punktu wielokąta. (Inaczej mówiąc, iloczyn skalarny jest dodatni dla tylnego wielokąta; zerowa wartość iloczynu skalarnego pokazuje wielokąt widziany jako krawędź.) Przy założeniu, że zostało wykonane przekształcenie perspektywiczne albo że jest potrzebny rzut prostokątny na płaszczyznę (x, y), kierunek rzutu jest $(0, 0, -1)$. W tym przypadku sprawdzanie iloczynu skalarnego sprowadza się do wybrania wielokąta jako tylnego tylko wówczas, gdy normalna do powierzchni ma ujemną współrzędną z. Jeżeli środowisko zawiera jeden wypukły wielościan, to jedynym obliczeniem związanym z określeniem powierzchni widocznych jest wyznaczenie ścian tylnych. W innym przypadku mogą występować wielokąty skierowane do przodu, takie jak C i E na rys. 13.7, które są częściowo albo całkowicie zasłonięte.

Jeżeli wielościany nie mają ścian przednich lub zostały one obcięte albo jeżeli wielokąty w ogóle nie są częścią wielościanu, to ściany skierowane do tyłu muszą być potraktowane specjalnie. Jeżeli wybieranie nie jest konieczne, to najprostsze podejście polega na tym, żeby wielokąty skierowane do tyłu traktować tak, jak gdyby były ścianami skierowanymi do przodu, zmieniając kierunek normalnej na przeciwny. W systemie PHIGS PLUS użytkownik może określić całkowicie niezależny zbiór właściwości każdej strony powierzchni.

Zauważmy, że promień rzutujący przechodzący przez wielościan przecina taką samą liczbę wielokątów skierowanych do tyłu jak wielokątów skierowanych do przodu. Punkt w rzucie wielościanu leży na rzutach takiej samej liczby wielokątów skierowanych do tyłu jak wielokątów skierowanych do przodu. Dlatego wybieranie wielokątów skierowanych do tyłu zmniejsza o połowę liczbę wielokątów, które trzeba wziąć pod uwagę dla każdego piksela w algorytmie powierzchni widocznych z precyzją obrazową. Średnio około połowa wielokątów w wielościanie jest skierowana do tyłu. A więc wybieranie ścian skierowanych do tyłu również na ogół prowadzi do zmniejszenia o połowę liczby wielokątów, które trzeba rozważyć w pozostałej części algorytmu wyznaczania powierzchni widocznych z precyzją obiektową. (Zauważmy jednak, że jest to prawdą jedynie w znaczeniu statystycznym. Na przykład podstawa ostrosłupa może być jedynym wielokątem obiektu skierowanym do tyłu albo do przodu.)

13.1.5. Podział przestrzenny

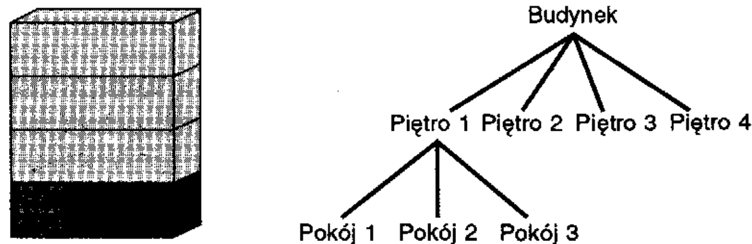
Podział przestrzenny umożliwia rozbicie dużego problemu na kilka mniejszych. Podstawowe podejście polega na przypisaniu obiektów albo ich rzutów do przestrzennie spójnych grup w czasie przetwarzania wstępnego. Na przykład możemy podzielić rzutnię regularną siatką prostokątną 2D i określić, w której komórce siatki leży rzut obiektu. Rzuty muszą być porównane ze względu na nakładanie się tylko z innymi rzutami, które znajdują się w odpowiednich polach siatki. Ta metoda jest stosowana w pracach [ENCA72; MAHN73; FRAN80; HEDG82]. Podział przestrzenny może być używany do nakładania regularnej siatki 3D na obiekty w środowisku. Proces określania, które obiekty są przecinane przez promień rzutujący, można wobec tego przyspieszyć przez wstępne określenie, które części przecina promień, a potem testowanie tylko tych obiektów, które leżą w tych częściach (p. 13.4).

Jeżeli obrazowane obiekty są nierównomiernie rozłożone w przestrzeni, to efektywniejsze może być stosowanie *podziału adaptacyjnego*, w którym wielkość każdej części zmienia się. Jedno z podejść do po-

działu adaptacyjnego polega na tym, żeby dzielić przestrzeń rekursywnie, dopóki dla każdej części nie zostanie spełnione pewne kryterium zakończenia podziału. Na przykład podział może zostać zatrzymany wówczas, gdy w danej części jest mniej obiektów od pewnej maksymalnej liczby [TAMM82]. Szczególnie atrakcyjne dla tego celu są drzewa czwórkowe, drzewa ósemkowe i drzewa BSP z p. 10.6.

13.1.6. Hierarchia

Jak widzieliśmy w rozdz. 7, hierarchia może być użyteczna do wiązania struktury i ruchu różnych obiektów. Zagnieżdżony model hierarchiczny, w którym każdy potomek jest traktowany jako część swojego przodka, może być również użyty do ograniczenia liczby porównań obiektów, które są potrzebne w algorytmie powierzchni widocznych [CALR76; RUBI80; WEGH84]. Obiekt na jednym poziomie hierarchii może służyć jako rozszerzenie dla swoich potomków, jeżeli są oni całkowicie w nim zawarci, jak pokazano na rys. 13.8. W tym przypadku jeżeli dwa obiekty w hierarchii nie przetną się, to obiekty niższego poziomu jednego z tych przodków nie muszą być testowane na przecięcie z obiektami będącymi potomkami drugiego. Podobnie, jeżeli stwierdzi się, że promień rzutujący przecina obiekt w hierarchii, to musi być testowany na przecięcie z potomkami obiektu. Takie wykorzystanie hierarchii jest ważną odmianą spójności obiektu.



Rys. 13.8. W celu ograniczenia liczby porównań obiektów można użyć hierarchii. Jedynie wówczas, gdy promień rzutujący przecina budynek i podłogę, trzeba wykonywać testowanie przecięć z pokojami od 1 do 3

W dalszej części tego rozdziału omówimy wiele algorytmów opracowanych do określania powierzchni widocznych. Koncentrujemy się tutaj na obliczeniu, które części powierzchni obiektu są widoczne, pozostawiając określenie barwy powierzchni do rozdz. 14. Opisując każdy algorytm zwracamy uwagę na jego zastosowanie do wielokątów, ale podkreślamy możliwość uogólnienia na inne obiekty.

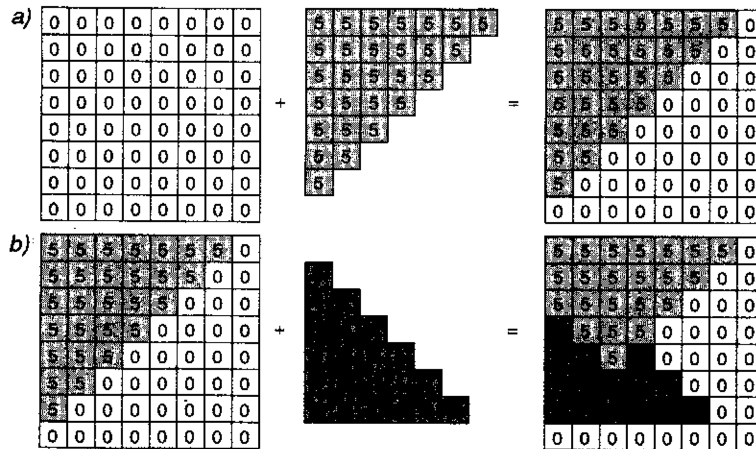
13.2. Algorytm z-bufora

Algorytm z-bufora, albo *bufora głębokości*, z grupy algorytmów z precyzją obrazową, został opracowany przez Catmulla [CATM74]; jest to jeden z najprostszych do realizacji sprzętowej albo programowej algorytmów znajdowania powierzchni widocznych. Wymaga on korzystania obok pamięci obrazu F, w której są pamiętane wartości barw, z z-bufora Z o takiej samej liczbie pozycji, w której jest pamiętana wartość z dla każdego piksela. Na początku bufor jest zerowany, co odpowiada zapamiętaniu wartości z dla tylnej ściany obcinającej, a do bufora obrazu jest wpisywana barwa tła. Największa wartość, jaka może być wpisana w z-buforze, reprezentuje z dla przedniej ściany obcinającej. Wielokąty są przeglądane wierszami; kolejność przeglądania wielokątów jest dowolna. W czasie procesu konwersji, jeżeli rozpatrywany punkt (x, y) nie jest dalej od obserwatora niż punkt, którego barwa i głębokość są zapisane w buforach, to nowa barwa i głębokość zastępują stare wartości. Pseudokod algorytmu z-bufora pokazano w programie 13.1. Procedury WritePixel i ReadPixel wprowadzone w rozdz. 3 są tutaj uzupełnione o procedury WriteZ i ReadZ, które zapisują i odczytują z-bufor.

Program 13.1 Pseudokod algorytmu z-bufora

```
void zBuffer
int pz;                /* Wartość z wielokąta dla piksela o współrzędnych (x, y) */
{
   for (y = 0; y < YMAX; y++) {
      for (x = 0; x < XMAX; x++) {
         WritePixel (x, y, BACKGROUND_VALUE);
         WriteZ (x, y, 0);
      }
   }
   for (każdy wielokąt) {
      for (każdy piksel w rzucie wielokąta) {
         pz = wartość z wielokąta dla piksela o współrzędnych (x, y);
         if (pz >= ReadZ (x, y)) {          /* Nowy punkt nie jest dalej */
            WriteZ (x, y, pz);
            WritePixel (x, y, barwa wielokąta dla piksela o współrzędnych (x, y));
         }
      }
   }
}
```

Nie jest potrzebne żadne wstępne sortowanie ani porównanie typu obiekt-obiekt. Cały proces to nic więcej jak szukanie największej wartości Z_i dla każdego zbioru par $\{Z_i(x, y), F_i(x, y)\}$ dla ustalonych x i y.



Rys. 13.9. z-bufor. Poziomy szarości reprezentują barwy, a wartości z są pokazane za pomocą cyfr: a) dodanie wielokąta o stałej wartości z do pustego z-bufora; b) dodanie innego wielokąta, który przecina pierwszy wielokąt

W pamięci obrazu i w z-buforze są zapisywane informacje związane z największym z napotkanym dotychczas dla każdego (x, y). Dlatego wielokąty pojawiają się na ekranie w kolejności, w jakiej są przetwarzane. Każdy wielokąt może być przeglądany wierszami, jak to opisano w p. 3.5. Na rysunku 13.9 pokazano dodanie dwóch wielokątów do obrazu. Barwa każdego piksela jest reprezentowana przez odcień szarości, a z jest reprezentowane przez liczbę. Pamiętając naszą dyskusję o spójności głębokości możemy uprościć obliczanie z dla każdego punktu w przeglądanym wierszu korzystając z faktu, że wielokąt jest płaski. Zwykle w celu obliczenia z powinniśmy rozwiązać równanie płaszczyzny $Ax + By + Cz + D = 0$ dla zmiennej z

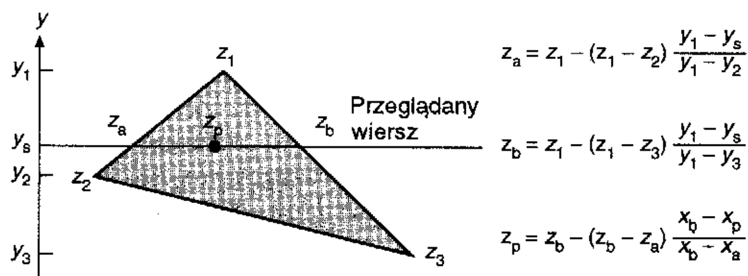
$$z = \frac{-D - Ax - By}{C} \tag{13.6}$$

Teraz jeżeli w punkcie (x, y) z równania (13.6) otrzymamy z_1, to w punkcie $(x + \Delta x, y)$ wartość z wynosi

$$z_1 - \frac{A}{C}(\Delta x) \tag{13.7}$$

Jeżeli $z(x, y)$ jest dane, to do obliczenia $z(x + 1, y)$ jest potrzebne tylko jedno odejmowanie, ponieważ iloraz A/C jest stały i $\Delta x = 1$. Podobne obliczenie przyrostowe można wykonać w celu obliczenia pierwszej wartości z w następnym przeglądanym wierszu, zmniejszając z o B/C dla każdego Δy. W przypadku gdy płaszczyzna nie została

określona albo gdy wielokąt nie jest płaski (por. p. 9.12), to wówczas $z(x, y)$ można określić interpolując współrzędne z wierzchołków wielokąta wzdłuż par krawędzi, a potem wzdłuż każdego przeglądanego wiersza (rys. 13.10). Obliczenia przyrostowe mogą być tutaj również wykorzystane. Zauważmy, że barwa piksela nie musi być obliczana, jeżeli warunkowe określanie widoczności piksela nie jest spełnione. A więc jeżeli obliczenia związane z cieniowaniem zajmują dużo czasu, to można zwiększyć efektywność wykonując zgrubne sortowanie obiektów, które mają być wyświetlone od najbliższego do najdalszego, i wyświetlać najpierw najbliższe obiekty. Algorytm z-bufora nie wymaga, żeby obiekty były wielokątami. Rzeczywiście jedną z dominujących zalet tej metody jest to, że może być używana do renderingu dowolnego obiektu, jeżeli dla każdego punktu rzutu można obliczyć barwę i wartość z; nie trzeba pisać żadnego bezpośredniego algorytmu przecinania.



Rys. 13.10. Interpolacja wartości z wzdłuż krawędzi wielokąta i wierszy przeglądania. z_a jest interpolowane między z_1 i z_2; z_b między z_1 i z_3; z_p między z_a i z_b

Algorytm z-bufora wykonuje sortowanie wagowe względem x i y, bez żadnych porównań, a przy sortowaniu względem z jest potrzebne tylko jedno porównanie na piksel dla każdego wielokąta zawierającego ten piksel. Czas zajmowany przez obliczenia związane z wyznaczaniem widocznych powierzchni jest w przybliżeniu niezależny od liczby wielokątów w obiektach, ponieważ średnio liczba pikseli pokrytych przez każdy wielokąt zmniejsza się wraz ze wzrostem liczby wielokątów w bryle widzenia. Dlatego średnia wielkość każdego zbioru przeglądanych par zmierza do wartości stałej. Oczywiście trzeba również wziąć pod uwagę narzut na konwersję wierszową wprowadzany przez dodatkowe wielokąty.

Chociaż algorytm z-bufora wymaga dużej pamięci dla z-bufora, jest łatwy do implementacji. Jeżeli jest problem z pamięcią, to obraz może być przeglądany pasami, tak żeby potrzebna była pamięć tylko na przetwarzany pas, kosztem wykonywania wielokrotnego przeglądania obiektów. Ze względu na prostotę z-bufora i brak dodatkowych struktur danych, zmniejszające się koszty pamięci zainspirowały kilka sprzętowych implementacji z-bufora. Ponieważ algorytm z-bufora działa z pre-

cyzją obrazową, mogą się pojawiać zakłócenia. Ten problem jest rozwiązywany za pomocą algorytmu A-bufora [CARP84], gdzie wykorzystuje się dyskretną aproksymację do bezwagowego próbkowania powierzchni.

z-bufor jest często implementowany sprzętowo dla liczb całkowitych od 16- do 32-bitowych, realizacje programowe natomiast (i niektóre sprzętowe) mogą wykorzystywać wartości zmiennopozycyjne. Chociaż 16-bitowy z-bufor daje wystarczający zakres dla wielu zastosowań CAD/CAM, 16 bitów zapewnia zbyt małą precyzję reprezentowania środowisk, w których obiekty zdefiniowane z dokładnością milimetrową są umieszczane w odległościach mierzonych w kilometrach. Sytuacja jest jeszcze gorsza, jeżeli jest wykorzystywane rzutowanie perspektywiczne; kompresja odległych wartości z wynikająca z dzielenia perspektywicznego ma istotny wpływ na porządkowanie głębokości i przecięcia odległych obiektów. Dwa punkty umieszczone blisko rzutni po przekształceniu mogą dać dwie różne wartości całkowite z, natomiast gdy są daleko od rzutni, mogą dać taką samą wartość z (por. zadanie 13.9 i [HUGH89]).

Skończona precyzja z-bufora jest odpowiedzialna za inny typ zakłócenia. Algorytmy konwersji wierszowej na ogół dają dwa różne zbiory pikseli przy renderingu wspólnej części dwóch krawędzi współliniowych, zaczynających się w różnych punktach. Niektórym z tych pikseli, wspólnych dla rysowanych krawędzi, mogą być również przypisane nieco różne wartości z ze względu na numeryczne niedokładności w wykonywaniu interpolacji z. Ten efekt jest najbardziej zauważalny dla wspólnych krawędzi na ścianach wielościanu. Niektóre z widocznych pikseli wzdłuż krawędzi mogą być częścią jednego wielokąta, reszta zaś pochodzi od sąsiada wielokąta. Problem można rozwiązać umieszczając dodatkowe wierzchołki w celu zapewnienia, żeby wierzchołki pojawiały się w tych samych punktach na wspólnej części dwóch krawędzi współliniowych.

Nawet gdy zakończył się rendering obrazu, z-bufor może być jeszcze użyteczny. Ponieważ jest to jedyna struktura danych wykorzystywana przez algorytm powierzchni widocznych, można zapamiętać ten bufor razem z obrazem i stosować go później przy mieszaniu z innymi obiektami, dla których można obliczyć z. Algorytm może być również tak zakodowany, żeby pozostawiać nie zmienioną zawartość bufora głębokości przy renderingu wybranych obiektów. Jeżeli z bufor jest tak maskowany, to można wpisać pojedynczy obiekt do oddzielnego zbioru płaszczyzn nakładek z poprawnie usuniętymi powierzchniami niewidocznymi (jeżeli obiekt jest jednowartościową funkcją x i y) i potem usunąć go bez wpływania na zawartość z-bufora. Stąd prosty obiekt taki jak siatka może być przesuwany po obrazie w kierunkach x, y i z i służyć jako kursor 3D, który zasłania i może być zasłaniany przez obiekty w środowisku. Można tworzyć przekroje warunkując zapisy do z-bufora i pamięci obrazu tym, czy wartość z jest poza płaszczyzną

obcinającą. Jeżeli wyświetlane obiekty mają jedną wartość z dla każdego (x, y), to zawartość z-bufora może być również użyta do obliczania powierzchni albo objętości.

Rossignac i Requicha [ROSS86] omawiają sposób adaptacji algorytmu do obsługi obiektów zdefiniowanych metodą CSG. Każdy piksel w rzucie powierzchni jest zapisywany tylko wówczas, gdy jest bliższy w sensie wartości z i jest na obiekcie CSG skonstruowanym z powierzchni. Zamiast pamiętać tylko punkt o najbliższej wartości z dla każdego piksela, Atherton sugeruje, żeby zapamiętywać listę wszystkich punktów uporządkowaną ze względu na z z towarzyszącą informacją identyfikującą powierzchnię w celu utworzenia *bufora obiektu* [ATHE81]. Etap przetwarzania końcowego określa, jak obraz ma być wyświetlony. Przetwarzając listę każdego piksela bez potrzeby ponownej konwersji wierszowej obiektów, można osiągnąć kilka efektów, np. przezroczystość, obcinanie i operacje boolowskie.

13.3. Algorytmy przeglądania

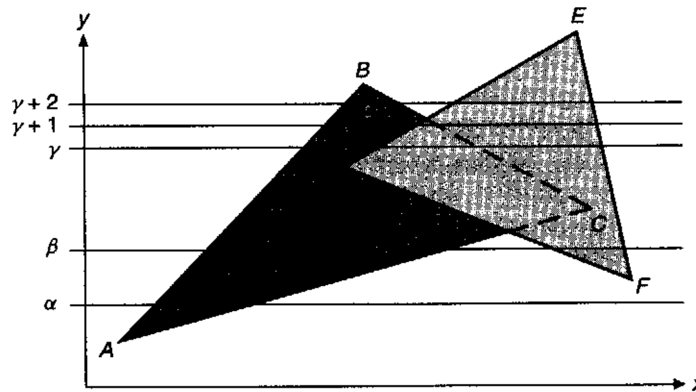
Algorytmy przeglądania, opracowane pierwotnie przez Wylie'a, Romneya, Evansa i Erdahla [WYLI67], Bouknighta [BOUK70a; BOUK70b] i Watkinsa [WATK70], działają z precyzją obrazową i tworzą obraz jednego wiersza na raz. Podstawowa koncepcja polega na rozszerzeniu algorytmu konwersji wierszowej opisanego w p. 3.5 oraz wykorzystaniu różnych form spójności, w tym spójności przeglądania wierszowego i spójności krawędziowej. Różnica polega na tym, że zajmujemy się nie jednym wielokątem, a kilkoma. W pierwszym kroku jest tworzona *tablica krawędzi* (ET) dla wszystkich niepoziomych krawędzi wszystkich wielokątów zrzutowanych na rzutnię. Tak jak poprzednio krawędzie poziome są pomijane. Pozycje w tablicy ET są podzielone na grupy ze względu na najmniejszą współrzędną y każdej krawędzi, a w obrębie grupy są porządkowane ze względu na rosnącą współrzędną x ich niższego punktu końcowego. Każda grupa zawiera:

1. Współrzędną x końca o mniejszej współrzędnej y.
2. Współrzędną y drugiego końca krawędzi.
3. Przyrost x, Δx, używany przy przejściu z jednego przeglądanego wiersza do następnego (x jest odwrotnością nachylenia krawędzi).
4. Numer identyfikacyjny wielokąta, wskazujący wielokąt, do którego należy krawędź.

Potrzebna jest również *tablica wielokątów* (PT), która – oprócz numeru identyfikacyjnego – zawiera przynajmniej następujące informacje o każdym wielokącie:

1. Współczynniki równania płaszczyzny.
2. Informację o cieniowaniu albo barwie wielokąta.
3. Wskaźnik boolowski we-wy wykorzystywany w przeglądaniu wiersza ustawiany początkowo na wartość *false*.

Na rysunku 13.11 pokazano rzut dwóch trójkątów na płaszczyznę (x, y); ukryte krawędzie są pokazane liniami przerywanymi. Uporządkowana tablica ET dla tej figury zawiera pozycje dla *AB*, *AC*, *FD*, *FE*, *CB* i *DE*. Tablica PT ma pozycje dla *ABC* i *DEF*.



Rys. 13.11. Dwa wielokąty przeglądane zgodnie z algorytmem przeglądania wierszy

Pozycja ET

x	y_{max}	Δx	ID	• →

Pozycja PT

ID	Równanie płaszczyzny	Informacja o cieniowaniu	We-wy

Zawartość AET

Przeglądany wiersz	Pozycje			
α	*AB*	*AC*		
β	*AB*	*AC*	*FD*	*FE*
$\gamma, \gamma+1$	*AB*	*DE*	*CB*	*FE*
$\gamma+2$	*AB*	*CB*	*DE*	*FE*

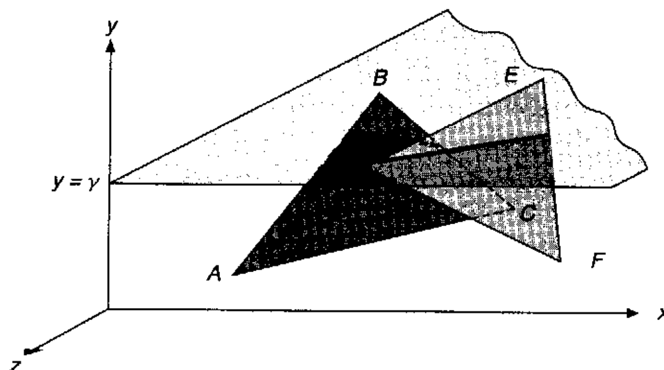
Rys. 13.12. Tablice ET, PT i AET dla algorytmu przeglądania wierszy

Tablica krawędzi aktywnych (AET) omawiana w p. 3.5 jest potrzebna również tutaj. Jest ona zawsze utrzymywana w porządku wynikającym z rosnących wartości x. Na rysunku 13.12 pokazano pozycje tablic ET, PT i AET. W chwili, gdy algorytm doszedł do przeglądania wiersza $y = \alpha$, tablica AET zawiera *AB* i *AC* (w tej kolejności). Krawędzie są przetwarzane od lewa do prawa. W celu przetworzenia *AB* najpierw negujemy wskaźnik we-wy wielokąta *ABC*. W tym przypadku wskaźnik przyjmuje wartość *true*; wobec tego segment jest teraz wewnątrz wielokąta i wielokąt musi być rozważany. Ponieważ segment jest tylko w jednym wielokącie (*ABC*), musi być widoczny i trzeba go pocieniować od

krawędzi AB do następnej krawędzi w tablicy AET, czyli krawędzi AC. Jest to przypadek spójności segmentu. Na tej krawędzi wskaźnik dla ABC jest zamieniany na *false* i segment już nie jest wewnątrz żadnego wielokąta. Następnie, ponieważ AC jest ostatnią krawędzią w AET, proces przeglądania wiersza zostaje zakończony. Tablica AET jest uaktualniana na podstawie tablicy ET i ponownie porządkowana ze względu na x, ponieważ niektóre z jej krawędzi mogłyby się przeciąć, i jest przetwarzany następny wiersz.

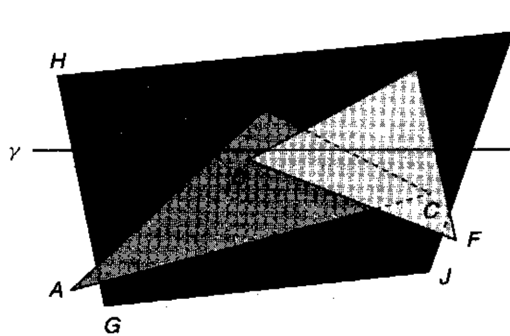
Gdy dojdzie się do przeglądania wiersza $y = \beta$, w tablicy AET jest następująca kolejność: AB, AC, FD, FE. Przetwarzanie odbywa się mniej więcej tak samo jak poprzednio. W jednym wierszu są dwa wielokąty, ale segment jest tylko w jednym wielokącie.

Znacznie ciekawsza sytuacja jest dla $y = \gamma$. Wejście do ABC powoduje, że jego wskaźnik przyjmuje wartość *true*. Barwa ABC jest używana dla segmentu aż do następnej krawędzi DE. W tym miejscu wskaźnik dla DEF również przyjmuje wartość *true* i segment jest w dwóch wielokątach. (Pożyteczne jest przechowywanie bezpośredniej listy wielokątów, których wskaźniki we-wy są *true*, oraz przechowywanie liczby określającej, ile wielokątów jest na liście.) Musimy teraz zdecydować, czy ABC czy DEF jest bliżej obserwatora; w tym celu obliczamy równania płaszczyzn obu wielokątów dla z przy $y = \gamma$ i x równym przecięciu wiersza $y = \gamma$ z krawędzią DE. Ta wartość x jest w tablicy AET w pozycji dla DE. W naszym przykładzie dla DEF z jest większe, dlatego wielokąt ten jest widoczny. Zakładając, że wielokąty nie przecinają się, segmentowi przypisuje się barwę taką jak w trójkącie DEF aż do krawędzi CB, gdzie wskaźnik trójkąta ABC przyjmuje wartość *false* i segment jest znowu tylko w jednym wielokącie DEF, którego barwa jest przypisana segmentowi aż do krawędzi FE. Na rysunku 13.13 pokazano wzajemne położenie dwóch wielokątów i płaszczyzny $y = \gamma$; dwie grube linie wskazują przecięcia wielokątów z płaszczyzną.

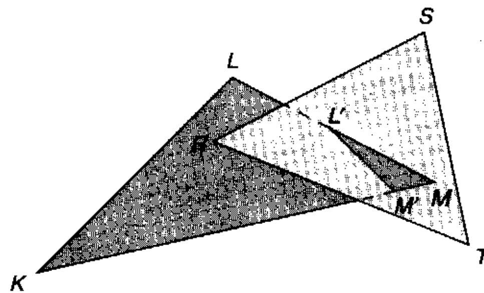


Rys. 13.13. Przecięcie wielokątów ABC i DEF z płaszczyzną $y = \gamma$

Załóżmy, że zarówno za *ABC*, jak i za *DEF* jest duży wielokąt *GHIJ* (rys. 13.14). Jeżeli przeglądając wiersz $y = \gamma$ dojdziemy do krawędzi *CB*, to segment będzie wciąż w wielokątach *DEF* i *GHIJ* i obliczenia głębokości są wykonywane ponownie. Tych obliczeń można jednak uniknąć, jeżeli założymy, że żaden z wielokątów nie przecina drugiego. To założenie oznacza, że gdy segment opuszcza *ABC*, wówczas relacja głębokości między *DEF* i *GHIJ* nie może się zmienić i *DEF* jest w dalszym ciągu z przodu. Dlatego obliczanie głębokości nie jest konieczne wówczas, gdy przy przeglądaniu wychodzi się poza zasłonięty wielokąt, a jest potrzebne tylko wówczas, gdy wychodzi się poza zasłaniający wielokąt.



Rys. 13.14. Trzy nie przecinające się wielokąty. Nie trzeba wyznaczać głębokości, gdy przeglądany wiersz y wychodzi z zasłoniętego wielokąta *ABC*, ponieważ nie przecinające się wielokąty zachowują swój względny porządek



Rys. 13.15. Wielokąt *KLM* przebija wielokąt *RST* wzdłuż linii *L'M'*

W celu poprawnego wykorzystania algorytmu dla przecinających się wielokątów z rys. 13.15 dzielimy *KLM* na *KLL'M'* i *L'MM'*, wprowadzając *fałszywą krawędź M'L'*. Można też tak zmodyfikować algorytm, żeby znajdować punkt przecięcia w przeglądanym wierszu w trakcie przetwarzania przeglądanego wiersza.

W innej modyfikacji tego algorytmu wykorzystuje się *spójność głębokości*. Zakładając, że wielokąty wzajemnie nie przecinają się, Romney zauważył, że, jeżeli dla przeglądanego wiersza w AET są te same krawędzie co w poprzednim przeglądanym wierszu i na dodatek w tej samej kolejności, to nie pojawia się żadna zmiana relacji głębokości w żadnym segmencie przeglądanego wiersza i nie są potrzebne nowe obliczenia związane z głębokością [ROMN69]. Wobec tego rekord widocznego segmentu w poprzednio przeglądanym wierszu określa segment w bieżącym wierszu. Tak jest w przypadku wierszy $y = \gamma$ i $y = \gamma + 1$ na rys. 13.11, dla których segmenty od AB do DE i od DE do FE są widoczne. Spójność głębokości jest jednak na tym rysunku tracona, gdy przejdziemy z wiersza $y = \gamma + 1$ do $y = \gamma + 2$, ponieważ krawędzie DE i CB zmieniają kolejność w AET (sytuacja, w której algorytm musi się przystosować). Zmieniają się w związku z tym segmenty i w tym przypadku rozciągają się od AB do CB i od DE do FE. Hamlin i Gear [HAML77] pokazują, jak spójność głębokości może być zachowana, nawet wówczas, gdy krawędzie zmieniają kolejność w AET.

Nie zastanawialiśmy się dotychczas, jak traktować tło. Najprostszy sposób polega na tym, żeby na początku wpisać do bufora obrazu barwę tła, tak żeby algorytm musiał tylko przetwarzać wiersze, które przecinają krawędzie. Inny sposób polega na włączeniu do definicji sceny dostatecznie dużego wielokąta, który jest dalej z tyłu niż wszystkie inne, jest równoległy do rzutni i jest odpowiednio cieniowany. Wreszcie można tak zmodyfikować algorytm, żeby barwa tła była umieszczana bezpośrednio w pamięci obrazu zawsze wówczas, gdy przeglądany punkt nie jest w żadnym wielokącie.

Chociaż przedstawione algorytmy dotyczą wielokątów, podejście typu przeglądanie wiersza było używane do ogólniejszych powierzchni, tak jak to opisano w p. 13.5.3. W celu zrealizowania tego ET i AET są zastępowane *tablicami powierzchni* i *tablicami aktywnych powierzchni*, segregowanymi ze względu na zakresy zmian (x, y). Gdy powierzchnia jest przesuwana z tablicy powierzchni do tablicy powierzchni aktywnych, można wykonać dodatkowe przetwarzanie. Na przykład powierzchnia może być zdekomponowana na zbiór wielokątów aproksymujących, które mogłyby być później odrzucone, gdy segment opuszcza przedział y powierzchni; eliminuje to potrzebę zachowywania wszystkich danych o powierzchni w czasie całego procesu renderingu. Program 13.2 zawiera pseudokod tego ogólnego algorytmu przeglądania wiersza, który dokonuje renderingu obiektów wielokątowych uzyskanych za pomocą regularyzowanych operatorów boolowskich dla konstruktywnej geometrii brył.

Program 13.2 Pseudokod ogólnego algorytmu przeglądania wierszy

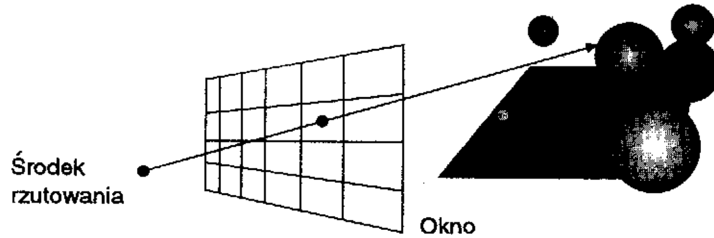
```
dodanie powierzchni do tablicy powierzchni;
inicjalizacja powierzchni aktywnych;

for (każdy przeglądany wiersz ) {
    uaktualnienie tablicy powierzchni aktywnych;

    for (każdy piksel w przeglądanym wierszu) {
        określenie powierzchni w tablicy powierzchni aktywnych, których rzut trafia
        w dany piksel;
        znalezienie najbliższej takiej powierzchni;
        Przypisanie pikselowi barwy najbliższej powierzchni;
    }
}
```

W metodzie z przeglądaniem wierszy, zachęcającej ze względu na swą prostotę, korzysta się z z-bufora do rozwiązywania problemu powierzchni widocznych [MYER75]. Dla każdego nowego wiersza jest zerowany z-bufor oraz bufor obrazu na jeden przeglądany wiersz i są one wykorzystywane do akumulowania segmentów. Ponieważ na bufory jest potrzebna pamięć dla jednego wiersza, łatwo można przystosować się do obsługi obrazów o wyjątkowo dużej rozdzielczości obrazów.

13.4. Wyznaczanie powierzchni widocznych metodą śledzenia promieni

Metoda śledzenia promieni określa widoczność powierzchni na zasadzie śledzenia umownych promieni światła od oka obserwatora do obiektów sceny[2]. Jest to dokładnie prototypowy algorytm z precyzją obrazu omawiany na początku tego rozdziału. Jest wybierany środek rzutowania (oko obserwatora) oraz pole wizualizacji na dowolnej rzutni. Można sobie wyobrazić, że pole wizualizacji jest podzielone na regularną siatkę (o wymaganej rozdzielczości), której elementy odpowiadają pikselom. Następnie dla każdego piksela w polu wizualizacji, ze środka rzutowania jest wypuszczany promień przez środek piksela w kierunku sceny (rys. 13.16). Barwa piksela jest ustawiana zgodnie z barwą obiektu dla najbliższego punktu przecięcia. Program 13.3 zawiera pseudokod dla tej prostej metody śledzenia promieni.

[2] Nazwa *metoda śledzenia promieni* jest wykorzystywana tylko w odniesieniu do algorytmu powierzchni widocznych z danego punktu, a także do rekursywnego algorytmu z p. 14.7.



Rys. 13.16. Ze środka rzutowania jest prowadzony promień przez każdy piksel w celu określenia najbliższego przecinanego obiektu

Program 13.3 Pseudokod dla prostej metody śledzenia promieni

```
wybranie środka rzutowania i pola wizualizacji na rzutni;
for (każdy przeglądany wiersz obrazu) {
  for (każdy piksel w przeglądanym wierszu) {
    wyznaczenie promienia ze środka rzutowania przez piksel;
    for (każdy obiekt w scenie) {
      if (obiekt jest przecinany i jest najbliższy z dotychczas rozważanych)
        rejestrowanie przecięcia i nazwy obiektu;
    }
    ustawienie barwy piksela zgodnie z barwą najbliższego przeciętego obiektu;
  }
}
```

Metoda śledzenia promieni była pierwotnie opracowana przez Appela [APPE68] oraz Goldsteina i Nagela [MAGI68; GOLD71]. Appel wykorzystywał rzadką siatkę promieni w celu wyznaczenia cieniowania, z uwzględnieniem sprawdzania, czy punkt znajduje się w cieniu. Goldstein i Nagel wykorzystywali swój algorytm do symulowania trajektorii torów pocisków i cząstek nuklearnych; dopiero później zastosowali go do grafiki. Appel pierwszy śledził promienie cieni, a Goldstein i Nagel jako pierwsi stosowali metodę śledzenia promieni do obliczania operacji boolowskich. Whitted [WHIT80] i Kay [KAY79a] rozszerzyli metodę śledzenia promieni tak, żeby było możliwe uwzględnianie odbić zwierciadlanych i załamań. W punkcie 14.7 omawiamy problemy cieni, odbić i załamań, a więc efektów, z rozwiązywania których metoda śledzenia promieni jest najbardziej znana; jest opisany tam pełny rekursywny algorytm śledzenia, który łączy wyznaczanie powierzchni widocznych z cieniowaniem. Tutaj traktujemy metodę śledzenia promieni tylko jako algorytm wyznaczania powierzchni widocznych.

