Навчально-методична література

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

Кафедра комп'ютерно-
інтегрованих технологій

КОМП'ЮТЕРНА ГРАФІКА

Книга 1

НАВЧАЛЬНИЙ ПОСІБНИК

для студентів спеціальності
151 «Автоматизація та комп'ютерно-інтегровані технології»

Тернопіль
2017

УДК 681.3

К63

Укладач
Тотосько О.В., канд. техн. наук, доцент,
Микитишин А.Г., канд. техн. наук, доцент,
Стухляк П.Д., докт. техн. наук, професор.

Рецензенти:
Марущак П.О., докт. техн. наук, професор.

Методичні вказівки розглянуто й затверджено на засіданні
методичного семінару кафедри комп'ютерно-інтегрованих технологій
Тернопільського національного технічного університету імені Івана Пулюя
протокол № 1 від 29 серпня 2016 р.

Схвалено та рекомендовано до друку науково-методичною комісією
факультету прикладних інформаційних технологій та електроінженерії
Тернопільського національного технічного університету імені Івана Пулюя
протокол № 1 від 29 серпня 2016 р.

К63 Комп'ютерна графіка : навчальний посібник : в 2-х кн.1. для студентів спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» / Укладачі : Тотосько О.В., Микитишин А.Г., Стухляк П.Д. – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2017 – 304 с.

УДК 681.3

Посібник складено з урахуванням матеріалів літературних джерел, наведених у переліку.

Відповідальний за випуск *Тотосько О.В*., канд. техн. наук, доцент.

© Тотосько О.В., Микитишин А.Г.,
Стухляк П.Д., ……...2017
© Тернопільський національний технічний університет імені Івана Пулюя, …….2017

ЗМІСТ

Зміст 3

Тема 1. ЗАГАЛЬНЕ ВВЕДЕННЯ В КОМП'ЮТЕРНУ ГРАФІКУ 8

1.1. Предмет і область застосування комп'ютерної графіки 8
1.2. Коротка історія 11
1.3. Технічні засоби підтримки комп'ютерної графіки 13
1.4. Питання й вправи 20
Тема 2. СУЧАСНІ АПАРАТНІ ЗАСОБИ РАСТРОВОЇ ГРАФІКИ 21

2.1. Основні поняття 21
2.2. Пристрої введення 22
2.2.1. Сканери 22
2.2.2. Цифрові фотоапарати й відеокамери 23
2.3. Пристрою виводу 25
2.3.1. Дисплеї 25
2.3.1.1. Дисплеї на ЕПТ 25
2.3.1.2. Рідкокристалічні дисплеї 26
2.3.1.4. Інші типи дисплеїв 27
2.3.2. Проектори 28
2.3.3 .Принтери 28
2.3.3.1. Матричні принтери 29
2.3.3.2. Струминеві принтери 29
2.3.3.3. Лазерні принтери 29
2.4. Архітектура графічної підсистеми ПК 30
2.4.1. Архітектура 30
2.4.2. Подання зображень 33
2.4.3. Програмний інтерфейс 34
2.5. Питання й вправи 36
Тема 3. КОЛІР У КОМП'ЮТЕРНІЙ ГРАФІЦІ 37

3.1. Про природу світла й кольору 37
3.2. Колірний графік МКО 39
3.3. Колірні моделі RGB і CMY 42
3.4. Колірні моделі HSV і HLS 44
3.5. Простір CIE Luv 47
3.6. Питання й вправи 50
Тема 4. ГЕОМЕТРИЧНІ ПЕРЕТВОРЕННЯ 51

4.1. Системи координат і вектори 51
4.2. Рівняння прямої на площині 56
4.2.1. Аналітичне подання кривих і поверхонь 57
4.3. Перетинання променю із площиною й сферою 59
4.4. Інтерполяція функцій однієї й двох змінних 60
4.4. Матриці 62
4.5. Геометричні перетворення (перенос, масштабування, обертання) 63

4.6. Перехід в іншу систему координат 66
4.7. Завдання обертання щодо довільної осі 68
4.8. Питання й вправи 69
Тема 5. ПОДАННЯ ГЕОМЕТРИЧНОЇ ІНФОРМАЦІЇ 71

5.1. Геометричні примітиви 71
5.2. Полігональні моделі 72
5.3. Воксельні моделі 73
5.4. Поверхні вільних форм (функціональні моделі) 73
5.5. Системи координат: світового, об'єктна, спостерігача й екранна 74
5.6. Однорідні координати. Завдання геометричних перетворень в однорідних координатах за допомогою матриць 76
5.7. Питання й вправи 79
Тема 6. АЛГОРИТМИ РАСТЕРИЗАЦІЇ ВІДРІЗКІВ, КІЛ І ЕЛІПСІВ 80

6.1. Введення в растеризацію кривих 80
6.2. Зображення відрізка із цілочисловими координатами кінців 81
6.2.1. Цифровий диференціальний аналізатор 82
6.2.2. Алгоритм Брезенхема 83
6.2.3. Алгоритм Кастла-Пітвея 84
6.3. Зображення відрізка з нецілочисловими координатами кінців 84
6.4. Зображення кіл 86
6.4.1. Алгоритм Брезенхема 87
6.5. Зображення еліпсів 90
6.5.1. Побудова по неявній функції 90
6.5.2. Побудова шляхом стиску кіл 91
Тема 7. ВІДСІКАННЯ (КЛіпування) ГЕОМЕТРИЧНИХ ПРИМІТИВІВ 92

7.1. Алгоритм Сазерленда-Коена відсікання прямокутною областю 92
7.2. Відсікання опуклим багатокутником 98
7.3. Кліпування багатокутників 100
7.4. Питання й вправи 102
Тема 8. ВИДАЛЕННЯ НЕВИДИМИХ ПОВЕРХОНЬ І ЛІНІЙ 104

8.1. Алгоритм Робертса 105
8.2. Алгоритм Варнока 107
8.3. Алгоритм Вейлера-Азертона 109
8.4. Метод Z-Буфера 110
8.5. Методи пріоритетів 111
8.6. Алгоритми порядкового сканування для криволінійних поверхонь 114
8.7. Метод двійкової розбивки простору 115
8.8. Метод трасування променів 116
8.9. Питання й вправи 120
Тема 9. ПРОЕКТУВАННЯ ПРОСТОРОВИХ СЦЕН 121

9.1. Основні типи проекцій 121
9.2. Паралельні проекції 122

9.3. Центральні проекції 124
9.4. Математичний апарат 125
9.5. Ортогональні проекції 125
9.6. Косокутні проекції 126
9.7. Центральні проекції 127
9.8. Спеціальні картографічні проекції. Екзотичні проекції земної сфери .. 129
9.9. Стереографічна проекція 130
9.10. Гномонічна проекція 131
9.11. Ортографічна проекція 131
9.12. Проекції на циліндр 132
9.13 Проекція Меркатора 133
9.14 Проекції на багатогранник 133
9.15. Незвичайні проекції 134
9.16. Питання й вправи 135
Тема 10. РАСТРОВЕ ПЕРЕТВОРЕННЯ ГРАФІЧНИХ ПРИМІТИВІВ 136
10.1. Алгоритм Брезенхема растрової дискретизації відрізка 136
10.2. Алгоритми Брезенхема растрової дискретизації кола й еліпса 140
10.3. Алгоритми заповнення областей 145
10.4. Питання й вправи 147
Тема 11. ЗАФАРБОВУВАННЯ. РЕНДЕРИНГ ПОЛІГОНАЛЬНИХ МОДЕЛЕЙ 148
11.1. Проста модель висвітлення 148
11.2. Зафарбування граней 152
11.2.1. Плоске зафарбовування 152
11.2.2. Зафарбування методом Гуро 152
11.2.3. Зафарбування методом Фонга 153
11.3. Більше складні моделі висвітлення 154
11.4. Усунення ступінчастості (антиелайзінг) 155
11.5 Питання й вправи 159
Тема 12. ВІЗУАЛІЗАЦІЯ ПРОСТОРОВИХ РЕАЛІСТИЧНИХ СЦЕН 161
12.1. Світло-Тіньовий аналіз 161
12.2. Метод випромінювання 163
12.3. Глобальна модель висвітлення із трасуванням променів 165
12.4. Текстури 168
12.5 Питання й вправи 170
Тема 13. АЛГОРИТМИ СТИСКУ ЗОБРАЖЕНЬ БЕЗ ВТРАТ 172
13.1. Необхідність стиску зображень 172
13.2. Неіснування ідеального алгоритму 172
13.3. Алгоритми кодування довжини повторення (RLE) 173
13.3.1. RLE – бітовий рівень 173
13.3.2. RLE - байтовий рівень 173
13.4. Словникові алгоритми 174

13.4.1. Алгоритм LZ77 174
13.4.2. Алгоритм LZW 177
13.4.3. Алгоритми статистичного кодування 182
13.4.4. Алгоритм Хаффмена 182
13.4.5. Арифметичне кодування 185
Тема 14. СТИСК ЗОБРАЖЕНЬ ІЗ ВТРАТАМИ 189

14.1. Необхідність стиску із втратами 189
14.2. Оцінка втрат 189
14.3. Зображення як функція 190
14.4. Дискретне Перетворення Фур'є 191
14.5. Дискретне косинусне перетворення 192
14.6. Алгоритм стиску зображень JPEG 192
14.7. Вейвлет-Перетворення 196
14.8. Фрактальний стиск 199
14.9. Список літератури 201
Тема 15. АЛГОРИТМИ СТИСКУ ВІДЕО 204

15.1. Введення 204
15.2. Основні поняття 204
15.3. Вимоги додатків до алгоритму 205
15.4. Визначення вимог 206
15.5. Огляд стандартів 207
15.6. Базові технології стиску відео 209
15.6.1. Опис алгоритму компресії 209
15.6.2. Загальна схема алгоритму 210
15.6.3. Використання векторів зсувів блоків 211
15.6.4. Можливості по розпаралелюваності 212
15.7. Інші шляхи підвищення ступеня стиску 212
15.7.1. Motion-JPEG 214
15.7.2. MPEG-1 214
15.7.3. H.261 215
15.7.4. H.263 215
15.7.5. MPEG-2 216
15.7.6. MPEG-4 217
15.7.7. Порівняння стандартів 220
15.8. Питання для самоконтролю 221
Тема 16. ЦИФРОВЕ ФОТО 222

16.1. Історія фотографії 222
16.2. Від плівки до цифрового фото 237
16.3. Умовна класифікація цифрових фотоапаратів 248
16.4. Сенсори цифрових фотоапаратів 258
Тема 17. ПОДАННЯ САЙТУ 268

17.1. Рух 269
17.2. Зменшення швидкості руху 269

17.2.1. Створення кліпу зі зменшенням швидкості руху 270
17.2.2. Додавання змінної швидкості руху 272
17.2.3. Ініціалізація руху клацанням миші 272
17.3. Додавання сліду від руху об'єкта 273
17.3.1. Зникнення сліду 275
17.3.2. Підвищення ефективності 276
17.4. Покадровий рух 277
17.4.1. Постійна швидкість 278
17.4.2. Коливальний рух 280
17.4.3. Зміна розмірів у русі 284
17.5. Керування рухом 286
17.5.1. Ковзання 287
17.5.2. Використання порожніх фільмів 288
17.6. Реалізація підходів на практиці 293
17.6.1. Оглядач зображень 293

Тема 1. ЗАГАЛЬНЕ ВВЕДЕННЯ В КОМП'ЮТЕРНУ ГРАФІКУ

Предмет і області застосування комп'ютерної графіки. Коротка історія розвитку комп'ютерної графіки. Технічні засоби підтримки комп'ютерної графіки: ЕПТ, пристрою введення, відеоадаптер, графопобудівувачі, принтери, сканери. Програмні засоби підтримки комп'ютерної графіки: драйвери пристроїв, бібліотеки графічних програм, спеціалізовані графічні системи й пакети програм

1.1. Предмет і область застосування комп'ютерної графіки

Комп'ютерна графіка – це область інформатики, що охоплює всі сторони формування зображень за допомогою комп'ютера. З'явившись в 1950-х роках, вона спочатку давала можливість виводити лише кілька десятків відрізків на екрані. У наші дні засобу комп'ютерної графіки дозволяють створювати реалістичні зображення, що не уступають фотографічним знімкам. Створено різноманітне апаратне й програмне забезпечення для одержання зображень всілякого виду й призначення – від простих креслень до реалістичних образів природних об'єктів. Комп'ютерна графіка використовується практично у всіх наукових і інженерних дисциплінах для наочності сприйняття й передачі інформації. Застосування її для підготовки демонстраційних слайдів уже вважається нормою. Тривимірні зображення використовуються в медицині (комп'ютерна томографія), картографії, поліграфії, геофізиці, ядерній фізиці й іншим областям. Телебачення й інші галузі індустрії розваг використовують анімаційні засоби комп'ютерної графіки (комп'ютерні ігри, фільми). Загальноприйнятою практикою вважається також використання комп'ютерного моделювання при навчанні пілотів і представників інших професій (тренажери). Знання основ комп'ютерної графіки зараз необхідно й інженерові, і вченому.