13.4.1. Obliczanie przecięć

Elementem kluczowym każdego programu śledzenia promieni jest zadanie wyznaczenia przecięcia obiektu z promieniem. W celu wykonania

tego zadania korzystamy z takiej samej reprezentacji parametrycznej, jaka była wprowadzona w rozdz. 3. Każdy punkt (x, y, z) wzdłuż promienia od (x_0, y_0, z_0) do (x_1, y_1, z_1) jest określany wartością parametru t taką, że:

$$x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0), \quad z = z_0 + t(z_1 - z_0) \tag{13.8}$$

Dla wygody określamy wartości Δx, Δy i Δz tak, że:

$$\Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0, \quad \Delta z = z_1 - z_0 \tag{13.9}$$

Wobec tego

$$x = x_0 + t\,\Delta x, \quad y = y_0 + t\,\Delta y, \quad z = z_0 + t\,\Delta z \tag{13.10}$$

Jeżeli środek rzutowania jest w (x_0, y_0, z_0), a środek piksela w oknie jest w (x_1, y_1, z_1), to t zmienia się od 0 do 1 między tymi punktami. Ujemne wartości t reprezentują punkty poza środkiem rzutowania, a wartości t większe niż 1 odpowiadają punktom po stronie rzutni dalszej od środka rzutowania. Dla każdego rodzaju obiektu musimy znaleźć reprezentację, która umożliwi określenie t na przecięciu obiektu z promieniem. Jednym z obiektów, dla których najłatwiej jest to zrobić, jest kula i stąd nadmiar kul obserwowanych w typowych obrazach obliczanych metodą śledzenia promieni! Kula o środku w (a, b, c) i promieniu r może być reprezentowana za pomocą równania

$$(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2 \tag{13.11}$$

Przecięcie znajduje się po rozwinięciu równania (13.11) i podstawieniu wartości x, y i z z równania (13.10); otrzymuje się wtedy zależności:

$$x^2 - 2ax + a^2 + y^2 - 2by + b^2 + z^2 - 2cz + c^2 = r^2 \tag{13.12}$$

$$(x_0 + t\Delta x)^2 - 2a(x_0 + t\Delta x) + a^2 + (y_0 + t\Delta y)^2 - 2b(y_0 + t\Delta y) + \\ + b^2 + (z_0 + t\Delta z)^2 - 2c(z_0 + t\Delta z) + c^2 = r^2 \tag{13.13}$$

$$x_0^2 + 2x_0\Delta xt + \Delta x^2t^2 - 2ax_0 - 2a\Delta xt + a^2 + \\ + y_0^2 + 2y_0\Delta yt + \Delta y^2t^2 - 2by_0 - 2b\Delta yt + b^2 + \\ + z_0^2 + 2z_0\Delta zt + \Delta z^2t^2 - 2cz_0 - 2c\Delta zt + c^2 = r^2 \tag{13.14}$$

Po uporządkowaniu otrzymuje się:

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] +$$
$$+ (x_0^2 - 2ax_0 + a^2 + y_0^2 - 2by_0 + b^2 + z_0^2 - 2cz_0 + c^2) - r^2 = 0 \tag{13.15}$$

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] +$$
$$+ (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0 \tag{13.16}$$

Równanie (13.16) jest równaniem kwadratowym ze względu na t, w którym współczynniki są wyrażone w zależności od stałych otrzymanych z równania kuli i równania promienia; stąd równanie może być rozwiązane jak zwykłe równanie kwadratowe. Jeżeli nie ma pierwiastków rzeczywistych, to kula i promień nie przecinają się; jeżeli jest tylko jeden pierwiastek rzeczywisty, to promień jest styczny do kuli. W przeciwnym przypadku dwa pierwiastki określają punkty przecięcia z kulą; ten, dla którego wartość t jest mniejsza, jest bliższy. Dobrze jest również tak znormalizować promień, żeby odległość od (x_0, y_0, z_0) do (x_1, y_1, z_1) była równa 1. W ten sposób otrzymuje się wartość t określającą odległość w jednostkach WC i upraszcza się obliczenia, ponieważ współczynnik przy t^2 w równaniu (13.16) staje się równy 1. W podobny sposób możemy otrzymać przecięcie promienia z ogólną powierzchnią drugiego stopnia omawianą w rozdz. 9.

Jak zobaczymy w rozdz. 14, musimy określić normalną do powierzchni w punkcie przecięcia po to, żeby określić barwę powierzchni. Jest to szczególnie łatwe w przypadku kuli, ponieważ (nieznormalizowana) normalna jest wektorem od środka do punktu przecięcia: kula o środku (a, b, c) ma normalną do powierzchni $((x - a)/r, (y - b)/r, (z - c)/r)$ w punkcie przecięcia (x, y, z).

Znajdowanie przecięcia promienia z wielokątem jest nieco trudniejsze. W celu wyznaczenia punktu przecięcia promienia z wielokątem najpierw sprawdzamy, czy promień przecina płaszczyznę wielokąta, a następnie czy punkt przecięcia leży w wielokącie. Ponieważ równanie płaszczyzny ma postać

$$Ax + By + Cz + D = 0 \tag{13.17}$$

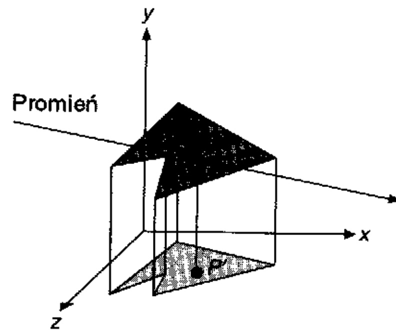
po podstawieniu zależności z równania (13.10) otrzymujemy:

$$A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0 \tag{13.18}$$

$$t(A\Delta x + B\Delta y + C\Delta z) + (Ax_0 + By_0 + Cz_0 + D) = 0 \tag{13.19}$$

$$t = -\frac{(Ax_0 + By_0 + Cz_0 + D)}{(A\,\Delta x + B\,\Delta y + C\,\Delta z)} \tag{13.20}$$

Jeżeli mianownik równania (13.20) jest równy 0, to promień i płaszczyzna są równoległe i nie przecinają się. Łatwy sposób sprawdzenia, czy punkt przecięcia leży wewnątrz wielokąta, polega na zrzutowaniu wielokąta i punktu prostopadle na jedną z trzech płaszczyzn układu współrzędnych (rys. 13.17). W celu uzyskania dokładnych wyników powinniśmy wybrać oś, wzdłuż której należy wykonać rzutowanie, żeby uzyskać największą powierzchnię rzutu. Odpowiada to współrzędnej, której współczynnik w równaniu płaszczyzny ma największą wartość bezwzględną. Rzut prostopadły wyznacza się na zasadzie pominięcia tej współrzędnej w wierzchołkach wielokąta i w punkcie. Test zawarcia punktu w wielokącie można wtedy wykonać całkowicie w 2D, korzystając z algorytmu naszkicowanego w p. 7.11.2.



Rys. 13.17. Sprawdzanie, czy promień przecina wielokąt. Wielokąt i punkt przecięcia promienia P z wielokątem są rzutowane na trzy płaszczyzny definiujące układ współrzędnych. Sprawdza się, czy zrzutowany punkt P' zawiera się w zrzutowanym wielokącie

Podobnie jak w przypadku algorytmu z-bufora zaletą metody śledzenia promieni jest to, że jedyną wykonywaną operacją przecinania jest znalezienie przecięcia promienia rzutującego z obiektem. Nie ma potrzeby wyznacznia przecięcia dwóch obiektów w scenie. Algorytm z-bufora aproksymuje obiekty jako zbiory wartości z wzdłuż promieni rzutujących, które przecinają obiekt. Metoda śledzenia promieni aproksymuje obiekty jako zbiór przecięć wzdłuż każdego promienia rzutującego, który przecina scenę. Możemy tak rozszerzyć algorytm z-bufora, żeby mógł być stosowany do nowego rodzaju obiektu, pisząc dla niego program konwersji wierszowej i obliczania wartości z. Podobnie możemy tak rozszerzyć program wyznaczający płaszczyzny widoczne metodą śledzenia promieni, żeby można było stosować go do nowego obiektu, po napisaniu dla niego programu wyznaczania przecięcia z promieniem.

W obu przypadkach musimy również napisać program obliczania normalnych do powierzchni potrzebnych do wyznaczania barwy. Algorytmy wyznaczania przecięć i normalnej do powierzchni zostały opracowane dla powierzchni algebraicznych [HANR83] i powierzchni parametrycznych [KAJI82; SEDE84; TOTH85; JOY86]. Przeglądy tych algorytmów można znaleźć w pracach [HAIN89; HANR89].

13.4.2. Problemy efektywności metody śledzenia promieni przy wyznaczaniu powierzchni widocznych

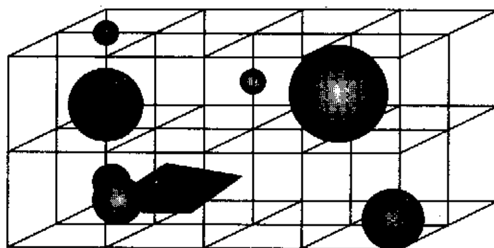
Algorytm z-bufora wyznacza, korzystając ze spójności, w każdym pikselu informację tylko o tych obiektach, których rzut znajduje się w danym pikselu. W omówionej, prostej, ale drogiej wersji algorytmu wyznaczania powierzchni widocznych metodą śledzenia promieni każdy promień wychodzący z oka jest przecinany z każdym obiektem w scenie. Obraz o rozdzielczości 1024 × 1024 zawierający 100 obiektów wymagałby więc wykonania 100 Mprzecięć. Nie jest zaskoczeniem stwierdzenie Whitteda, że dla typowych scen 75-95% i więcej czasu systemu było zużywane na wyznaczanie przecięć [WHIT80]. Konsekwencją tego jest to, że omawiane tutaj podejścia do poprawy efektywności metody śledzenia przy wyznaczaniu powierzchni widocznych dążą do przyspieszenia obliczania jednego przecięcia albo w ogóle do unikania takich obliczeń. Jak zobaczymy w p. 14.7, rekursywny program śledzenia promieni śledzi dodatkowe promienie wychodzące z punktów przecięcia w celu określenia barwy piksela. Dlatego niektóre metody omawiane w p. 13.1, np. przekształcenie perspektywiczne i wybierania tylnych ścian, nie są użyteczne w ogólnym przypadku, ponieważ nie wszystkie promienie są emitowane z tego samego środka rzutowania.

Optymalizacja obliczeń związanych z wyznaczaniem przecięć. Wiele z elementów w równaniach do wyznaczania przecięć zawiera wyrażenia, które są stałe albo w całym obrazie, albo dla określonego promienia. Elementy te mogą być obliczone wcześniej, tak jak na przykład rzut prostokątny wielokąta na płaszczyznę. Przy odpowiedniej analizie można opracować szybkie metody znajdowania przecięć; nawet prosta zależność przecięcia z kulą podana w p. 13.4.1 może być ulepszona [HAIN89]. Jeżeli dokona się takiego przekształcenia promieni, żeby biegły wzdłuż osi z, to to samo przekształcenie może być użyte do każdego obiektu i każde przecięcie występuje dla $x = y = 0$. Ten krok upraszcza obliczanie przecięć i umożliwia określanie najbliższego obiektu w wyniku sortowania względem z. Następnie za pomocą odwrotnego przekształcenia można punkt przecięcia przekształcić z powrotem i dalej wykonywać obliczenia związane z wyznaczaniem barwy.

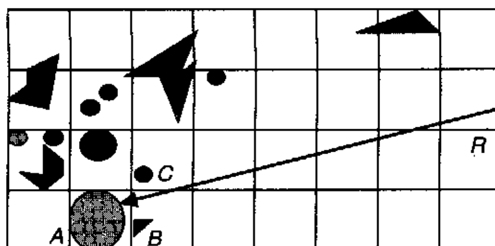
Bryły ograniczające umożliwiają w sposób szczególnie atrakcyjny zmniejszanie ilości czasu potrzebnego do wyznaczania przecięć. Obiekt, który jest względnie drogi, jeśli chodzi o wyznaczanie przecięć, może zostać otoczony przez bryłę ograniczającą, dla której taki test jest znacznie prostszy, na przykład przez kulę [WHIT80], elipsoidę [BOUV85] albo prostopadłościan [RUBI80; TOTH85]. Jeżeli promień nie przecina bryły ograniczającej, to nie trzeba wykonywać testu dla obiektu.

Hierarchie. Chociaż bryły ograniczające same nie określają kolejności albo częstotliwości testów przecięć, mogą być zorganizowane w struktury hierarchiczne z obiektami w liściach i wewnętrznymi węzłami, które otaczają swoich potomków [RUBI80; WEGH84; KAY86]. Mamy pewność, że bryła potomka nie jest przecinana przez promień, jeżeli nie jest przecinana bryła przodka. Dlatego jeżeli zaczniemy testy przecinania od korzenia, to możemy w trywialny sposób odrzucić wiele gałęzi tej hierarchii (a co za tym idzie wiele obiektów).

Podział przestrzenny. Hierarchia brył otaczających organizuje obiekty w porządku wstępującym; podział przestrzenny natomiast dzieli przestrzeń w porządku zstępującym. Najpierw oblicza się prostopadłościan otaczający scenę. W jednym z podejść ten otaczający prostopadłościan jest dzielony na regularną siatkę (rys. 13.18). Z każdym elementem podziału jest związana lista obiektów, które się w nim zawierają, całkowicie albo częściowo. Listy są wypełniane na zasadzie przypisania każdego obiektu do jednego lub więcej elementów podziału, do których ten obiekt należy. Teraz, jak pokazano na rys. 13.19 dla 2D, promień musi być przecięty tylko z tymi obiektami, które są zawarte w elementach podziału, przez które ten promień przechodzi. Ponadto elementy podziału trzeba sprawdzać w kolejności przechodzenia promienia; gdy tylko zostanie znaleziony element podziału, w którym jest przecięcie, nie trzeba sprawdzać dalszych elementów podziału. Zauważmy, że musimy uwzględnić wszystkie pozostałe obiekty w elemencie podziału po to, żeby sprawdzić, które przecięcie jest najbliżej. Ponieważ elementy podziału są utworzone przez regularną siatkę, każdy kolejny element podziału leżący wzdłuż promienia można obliczyć korzystając z wersji



Rys. 13.18. Scena jest podzielona na regularną siatkę brył o takich samych wymiarach

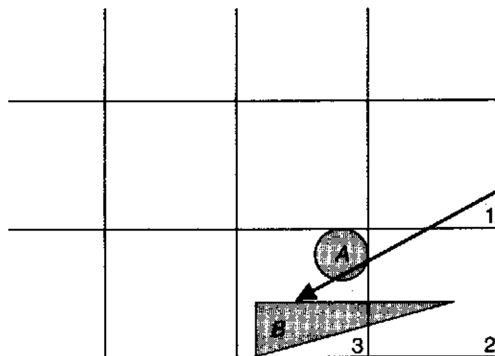


Rys. 13.19. Podział przestrzenny. Promień *R* musi być przecięty tylko z obiektami *A*, *B* i *C*, ponieważ inne elementy podziału, przez które przechodzi, są puste

3D algorytmu rysowania odcinka omówionego w p. 3.2.2, zmodyfikowanego w taki sposób, żeby określać listę wszystkich elementów podziału, przez które promień przechodzi [FUJI85; AMAN87].

Jeżeli promień przecina obiekt w danym elemencie podziału, to trzeba sprawdzić, czy samo przecięcie należy do elementu podziału; może się zdarzyć, że znalezione przecięcie jest już w innym elemencie podziału i że może istnieć bliższe przecięcie dla innego obiektu. Na przykład na rys. 13.20 obiekt *B* jest przecinany w elemencie podziału 3, chociaż został napotkany w elemencie podziału 2. Musimy kontynuować przeglądanie elementów podziału dopóty, dopóki nie znajdziemy przecięcia w bieżąco przeglądanym elemencie podziału, w tym przypadku *A* w elemencie podziału 3. W celu uniknięcia ponownego przeliczania przecięć promienia z obiektem, który jest w kilku elementach podziału, po pierwszym napotkaniu punkt przecięcia i numer identyfikacyjny mogą być wraz z obiektem zapamiętane w pomocniczej pamięci.

Dippe i Swenson [DIPP84] omawiają adaptacyjny algorytm podziału, który tworzy elementy podziału o nierównych wielkościach. W innym adaptacyjnym podziale przestrzennym dokonuje się podziału sceny według metody drzewa ósemkowego [GLAS84]. W tym przypadku w celu

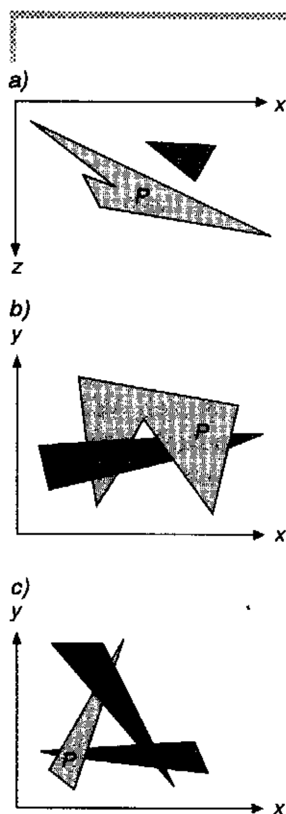


Rys. 13.20. Obiekt może być przecięty w innym wokselu niż bieżący

określenia kolejnych elementów podziału leżących wzdłuż promienia [SAME89b] można zastosować algorytm wyszukiwania sąsiadów w drzewie ósemkowym, naszkicowany w p. 10.6.3. Drzewa ósemkowe i inne hierarchiczne podziały przestrzenne mogą być traktowane jako specjalny przypadek hierarchii, w którym mamy pewność, ze potomkowie węzła wzajemnie nie przecinają się. Ponieważ takie podejście dopuszcza podział adaptacyjny, decyzja o dalszym podziale elementu podziału może zależeć od liczby obiektów w podpodziale albo od kosztu przecinania obiektów. Jest to korzystne w heterogenicznych, nierówno rozłożonych środowiskach.

13.5. Inne rozwiązania

13.5.1. Algorytmy z listą priorytetów



Rys. 13.21. Kilka przypadków, w których przedziały zmian z wielokątów przecinają się

W *algorytmach z listą priorytetów* określa się kolejność wyświetlania obiektów, przy czym zapewnia się to, że jeżeli rendering obiektów będzie następował w tej kolejności, to uzyska się poprawne obrazy. Na przykład, jeżeli żaden obiekt nie nakłada się na inny wzdłuż współrzędnej z, to musimy jedynie wykonać segregowanie obiektów ze względu na rosnące wartości z i wykonywać rendering w tej kolejności. Dalsze obiekty są przesłaniane przez bliższe obiekty, ponieważ piksele z bliższych wielokątów przykrywają te pochodzące z dalszych wielokątów. Jeżeli wystąpi nakładanie się obiektów względem współrzędnej z, to nadal możemy określić poprawny porządek, tak jak na rys. 13.21a. Jeżeli obiekty cyklicznie nakładają się tak jak na rys. 13.21b i c albo przecinają się, to nie istnieje poprawna kolejność. W takich przypadkach będzie konieczne podzielenie jednego lub więcej obiektów po to, żeby umożliwić uzyskanie porządku liniowego.

Algorytmy z listą priorytetów są hybrydami, które łączą operacje z precyzją obiektową i operacje z precyzją obrazową. Porównywanie głębokości i podział obiektów są wykonywane z precyzją obiektową. Jedynie konwersja wierszowa przy przeglądaniu, która zależy od zdolności urządzenia graficznego do zapisywania pikseli na miejscu pikseli należących do poprzednio narysowanych obiektów, jest wykonywana z precyzją obrazową. Ponieważ jednak lista posortowanych obiektów jest tworzona z precyzją obiektową, może być ponownie wyświetlona poprawnie z dowolną rozdzielczością. Jak zobaczymy, algorytmy z listą priorytetową różnią się sposobem określania kolejności w wyniku sortowania, jak również sposobem wyboru obiektu do podziału i czasem dokonywania podziału. Sortowanie nie musi odbywać się względem z; niektóre obiekty mogą być dzielone, chociaż nie nakładają się cyklicznie ani nie przecinają się wzajemnie,

oraz podział może być wykonywany nawet niezależnie od pozycji obserwatora.

Algorytm sortowania ze względu na głębokość. Podstawowa idea algorytmu sortowania ze względu na głębokość, opracowanego przez Newella, Newella i Sancha [NEWE72], polega na rysowaniu wielokątów w pamięci obrazu w kolejności malejącej odległości od punktu obserwacji. Są wykonywane trzy koncepcyjne kroki:

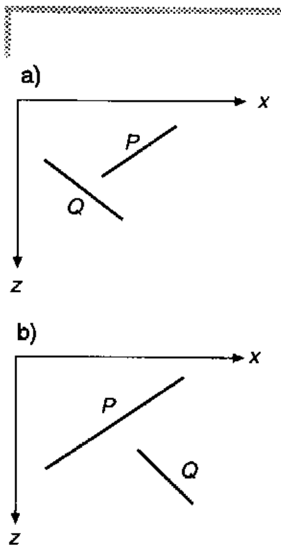
1. Sortowanie wielokątów ze względu na najmniejszą (największą) współrzędną z każdego wielokąta.
2. Rozstrzygnięcie wszystkich niejednoznaczności, które mogą powstawać w wyniku sortowania ze względu na nakładanie się przedziałów wartości z w wielokątach; w razie potrzeby wykonuje się podział wielokątów.
3. Przeglądanie wierszami każdego wielokąta według rosnącej kolejności najmniejszej współrzędnej z (to znaczy od tyłu do przodu).

Rozważmy wykorzystanie bezpośredniego priorytetu, tak jak w przypadku rzutów w SPHIGS. Bezpośredni priorytet zajmuje miejsce minimalnej wartości z i nie może być żadnych niejednoznaczności ze względu na głębokość, ponieważ o każdym priorytecie zakłada się, że może odpowiadać różnym płaszczyznom stałego z. Uproszczona wersja algorytmu sortowania ze względu na głębokość jest często określana jako *algorytm malarski*, z uwagi na podobieństwo do tego, jak malarz może malować bliższe obiekty na wcześniej namalowanych bardziej odległych obiektach. Środowiska, w których każdy obiekt jest w innej płaszczyźnie stałego z, takich jak rozmieszczanie elementów w strukturach VLSI, w kartografii i w zarządzaniu oknami, są określane jako 2 1/2D i można w nich poprawnie stosować algorytm malarski. Algorytm malarski może być wykorzytywany do sceny, w której wielokąty nie znajdują się na płaszczyźnie o stałym z, jeżeli posortuje się wielokąty ze względu na minimalną współrzędną z albo ze względu na współrzędną z ich środka ciężkości, z pominięciem kroku 2. Chociaż taki sposób umożliwia konstruowanie scen, w ogólnym przypadku nie ma gwarancji, że uzyska się poprawne uporządkowanie.

Na rysunku 13.21 pokazano kilka rodzajów niejednoznaczności, które trzeba rozwiązywać w ramach kroku 2. Jak się to robi? Oznaczmy przez P wielokąt będący w danym momencie na końcu posortowanej listy. Zanim dokonamy rasteryzacji tego wielokąta do pamięci obrazu, trzeba go sprawdzić z każdym wielokątem Q, którego zakres wartości z przecina się z zakresem wartości z w wielokącie P, w celu sprawdzenia, że P nie może zasłaniać Q i że P może wobec tego być zapisane przed Q. Wykonuje się do pięciu testów o rosnącej złożoności. Jeżeli jeden z tych testów uda się, to P nie zasłania Q i testuje się następny

wielokąt Q, nakładający się z P względem współrzędnej z. Jeżeli wszystkie takie wielokąty przejdą przez testy, to wykonuje się rasteryzację wielokąta P i kolejny wielokąt na liście staje się nowym wielokątem P. Pięć testów, o których była mowa, wygląda następująco:

1. Czy zakresy współrzędnych x wielokątów nie przecinają się?
2. Czy zakresy współrzędnych y wielokątów nie przecinają się?
3. Czy P jest całkowicie po stronie płaszczyzny Q niewidocznej dla obserwatora? (Tak nie jest na rys. 13.21a, natomiast jest to prawdą dla rys. 13.22a.)
4. Czy Q jest całkowicie po tej samej stronie płaszczyzny P co obserwator? (Tak nie jest na rys. 13.21a, natomiast jest to prawdą dla rys. 13.22b.)
5. Czy rzuty wielokątów na płaszczyznę (x, y) nakładają się? Można to określić porównując krawędzie jednego wielokąta z krawędziami drugiego.



Rys. 13.22. Możliwe orientacje wielokątów: a) test 3 jest prawdziwy, b) test 3 nie jest spełniony, natomiast test 4 jest spełniony

Jeżeli żaden z pięciu testów nie będzie spełniony, to zakładamy na moment, że P rzeczywiście zasłania Q, i sprawdzamy, czy można dokonać rasteryzacji Q przed P. Testów 1, 2 i 5 nie trzeba powtarzać, korzysta się natomiast z nowych wersji testów 3 i 4 z zamienionymi wielokątami:

3'. Czy Q jest całkowicie po stronie płaszczyzny P niewidocznej dla obserwatora?

4'. Czy P jest całkowicie po tej samej stronie płaszczyzny Q co obserwator?

W przypadku z rys. 13.21a test 3' udaje się. Dlatego przesuwamy Q na koniec listy i staje się on nowym P. W przypadku z rys. 13.21b testy w dalszym ciągu nie są rozstrzygające; w rzeczywistości nie istnieje kolejność, w jakiej można by dokonać poprawnej rasteryzacji. Dlatego trzeba dokonać podziału P albo Q przez płaszczyznę drugiego wielokąta (porównajmy z p. 3.11 poświęconym obcinaniu wielokąta i potraktujmy krawędź obcinania jako płaszczyznę obcinającą). Oryginalny nie podzielony wielokąt jest odrzucany, a jego części są umieszczane na liście we właściwej kolejności ze względu na z i algorytm jest wykonywany jak poprzednio.

Na rysunku 13.21c pokazano bardziej subtelny przypadek. Istnieje taka orientacja P, Q i R, że każdy wielokąt zawsze może być przesunięty na koniec listy po to, żeby ustawić go poprawnie względem jednego, ale nie obu pozostałych wielokątów. Prowadziłoby to do nieskończonej pętli. W celu uniknięcia zapętlenia musimy zmodyfikować nasze podejście przez oznaczanie każdego wielokąta przesuwanego na koniec listy. Wtedy za każdym razem, kiedy pierwszych pięć testów zawodzi i bieżący wielokąt jest oznaczony, nie próbujemy testów 3' i 4'. Zamiast tego

dzielimy albo P, albo Q (tak jak gdyby oba testy 3′ i 4′ nie udały się) i ponownie umieszczamy powstałe części w liście.

Drzewa binarnego podziału przestrzeni. Algorytm drzewa binarnego podziału przestrzeni (BSP) był opracowany przez Fuchsa, Kedema i Naylora [FUCH80; FUCH83]; korzystali oni z pracy Schumackera [SCHU69]. Algorytm drzewa BSP jest bardzo wydajną metodą obliczania zależności widoczności wśród statycznej grupy wielokątów 3D oglądanych z dowolnego punktu obserwacji. Zapewnia on kompromis między początkowym czasochłonnym i pamięciochłonnym przetwarzaniem wstępnym a liniowym algorytmem wyświetlania, który jest wykonywany za każdym razem, gdy jest potrzebna nowa specyfikacja rzutowania. Dlatego algorytm dobrze się nadaje do zastosowań, w których zmienia się punkt obserwacji, natomiast same obiekty nie zmieniają się.

W algorytmie drzewa BSP skorzystano ze spostrzeżenia, że rasteryzacja wielokąta zostanie wykonana poprawnie (to znaczy nie będzie on zasłaniał niepoprawnie ani nie będzie zasłaniany niepoprawnie), jeżeli wszystkie wielokąty znajdujące się po stronie niewidocznej dla obserwatora zostaną poddane konwersji wcześniej, potem zostanie wykonana konwersja tego wielokąta, a następnie wszystkich wielokątów znajdujących się po tej samej stronie co obserwator. Musimy zapewnić to dla każdego wielokąta.

Algorytm zapewnia łatwe określenie poprawnej kolejności rasteryzacji dzięki tworzeniu drzewa binarnego wielokątów (drzewo BSP). W korzeniu drzewa BSP jest wielokąt wybrany spośród wielokątów, które mają być wyświetlone; algorytm działa poprawnie niezależnie od pierwotnego wyboru wielokąta. Wielokąt znajdujący się w korzeniu jest wykorzystywany do podziału środowiska na dwie półprzestrzenie. Jedna półprzestrzeń zawiera wszystkie pozostałe wielokąty znajdujące się przed wielokątem korzenia (zależy to od zwrotu normalnej wielokąta korzenia); w drugiej półprzestrzeni są umieszczane wszystkie wielokąty znajdujące się za wielokątem korzenia. Każdy wielokąt, który leży po obu stronach płaszczyzny wielokąta będącego w korzeniu, jest dzielony przez tę płaszczyznę, a jego przednie i tylne części są przypisane do odpowiednich półprzestrzeni. Jeden wielokąt w każdej półprzestrzeni, przedniej i tylnej względem wielokąta będącego w korzeniu, staje się jego przednim albo tylnym potomkiem i każdy potomek jest rekursywnie wykorzystywany do przydzielenia pozostałych wielokątów do jego półprzestrzeni w podobny sposób. Algorytm kończy się, gdy każdy węzeł zawiera tylko jeden wielokąt.

Godne uwagi jest to, że drzewo BSP może być przeglądane w zmodyfikowany sposób tak, żeby uzyskać poprawnie uporządkowaną według priorytetów listę dla dowolnego punktu obserwacji. Weźmy pod uwagę wielokąt będący w korzeniu. Dzieli on pozostałe wielokąty na dwa zbiory, z których każdy leży całkowicie po jednej stronie płaszczyzny wielokąta znajdującego się w korzeniu. Algorytm musi gwarantować

wyświetlanie zbiorów w poprawnym, względnym porządku po to, żeby zapewnić zarówno to, żeby wielokąty z jednego zbioru nie mieszały się z innymi, jak i to, żeby wielokąt będący korzeniem był wyświetlany poprawnie i we właściwej kolejności względem innych. Jeżeli obserwator jest w przedniej półprzestrzeni względem wielokąta znajdującego się w korzeniu, to algorytm musi najpierw wyświetlić wszystkie wielokąty tylnej półprzestrzeni względem korzenia (te, które mogłyby zostać zasłonięte przez wielokąt znajdujący się w korzeniu), potem wielokąt znajdujący się w korzeniu i wreszcie wszystkie wielokąty z przedniej półprzestrzeni (te, które mogłyby zasłonić wielokąt znajdujący się w korzeniu). W przypadku gdy obserwator jest w tylnej półprzestrzeni, algorytm musi najpierw wyświetlić wszystkie wielokąty z przedniej półprzestrzeni korzenia, potem wielokąt znajdujący się w korzeniu i wreszcie wszystkie wielokąty w jego tylnej półprzestrzeni. Wybieranie ścian tylnych można zrealizować na zasadzie niewyświetlania wielokąta, jeżeli oko jest w jego tylnej półprzestrzeni. Każdy z potomków korzenia jest przetwarzany rekursywnie za pomocą tego algorytmu. Pseudokod zarówno dla fazy tworzenia drzewa, jak i dla fazy wyświetlania jest podany w książce [FOLE90].

Podobnie jak w algorytmie sortowania ze względu na głębokość, algorytm z drzewem BSP wykonuje przecinanie i sortowanie z precyzją obiektową i wykorzystuje możliwości urządzenia rastrowego do zapisywania na poprzedniej zawartości z precyzją obrazową. Inaczej niż przy sortowaniu ze względu na głębokość, wykonuje się tu wszystkie podziały wielokątów w fazie przetwarzania wstępnego, która musi być powtarzana tylko wówczas, gdy zmienia się środowisko. Zauważmy, że może się tu pojawić więcej wielokątów dzielonych niż w metodzie sortowania ze względu na głębokość.

Algorytmy z listą priorytetów umożliwiają korzystanie ze sprzętowej rasteryzacji wielokątów, która z reguły zapewnia większą szybkość niż przy sprawdzaniu głębokości w każdym pikselu. Algorytmy sortowania ze względu na głębokość i z drzewem BSP wyświetlają wielokąty w kolejności od tyłu do przodu, przy czym ewentualnie te później wyświetlane zasłaniają te odleglejsze. Dlatego podobnie jak w algorytmie *z*-bufora obliczenia związane z wyznaczeniem barwy mogą być wykonywane więcej niż raz dla piksela. Wielokąty mogą być również wyświetlane w kolejności od przodu do tyłu i każdy piksel wielokąta może być zapisywany tylko wówczas, gdy jeszcze nie był zapisany.

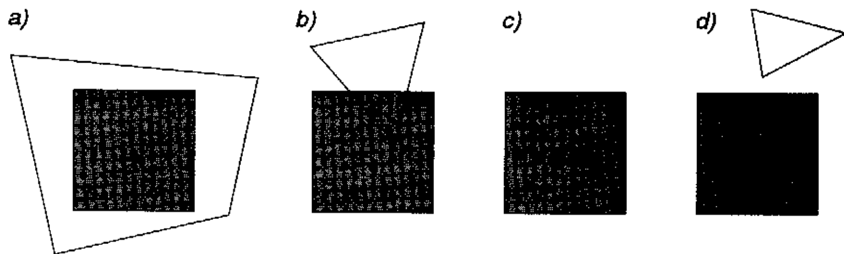
Jeżeli do usuwania linii niewidocznych jest wykorzystywany algorytm z listą priorytetów, to trzeba zwrócić specjalną uwagę na nowe krawędzie wprowadzane przez proces dzielenia. Jeżeli te krawędzie są poddawane rasteryzacji podobnie jak krawędzie oryginalnego wielokąta, to pojawią się w obrazie jako niepożądane zakłócenia; trzeba je więc oznaczać po to, żeby je omijać przy rasteryzacji.

13.5.2. Algorytmy podziału powierzchni

Wszystkie *algorytmy podziału powierzchni* wykorzystują strategię dziel i zwyciężaj podziału przestrzennego na rzutni. Sprawdza się powierzchnię zrzutowanego obrazu. Jeżeli można łatwo zdecydować, które wielokąty w tym obszarze są widoczne, to są one wyświetlane. W przeciwnym przypadku powierzchnia jest dzielona na mniejsze powierzchnie, do których rekursywnie stosuje się proces decyzji logicznych. Gdy powierzchnia maleje, wówczas mniej wielokątów pokrywa każdą powierzchnię i w końcu staje się możliwe podjęcie decyzji. W tym podejściu wykorzystuje się spójność powierzchniową, ponieważ odpowiednio małe obszary obrazu będą zawarte w najwyżej jednym widocznym wielokącie.

Algorytm Warnocka. Algorytm podziału powierzchni opracowany przez Warnocka [WARN69] dzieli każdy obszar na cztery jednakowe kwadraty. W każdym kroku rekursywnego procesu podziału rzut każdego wielokąta może się znaleźć w jednej z czterech relacji względem odpowiedniego obszaru (rys. 13.23):

1. *Wielokąty otaczające* całkowicie zawierają rozpatrywany obszar (pocieniowany kwadrat na rys. 13.23a.
2. *Wielokąty przecinające* przecinają obszar (rys. 13.23b).
3. *Wielokąty zawarte* są całkowicie wewnątrz obszaru (rys. 13.23c).
4. *Wielokąty rozłączne* są całkowicie na zewnątrz obszaru (rys. 13.23d).



Rys. 13.23. Cztery relacje między rzutami wielokąta a elementem obszaru: a) otaczanie; b) przecinanie; c) zawieranie; d) rozłączność

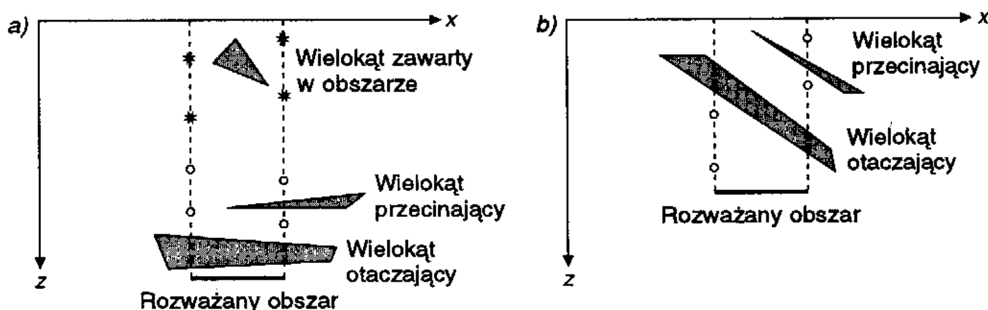
Wielokąty rozłączne oczywiście nie mają żadnego wpływu na rozważany obszar. Część wielokąta przecinającego, która jest na zewnątrz obszaru, również nie jest istotna, natomiast część wielokąta przecinającego, która jest wewnątrz obszaru, powinna być traktowana tak samo jak wielokąt zawarty w obszarze.

W czterech przypadkach można łatwo podjąć decyzję co do obszaru i nie musi on być dalej dzielony:

1. Wszystkie wielokąty są rozłączne względem obszaru. Obszar można wypełnić barwą tła.
2. Jest tylko jeden wielokąt przecinający lub tylko jeden wielokąt zawarty. Obszar jest najpierw wypełniany barwą tła, a potem wykonuje się rasteryzację wielokąta zawartego lub części wielokąta przecinającego.
3. Jest jeden wielokąt otaczający i nie ma wielokątów przecinających ani zawartych. Obszar jest wypełniany barwą wielokąta otaczającego.
4. Jest więcej niż jeden wielokąt przecinający, zawarty albo otaczający obszar, ale tylko jeden wielokąt otaczający jest przed wszystkimi innymi wielokątami.

Sprawdzenie, czy wielokąt otaczający jest z przodu, wykonuje się na zasadzie obliczania współrzędnych z płaszczyzn wszystkich wielokątów otaczających, przecinających i zawartych we wszystkich czterech rogach obszaru; jeżeli jest wielokąt otaczający, dla którego współrzędne z czterech rogów są większe (bliższe obserwatorowi) niż dla każdego innego wielokąta, to cały obszar może być wypełniony barwą tego wielokąta otaczającego.

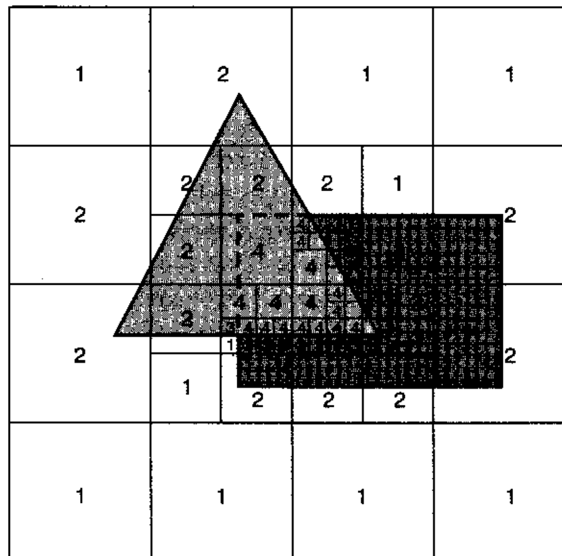
Przypadki 1, 2 i 3 są łatwe do zrozumienia. Przypadek 4 jest dodatkowo zilustrowany na rys. 13.24. W części a) wszystkie cztery przecięcia wielokąta otaczającego są bliższe punktu obserwacji (który jest w nieskończoności na osi z) niż jakiekolwiek inne przecięcia. W konsekwencji cały obszar jest wypełniony barwą wielokąta otaczającego. W części b) nie można podjąć decyzji, chociaż wielokąt otaczający wydaje się być z przodu wielokąta przecinającego, ponieważ z lewej strony płaszczyzna wielokąta przecinającego jest przed płaszczyzną wielokąta otaczającego. Zauważmy, że algorytm z sortowaniem głębokości akceptuje ten przypadek bez dalszego podziału, jeżeli wielokąt przecinający jest całkowicie



Rys. 13.24. Dwa przykłady przypadku 4 w podziale rekursywnym: a) wielokąt otaczający jest w każdym rogu bliższy rozpatrywanego obszaru; b) płaszczyzna wielokąta przecinającego jest bliższa z lewej strony rozważanego obszaru; znak × oznacza przecięcie z płaszczyzną wielokąta otaczającego; znak ○ oznacza przecięcie z płaszczyzną wielokąta przecinającego; znak * oznacza przecięcie z płaszczyzną wielokąta zawartego w obszarze

po tej stronie wielokąta otaczającego, która jest dalej od punktu obserwacji. Algorytm Warnocka zawsze jednak dzieli obszar w celu uproszczenia problemu. Po podziale trzeba ponownie sprawdzić tylko wielokąty zawarte i przecinające: wielokąty otaczające i rozłączne względem oryginalnego obszaru są wielokątami otaczającymi i rozłącznymi każdego obszaru otrzymanego w wyniku podziału.

Do tego momentu algorytm działał z precyzją obiektową, z wyjątkiem rasteryzacji tła i wielokątów obciętych w czterech przypadkach. Jednak te operacje rasteryzacji z precyzją obrazową można zastąpić operacjami z precyzją obiektową, które dają dokładną reprezentację widocznych powierzchni: albo kwadrat o wielkości obszaru (przypadki 1, 3 i 4), albo wielokąt obcięty do obszaru, wraz z boolowskim uzupełnieniem względem obszaru reprezentującego widzialną część tła (przypadek 2). A co z przypadkami innymi niż te cztery? Jedno z rozwiązań polega na zatrzymaniu podziału po osiągnięciu rozdzielczości urządzenia wyświetlającego. Dlatego w urządzeniu wyświetlającym o rozdzielczości rastra 1024 × 1024 trzeba najwyżej 10 poziomów podziału. Jeżeli po tej maksymalnej liczbie podziałów nie wystąpi żaden z przypadków od 1 do 4, to oblicza się głębokość wszystkich odpowiednich wielokątów w środku tego nie podzielonego obszaru o wielkości piksela. Wielokąt o najbliższej współrzędnej z określa barwę obszaru. Można też ze względu na problem eliminacji zakłóceń użyć kilku dalszych poziomów podziału w celu określenia barwy piksela na zasadzie przypisania każdemu obszarowi podpikselowemu wag zależnych od wielkości obszaru. Ta właśnie



Rys. 13.25. Podział obszaru na kwadraty

operacja wykonywana wówczas, gdy dla obszaru nie zachodzi jeden z prostych przypadków, w efekcie decyduje o tym, że jest to podejście z precyzją obrazową.

Na rysunku 13.25 pokazano prostą scenę i podział potrzebny do jej wyświetlenia. Liczba w każdym obszarze podziału odpowiada jednemu z czterech przypadków; tam gdzie nie jest wpisana żadna liczba, nie zachodzi żaden z tych czterech przypadków. Można porównać to podejście z podziałem przestrzennym 2D wykonywanym przez drzewa czwórkowe (p. 10.6.3).

13.5.3. Algorytmy wyznacznia powierzchni krzywoliniowych

Wszystkie dotychczas prezentowane algorytmy, z wyjątkiem z-bufora, były zdefiniowane dla ścian wielokątowych. Obiekty takie jak powierzchnie krzywoliniowe z rozdz. 9 muszą być najpierw aproksymowane za pomocą wielu małych ścianek, zanim można użyć wielokątowej wersji któregoś z algorytmów. Chociaż można zrobić taką aproksymację, często lepiej jest bezpośrednio wykonać rasteryzację powierzchni krzywoliniowej, eliminując zakłócenia wielokątowe i dodatkową pamięć potrzebną do aproksymacji wielokątowej.

W grafice komputerowej są popularne powierzchnie drugiego stopnia, omawiane w p. 9.4. Algorytmy wyznaczania powierzchni widocznych dla powierzchni drugiego stopnia zostały opracowane przez Weissa [WEIS66], Woona [WOON71], Mahla [MAHL72], Levina [LEVI76] i Sarraga [SARR83]. Znajdują oni przecięcia dwóch powierzchni drugiego stopnia, co prowadzi do równań czwartego stopnia dla zmiennych x, y i z, których pierwiastki trzeba znajdować numerycznie. Levin redukuje to do problemu drugiego stopnia przez parametryzowanie krzywych przecięcia. Kule, będące specjalnym przypadkiem powierzchni drugiego stopnia, są najłatwiejsze z tego punktu widzenia; są one szczególnie interesujące ze względu na to, że cząsteczki są często wyświetlane jako zbiory barwnych kul (por. fot. 22). Opracowano wiele algorytmów wyświetlania cząsteczek [KNOW77; STAU78; MAX79; PORT79; FRAN81; MAX84]. W punkcie 13.4 jest mowa o tym, jak wykonać rendering kuli metodą śledzenia promieni.

Jeszcze większą elastyczność można osiągnąć w przypadku parametrycznych powierzchni sklejanych omawianych w rozdz. 9, ponieważ są one bardziej ogólne i umożliwiają zapewnienie ciągłości stycznej na granicach płatów. Catmull [CATM74; CATM75] jako pierwszy opracował algorytm wyświetlania dla powierzchni bikubicznych. Zgodnie z algorytmem Warnocka płat jest rekursywnie dzielony w kierunkach s i t na cztery płaty dopóty, dopóki ich rzuty nie pokrywają więcej niż 1 piksel.

Algorytm z-bufora określa, czy płat jest widoczny w tym pikselu. Jeżeli tak, to oblicza się barwę i umieszcza ją w pamięci obrazu. Program 13.4 zawiera pseudokod dla tego algorytmu. Ponieważ sprawdzanie wielkości samego płatu jest czasochłonne, można w zamian użyć czworokąta określonego przez wierzchołki płatu.

Program 13.4
Pseudokod dla algorytmu rekursywnego podziału Catmulla

```
for (każdy płat) {
  umieszczenie płatu na stosie;
  while (stos nie jest pusty) {
    zdjęcie płatu ze stosu;
    if (płat pokrywa mniej niż 1 piksel) {
      if (piksel płatu jest bliższy w sensie z)
        określenie barwy i rysowanie;
    }
    else {
      podział płatu na cztery podpłaty;
      umieszczenie podpłatów na stosie;
    }
  }
}
```

W innym podejściu wykorzystuje się adaptacyjny podział każdego płata bikubicznego dopóty, dopóki dzielony płat nie znajdzie się w zakresie przyjętej tolerancji płaskości. Ta tolerancja zależy od rozdzielczości urządzenia wyświetlającego i orientacji dzielonego obszaru względem rzutni, tak że jest eliminowane niepotrzebne dzielenie. Płat może być dzielony tylko w jednym kierunku, jeżeli jest już wystarczająco płaski w drugim kierunku. Po dostatecznym podziale płat może być traktowany jak czworokąt. Małe wielokąty określone przez cztery rogi każdego płatu są przetwarzane za pomocą algorytmu przeglądania wierszami, co umożliwia mieszanie powierzchni wielokątowych i bikubicznych. Algorytmy wykorzystujące tę podstawową ideę zostały opracowane przez Lane'a i Carpentera [LANE80] oraz Clarka [CLAR79]; są one opisane w książce [FOLE90].

Podsumowanie

Sutherland, Sproull i Schumacker [SUTH74a] podkreślają, że istotą problemu wyznaczania powierzchni widocznych jest sortowanie. Pokazaliśmy wiele wariantów sortowania i przeglądania w algorytmach – efektywne sortowanie jest krytycznym elementem dla wyznaczania powierzchni widocznych. Równie ważne jest unikanie sortowania ponad to, co jest rzeczywiście niezbędne; osiąga się ten cel na ogół dzięki wykorzystaniu spójności. Na przykład algorytm przeglądania wierszami wykorzys-

tuje spójność w wierszu w celu wyeliminowania konieczności pełnego sortowania ze względu na x dla każdego przeglądanego wiersza.

Algorytmy można klasyfikować ze względu na kolejność sortowania. Algorytm typu zstępującego sortuje względem z, a potem względem x i y (dzięki wykorzystaniu przedziałów zmian w testach 1 i 2); dlatego jest określany jako algorytm zxy. Algorytmy przeglądania wierszy sortują względem y (metodą sortowania grupowego), potem następuje sortowanie względem x (początkowo wykorzystuje się sortowanie przez wstawienie, potem w czasie przeglądania każdego wiersza metodą bąbelkową) i wreszcie przeszukiwanie względem z w celu znalezienia wielokąta najbliższego punktowi obserwacji; są to więc algorytmy yxz. Algorytm Warnocka wykonuje równoległe sortowanie względem x i y, a potem następuje szukanie względem z; zatem jest to algorytm $(xy)z$ (sortowanie względem kombinacji wymiarów jest zaznaczone przez nawiasy). Algorytm z-bufora nie wykonuje bezpośredniego sortowania i szuka tylko względem z; jest on określany jako algorytm (xyz).

Sancha dowodził, że kolejność sortowania nie jest istotna: nie ma istotnych korzyści z sortowania najpierw względem którejś osi, ponieważ, przynajmniej w zasadzie, przeciętny obiekt jest równie złożony we wszystkich trzech kierunkach [SUTH74a]. Scena graficzna jednak, taka jak obraz Hollywood, może być tak skonstruowana, żeby wyglądała najlepiej z jakiegoś punktu obserwacji, a to może wymagać większej złożoności przy budowaniu sceny wzdłuż jednej osi niż wzdłuż innych. Nawet jeżeli założymy prawie symetryczną złożoność obiektu, to nadal nie wszystkie algorytmy są równie efektywne; różnią się efektywnością wykorzystania spójności w celu unikania sortowania i innych obliczeń oraz kompromisami, jeśli chodzi o wykorzystanie pamięci i czasu. Wyniki podane w pracy [SUTH74a, tabl. VII], które porównują szacowaną wydajność czterech podstawowych algorytmów przez nas omówionych, są podsumowane w tabl. 13.1. Autorzy sugerują, że ponieważ są to tylko oszacowania, niewielkie różnice powinny być pominięte, ale że: „uważamy, że możemy wykonać porównania różnych algorytmów po to, żeby dowiedzieć się czegoś o efektywności różnych metod" [SUTH74a, str. 52].

Tablica 13.1 Względne oszacowanie wydajności czterech algorytmów określania powierzchni widocznych

	Liczba wielokątów w scenie		
Algorytm	100	2500	60 000
Sortowanie ze względu na głębokość	1*	10	507
z-bufor	54	54	54
Przeglądanie wierszy	5	21	100
Podział obszaru Warnocka	11	64	307

* Elementy w tablicy zostały tak znormalizowane, żeby w tym miejscu było 1.

Algorytm sortowania ze względu na głębokość jest efektywny dla niedużej liczby wielokątów, ponieważ proste testy nakładania się prawie zawsze wystarczają do podjęcia decyzji, czy wielokąt może być poddany rasteryzacji. Przy większej liczbie wielokątów częściej są potrzebne bardziej złożone testy i jest bardziej prawdopodobne, że będzie potrzebny dodatkowy podział. Algorytm z-bufora charakteryzuje się stałą wydajnością, ponieważ – gdy rośnie liczba wielokątów w scenie, maleje liczba pikseli pokrywanych przez jeden wielokąt. Są jednak duże wymagania co do pamięci. Indywidualne testy i obliczenia wykorzystywane w algorytmie podziału Warnocka są relatywnie bardziej złożone i jest on z reguły wolniejszy niż inne metody. Obok tych nieformalnych oszacowań były również prowadzone prace związane z formalizacją problemu powierzchni widocznych i analizą ich złożoności obliczeniowej [GILO78; FOUR88; FIUM89]. Na przykład Fiume [FIUM89] dowodzi, że algorytmy powierzchni widocznych z precyzją obiektową mają dolne ograniczenie gorsze niż w przypadku sortowania.

Ogólnie porównywanie algorytmów wyznaczania powierzchni widocznych jest trudne, ponieważ nie wszystkie algorytmy obliczają tę samą informację z tą samą dokładnością. Na przykład omawialiśmy algorytmy, które ograniczają rodzaje obiektów, zależności między obiektami, a nawet dopuszczalne rodzaje rzutów. Jak zobaczymy w następnym rozdziale, wybór algorytmu wyznaczania powierzchni widocznych zależy również od wymaganego rodzaju cieniowania. Jeżeli jest wykorzystywana kosztowna procedura wyznaczania barw, to lepiej jest wybrać algorytm wyznaczania powierzchni widocznych, który cieniuje tylko części widoczne obiektów, tak jak w przypadku algorytmu z przeglądaniem wierszy. W tym przypadku wybór algorytmu sortowania ze względu na głebokość byłby złym wyborem, ponieważ zakłada on rysowanie wszystkich obiektów w całości. Jeżeli jest ważna praca interakcyjna, to popularne jest korzystanie ze sprzętowego algorytmu z-bufora. Algorytm drzewa BSP może szybko generować nowe rzuty środowiska statycznego, ale wymaga dodatkowego przetwarzania przy zmianach środowiska. Algorytmy z przeglądaniem wierszy dopuszczają wyjątkowo dużą rozdzielczość, ponieważ struktury danych muszą reprezentować w pełni obliczone wersje tylko tych prymitywów, które mają wpływ na przeglądany wiersz. Tak jak w dowolnym algorytmie istotnymi czynnikami są również czas poświęcony na implementację algorytmu i łatwość jego modyfikacji (na przykład w celu uwzględnienia nowego prymitywu).

Istotnym elementem przy implementacji algorytmu wyznaczania powierzchni widocznych jest rodzaj wspomagania sprzętowego. Jeżeli jest dostępny komputer równoległy, to musimy zauważyć, że w każdym miejscu, gdzie algorytm korzysta ze spójności, są potrzebne wyniki

poprzednich obliczeń. Wykorzystywanie równoległości może powodować pomijanie skądinąd użytecznej formy spójności. Metoda śledzenia promieni jest szczególnie przydatna do implementacji równoległej, ponieważ w najprostszej postaci każdy piksel jest obliczany niezależnie.

Zadania

13.1. Udowodnij, że przekształcenie M w p. 13.1.2 zachowuje: a) odcinki, b) płaszczyzny i c) zależności związane z głębokością.

13.2. Dla danej płaszczyzny $Ax + By + Cz + D = 0$ zastosuj przekształcenie M z punktu 13.1.2 i znajdź nowe współczynniki równania płaszczyzny.

13.3. Jak można rozszerzyć algorytm przeglądania wierszy, żeby mógł dawać sobie radę ze wspólnymi krawędziami? Czy wspólna krawędź powinna być reprezentowana raz jako wspólna krawędź, czy dwa razy jako brzeg każdego wielokąta, bez pamiętania informacji o tym, że jest to wspólna krawędź? Gdy obliczy się głębokość dwóch wielokątów dla ich wspólnej krawędzi, wówczas te głebokości będą oczywiście takie same. Który wielokąt powinien być zadeklarowany jako widoczny wówczas, gdy przy przeglądaniu dochodzi się do nich obu?

13.4. Wyjaśnij, jak poszczególne algorytmy: z-bufora, sortowania ze względu na głębokość, Warnocka i drzewa BSP będą się zachowywały w przypadku wąskiego wielokąta. Czy takie wielokąty są szczególnym przypadkiem, który musi być traktowany niezależnie, czy też mogą być obsługiwane przez podstawowy algorytm?

13.5. Jak algorytmy wymienione w zadaniu 13.4 mogą być zaadaptowane do wielokątów zawierających dziury?

13.6. Jedną z zalet algorytmu z-bufora jest to, że prymitywy mogą być rozpatrywane w dowolnej kolejności. Czy oznacza to, że dwa obrazy otrzymane po wysłaniu prymitywów w różnej kolejności będą miały identyczne wartości w swoich z-buforach i pamięciach obrazu? Wyjaśnij odpowiedź.

13.7. Rozważ problem mieszania dwóch obrazów o identycznej wielkości reprezentowanych przez zawartości w z-buforach i pamięciach obrazu. Jeżeli znamy z_{min} i z_{max} każdego obrazu i wartości, którym początkowo odpowiadają, to, czy możemy poprawnie zmieszać te obrazy? Czy jest potrzebna jakaś dodatkowa informacja?

13.8. W punkcie 13.2 są poruszone problemy kompresji wartości z powodowane przez rendering rzutu perspektywicznego z wykorzystaniem całkowitoliczbowego z-bufora. Należy wybrać parametry rzutu perspektywicznego i pewną liczbę punktów obiektu. Pokaż, jak w przekształceniu perspektywicznym dwa punkty leżące blisko środka rzutowania są odwzorowywane na różne wartości z, podczas gdy dwa punkty znaj-

dujące się względem siebie w tej samej odległości, ale umieszczone dalej od środka rzutowania są odwzorowywane na jedną wartość z.

13.9. a. Załóż, że bryła widzenia V ma płaszczyzny obcinające przednią i tylną odpowiednio w odległościach F i B (obie wartości dodatnie!) od punktu VRP, mierzone wzdłuż kierunku DOP. Załóż, że odległość od COP do VRP mierzona wzdłuż DOP wynosi w oraz że VRP jest między COP a tylną płaszczyzną obcinającą (tak jak na rys. 6.16). Określ $f = w - F$ i $b = w + B$ tak, żeby f było odległością od COP do płaszczyzny przedniej, a b odległością od COP do płaszczyzny tylnej. Zrób teraz to samo z bryłą widzenia V' i określ podobnie f' i b'. Po przekształceniu w kanoniczną bryłę widzenia tylna płaszczyzna obcinająca V znajdzie się w $z = -1$, a przednia w $z = A$. Podobnie dla bryły V' płaszczyzna przednia znajdzie się w $z = A'$. Pokaż, że jeżeli $f/b = f'/b'$, to $A = A'$ i odwrotnie. Krótko mówiąc, zakres wartości z po przekształceniu w kanoniczną bryłę widzenia zależy tylko od stosunku odległości od COP do płaszczyzn przedniej i tylnej.

b. Część a zadania pokazuje, że przy rozważaniu wpływu perspektywy należy wziąć pod uwagę tylko stosunek odległości płaszczyzn przedniej i tylnej (od COP). Dlatego można po prostu badać kanoniczną bryłę widzenia o różnych wartościach odległości płaszczyzny przedniej. Załóż wobec tego, że masz kanoniczną perspektywiczną bryłę widzenia z przednią płaszczyzną obcinającą $z = A$ i tylną płaszczyzną obcinającą $z = -1$ i dokonaj jej przekształcenia perspektywicznego w równoległą bryłę widzenia między $z = 0$ i $z = -1$. Napisz wzór określający przekształconą współrzędną z w zależności od początkowej współrzędnej z. (Odpowiedź będzie oczywiści zależała od A). Załóż, że przekształcone wartości z w równoległej bryle widzenia są pomnożone przez 2^n, a potem zaokrąglone do wartości całkowitej (to znaczy są odwzorowywane do całkowitoliczbowego z-bufora). Znajdź dwie wartości z, które są tak daleko, jak to tylko jest możliwe, ale które po tym przekształceniu są odwzorowywane na tę samą liczbę całkowitą. (Odpowiedź będzie zależała od n i od A).

c. Załóż, że trzeba uzyskać obraz, w którym stosunek f do b jest równy R a obiekty, które są w większej odległości niż Q (wzdłuż osi z) mają być odwzorowane na różne wartości w z-buforze. Korzystając z wyników z części b zadania napisz wzór określający potrzebną liczbę bitów w z-buforze.

13.10. W czasie śledzenia promienia na ogół trzeba tylko obliczyć, czy promień przecina przedział zmian, a niekoniecznie rzeczywiste punkty przecięcia. Przedstaw równanie przecięcia promień-kula (wzór (13.16)), korzystając z równania kwadratowego i pokaż, jak można je uprościć, żeby sprawdzić, czy promień i kula przecinają się.

13.11. Śledzenie promieni może być również wykorzystane do wyznaczenia właściwości obiektów za pomocą całkowania numerycznego. Pełny zbiór przecięć promienia z obiektem daje całkowitą część promienia, która jest wewnątrz obiektu. Pokaż, jak można oszacować objętość bryły prowadząc strumień równoległych promieni przez ten obiekt.

13.12. Znajdź przecięcie promienia z powierzchnią drugiego stopnia. Zmodyfikuj metodę używaną do znalezienia przecięcia z kulą (wzory (13.12)-(13.15)), tak żeby było możliwe wykorzystanie definicji powierzchni drugiego stopnia podanej w p. 9.4.

13.13. Zaimplementuj jeden z podanych w tym rozdziale algorytmów wyznaczania powierzchni widocznych dla wielokątów (na przykład algorytm z-bufora albo algorytm przeglądania wierszami).

13.14. Zaimplementuj prosty algorytm śledzenia promieni dla kul i wielokątów. (Wybierz jeden z modeli oświetlenia z p. 14.1.) Popraw parametry programu korzystając z podziału przestrzennego albo hierarchi ograniczających brył.

14. Oświetlanie i cieniowanie

W tym rozdziale omawiamy, jak określać barwę powierzchni biorąc pod uwagę położenie, orientację oraz charakterystyki powierzchni i oświetlających je źródeł światła. Pokażemy kilka różnych *modeli oświetlenia*, które opisują czynniki określające barwę powierzchni w danym punkcie. Modele oświetlenia są często określane jako *modele świateł* albo *modele cieniowania*. Tutaj jednak będziemy traktowali określenie *model cieniowania* jako ogólniejszy w stosunku do modelu oświetlenia. Model cieniowania określa, kiedy jest stosowany model oświetlenia i jakie otrzymuje on argumenty. Na przykład w niektórych modelach cieniowania jest wykorzystywany model oświetlenia dla każdego piksela w obrazie, w innych zaś korzysta się z modelu oświetlenia tylko dla niektórych pikseli, a barwa pozostałych pikseli jest określana na zasadzie interpolacji.

Porównując w poprzednim rozdziale dokładność wykonywania obliczeń przy wyznaczaniu powierzchni widocznych, rozróżnialiśmy algorytmy, które wykorzystują bieżącą geometrię obiektu, od tych, które wykorzystują aproksymację wielokątową oraz od algorytmów z precyzją obrazową i z precyzją obiektową. We wszystkich jednak przypadkach jedynym kryterium określania bezpośredniej widzialności obiektu w obszarze piksela jest to, czy coś leży między obiektem a obserwatorem wzdłuż promienia rzutującego przechodzącego przez piksel. Interakcja między światłami i powierzchniami jest znacznie bardziej skomplikowana. Badacze zajmujący się grafiką komputerową często aproksymowali odpowiednie reguły optyki i radiacji termicznej albo w celu uproszczenia obliczeń, albo ze względu na to, że dokładniejsze modele nie były znane w środowisku ludzi zajmu-

jących się grafiką. W konsekwencji wiele modeli oświetlania i cieniowania tradycyjnie wykorzystywanych w grafice komputerowej zawiera wiele trików i uproszczeń, które nie mają uzasadnienia w teorii, ale zupełnie dobrze potwierdzają się w praktyce. W pierwszej części tego rozdziału omówiono te proste modele, które są wciąż w powszechnym użyciu, ze względu na to, że umożliwiają uzyskiwanie atrakcyjnych i użytecznych wyników przy minimalnych nakładach obliczeniowych.

Zaczynamy w p. 14.1 od omówienia prostych modeli oświetlenia, w których bierze się pod uwagę jeden punkt powierzchni i źródła światła oświetlające go bezpośrednio. Najpierw pokażemy modele oświetlenia dla monochromatycznych świateł i powierzchni , a potem pokażemy, jak te obliczenia można uogólnić na systemy barw omówione w rozdz. 11. W punkcie 14.2 opisano najpopularniejsze modele cieniowania, które są używane z tymi modelami oświetlenia. W punkcie 14.3 rozszerzono te modele o symulację powierzchni teksturowanych.

Modelowanie załamania, odbicia i cieniowania wymaga dodatkowych obliczeń, które są bardzo podobne do eliminowania powierzchni niewidocznych; często te dwa zagadnienia są łączone ze sobą. Te efekty pojawiają się, ponieważ niektóre z powierzchni niewidocznych nie są tak na prawdę całkowicie niewidoczne – są one widziane przez inne powierzchnie, są odbijane od innych powierzchni albo rzucają cienie na powierzchnię cieniowaną! W punktach 14.4 i 14.5 jest mowa o tym, jak modelować niektóre z tych efektów.

W punktach 14.6 do 14.8 opisano *ogólne modele oświetlenia*, w których bierze się pod uwagę odbicia światła między wszystkimi powierzchniami: metodę rekursywnego śledzenia promieni i metodę energetyczną. Rekursywne śledzenie promieni rozszerza algorytm śledzenia promieni dla powierzchni widocznych wprowadzony w poprzednim rozdziale tak, żeby można było połączyć zagadnienia widzialności, oświetlenia i cieniowania w każdym pikselu. Metody energetyczne modelują równowagę energii w systemie powierzchni; określają one oświetlenie zbioru próbkowanych punktów w środowisku w sposób niezależny od położenia obserwatora, przed określeneim powierzchni widocznych z danego punktu obserwacji. Dokładniejsze omówienie różnych modeli oświetlenia i cieniowania można znaleźć w pracach [GLAS89; HALL89].

Wreszcie w p. 14.9 przyjrzymy się kilku różnym potokom graficznym integrującym metody rasteryzacji omówione w tym i w poprzednich rozdziałach. Pokażemy niektóre sposoby implementowania tych możliwości w celu utworzenia systemów zarówno efektywnych, jak i możliwych do rozbudowywania.

14.1. Modele oświetlenia

14.1.1. Światło otoczenia

Prawdopodobnie najprostszy możliwy model oświetlenia to taki model, który pośrednio był używany we wcześniejszych rozdziałach tej książki: każdy obiekt jest wyświetlany z charakterystycznym dla niego natężeniem. O tym modelu, który nie zawiera zewnętrznych źródeł światła, możemy myśleć jako o modelu opisującym raczej nierealistyczny świat obiektów nie odbijających i samoświecących. Każdy obiekt wygląda jak monochromatyczny szkielet dopóty, dopóki jego poszczególne części, np. wielokąty wielościanu, nie otrzymają różnych barw przy tworzeniu obiektu. Efekt ten pokazano na fot. 27.

Model oświetlenia może być wyrażony za pomocą *równania oświetlenia* dla zmiennych związanych z punktem na cieniowanym obiekcie. Równanie oświetlenia dla tego prostego modelu ma następującą postać:

$$I = k_i \tag{14.1}$$

przy czym I jest wynikowym natężeniem światła, a współczynnik k_i jest własnym natężeniem obiektu. Ponieważ równanie oświetlenia nie zawiera wyrazów, które zależą od położenia cieniowanego punktu, możemy obliczyć je raz dla całego obiektu. Proces obliczania równania oświetlenia w jednym albo w wielu punktach obiektu jest często określany jako *oświetlanie obiektu.*

Wyobraźmy sobie teraz, że zamiast samoświecenia jest rozproszone bezkierunkowe źródło światła, które wytwarza liczne odbicia światła od wielu powierzchni znajdujących się w środowisku. Jest to tak zwane *światło otoczenia.* Jeżeli założymy, że światło otoczenia pada jednakowo na wszystkie powierzchnie ze wszystkich kierunków, to równanie oświetlenia przyjmie postać

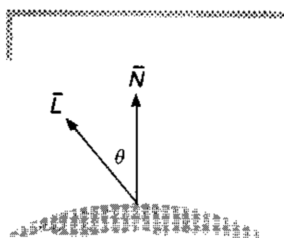
$$I = I_a k_a \tag{14.2}$$

przy czym I_a jest natężeniem światła otoczenia, o którym zakłada się, że jest stałe dla wszystkich obiektów. Natężenie światła otoczenia odbitego od powierzchni obiektu jest określone współczynnikiem odbicia światła rozproszonego k_a zmieniającego się w przedziale od 0 do 1. Współczynnik odbicia światła rozproszonego jest właściwością materiału. Razem z innymi właściwościami materiału, o których będziemy mówili, może on być traktowany jako cecha materiału, z którego jest wykonana powierzchnia. Podobnie jak niektóre inne właściwości

współczynnik odbicia światła otoczenia jest wprowadzony ze względu na doświadczalną wygodę i nie odpowiada bezpośrednio żadnej fizycznej właściwości rzeczywistych materiałów. Co więcej, samo światło otoczenia nie jest specjalnie interesujące. Jak zobaczymy później, jest ono wykorzystywane w celu uwzględnienia wszystkich złożonych dróg, którymi światło może dotrzeć do obiektu, które nie są w żaden inny sposób uwzględniane w równaniu oświetlenia. Oświetlenie przez światło otoczenia pokazano na fot. 27.

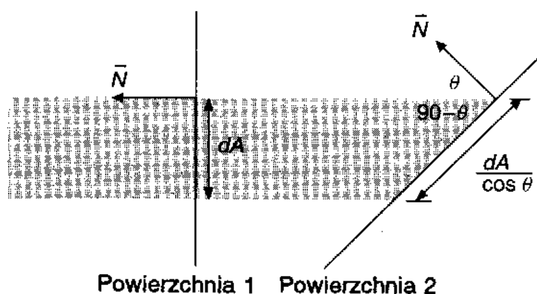
14.1.2. Odbicie rozproszone

Chociaż obiekty oświetlane światłem otoczenia świecą słabiej lub silniej, wprost proporcjonalnie do natężenia światła otoczenia, wciąż są oświetlone jednolicie na całej powierzchni. Rozważmy teraz oświetlenie obiektu przez *punktowe źródło światła*, którego promienie rozchodzą się równomiernie we wszystkich kierunkach z jednego punktu. Jasność obiektu zmienia się od jednej części do drugiej, zależnie od kierunku i odległości od źródła światła.



Rys. 14.1. Odbicie rozproszone

Odbicie lambertowskie. Matowe powierzchnie bez połysku, np. kreda, charakteryzują się odbiciem rozproszonym, znanym również jako *odbicie lambertowskie*. Takie powierzchnie wydają się równie jasne ze wszystkich kierunków obserwacji. Dla danej powierzchni jasność zależy tylko od kąta θ między kierunkiem $\overline{L}$ do źródła światła i normalną do powierzchni $\overline{N}$ (rys. 14.1). Zobaczmy, dlaczego tak się dzieje. Odgrywają tu rolę dwa czynniki. Po pierwsze, na rys. 14.2 widać, że strumień światła padający na powierzchnię pokrywa obszar, którego wielkość jest odwrotnie proporcjonalna do kosinusa kąta θ, tworzonego przez strumień z $\overline{N}$. Jeżeli strumień światła ma nieskończenie mały przekrój dA, to obszar przecięcia powierzchni przez strumień wynosi $dA/\cos\theta$. Dlatego dla padającej wiązki światła strumień światła, który pada na



Rys. 14.2. Strumień (pokazany w przekroju 2D) o nieskończenie małym przekroju dA padający pod kątem θ przecina obszar $dA/\cos\theta$

jednostkę powierzchni oświetlanej, jest proporcjonalny do $\cos\theta$. Jest to prawdziwe dla każdej powierzchni niezależnie od materiału, z którego jest wykonana.

Po drugie, musimy wziąć pod uwagę strumień światła dochodzący do oka obserwatora. Dla powierzchni lambertowskich obowiązuje właściwość określana jako *prawo Lamberta*, zgodnie z którym natężenie światła odbitego od powierzchni elementarnej dA w kierunku obserwatora jest wprost proporcjonalne do kosinusa kąta między kierunkiem do obserwatora a $\overline{N}$. Ponieważ pole obserwowanej powierzchni jest odwrotnie proporcjonalne do kosinusa tego kąta, te dwa czynniki znoszą się. Na przykład, gdy kąt widzenia rośnie, obserwator widzi większy fragment powierzchni, ale natężenie światła odbijane pod tym kątem przez jednostkę powierzchni jest proporcjonalnie mniejsze. Dlatego dla powierzchni lambertowskich stosunek natężenia światła widzianego przez obserwatora do powierzchni elementarnej jest niezależny od kierunku patrzenia i jest proporcjonalny tylko do $\cos\theta$, przy czym θ jest kątem padania światła.

Równanie oświetlenia dla odbicia rozproszonego ma postać

$$I = I_p k_d \cos\theta \tag{14.3}$$

przy czym I_p jest natężeniem światła (światłością) punktowego źródła światła; współczynnik odbicia rozproszonego materiału k_d jest stałą o wartości z przedziału od 0 do 1 i jest różny dla różnych materiałów. Kąt θ musi być między 0 a 90°, jeżeli źródło światła ma mieć bezpośredni wpływ na cieniowany punkt. Oznacza to, że traktujemy powierzchnię jako samoprzesłaniającą się i światło rzucone z punktu leżącego za powierzchnią nie oświetla jej. Zamiast bezpośrednio uwzględniać tutaj i w następnych równaniach czynnik $\max(\cos\theta, 0)$, zakładamy, że θ leży w dopuszczalnym zakresie. Gdy chcemy oświetlić powierzchnię samozasłaniającą się, możemy użyć $\mathrm{abs}(\cos\theta)$ w celu odwrócenia normalnej do powierzchni dla θ z zakresu 90 do 180°. Powoduje to takie samo traktowanie obu stron powierzchni, jak gdyby powierzchnia była oświetlona przez dwa przeciwne światła.