Кінцевим результатом застосування засобів комп'ютерної графіки є зображення, що може використовуватися для різних цілей. Оскільки найбільша кількість інформації людина одержує за допомогою зору, уже в древні часи з'явилися схеми й карти, використовувані при будівництві, у географії й в астрономії.

Сучасна комп'ютерна графіка – це досить складна, ґрунтовно пророблена й різноманітна науково-технічна дисципліна. Деякі її розділи, такі як геометричні перетворення, способи опису кривих і поверхонь, до теперішнього часу вже досліджені досить повно. Ряд областей продовжує активно розвиватися: методи растрового сканування, видалення невидимих ліній і поверхонь, моделювання кольору й освітленості, текстурування, створення ефекту прозорості й напівпрозорості й ін.

Сфера застосування комп'ютерної графіки включає чотири основних області.

1. Відображення інформації

Проблема подання накопиченої інформації (наприклад, даних про кліматичні зміни за тривалий період, про динаміку популяцій тваринного миру, про екологічний стан різних регіонів і т.п.) найкраще може бути вирішена за допомогою графічного відображення.

Жодна з областей сучасної науки не обходиться без графічного подання інформації. Крім візуалізації результатів експериментів і аналізу даних спостережень існує велика область математичного моделювання процесів і явищ, що просто немислима без графічного виводу. Наприклад, описати процеси, що протікають в атмосфері або океані, без відповідних наочних картин течій або полів температури практично неможливо. У геології в результаті обробки тривимірних натурних даних можна одержати геометрію шарів, що залягають на великій глибині.

У медицині в цей час широко використовуються методи діагностики, що використовують комп'ютерну візуалізацію внутрішніх органів людини. Томографія (зокрема, ультразвукове дослідження) дозволяє одержати тривимірну інформацію, що потім піддається математичній обробці й виводиться на екран. Крім цього застосовується й двовимірна графіка: енцефалограми, міограми, виведені на екран комп'ютера або графопобудовувача.

2. Проектування

У будівництві й техніці, креслення давно являє собою основу проектування нових споруджень або виробів. Процес проектування з необхідністю є ітеративним, тобто конструктор перебирає безліч варіантів з метою вибору оптимального по яких-небудь параметрах. Не останню роль у цьому грають вимоги замовника, що не завжди чітко уявляє собі кінцеву мету й технічні можливості. Побудова попередніх макетів – досить довга й дорога справа. Сьогодні існують розвинені програмні засоби автоматизації проектно-конструкторських робіт (САПР), що дозволяють швидко створювати креслення об'єктів, виконувати різні розрахунки й т.п. Вони дають можливість не тільки зобразити проекції виробу, але й розглянути його в об'ємному виді з різних сторін. Такі засоби також надзвичайно корисні для дизайнерів інтер'єра, ландшафту.

3. Моделювання

Під моделюванням у цьому випадку розуміється імітація різного роду ситуацій, що виникають, наприклад, при польоті літака або космічного апарата, русі автомобіля й т.п. В англійській мові це найкраще передається терміном simulation. Але моделювання використовується не тільки при створенні різного роду тренажерів. У телевізійній рекламі, у науково-популярних і інших фільмах тепер синтезуються об'єкти, що рухаються, що візуально мало уступають тим, які можуть бути отримані за допомогою кінокамери. Крім того, комп'ютерна графіка надала кіноіндустрії можливості створення спецефектів, які в попередні роки були попросту неможливі. В останні роки широко поширилася ще одна сфера застосування комп'ютерної графіки – створення віртуальної реальності.

4. Графічний користувальницький інтерфейс

На ранньому етапі використання дисплеїв як одного з пристроїв комп'ютерного виводу інформації діалог "людина-комп'ютер" в основному здійснювався в алфавітно-цифровому виді. Тепер же практично всі системи програмування застосовують графічний інтерфейс. Особливо значно виглядають розробки в області мережі Internet. Існує безліч різних програм-

браузерів, що реалізують у тім або іншому виді засобу спілкування в мережі, без яких доступ до неї важко собі представити. Ці програми працюють у різних операційних середовищах, але реалізують, по суті, ті самі функції, що включають вікна, банери, анімацію й т.д.

У сучасній комп'ютерній графіці можна виділити наступні основні напрямки: образотворча комп'ютерна графіка, обробка й аналіз зображень, аналіз сцен (перцептивна комп'ютерна графіка), комп'ютерна графіка для наукових абстракцій (когнітивна комп'ютерна графіка, тобто графіка, що сприяє пізнанню).

Образотворча комп'ютерна графіка своїм предметом має синтезовані зображення. Основні види завдань, які вона вирішує, зводяться до наступного:

- побудова моделі об'єкта й формування зображення;
- перетворення моделі й зображення;
- ідентифікація об'єкта й одержання необхідної інформації.

Обробка й аналіз зображень стосуються в основному дискретного (цифрового) подання фотографій і інших зображень. Засоби комп'ютерної графіки тут використовуються для:

- підвищення якості зображення;
- оцінки зображення – визначення форми, місця розташування, розмірів і інших параметрів необхідних об'єктів;
- розпізнавання образів – виділення й класифікації властивостей об'єктів (при обробці аерокосмічних знімків, введенні креслень, у системах навігації, виявлення й наведення).

Аналіз сцен пов'язаний з дослідженням абстрактних моделей графічних об'єктів і взаємозв'язків між ними. Об'єкти можуть бути як синтезованими, так і виділеними на фотознімках. До таких завдань ставляться, наприклад, моделювання "машинного зору" (роботи), аналіз рентгенівських знімків з виділенням і відстеженням об'єкта, що цікавить (внутрішнього органу), розробка систем відеоспостереження.

Когнітивна комп'ютерна графіка – тільки новий напрямок, що формується, поки ще недостатньо чітко визначений. Це – комп'ютерна графіка для наукових абстракцій, що сприяє народженню нового наукового знання. Технічною основою для неї є потужні ЕОМ і високопродуктивні засоби візуалізації.

Одним з найбільш ранніх прикладів використання когнітивної комп'ютерної графіки є робота Ч. Страуса "Несподіване застосування ЕОМ у чистій математиці" (ТІІЕР, т. 62, № 4, 1974, с. 96 – 99). У ній показане, як для аналізу складних алгебраїчних кривих використовується "n-мірна" дошка на основі графічного термінала. Користуючись пристроями введення, математик може легко одержувати геометричні зображення результатів спрямованої зміни параметрів досліджуваної залежності. Він може також легко управляти поточними значеннями параметрів, "поглиблюючи тим самим своє розуміння ролі варіацій цих параметрів". У результаті отримано "кілька нових теорем і визначені напрямки подальших досліджень".

У даному курсі передбачається розглянути наступні питання:

- подання зображення в комп'ютерній графіці;
- способи підготовки зображення до візуалізації;
- методи виводу зображення на екран;
- методи роботи із зображенням;
- методи обчислювальної геометрії.

1.2. Коротка історія

У цьому короткому історичному нарисі ми будемо згадувати алгоритми й методи, з якими читач зможе познайомитися в даному курсі. Ці попередні згадування не повинні бентежити при першому прочитанні. По завершенні курсу можна повернутися до цього розділу й пройти його заново.

Комп'ютерна графіка в початковий період свого виникнення була далеко не настільки ефектною, якою вона стала в даний момент. У ті роки комп'ютери перебували на ранній стадії розвитку й були здатні відтворювати тільки найпростіші контури (лінії). Ідея комп'ютерної графіки не відразу була підхоплена, але її можливості швидко росли, і поступово вона стала займати одну з найважливіших позицій в інформаційних технологіях.

Першою офіційно визнаною спробою використання дисплея для виводу зображення з ЕОМ з'явилося створення в Масачусетському технологічному університеті машини Whirlwind-I в 1950 м. Таким чином, виникнення комп'ютерної графіки можна віднести до 1950-х років. Сам же термін "комп'ютерна графіка" придумав в 1960 р. співробітник компанії Boeing У. Феттер.

Перше реальне застосування комп'ютерної графіки зв'язують із ім'ям Дж. Уїтні. Він займався кіновиробництвом в 60-х роках і вперше використовував комп'ютер для створення титрів до кінофільму.

Наступним кроком у своєму розвитку комп'ютерна графіка зобов'язана Айвену Сазерленду, що в 1961 р., ще будучи студентом, створив програму малювання, названу їм Sketchpad (альбом для малювання). Програма використовувала світлове перо для малювання найпростіших фігур на екрані. Отримані картинки можна було зберігати й відновлювати. У цій програмі було розширене коло основних графічних примітивів, зокрема, крім ліній і крапок був введений прямокутник, що задавався своїми розмірами й розташуванням.

Спочатку комп'ютерна графіка була векторної, тобто зображення формувалося з тонких ліній. Ця особливість була пов'язана з технічною реалізацією комп'ютерних дисплеїв. Надалі більше широке застосування одержала растрова графіка, заснована на поданні зображення на екрані у вигляді матриці однорідних елементів (пікселів).

У тім же 1961 р. студент Стів Рассел створив першу комп'ютерну відеогру Spacewar ("Зоряна війна"), а науковий співробітник Bell Labs Едвард Зеджек створив анімацію "Simulation of a two-giro gravity control system".

У зв'язку з успіхами в області комп'ютерної графіки великі корпорації почали проявляти до неї інтерес, що у свою чергу стимулювало прогрес в області її технічної підтримки.

Університет штату Юта стає центром досліджень в області комп'ютерної графіки завдяки Д.Евансу й А.Сазерленду, які в цей час були самими помітними фігурами в цій області. Пізніше їхнє коло стало швидко розширюватися. Учнем Сазерленда став Е.Кетмул, майбутній творець алгоритму видалення невидимих поверхонь із використанням Z-Буфера (1978). Тут же працювали Дж. Варнок, автор алгоритму видалення невидимих граней на основі розбивки області (1969) і засновник Adobe System (1982), Дж. Кларк, майбутній засновник компанії Silicon Graphics (1982). Всі ці дослідники дуже сильно просунули алгоритмічну сторону комп'ютерної графіки.

У тім же 1971 р. Гольдштейн і Нагель уперше реалізували метод трасування променів з використанням логічних операцій для формування тривимірних зображень.

В 1970-і роки відбувся різкий стрибок у розвитку обчислювальної техніки завдяки винаходу мікропроцесора, у результаті чого почалася мініатюризація комп'ютерів і швидкий ріст їхньої продуктивності. І в цей же час починає інтенсивно розвиватися індустрія комп'ютерних ігор. Одночасно комп'ютерна графіка починає широко використовуватися на телебаченні й у кіноіндустрії. Дж.Лукас створює відділення комп'ютерної графіки на Lucasfilm.

В 1977 р. з'являється новий журнал "Computer Graphics World".