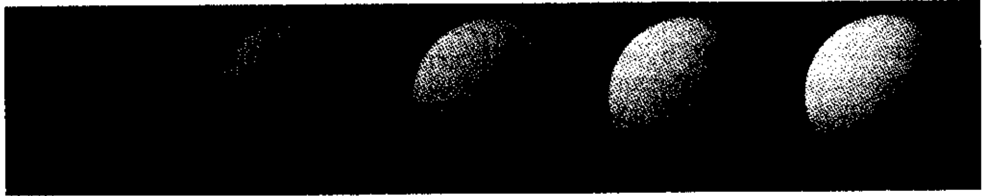
Przy założeniu, że wektory $\overline{N}$ i $\overline{L}$ zostały znormalizowane (por. p. 5.1), możemy przepisać równanie (14.3) korzystając z iloczynu skalarnego

$$I = I_p k_d (\overline{N} \cdot \overline{L}) \tag{14.4}$$

Normalną do powierzchni $\overline{N}$ można obliczyć korzystając z metod omówionych w rozdz. 9. Jeżeli normalne do wielokątów zostały wcześniej obliczone i przekształcone za pomocą tej samej macierzy, która jest wykorzystywana do wierzchołków wielokąta, to nie należy wykonywać takich przekształceń modelowania, jak pochylenia albo niejednorodne skalowanie; te przekształcenia nie zachowują kątów i mogą prowadzić

do tego, że niektóre normalne nie będą już prostopadłe do swoich wielokątów. Odpowiednia metoda przekształcania normalnych dla dowolnych przekształceń obiektu jest opisana w p. 5.7. W każdym przypadku równanie oświetlenia trzeba obliczać w układzie WC (albo w każdym innym układzie do niego izometrycznym), ponieważ zarówno przekształcenie normalizujące, jak i przekształcenie perspektywiczne będzie modyfikowało kąt θ.

Gdy punktowe źródło światła jest dostatecznie daleko od cieniowanego obiektu, wówczas kąt między promieniem z tego źródła a wszystkimi powierzchniami mającymi tę samą normalną jest w zasadzie stały. W tym przypadku mówimy o *kierunkowym źródle światła* i $\overline{L}$ jest stałe dla źródła światła.

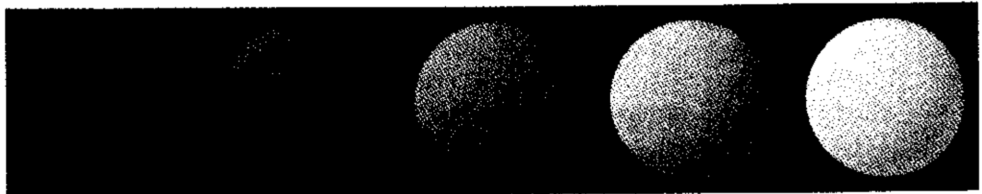


Rys. 14.3. Kule cieniowane za pomocą modelu odbicia rozproszonego (równanie (14.4)). Od lewej do prawej: k_d = 0,4, 0,55, 0,7, 0,85, 1,0. (Za zgodą Davida Kurlandera z Columbia University.)

Na rysunku 14.3 pokazano ciąg obrazów kuli oświetlonej punktowym źródłem światła. Jasność każdego piksela, w którym kula była widoczna, została obliczona za pomocą modelu oświetlenia z równania (14.4). Obiekty oświetlone w ten sposób wyglądają nienaturalnie, tak jak gdyby światło flesza oświetlało obiekt w ciemnym pomieszczeniu. Dlatego na ogół dodaje się światło otoczenia po to, żeby uzyskać bardziej realistyczne równanie oświetlenia

$$I = I_a k_a + I_p k_d (\overline{N} \cdot \overline{L}) \tag{14.5}$$

Równanie (14.5) zostało wykorzystane do wykonania rys. 14.4.



Rys. 14.4. Kule cieniowane za pomocą modelu światła otoczenia i odbicia rozproszonego (wzór (14.5)). Dla wszystkich kul: $I_a = I_p = 1{,}0$, $k_d = 0{,}4$. Od lewej do prawej: k_a = 0,0, 0,15, 0,30, 0,45, 0,60. (Za zgodą Davida Kurlandera z Columbia University.)

Tłumienie źródła światła. Jeżeli rzuty dwóch równoległych powierzchni z identycznego materiału nakładają się w obrazie, to równanie (14.5) nie określa, gdzie kończy się jedna powierzchnia, a gdzie zaczyna druga, niezależnie od ich odległości od źródła światła. Żeby to obejść, wprowadzamy współczynnik tłumienia źródła światła f_{att} i otrzymujemy równanie

$$I = I_a k_a + f_{att} I_p k_d (\overline{N} \cdot \overline{L}) \tag{14.6}$$

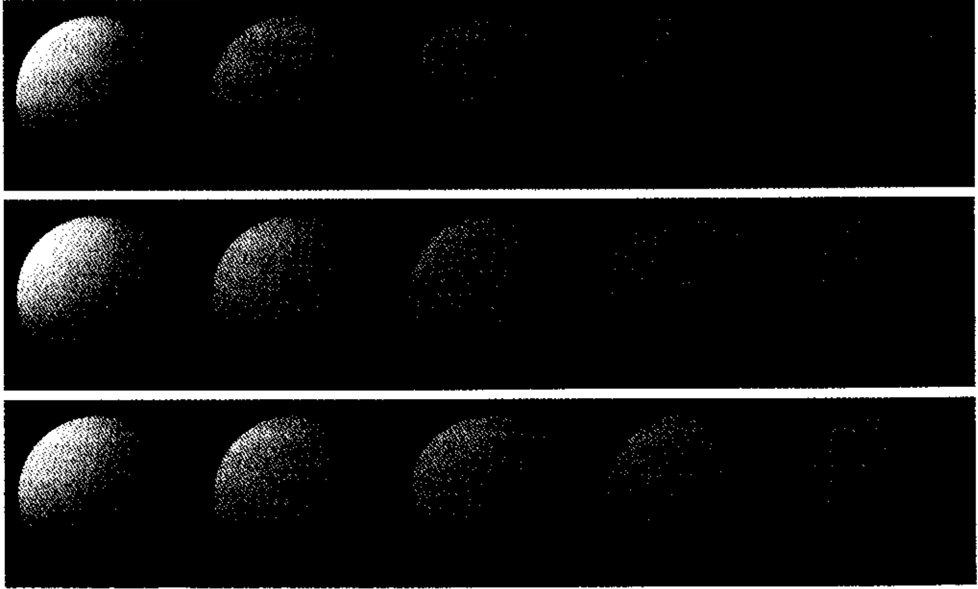
Oczywistym elementem branym pod uwagę przy wyborze f_{att} jest to, że strumień światła z punktowego źródła światła, który osiąga daną część powierzchni, maleje odwrotnie proporcjonalnie do kwadratu d_L, czyli odległości, jaką przebywa światło od źródła punktowego do powierzchni. W tym przypadku

$$f_{att} = \frac{1}{d_L^2} \tag{14.7}$$

Zastosowanie wzoru (14.7) w praktyce często nie daje dobrych efektów. Jeżeli źródło jest daleko, to $1/d_L^2$ zmienia się w niewielkim stopniu; jeżeli jest bardzo blisko, to zmienia się bardzo i powstają istotne różnice barw dla powierzchni o takim samym kącie θ między $\overline{N}$ i $\overline{L}$. Chociaż to zachowanie jest poprawne dla punktowego źródła światła, obiekty, które widzimy w świecie rzeczywistym, na ogół nie są oświetlane przez źródła punktowe i nie są cieniowane za pomocą uproszczonych modeli oświetlenia stosowanych w grafice komputerowej. Żeby sprawę jeszcze skomplikować, w początkach rozwoju grafiki badacze umieszczali punktowe źródło światła dokładnie w miejscu, w którym znajdował się punkt obserwacji. Dążyli oni do tego, żeby czynnik f_{att} aproksymował niektóre efekty tłumienia atmosferycznego między obserwatorem a obiektem. Rozsądnym kompromisem, który umożliwia zwiększenie zakresu efektów w porównaniu z prostym tłumieniem z kwadratem odległości, jest przyjęcie, że

$$f_{att} = \min\left(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1\right) \tag{14.8}$$

Tutaj c_1, c_2 i c_3 są stałymi określanymi przez użytkownika, związanymi ze źródłem światła. Stała c_1 zapewnia to, że mianownik nie stanie się zbyt mały, gdy światło jest blisko, a całe wyrażenie jest ograniczone od



Rys. 14.5. Kule cieniowane z wykorzystaniem światła otoczenia i odbiciem rozproszonym z uwzględnieniem czynnika odpowiedzialnego za tłumienie źródła światła (równania (14.6) i (14.8)). Dla wszystkich kul: $I_a = I_p = 1{,}0$, $k_a = 0{,}1$, $k_d = 0{,}9$. Od lewej do prawej odległość kuli od źródła światła wynosi 1,0, 1,375, 1,75, 2,125, 2,5. W górnym rzędzie $c_1 = c_2 = 0{,}0$, $c_3 = 1{,}0(1/d_L^2)$. W środkowym rzędzie: $c_1 = c_2 = 0{,}25$, $c_3 = 0{,}5$. W dolnym rzędzie: $c_1 = 0{,}0$, $c_2 = 1{,}0$, $c_3 = 0{,}0(1/d_L^2)$. (Za zgodą Davida Kurlandera z Columbia University.)

góry przez 1 po to, żeby zapewnić zawsze tłumienie. Na rysunku 14.5 wykorzystano ten model oświetlenia z różnymi stałymi do pokazania różnych możliwych do uzyskania efektów.

Światła barwne i powierzchnie. Dotychczas opisywaliśmy światła i powierzchnie monochromatyczne. Barwne światła i powierzchnie są na ogół opisywane przez oddzielne równania pisane dla każdej składowej modelu barw. Barwę odbicia rozproszonego obiektu reprezentujemy przez jedną wartość O_d dla każdej składowej. Na przykład, trójka (O_{dR}, O_{dG}, O_{dG}) określa współczynniki odbicia rozproszonego dla składowych czerwonej, zielonej i niebieskiej w systemie barw RGB. W tym przypadku trzy składowe podstawowe I_{pR}, I_{pG} i I_{pB} są odbijane odpowiednio w proporcjach $k_d O_{dR}$, $k_d O_{dG}$ i $k_d O_{dB}$. Stąd, dla składowej czerwonej

$$I_R = I_{aR} k_a O_{dR} + f_{att} I_{pR} k_d O_{dR} (\overline{N} \cdot \overline{L}) \tag{14.9}$$

Podobne równania są wykorzystywane dla I_G i I_B, składowych zielonej i niebieskiej. Wykorzystanie jednego współczynnika do skalowania wyrażenia w każdym z równań umożliwia użytkownikowi sterowa-

nie współczynnikami odbicia światła otoczenia i odbicia rozproszonego bez zmiany proporcji ich składników. Inne sformułowanie jest bardziej zwarte, ale mniej wygodne do sterowania; wykorzystuje się w nim oddzielne współczynniki dla każdego elementu, na przykład podstawiając k_{aR} zamiast $k_a O_{dR}$ i k_{dR} zamiast $k_d O_{dR}$.

Przyjmuje się tutaj upraszczające założenie, że trzy składowe modelu barw mogą całkowicie modelować oddziaływanie światła z obiektami. To założenie nie jest słuszne, ale umożliwia łatwą implementację i często daje akceptowalne obrazy. W teorii równanie oświetlenia powinno być przeliczane w sposób ciągły dla modelowanego zakresu widmowego; w praktyce jest ono przeliczane tylko dla pewnej liczby dyskretnych próbek widmowych. Zamiast ograniczać się do określonego modelu barw, bezpośrednio wskazujemy te człony w równaniu oświetlenia, które są zależne od długości fali przypisując im indeks λ. Wobec tego równanie (14.9) przyjmuje postać

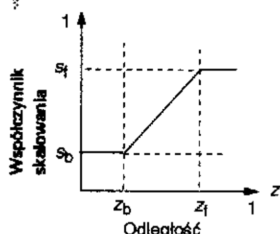
$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{\mathrm{att}} I_{p\lambda} k_d O_{d\lambda} (\overline{N} \cdot \overline{L}) \tag{14.10}$$

14.1.3. Tłumienie atmosferyczne

W celu symulowania tłumienia atmosferycznego między obiektem a obserwatorem w wielu systemach wykorzystuje się metodę tłumienia w funkcji odległości. W tej metodzie, która ma swe początki w sprzęcie grafiki komputerowej, dalsze obiekty są odtwarzane z mniejszym natężeniem niż bliższe. Standard PHIGS PLUS zaleca stosowanie tego podejścia; umożliwia ono również przybliżenie przesunięcia barw wynikającego z istnienia atmosfery. W NPC definiuje się przednią i tylną płaszczyznę odniesienia dla tłumienia w funkcji odległości; z każdą z tych płaszczyzn jest związany współczynnik skalowania s_f i s_b zmieniający się od 0 do 1. Współczynnik skalowania określa przechodzenie od początkowego natężenia do natężenia $I_{dc\lambda}$ wynikającego z tłumienia barwy w funkcji odległości. Celem jest zmodyfikowanie poprzednio obliczonej wartości I_λ tak, żeby otrzymać wartość I'_λ dla wyświetlania wynikającą z tłumienia w funkcji odległości. Dla danej wartości z_0 współrzędnej z obiektu wyznacza się współczynnik skalowania s_0, który będzie wykorzystany do interpolowania między I_λ a I_{dc}, i otrzymujemy

$$I'_\lambda = s_0 I_\lambda + (1 - s_0) I_{dc\lambda} \tag{14.11}$$

Jeżeli z_0 jest przed przednią płaszczyzną odniesienia dla tłumienia w funkcji głębokości, dla której współrzędna z jest równa z_f, to $s_0 = s_f$.

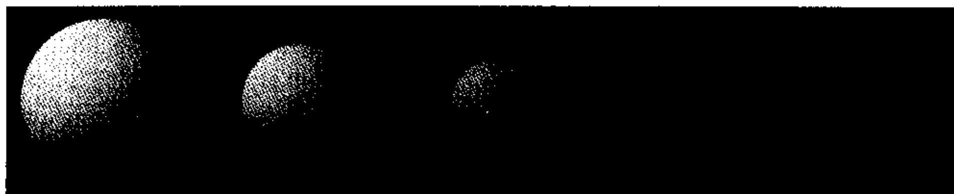


Rys. 14.6. Obliczanie współczynnika skalowania dla tłumienia atmosferycznego

Jeżeli z_0 jest za tylną płaszczyzną odniesienia o współrzędnej z_b, to $s_0 = s_b$. Wreszcie, jeżeli z_0 jest między płaszczyznami, to

$$s_0 = s_b + \frac{(z_0 - z_b)(s_f - s_b)}{z_f - z_b} \tag{14.12}$$

Zależność między s_0 i z_0 pokazano na rys. 14.6 Na rysunku 14.7 pokazano kule cieniowane z uwzględnieniem tłumienia w funkcji odległości. W celu uniknięcia komplikowania równań przy dalszym rozwijaniu modelu oświetlenia pomijamy efekt tłumienia w funkcji odległości.

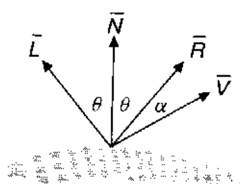


Rys. 14.7. Kule cieniowane z wykorzystaniem efektu tłumienia w funkcji odległości (równania (14.5), (14.11) i (14.12)). Odległość od światła jest stała. Dla wszystkich kul: $I_a = I_p = 1{,}0$, $k_a = 0{,}1$, $k_d = 0{,}9$, $z_f = 1{,}0$, $z_b = 0{,}0$, $s_f = 1{,}0$, $s_b = 0{,}1$, promień = 0,09. Od lewej do prawej z dla przodu kuli wynosi 1,0, 0,77, 0,55, 0,32, 0,09. (Za zgodą Davida Kurlandera z Columbia University.)

14.1.4. Odbicie zwierciadlane

Odbicie zwierciadlane można zaobserwować na każdej błyszczącej powierzchni. Oświetlmy jabłko jasnym białym światłem; rozświetlenie jest spowodowane odbiciem zwierciadlanym, a światło odbite od reszty jabłka jest wynikiem odbicia rozproszonego. Zauważmy również, że w miejscu rozświetlenia jabłko wydaje się nie czerwone, ale białe tak jak padające światło. Obiekty takie jak woskowane jabłka albo błyszczące plastyki mają przezroczyste powierzchnie; na przykład plastyki na ogół składają się z cząsteczek pigmentu zanurzonych w przezroczystym materiale. Światło odbite zwierciadlanie od bezbarwnej powierzchni ma prawie taką samą barwę jak źródło światła.

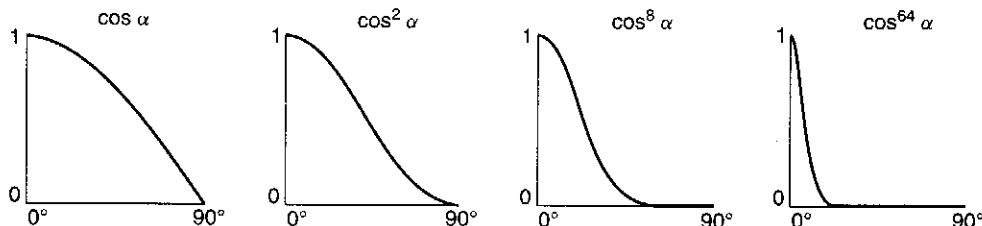
Poruszmy teraz głową i zauważmy, jak porusza się rozświetlenie. Dzieje się tak, ponieważ błyszczące powierzchnie odbijają światło niejednakowo w różnych kierunkach; na idealnie błyszczącej powierzchni, np. na idealnym zwierciadle, światło odbija się tylko w kierunku odbicia $\overline{R}$, będącym zwierciadlanym odbiciem $\overline{L}$ względem $\overline{N}$. Dlatego obser-



Rys. 14.8. Odbicie zwierciadlane

wator może widzieć światło odbite zwierciadlanie od lustra tylko wówczas, gdy kąt α na rys. 14.8 jest równy zeru; α jest kątem między $\overline{R}$ a kierunkiem do obserwatora $\overline{V}$.

Model oświetlenia Phonga. Phong Bui-Tuong [BUIT75] opracował popularny model oświetlenia dla nieidealnych obiektów odbijających, takich jak jabłko. W modelu zakłada się, że maksimum odbicia zwierciadlanego występuje dla α równego zeru i szybko spada ze wzrostem kąta α. Ten szybki spadek jest aproksymowany przez $\cos^n \alpha$, przy czym n jest *wykładnikiem odbicia zwierciadlanego* charakterystycznym dla danego materiału. Wartości n zmieniają się typowo od 1 do kilkuset, zależnie od symulowanego materiału. Dla wartości 1 występuje szeroki łagodny spadek, a większe wartości symulują ostre, zogniskowane rozświetlenie (rys. 14.9). Dla idealnego obiektu odbijającego n byłoby równe nieskończoności. Podobnie jak poprzednio ujemną wartość $\cos \alpha$ traktujemy jak zero. W modelu oświetlenia Phong wykorzystał wcześniejsze prace innych badaczy, na przykład Warnocka [WARN69], który stosował czynnik $\cos^n \theta$ do modelowania odbicia zwierciadlanego przy założeniu, że źródło światła jest w punkcie, w którym znajduje się obserwator. Jednak Phong był pierwszym, który uwzględniał różne położenia dla obserwatora i światła.



Rys. 14.9. Różne wartości $\cos^n \alpha$ w modelu oświetlenia Phonga

Stosunek natężenia światła odbijanego zwierciadlanie do natężenia światła padającego może również zależeć od kąta padania θ. Jeżeli oznaczymy go przez $W(\theta)$, to model Phonga wygląda następująco:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} \cos\theta + W(\theta) \cos^n \alpha] \quad (14.13)$$

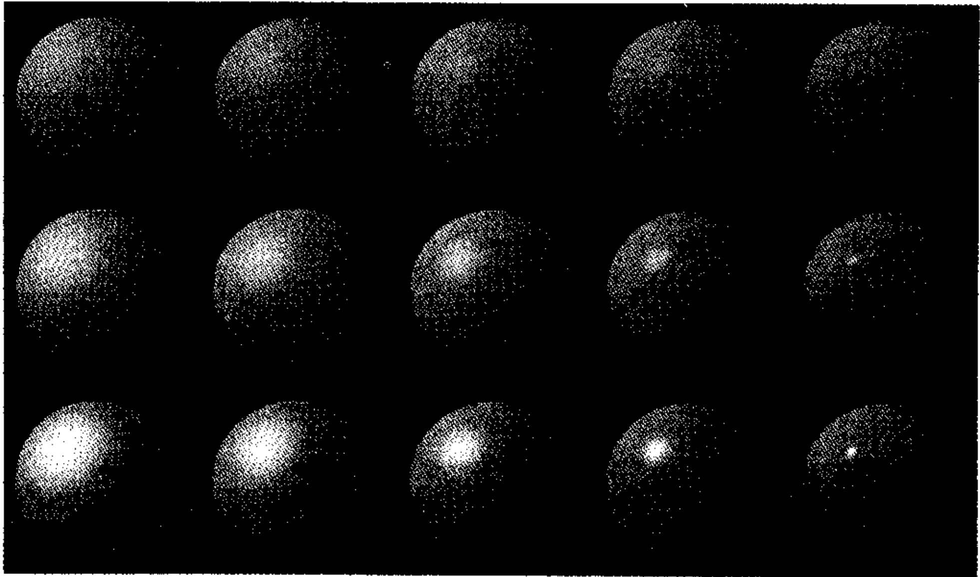
Jeżeli wektory odbicia $\overline{R}$ i obserwacji $\overline{V}$ są znormalizowane, to $\cos\alpha = \overline{R} \cdot \overline{V}$. Dodatkowo $W(\theta)$ na ogół ma stałą wartość k_s, określaną jako *współczynnik odbicia zwierciadlanego*, i zawartą w przedziale 0 do 1. Wartość k_s jest wybierana eksperymentalnie tak, żeby uzyskać dobry wynik. Wtedy równanie (14.13) można przepisać w postaci

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} (\overline{N} \cdot \overline{L}) + k_s (\overline{R} \cdot \overline{V})^n] \quad (14.14)$$

Zauważmy, że barwa składnika odbicia zwierciadlanego w modelu Phonga nie zależy od właściwości materiału; dlatego ten model dobrze nadaje się do modelowania odbić zwierciadlanych od powierzchni plastykowych. Jak pokażemy w p. 14.1.7, odbicie zwierciadlane zależy od właściwości samej powierzchni i ogólnie może mieć inną barwę niż dla odbicia rozproszonego, gdy powierzchnia materiału jest złożona z kilku substancji. Możemy uwzględnić ten efekt w pierwszym przybliżeniu modyfikując wzór (14.14) w następujący sposób:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{\text{att}} I_{p\lambda} [k_d O_{d\lambda} (\overline{N} \cdot \overline{L}) + k_s O_{s\lambda} (\overline{R} \cdot \overline{V})^n] \qquad (14.15)$$

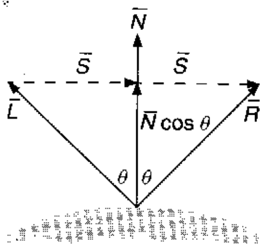
przy czym $O_{s\lambda}$ jest barwą odbicia zwierciadlanego obiektu. Na rysunku 14.10 pokazano oświetloną kulę uzyskaną za pomocą równania (14.14) dla różnych wartości k_s i n.



Rys. 14.10. Kule cieniowane za pomocą modelu oświetlenia Phonga (równanie (14.14)) dla różnych wartości k_s i n. Dla wszystkich kul: $I_a = I_p = 1{,}0$, $k_a = 0{,}1$, $k_d = 0{,}45$. Od lewej do prawej $n = 3{,}0, 5{,}0, 10{,}0, 27{,}0, 200{,}0$. Od góry do dołu $k_s = 0{,}1, 0{,}25, 0{,}5$. (Za zgodą Davida Kurlandera z Columbia University.)

Obliczanie wektora odbicia. W celu obliczenia $\overline{R}$ musimy odbić zwierciadlanie $\overline{L}$ względem $\overline{N}$. Jak pokazano na rys. 14.11, można to zrobić korzystając z prostych przekształceń geometrycznych. Ponieważ $\overline{N}$ i $\overline{L}$ są znormalizowane, rzut $\overline{L}$ na $\overline{N}$ jest równy $\overline{N} \cos\theta$. Zauważmy, że $\overline{R} = \overline{N} \cos\theta + \overline{S}$, przy czym $|\overline{S}|$ jest równe $\sin\theta$. Korzystając z odej-

mowania wektorów i podobieństwa trójkątów otrzymujemy, że $\overline{S}$ jest po prostu równe $\overline{N}\cos\theta - \overline{L}$. Stąd $\overline{R} = 2\overline{N}\cos\theta - \overline{L}$. Podstawiając $\overline{N}\cdot\overline{L}$ zamiast $\cos\theta$ i $\overline{R}\cdot\overline{V}$ zamiast $\cos\alpha$ otrzymujemy:



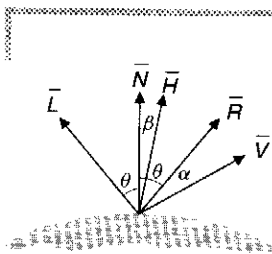
Rys. 14.11. Obliczanie wektora odbicia

$$\overline{R} = 2\overline{N}(\overline{N}\cdot\overline{L}) - \overline{L} \tag{14.16}$$

$$\overline{R}\cdot\overline{V} = (2\overline{N}(\overline{N}\cdot\overline{L}) - \overline{L})\cdot\overline{V} \tag{14.17}$$

Jeżeli źródło światła jest w nieskończoności, to $\overline{N}\cdot\overline{L}$ jest stałe dla danego wielokąta, natomiast $\overline{R}\cdot\overline{V}$ zmienia się wzdłuż wielokąta. Dla powierzchni krzywoliniowych albo dla światła, które nie jest w nieskończoności, zarówno $\overline{N}\cdot\overline{L}$, jak i $\overline{R}\cdot\overline{V}$ zmieniają się wzdłuż powierzchni.

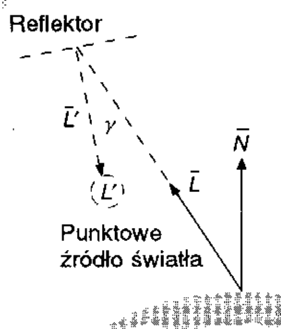
Wektor połowiczny. W alternatywnym sformułowaniu modelu oświetlenia Phonga wykorzystuje się *wektor połowiczny* $\overline{H}$, nazywany tak ze względu na to, że jego kierunek jest w połowie między kierunkami do źródła światła i do obserwatora (rys. 14.12). $\overline{H}$ jest również znane jako kierunek maksymalnego rozświetlenia. Gdyby powierzchnia była tak zorientowana, że jej normalna byłaby w tym samym kierunku co $\overline{H}$, wówczas obserwator widziałby najjaśniejsze odbicie zwierciadlane, ponieważ $\overline{R}$ i $\overline{V}$ wskazywałyby ten sam kierunek. Nowy czynnik odbicia zwierciadlanego może być wyrażony jako $(\overline{N}\cdot\overline{H})^n$, przy czym $\overline{H} = (\overline{L} + \overline{V})/|\overline{L} + \overline{V}|$. Gdy źródło światła i obserwator są w nieskończoności, użycie $\overline{N}\cdot\overline{H}$ daje zyski obliczeniowe, ponieważ $\overline{H}$ jest stałe. Zauważmy, że kąt β miedzy $\overline{N}$ i $\overline{H}$ nie jest równy kątowi α między $\overline{R}$ i $\overline{V}$ i ten sam współczynnik zwierciadlany daje inne wyniki w dwóch rozważanych przypadkach (por. zadanie 14.1). Chociaż korzystanie z czynnika $\cos^n$ umożliwia generowanie lśniących powierzchni, trzeba pamiętać, że wynika on z obserwacji empirycznych, a nie z teoretycznego procesu odbicia zwierciadlanego.



Rys. 14.12. Wektor połowiczny $\overline{H}$ jest pośrodku między kierunkami do źródła światła i do obserwatora

14.1.5. Ulepszenie modelu punktowego źródła światła

Rzeczywiste źródła światła nie promieniują jednakowo we wszystkich kierunkach. Warn [WARN83] zaproponował łatwe w implementacji sterowanie, które może być dodane do dowolnego równania oświetlenia w celu modelowania pewnej kierunkowości świateł wykorzystywanych przez fotografów. W modelu Phonga punktowe źródło światła ma tylko natężenie i położenie. W modelu Warna światło L jest modelowane przez punkt na hipotetycznej powierzchni odbijającej zwierciadlanie



Rys. 14.13. Model oświetlenia Warna. Światło jest modelowane jako odbicie zwierciadlane od jednego punktu oświetlonego przez punktowe źródło światła

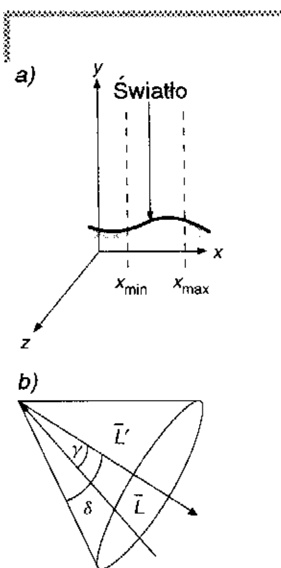
(rys. 14.13). Ta powierzchnia jest oświetlona przez punktowe źródło światła L' w kierunku $\overline{L'}$. Załóżmy, że wektor $\overline{L'}$ jest normalny do hipotetycznej powierzchni odbijającej. Wtedy możemy skorzystać z równania oświetlenia Phonga w celu określenia natężenia L w punkcie obiektu w zależności od kąta γ między $\overline{L}$ i $\overline{L'}$. Jeżeli dalej założymy, że powierzchnia odbijająca odbija tylko światło zwierciadlane i ma współczynnik zwierciadlany równy 1, to natężenie światła w punkcie obiektu

$$I_{L'\lambda}\cos^p\gamma \tag{14.18}$$

przy czym $I_{L'\lambda}$ jest natężeniem światła hipotetycznego punktowego źródła światła, p jest wykładnikiem zwierciadlanym dla powierzchni odbijającej, γ jest kątem między $-\overline{L}$ a hipotetyczną normalną do powierzchni $\overline{L'}$ skierowaną do L'. Równanie (14.18) modeluje symetryczne skierowane źródło światła, którego osią symetrii jest $\overline{L'}$, kierunek, o którym można myśleć, że wskazuje punktowe światło. Korzystając z iloczynu skalarnego równanie (14.18) możemy napisać jako

$$I_{L'\lambda}(-\overline{L}\cdot\overline{L'})^p \tag{14.19}$$

Ponownie ujemny iloczyn skalarny traktujemy jako zero. Wobec tego wzór (14.19) możemy podstawić za natężenie źródła światła $I_{p\lambda}$ w równaniu (14.15) albo w innym równaniu oświetlenia. Im jest większa wartość p, tym więcej światła koncentruje się wzdłuż $\overline{L'}$. Wobec tego duża wartość p może symulować kierunkową plamę światła, a mała wartość p może symulować bardziej rozproszone światło reflektora. Jeżeli p jest równe 0, to światło działa jak jednolicie promieniujące źródło punktowe. Na rysunku 14.15a, b, c pokazano efekty dla różnych wartości p.



Rys. 14.14. Stosowanie klap (a) i stożków (b)

W celu ograniczenia oddziaływania źródła światła do pewnego fragmentu sceny Warn implementował *klapy* i *stożki*. Klapy, modelowane na podobieństwo przysłon w profesjonalnych reflektorach wykorzystywanych w fotografice, ograniczają oddziaływanie światła do określonego przedziału wartości x, y i z we współrzędnych świata. Każde światło ma sześć klap odpowiadających określonym przez użytkownika minimalnym i maksymalnym wartościom każdej współrzędnej. Jeżeli określa się cień punktu, to model oświetlenia oblicza się dla światła tylko wówczas, gdy współrzędne punktu znajdują się w przedziale wyznaczonym przez minimalne i maksymalne współrzędne tych klap, które są włączone. Na przykład jeżeli $\overline{L'}$ jest równoległe do osi y, to klapy x i y mogą ostro ograniczyć wpływ światła, podobnie jak w fotografice przysłony reflektorów w stosunku do światła. Na rysunku 14.14 pokazano wykorzystanie klap x w takiej sytuacji. Klapy z mogą być również użyte w celu ograniczenia światła w sposób, który nie ma fizycznego

odpowiednika i który umożliwia oświetlenie tylko obiektów znajdujących się w określonej odległości od światła. Na rysunku 14.15d sześcian jest ustawiony równolegle do osi układu współrzędnych i dwie pary klap umożliwiają uzyskanie pokazanego efektu.

Warn umożliwił tworzenie ostro zarysowanej plamy świetlnej dzięki użyciu stożka, którego wierzchołek jest w źródle światła i którego oś leży wzdłuż $\overline{L'}$. Jak pokazano na rys. 14.14b, stożek z kątem rozwarcia δ może być użyty do ograniczenia wpływu źródła światła na zasadzie obliczenia modelu oświetlenia tylko wówczas, gdy $\gamma < \delta$ (albo gdy $\cos\gamma > \cos\delta$, ponieważ $\cos\gamma$ już został obliczony). W modelu oświetlenia w standardzie PHIGS PLUS jest włączony człon Warna $\cos^p\gamma$ oraz kąt stożka δ. Na rysunku 14.15e pokazano użycie stożka do ograniczenia światła z rys. 14.15c. Na fotografii 21 pokazano samochód, przy którego renderingu wykorzystano mechanizmy sterowania z modelu Warna.

Rys. 14.15. Sześcian i płaszczyzna oświetlone przy wykorzystaniu mechanizmów sterowania Warna: a) jednolicie promieniujący punkt świetlny (albo $p = 0$); b) $p = 4$; c) $p = 32$; d) klapy; e) stożek z $\delta = 18°$. (Za zgodą Davida Kurlandera z Columbia University.)

14.1.6. Wiele źródeł światła

Jeżeli jest m źródeł światła, to czynniki dla poszczególnych świateł sumują się:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \leqslant i \leqslant m} f_{\text{att}_i} I_{p\lambda i} [k_d O_{d\lambda} (\overline{N} \cdot \overline{L}_i) + k_s O_{s\lambda} (\overline{R}_i \cdot \overline{V})^n] \tag{14.20}$$

Sumowanie stwarza nową możliwość powstania błędu, ponieważ I_λ może teraz przekroczyć maksymalną wyświetlaną wartość piksela. (Chociaż może się to również zdarzyć dla pojedynczego światła, można tego łatwo uniknąć wybierając odpowiednio f_{att} i materiał.) W celu uniknięcia nadmiaru można korzystać z kilku rozwiązań. Najprostsze polega na obcinaniu każdej wartości I_λ indywidualnie do jej maksymalnej wartości. W innym rozwiązaniu bierze się pod uwagę wszystkie wartości I_λ piksela. Jeżeli przynajmniej jedna z nich jest za duża, to każda jest

dzielona przez największą po to, żeby zachować odcień barwy i nasycenie kosztem jasności. Jeżeli wszystkie wartości piksela mogą być obliczone przed wyświetleniem, to można zastosować przekształcenia typowe dla przetwarzania obrazów do całego obrazu po to, żeby sprowadzić wartości do pożądanego zakresu. Hall [HALL89] omawia kompromisy związane z tymi i innymi metodami.

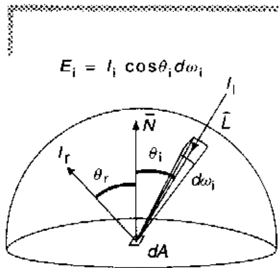
14.1.7. Modele oświetlenia mające podłoże fizyczne

Modele oświetlenia omawiane w poprzednich punktach wynikały głównie ze zdrowego rozsądku i praktycznego podejścia do grafiki. Chociaż użyte równania aproksymują niektóre ze sposobów oddziaływania światła z obiektami, nie mają podłoża fizycznego. W tym punkcie zajmiemy się modelami oświetlenia wynikającymi z fizyki; skorzystamy przy tym z pracy Cooka i Torrance'a [COOK82].

Dotychczas korzystaliśmy z określenia natężenia bez podania jego definicji, odwołując się nieformalnie do natężenia źródła światła, punktu powierzchni albo piksela. Teraz jest już czas na sformalizowanie naszych określeń przez wprowadzenie terminologii radiometrycznej stosowanej w badaniach promieniowania termicznego, które są podstawą naszego rozumienia sposobu oddziaływania światła z obiektami [NICO77; SPARR;78; SIEG81; IES87]. Zaczynamy od pojęcia *strumienia*, który jest szybkością, z jaką energia świetlna jest emitowana, i jest mierzony w watach (W). Aby mówić o wielkości strumienia emitowanego albo odbieranego z określonego kierunku, jest nam potrzebna koncepcja *kąta bryłowego* – kąta przy wierzchołku stożka. Kąt bryłowy jest mierzony w zależności od powierzchni na kuli przecinanej przez stożek, którego wierzchołek leży w środku kuli. *Steradian* (sr) jest to kąt bryłowy takiego stożka, który przecina powierzchnię równą kwadratowi promienia kuli r. Jeżeli punkt jest na powierzchni, interesuje nas półkula nad tą powierzchnią. Ponieważ powierzchnia kuli wynosi $4\pi r^2$, z półkulą jest zatem związany kąt bryłowy $4\pi r^2/2r^2 = 2\pi$ sr. Wyobraźmy sobie rzut kształtu obiektu na półkulę umieszczoną centrycznie wokół punktu powierzchni, który spełnia funkcję środka rzutowania. Kąt bryłowy ω związany z obiektem jest określony przez powierzchnię na półkuli zajętą przez rzut, podzieloną przez kwadrat promienia półkuli (dzielenie eliminuje zależność od wielkości półkuli).

Natężenie promieniowania jest określone przez strumień wypromieniowany przez jednostkowy kąt bryłowy w określonym kierunku i jest mierzone w W/sr. Mówiąc dotychczas o natężeniu w odniesieniu do źródła punktowego mieliśmy na myśli jego natężenie promieniowania.

Luminancja energetyczna jest natężeniem promieniowania na jednostkę obszaru powierzchni w rzucie perspektywicznym i jest mierzona w $W/(sr \cdot m^2)$. *Obszar powierzchni w rzucie perspektywicznym*, znany również jako *rzut obszaru powierzchni*, odnosi się do rzutu powierzchni na płaszczyznę prostopadłą do kierunku promieniowania. Obszar powierzchni w rzucie perspektywicznym znajduje się mnożąc obszar powierzchni przez $\cos\theta_r$, przy czym θ_r jest kątem promieniowanego światła względem normalnej do powierzchni. Mały kąt bryłowy $d\omega$ może być aproksymowany jako obszar powierzchni w rzucie perspektywicznym podzielony przez kwadrat odległości od obiektu do punktu, w którym kąt bryłowy jest obliczany. Gdy używaliśmy określenia natężenie w odniesieniu do powierzchni, mieliśmy na myśli luminancję energetyczną. Wreszcie *natężenie napromieniowania*, znane również jako *gęstość strumienia*, jest to strumień padający na jednostkę powierzchni (nie w rzucie perspektywicznym) i jest mierzone w W/m^2.



Rys. 14.16. Promieniowanie padające i odbite

W grafice jesteśmy zainteresowani zależnością światła padającego na powierzchnię od światła odbitego od powierzchni i emitowanego przez powierzchnię. Popatrzmy na rys. 14.16. Natężenie napromieniowania światła padającego

$$E_i = I_i(\overline{N} \cdot \overline{L})d\omega_i$$

przy czym I_i jest luminancją energetyczną światła padającego, $\overline{N} \cdot \overline{L} = \cos\theta_i$. Ponieważ natężenie napromieniowania jest wyrażone na jednostkę powierzchni, a luminancja energetyczna jest wyrażona na jednostkę powierzchni w rzucie perspektywicznym, mnożenie przez $\overline{N} \cdot \overline{L}$ zamienia zdolność promieniowania na równoważną wielkość przypadającą na jednostkę powierzchni nie w rzucie perspektywicznym.

Nie wystarczy brać pod uwagę tylko I_i (padające promieniowanie) przy określaniu I_r (odbite promieniowanie); zamiast tego trzeba brać pod uwagę E_i (padające natężenie napromieniowania). Na przykład strumień padający ze źródła o takiej samej luminancji jak drugie źródło, ale o większym kącie bryłowym ω_i, daje proporcjonalnie większe natężenie promieniowania E_i i powoduje, że powierzchnia wydaje się proporcjonalnie jaśniejsza. Stosunek luminancji obserwowanej z danego kierunku do natężenia napromieniowania padającego z innego kierunku jest znany jako *dwukierunkowa zdolność odbijania*, która dla powierzchni nielambertowskich jest funkcją kierunków padania i odbicia

$$\rho = \frac{I_r}{E_i}$$

Jak widzieliśmy, w grafice komputerowej jest wygodnie traktować dwukierunkową zdolność odbijania jako złożenie odbijania rozproszonego i zwierciadlanego. Stąd

$$\rho = k_d \rho_d + k_s \rho_s$$

przy czym ρ_d i ρ_s są odpowiednio dwukierunkową zdolnością odbijania rozproszonego i zwierciadlanego, a k_d i k_s są odpowiednio współczynnikami odbicia rozproszonego i zwierciadlanego wprowadzonymi wcześniej w tym rozdziale.

Model powierzchni Torrance'a-Sparrowa [TORR66; TORR67], opracowany przez fizyków, jest fizycznym modelem powierzchni odbijającej. Blinn jako pierwszy zaadaptował model Torrance'a-Sparrowa dla potrzeb grafiki komputerowej, podając matematyczne szczegóły i porównując go z modelem Phonga [BLIN77a]; Cook i Torrance [COOK82] jako pierwsi zaaproksymowali rozkład widmowy światła odbitego w implementacji modelu.

W modelu Torrance'a-Sparrowa zakłada się, że powierzchnia jest izotropowym zbiorem płaskich mikroskopijnych ścianek, z których każda jest idealnym reflektorem. Geometria i rozkład tych mikrościanek oraz kierunek światła (zakłada się, że światło pochodzi z nieskończenie odległego źródła i że wobec tego wszystkie promienie są równoległe) określają natężenie i kierunek odbicia zwierciadlanego jako funkcję I_p (natężenie światła punktowego źródła światła) $\overline{N}$, $\overline{L}$ i $\overline{V}$. Pomiary eksperymentalne pokazują bardzo dobrą zgodność między faktycznym odbiciem a odbiciem wynikającym z tego modelu [TORR67].

Dla zwierciadlanej składowej dwukierunkowej zdolności odbijania Cook i Torrance korzystają ze wzoru

$$\rho_s = \frac{F_\lambda}{\pi} \frac{DG}{(\overline{N}\cdot\overline{V})(\overline{N}\cdot\overline{L})} \tag{14.21}$$

w którym D jest funkcją rozkładu orientacji mikrościanek, G jest *geometrycznym współczynnikiem tłumienia*, który reprezentuje efekty wzajemnego maskowania i zacieniania mikrościanek, F_λ jest czynnikiem Fresnela obliczonym z równania Fresnela, który dla odbicia zwierciadlanego wiąże światło padające ze światłem odbitym dla gładkiej powierzchni każdej mikrościanki. π w mianowniku ma uwzględniać chropowatość powierzchni (sposób wyprowadzenia równania można znaleźć w pracy [JOY88, str. 227-230]). Czynnik $\overline{N}\cdot\overline{L}$ zapewnia proporcjonalność do powierzchni (a więc do liczby mikrościanek), którą obserwator widzi w jednostce obszaru powierzchni w rzucie perspektywicznym. Szczegóły na temat składowych wyrazów równania (14.21) można znaleźć w p. 16.7 książki [FOLE90]; tu przedstawiamy tylko wyniki.

Na fotografii 40 pokazano dwie wazy miedziane odtworzone za pomocą modelu Cooka-Torrance'a, przy wyświetleniu których wykorzystano dwukierunkowy współczynnik odbicia miedzi dla czynnika rozproszonego. Dla pierwszej wazy czynnik zwierciadlany jest modelowany za pomocą współczynnika odbicia od zwierciadła winylowego; otrzymane wyniki są podobne do wyników otrzymanych za pomocą oryginalnego modelu oświetlenia Phonga według równania (14.14). W przypadku drugiej wazy jest modelowany czynnik zwierciadlany z odbiciem od zwierciadła miedzianego. Zauważmy, jak uwzględnienie zależności barwy rozświetlenia zwierciadlanego od kąta padania i powierzchni materiału tworzy bardziej przekonywający obraz powierzchni metalicznej.

Ogólnie, składowe otoczenia, odbicia rozproszonego i zwierciadlanego tworzą barwę materiału zarówno dla dielektryków, jak i przewodników. Obiekty, np. wykonane z plastyku, na ogół mają składową odbicia rozproszonego i zwierciadlanego o różnych barwach. Dla metali na ogół występuje niewielkie odbicie rozproszone; mają one barwę składowej zwierciadlanej, zmieniającą się od tej dla metalu do tej dla źródła światła, gdy θ_i zbliża się do 90°. Ta obserwacja sugeruje zgrubną aproksymację dla modelu Cooka-Torrance'a wykorzystującego równanie (14.15) z $O_{s\lambda}$ wybranym na zasadzie interpolacji z tablicy pośredniej bazującej na θ_i.

Opracowano kilka ulepszeń i uogólnień modelu oświetlenia Cooka-Torrance'a; por. na przykład prace [KAJI85; CABR87; WOLF90]. W ostatniej pracy He i innych [HE92] pokazano szybką i dokładną metodę stosowania takich modeli fizycznych. Warto zauważyć, że praca [HE92] jest publikacją multimedialną, która umożliwia czytelnikowi badanie interakcyjne wpływu wielu czynników w równaniu (14.21).

14.2. Modele cieniowania wielokątów

Powinno być jasne, że możemy pocieniować dowolną powierzchnię obliczając normalną do powierzchni w każdym widocznym punkcie i stosując odpowiedni model oświetlenia w tym punkcie. Niestety, taki bezpośredni model cieniowania jest drogi. W tym punkcie opiszemy bardziej efektywne modele oświetlenia dla powierzchni zdefiniowanych przez wielokąty i siatki wielokątów.

14.2.1. Cieniowanie stałą wartością

Najprostszy model cieniowania wielokąta polega na cieniowaniu *stałą wartością*, określanym również jako *cieniowanie ścian* albo *cieniowanie płaskie*. W tym podejściu model oświetlenia jest wykorzystywany tylko

raz w celu określenia jednej wartości natężenia, która jest później używana do cieniowania całego wielokąta. W istocie próbkujemy wartość równania oświetlenia raz dla całego wielokąta i utrzymujemy tę wartość w całym wielokącie w celu rekonstrukcji barwy wielokąta. Takie podejście jest dobre, jeżeli jest spełnionych kilka założeń:

1. Źródło światła jest w nieskończoności, a więc $\overline{N} \cdot \overline{L}$ jest stałe na całej powierzchni wielokąta.
2. Obserwator jest w nieskończoności, a więc $\overline{N} \cdot \overline{V}$ jest stałe na całej powierzchni.
3. Wielokąt reprezentuje faktyczną powierzchnię modelowaną i nie jest aproksymacją powierzchni krzywoliniowej.

Jeżeli jest wykorzystywany algorytm powierzchni widocznej, który generuje listę wielokątów, taki jak algorytm z listą priorytetową, całe cieniowanie może korzystać z powszechnie dostępnego prymitywu jednobarwnego wielokąta 2D.

Jeżeli któreś z pierwszych dwóch założeń nie jest spełnione, to, gdybyśmy chcieli zastosować stałe cieniowanie, byłaby potrzebna metoda określania jednej wartości dla każdego $\overline{L}$ i $\overline{V}$. Na przykład wartości mogłyby być obliczane dla środka wielokąta albo dla pierwszego wierzchołka wielokąta. Oczywiście przy stałym cieniowaniu nie ma zmian cieniowania wzdłuż wielokąta, które powinny się w takiej sytuacji pojawić.

14.2.2. Cieniowanie z interpolacją

Jako alternatywę obliczania równania oświetlenia w każdym punkcie wielokąta Wylie, Romney, Evans i Erdahl [WYLI67] jako pierwsi zastosowali do cieniowania interpolację, w której informacja o cieniowaniu jest liniowo interpolowana w trójkącie na podstawie wartości określonych dla jego wierzchołków. Gouraud [GOUR71] uogólnił tę metodę na dowolne wielokąty. Ta metoda jest szczególnie łatwa dla algorytmu przeglądania wierszami, który już i tak interpoluje wartość z wzdłuż odcinka przeglądanego na podstawie wartości z obliczonych dla końców tego odcinka.

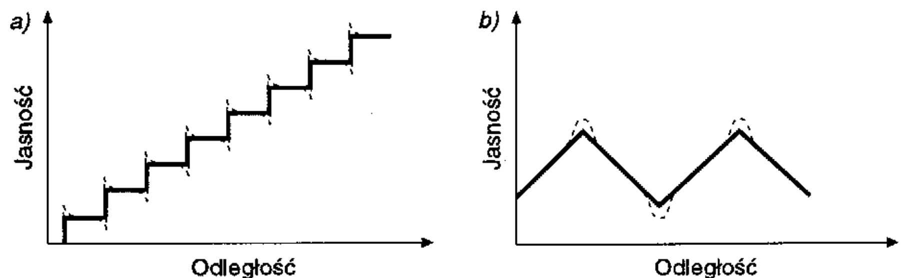
W celu zwiększenia efektywności można korzystać z równania różnicowego, podobnego do równania z p. 13.2 do określania wartości z w każdym pikselu. Chociaż interpolacja z jest fizycznie poprawna (przy założeniu, że wielokąt jest płaski), zauważmy, że cieniowanie z interpolacją nie, ponieważ aproksymuje ono tylko obliczanie modelu oświetlenia w każdym punkcie wielokąta.

Nasze ostatnie założenie, że wielokąt dokładnie reprezentuje powierzchnię modelowaną, najczęściej jest niepoprawne i ma znacznie istotniejszy wpływ na wynikowy obraz niż błąd w jednym z pozostałych

dwóch założeń. Wiele obiektów jest krzywoliniowych a nie wielościanowych, jednak reprezentowanie ich jako siatki wielokątów umożliwia stosowanie efektywnych algorytmów wyznaczania powierzchni widocznych dla wielokątów. Dalej omawiamy, jak wykonać rendering siatki *wielokątowej, żeby wyglądała możliwie podobnie do powierzchni krzywoliniowej.*

14.2.3. Cieniowanie siatki wielokątowej

Załóżmy, że chcemy aproksymować powierzchnię krzywoliniową za pomocą siatki wielokątowej. Jeżeli każda ściana wielokątowa w siatce jest cieniowana niezależnie, to jest łatwa do odróżnienia od sąsiadów o innej orientacji i otrzymuje się wygląd jak na fot. 28. Tak jest, jeżeli wielokąty są wyświetlane ze stałą barwą, z cieniowaniem z interpolacją, a nawet z oświetleniem obliczanym dla każdego piksela; wynika to stąd, że dwa sąsiednie wielokąty o różnej orientacji mają różne jasności wzdłuż swoich krawędzi. Proste rozwiązanie polegające na użyciu dokładniejszej siatki okazuje się zaskakująco nieefektywne, ponieważ postrzegana różnica w cieniowaniu między sąsiednimi ściankami jest uwydatniana przez efekt pasm Macha (odkryty przez Macha w 1865 r. i opisany szczegółowo w pracy [RATL72]), który uwypukla zmianę jasności na każdej krawędzi, gdzie jest nieciągłość amplitudy albo pochodnej. Na krawędzi między dwiema ścianami ciemna ściana wygląda ciemniej, a jasna jaśniej. Na rysunku 14.17 pokazano dla dwóch przypadków faktyczne i postrzegane zmiany jasności wzdłuż powierzchni.



Rys. 14.17. Dwa przykłady rzeczywistej i postrzeganej jasności dla efektu pasm Macha. Linie przerywane odpowiadają jasności postrzeganej; linie ciągłe reprezentują rzeczywistą jasność

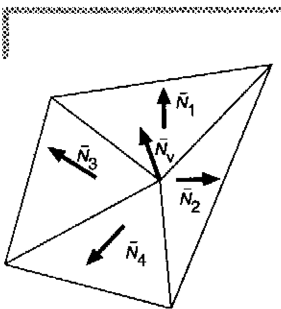
Efekt pasm Macha jest powodowany przez *poziome hamowanie* receptorów w oku. Im więcej światła otrzymuje receptor, tym silniej oddziałuje hamująco na odpowiedź receptora sąsiedniego. Odpowiedź receptora na światło jest hamowana przez sąsiednie receptory w zależności

odwrotnie proporcjonalnej do odległości od sąsiedniego receptora. Receptory będące bezpośrednio po jaśniejszej stronie dają silniejszą odpowiedź niż te, które są dalej od krawędzi, ponieważ otrzymują one mniejszy sygnał hamowania od sąsiadów po ciemniejszej stronie. Podobnie receptory znajdujące się bezpośrednio po ciemnej stronie zmiany jasności dają słabszą odpowiedź niż znajdujące się dalej w ciemnym obszarze, ponieważ otrzymują silniejsze sygnały hamowania od swoich sąsiadów po jaśniejszej stronie. Efekt pasm Macha widać wyraźnie na fot. 28, zwłaszcza między sąsiednimi wielokątami o bliskich barwach.

W opisanych modelach cieniowania wielokąta barwa każdego wielokąta jest wyznaczana indywidualnie. Dwa podstawowe modele cieniowania dla siatek wielokątowych wykorzystują informacje pochodzącą z sąsiednich wielokątów do symulowania gładkiej powierzchni. Są one znane jako cieniowanie Gourauda i cieniowanie Phonga (są wymienione w kolejności wynikającej ze wzrostu złożoności i efektu realizmu), od nazwisk twórców tych modeli. Współczesne stacje graficzne 3D na ogół mają wspomaganie sprzętowe albo sprzętowo-programowe dla jednej z tych metod.

14.2.4. Cieniowanie Gourauda

Cieniowanie Gourauda [GOUR71], określane również jako *cieniowanie na zasadzie interpolowania jasności* albo *cieniowanie na zasadzie interpolowania barwy*, eliminuje nieciągłości jasności. Cieniowanie Gourauda wykorzystano przy tworzeniu obrazu na fot. 29. Chociaż większość efektu pasm Macha z fot. 28 nie jest już widoczna na fot. 29, jasne pręgi na takich obiektach jak torus czy stożek są pasmami Macha powodowanymi przez szybkie chociaż nie nieciągłe zmiany nachylenia krzywej jasności; cieniowanie Gourauda nie eliminuje całkowicie takich zmian jasności.

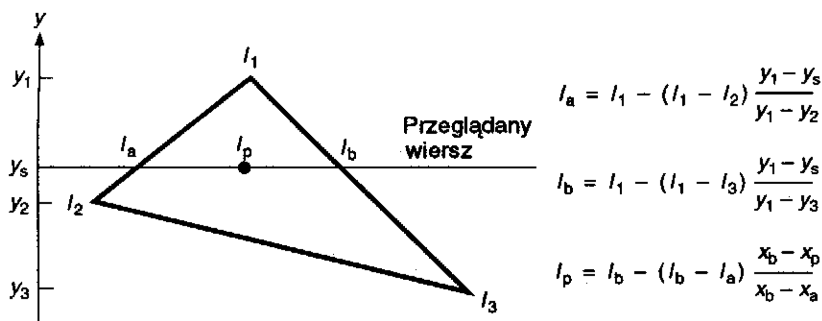


Rys. 14.18. Znormalizowane normalne do powierzchni wielokątów mogą być uśrednione w celu obliczenia normalnych dla wierzchołków. Uśredniona normalna $\bar{N}_v$ jest równa $\sum_{1 \leq i \leq n} \bar{N}_i \Big/ \left| \sum_{1 \leq i \leq n} \bar{N}_i \right|$

Cieniowanie Gourauda rozszerza koncepcję cieniowania z interpolacją stosowanego do poszczególnych wielokątów dzięki interpolowaniu wartości oświetlenia z wierzchołków wielokąta, wyznaczonego przy uwzględnieniu aproksymowanej powierzchni. Proces cieniowania Gourauda wymaga, żeby była znana normalna dla każdego wierzchołka siatki wielokątowej. Gouraud mógł obliczać te normalne dla wierzchołków bezpośrednio z analitycznego opisu powierzchni. Alternatywnie, jeżeli normalne dla wierzchołków nie są zapisane z siatką i nie mogą być określone bezpośrednio dla bieżącej powierzchni, to Gouraud sugerował, żeby je aproksymować na zasadzie uśredniania normalnych do powierzchni wszystkich ścian wielokątowych, dla których dany wierzchołek jest wspólny (rys. 14.18). Jeżeli krawędź ma być widoczna (tak

jak na połączeniu między skrzydłem samolotu a kadłubem), to znajdujemy dwie normalne dla wierzchołków, po jednej dla każdej strony krawędzi, na zasadzie uśrednienia normalnych do wielokątów z każdej strony krawędzi oddzielnie.

Następny krok w cieniowaniu Gourauda polega na znalezieniu *jasności w wierzchołkach* przy wykorzystaniu normalnych w wierzchołkach za pomocą wybranego modelu oświetlenia. Wreszcie, każdy wielokąt jest cieniowany na zasadzie interpolacji liniowej między wierzchołkami wzdłuż każdej krawędzi, a potem między krawędziami wzdłuż każdego przeglądanego wiersza (rys. 14.19) w taki sam sposób jak przy interpolowaniu wartości z w p. 13.2. Określenie *cieniowanie Gourauda* jest często uogólniane na cieniowanie metodą interpolacji jasności nawet jednego wielokąta albo na interpolację dowolnych barw związanych z wierzchołkami wielokąta.

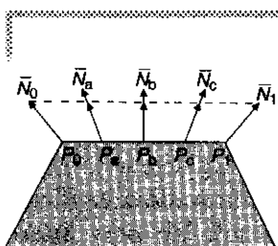


Rys. 14.19. Interpolacja jasności wzdłuż krawędzi wielokąta i przeglądanych wierszy

Interpolacja wzdłuż krawędzi może być łatwo zintegrowana z algorytmem wyznaczania powierzchni widocznych metodą przeglądania wierszy opisanym w p. 13.3. Dla każdej krawędzi zapamiętujemy dla każdej składowej barwy wartość początkową i zmianę jasności dla każdej zmiany jednostkowej dla współrzędnej y. Widzialny segment w wierszu jest wypełniany na zasadzie interpolowania wartości jasności dwóch krawędzi ograniczających ten segment. Tak jak we wszystkich algorytmach interpolacji liniowej, w celu zwiększenia efektywności można zastosować równanie różnicowe.

14.2.5. Cieniowanie Phonga

Cieniowanie Phonga [BUIT75], znane również jako *cieniowanie z interpolacją wektora normalnego*, interpoluje wektor normalny $\overline{N}$ do powierzchni zamiast jasności. Interpolacja ma miejsce wzdłuż segmentu



Rys. 14.20. Interpolacja wektora normalnego (Za pracą [BUIT75].)

w przeglądanym wierszu, między normalnymi dla początku i dla końca segmentu. Te normalne same są interpolowane wzdłuż krawędzi wielokąta na podstawie normalnych w wierzchołkach, które są obliczane, jeśli to jest konieczne, tak jak w przypadku cieniowania Gouraude. Interpolacja wzdłuż krawędzi znowu może być wykonana za pomocą obliczeń przyrostowych, przy czym wszystkie trzy składowe wektora normalnego są inkrementowane przy przejściu od przeglądanego wiersza do następnego przeglądanego wiersza. Dla każdego piksela wzdłuż przeglądanego wiersza interpolowana normalna jest normalizowana i jest z powrotem odwzorowywana do układu WC lub izometrycznego do niego i wykonuje się nowe obliczenie jasności za pomocą jakiegoś modelu oświetlenia. Na rysunku 14.20 pokazano dwie normalne dla krawędzi i normalne interpolowane na ich podstawie, przed i po normalizacji.

Fotografie 30 i 31 otrzymano przy zastosowaniu odpowiednio cieniowania Gourauda i cieniowania Phonga oraz równania oświetlenia z członem dla odbicia zwierciadlanego. Dla takich modeli oświetlenia cieniowanie Phonga daje istotne polepszenie w stosunku do cieniowania Gourauda, ponieważ rozjaśnienia są reprodukowane z większą wiernością, jak to widać na rys. 14.21. Zastanówmy się, co się stanie, jeżeli n w członie $\cos^n\alpha$ cieniowania Phonga oświetlenia jest duże i dla jednego wierzchołka jest bardzo mały kąt α, dla każdego natomiast z sąsiednich wierzchołków kąt α jest duży. Jasność związana z wierzchołkiem, dla którego kąt α jest mały, będzie odpowiednia dla rozświetlenia, a dla innych jasności pojawią się wartości, które nie odpowiadają rozświetleniom. Jeżeli użyje się cieniowania Gourauda, to jasność wzdłuż wielokąta jest liniowo interpolowana między jasnością rozświetlenia a mniejszymi jasnościami sąsiednich wierzchołków, rozprzestrzeniając rozświetlenie po powierzchni wielokąta (rys. 14.21a). Porównajmy to z ostrym spadkiem jasności rozświetlenia występującym wówczas, gdy wykorzystuje się normalne interpolowane liniowo do obliczenia czynnika $\cos^n\alpha$ w każdym pikselu

Rys. 14.21. Model oświetlenia dla odbicia zwierciadlanego używany w cieniowaniu Gourauda i Phonga. Rozświetlony jest lewy wierzchołek: a) cieniowanie Gourauda; b) cieniowanie Phonga. Rozświetlenie we wnętrzu wielokąta: c) cieniowanie Gourauda; d) cieniowanie Phonga. (Za zgodą Davida Kurlandera z Columbia University.)

(rys. 14.21b). Jeżeli rozświetlenie nie trafia na wierzchołek, to cieniowanie Gouraude może je całkowicie pominąć (rys. 14.21c), ponieważ żaden punkt wewnętrzny nie może być jaśniejszy od najjaśniejszego wierzchołka, od którego zaczyna się interpolacja. Przy cieniowaniu Phonga jest możliwe usytuowanie rozświetlenia wewnątrz wielokąta (rys. 14.21d). Porównajmy rozświetlenia na piłce na fot. 30 i 31.

Nawet dla modelu oświetlenia, w którym nie bierze się pod uwagę współczynnika odbicia zwierciadlanego, wyniki interpolacji wektora normalnego są na ogół lepsze od interpolacji jasności, ponieważ w każdym punkcie jest używana aproksymacja normalnej. To w większości przypadków redukuje problemy pasm Macha, ale znacznie zwiększa koszt cieniowania w bezpośredniej implementacji, ponieważ interpolowana normalna musi być normalizowana za każdym razem, gdy jest używana w modelu oświetlenia. Duff [DUFF79] opracował kombinację równań różnicowych i tablicę pośrednią dla przyspieszenia obliczeń. Bishop i Weimer [BISH86] pokazują doskonałą aproksymację cieniowania Phonga przy wykorzystaniu rozwinięcia w szereg Taylora, które oferuje większy wzrost szybkości cieniowania.

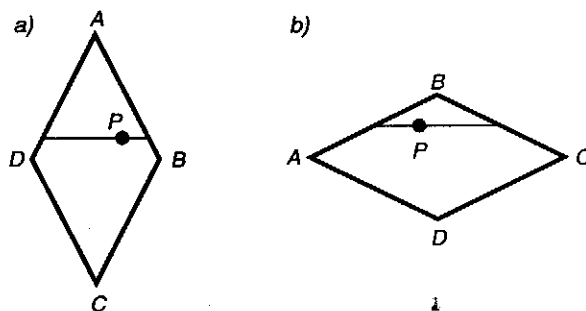
14.2.6. Problemy z cieniowaniem z interpolacją

Jest wiele problemów wspólnych dla tych wszystkich modeli interpolowanego cieniowania; kilka z nich teraz omówimy.

Szkielet wielokątowy. Niezależnie od tego, jak dobrą aproksymację powierzchni krzywoliniowej daje interpolowany model cieniowania, widoczny jest szkielet wielokątowy krawędzi siatki. Możemy tę sytuację poprawić dzieląc powierzchnię na większą liczbę mniejszych wielokątów, ale wiąże się to z odpowiednim wzrostem kosztów.

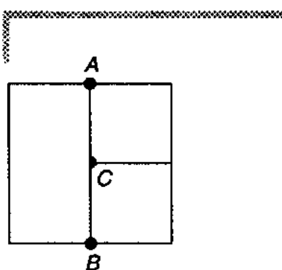
Zakłócenia perspektywiczne. Wprowadzane są anomalie, ponieważ interpolacja jest wykonywana po przekształceniu perspektywicznym w układzie współrzędnych ekranu 3D, a nie w układzie WC. Na przykład interpolacja liniowa powoduje, że parametr cieniowania na rys. 14.19 jest zwiększany o stałą wartość przy przejściu od linii do linii wzdłuż każdej krawędzi. Zastanówmy się, co się stanie, jeżeli wierzchołek 1 jest bardziej odległy niż wierzchołek 2. Skrót perspektywiczny oznacza, że różnica między kolejnymi liniami nie przekształconej wartości z wzdłuż krawędzi wzrasta w kierunku dalszej współrzędnej. Dlatego jeżeli $y_s = (y_1 + y_2)/2$, to $I_s = (I_1 + I_2)/2$, ale z_s nie będzie równe $(z_1 + z_2)/2$. Ten problem można zredukować korzystając z większej liczby mniejszych wielokątów. Zmniejszenie wielkości wielokątów zwiększa liczbę punktów, w których jest próbkowana informacja, która ma być interpolowana, i stąd wzrasta dokładność cieniowania.

Zależność od orientacji. Wyniki uzyskiwane za pomocą modeli cieniowania z interpolacją zależą od orientacji rzutowanych wielokątów. Ponieważ wartości są interpolowane między wierzchołkami, a następnie wzdłuż poziomych linii przeglądania, wyniki mogą się różnić przy obracaniu wielokąta (rys. 14.22). Ten efekt jest szczególnie oczywisty, gdy orientacja zmienia się wolno między kolejnymi ramkami animacji. Podobny problem może również wystąpić przy określaniu powierzchni widocznych, gdy wartość z w każdym punkcie jest interpolowana na podstawie wartości z przypisanych do każdego wierzchołka. Oba problemy można rozwiązać dekomponując wielokąty na trójkąty (por. zadanie 14.2). Duff [DUFF79] sugeruje metody interpolacji niezależne od obrotów, które rozwiązują ten problem bez potrzeby dekompozycji; są to jednak kosztowne metody.



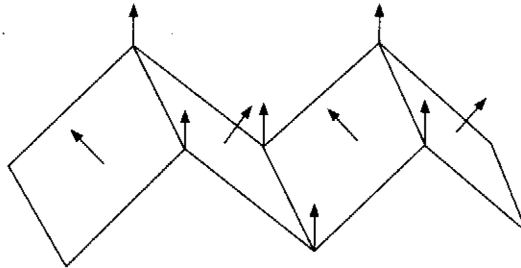
Rys. 14.22. Wartości interpolowane otrzymane dla punktu P dla tego samego wielokąta przy różnych orientacjach są różne dla przypadków a) i b). W przypadku a) punkt P jest interpolowany na podstawie A, B i D, a w przypadku b) na podstawie A, B i C

Wspólne wierzchołki. Nieciągłości cieniowania mogą wystąpić wówczas, gdy dwa sąsiednie wielokąty nie mają wspólnego wierzchołka, który leży na ich wspólnej krawędzi. Weźmy pod uwagę trzy wielokąty z rys. 14.23, na którym wierzchołek C jest wspólny dla dwóch wielokątów z prawej strony, nie jest natomiast wspólny dla dużego wielokąta z lewej strony. Informacja o cieniowaniu określona bezpośrednio w C dla wielokątów z prawej strony na ogół nie będzie taka sama jak informacja interpolowana w tym punkcie na podstawie wartości A i B dla wielokąta z lewej strony. W wyniku powstanie nieciągłość w cieniowaniu. Nieciągłość tę można wyeliminować umieszczając w wielokącie z lewej strony dodatkowy wierzchołek, który niesie wspólną informację o cieniowaniu. W celu wyeliminowania tego problemu możemy wstępnie przetworzyć wielokątową bazę danych; jeżeli wielokąty będą dzielone na bieżąco (to znaczy z wykorzystaniem algorytmu wyznaczania powierzchni widocznych z drzewem BSP), to można wprowadzić nowy wierzchołek do współdzielonej krawędzi.



Rys. 14.23. Wierzchołek C jest wspólny dla dwóch wielokątów z prawej strony, nie jest natomiast wspólny dla dużego wielokąta z lewej strony

Niereprezentatywne normalne związane z wierzchołkiem. Obliczone normalne związane z wierzchołkami mogą nie reprezentować dokładnie geometrii powierzchni. Na przykład, jeżeli obliczymy normalne związane z wierzchołkami na zasadzie uśredniania normalnych do powierzchni mających wspólny wierzchołek, to wszystkie normalne związane z wierzchołkiem z rys. 14.24 będą do siebie równoległe, co da w efekcie niewielką zmianę albo w ogóle brak zmiany w cieniowaniu dla odległego źródła światła. Dalsza dekompozycja wielokątów przed obliczeniem normalnej związanej z wierzchołkiem rozwiąże ten problem.



Rys. 14.24. Problemy z obliczaniem normalnych związanych z wierzchołkami. Wszystkie normalne związane z wierzchołkami są do siebie równoległe

Chociaż te problemy stymulowały wiele prac nad algorytmami bezpośredniego renderingu dla powierzchni krzywoliniowych, wielokąty są na tyle szybsze (i łatwiejsze) do przetwarzania, aby wciąż stanowić podstawę większości systemów renderingu.

14.3. Szczegóły powierzchni

Stosując każdy z opisanych dotychczas modeli cieniowania do płaskich albo bikubicznych powierzchni uzyskamy gładkie, jednolite powierzchnie – istotnie odróżniające się od większości powierzchni, które obserwujemy i do których jesteśmy przyzwyczajeni. Dalej omawiamy różne metody symulowania brakujących szczegółów powierzchni.

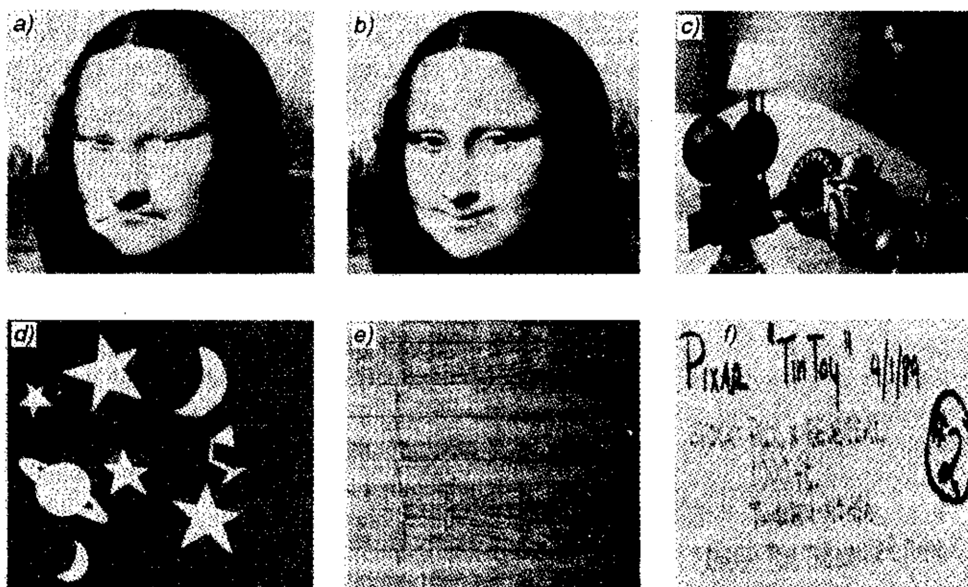
14.3.1. Detale wielokątowe

Najprostszy sposób dodania wielu szczegółów polega na użyciu *detali wielokątowych* w celu pokazania różnych cech (takich jak drzwi, okna i napisy) na tle wielokąta (takiego jak na przykład ściana budynku). Każdy wielokąt detalu leży w tej samej płaszczyźnie co wielokąt bazowy i ma przypisane wskaźniki, które sygnalizują, że nie ma potrzeby poró-

wnywania go z innymi wielokątami w czasie określania powierzchni widocznych. W czasie cieniowania wielokąta jego detale i właściwości materiałów mają pierwszeństwo w stosunku do pokrywanych części wielokąta bazowego.