У середині 1970-х років графіка продовжує розвиватися убік все більшої реалістичності зображень. Е.Кетмул в 1974 р. створює перші алгоритми текстурування криволінійних поверхонь. В 1975 р. з'являється згаданий раніше метод зафарбовування Фонга. В 1977 р. Дж.Млинець пропонує алгоритми реалістичного зображення шорсткуватих поверхонь (мікрорельєфів); Ф.Кроу розробляє методи усунення східчастого ефекту при зображенні контурів (антиелайзинг). Дж.Брезенхем створює ефективні алгоритми побудови растрових образів відрізків, окружностей і еліпсів. Рівень розвитку обчислювальної техніки до цього часу вже дозволив використовувати алгоритми, що вимагають більших обсягів пам'яті, і в 1978 р. Кетмул пропонує метод Z-Буфера, у якому використовується область пам'яті для зберігання інформації про «глибину» кожного пікселя екранного зображення. У цьому ж році Сайрус і Бек розвивають алгоритми клипірування (відсікання) ліній. А в 1979 р. Кей і Грінберг уперше реалізують зображення напівпрозорої поверхні.

В 1980 р. Т.Уіттед розробляє загальні принципи трасування променів, що включають відбиття, переломлення, затінення й методи антиелайзинга. В 1984 р. групою дослідників (Горел, Торренс, Гринберг і ін.) була запропонована модель випромінювальності, одночасно розвиваються методи прямокутного кліпування областей.

В 1980-і роки з'являється цілий ряд компаній, що займаються прикладними розробками в області комп'ютерної графіки. В 1982 р. Дж.Кларк створює Silicon Graphics, тоді ж виникає Ray Tracing Corporation, Adobe System, в 1986 р. компанія Pixar відокремлюється від Lukasfilm.

У ці роки комп'ютерна графіка вже міцно впроваджується в кіноіндустрію, розвиваються додатки до інженерних дисциплін. В 1990-е роки у зв'язку з виникненням мережі Internet у комп'ютерної графіки з'являється ще одна сфера додатка.

Тут перераховані далеко не всі серйозні кроки на шляху розвитку графіки, але більше докладне знайомство з її історією вимагає досить гарного подання про теорію й алгоритми цієї дисципліни, тому ми обмежуємося лише коротким оглядом. Неважко помітити, що пріоритет у розвитку даного напрямку в інформаційних технологіях досить міцно втримують американські дослідники. Але й у вітчизняній науці теж були свої розробки, серед яких можна назвати ряд технічних реалізацій дисплеїв, виконаних у різні роки:

1968, ОЦ АН СРСР, машина БЕСМ-6, імовірно, перший вітчизняний растровий дисплей з відеопам'яттю на магнітному барабані;

1972, Інститут автоматики і електрометрії (Іаіе), векторний дисплей "Символ";

1973, Іаіе, векторний дисплей "Дельта";

1977, Іаіе, векторний дисплей ЕПГ-400;

1982, Київ, НДІ периферійного встаткування, векторний дисплей СМ-7316, 4096 символів, дозвіл 2048?2048;

1979-1984, Інститут прикладної фізики, серія растрових кольорових напівтонових дисплеїв "Гама". Останні дисплеї даної серії мали таблицю кольорів, підтримували вікна, плавне масштабування.

Таким чином, у процесі розвитку комп'ютерної графіки можна виділити кілька етапів.

В 1970-і роки вона формувалася як наукова дисципліна. У цей час розроблялися основні методи й алгоритми: відсікання, растрове розгорнення графічних примітивів, зафарбування візерунками, реалістичне зображення просторових сцен (видалення невидимих ліній і граней, трасування променів, що випромінюють поверхні), моделювання освітленості.

В 1980-е графіка розвивається більше як прикладна дисципліна. Розробляються методи її застосування у всіляких областях людської діяльності.

В 1990-е роки методи комп'ютерної графіки стають основним засобом організації діалогу " людина-комп'ютер" і залишаються такими по теперішній час.

1.3. Технічні засоби підтримки комп'ютерної графіки

Розвиток комп'ютерної графіки багато в чому обумовлено розвитком технічних засобів її підтримки. Насамперед це пристрою виводу, якими є дисплеї. У цей час існує кілька типів дисплеїв, що використовують електронно-променеву трубку, а також дисплеї на рідкокристалічні індикаторах і інші їхні види. Нас цікавлять головним чином функціональні можливості дисплеїв, тому ми не будемо стосуватися їхнього внутрішнього пристрою й електронних схем.

Виникнення комп'ютерної графіки, як уже говорилося раніше, можна віднести до 50-х років. Дисплейна графіка на першому етапі свого розвитку використовувала електронно-променеві трубки (ЕПТ) **з довільним**

скануванням променів для виводу у вигляді зображення інформації з ЕОМ. З експерименту в Масачусетському технологічному інституті почався етап розвитку векторних дисплеїв (дисплеїв з довільним скануванням променів).

Найпростішим із пристроїв на ЕПТ є **дисплей на запам'ятовувальній трубці** із прямим копіюванням зображення. Запам'ятовувальна трубка має властивість тривалого часу післясвітлення: зображення залишається видимим протягом тривалого часу (до однієї години). При виводі зображення інтенсивність електронного променю збільшують до рівня, при якому відбувається запам'ятовування сліду променю на люмінофорі. Складність зображення практично не обмежена. Стирання відбувається шляхом подачі на всю трубку спеціальної напруги, при якому освітлення зникає, і ця процедура займає приблизно 0,5 с. Тому зображення, отримані на екрані, не можна стерти частково, а стало бути, динамічні зображення або анімація на такому дисплеї неможливі. Дисплей на запам'ятовувальній трубці є векторним, або дисплеєм з довільним скануванням, тобто він дозволяє провести відрізок з однієї адресуемої крапки в будь-яку іншу. Його досить легко програмувати, але рівень інтерактивності в нього нижче, ніж у ряду дисплеїв інших типів через низьку швидкість і погані характеристики видалення.

Наступний тип – це **векторні дисплеї з регенерацією зображення**. При переміщенні променів по екрані в крапці, на якій потрапив промінь, збуджується освітлення люмінофора екрана. Це освітлення досить швидко припиняється при переміщенні променю в іншу позицію (звичайний час післясвітлення - менш 0,1 с). Тому, для того щоб зображення було постійно видимим, доводиться його "перемальовувати" (регенерувати зображення) 50 або 25 разів у секунду. Необхідність регенерації зображення вимагає збереження його опису в спеціально виділеній пам'яті, називаною пам'яттю регенерації. Сам опис зображення називається дисплейним файлом. Зрозуміло, що такий дисплей вимагає досить швидкого процесора для обробки дисплейного файлу й керування переміщенням променю по екрані.

Звичайно серійні векторні дисплеї встигали 50 разів у секунду будувати тільки близько 3000-4000 відрізків. При більшому числі відрізків зображення починає мерехтіти, тому що відрізки, побудовані на початку чергового циклу, повністю гаснуть до того моменту, коли будуть будуватися останні.

Іншим недоліком векторних дисплеїв є мале число градацій по яскравості (звичайно від двох до чотирьох). Були розроблені, але не знайшли широкого застосування двох- і триколірні ЕПТ, що також забезпечували кілька градацій яскравості.

У векторних дисплеях легко стерти будь-який елемент зображення – досить при черговому циклі побудови видалити елемент, з дисплейного файлу.

Текстовий діалог підтримується за допомогою алфавітно-цифрової клавіатури. Непрямий графічний діалог, як і у всіх інших дисплеях, здійснюється переміщенням перехрестя (курсору) по екрані за допомогою тих або інших засобів керування перехрестям – координатних коліс, що керується джойстиком, трекболом, планшетом і т.д. Відмітною рисою векторних дисплеїв є можливість безпосереднього графічного діалогу, що полягає в простій вказівці за допомогою світлового пера об'єктів на екрані (ліній, символів і т.д.).

Векторні дисплеї звичайно підключаються до ЕОМ високошвидкісними каналами зв'язку. Перші серійні векторні дисплеї за кордоном з'явилися наприкінці 1960-х років.

Прогрес у технології мікроелектроніки привів до того, що із середини 1970-х років переважне поширення одержали дисплеї з растровим скануванням променями. Растровий пристрій можна розглядати як матрицю дискретних крапок (пікселів), кожна з яких може бути підсвічена. Таким чином, воно є точковим пристроєм, що малює. Тому будь-який зображуваний на екрані дисплея відрізок будується за допомогою послідовності крапок, що апроксимують ідеальну траєкторію відрізка, подібно тому, як можна будувати зображення по клітках на картковому листку паперу. При цьому відрізок виходить прямим тільки у випадках, коли він горизонтальний, вертикальний або спрямований під кутом 45° до горизонталі. Всі інші відрізки виглядають як послідовність "сходів" (східчастий ефект).

При побудові зображення в растрових графічних пристроях використовується буфер кадру, що представляє собою велику безперервну ділянку пам'яті комп'ютера. Для кожної крапки в растрі приділяється як мінімум один біт пам'яті. Буфер кадру сам по собі не є пристроєм виводу, він лише використовується для зберігання малюнка. Найбільше часто як пристрій виводу, використовуваного з буфером кадру, виступає відеомонітор.

Щоб зрозуміти принципи роботи растрових дисплеїв, ми розглянемо загалом пристрій кольорової растрової електронно-променевої трубки. Зображення на екрані виходить за допомогою сфальцьованого електронного променю, що, потрапляючи на екран, покритий люмінофором, дає яскраву колірну гаммк. Промінь у растровому дисплеї може відхилятися тільки в строго певні позиції на екрані, що утворять своєрідну мозаїку. Люмінофорне покриття теж не безупинне, а являє собою безліч близько розташованих дрібних крапок, куди може позиціонуватися промінь. Дисплей, що формує чорно-білі зображення, має одну електронну гармату, і її промінь висвітлює однотонні колірні плями. У кольоровий ЕПТ перебувають три електронних гармати, по однієї на кожний основний колір: червоний, зелений і синій. Електронні пушки часто об'єднані в трикутний блок, що відповідає трикутним блокам червоного, зеленого й синього люмінофорів на екрані. Електронні промені від кожної з гармат, проходячи через спеціальну тіньову маску, попадають точно на пляму свого люмінофора. Зміна інтенсивності кожного із трьох променів дозволяє одержати не тільки три основних кольори, але й кольори, одержувані при їхньому змішанні в різних пропорціях, що дає дуже велику кількість відтінків для кожного пікселя екрана.

Дисплеї на **рідкокристалічних індикаторах** працюють аналогічно індикаторам в електронних годинниках, але, звичайно, зображення складається не з декількох великих сегментів, а з великого числа окремо керованих крапок. Ці дисплеї мають найменші габарити й енергоспоживання, тому широко використовуються в портативних комп'ютерах. Вони мають як переваги, так і недоліки в порівнянні з дисплеями на ЕПТ. Хоча історично такий спосіб виводу зображення з'явився раніше, ніж растровий дисплей з ЕПТ, але швидко розвиватися він почав значно пізніше. Ці дисплеї також є растровими

пристроями (їх теж можна представити як матрицю елементів – рідких кристалів).

Існують і інші види дисплеїв, наприклад плазмова панель, але ми не будемо їх стосуватися, оскільки вони також є растровими, а технічна реалізація не є предметом нашого курсу. Важливо те, що розглянуті нами алгоритми розроблені для растрових графічних дисплеїв, а загальні принципи роботи цих пристроїв нам зрозумілі.

Крім дисплеїв, як пристрої виводу зображень використовуються плотери (графопобудовувачі), призначені для виводу графічної інформації на папір. Ранні графічні пакети були орієнтовані саме на модель пір'яного плотера, що формує зображення за допомогою пера. Перо може переміщатися уздовж двох напрямних, відповідним двом координатним осям, причому воно може перебувати у двох станах – піднятому й опущеному. У піднятому стані воно просто переміщається над поверхнею паперу, а в опущеному залишає на папері лінії, що формують зображення. Таким чином, плотер ближче до векторних дисплеїв, але відрізняється від них тим, що стирати виведені зображення неможливо. Тому для них зображення спочатку повністю формується в пам'яті комп'ютера, а потім виводиться.

Крім того, варто згадати принтери, що виводять зображення на папір або плівку. Зображення, одержуване за допомогою сучасних принтерів, також формується як крапкове (растрове), але, як правило, із кращим дозволом, чим екранне. Як і у випадку із графопобудовувачі, стерти зображення або його частину неможливо.

Тепер зробимо невеликий огляд пристроїв введення інформації, що дозволяють вирішувати різні завдання комп'ютерної графіки, не вдаючись у деталі фізичних принципів їхньої роботи. Ці пристрої дозволяють організувати діалог "людин-комп'ютер", а особливості конструкції кожного пристрою дозволяють йому спеціалізуватися на виконанні певного кола завдань. Нас вони цікавлять саме як логічні пристрої, тобто з погляду виконуваних ними функцій.

Першу групу пристроїв, за допомогою яких користувач може вказати позицію на екрані, назвемо **пристроями вказівки** (**pointing device**): миша, трекбол (trackball), світлове перо (lightpen), джойстик (joystick), спейсбол (spaceball). Практично всі пристрої цієї групи оснащені парою або декількома кнопками, які дозволяють сформувати й передати в комп'ютер які-небудь сигнали або переривання.

Рис. 1.1. Миша

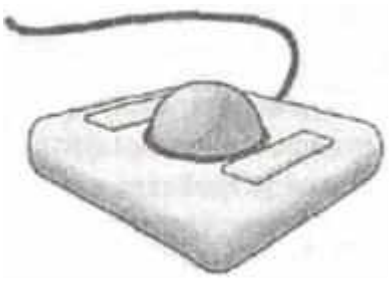


Рис. 1.2. Трекбол

Миша (рис. 1.1) і трекбол (рис. 1.2) схожі не тільки по призначенню, але часто й по конструкції. У механічній миші й трекболі обертання кульки перетвориться за допомогою пари перетворювачів у сигнали, передані в комп'ютер. Перетворювачі вимірюють обертання щодо двох взаємно перпендикулярних осей. Існує дуже багато модифікацій пристроїв цих груп. В оптичній миші використовуються не механічні, а оптичні чутливі елементи для виміру переміщення: виміряється відстань шляхом підрахунку штрихів на спеціальній підложці. Маленькі трекболи широко застосовуються в портативних комп'ютерах, де їх вбудовують прямо в клавіатуру.

У деякі клавіатури вбудовуються прилади, чутливі до тиску, які виконують ті ж функції, що й миша або трекбол, але при цьому в них відсутні рухливі елементи. Перетворювачі в таких пристроях вимірюють величину тиску на невеликий опуклий набалдашник, розміщений між двома кнопками в середній частині клавіатури. Вони, як і трекбол, використовуються переважно в портативних комп'ютерах.

Вихідні сигнали миші або трекбола можна розглядати як дві незалежні величини й перетворювати їх у координати положення на двовимірній площині екрана або в якій-небудь іншій системі координат. Лічені із пристрою значення можна відразу ж використовувати для керування спеціальною оцінкою (курсором) на екрані.

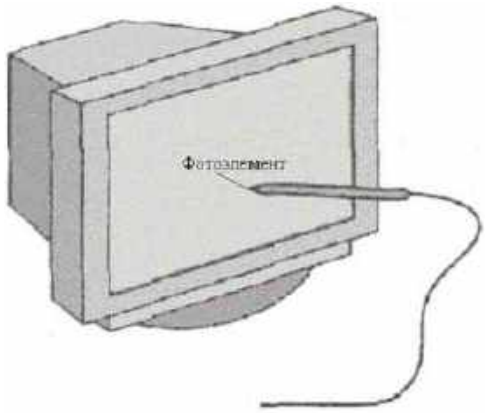


Рис. 1.3. Світлове перо

Ветераном серед пристроїв введення в комп'ютерній графіці є пристрій, назване при його створенні світловим пером. Уперше воно з'явилося у вже згаданому проекті А. Сазерленда Sketchpad. Світлове перо містить фоточутливий елемент (рис. 1.3), що при наближенні до екрана сприймає випромінювання, породжуване при зіткненні електронів з люмінофорним покриттям екрана. Якщо потужність світлового імпульсу перевищує певний поріг, фоточутливий елемент формує імпульс, що передається в комп'ютер. Аналізуючи зсув за часом цього імпульсу відносно початку циклу регенерації, комп'ютер може точно визначити координати тої крапки екрана, порушення якої "висвітило" фотоелемент. Таким чином, у розпорядженні користувача виявляється пристрій безпосередньої вказівки, що працює прямо із

зображенням на екрані. У цей час цей пристрій уже практично вийшло із уживання: воно витиснуто більше простим й надійним – мишею.

Ще один пристрій, що досить активно використовується в мультимедійних додатках, а також у різного роду комп'ютерних тренажерах – джойстик (рис. 1.4). Переміщення джойстика у двох взаємно перпендикулярних напрямках сприймається перетворювачами, інтерпретується як вектор швидкості, а отримані значення використовуються для керування положенням маркера на екрані. Обробка сигналу виконується таким чином, що нерухливий джойстик у якому- або проміжному положенні не змінює положення маркера, а чим далі джойстик відхилений від початкового положення, тим швидше маркер переміщається по екрані. Таким чином, джойстик відіграє роль пристрою введення зі змінною чутливістю. Інша перевага джойстика – наявність силового зворотного зв'язка, забезпеченої наявністю різного роду пружин. При цьому користувач почуває, що чим далі відхиллено джойстик, тим більше зусилля потрібно для його подальшого руху. Це саме ті властивості, які потрібні при роботі з різного роду симуляторами, а також у комп'ютерних іграх.

Рис. 1.4. Джойстик

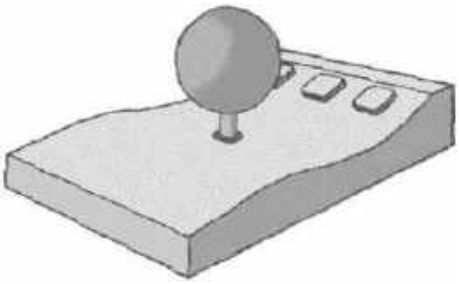


Рис. 1.5. Спейсбол

Спейсбол – це "тривимірне" пристрій введення. Хоча й існують різні конструкції таких пристроїв, вони усе ще не одержали широкого поширення, оскільки програють популярним двовимірним пристроям як за вартістю, так і по технічних характеристиках. Спейсбол схожий на джойстик, але відрізняється від нього тим, що він має вигляд закріпленого на рукоятці кулі, причому рукоятка в цій конструкції нерухлива (рис. 1.5). Куля має датчики тиску, які вимірюють зусилля, що прикладається користувачем. Куля може вимірювати не тільки складові зусилля в трьох основних напрямках (зверху долілиць, від себе або на себе, уліво-вправо), але й обертання щодо трьох осей. Таким чином, цей пристрій здатний передавати в комп'ютер шість незалежних параметрів (тобто має шість ступенів волі), що характеризують як поступальний рух, так і обертання.

Існують і інші тривимірні системи виміру й введення, що використовують найсучасніші технології, наприклад лазерні. У системах віртуальної реальності використовуються більше складні пристрої, що дозволяють динамічно відслідковувати положення й орієнтацію користувача. Для додатків, пов'язаних із сучасною робототехнікою й моделюванням віртуальної реальності, іноді потрібні пристрої, що володіють ще більшим числом ступенів волі, чим спейсбол. Останнім часом з'явилися нові розробки в цьому напрямку, зокрема -

рукавички із системою датчиків, які здатні вловлювати руху окремих частин руки людини (рис. 1.6).

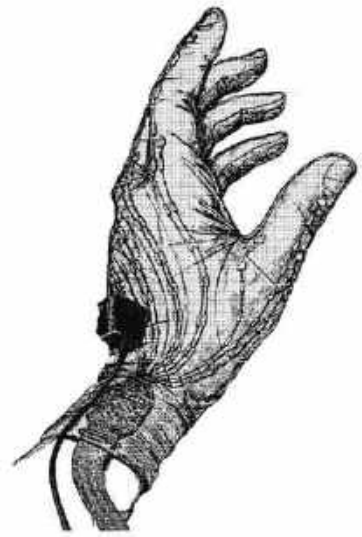


Рис. 1.6. Рукавичка для введення даних

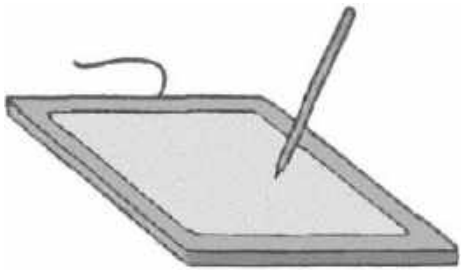


Рис. 1.7. Планшет

При використанні миші або трекбола аналізується відносне положення пристрою. Якщо перемістити покажчик на екрані яким-небудь способом в інше місце, не обертаючи при цьому кулька миші або трекбола, то подальші сигнали будуть зміщати покажчик щодо нової позиції. Можна також акуратно перемістити мишу без обертання кульки й це не приведе до переміщення курсору на екрані. Абсолютні координати пристрою не зчитуються обробною програмою. Але при уведенні в комп'ютер графіків прикладній програмі найчастіше потрібні абсолютні координати пристрою введення. Таку можливість забезпечують різного роду планшети (рис. 1.7). У планшеті застосовується, як правило, ортогональна сітка проводів, розташована під його поверхнею. Положення пера визначається через електромагнітну взаємодію сигналів, що проходять від проводів до щупа. Іноді як планшет використовуються чутливі до дотику прозорі екрани, які наносяться на поверхню ЕПТ. Невеликі екрани такого типу розміщаються іноді на клавіатурі портативних комп'ютерів. Чутливі панелі можна використовувати в режимах як абсолютних, так і відносних координат.

Для растрового введення зображень використовуються сканери, що дозволяють не тільки ввести образ у комп'ютер, але й зробити їхню обробку й документування. Одна з важливих областей застосування сканерів – введення текстів. При цьому обробка уведеного зображення виконується програмним забезпеченням розпізнавання текстів, що у цей час стало вже досить розвиненим. У САПР сканери використовуються для автоматизації введення раніше підготовленої конструкторської документації. У цьому випадку проблема полягає в тім, що дані від сканера представлені в растровій, а не векторній формі, і потрібне виконання зворотного перетворення «растр-вектор». Це завдання дуже складне: необхідно розпізнавати різні зображення й тексти, у тому числі рукописні, ураховувати, що лінія може при скануванні не

тільки одержати різну ширину на різних ділянках, але й виявитися розірваної й т.д. Для рішення цього завдання засобів однієї лише комп'ютерної графіки недостатньо: необхідний залучення й інші дисципліни.

Всі перераховані вище пристрої введення з погляду передачі інформації прикладним програмам варто розглядати як логічні. Функціонування систем введення характеризується тим, яку інформацію пристрій передає в програму, коли і як воно передає цю інформацію. Ці питання стають особливо істотними при розробці користувальницького інтерфейсу.

1.4. Питання й вправи

1. Назвіть чотири основні області застосування комп'ютерної графіки.
2. Які основні напрямки розвитку комп'ютерної графіки? Які завдання вони вирішують?
3. Де й коли вперше був використаний дисплей як пристрій виводу ЕОМ?
4. Ким і коли була розроблена перша інтерактивна програма для малювання?
5. Назвіть основних розроблювачів методів зафарбовування гладких поверхонь.
6. Хто є автором ряду алгоритмів побудови растрових образів різних геометричних об'єктів?
7. Назвіть авторів алгоритмів видалення невидимих ліній.
8. У чому складається основне розходження між дисплеями з довільним скануванням і растровим скануванням?
9. Чим відрізняється дисплей на запам'ятовувальній трубці від векторного дисплея з регенерацією зображення?
10. Які основні принципи роботи кольорової растрової електронно-променевої трубки?
11. Як працює пір'яний плотер?
12. Назвіть основні пристрої введення, що використовуються в комп'ютерній графіці.
13. Які із пристроїв введення дають можливість працювати в абсолютних координатах?
14. Перелічите області застосування сканерів.

Тема 2. СУЧАСНІ АПАРАТНІ ЗАСОБИ РАСТРОВОЇ ГРАФІКИ

Основні поняття. Пристрою введення: сканери, цифрові фотоапарати й відеокамери. Пристрою виводу: дисплеї на ЕПТ, рідкокристалічні дисплеї й інші типи дисплеїв; Проектори; Принтери. Архітектура графічної підсистеми ПК: архітектура, подання зображень, програмний інтерфейс

У цьому розділі дається короткий огляд основних пристроїв, що використовують растрове подання для відображення двовимірної інформації. Як приклад реалізації апаратних засобів для роботи з растровою графічною інформацією розглянута архітектура сучасних комп'ютерів типу ПК (англ. PC – Personal Computer[1)]).