14.3.2. Odwzorowanie tekstury

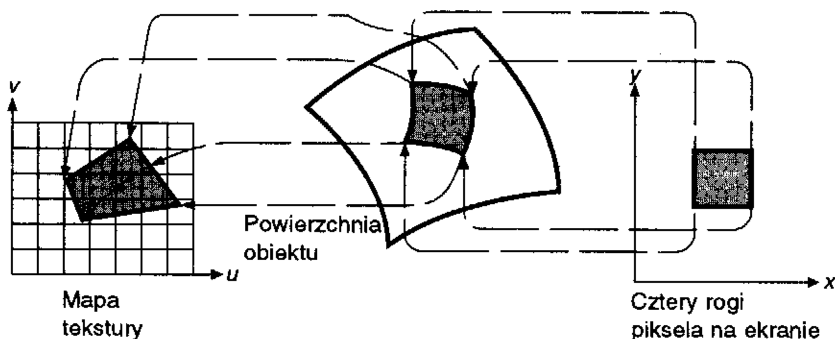
W miarę jak szczegóły stają się coraz drobniejsze i zawiłe, bezpośrednie modelowanie ich za pomocą wielokątów i innych prymitywów geometrycznych staje się coraz mniej praktyczne. Alternatywa polega na odwzorowaniu na powierzchnię obrazu wprowadzonego w postaci cyfrowej albo syntetycznego – jest to metoda zaproponowana przez Catmulla [CATM74] i ulepszona przez Blinna i Newella [BLIN76]. To podejście jest znane jako *odwzorowanie tekstury* albo *odwzorowanie wzoru*; obraz jest określany jako mapa tekstury, a jego poszczególne elementy są często określane jako *teksele*. Prostokątna mapa tekstury jest określona w jej własnym układzie współrzędnych tekstury (u, v). Tekstura może być również zdefiniowana jako procedura. Na fotografii 34 pokazano kilka przykładów odwzorowania tekstury z wykorzystaniem tekstur pokazanych na rys. 14.25. Przy renderingu każdego piksela wybrane



Rys. 14.25. Tekstury wykorzystane do utworzenia fot. 34: a) niezadowolona Mona Liza; b) uśmiechnięta Mona Liza; c) obraz; d) czapka czarnoksiężnika; e) podłoga; f) etykieta filmu. (Za zgodą firmy Pixar © 1990. Rendering został wykonany przez Thomasa Williamsa i H. B. Siegela za pomocą programu PhotoRealistic RenderMan™ firmy Pixar)

teksele są używane albo do zastąpienia, albo do skalowania jednej lub więcej właściwości materiałów powierzchni, takich jak składowe barwy odbicia rozproszonego. Często jeden piksel jest pokrywany przez kilka tekseli. W celu uniknięcia zakłóceń musimy uwzględnić wszystkie istotne teksele.

Jak pokazano na rys. 14.26, odwzorowanie tekstury może być zrealizowane w dwóch krokach. Proste podejście zaczyna się od odwzorowania czterech rogów piksela na powierzchnię. Dla płata bikubicznego takie odwzorowanie naturalnie definiuje zbiór punktów w układzie współrzędnych (s, t) powierzchni. Następnie punkty rogów piksela w układzie współrzędnych (s, t) powierzchni są odwzorowywane do układu współrzędnych (u, v) tekstury. Cztery punkty (u, v) w mapie tekstury definiują czworokąt, który aproksymuje bardziej złożony kształt, na który piksel może być odwzorowywany ze względu na krzywiznę powierzchni. Obliczamy wartość piksela sumując wszystkie teksele, które leżą wewnątrz czworokąta, przypisując każdemu wagę określoną przez część piksela leżącą w czworokącie. Jeżeli przekształcony punkt w układzie (u, v) wypadnie poza mapę tekstury, to można przyjąć, że mapa tekstury jest powielana tak jak wzory z p. 2.1.3. Zamiast korzystać z odwzorowania identycznościowego między (s, t) a (u, v) możemy zdefiniować odpowiedniość między czterema rogami prostokąta 0 do 1 (s, t) i czworokąta w (u, v). Gdy powierzchnia jest wielokątem, to często przypisuje się współrzędne mapy tekstury bezpośrednio do jego wierzchołków. Ponieważ, jak widzieliśmy, wartości interpolowane liniowo wzdłuż dowolnego wielokąta mogą zależeć od orientacji, można najpierw dekomponować wielokąt na trójkąty. Jednak nawet po triangulacji interpolacja liniowa będzie powodowała zakłócenia w przypadku rzutu perspektywicznego. To zakłócenie będzie bardziej zauważalne niż zakłócenie powstające w czasie interpolowania innych informacji o cieniowaniu, ponieważ cechy tekstury nie będą dobrze oddane w rzucie perspektywicznym. Możemy uzyskać przybliżone rozwią-



Rys. 14.26. Odwzorowanie tekstury od piksela do powierzchni i do mapy tekstury

zanie tego problemu dekomponując wielokąt na kilka mniejszych albo dokładne rozwiązanie, dużym kosztem, wykonując dzielenie perspektywiczne w czasie interpolacji. Szczegółowy przegląd metod odwzorowania tekstury można znaleźć w pracy [HECK86b]

14.3.3. Odwzorowanie nierówności powierzchni

Odwzorowanie tekstury wpływa na cieniowanie powierzchni, powierzchnia natomiast w dalszym ciągu wydaje się geometrycznie gładka. Jeżeli mapa tekstury jest fotografią chropowatej powierzchni, to cieniowana powierzchnia nie będzie wyglądała dobrze, ponieważ kierunek źródła światła użytego do utworzenia mapy tekstury jest na ogół różny od kierunku źródła światła oświetlającego powierzchnię. Blinn [BLIN78b] opracował metodę uzyskiwania wyglądu zmodyfikowanej geometrii powierzchni, która nie wymaga bezpośredniego modelowania geometrycznego. W jego podejściu wykorzystuje się zakłócanie normalnej do powierzchni, zanim zostanie ona wykorzystana w modelu oświetlenia, tak jak niewielka chropowatość powierzchni zakłócałaby normalną do powierzchni. Ta metoda jest znana jako *odwzorowanie nierówności powierzchni* i bazuje na odwzorowaniu tekstury.

Mapa nierówności jest to tablica przesunięć, z których każde może być wykorzystane do symulowania przesunięcia punktu na powierzchni trochę powyżej albo trochę poniżej rzeczywistego położenia punktu. Wynik odwzorowania nierówności może być całkiem przekonujący. Obserwatorzy często nie zwracają uwagi na to, że tekstura obiektu nie wpływa na krawędzie sylwetki obiektu. Na fotografiach 38 i 39 pokazano dwa przykłady odwzorowania nierówności.

14.3.4. Inne podejścia

Chociaż odwzorowanie 2D może być efektywne w wielu sytuacjach, często nie uzyskuje się dobrych wyników. Tekstury często zatracają swój pierwotny wygląd 2D po odwzorowaniu na powierzchnie krzywoliniowe i występują problemy przy łączeniu tekstur. Na przykład gdy tekstura słoi drewna jest odwzorowywana na powierzchnię obiektu krzywoliniowego, to obiekt będzie wyglądał tak, jak gdyby był namalowany tą teksturą. Peachey [PEAC85] i Perlin [PERL85] badali wykorzystanie tekstur bryłowych do poprawnego renderingu obiektów wyciętych z drewna albo marmuru. Przy tym podejściu opisanym w rozdz. 20 książki [FOLE90] tekstura jest funkcją 3D położenia na obiekcie.

Można również odwzorowywać inne właściwości powierzchni. Na przykład Cook implementował *odwzorowanie przesunięcia*, w którym jest przesuwany bieżący punkt powierzchni, a nie tylko normalna do powierzchni [COOK84]; ten proces, który musi być wykonany przed określeniem widocznych powierzchni, był stosowany do zmodyfikowania powierzchni stożka i torusa na fot. 35. Wykorzystanie fraktali do tworzenia geometrii bogatej w szczegóły na podstawie początkowego prostego opisu geometrii omówiono w p. 9.5.1.

Dotychczas robiliśmy milczące założenie, że proces cieniowania punktu na obiekcie nie zależy od reszty tego obiektu albo od jakiegoś innego obiektu. Ale obiekt może być w rzeczywistości zasłaniany przez inny obiekt znajdujący się między nim a źródłem światła; może przepuszczać światło, dzięki czemu przez ten obiekt może być widoczny inny obiekt; albo też może odbijać inne obiekty, co umożliwia oglądanie dzięki niemu innych obiektów. W następnych punktach opisujemy, w jaki sposób można modelować niektóre z tych efektów.

14.4. Cienie

Algorytmy wyznaczania powierzchni widocznych określają, które powierzchnie mogą być widziane z punktu obserwacji; algorytmy wyznaczania cieni określają, które powierzchnie mogą być „widziane" ze źródła światła. Dlatego algorytmy wyznaczania powierzchni widocznych i algorytmy wyznaczania cieni są w zasadzie takie same. Powierzchnie, które są widoczne ze źródła światła, nie są w cieniu; te, które nie są widoczne ze źródła światła, są w cieniu. Dla wielu źródeł światła powierzchnię trzeba zaklasyfikować ze względu na każde z nich.

Zajmiemy się teraz algorytmami wyznaczania cienia dla punktowych źródeł światła; niepunktowe źródła są omawiane w rozdz. 16 książki [FOLE90]. Widzialność z punktowego źródła światła, podobnie jak widzialność z punktu obserwacji jest typu wszystko albo nic. Gdy punkt na powierzchni nie może być widziany ze źródła światła, wówczas trzeba to uwzględnić w obliczaniach związanych z oświetlaniem. Uwzględnienie cieni w równaniu oświetlenia prowadzi do zależności

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \leq i \leq m} S_i f_{\mathrm{att}_i} I_{p\lambda i} [k_d O_{d\lambda} (\overline{N} \cdot \overline{L}_i) + k_s O_{s\lambda} (\overline{R}_i \cdot \overline{V})^n] \qquad (14.22)$$

w której

$$S_i = \begin{cases} 0, \text{ jeżeli światło nie dociera do punktu} \\ 1, \text{ jeżeli światło dociera do punktu} \end{cases}$$

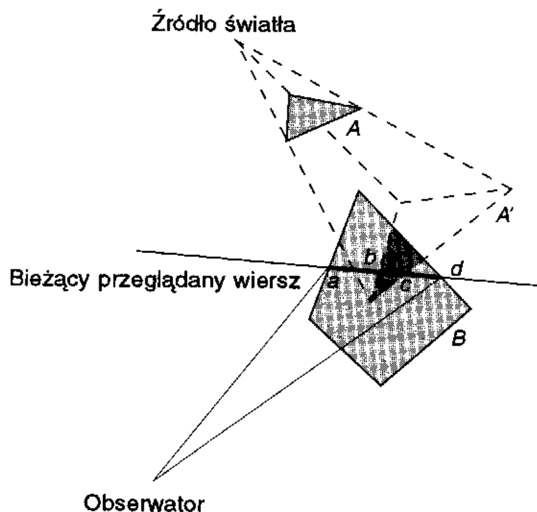
Zauważmy, że obszary znajdujące się w cieniu dla wszystkich źródeł światła są nadal oświetlone przez światło otoczenia.

Chociaż obliczanie cieni wymaga obliczania widzialności ze źródła światła, tak jak na to zwracaliśmy już uwagę, jest możliwe generowanie „fałszywych" cieni bez wykonywania jakichkolwiek testów widzialności. Takie cienie mogą być tworzone efektywnie przez przekształcenie każdego obiektu w jego rzut wielokątowy z punktowego źródła światła na wybraną płaszczyznę podstawy, bez obcinania przekształconego wielokąta do powierzchni, na którą jest rzucany cień, albo sprawdzania, czy jest blokowany przez ingerujące powierzchnie [BLIN88]. Te cienie są potem traktowane jako detale wielokątowe powierzchni. W ogólnym przypadku, kiedy te fałszywe cienie nie są odpowiednie, są możliwe różne podejścia do generownia cieni. Moglibyśmy najpierw wykonać wszystkie obliczenia związane z cieniem, przepleść je w różny sposób z przetwarzaniem powierzchni widocznych albo nawet wykonać je po zakończeniu przetwarzania związanego z powierzchniami widocznymi. Tutaj omówimy dwie klasy algorytmów wyznaczania cieni. Później w p. 14.7 i 14.8 omówimy, jak uwzględnia się cienie w metalach z globalnym oświetleniem. Przegląd innych algorytmów wyznaczania cieni zawarto w książce [FOLE90]. Dla uproszczenia opisu zakładamy, że wszystkie obiekty są wielokątami, chyba że zostanie to zaznaczone inaczej.

14.4.1. Generowanie cieni metodą przeglądania wierszy

Jedna z najstarszych metod generowania cieni polega na takim rozszerzeniu algorytmu przeglądania wierszami, żeby przepleść obliczenia związane z wyznaczaniem cieni z obliczeniami związanymi z wyznaczaniem powierzchni widocznych [APPE68; BOUK70b]. Krawędzie wielokątów, które mogłyby potencjalnie rzucać cienie, są rzutowane z wykorzystaniem źródła światła jako środka rzutowania, na wielokąty przecinające bieżącą przeszukiwaną linię. Gdy przeglądana linia przecina jedną z tych krawędzi cienia, wówczas odpowiednio modyfikuje się barwy pikseli obrazu.

W bezpośredniej implementacji tego algorytmu trzeba obliczyć wszystkie $n(n-1)$ rzutów każdego wielokąta na każdy inny wielokąt. Zamiast tego Bouknight i Kelley [BOUK70b] wykorzystują krok wstępnego przetwarzania, w którym wszystkie wielokąty są rzutowane na kulę otaczającą źródło światła, przy czym źródło światła jest środkiem rzutu. Pary rzutów, które nie pokrywają się mogą być wyeliminowane i można zidentyfikować pewną liczbę specjalnych przypadków, co umożliwi ograniczenie liczby par wielokątów, które trzeba wziąć pod uwagę



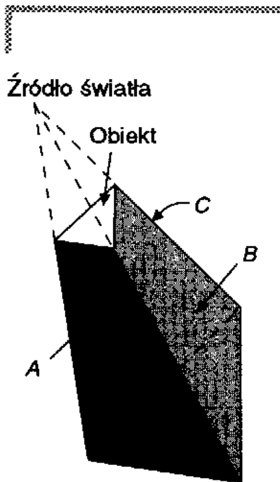
Rys. 14.27. Algorytm Bouknighta i Kelley'a wyznaczania cienia metodą przeglądania wierszy. Wielokąt *A* rzuca cień *A'* na płaszczyznę *B*

w dalszej części algorytmu. Następnie, traktując źródło światła jako środek rzutowania, autorzy obliczają rzut każdego wielokąta na płaszczyznę każdego z tych wielokątów, co do których stwierdzono, że mogłyby dawać cień (rys. 14.27). Z każdym z tych rzutów wielokątów dających cień jest związana informacja o wielokącie rzucającym cień i tym, na który ten cień jest potencjalnie rzucany. W algorytmie przeglądania wierszami są pamiętane przecinane krawędzie zwykłego wielokąta, natomiast przy przeglądaniu ze względu na cienie są pamiętane przecinane krawędzie rzutu wielokąta rzucającego cień – stąd wiadomo, w którym rzucie wielokąta dającego cień jesteśmy w danym momencie przeglądanie wiersza. Po obliczeniu cienia dla segmentu stwierdzamy, że jest on w cieniu, jeżeli przy przeglądaniu ze względu na cień jesteśmy w jednym z rzutów cienia na płaszczyznę wielokąta. Dlatego segment *bc* na rys. 14.27 jest w cieniu, a segmenty *ab* i *cd* nie są w cieniu. Zauważmy, że algorytm nie wymaga analitycznego obcinania rzutu wielokąta cieniowania z wielokątami, na które jest rzutowany cień.

14.4.2. Bryły cienia

Crow [CROW77] opisuje, jak generować cienie tworząc dla każdego obiektu bryłę cienia, jaką obiekt wytwarza dla danego źródła światła. Bryła cienia określona przez źródło światła i obiekt jest ograniczona przez zbiór niewidocznych *wielokątów cienia*. Jak pokazano na

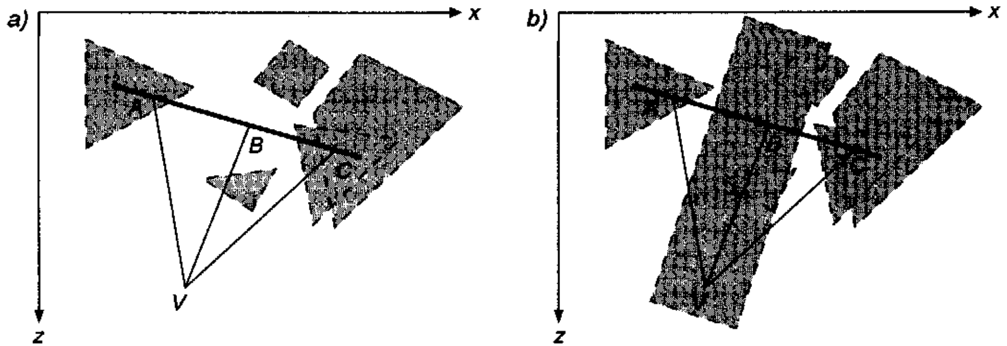
rys. 14.28, dla każdej krawędzi sylwetki obiektu względem źródła światła jest określony jeden czworokąt cienia. Trzy boki wielokąta cienia są zdefiniowane przez krawędź sylwetki obiektu i dwie linie wychodzące ze źródła światła i przechodzące przez końce krawędzi. Każdy wielokąt cienia ma normalną, która jest skierowana na zewnątrz bryły cienia. Bryły cienia są generowane tylko dla wielokątów skierowanych w stronę światła. W implementacji opisanej przez Bergerona [BERG86a] objętość bryły – i każdy z jej wielokątów – jest ograniczona z jednej strony przez wielokąt oryginalnego obiektu, z drugiej strony przez przeskalowaną kopię wielokąta obiektu, którego normalna została odwrócona. Ta przeskalowana kopia jest umieszczona w takiej odległości od źródła światła, że dalej już można przyjąć, że natężenie oświetlenia jest pomijalne. Można traktować tę odległość jako wielkość określającą *sferę wpływu* źródła światła. Dowolny punkt poza tą sferą wpływu jest efektywnie w cieniu i nie wymaga żadnych dodatkowych obliczeń związanych z cieniem. W istocie nie ma potrzeby generowania bryły cienia dla dowolnego obiektu znajdującego się całkowicie na zewnątrz sfery wpływu. Możemy uogólnić to podejście tak, żeby można je było stosować do źródeł promieniujących niejednorodnie, wprowadzając obszar wpływu, na przykład przez wybieranie obiektów na zewnątrz klap i stożka określonych dla światła. Bryłę cienia można również dalej obciąć do bryły widzenia, jeżeli jest ona z góry znana. Algorytm traktuje wielokąty bryły cienia również jako wielokąty rzucające cienie.



Rys. 14.28. Bryła cienia jest zdefiniowana przez źródło światła i obiekt

Same wielokąty rzucające cień nie podlegają renderingowi, są natomiast używane do określenia, czy obiekty znajdują się w cieniu. Z punktu widzenia obserwatora wielokąt skierowany do przodu i rzucający cień (wielokąt A albo B na rys. 14.28) powoduje, że obiekty poza nim są w cieniu; wielokąt skierowany do tyłu (wielokąt C) kasuje wpływ wielokąta przedniego. Weźmy pod uwagę wektor z punktu obserwacji V do punktu na obiekcie. Punkt jest w cieniu, jeżeli wektor przecina więcej przednich niż tylnych wielokątów cienia. Dlatego punkty A i C na rys. 14.29a są w cieniu. Jest to jedyny przypadek, kiedy punkt jest w cieniu, a V nie; stąd punkt B jest oświetlony. Jeżeli V jest w cieniu, to pojawia się dodatkowy przypadek, kiedy punkt jest w cieniu: kiedy jeszcze nie zostały napotkane wszystkie tylne wielokąty rzucające cień dla wielokątów obiektu rzucających cień na oko. Dlatego punkty A, B i C z rys. 14.29b są w cieniu, chociaż wektor z V do B przecina tę samą liczbę przednich i tylnych wielokątów cienia (rys. 14.29a).

Możemy obliczyć, czy punkt jest w cieniu, przypisując każdemu przedniemu (względem obserwatora) wielokątowi rzucającemu cień wartość $+1$ i każdemu tylnemu wielokątowi rzucającemu cień wartość -1. W liczniku początkowo jest ustawiona liczba brył cienia, które zawierają oko; zawartość licznika jest zwiększana o wartości związane



Rys. 14.29. Określanie, czy punkt jest w cieniu z punktu widzenia obserwatora znajdującego się w *V*. Linie przerywane określają bryły cienia (zaznaczone na szaro): a) *V* nie jest w cieniu; punkty *A* i *C* są w cieniu; punkt *B* jest oświetlony; b) *V* jest w cieniu; punkty *A*, *B* i *C* są w cieniu

ze wszystkimi wielokątami rzucającymi cień między okiem a punktem na obiekcie. Punkt jest w cieniu, jeżeli wartość licznika jest dodatnia w tym punkcie. Liczba brył cienia zawierających oko jest obliczana tylko raz dla każdego punktu obserwacji; bierze się wartość przeciwną do sumy wartości wszystkich wielokątów rzucających cień przecinanych przez dowolny promień od oka do nieskończoności.

W przypadku wielokrotnych źródeł światła można utworzyć niezależny zbiór brył cienia dla każdego źródła światła, oznaczając wielokąty rzucające cień danej bryły za pomocą identyfikatora odpowiedniego źródła światła i przechowując oddzielny licznik dla każdego źródła światła. Brotman i Badler [BROT84] zaimplementowali wersję algorytmu z bryłą cieni z wykorzystaniem *z*-bufora, a Bergeron [BERG86a] omawia implementację na zasadzie przeglądania linii, która jest efektywna dla dowolnych obiektów wielościennych zawierających wielokąty niepłaskie. Chin i Feiner [CHIN89] opisują algorytm z precyzją obiektową, który buduje jedną bryłę cienia dla środowiska wielokątowego, korzystając z modelowania brył metodą drzewa BSP omówioną w p. 10.6.4.

14.5. Przezroczystość

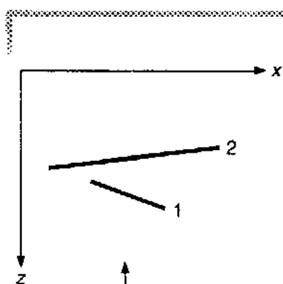
Powierzchnia może odbijać światło zwierciadlanie w sposób rozproszony, może być również przezroczysta lub półprzezroczysta i przepuszczać światło. Na ogół przez przezroczyste materiały, np. szkło, można widzieć to, co jest z tyłu, chociaż w ogólnym przypadku promienie są załamywane. W przypadku materiałów półprzezroczystych, np. matowe szkło, występuje przepuszczanie rozproszone. Promienie przechodzące

przez półprzezroczyste materiały zmieniają kierunki na powierzchniowych albo wewnętrznych nieregularnościach i dlatego obiekty widziane przez półprzezroczyste materiały są rozmazane.

14.5.1. Przezroczystość bez załamań

W najprostszym podejściu przy modelowaniu przezroczystości pomija się efekt załamania i promienie światła nie są załamywane przy przejściu przez powierzchnię. Wobec tego wszystko, co jest widzialne wzdłuż linii widzenia przez przezroczystą powierzchnię, jest również umieszczone geometrycznie na tej linii widzenia. Chociaż przezroczystość bez efektu załamania nie jest realistyczna, często można uzyskać bardziej użyteczny efekt niż z uwzględnieniem załamania. Na przykład można otrzymać obraz nie zakłócony przez powierzchnię. Jak zauważyliśmy wcześniej, pełny realizm fotograficzny nie zawsze jest celem przy generowaniu obrazów.

Na ogół stosuje się dwie różne metody aproksymowania sposobu łączenia barw dwóch obiektów, gdy jeden obiekt jest widziany przez drugi. Będziemy używali w związku z tym dwóch nazw mówiąc odpowiednio o *przezroczystości interpolowanej* albo *filtrowanej*.



Rys. 14.30. Przekrój dwóch wielokątów

Przezroczystość interpolowana. Zastanówmy się, co się stanie, jeżeli przezroczysty wielokąt 1 znajduje się między obserwatorem a nieprzezroczystym wielokątem 2 (rys. 14.30). Przezroczystość interpolowana określa barwę piksela na przecięciu rzutów dwóch wielokątów na zasadzie liniowej interpolacji między barwami obliczonymi dla każdego z dwóch wielokątów:

$$I_\lambda = (1 - k_{t1})I_{\lambda 1} + k_{t1}I_{\lambda 2} \tag{14.23}$$

Współczynnik przepuszczania k_{t1} jest miarą przezroczystości wielokąta 1 i przyjmuje wartości od 0 do 1. Gdy wartość k_{t1} wynosi 0, wówczas wielokąt jest nieprzezroczysty i nie przepuszcza światła; gdy k_{t1} wynosi 1, wówczas wielokąt jest idealnie przezroczysty i nic nie wnosi do natężenia I_λ; wartość $1 - k_{t1}$ jest określana jako *nieprzezroczystość* wielokąta. O interpolowanej przezroczystości można myśleć jak o modelowaniu wielokąta, który składa się z drobnej siatki nieprzezroczystego materiału, przez który można widzieć inne obiekty; k_{t1} jest częścią powierzchni siatki, przez którą można patrzeć. Całkowicie przezroczysty wielokąt w ten sposób przetwarzany nie będzie miał żadnych odbić zwierciadlanych. W celu uzyskania bardziej realistycznego efektu możemy interpolować tylko składowe otoczenia i odbicia rozproszonego wielokąta 1 z pełną barwą wielokąta 2, a potem dodawać składową zwierciadlaną wielokąta 1 [KAY79b].

Inne podejście, określane często jako *przezroczystość typu „ekran-drzwi"*, dosłownie implementuje siatkę; rendering jest wykonywany tylko dla niektórych pikseli związanych z rzutem obiektu przezroczystego. Mniej znaczące bity adresu piksela (x, y) są używane do indeksowania bitowej maski przezroczystości. Jeżeli indeksowany bit wynosi 1, to piksel jest zapisywany; w przeciwnym przypadku jest tłumiony i jest widoczny następny najbliższy wielokąt w tym pikselu.

Przezroczystość filtrowana. Przy przezroczystości filtrowanej wielokąt jest traktowany jako filtr przezroczysty, który selektywnie przepuszcza różne długości fali; może on być modelowany przez zależność

$$I_\lambda = I_{\lambda 1} + k_{t1} O_{t\lambda} I_{\lambda 2} \tag{14.24}$$

w której $O_{t\lambda}$ jest „barwą przezroczystości" wielokąta 1. Filtr kolorowy może być modelowany przez wybieranie różnych wartości $O_{t\lambda}$ dla każdej wartości λ. Jeżeli przed tymi wielokątami są dodatkowe wielokąty przezroczyste, to zarówno w interpolowanej, jak i filtrowanej przezroczystości rekursywnie wykonuje się obliczenia dla wielokątów w kolejności od tyłu do przodu, wykorzystując za każdym razem wcześniej obliczone I_λ jako $I_{\lambda 2}$.

Implementacja przezroczystości. Do realizacji efektu przezroczystości można łatwo adaptować różne algorytmy wyznaczania powierzchni widocznych, w tym algorytm przeglądania linii i algorytm z listą priorytetów. W algorytmach z listą priorytetów w czasie przeglądania wielokąta jest odczytywana barwa piksela, który ma być pokryty przez przezroczysty wielokąt, i jest ona wykorzystywana w modelu oświetlenia.

W większości systemów z z-buforem jest wspomaganie przezroczystości typu „ekran-drzwi", ponieważ obiekty przezroczyste mogą być mieszane z obiektami nieprzezroczystymi i mogą być rysowane w dowolnej kolejności. Dodanie efektów przezroczystości, które wykorzystują równania (14.23) i (14.24) do algorytmu z-bufora, jest trudniejsze, ponieważ rendering wielokątów odbywa się w kolejności ich napotkania. Wyobraźmy sobie rendering kilku nakładających się przezroczystych wielokątów, po których występuje jeden nieprzezroczysty. Chcielibyśmy przesunąć wielokąt nieprzezroczysty za odpowiednie wielokąty przezroczyste. Niestety z-bufor nie pamięta informacji potrzebnej do określenia, które wielokąty przezroczyste są przed wielokątem nieprzezroczystym ani nawet względnego porządku wielokątów. Jedno proste, chociaż niepoprawne podejście polega na wykonywaniu renderingu wielokątów przezroczystych na końcu, łącząc ich barwy z tymi już znajdującymi się w buforze ramki, ale bez modyfikacji z-bufora; jeżeli jednak nakładają się dwa przezroczyste wielokąty, to ich względna odległość nie jest brana pod uwagę.

Kay i Greenberg [KAY79b] zaimplementowali użyteczną aproksymację dla rosnącego tłumienia, które pojawia się blisko krawędzi

sylwetek cienkich zakrzywionych powierzchni, gdzie światło przechodzi przez grubszą warstwę materiału. Definiują oni k_t w zależności od nieliniowej funkcji składowej z wektora normalnego do powierzchni po przekształceniu perspektywicznym

$$k_t = k_{t\min} + (k_{t\max} - k_{t\min})(1 - (1 - z_N)^m)$$

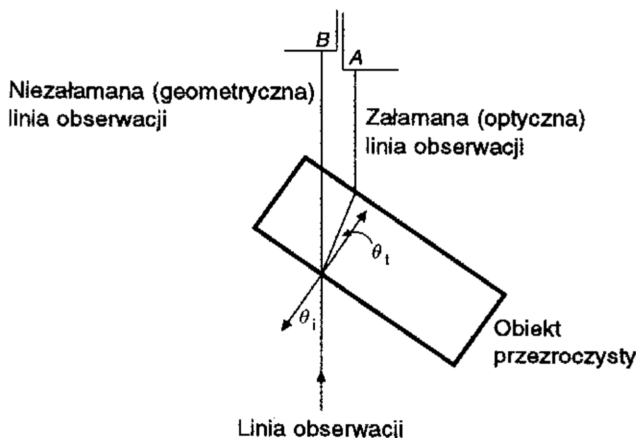
przy czym $k_{t\min}$ i $k_{t\max}$ są minimalną i maksymalną przezroczystością, z_N jest składową znormalizowanej normalnej do powierzchni w punkcie, w którym k_t jest obliczane, m jest wykładnikiem potęgi (zazwyczaj 2 albo 3). Większe wartości m modelują cieńszą powierzchnię. Ta nowa wartość k_t może być użyta jako k_{t1} w równaniu (14.23) albo (14.24).

14.5.2. Przezroczystość z załamaniem

Przezroczystość z załamaniem jest znacznie trudniejsza do modelowania niż przezroczystósc bez załamania, ponieważ linie widzenia geometryczna i optyczna różnią się. Jeżeli na rys. 14.31 uwzględnimy załamanie, to poprzez przezroczysty obiekt jest widoczny obiekt A (leży on na pokazanej linii widzenia); jeżeli pominie się załamanie, to jest widoczny obiekt B. Zależność między kątem padania θ_i a kątem załamania θ_t jest określona prawem Snella

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_{t\lambda}}{\eta_{i\lambda}} \tag{14.25}$$

przy czym $\eta_{i\lambda}$ i $\eta_{t\lambda}$ są *współczynnikami załamania* materiałów, przez które przechodzi światło. Współczynnik załamania materiału jest to stosunek prędkości światła w próżni do prędkości światła w materiale. Ten



Rys. 14.31. Załamanie

współczynnik zmienia się wraz z długością fali świetlnej, a nawet z temperaturą. Dla próżni współczynnik załamania wynosi 1,0; zbliżona wartość jest dla powietrza; inne materiały mają większe wartości współczynników. Zależność współczynnika odbicia od długości fali występuje w wielu przypadkach załamania jako *dyspersja* – znane, ale trudne do modelowania zjawisko rozszczepiania załamanego światła na jego widmo [THOM86; MUSG89].

Obliczanie wektora załamania. Wektor jednostkowy w kierunku załamania $\overline{T}$ można obliczyć jako

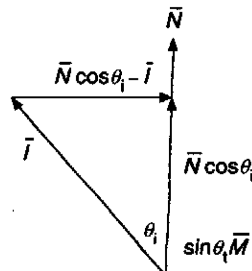
$$\overline{T} = \sin\theta_t \overline{M} - \cos\theta_t \overline{N} \tag{14.26}$$

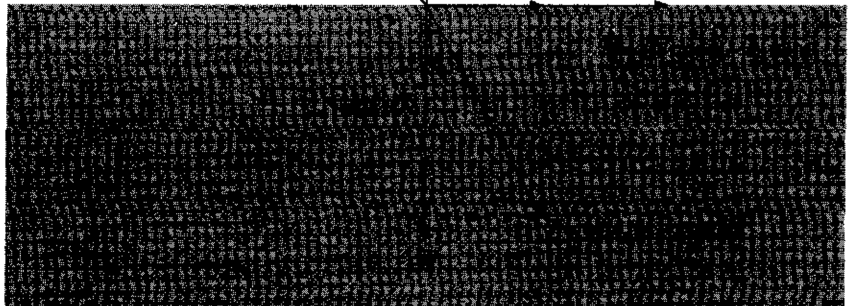
przy czym $\overline{M}$ jest wektorem jednostkowym prostopadłym do $\overline{N}$ w płaszczyźnie określonej przez promień padający $\overline{I}$ oraz wektor $\overline{N}$ [HECK84] (rys. 14.32). Przypominając wykorzystanie $\overline{S}$ w obliczaniu wektora odbicia $\overline{R}$ w p. 14.1.4 widzimy, że $\overline{M} = (\overline{N}\cos\theta_i - \overline{I})/\sin\theta_i$. Po podstawieniu otrzymujemy

$$\overline{T} = \frac{\sin\theta_t}{\sin\theta_i}(\overline{N}\cos\theta_i - \overline{I}) - \cos\theta_t \overline{N} \tag{14.27}$$

Jeżeli przyjmiemy, że $\eta_{r\lambda} = \eta_{i\lambda}/\eta_{t\lambda} = \sin\theta_t/\sin\theta_i$, to po uporządkowaniu wyrazów otrzymamy

$$\overline{T} = (\eta_{r\lambda}\cos\theta_i - \cos\theta_t)\overline{N} - \eta_{r\lambda}\overline{I} \tag{14.28}$$





Rys. 14.32. Obliczanie wektora załamania

Zauważmy, że $\cos\theta_i$ jest równy $\overline{N}\cdot\overline{I}$ i że $\cos\theta_t$ można obliczyć jako

$$\cos\theta_t = \sqrt{1-\sin^2\theta_t} = \sqrt{1-\eta_{r\lambda}^2\sin^2\theta_i} = \sqrt{1-\eta_{r\lambda}^2(1-(\overline{N}\cdot\overline{I})^2)} \tag{14.29}$$

Wobec tego

$$\overline{T} = (\eta_{r\lambda}(\overline{N}\cdot\overline{I}) - \sqrt{1-\eta_{r\lambda}^2(1-(\overline{N}\cdot\overline{I})^2)})\overline{N} - \eta_{r\lambda}\overline{I} \tag{14.30}$$

Całkowite odbicie wewnętrzne. Jeżeli światło przechodzi z jednego ośrodka do drugiego, dla którego współczynnik załamania jest mniejszy, to kąt θ_t przepuszczanego promienia jest większy niż kąt θ_i. Jeżeli θ_i stanie się dostatecznie duży, to θ_t przekroczy 90° i promień będzie odbijany od granicy między ośrodkami, a nie przepuszczany. To zjawisko jest znane jako *całkowite odbicie wewnętrzne* i najmniejszy kąt θ_i, dla którego się to pojawia, jest nazywany *kątem granicznym*. Całkowite odbicie wewnętrzne można łatwo zaobserwować patrząc przez przednią ścianę napełnionego akwarium i usiłując zobaczyć rękę przez ściankę boczną. Gdy kąt obserwacji jest większy niż kąt graniczny, wówczas jedynymi widzialnymi częściami ręki są te, które są mocno przyciśnięte do szyby akwarium, bez żadnej warstwy powietrza (które ma mniejszy współczynnik odbicia niż szkło albo woda). Kąt graniczny to taka wartość kąta θ_i, dla której $\sin\theta_t$ jest równy 1. Jeżeli w równaniu (14.25) wstawimy 1 zamiast $\sin\theta_t$, to otrzymamy, że kąt graniczny wynosi $\sin^{-1}(\eta_{t\lambda}/\eta_{i\lambda})$. Całkowite odbicie wewnętrzne pojawia się wówczas, gdy pierwiastek w równaniu (14.30) staje się urojony.

W punkcie 14.7 omówiono wykorzystanie prawa Snella do modelowania przezroczystości w metodzie śledzenia promieni.

14.6. Algorytmy oświetlenia globalnego

Model oświetlenia umożliwia wyznaczenie barwy w danym punkcie w zależności od światła emitowanego bezpośrednio przez źródła światła i świateł, które docierają do punktu po odbiciu od i przepuszczeniu przez własną i inne powierzchnie. To pośrednio odbite i przepuszczane światło jest często określane jako *oświetlenie globalne*. *Oświetlenie lokalne* jest to światło, które dochodzi bezpośrednio od źródeł światła do cieniowanego punktu. Dotychczas modelowaliśmy oświetlenie globalne za pomocą czynnika oświetlenia otoczenia, który miał stałą wartość dla wszystkich punktów we wszystkich obiektach. Nie zależał on od pozycji obiektu lub obserwatora albo od obecności lub braku bliskich obiek-

tów, które mogłyby blokować światło otoczenia. Dodatkowo pokazaliśmy kilka ograniczonych efektów oświetlenia możliwych do uzyskania przez uwzględnienie cieni i przezroczystości.

Wiele światła w rzeczywistych środowiskach nie pochodzi bezpośrednio od źródeł światła. Do generowania obrazów, które podkreślają udział oświetlenia globalnego, były używane dwie różne klasy algorytmów. W punkcie 14.7 omówiono rozszerzenie na algorytm śledzenia promieni z wyznaczaniem powierzchni widocznych, w którym są połączone problemy wyznaczania powierzchni widocznych i cieniowanie w celu wyznaczania cieni, odbić i załamania promieni. Globalne odbicie zwierciadlane i przepuszczanie uzupełniają lokalne oświetlenie zwierciadlane, rozproszone i otoczenia dla powierzchni. Metoda energetyczna omawiana w p. 14.8 całkowicie rozdziela obliczanie cieniowania i wyznaczanie powierzchni widocznych. W metodzie energetycznej modeluje się oddziaływanie środowiska ze źródłami światła, najpierw na etapie niezależnym od położenia obserwatora, a potem oblicza się jeden lub kilka obrazów dla potrzebnych punktów obserwacji, korzystając z konwencjonalnych algorytmów określania powierzchni widocznych i interpolacji cieniowania.