2.1. Основні поняття

У попередньому розділі було розказано про растр як про абстрактний математичний об'єкт. При фізичній реалізації растрів виникають додаткові характеристики. Основний з них є **Роздільна здатність** (англ. resolution) – кількість крапок фізичного растра в одиниці довжини. Найчастіше це

– ppi – пікселів на дюйм (англ. pixels per inch[2)]).

– dpi – крапок на дюйм (англ. dots per inch). Правильно вживати стосовно принтерів, хоча широко використовується скрізь замість ppi.

– spi – проб на дюйм. (англ. samples per inch) Іноді вживається стосовно сканерів замість dpi.

У більшості випадків пікселі растра розташовані рівномірно по ширині й висоті, і для таких растрів роздільна здатність однакова по ширині й висоті. Якщо це не так (для деяких принтерів і сканерів зокрема), то в цьому випадку звичайно пишуть «1200×2400 dpi», де 1200 – роздільна здатність по горизонталі, а 2400 – по вертикалі. Також термін дозвіл уживається в іншому змісті – як розмір растра в пікселях. Наприклад, "розширення дорівнює 1280×1024".

З погляду подання кольори важливі два параметри:

– **Колірна гама**, це поняття було уведено раніше;

– **Динамічний діапазон** (англ. dynamic range) – діапазон рівнів яскравості для даного пристрою. Це поняття прийшло із традиційної (плівкової) фото- і відеозйомки, у якій було однієї з важливих характеристик плівки. Важливо відзначити, що динамічний діапазон залежить як від характеристик пристрою відображення, так і від способу подання графічної інформації. Наприклад, типове подання кожного з компонентів кольору в RGB-Моделі 8 бітами, що дає всього 256 можливих рівнів яскравості.

Досить широкою колірною гамою володіють плазмові дисплеї й фотографічна плівка, а самий вузької – монохромний пристрої (на діаграмі кольоровості відповідна область вироджується в крапку) (див. рис. 1.9).

Для подання графічної інформації в пам'яті використовується термін **Глибина кольору** (англ. color depth) – кількість інформації для подання одного кольору. Виміряється в bpp – bits per pixel (англ. біт на піксель). Типові значення: 1 (монохромне зображення), 8, 16, 24, 48.

Більш докладно це поняття розглядається в розділі 2.4.

2.2. Пристрої введення

2.2.1. Сканери

Сканер – пристрій одержання зображень високого розширення (до 11000 spi). Принцип роботи складається в послідовному висвітленні скануємого матеріалу ксеноновою або флуоресцентною лампою й реєстрації відбитого кольору ПЗС[3)](англ. CCD) матрицею (за винятком барабанних сканерів, де використовується фотопомножувач). Якщо матеріал є напівпрозорим (наприклад, фотоплівка), то він, навпаки, підсвітлюється зі зворотної до сенсора сторони.

По типі сканування такі пристрої класифікуються в такий спосіб.

– **Барабанні сканери** ставляться до класу high-end. Скануємий об'єкт (найчастіше фотоплівка) спочатку вимочується в спеціальному розчині, а потім міститься на барабан, що обертається перед переміщається в одному напрямку фотоелектронним помножувачем. Ці пристрої набагато більше чутливі, чим Пзс-Матриці, тому й роздільна здатність у них значно вище – від 4000 до 11000 spi. Глибина кольору – 48 біт/піксель. Такі сканери мають порівняно великі габарити.

– **Планшетні сканери** – переважний на даний момент тип сканерів. Під склом одного зі стандартних розмірів (найчастіше А4; рідше А3) перебуває скануюча головка, що послідовно проходить всю площа під склом у процесі сканування. Для сканування фотоплівки в кришку може бути вбудована лампа підсвічування (що й показано на рис. 2.1). Типові розширення – 600-3200 spi. Глибина кольору – до 48 біт/піксель.

– **Протяжні сканери** призначені для сканування аркушів паперу заданої ширини. Скануюча головка переміщається тільки в одному напрямку, а протяжний механізм забезпечує послідовний зсув паперу відносно скануючої головки уздовж іншого виміру. Такі пристрої компактніші планшетних сканерів і дозволяють сканувати більше протяжні в одному вимірі матеріали.

– **Ручні сканери**. Скануюча головка переміщається над скануємою поверхнею за допомогою руки людини. Відповідно, для даного типу сканерів потрібно більше складне програмне забезпечення, що забезпечувало б правильне сполучення окремих фрагментів (тому що траєкторія переміщення руки не ідеальна). До цього типу ставляться самі компактні сканери у формі авторучки. У зв'язку з порівняльною трудомісткістю й складністю обробки результатів останнім часом інтерес до них незначний.

– **Слайд-Сканери** схожі на планшетні, але призначені спеціально для сканування фотоплівки. Важливою відмінністю від перших є відсутність скла між сенсором і плівкою. Це важливо, тому що стекло є додатковим проміжним середовищем, що переломлює світлові промені й, відповідно, що вносить певні перекручування, помітні при скануванні з високим дозволом.

Рис. 2.1. Планшетний сканер

2.2.2. Цифрові фотоапарати й відеокамери

Рис. 2.2. Цифровий фотоапарат

Цифрові фотоапарати й відеокамери аналогічні по принципах дії (система лінз, що проектує світло, що попадає в об'єктив, на невелику плоску площадку) традиційним аналоговим фото- і відеокамерам, відповідно. Основне розходження полягає в тому, що в аналогових пристроях на цій площадці перебуває кадр світлочутливої плівки, а в цифрових пристроях – світлочутлива матриця ПЗЗ (англ. CCD) або КМОП (англ. CMOS) сенсорів. Тому що ці сенсори чутливі тільки до яскравості, то для одержання кольорового зображення застосовують світлофільтри. Існує два підходи.

– Перший варіант. Для одержання одного зображення інформація знімається з матриці сенсора 3 рази підряд, при цьому щораз використовується світлофільтр для одного з базисних кольорів RGB. Цей процес порівняно повільний й складний, тому така технологія застосовується тільки в high-end цифрових фотокамерах при зйомці нерухливих об'єктів.

– Другий варіант. За допомогою призми світловий потік розкладається на три, відповідним базисним RGB-кольором. Кожний із цих потоків направляється на свою Пзз-Матрицю. Тому ця технологія одержала назву 3CCD. Вона

використовується у відеокамерах верхнього цінового діапазону. Для фотографії вона поки занадто дорога.

Мікросвітлофільтри

Кожний елемент ПЗЗ матриці має свій світлофільтр, що відповідає одному з базисних RGB кольорів. Звичайно у квадраті 2×2 розташовуються 2 зелених, 1 синій і 1 червоний елемент, т.зв. **маска Байера** (англ. Bayer mask), см. рис. 2.3. Для одержання звичайного зображення використовуються різні алгоритми інтерполяції. Це найпоширеніша технологія як для фото-, так і для відеокамер.

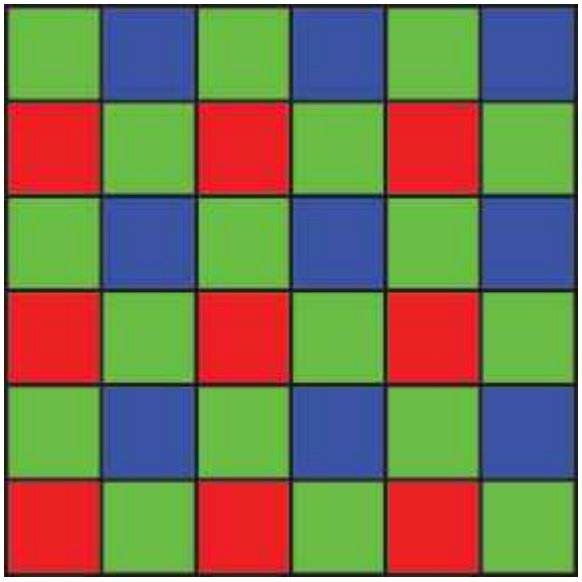


Рис. 2.3. Маска Байера.

Відеокамери нижнього цінового діапазону знімають відео із **міжстроковим розгорненням**, тобто кадр складається із двох послідовно знятих напівкадрів, що складаються тільки з парних і непарних рядків зображення, відповідно. При швидкому переміщенні об'єктів, що знімаються, або самої камери через помітну різницю в напівкадрах виникають неприємні артефакти. Для рішення цієї проблеми при наступній обробці знятого відео виробляється **деінтерлейсинг** (англ. deinterlacing).

Поняття роздільної здатності для таких пристроїв звичайно не застосовується[4].Часто вживається поняття **мегапіксель** (Мп), рівне загальній кількості пікселів на матриці сенсора, діленому на 1024^2. Розрішення, як розмір растра, варіюється від 640-480 (0,29 Мп) до 7216-5412 (39 Мп) для фотокамер і від 320-240 (0,07 Мп) до 7680-4320 (33 Мп) – для відеокамер.

Глибина кольору в більшості фото- і відеокамер – стандартні 24 біта/піксель. У професійних моделей вона більше – до 48 біт/піксель.

Аспектове співвідношення у фотокамер – 4:3 (як у стандартних дисплеїв) або 3:2 (відповідає плівці). У відеокамер – 4:3 або 16:9 (т.зв. **широкоекранне** відео).

2.3. Пристрою виводу

2.3.1. Дисплеї

Дисплеї є найпоширенішим типом пристроїв виводу графічної інформації. Вони застосовуються скрізь – від мобільних телефонів до рекламних щитів.

2.3.1.1. Дисплеї на ЕПТ

Найпоширеніший[5)] тип дисплеїв середнього розміру. Заснований на використанні ЕПТ – Електронно-Променевої Трубки (англ. CRT). ЕПТ улаштована в такий спосіб.

Електрони спочатку прискорюються електромагнітним полем, а потім відхиляються в потрібному напрямку перпендикулярним полем (ця частина ЕПТ називається **гарматою**). При влученні на поверхню екрана вони викликають світіння встановлених там часток люмінофора, що і сприймається спостерігачем. Чим більше прискорення (а отже, більшу енергію) одержав електрон, тим яскравіше світиться люмінофор. У кольорових дисплеїв є 3 пушки для кожного з базисних RGB-Кольорів, а по поверхні екрана рівномірно розподілені частки люмінофора, що відповідають базисним кольорам. Щоб електрони, випущені з відповідної гармати, потрапили тільки на люмінофор свого кольору перед екраном ставиться **щілинна маска** або **апертурні ґрати.** Час післясвітлення люмінофора незначний, і зображення постійно обновляється; для виводу одного кадру гармата випускає електрони послідовно ліворуч праворуч у кожному рядку й по рядках знизу нагору. Кількість відновлень усього зображення в секунду зветься **частота відновлення** або **частота розгорнення** (англ. refresh rate). Вона повинна бути досить високої (> 75 Гц), щоб не виникало стомлюючого ока мерехтіння.

Рис. 2.4. Дисплей на ЕПТ

Проблемою моніторів на ЕПТ є те, що яскравість світіння люмінофора залежить від енергії електрона, що потрапив на нього (у свою чергу обумовленої напругою в пушку Vs) не лінійно, $I \sim V_s^{\gamma}$ а за законом. Для більшості ЕПТ $\gamma \sim 2,2$. Для того щоб компенсувати цей ефект, застосовують

т.зв. **гамма-корекцію**, тобто подачу на гармату скоректованої напруги $V_c = V_s^{(1/\gamma)}$.

Гамма-корекція може також здійснюватися програмно, що відповідає зміною кольору пікселів. Для кольорових ЕПТ дисплеїв, загалом кажучи, γ своя для кожного з базисних RGB-Кольорів, але в простих системах це не враховують.

Основними недоліками дисплеїв на ЕПТ є порівняно більші розмір і вага, а також геометричні перекручування на периферії екрана. Втім, останній недолік у сучасних пристроях усунутий. Основні переваги – гарна передача кольору й здатність працювати в широкому діапазоні розширення екрана, на відміну від Рк-Дисплеїв.

Дозвіл залежить від щільності часток люмінофора на поверхні дисплея. Типові значення – 85-130 ppi. Практично всі ЕПТ-Дисплеї мають аспектові відношення 4:3.

2.3.1.2. Рідкокристалічні дисплеї

Рис. 2.5. Рідкокристалічний дисплей

У цей час займають домінуючу позицію (у порівнянні з ЕПТ) як дисплеї персональних комп'ютерів. Єдиний вид дисплеїв, використовуваний у ноутбуках на справжній момент. Улаштовано вони в такий спосіб:

Позад дисплея вбудована лампа, світло від якої проходить або не проходить через екран. Екран рідкокристалічного дисплея складається з 5 шарів: із двох сторін шари поляризаційних фільтрів і електродів, а всередині – шар рідких кристалів. Для кожного пікселя шар рідких кристалів складається з декількох молекул у ряд. При відсутності напруги цей ряд має форму спіралі й світло повністю проходить через зовнішній фільтр (тобто піксель світиться). При подачі напруги на електроди молекули розпрямляються в рівний ряд і світло йде перпендикулярно зовнішньому фільтру й не проходить через нього (тобто піксель – темний). Величина напруги дозволяє регулювати яскравість. Кольорове зображення формується, як і у фотоапаратах, за допомогою мікросвітлофільтрів.

Основними перевагами рідкокристалічних дисплеїв є менші, чим в Епт-Дисплеїв, розмір у глибину, вагу й енергоспоживання, більша чіткість. Крім цього відсутнє мерехтіння зображення, що спостерігається в Епт-Дисплеїв, що приводить до меншого стомлення очей. Недоліками є: гірша, чим в Епт-Дисплеїв, передача кольору; колірні перекручування при косих кутах зору; великий середній час реакції (час перемикання з одного кольору на інший, типове значення – 25 мс, тобто максимум 40 кадр/с), що приводить до "змазування" динамічно мінливих зображень (насамперед у відео й комп'ютерних іграх), а також недостатньо темний чорний колір (тому що не все світло вдається блокувати кристалом).

Типове розширення таке ж, як і в Епт-Дисплеїв, – 85-130 ppi. Аспектові відношення – 4:3 або 16:10 (т.зв. **широкоекранні дисплеї**).

2.3.1.4. Інші типи дисплеїв

Також існують менш розповсюджені типи дисплеїв, у тому числі й ті, які тільки з'являються на ринку.

– **Плазмені панелі**. У плазмових панелях, подібно Рк-Панелям, екран складається з декількох шарів; так само, як і в Рк-Панелей, із двох сторін підведені електроди, тільки між ними перебувають уже не рідкі кристали, а суміш інертних газів неону й ксенону. При подачі напруги на електроди через суміш газів починає проходити струм, що приводить до випущення іонів, які, як і в Епт-Дисплеях, потрапляючи на частки, що перебувають у верхньому шарі, люмінофора викликають його світіння. Тому що в плазмених дисплеях використовується люмінофор, подібний тому, що використовується в Епт-Дисплеях, те й колірні гами в них близькі. Основні переваги: вони мають малу глибину (порядку 10) і в той же час легко можна одержати дисплей великого розміру; крім цього вони мають відмінну яскравість. До недоліків варто віднести високу ціну й велике енергоспоживання, порівнянне з ЕПТ для однакової площі екрана.

– **Дисплеї на світлодіодах**. Світловипромінюючий діод (англ. LED – Light Emission Diode) – це напівпровідниковий діод, що володіє додатковою властивістю випущення фотонів певного кольору при проходженні через нього електричного струму. Для побудови кольорового дисплея для кожного пікселя береться три **світлодіоди** – з відповідними червоними, зеленому й синьому кольорами випромінювання. Безліч таких трійок, розташованих на прямокутній сітці, і утворять екран. Використовуються в основному для більших зовнішніх дисплеїв (реклама й т.п.).

– **Дисплеї на органічних світлодіодах**. Органічні **світлодіоди** (англ. OLED – Organic LED) – це **світлодіоди**, при виробництві яких використовуються органічні матеріали, зокрема полімери, які мають властивість гнучкості, що дозволяє робити гнучкі дисплеї. Також їхньою перевагою є те, що їх можна робити шляхом процесу, що нагадує струменений друк, тобто порівняно дешево. У цей час вони використовуються в основному в портативних пристроях, таких як МРЗ-плейери, але можливо знайдуть більше широке застосування в майбутньому.

– **"Електронний папір"**. Гнучкі дисплеї, здатні замінити звичайний папір. Від інших типів відрізняються тим, що розраховані не на постійне відновлення зображення, а навпаки, на його тривале збереження без електричної енергії. Це є одним з основних вимог. Розроблено кілька технологій, що задовольняють цим вимогам, на даний момент вони перебувають у стадії прототипів.

Існують ще деякі перспективні технології, які потенційно можуть служити для виробництва дисплеїв, але вони залишаються за рамками даної книги через свою незрілість.

2.3.2. Проектори

Проектори використовуються для демонстрації зображень більших розмірів. Для цього застосовуються системи лінз, що проектують маленьке зображення на великий екран. За технологією побудови первинного зображення усередині проектора діляться на:

– **Проектори на ЕПТ**. Ця технологія розглянута вище в розділі про ЕПТ-Дисплеї.

– **Проектори на РК**. Ця технологія розглянута вище в розділі про Рк-Дисплеї.

– **Проектори на технології DLP**. DLP – Digital Light Processing (англ. Цифрова Обробка Світла) – фірмова технологія компанії Texas Instruments. Для створення первинного зображення в таких проекторах використовується лампа, що висвітлює систему мікродзеркал (по одному на піксель). При постійній швидкій зміні положення цих дзеркал від повного пропущення світла до повного його блокування виходять відтінки сірого. Для одержання кольорового зображення використовують два методи: або обертове колірне коло з базисними RGB-Кольорами (при проходженні відповідного світлофільтра електромеханіка підбудовує дзеркала для відповідного колірного каналу), або біле світло спочатку розкладається призмою, потім потік для кожної RGB-Компонента проходять через свою систему дзеркал, і потім вони знову з'єднуються.

Слід зазначити, що для проекторів потрібні внутрішні дисплеї підвищеної яскравості, тому що площа зображення при проектуванні значно збільшується.

Типовий дозвіл – 1024-768, а максимальне досягає 2048-1080 для деяких DLP-Моделей. Типове аспектове відношення – 4:3, хоча є й різні широкоекранні варіанти.

2.3.3 .Принтери

Використання принтерів є найпоширенішим способом виводу двовимірної інформації на папір, та й не тільки на неї[6].

Варто помітити, що багато принтерів самі по собі можуть мати досить складні вбудовані Процесори Растрових Зображень (англ. RIP - Raster Image Processor), для формування растрових зображень. Найчастіше такий процесор може растеризувати зображення, описане програмою мовою PostScript [11] або PCL[7],які були спеціально створені для цих цілей. Проте останнім часом можливість апаратної растеризації мало використовується, і більшість

зображень приходить у принтер у вигляді готових растрів, тому що програмна растеризація – це більше гнучкий підхід[8)].

2.3.3.1. Матричні принтери

Самою низькоякісною технологією друку володіють матричні принтери. Друкуюча головка, що переміщається в одному вимірі по ширині сторінки, складається з декількох голок (звичайно 9 або 24). Фарба нанесена на стрічку, що перебуває між папером і головкою. При ударі голки по папері на ній залишається слід від фарби зі стрічки, у такий спосіб і виходить зображення. Були спроби створення кольорових матричних принтерів, але якість була неприйнятною. Вивід нетекстової інформації прийнятної якості важко доступний.

Роздільна здатність – від 60-72 до 360-360 dpi.

Основна перевага – дешевий друк, основні недоліки – низька якість і високий рівень шуму. Зараз практично витиснуті струминевими й лазерними принтерами, за винятком застосування в касових апаратах.

2.3.3.2. Струминеві принтери

Розташовуються в середньому ціновому діапазоні і є найпоширенішим рішенням для кольорового друку (тому що кольорові лазерні принтери поки ще досить дороги). Принцип дії наступний:

Друкуюча головка, що переміщається по ширині паперу, складається з безлічі мікрокамер з мікросоплами, при пропущенні електричного імпульсу через мікрокамеру в ній утвориться міхур, що виштовхує із сопла краплю фарби на папір. У кольорових принтерах (а це практично все струминеві моделі на даний момент) застосовується колірна модель CMYK (див. розділ 1.2). При цьому кожному із цих базисних кольорів відповідає свій резервуар з фарбою. Чорний колір уводиться для економії кольорових фарб. У дорогих моделях застосовується й більша кількість фарб для поліпшення передачі кольору.

Роздільна здатність – від 600-300 до 5760-1440 dpi.

Основна перевага – порівняльна дешевизна пристрою, основний недолік - дорожнеча видаткових матеріалів.

2.3.3.3. Лазерні принтери

Лазерні принтери є найефективнішими з погляду вартості друку сторінки. У цей час переважають у чорно-білому друці.

Принцип дії наступний:

Спочатку на всю поверхню барабана з фотопровідним покриттям наноситься позитивний заряд, звичайно за допомогою коронуючого електрода. Потім деякі ділянки барабана висвітлюються лазерним променем, що приводить до зняття заряду в цих місцях. Потім поверхня барабана проходить через порошкоподібний **тонер**, позитивно заряджені часточки якого відштовхуються від заряджених ділянок барабана й прилипають до незарядженого. Після цього тонер з барабана переноситься на папір, що для цього попередньо заряджається негативно за допомогою іншого коронуючого

електрода. Папір потім піддається нагріванню, при якому часточки тонера міцно приплавляють до неї.

Роздільна здатність – від 300 до 1200 dpi.

Основні переваги – висока чіткість і швидкість друку, основний недолік – порівняльна дорожнеча самих пристроїв (особливо кольорових).

2.4. Архітектура графічної підсистеми ПК

2.4.1. Архітектура

За вивід графічної інформації на дисплей у ПК відповідає спеціальний набір мікросхем, поміщається звичайно на окрему плату, що називається **відеоплатою** або **відеокартою**. Основним завданням є перетворення образа екрана, що перебуває в пам'яті, т.зв. **кадрового буфера** (англ. frame buffer), у набір сигналів, зрозумілих дисплею. Для підключення дисплеїв до комп'ютерів використовуються стандарти, що встановлюють логічні й фізичні параметри з'єднання. У цей час два найпоширеніших з них – це аналоговий VGA і цифровий DVI. Перший використовується для підключення як аналогових по суті дисплеїв на ЕПТ, так і цифрових рк-дисплеїв (аналого-цифрове перетворення в цьому випадку відбувається в самому дисплеї), другий – винятково для Рк-Дисплеїв.

На відеокарті присутня відеопам'ять, характерною рисою якої є те, що вона двухпортова – приєднана як до шини, по якій передаються дані від центрального процесора, так і до мікросхеми, відповідальної за вивід на дисплей, і вони можуть одночасно звертатися до неї. У дешевих пристроях як відеопам'ять використовується частина основної пам'яті, що значно сповільнює обробку даних на відеокарті. Обсяг пам'яті повинен бути достатнім для зберігання даних кадрового буфера, а бажано ще й **вторинного буфера** (англ. back buffer), якщо використовується технологія **подвійний буферизації**. Справа в тому, що якщо міняти значення пікселів прямо в момент виводу їх на екран, то при високій частоті відновлення можуть виникати артефакти, пов'язані з тим, що на екран виводиться ще не відбудоване до кінця зображення. Щоб цього уникнути, при подвійний буферизації під час виводу зображення з області відеопам'яті, що призначена кадровим буфером (називаним у цьому випадку також **первинним буфером** (англ. front buffer)), зображення наступного кадру будується у вторинному буфері, а при показі наступного кадру ці області пам'яті міняються ролями. Ця технологія використовується для показу динамічних зображень, таких як ігри. Додаткова відеопам'ять також прискорює обробку графіки, дозволяючи тримати додаткові графічні елементи, які відображаються в кадровому буфері за допомогою **блітинга** (про це див. нижче).