Rozróżnienie algorytmów zależnych od obserwatora, takich jak śledzenie promieni, niezależnych od obserwatora, takich jak metoda energetyczna, jest ważne. Algorytmy zależne od obserwatora dokonują dyskretyzacji rzutni w celu określenia punktów, w których trzeba obliczyć równanie oświetlenia dla danego kierunku obserwacji. W metodach zależnych od obserwatora algorytmy dokonują dyskretyzacji otoczenia i wykonują obliczenia w celu pozyskania dostatecznej informacji do obliczenia równania oświetlenia w dowolnym punkcie i z dowolnego kierunku. Algorytmy zależne od kierunku obserwacji dobrze nadają się do analizy zjawisk odbicia, które silnie zależą od położenia obserwatora; algorytmy te mogą być również wykorzystane przy modelowaniu zjawisk odbicia rozproszonego, które niewiele zmieniają się na dużych powierzchniach obrazu albo między obrazami otrzymanymi z różnych punktów obserwacji. Algorytmy niezależne od obserwatora modelują efektywnie zjawiska odbicia rozproszonego, wymagają natomiast ogromnej pamięci do zapamiętania odpowiednich informacji o zjawisku odbicia zwierciadlanego.

Ostatecznie wszystkie te podejścia próbują rozwiązać równanie, które Kajiya [KAJI86] określa jako *równanie renderingu*, opisujące światło emitowane od jednego punktu do drugiego w zależności od natężenia światła emitowanego z pierwszego punktu do drugiego i natężenia światła emitowanego do wszystkich innych punktów, które osiąga pierwszy punkt i jest odbijane od pierwszego punktu do drugiego. Światło przesyłane od każdego z tych innych punktów do pierwszego jest z kolei

wyrażane rekursywnie przez równanie renderingu. Kajiya przedstawia równanie renderingu w następujący sposób:

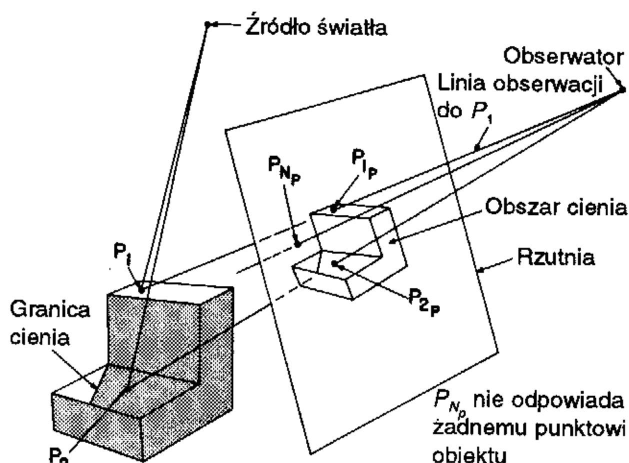
$$I(x, x') = g(x, x')[\varepsilon(x, x') + \int_S \rho(x, x', x'')I(x', x'')dx''] \quad (14.31)$$

przy czym x, x' i x'' są punktami w otoczeniu; $I(x, x')$ jest związane z natężeniem przechodzącym od x'' do x; $g(x, x')$ jest czynnikiem geometrycznym równym 0, gdy x i x' są dla siebie niewidoczne, i $1/r^2$, gdy są wzajemnie widoczne, gdzie r jest odległością między nimi; $\varepsilon(x, x')$ jest związane z natężeniem światła emitowanego od x' do x. Początkowe obliczenie $g(x, x')\varepsilon(x, x')$ dla x w punkcie obserwacji realizuje określenie powierzchni widocznej w kuli wokół x. Całka jest obliczana po wszystkich punktach na wszystkich powierzchniach S. $\rho(x, x', x'')$ jest związane z natężeniem światła odbitego (włączając zarówno odbicie zwierciadlane, jak i rozproszone) od x'' do x od powierzchni w punkcie x'. A więc równanie renderingu stwierdza, że światło z x', które dociera do x, składa się ze światła emitowanego przez samo x' i światła rozpraszanego przez x' do x od wszystkich innych powierzchni, które same emitują światło i rekursywnie rozpraszają światło od innych powierzchni.

Jak zobaczymy, to, jak skuteczne jest to podejście przy rozwiązywaniu równania renderingu, zależy w dużej mierze od tego, jak są uwzględniane pozostałe czynniki, i od rekursji, od sposobu rozwiązania kombinacji odbicia rozproszonego i zwierciadlanego oraz od tego, jak dobrze są modelowane zależności widoczności między powierzchniami.

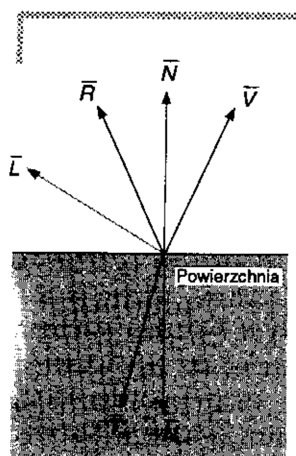
14.7. Rekursywna metoda śledzenia promieni

W tym punkcie rozszerzymy podstawowy algorytm śledzenia promieni z p. 13.4 tak, żeby uwzględnić cienie, odbicia i załamanie. Ten prosty algorytm określał barwę piksela w najbliższym przecięciu promienia wychodzącego z oka z obiektem, przy użyciu jednego z opisanych wcześniej modeli oświetlenia. W celu obliczenia cieni wyślemy dodatkowy promień z punktu przecięcia do każdego źródła światła. Pokazano to na rys. 14.33 dla jednego źródła światła; rysunek jest reprodukcją z pracy Appela [APPE68], pierwszej opublikowanej pracy na temat śledzenia promieni w grafice komputerowej. Jeżeli jeden z tych promieni związanych z wyznaczaniem cienia przecina na swojej drodze jakiś obiekt, to ten obiekt w tym punkcie jest w obszarze cienia i algorytm cieniowania pomija wkład źródła światła związanego z analizowanym promieniem wyznaczającym cień.



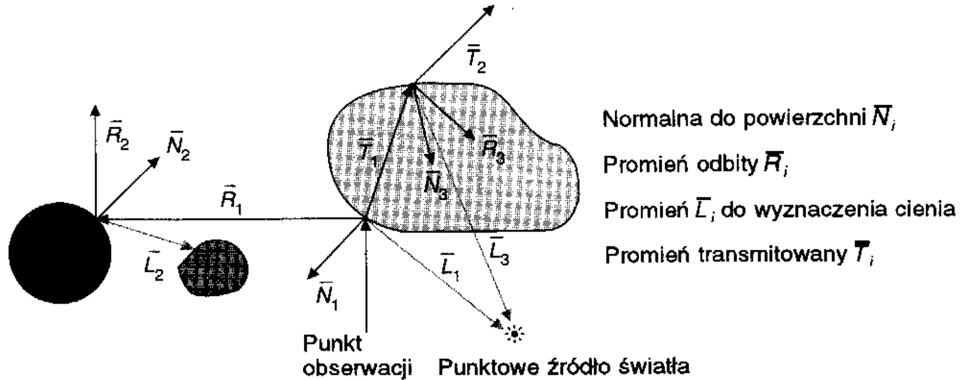
Rys. 14.33. Określanie, czy punkt obiektu jest w cieniu. (Za zgodą Artura Appela z IBM T.J. Watson Research Center.)

Model oświetlenia opracowany przez Whitteda [WHIT80] i Kaya [KAY79a] w zasadniczy sposób rozszerzył metodę śledzenia promieni o odbicie zwierciadlane i przezroczystość z załamaniem. Fotografia 41 jest jednym z pierwszych obrazów wygenerowanych z tymi efektami. Obok promieni do wyznaczania cieni rekursywny algorytm śledzenia promieni warunkowo wysyła promienie odbite i promienie załamane (rys. 14.34). Promienie do wyznaczania cienia, odbite i załamania są często określane jako *promienie wtórne* w odróżnieniu od *promieni pierwotnych* wychodzących z oka. Jeżeli obiekt odbija zwierciadlanie, to promień odbity jest odbijany od powierzchni symetrycznie względem normalnej do powierzchni w kierunku $\overline{R}$, który można obliczyć tak jak w p. 14.1.4. Jeżeli obiekt jest przezroczysty i nie występuje całkowite wewnętrzne odbicie, to jest wysyłany promień załamany do wnętrza obiektu wzdłuż $\overline{T}$ pod kątem określonym przez prawo Snella, jak to opisano w p. 14.5.2. (Zauważmy, że nasz promień padający może być zorientowany odwrotnie względem promieni z tego punktu.)



Rys. 14.34. Z punktu przecięcia wychodzą promienie odbite, załamane i do wyznaczania cieni

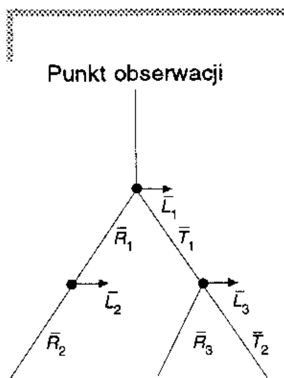
Każdy z tych odbijanych i załamanych promieni może z kolei rekursywnie wysyłać promienie do wyznaczania cieni, promienie odbite i załamane (rys. 14.35). Promienie tworzą więc drzewo promieni pokazane na rys. 14.36. W algorytmie Whitteda gałąź się kończy, gdy promienie odbity i załamany nie przecinają obiektu, a osiągnięto pewną określoną przez użytkownika maksymalną głębokość, albo gdy w systemie zabraknie pamięci. Drzewo jest obliczane metodą zstępującą i jasność każdego węzła jest obliczana jako funkcja jasności potomków.



Rys. 14.35. Promienie rekursywnie wysyłają inne promienie

Równanie oświetlenia Whitteda możemy przedstawić w postaci

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \leqslant i \leqslant m} S_i f_{\mathrm{att}_i} I_{p\lambda_i} [(k_d O_{d\lambda} (\overline{N} \cdot \overline{L_i}) + k_s (\overline{N} \cdot \overline{H_i})^n] + \\ + k_s I_{r\lambda} + k_t I_{t\lambda} \tag{14.32}$$



Rys. 14.36. Drzewo promieni dla rys. 14.35

przy czym $I_{r\lambda}$ jest natężeniem światła promienia odbitego, k_t jest współczynnikiem przepuszczania przyjmującym wartości w przedziale od 0 do 1, a $I_{t\lambda}$ jest natężeniem promienia załamanego. Wartości $I_{r\lambda}$ i $I_{t\lambda}$ są obliczane na zasadzie rekurencyjnego obliczania równania (14.32) dla najbliższej powierzchni, którą przecinają promienie odbity i przepuszczany. W celu aproksymowania tłumienia w funkcji odległości Whitted pomnożył I_λ obliczone dla każdego promienia przez odwrotność odległości przebywanej przez promień. Zamiast traktować S_i jako funkcję delta, jak w równaniu (14.22), przyjął on, że jest to funkcja ciągła k_t obiektów przecinanych przez promienie do wyznaczania cieni, tak że obiekt przezroczysty przesłania mniej światła niż obiekt nieprzezroczysty w punktach, które są w cieniu.

Program 14.1 zawiera pseudokod dla prostej rekursywnej metody śledzenia promieni. RT_trace określa najbliższe przecięcie promienia z obiektem i wywołuje RT_shade w celu określenia barwy w tym punkcie. Najpierw RT_shade określa barwę wynikającą ze światła otoczenia w punkcie przecięcia. Następnie jest wysyłany promień do wyznaczenia cienia w kierunku każdego źródła światła po tej stronie powierzchni, która ma być cieniowana, w celu określenia udziału świateł w barwie. Nieprzezroczysty obiekt blokuje światło całkowicie, a przezroczysty obiekt skaluje wkład danego źródła światła. Jeżeli nie jesteśmy zbyt głeboko w drzewie śledzenia, to są wykonywane odwołania rekursywne do RT_trace w celu przeprowadzenia obliczeń dla promieni odbijanych

od odbijających obiektów i załamanych promieni dla przezroczystych obiektów. Ponieważ do obliczenia kierunków załamanych promieni są potrzebne współczynniki załamania dla dwóch materiałów, z każdym promieniem może być związany współczynnik załamania materiału, w którym promień się porusza. RT_trace przechowuje drzewo promieni tylko dopóty, dopóki jest potrzebne do określenia barwy bieżącego piksela. Gdyby można było przechowywać drzewa promieni dla całego obrazu, wówczas właściwości powierzchni mogłyby być zmieniane i względnie szybko można by na nowo obliczyć obraz, przy koszcie ponownego przeliczenia drzew. Sequin i Smyrl [SEQU89] przedstawiają metodę, która minimalizuje czas i zajętość pamięci potrzebną do przetwarzania i pamiętania drzew promieni.

Program 14.1 Pseudokod dla prostej metody śledzenia promieni bez eliminacji zakłóceń

```
wybranie środka rzutu i pola wizualizacji;
for (każda linia obrazu) {
   for (każdy piksel w linii) {
      określenie promienia ze środka rzutowania przechodzącego przez piksel;
      pixel = RT_trace (ray, 1);
   }
}

/* Przecięcie promienia z obiektem i obliczenie barwy dla najbliższego przecięcia. */
/* Depth jest równe bieżącej głębokości w drzewie promieni. */

RT_color RT_trace ( RT_ray ray, int depth )
{
   określenie najbliższego przecięcia promienia z obiektem;
   if (trafiony obiekt) {
        obliczenie normalnej w punkcie przecięcia;
        return RT_shade (najbliższe przecięcie z obiektem, ray, intersection,
                         normal, depth);
   }
   else
      return BACKGROUND_VALUE;
}

/* Obliczenie barwy w punkcie obiektu; śledzenie promieni do wyznaczenia cieni, odbić
   i załamania */

RT_color RT_shade (
      RT_object object,     /* Przecięcie z obiektem */
      RT_ray ray,           /* Padający promień */
      RT_point point,       /* Punkt przecięcia, dla którego trzeba wyznaczyć barwę */
      RT_normal normal,     /* Normalna w punkcie */
      int depth )           /* Głębokość w drzewie promieni */
{
```

```
RT_color color;              /* Barwa promienia */
RT_ray rRay, tRay, sRay;     /* Promienie odbity, załamany i cienia */
RT_color rColor, tColor;     /* Barwy dla promienia odbitego i załamanego */

  color = ambient term;
  for (każde światło) {
     sRay = promień od punktu do światła;
     if (iloczyn skalarny normalnej i kierunku światła jest dodatni) {
        obliczenie, w jakim stopniu światło jest blokowane przez powierzchnie
        nieprzezroczyste i przezroczyste, i wykorzystanie wyników do skalowania
        czynników rozproszonego i zwierciadlanego przed dodaniem ich do barwy;
        }
     }
     if ( depth < maxDepth ) {        /* Powrót, jeżeli głębokość jest zbyt duża */
        if (obiekt odbija światło) {
           rRay = promień odbity w punkcie;
           rColor = RT_trace (rRay, depth + 1);
           skalowanie rColor za pomocą współczynnika odbicia i dodanie do color;
        }
        if (obiekt jest przezroczysty) {
           tRay = promień załamany w punkcie;
           if (nie występuje całkowite wewnętrzne odbicie) {
              tColor = RT_trace (tRay, depth + 1);
              skalowanie tColor za pomocą współczynnika przepuszczania i dodanie do color;
        }
     }
  }
  return color;      /* Przesłanie barwy promienia */
}
```

Na rysunku 14.35 pokazano podstawowy problem związany z uwzględnieniem modelu załamania w metodzie śledzenia promieni: promień cienia $\overline{L_3}$ nie jest załamany na drodze światła. Gdybyśmy po prostu załamali $\overline{L_3}$ w punkcie, gdzie wychodzi z dużego obiektu, to nie skończyłby się w źródle światła. Dodatkowo gdy są określone ścieżki promieni załamanych, dla każdego promienia jest wykorzystywany jeden współczynnik załamania.

Metoda śledzenia promieni jest szczególnie podatna na problemy powodowane przez ograniczoną precyzję obliczeń. Ujawniają się one przy obliczaniu obiektów, które przecinają się z wtórnymi promieniami. Po obliczeniu współrzędnych x, y i z punktu przecięcia obiektu widocznego dla oka, są one używane do określania punktu początkowego wtórnego promienia, dla którego musimy określić parametr t (p. 13.4.1). Jeżeli obiekt, który właśnie był przecięty, jest przecinany przez nowy promień, to często będzie on miał małą niezerową wartość t ze względu na ograniczenia numeryczne. Jeżeli nic się nie zrobi, to fałszywe przecięcie może spowodować problemy wizualne. Na przykład, gdyby promień był pro-

mieniem do wyznaczania cienia, wówczas obiekt mógłby być potraktowany jako blokujący światło sam dla siebie, co prowadziłoby do powstania plam na powierzchni niepoprawnie rzucającej cień na siebie samą. Prosty sposób rozwiązania tego problemu dla promieni do wyznaczania cieni polega na tym, żeby obiekt, z którego jest prowadzony wtórny promień, potraktować jako przypadek specjalny, tak żeby nie wykonywać dla niego testów przecięć. Oczywiście to nie działa, jeżeli występują obiekty, które rzeczywiście mogą się same przesłaniać, albo jeżeli transmitowane promienie powinny przejść przez obiekt i odbić się od wnętrza tego samego obiektu. Ogólniejsze rozwiązanie polega na obliczeniu abs(t) dla przecięcia w celu porównania z małą wartością tolerancji i pomijanie jej, jeżeli jest mniejsza niż ta tolerancja.

Praca Whitteda przedstawiona na konferencji SIGGRAPH'79 [WHIT80] oraz film, który był wykonany przy wykorzystaniu opisanego algorytmu, zapoczątkowały renesans zainteresowania metodą śledzenia promieni. Rekursywne śledzenie promieni umożliwia realizację wielu efektów robiących wrażenie, np. rzucane cienie, odbicia zwierciadlane i załamanie – które były niemożliwe albo bardzo trudne do uzyskania wcześniej. Dodatkowo prosty program dla metody śledzenia promieni jest łatwy do implementacji. W konsekwencji wiele wysiłku włożono w ulepszenie zarówno efektywności algorytmu, jak i jakości obrazu. Więcej szczegółów można znaleźć w p. 16.12 książki [FOLE90] oraz w pracy [GLAS89].

14.8. Metody energetyczne

Chociaż metoda śledzenia promieni daje bardzo dobre wyniki przy modelowaniu odbić zwierciadlanych i bezdyspersyjnego załamania przy przezroczystości, korzysta jednak z członu dla bezkierunkowego oświetlenia otoczenia razem ze wszystkimi innymi elementami oświetlenia globalnego. Podejścia wykorzystujące inżynierskie modele wymiany ciepła dla emisji i odbicia promieniowania eliminują konieczność korzystania z członu związanego ze światłem otoczenia dzięki dokładniejszemu potraktowaniu odbić międzyobiektowych. Algorytmy te były po raz pierwszy wprowadzone przez Gorala, Torrance'a, Greenberga i Battaile'a [GORA84] oraz Nishita i Nakamae [NISH85]; w tych algorytmach zakłada się zachowanie energii światła w zamkniętym środowisku. Cała energia emitowana albo odbijana od każdej powierzchni jest odbijana lub absorbowana przez inne powierzchnie. Energia opuszczająca powierzchnię jest sumą energii emitowanej przez powierzchnię, odbijanej od powierzchni albo transmitowanej od powierzchni do innych powierzchni. W konsekwencji metody wykorzystujące obliczanie promie-

niowanych energii przez powierzchnie w środowisku są określane jako *metody energetyczne*. W przeciwieństwie do konwencjonalnych metod renderingu w metodach energetycznych najpierw określa się wszystkie interakcje światła w otoczeniu w sposób niezależny od obserwatora. Następnie wykonuje się rendering dla jednego lub kilku rzutów, z czym wiąże się tylko dodatkowe obliczanie powierzchni widocznych i interpolacyjne cieniowanie.

14.8.1. Równanie energetyczne

W rozważanych wcześniej algorytmach cieniowania źródła światła były zawsze traktowane niezależnie od oświetlanych przez nie powierzchni. W metodach energetycznych każda powierzchnia może emitować światło; wobec tego wszystkie źródła światła są modelowane jako mające powierzchnię. Wyobraźmy sobie podział środowiska na skończoną liczbę n dyskretnych płatów o skończonej wielkości, emitujących i odbijających światło jednolicie na całej swojej powierzchni. Jeżeli przyjmiemy, że każdy płat jest nieprzezroczystym lambertowskim rozpraszającym emiterem i reflektorem, to dla powierzchni i

$$B_i = E_i + \rho_i \sum_{1 \leqslant j \leqslant n} B_j F_{j-i} \frac{A_j}{A_i} \tag{14.33}$$

przy czym B_i i B_j są promienistościami płatów i oraz j mierzonymi w energia/jednostka czasu/jednostka powierzchni (to jest w W/m^2), E_i jest prędkością, z jaką światło jest emitowane z płatu i i ma tę samą jednostkę co promienistość, ρ_i jest zdolnością odbijania płatu i i jest bezwymiarowe, F_{j-i} jest bezwymiarowym *współczynnikiem sprzężenia* albo *współczynnikiem konfiguracji*, który określa, jaka część energii opuszczająca całość płatu j dociera do całego płatu i, z uwzględnieniem kształtu i względnej orientacji obu płatów i obecności wszelkich zakłócających płatów. A_i i A_j są powierzchniami płatów i oraz j.

Z równania (14.33) wynika, że energia opuszczająca jednostkę powierzchni jest sumą światła emitowanego i światła odbijanego. Odbite światło jest obliczane na zasadzie skalowania sumy światła padającego przez zdolność odbijania. Z kolei światło padające jest sumą światła opuszczającego całość każdego płatu w środowisku skalowaną przez część tego światła dochodzącą do jednostki powierzchni odbierającego płatu. $B_j F_{j-i}$ jest ilością światła opuszczającego jednostkę powierzchni A_j, która dociera do wszystkich A_i. Wobec tego w celu określenia świateł opuszczających wszystkie A_j, które docierają do jednostki powierzchni A_i, konieczne jest pomnożenie przez stosunek powierzchni A_j/A_i.

Między współczynnikami sprzężenia w środowisku rozpraszającym obowiązuje wygodna prosta zależność odwrotności

$$A_i F_{i-j} = A_j F_{j-i} \tag{14.34}$$

Stąd równanie (14.33) można uprościć do postaci

$$B_i = E_i + \rho_i \sum_{1 \leqslant j \leqslant n} B_j F_{i-j} \tag{14.35}$$

Po uporządkowaniu otrzymuje się

$$B_i - \rho_i \sum_{1 \leqslant j \leqslant n} B_j F_{i-j} = E_i \tag{14.36}$$

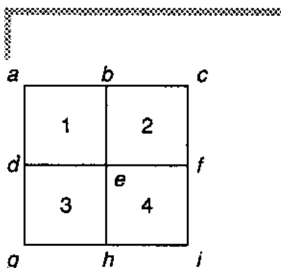
A więc oddziaływanie światła między płatami w środowisku może być zapisane jako układ równań:

$$\begin{bmatrix} 1-\rho_1 F_{1-1} & -\rho_1 F_{1-2} & \ldots & -\rho_1 F_{1-n} \\ -\rho_2 F_{2-1} & 1-\rho_2 F_{2-2} & \ldots & -\rho_2 F_{2-n} \\ \cdot & \cdot & \ldots & \cdot \\ \cdot & \cdot & \ldots & \cdot \\ \cdot & \cdot & \ldots & \cdot \\ -\rho_n F_{n-1} & -\rho_n F_{n-2} & \ldots & 1-\rho_n F_{n-n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \cdot \\ \cdot \\ \cdot \\ E_n \end{bmatrix} \tag{14.37}$$

Zauważmy, że trzeba wziąć pod uwagę udział płatu w jego własnej odbitej energii (na przykład może być on wklęsły); tak więc w ogólnym przypadku każdy człon wzdłuż przekątnej nie musi być równy 1. Równanie (14.37) musi być rozwiązane dla każdego pasma długości fal branego pod uwagę w modelu oświetlenia, ponieważ ρ_i i E_i zależą od długości fali. Współczynniki sprzężenia są jednak niezależne od długości fali i są tylko funkcją geometrii, a wobec tego nie muszą być ponownie przeliczane, jeżeli zmieni się źródło światła albo współczynnik odbicia powierzchni.

Równanie (14.37) można rozwiązać korzystając z metody iteracyjnej Gaussa-Seidela [PRES88], co daje w wyniku promienistość dla każdego płatu. Potem można dokonać renderingu płatów z dowolnego punktu obserwacji za pomocą zwykłego algorytmu wyznaczania powierzchni widocznych; zbiory promienistości obliczonych dla pasm długości fal dla każdego płatu są jasnościami płatów. Zamiast korzystać z cieniowania ścianek, możemy obliczać promienistości wierzchołków na podstawie promienistości płatów po to, żeby umożliwić interpolację jasności dla cieniowania.

Cohen i Greenberg [COHE85] sugerują następujące podejście do określania promienistości wierzchołków. Jeżeli wierzchołek jest wewnętrzny dla powierzchni, to przypisuje się mu średnią wartości promienistości dla płatów, do których należy. Jeżeli jest na krawędzi, to znajduje się najbliższy wewnętrzny wierzchołek v. Promienistość wierzchołka krawędziowego po uśrednieniu z B_v powinna być równa średniej z promienistości płatów, do których należy wierzchołek krawędziowy. Weźmy pod uwagę płaty z rys. 14.37. Promienistość dla wierzchołka wewnętrznego e jest równa $B_e = (B_1 + B_2 + B_3 + B_4)/4$. Przy obliczaniu promienistości wierzchołka krawędziowego b znajduje się jego najbliższy wierzchołek wewnętrzny e; b jest wspólne dla płatów 1 i 2. Stąd w celu obliczenia B_b korzystamy z poprzedniej definicji: $(B_b + B_e)/2 = (B_1 + B_2)/2$. Rozwiązując ze względu na B_b, otrzymujemy $B_b = B_1 + B_2 - B_e$. Wewnętrznym wierzchołkiem najbliższym a jest również e; a jest częścią płatu 1. Wobec tego, ponieważ $(B_a + B_e)/2 = B_1$, otrzymujemy $B_a = 2B_1 - B_e$. Promienistość dla innych wierzchołków oblicza się podobnie.



Rys. 14.37. Obliczanie promienistości wierzchołka na podstawie promienistości płatów

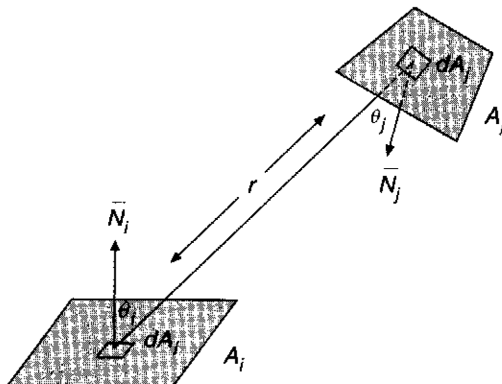
Pierwsza metoda energetyczna była zaimplementowana przez Gorala i in. [GORA84], który wykorzystywał całki krzywoliniowe do obliczania dokładnych współczynników sprzężenia dla środowisk wypukłych bez powierzchni zasłaniających, tak jak na fot. 43. Zauważmy poprawne efekty *rozlewania barwy* związane z odbiciem rozproszonym między sąsiednimi powierzchniami, widoczne zarówno w modelu, jak i w uzyskanym obrazie: rozpraszające powierzchnie są zabarwione barwami innych rozpraszających powierzchni, które one odbijają. Jednak na to, żeby metody energetyczne stały się przydatne w praktyce, trzeba najpierw opracować sposoby obliczania współczynników sprzężenia między powierzchniami.

14.8.2. Obliczanie współczynników sprzężenia

Cohen i Greenberg [COHE85] zaadaptowali algorytm wyznaczania powierzchni widocznych z precyzją obrazową do efektywnego aproksymowania współczynników kształtu dla powierzchni przesłaniających. Weźmy pod uwagę dwa płaty pokazane na rys. 14.38. Współczynnik sprzężenia od elementarnrj powierzchni dA_i do elementarnej powierzchni dA_j jest równy

$$dF_{di-dj} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij} dA_j \tag{14.38}$$

Dla promienia między elementarnymi powierzchniami dA_i i dA_j na rys. 14.38 θ_i jest kątem między promieniem a normalną do A_i, θ_j jest kątem między promieniem a normalną do A_j i r jest długością promienia.



Rys. 14.38. Obliczanie współczynnika sprzężenia między płatem a elementarną powierzchnią

H_{ij} wynosi 1 albo 0, zależnie od tego, czy dA_j jest widoczne z dA_j. W celu obliczenia F_{di-j} współczynnika sprzężenia od elementarnej powierzchni dA_i do skończonej powierzchni A_j, musimy wykonać całkowanie po powierzchni płatu j. Stąd

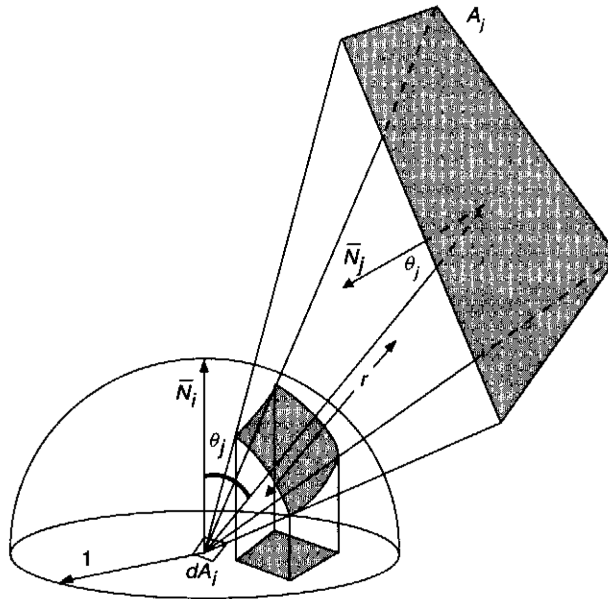
$$F_{di-j} = \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij} dA_j \tag{14.39}$$

Wreszcie współczynnik sprzężenia od A_i do A_j jest średnią z równania (14.39) po płacie i:

$$F_{i-j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij} dA_j dA_i \tag{14.40}$$

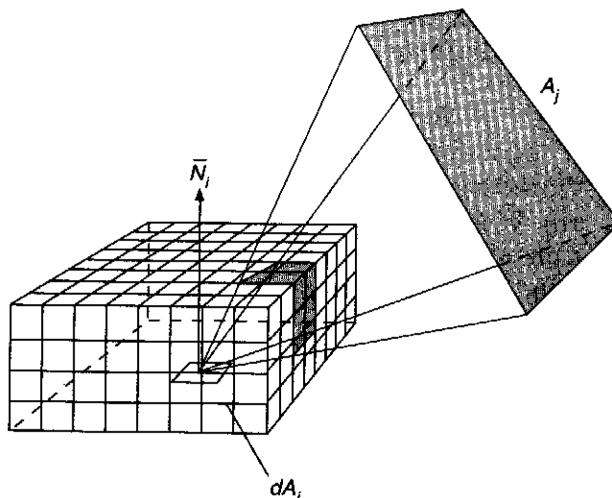
Jeżeli założymy, że wartość w punkcie środkowym płatu jest typową wartością dla innych punktów płatu, to F_{i-j} można aproksymować za pomocą F_{di-j} obliczonego dla dA_i w środku płatu i.

Nusselt pokazał [SIEG81], że obliczanie F_{di-j} jest równoważne rzutowaniu tych części A_j, które są widzialne z dA_i na jednostkową półkulę ze środkiem w dA_i, a następnie rzutowaniu tego zrzutowanego obszaru prostokątnie na jednostkowe koło w podstawie półkuli i dzieleniu przez pole koła (rys. 14.39). Rzutowanie na jednostkową półkulę odpowiada $\cos\theta_j/r^2$ w równaniu (14.39), rzutowanie na podstawę odpowiada mnożeniu przez $\cos\theta_i$, a dzielenie przez powierzchnię koła jednostkowego jest związane z π w mianowniku.



Rys. 14.39. Obliczanie współczynnika sprzężenia między elementarną powierzchnią a płatem za pomocą metody Nusselta. Stosunek powierzchni rzutowanej na podstawę półkuli do całej podstawy stanowi wartość współczynnika sprzężenia. (Według [SIEG81].)

Zamiast analitycznego rzutowania każdego A_j na półkulę Cohen i Greenberg opracowali efektywny algorytm działający z precyzją obrazową, który rzutuje na górną połowę sześcianu o środku w dA_i, przy czym górna powierzchnia sześcianu jest równoległa do powierzchni (rys. 14.40). Każda ściana tego *półsześcianu* jest dzielona na pewną liczbę jednakowych komórek kwadratowych. (Rozdzielczości wykorzystane w obrazach przedstawionych w tej książce są w zakresie od 50 × 50 do kilkuset na ścianie.) Wszystkie inne płaty są obcinane do ostrosłupa widzenia określonego przez środek sześcianu i każdą z pięciu górnych ścian, a potem każdy z obciętych płatów jest rzutowany na odpowiednią ścianę półsześcianu. Jest tu wykorzystywany algorytm z *buforem pozycji* [WEGH84], który rejestruje tożsamość najbliższego przecinanego płatu w każdej komórce. Te bufory pozycji mogą być obliczane na zasadzie wykonywania algorytmu z-bufora dla każdej strony półsześcianu i rejestrowaniu numeru najbliższego płatu w każdej komórce zamiast barwy. Z każdą komórką półsześcianu jest związany obliczony wcześniej *współczynnik sprzężenia delta* zależny od jej pozycji. Dla dowolnego płatu j, F_{di-j} może być aproksymowane w wyniku sumowania wartości współczynników sprzężenia delta związanych ze wszystkimi komórkami półsześcianu, które zawierają numer płatu j. Ponieważ wiele obliczeń wykonywanych z wykorzystaniem półsześcianu jest związa-



Rys. 14.40. Półsześcian stanowi górną połowę sześcianu, w którego środku znajduje się płat. (Według [COHE85].)

nych z obliczaniem buforów pozycji, można wykorzystać istniejący z-bufor sprzętowy. Półsześcian jest podatny na zakłócenia, ponieważ wykorzystuje on operacje z precyzją obrazową.

14.8.3. Progresywne ulepszanie

Biorąc pod uwagę wysokie koszty wykonywania algorytmu energetycznego opisanego dotychczas, sensowne jest pytanie, czy możliwe jest przyrostowe przybliżanie wyników algorytmu. Czy możemy szybko utworzyć użyteczne obrazy, być może niedokładne, a potem je sukcesywnie polepszać, zwiększając dokładność wówczas, gdy jest więcej czasu do dyspozycji? Podejście energetyczne opisane w poprzednich punktach nie pozwoli nam na to z dwóch powodów. Po pierwsze, cała iteracja Gaussa-Seidela musi być wykonana, zanim będzie dostępne oszacowanie promieniowania przez płat. Po drugie, współczynniki sprzężenia są obliczane między wszystkimi płatami na początku i muszą być zapamiętane przez czas wszystkich obliczeń, co wymaga czasu $O(n^2)$ i pamięci. Cohen, Chen, Wallace i Greenberg [COHE88] opracowali algorytm energetyczny z progresywnym ulepszaniem, który daje odpowiedź na oba te problemy.

Zajmijmy się podejściem opisanym dotychczas. Obliczając i-ty wiersz w równaniu (14.37) uzyskuje się oszacowanie promienistości B_i płatu i, wyrażonej równaniem (14.35), na podstawie oszacowania pro-

mienistości innych płatów. Każdy wyraz w sumie z równania (14.35) reprezentuje wpływ płatu j na promienistość płatu i:

$$B_i \text{ ze względu na } B_j = \rho_i B_j F_{i-j} \text{ dla wszystkich } j \qquad (14.41)$$

Wobec tego przy tym podejściu gromadzi się światło od reszty środowiska. W metodzie progresywnego ulepszania wysyła się promieniowanie z płatu do otoczenia. Bezpośrednia metoda wykonania tego polega na zmodyfikowaniu równania (14.41) w następujący sposób:

$$B_i \text{ ze względu na } B_i = \rho_j B_i F_{j-i} \text{ dla wszystkich } j \qquad (14.42)$$

Mając oszacowanie B_i, wkład płatu i do reszty otoczenia może być określony z równania (14.42) dla każdego płatu j. Niestety, wymagałoby to znajomości F_{j-i} dla każdego j, przy czym każda taka wartość jest określana za pomocą oddzielnego półsześcianu. Powoduje to równie duży narzut czasowo-pamięciowy jak w podejściu oryginalnym. Korzystając z zależności odwrotnej do równania (14.34) możemy przepisać równanie (14.42) jako

$$B_j \text{ ze względu na } B_i = \rho_j B_i F_{i-j} \frac{A_i}{A_j} \text{ dla wszystkich } j \qquad (14.43)$$

Obliczenie tego równania dla każdego j wymaga tylko współczynników sprzężenia obliczanych za pomocą półsześcianu z płatem j w środku. Jeżeli można szybko obliczyć współczynniki sprzężenia od płatu i (na przykład z wykorzystaniem sprzętowego z-bufora), to można je odrzucać od razu, gdy promienistość płatu i została obliczona. A więc tylko jeden półsześcian i związane z nim współczynniki sprzężenia muszą być obliczane i pamiętane równocześnie.