Доступ до відеопам'яті з боку процесора може бути організований подвійно – або відеопам'ять, як частина адрес, включається в адресний простір процесора, або для копіювання даних між основною й відеопам'яттю контролеру на відеокарті посилає спеціальна команда й копіювання відбувається за допомогою DMA (від англ. Direct Memory Access – мікросхема,

що дозволяє здійснювати передачу даних в/з оперативної пам'яті периферійним пристроям без участі центрального процесора).

Мікросхема, відповідальна за вивід на дисплей, постійно сканує відеопам'ять, перетворить її у форму, що відповідає інтерфейсу дисплея, і формує вихідний сигнал, переданий по кабелі на дисплей (див. рис. 2.6). Якщо відеоплата оснащена аналоговим виходом, то в неї повинен бути вбудований цифро-аналоговий перетворювач, називаний RAMDAC – Random Access Memory Digital to Analog Converter. Тому що інформація про пікселях передається послідовно, те RAMDAC повинен мати досить високу тактову частоту, щоб дозволяти виводити зображення високого розширення з достатньою частотою відновлення. Наприклад, зображення 1600×1200 для виводу із частотою 75 Гц вимагає частоти RAMDAC рівної 1600×1200×75 = 144 Мгц. Частота роботи RAMDAC є важливою характеристикою відеокарти.

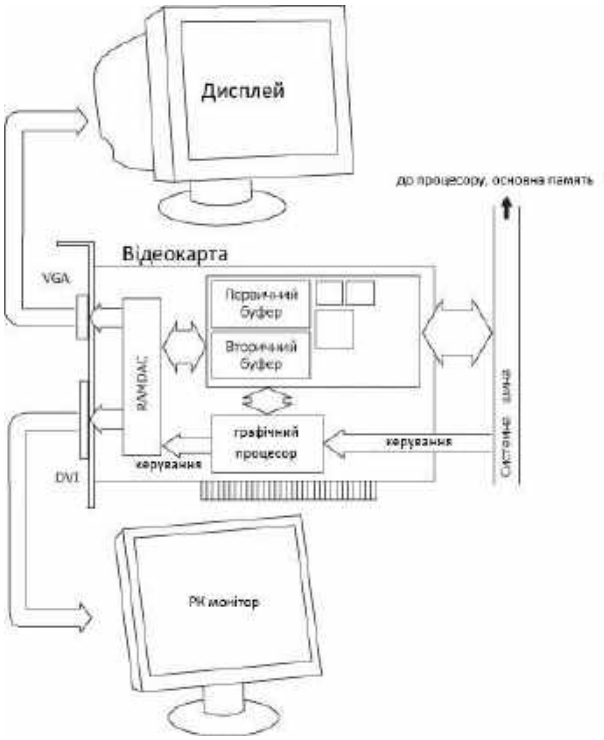


Рис. 2.6. Архітектура відеопідсистеми ПК

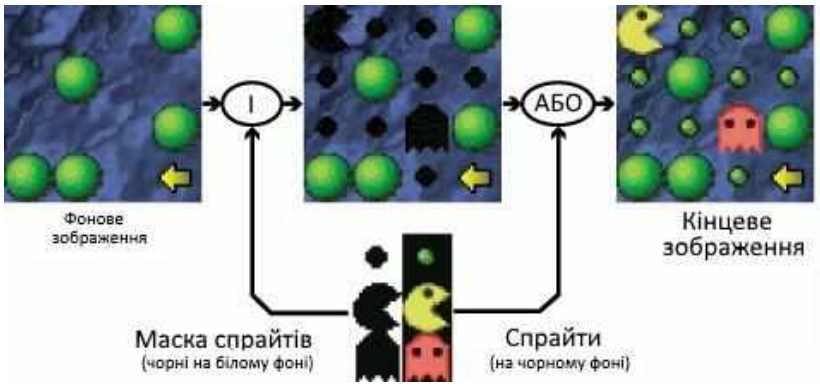


Рис. 2.7. Приклад зображення спрайтів

Також на відеоплаті втримується графічний процесор, здатний швидко виконувати основні операції по роботі із зображеннями у відеопам'яті, які можуть бути розділені на кілька класів.

– **Робота із прямокутними блоками**. Ця мікросхема називається **бліттер,** тому що основна операція яку вона робить – це BitBlt (Bit Block Transfer), тобто копіювання прямокутного блоку зображення в інше зображення з можливим застосуванням побітових логічних операцій (І, АБО, ЩО ВИКЛЮЧАЄ АБО). Це часто використовувана операція для приміщення об'єктів довільної форми, т.зв. **спрайтів**, на зображення. Приклад того, як це можна зробити за допомогою двох BitBlt операцій див. на рис. 2.7 (для розуміння необхідно врахувати, що пікселі чорного кольору складаються з одних 0 бітів, а білого – з одних 1 бітів). Також відеокарта може підтримувати операцію StretchBlt, – це те ж саме, що й BitBlt, але з розтяганням по осях.

– **Растеризація примітивно** дозволяє робити растеризацію найпростіших об'єктів, таких як відрізки, кола, еліпси, прямокутники, багатокутники. Також може підтримуватися заливання одноколірних зон іншим кольором або по шаблоні. При цьому може використовуватися й апаратний **антиаліасинг**. Ці алгоритми докладно розглянуті в даній книзі. До цієї групи також можна віднести апаратну підтримку відображення курсору.

– **Підтримка виводу символів**. Цей блок відповідає за вивід символів на екран певним шрифтом. Іноді шрифт можна варіювати або завантажувати з основної пам'яті свій. Даний блок активно використовується, коли відеокарта перебуває в **текстовому режимі,** коли дисплей логічно ділиться на певну кількість прямокутних осередків, найчастіше 80×25, у кожну з яких може бути поміщений один символ з обмеженого піднабору ASCII[9)].Вид кожного із цих символів визначається в спеціальній таблиці відеокарти, що може бути змінена. У цей час цей режим використовується при завантаженні ПК, а також при роботі в режимі термінала (частіше використовується в ОС Linux), див. рис. 2.8.

– **Апаратне прискорення відео й фільтрація зображення**. Кодування й декодування відео – дуже ресурсномістка операція, пов'язана з обробкою більших обсягів даних. Деякі відеокарти здатні апаратно декодувати **відеопотік**, тобто послідовність стислих відеоданих, що відповідає певному формату. Найчастіше це стандарт MPEG-210 яким закодовані фільми на DVD[11)].У сучасних відеокартах також починає з'являтися підтримка **Телебачення високої чіткості** (англ. HDTV – High Definition Television). Також можливо й апаратне масштабування відео. Апаратне відображення відео в частині екрана зветься **оверлея** (англ. overlay). Деякі відеокарти також можуть апаратно робити фільтрацію зображень, а також здійснювати гамма-корекцію (див. розділ 2.3).

Практично всі сучасні відеокарти містять також процесор обробки тривимірної графіки, але ця тема виходить за рамки даної книги.

Додатково на тій же платі часто виведені стандартні рознімання для підключення телевізорів і відеотехніки (т.зв. TV In/Out) і встановлені

мікросхеми первинної обробки цієї інформації, зокрема, YUV/RGB перетворення.

Деякі сучасні відеокарти дозволяють підключати відразу кілька дисплеїв одночасно.

2.4.2. Подання зображень

Тепер варто докладніше зупинитися на форматах подання растрових зображень у пам'яті. Звичайно пікселі описуються набором не негативних цілих чисел. Зображення звичайно зберігаються пострічково; у кожному рядку інформація про пікселях упорядкована ліворуч праворуч, а рядки відповідно розташовуються один по одному зверху долілиць[12)].При цьому, як правило, кожний наступний рядок починається із границі машинного слова[13)](при цьому, можливо, кінець попереднього рядка доповнюється нулями), тому що це оптимально для обробки процесором.

По глибині кольору зображення діляться на:

Глибина кольору в bpp	Назва	Примітки
1	Монохромне	Представимо 2 кольору – чорний і білий
8	Палітрове	Байт є індексом у таблиці палітри, за допомогою цієї палітри представимо будь-яке 256-елементна підмножина всіх True Color кольорів
8	Напівтонове (Відтінки сірого)	Містить тільки один канал інтенсивності без колірної інформації з точністю 8 біт. Можна розглядати як підвид палітрового, де елементи палітри відповідають відтінкам сірого
16	High Color	Повнокольорове, кількість біт на кожний компонент R-G-B: 5-5-5 (тобто насправді 15 біт, один не використовується) або 5-6-5
24	True Color	Повнокольорове, на кожну з RGB компонент по 8 біт
32	True Color	Те ж саме, що й попереднє, тільки додається ще один байт для вирівнювання інформації з машинного слова. Цей байт може також використовуватися як альфа-канал, тобто задавати ступінь прозорості

2-2- і 4-бітні режими CGA і EGA, що застосовувалися в ПК в 1980-х рр., але застарілі на даний момент, у таблиці не представлені.

У палітровому 8-бітному режимі перетворення з палітрового у відображуваний колір виконується апаратно пристроєм RAMDAC, у регістрах якого й утримується палітра. Цей режим зараз уже мало використовується через занадто обмежену кількість одночасно відображуваних кольорів.

Як неважко підрахувати, True Color дозволяє представити 16,7 млн можливих відтінків, чого цілком достатньо для повноцінного сприйняття людиною більшості фотографій. У той же час, у деяких випадках помітна недостатність діапазону яскравості. Серйозною проблемою є також втрата точності при операціях над пікселями із повночисловим описом. Наприклад, при роботі в True Color режимі спочатку зменшимо яскравість у два рази, а потім збільшимо. При цьому молодший значущий біт кожного компонента обнулиться, тому що спочатку біти були зрушені вправо, а потім назад уліво з доповненням 0. Тому, незважаючи на те, що графічна підсистема ПК поки може відтворювати зображення з не більш ніж 8 біт/канал, деякі програми обробки зображень мають режим 16 біт/канал (48 bpp), служби для запобігання втрати точності при редагуванні.

До того ж деякі із пристроїв введення, розглянутих вище, дозволяють одержувати зображення з більшою глибиною кольору. Деякі high-end відеокарти вже здатні працювати з точністю 10 і 12 біт/канал (тобто 30 і 36 bpp).

Останнім часом стає досить популярної концепція **Зображень зі збільшеним динамічним діапазоном** (англ. HDRI – High Dynamic Range Images), які могли б одночасно описувати дуже різний ступінь освітленості, комбінуючи, наприклад, зйомку фотоапаратом відразу з декількома значеннями експозиції. У багатьох реалізаціях кожний канал описується числом із плаваючою крапкою одинарної точності (розміром 32 біта). Сучасні дисплеї можуть однак показувати такі зображення тільки з певною експозицією.

2.4.3. Програмний інтерфейс

Найперші ПК, що з'явилися на початку 1980-х років, працювали винятково в текстовому режимі. У цьому режимі найменшим примітивом при виводі на дисплей є символ цілком, а не окремі пікселі. Хоча можна було управляти видом цих символів, завантаживши відповідні монохромні растри в спеціальну таблицю відеокарти[14)].

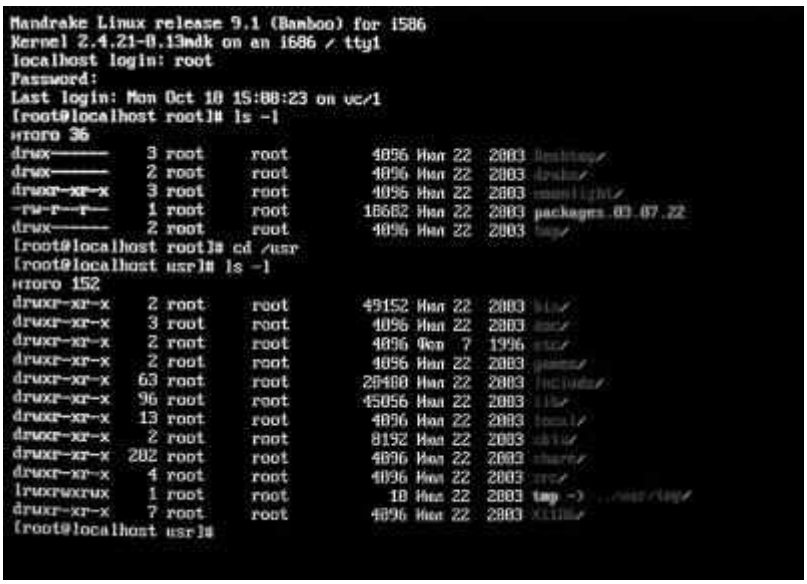


Рис. 2.8. Текстовий режим в ОС Linux

Потім з'явилися карти із графічними можливостями. У зв'язку з малим розміром адресного простору (1 Мб) процесора Intel 8086 доводилося відображати тільки частина відеопам'яті в адресний простір процесора й спеціальних команд задавати, яка саме це частина. Така технологія одержала назву **bank switching**. Команди відеокарті посилали шляхом переривань або запису інформації безпосередньо в її апаратний порт. Для використання додаткових можливостей відеокарт розроблювачам прикладних програм і ігор доводилося самим реалізовувати найпростіші операції для кожного їхнього типу, тому що підтримка відеокарт із боку ОС[15)] була мінімальною.