Gdy tylko energia z płatu została wypromieniowana, wybiera się inny płat. Płat może być ponownie wybrany do wypromieniowania energii, gdy padło na niego nowe światło z innych płatów. Dlatego nie jest wypromieniowana całkowita obliczona energia z płata i, a raczej energia ΔB_i, którą płat i otrzymał od ostatniego wypromieniowania energii. Algorytm działa iteracyjnie do chwili, kiedy zostanie osiągnięta wymagana tolerancja. Zamiast wybierać płaty w dowolnym porządku, jest sensowne wybieranie płatu, który wniesie największą zmianę. Jest to płat, który ma największą energię do wypromieniowania. Ponieważ energia jest mierzona na jednostkę powierzchni, wybierany jest płat i, dla którego wartość $\Delta B_i A_i$ jest największa. Początkowo $B_i = \Delta B_i = E_i$ dla wszystkich płatów, przy czym B_i jest różne od zera tylko dla źródeł światła. Program 14.2 zawiera pseudokod dla jednej iteracji.

Program 14.2 Pseudokod wysyłania energii z płatu

```
wybranie płatu i;

obliczenie F_{i-j} dla każdego płatu j;

for (każdy płat j) {
        ΔRadiosity = ρ_j ΔB_i F_{i-j} A_i / A_j;
        ΔB_j += ΔRadiosity;
        B_j += ΔRadiosity;
}

ΔB_i = 0;
```

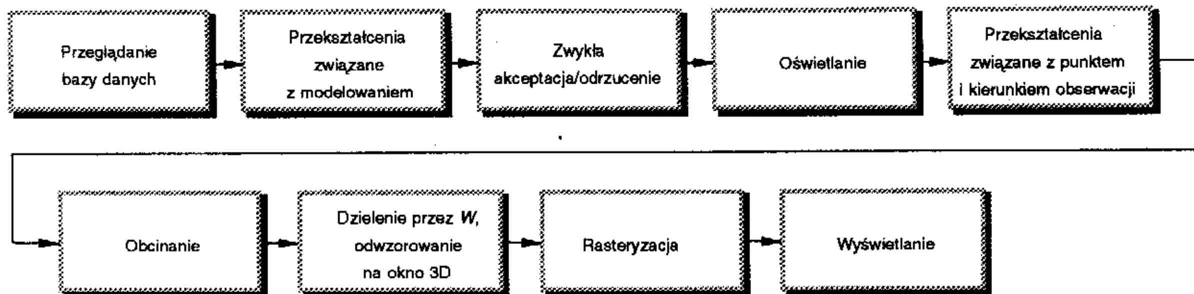
Każde wykonanie pseudokodu z programu 14.2 spowoduje wysłanie nie wysłanej jeszcze energii przez inny płat do otoczenia. Dlatego jedynymi powierzchniami, które są oświetlone po pierwszym wykonaniu, są te, które są źródłami światła, i te, które są oświetlone bezpośrednio przez pierwszy płat, który wysyła energię. Jeżeli na końcu każdego wykonania wykonuje się rendering obrazu, to pierwszy obraz będzie względnie ciemny, a następne będą coraz jaśniejsze. Chcąc zwiększyć użyteczność początkowych obrazów możemy do wysyłanych energii dodać czynnik związany ze światłem otoczenia. Przy każdym dodatkowym przejściu przez pętlę czynnik związany ze światłem otoczenia będzie zmniejszany aż w końcu zniknie. Na fotografii 44, przy którego renderingu było wykorzystane światło otoczenia, pokazano etapy tworzenia obrazu po 1, 2, 24 i 100 iteracjach.

14.9. Potok renderingu

Teraz po poznaniu różnych sposobów określania powierzchni widocznych, oświetlania i wyznaczania barwy pokażemy, jak te procesy pasują do standardowego potoku graficznego wprowadzonego w rozdz. 7. Dla uproszczenia założymy środowisko wielokątowe, dopóki nie przyjmiemy, że jest inaczej. W rozdziale 18 książki [FOLE90] podano bardziej szczegółową dyskusję, jak te potoki można zrealizować sprzętowo.

14.9.1. Potoki dla oświetlenia lokalnego

z-bufor i cieniowanie Gourauda. Prawdopodobnie najbardziej oczywista modyfikacja potoku występuje w systemie, który wykorzystuje algorytm z-bufora określania powierzchni widocznych przy renderingu wie-



Rys. 14.41. Potok renderingu dla z-bufora i cieniowania Gourauda

lokątów z cieniowaniem Gourauda (rys. 14.41). Algorytm z-bufora ma tę zaletę, że prymitywy można przetwarzać w dowolnej kolejności. Tak jak poprzednio, prymitywy otrzymuje się w wyniku przeglądania bazy danych i są one przekształcane za pomocą przekształcenia modelowania w układ WC.

Z prymitywami mogą być związane normalne do powierzchni, które zostały określone w czasie tworzenia modelu. Ponieważ na etapie oświetlania będą potrzebne normalne do powierzchni, trzeba pamiętać o tym, żeby poprawnie wykonać przekształcenia związane z normalnymi. Dalej, nie możemy po prostu pominąć zapamiętanych normalnych i próbować później liczyć je na nowo, korzystając z poprawnie przekształconych wierzchołków. Normalne zdefiniowane z obiektami mogą reprezentować prawdziwą geometrię powierzchni albo mogą określać zdefiniowane przez użytkownika powierzchniowe efekty mieszania, a nie tylko wynikać ze średnich dla normalnych wspólnych ścian w wielokątowej aproksymacji siatkowej.

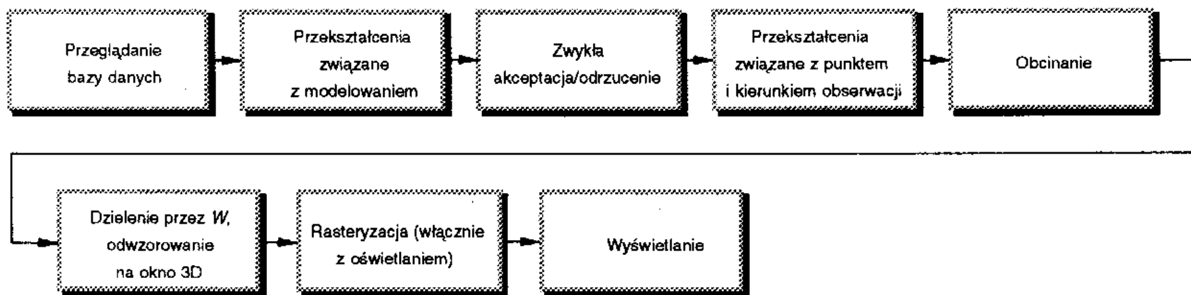
Nasz następny krok polega na wybieraniu prymitywów, które znajdują się całkowicie poza oknem i wybraniu tylnych ścian. Ten etap zwykłego odrzucania na ogół jest wykonywany w tym kroku, ponieważ chcemy wyeliminować niepotrzebne obliczenia w następnym kroku związanym z oświetlaniem. Teraz, ponieważ korzystamy z cieniowania Gourauda, dla każdego wierzchołka obliczamy równanie oświetlenia. Ta operacja musi być wykonana w układzie WC (albo w każdym innym układzie współrzędnych izometrycznym do niego) przed przekształceniami związanymi z rzutowaniem (które mogą obejmować przekształcenia związane z pochyleniem i perspektywą) w celu zachowania poprawnego kąta i odległości od każdego źródła światła do powierzchni. Gdyby normalne do powierzchni nie były dostarczone razem z obiektem, mogą być obliczone bezpośrednio przed oświetleniem wierzchołków. Wybieranie i oświetlanie są często wykonywane w układzie współrzędnych oświetlenia, który jest przekształconym układem WC (na przykład

układem VRC, gdy jest tworzona macierz orientacji rzutowania za pomocą standardowych narzędzi PHIGS).

Następnie obiekty są przekształcane w układ NPC za pomocą przekształcenia rzutowania i obcinane do bryły widzenia. Jest wykonywane dzielenie przez W i następuje odwzorowanie obiektów na pole wizualizacji. Jeżeli obiekt jest częściowo obcięty, to muszą być obliczone poprawne wartości jasności dla wierzchołków utworzonych w czasie obcinania. W tym punkcie obcięty prymityw jest kierowany do algorytmu z-bufora, który wykonuje rasteryzację, przeplatając wierszową konwersję z interpolacją potrzebną do obliczania wartości z i barwy dla każdego piksela. Jeżeli ostatecznie zostanie określone, że piksel jest widoczny, to jego barwa może być dalej zmodyfikowana w celu uwzględnienia wpływu głębokości (równanie (14.11)), co nie jest tutaj pokazane.

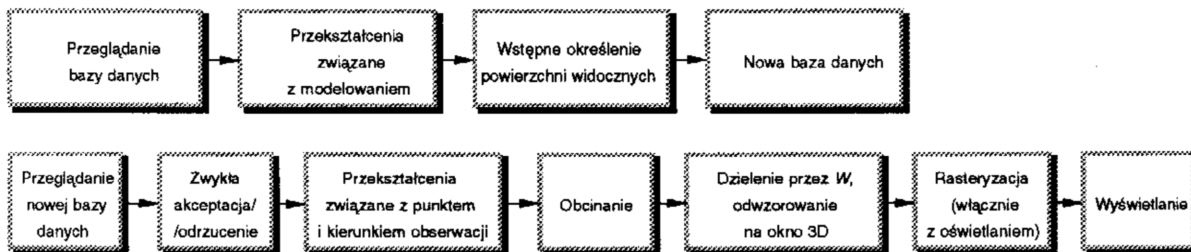
Chociaż ten potok wydaje się oczywisty, jest wiele nowych elementów, które trzeba wziąć pod uwagę żeby, uzyskać efektywną i poprawną implementację. Na przykład weźmy pod uwagę problemy powstające w przypadku powierzchni krzywoliniowych, takich jak płaty B-sklejane, które muszą być pokryte mozaiką. Pokrywanie mozaiką powinno być wykonywane po przekształceniu w układ współrzędnych, w którym można określić wielkość ekranu. Umożliwia to adaptacyjne ustalenie wielkości mozaiki i ogranicza ilość danych, które trzeba poddać przekształceniom. Prymitywy pokryte mozaiką muszą być oświetlone w układzie współrzędnych izometrycznym do układu współrzędnych świata. Abi-Ezzi [ABIE89] analizuje ten problem i proponuje bardziej efektywne, chociaż bardziej złożone sformułowanie potoku, który zawiera pętle sprzężenia zwrotnego. Ten nowy potok wykorzystuje układ współrzędnych oświetlenia, który jest izometrycznym (przekształceniem ciała sztywnego – euklidesowym) przekształceniem układu WC i jest obliczeniowo zbliżony do układu DC, co umożliwia efektywne podejmowanie decyzji co do pokrycia mozaiką.

z-bufor i cieniowanie Phonga. Jeżeli chcemy stosować cieniowanie Phonga, to prosty potok musi być zmodyfikowany tak jak na rys. 14.42. Ponieważ cieniowanie Phonga interpoluje normalne do powierzchni a nie jasności, wierzchołki nie mogą być oświetlone wcześniej w potoku. Każdy obiekt musi być obcięty (z właściwie interpolowanymi normalnymi tworzonymi dla każdego nowo utworzonego wierzchołka), przekształcony za pomocą przekształceń rzutowania i przesłany do algorytmu z-bufora. Wreszcie jest wykonywane oświetlenie z interpolowanymi normalnymi do powierzchni, które są wyznaczane w czasie konwersji wierszowej. Dlatego każdy punkt i jego normalna muszą być z powrotem odwzorowane do układu współrzędnych, który jest izometryczny do układu WC, w celu obliczenia równania oświetlenia.



Rys. 14.42. Potok renderingu dla z-bufora i cieniowania Phonga

Algorytm z listami priorytetów i cieniowaniem Phonga. Gdy jest wykorzystywany algorytm z listami priorytetów, prymitywy otrzymane w wyniku przeglądania i przetworzone za pomocą przekształceń modelowania są umieszczane w niezależnej bazie danych, takiej jak drzewo BSP, w ramach wstępnego określania powierzchni widocznych. Na rysunku 14.43 pokazano potok dla algorytmu z drzewem BSP, dla którego wstępne określenie powierzchni widocznych nie zależy od obserwatora. Jak zauważyliśmy w rozdz. 7, program użytkowy i pakiet graficzny mogą mieć swoje niezależne bazy danych. Tutaj widzimy, że rendering może potrzebować jeszcze jednej bazy danych. Ponieważ w tym przypadku wielokąty są dzielone, musi być tworzona poprawna informacja dla nowo tworzonych wierzchołków. Teraz można przeszukać bazę danych renderingu tak, żeby uzyskać prymitywy w poprawnej kolejności od tyłu do przodu. Narzut związany z tworzeniem takiej bazy danych może być oczywiście akceptowany przy tworzeniu wielu obrazów. Dlatego pokazaliśmy to jako oddzielny potok, którego wyjściem jest nowa baza danych. Prymitywy uzyskiwane z bazy danych renderingu są obcinane, normalizowane i są przekazywane do następnych etapów obliczeń w potoku. Te etapy są skonstruowane w większości tak jak w przypadku potoku z z-buforem, z wyjątkiem tego, że jedyny pro-



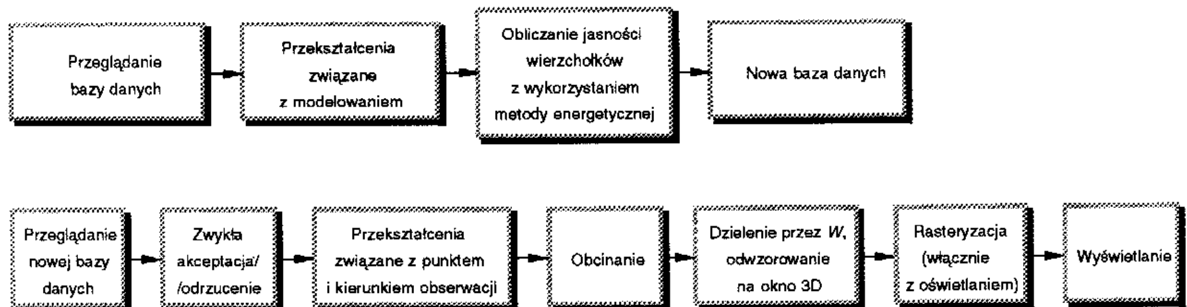
Rys. 14.43. Potok renderingu dla algorytmu z listą priorytetów i cieniowaniem Phonga

ces określania powierzchni widocznych, który musi być wykonany, musi gwarantować to, żeby każdy wielokąt poprawnie przesłaniał każdy przecinany wielokąt wcześniej poddany konwersji.

14.9.2. Potoki dla oświetlenia globalnego

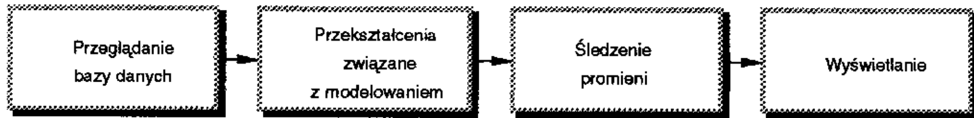
Dotychczas pomijaliśmy oświetlenie globalne. Jak wcześniej zauważyliśmy, uwzględnienie informacji o efektach związanych z oświetleniem globalnym wymaga informacji o zależnościach geometrycznych między obiektem poddawanym renderingowi a innymi obiektami świata. W przypadku cieni, które zależą tylko od położenia źródła światła, a nie od położenia obserwatora, wstępne przetwarzanie środowiska w celu dodania wielokątów reprezentujących cienie umożliwia użycie skądinąd konwencjonalnego potoku.

Metoda energetyczna. Algorytmy energetyczne dostarczają interesującego przykładu, w jaki sposób można wykorzystać zalety konwencjonalnego potoku w celu otrzymania efektów oświetlenia globalnego. Algorytmy z p. 14.8 przetwarzają obiekty i przypisują im zbiór jasności wierzchołków niezależnych od obserwatora. Te obiekty mogą z kolei być skierowane do zmodyfikowanej wersji potoku z *z*-buforem i cieniowaniem Gourauda (rys. 14.44), w którym jest wyeliminowany etap oświetlenia.



Rys. 14.44. Potok renderingu dla metody energetycznej i cieniowania Gourauda

Metoda śledzenia promieni. Wreszcie rozważmy metodę śledzenia promieni, dla której potok pokazano na rys. 14.45 i który jest najprostszy, ponieważ te obiekty, które są widoczne w każdym pikselu i ich oświetlenie są określone całkowicie w układzie WC. Po otrzymaniu obiektów z bazy danych i przekształceniu za pomocą przekształceń



Rys. 14.45. Potok renderingu dla metody śledzenia promieni

związanych z modelowaniem, są one ładowane do bazy danych (w układzie WC) programu śledzenia promieni, który zawiera optymalizację obliczeń związanych z przecinaniem promieni.

14.9.3. Metoda progresywnych ulepszeń

W jednej z interesujących modyfikacji potoków bierze się pod uwagę fakt, że obraz jest oglądany przez pewien czas. Zamiast próbować dokonać renderingu końcowej wersji całego obrazu, możemy najpierw dokonać zgrubnego renderingu, a potem progresywnie ulepszać go. Na przykład pierwszy obraz mógłby być bez usuniętych zakłóceń, mógłby zawierać uproszczone modele obiektów i prostsze cieniowanie. Gdy użytkownik ogląda obraz, wówczas puste cykle mogłyby być wykorzystane do poprawienia jego jakości [FORR85]. Jeżeli jest pewna miara, która umożliwi określenie, co robić dalej, to ulepszanie może następować adaptacyjnie. Bergman, Fuchs, Grant i Spach [BERG86b] opracowali taki system, w którym stosuje się różne metody heurystyczne do określania, jak powinien on wykorzystywać czas. Na przykład wielokąt jest cieniowany metodą Gourauda, a nie za pomocą stałej barwy jedynie wówczas, gdy zakres jasności wierzchołków przekracza wartość progową. Obie metody śledzenia promieni [PAIN89] i energetyczna [COHE88] są podatne na progresywne ulepszanie.

Podsumowanie

W rozdziale tym poznaliśmy wiele różnych modeli oświetlenia, niektóre z nich były inspirowane koniecznością zwiększenia efektywności, a inne próbą uwzględnienia fizycznego opisu oddziaływania światła z powierzchnią. Zobaczyliśmy, jak interpolacja może być użyta w modelach cieniowania, zarówno w celu zminimalizowania liczby punktów, w których trzeba obliczać równanie oświetlenia, jak i umożliwienia aproksymacji powierzchni krzywoliniowych za pomocą siatek wielokąto-

wych. Przeciwstawiliśmy podejście polegające na oświetleniu lokalnym, w którym każdy punkt powierzchni jest traktowany w izolacji i światła oświetlają każdy punkt niezależnie, podejściu globalnemu, które uwzględnia załamanie i odbicie od innych obiektów w środowisku.

Jak podkreślaliśmy, mnogość algorytmów oświetlenia i cieniowania daje możliwość tworzenia różnych obrazów dla tej samej sceny przy takich samych parametrach obserwacji. Decyzja o tym, których algorytmów należy użyć, zależy od wielu czynników, w tym od celów, dla jakich wykonuje się rendering obrazu. Chociaż często rezygnuje się z fotorealizmu na rzecz efektywności, postępy w rozwoju algorytmów i sprzętu wkrótce doprowadzą do fizycznie poprawnych implementacji z globalnymi modelami oświetlenia działającymi w czasie rzeczywistym. Jeżeli jednak efektywność nie jest problemem, to możemy mimo to wybrać rendering niektórych obrazów bez tekstury, cieni albo załamania, ponieważ w niektórych przypadkach będzie to najlepszym sposobem przekazywania potrzebnej informacji obserwatorowi.

Zadania

14.1. a) Opisz różnice w wyglądzie, jakich można by się spodziewać, między modelem oświetlenia Phonga, w którym wykorzystuje się $(\overline{N} \cdot \overline{H})^n$, a tym, który wykorzystuje $(\overline{R} \cdot \overline{V})^n$. b) Pokaż, że $\alpha = 2\beta$, gdy wszystkie wektory na rys. 14.12 są koplanarne. c) Pokaż, że ta zależność nie jest słuszna w ogólnym przypadku.

14.2. Udowodnij, że wyniki interpolacji informacji związanych z wierzchołkami wzdłuż krawędzi wielokąta i wierszy są w przypadku trójkątów niezależne od orientacji.

14.3. Załóż, że ten sam promień rzutujący przecina trzy wielokąty A, B i C w kolejności wynikającej z rosnącej odległości od obserwatora. Pokaż, że w ogólnym przypadku, jeżeli wielokąty A i B są przezroczyste, to barwa obliczona dla piksela na przecięciu ich rzutów będzie zależała od tego, czy równanie (14.23) jest obliczane dla wielokątów A i B traktowanych jako wielokąty 1 i 2 albo jako wielokąty 2 i 1.

14.4. Rozważ użycie odwzorowania tekstury do zmodyfikowania albo zastąpienia różnych właściwości materiału. Wymień efekty, jakie można utworzyć odwzorowując poszczególne właściwości albo ich kombinacje.

14.5. Jakie inne efekty oświetlenia można zaproponować w celu uogólnienia klap i stożków Warna?

14.6. Zaimplementuj prostą metodę śledzenia promieni, korzystając z materiału podanego w p. 13.4 i 14.7.

14.7. Wyjaśnij, dlaczego oświetlenie musi być wykonane przed obcinaniem w potoku z rys. 14.41.

14.8. Zaimplementuj program testujący dla przeprowadzenia eksperymentów z lokalnymi modelami oświetlenia. Zapamiętaj obraz, który zawiera dla każdego piksela indeks widocznej powierzchni do tablicy właściwości materiałów, normalną do powierzchni, odległość od obserwatora i odległość od znormalizowanego wektora do jednego lub wielu źródeł światła. Pozwól użytkownikowi modyfikować równanie oświetlenia, intensywność, barwę świateł i właściwości powierzchni. Za każdym razem, gdy jest wykonywana zmiana, należy dokonać renderingu powierzchni. Wykorzystaj równanie (14.20) z tłumieniem źródła światła (równanie (14.8)) i uwzględnieniem wpływu odległości (równanie (14.11)).

Skorowidz

A

Addytywne dodawanie barw 507
Adresowalność 180
Aliasing 33, 167
Animacja 538
–, język 540
–, ramka
–, – kluczowa 539
–, – pośrednia 539
–, sterowanie 540
–, zakłócenia czasowe 538
API 303, 369
Aproksymacja półtonowa 493
Architektura systemu wyświetlania 100

B

Barwa 497
–, dobór 518
– dopełniająca 505
–, gama barw 505
–, interakcyjny wybór 515
–, interpolacja 516
–, modele 506
–, – CMY 508
–, – CMYK 509
–, – HSV 511
–, – RGB 507
–, – YIQ
– niespektralna 505
Béziera krzywe 415, 421
– powierzchnie 441
Bouknight-Kelley, algorytm 620
B-rep 467
Bresenham 108, 119
Bryła cienia 621
– ograniczająca 552
– widzenia 263
– – kanoniczna 278
B-sklejane krzywe 427
– – – jednorodne nieułamkowe 427
– – – – nieułmkowe 432
– – – – ułamkowe (NURBS) 435
– powierzchnie 442
BSP algorytm 577
– drzewo 479
Bufor głębokości 557
– z 643, 645, 557

C

Catmull, algorytm 582
Chromatyczności wykres 502
Ciągłość krzywej geometryczna 413
– – parametryczna 414
CIE 502
Ciekłokrystaliczny wyświetlacz 189
Cienie 535
–, wyznaczanie 619
Cieniowanie 532, 589
– Gourauda 608, 610
– interpolacyjne 533
– Phonga 611
– siatki wielokątowej 609
– stałą wartością 607
– wielokątów 607
– z interpolacją 608
–, problemy 613
Cień (barwa) 498
CMY, model 509
CMYK, model 509
Cohen-Greenberg, algorytm 640
Cohen-Sutherland 147
Cook-Torrance, model 606
CSG 481
Cyrus-Beck, algorytm 153
Czujnik położenia 3D 380
Czystość pobudzenia 498

D

Detale wielokątowe 615
Długość fal dominująca 498
Dobór barw 519

Dopasowanie barw, funkcja 501
Dopełniająca barwa 505
Drążek sterowniczy 209
Drukarka atramentowa 182
– laserowa 181
– matrycowa 180
– termiczna 183
Drzazgi 130
Drzewa BSP 479
– czwórkowe 475
– ósemkowe 475
– –, operacje boolowskie 477
– –, znajdowanie sąsiadów 478
Drutowy model 460
Dyfuzja błędów 495
Dynamika 537

E

Elektroluminescencyjny wyświetlacz 191
Energetyczna metoda 635, 647
Eulera operatory 471
– reguła 469

F

Filtr prostopadłościenny 171
– stożkowy 172
Floyd-Steinberg, metoda dyfuzji błędów 495
Fotorealizm 523
Fraktale 447
–, samopodobieństwo 448
–, wymiar 448
–, zbiór
–, – Julia-Fatou 449
–, – Mandelbrota 449
Funkcja bazowa 416

G

Gama barw 505
Gamma korekcja 491
Gęstość strumienia 605
GKS 303
Głębia ostrości 536
Gouraud, cieniowanie 608, 610, 643
Gumki metoda 395

H

Hermite'a krzywe 415, 416
– powierzchnie 439
Hierarchia obiektów 303, 310, 556
HOOPS 303, 369
HSV, model 511

I

Iloczyn skalarny 218
– wektorowy 221
Iluminant C 503
Interakcja, metody 374
–, menu 387
–, model zastosowania 38
–, obsługa 41
–, schemat 37
–, sprzęt 375
–, wybór
–, – barw 515
–, metodą wprowadzania tekstu 391
–, – – wartości 391
–, wyświetlanie modelu 39
–, zadania 374
–, – pozycjonowania 382
–, wybór 383
–, – złożone 395
Interpolacja barw 516

J

Jaskrawość 490, 497
Jasność 497
Julia-Fatou, zbiór 449

K

Kajiya, równanie 629
Kamera, model 536
– syntetyczna 253
Kąt rastra 494
Klapy Warna 602
Klawiatura 211, 377
Kolorymetria 498
Konstruktywna geometria brył 481
Konwersja odcinków 105
– –, algorytm przyrostowy 106
– –, – z punktem środkowym 108
– –, – – Wu-Rokne 113
– okręgów 118
Kopie trwałe 179
Kopiowanie prymitywów 466
Korelacja wskazywania 356
Krzywe Béziera 415, 421
– B-sklejane 427
– – jednorodne nieułamkowe 427
– – niejednorodne
– – – nieułmkowe 432
– – – ułamkowe (NURBS) 435
– Hermite'a 415, 416
– parametryczne, rysowanie 429
–, porównanie 437
– sklejane 415
Kula śledząca 209

L

LCD 189
LUT 192
Lamberta prawo 593
Lokalizatory 207, 376
Luminancja 490, 498
– energetyczna 605

M

Macha pasma 609, 613
Macierz 220
– bazowa 415
– geometrii 415
–, odwracanie 222
– ortonormalna 230
–, transpozycja 222
Mandelbrota zbiór 449
Mapa bitowa 28
– nierówności 618
Malarski algorytm 575
Materiał, właściwości 534
Menu 387
– chwilowe 389
– hierarchiczne 388
– jednopoziomowe 387
–, klucze funkcyjne 390
–, pola dialogowe 395
–, rozmieszczenie 388

Menu rozwijane 389
Mikrowzory, metoda 494
Model cieniowania 589
– – Gourauda 610, 643
– – Phonga 611
– – wielokątów 607
– oświetlenia 589, 591
– – Cooka-Torrance'a 606
– – Phonga 599, 645
– – Warna 602
– – Whitteda 632
– świateł 589
– Torrance'a-Sparrowa 606
–, zastosowania 312
Modelowanie brył 459
– –, b-rep 468
– –, CSG 481
– –, interfejs użytkownika 486
– –, kopiowanie prymitywów 466
– –, model drutowy 460
– –, podział przestrzenny 473
– –, porównanie metod reprezentacji 485
– –, regularyzowane operacje boolowskie 461
– –, reprezentacje z przesuwaniem 467
– geometryczne 305
– –, hierarchia modeli 307
– powierzchni 403
–, siatki wielokątowe 404
Monitor, częstotliwość krytyczna 187
–, – odświeżania 187
–, maska
–, – delte-delta 187
–, – delta PIL 187
–, migotanie 187
–, pasmo 187
Munsell, zbiór barw 497
Myszka 209

N

Natężenie napromieniowania 605
– promieniowania 604
– światła 490
NTSC 205
NURBS 435

O

Obcinanie 143, 174
– odcinków 143
– –, algorytm Cohena-Sutherlanda 147
– –, – parametryczny 152
– okręgów 157
– punktów 143
– we współrzędnych jednorodnych 292
– w funkcji odległości 530
– wielokątów 158
– –, algorytm Sutherlanda-Hodgmana 158
– 3D 291
Obliczenia przyrostowe 122
Obszary ograniczające 553
–, testowanie minmax 553
Odbicie lambertowskie 592
– rozproszone 592
– wewnętrzne, całkowite 628
– zwierciadlane 598
Odcień barwy 499
Odpowiedź oka, funkcja 500
Odwzorowanie nierówności powierzchni 618
– przesunięcia 619
– tekstury 617
Okno 235
OpenGL 303, 369
Operacje boolowskie 471
– regularyzowane 461
Oświetlanie 532, 589
– globalne 628
– lokalne 628
– obiektu 591

P

Para stereo 541
Parametryczna reprezentacja krzywych 410
Perspektywa dwupunktowa 259
– jednopunktowa 258
– powietrzna 530
Perspektywiczne przekształcenie 549
PEX 369
PHIGS 45, 303, 369
PHIGS PLUS 303, 370
Phong, model oświetlenia 599, 645
Piksel 28
Ploter bębnowy 181
– płaski 181
Płaszczyzna rzutowania 262
Płaty bikubiczne 402
Podział adaptacyjny 555
– przestrzenny bryły 473
– – problemu 555
Pogrubione prymitywy 140
Potok renderingu 643
– – dla oświetlenia
– – – – globalnego 643
– – – – lokalnego 643
Powierzchnie krzywoliniowe drugiego stopnia 446
– – trzeciego stopnia 438
– – Béziera 441
– – B-sklejana 442
– – Hermite'a 439
– –, modelowanie 534
– – parametryczne bikubiczne 438
– –, wyświetlanie 444
Powierzchnie widoczne 532, 545
– –, algorytm BSP 577
– –, – Catmulla 582
– –, – malarski 575
– –, – przeglądania 561
– –, – śledzenia promieni 566
– –, – z listą priorytetów 574
– –, – podziału powierzchni 579
– –, – Warnocka 579
– –, – wyznaczania dla powierzchni krzywoliniowych 582
– –, –, porównanie 584
– –, – z-bufora 557
– –, – z precyzją obiektową 546
– –, – – – obrazową 546
Półsześcian 640
Półtony 494
Procesor wyświetlania 197
Progresywne ulepszanie 641, 648
Promienistość wierzchołka 638
Prostokąt ograniczający 552

Prymitywy graficzne 46
Przeciąganie, metoda 397
Przecięcia, metoda obliczania 567
Przekształcenia geometryczne 215
–, okno na pole wizualizacji 234
–, składanie 232
–, współrzędne jednorodne 226
– – 2D 224
– – –, obracanie 225, 229
– – –, pochylanie 231
– – –, przesuwanie 224, 228
– – –, skalowanie 225, 229
– – 3D 238
– – –, macierze 239
– – –, składanie 242
Przekształcenie perspektywiczne 549
Przenośność oprogramowania 33
Przesuwanie wzoru 467
Przezroczystość 535, 623
– filtrowana 625
–, implementacja 625
– interpolowana 624
– typu ekran-drzwi 624
– z załamaniem 626

R

Ramka kluczowa 539
– pośrednia 539
Rasteryzacja 174, 524
Rastrowy system wyświetlania 192
Realistyczny obraz 524
Reprezentowanie brył 460
– krzywych 402
– powierzchni 402
Rękawiczka DataGlove 381
RGB, model 507
Rendering 524
–, metody dla obrazów cieniowanych 532
–, potok 643
–, równanie 628
Rozdzielczość 180
Rozlewanie barwy 638
Rozmycie ruchu 536
Równanie energetyczne 636
– oświetlenia 591
– renderingu 629
Rzutowanie 3D 253
Rzuty, implementacja 280
–, macierze
– perspektywiczne 257, 268, 280, 529
– prostokątne 528
– równoległe 256, 259, 273, 280
– – aksonometryczne 260
– – ukośne 261

S

Siatki wielokątowe 403
Skalowanie jednorodne 225
– niejednorodne 225
Sklejane krzywe 415
Snella prawo 626
SPHIGS 45, 303
–, atrybuty 344
–, edycja 350
–, hierarchia 331
–, implementacja 364
–, interakcja 355
–, prymitywy wyjściowe 317
–, przekształcenia 326
–, rendering 349
–, rzutowanie 321
–, składanie macierzy 343
–, struktury 315
–, wyświetlanie struktur 320
Skaner 212
Spójność 548
– implikowana 548
– głębokości 548
– krawędziowa 125, 131, 548
– liniowa 548
– międzywierszowa 124
– obiektów 548
– powierzchni 548
– przestrzenna 124
– ramek 548
– segmentowa 124
– ściany 548
SRGP 45
–, atrybuty 52
–, – grupa 59
–, barwa 54
–, tabela barw 54
–, styl linii 52
–, szerokość linii 52
–, interakcja 62
–, –, czynniki ludzkie 62
–, –, korelacja wskazywania 74
–, logiczne urządzenia wejściowe 63
–, – tryb
–, –, – próbkowania 65, 68
–, – – zdarzeń 65, 69
–, ustawianie stanu urządzeń i atrybutów 76
–, kanwa 79
–, ograniczenia 90
–, operacje logiczne 85
–, prostokąt obcinający 83
–, prymitywy graficzne 46
–, elipsa 51
–, łamana 48
–, wielokąt 50
–, wypełnianie 55
–, znaczniki 49
–, tekst 59
Stacja graficzna 178
Standardy graficzne 33
Stereopsja 541
Sterownik wyświetlania 204
Stożek barw CIE 503
Stożki Warna 602
Subtraktywne barwy 508
Sutherland-Hodgman, algorytm 158
Sztuczna rzeczywistość 380

Ś

Ściana przednia 554
– tylna 554
– –, wybieranie 554
Śledzenie promieni, metoda 630, 647, 566
– –, efektywność 571
– –, obliczanie przecięć 567
Światło achromatyczne 489
– barwne 489
– otoczenia 591

T

Tabela barw 196
Tabliczka 207

Teksel 616
Tekst 174
Tekstura 534, 531
–, odwzorowanie 616
Tinta 498
Tłumienie atmosferyczne 597
Ton 498
Torrance-Sparrow, model 606
Trójpobudzeniowa teoria 499

U

Układy współrzędnych 298
– –, zmiana 247
Urządzenia wejściowe 373
– – do wprowadzania wartości 378
– – interakcyjne 379
– –, klawiatury 377
– –, lokalizatory 376
– – wybierające 378

W

Warn, model oświetlenia 602
Warnock, algorytm 579
Wektor 217
– odbicia 600
– połowiczny 601
– załamania 627
Wektoryzacja 213
Whitted, model oświetlenia 631
Widoczne linie 531
Widzenie dwuoczne 541
Wielokąt rozpięty 423
Wizualizacja naukowa 538
Wokselowa reprezentacja 474
Współczynnik konfiguracji 636
– sprzężenia 636
– –, obliczanie 638
– załamania 626
Współrzędne ekranu 234
– jednorodne 226
– świata 234
Wypełnianie 124, 174
– prostokąta 124
– wielokąta 126
– wzorami 13
Wyświetlanie, metody 185
Wyznacznik 221

Y

YIQ, model 509

Z

Załamanie 626
Zakłócenia, metody usuwania 167, 174
–, – –, próbkowanie
–, – – bezwagowe 168
–, – – wagowe 170
z-bufor 643, 645, 557
Zastosowania grafiki komputerowej 22
Znaki, generowanie 162

Ź

Źródło światła kierunkowe 594
– – punktowe 533, 592
– –, tłumienie 595