З появою ОС із графічним інтерфейсом ситуація змінилася. Прошарок між прикладною програмою й апаратурою став "товстіше". Безпосередньо на низькому рівні відеокартою управляє її **драйвер** – програма, що поставляється, як правило, самою фірмою-розроблювачем відеокарти. А прикладна програма звертається до нього через виклики чітко певного загального для всіх драйверів абстрактного інтерфейсу (англ. API – Application Programming Interface). Таким чином, з'явилася апаратна незалежність, що явилося важливим кроком уперед, з обліком безлічі відеокарт із обмеженою сумісністю один з одним.

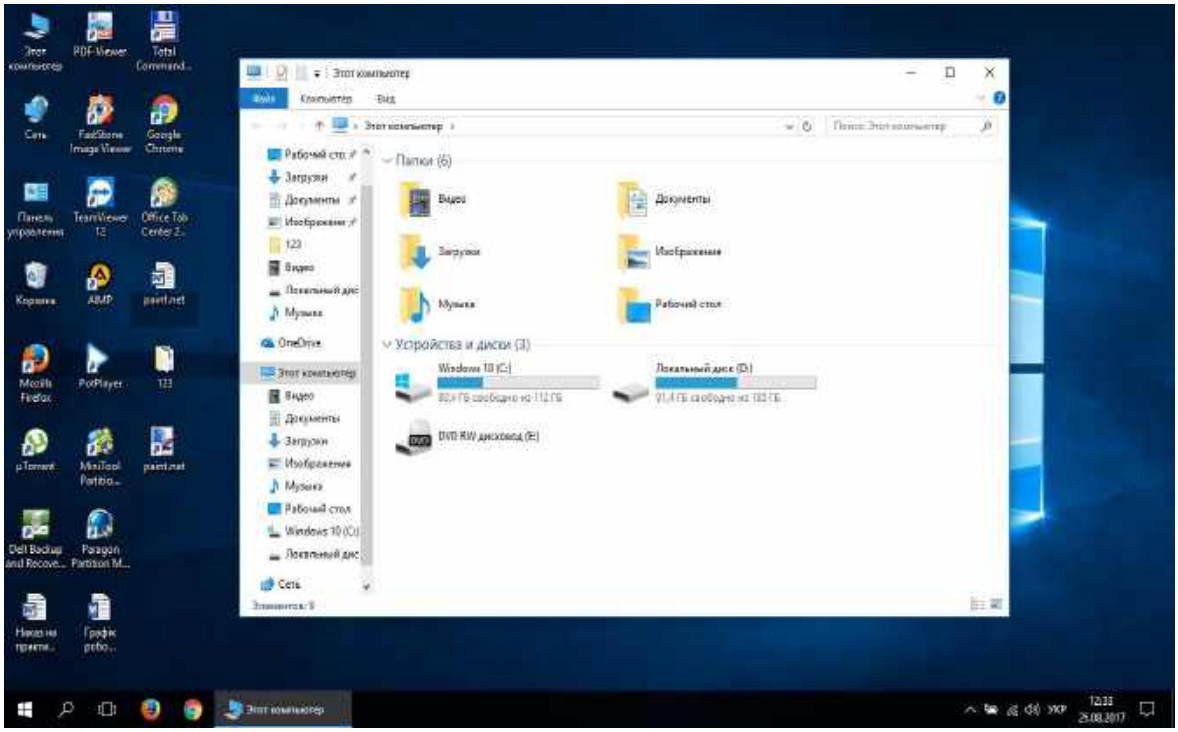


Рис. 2.9. Графічний інтерфейс ОС Windows

Типові функції такого інтерфейсу містять у собі саме операції **бліттинга** й растеризації примітивів, а також роботу з палітрами, хоча зараз палітри вже майже не використовуються.

В UNIX-Подібних ОС графічний інтерфейс надається системою X Windows, що працює за принципом "клієнт-сервер". Програма-Клієнт відправляє який-небудь запит API по мережі (хоча для самої програми це виглядає як просто виклик функції); одержавши цей запит, програма-сервер

відповідає за його виконання. Хоча така схема і є гнучкою (можна, наприклад, мати кілька дисплеїв в одного комп'ютера або, навпаки, багато комп'ютерів, підключених до одного дисплея), але в той же час вона вимагає й додаткових витрат на передачу даних по мережевому протоколі. Насправді, у ПК не використовується ця гнучкість, а всі запити передаються в рамках однієї системи (через поділювану між процесами клієнта й сервера пам'ять).

1) Споконвічно вони називалися IBM PC, тому що виробництво таких комп'ютерів було почато фірмою IBM в 1981 році.
2) 1 дюйм = 25,4 мм
3) ПЗЗ – "Прилад із Зарядовим Зв'язком"
4) хоча говорять про розв'язну здатність із погляду фотографії – розмірі найменшого помітного об'єкта
5) з урахуванням телевізорів
6) Наприклад, існують принтери, що друкують на лазерних дисках.
7) англ. Printer Control Language, розроблений фірмою Hewlett Packard.
8) Іноді це називають "Software RIP".
9) American Standard Code for Information Interchange – розповсюджена 8-бітне кодування символів
10) сокр. від Motion Picture Experts Group – група дослідників, що розробила даний стандарт
11) Digital Video Disc – найпоширеніший стандарт цифрового відео
12) Як не дивно, у форматі BMP за замовчуванням рядка розташовані знизу нагору, що приводить до додаткових витрат на операції із цими файлами.
13) у сучасних ПК це найчастіше 4 байти = 32 біта
14) Таким чином, зокрема, вироблялася русифікація DOS програм.
15) Операційна Система

2.5. Питання й вправи

1. Назвіть основні характеристики растрової комп'ютерної графіки.
2. Які параметри використовуються для подання кольору?

Тема 3. КОЛІР У КОМП'ЮТЕРНІЙ ГРАФІЦІ

Колірні моделі: RGB, HSV, CMY і інші. Перехід від однієї моделі до іншої. Колірний графік МКО. Однорідні колірні простори Luv, PHS.

3.1. Про природу світла й кольору

Світло як фізичне явище являє собою потік електромагнітних хвиль різної довжини й амплітуди. Око людини, будучи складною оптичною системою, сприймає ці хвилі в діапазоні довжин приблизно від 350 до 780 нм. Світло сприймається або безпосередньо від джерела, наприклад, від освітлювальних приладів, або як відбитий від поверхонь об'єктів або переломлений при проходженні крізь прозорі й напівпрозорі об'єкти. Колір – це характеристика сприйняття оком електромагнітних хвиль різної довжини, оскільки саме довжина хвилі визначає для ока видимий колір. Амплітуда, що визначає енергію хвилі (пропорційну квадрату амплітуди), відповідає за яскравість кольору. Таким чином, саме поняття кольору є особливістю людського "бачення" навколишнього середовища.

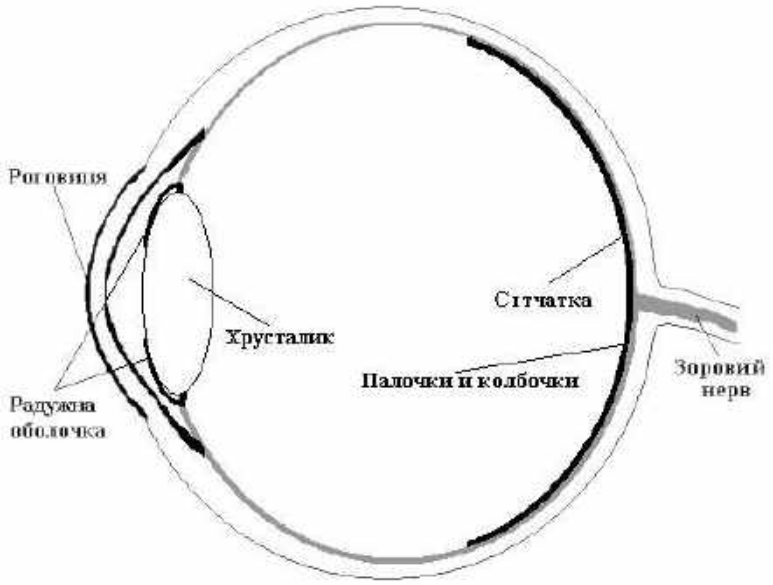


Рис. 3.1. Око людини

На рис. 3.1 схематично зображене око людини. Фоторецептори, розташовані на поверхні сітківки, відіграють роль приймачів світла. Кришталик – це своєрідна лінза, що формує зображення, а райдужна оболонка виконує роль діафрагми, регулюючи кількість світла, що пропускається усередину ока. Чутливі клітки ока неоднаково реагують на хвилі різної довжини. *Інтенсивність* світла є міра енергії світла, що впливає на око, а *яскравість* – це міра сприйняття оком цього впливу. Інтегральна крива спектральної чутливості ока наведена на рис. 3.2; це **стандартна крива Міжнародної комісії з висвітлення (МКО, або CIE – Comission International de l'Eclairage)**.

Фоторецептори підрозділяються на два види: палички й колбочки. Палички є високочутливими елементами й працюють в умовах слабкого висвітлення. Вони нечутливі до довжини хвилі й тому не "розрізняють" кольору. Колбочки ж, навпаки, мають вузьку спектральну криву й "розрізняють" кольору. Паличок існує тільки один тип, а колбочки підрозділяються на три види, кожний з яких чутливий до певного діапазону довжин хвиль (довгі, середні або короткі.) Чутливість їх також різна.

На рис. 3.3 представлено криві чутливості колбочок для всіх трьох видів. Видно, що найбільшою чутливістю володіють колбочки, що сприймають кольори зеленого спектра, небагато слабкіше – "червоні" колбочки й істотно слабкіше – "сині".

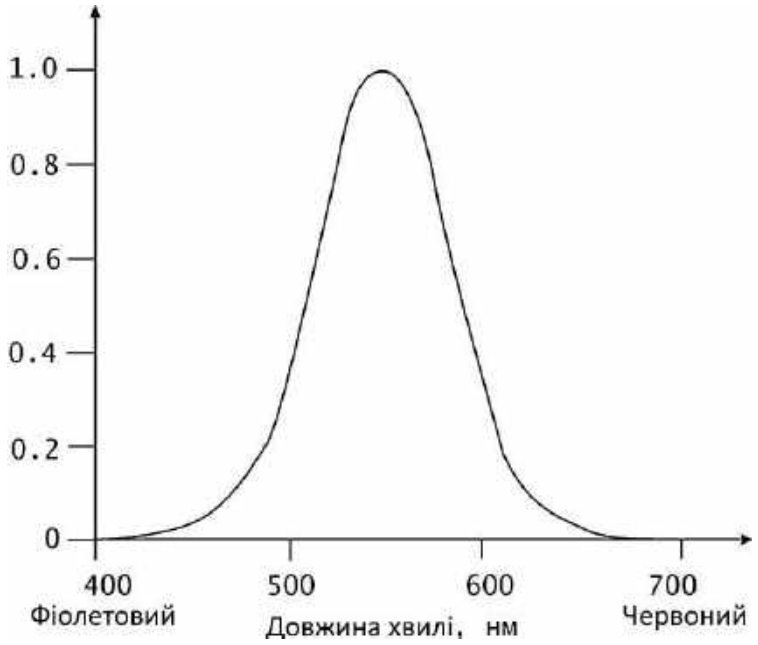


Рис. 3.2. Інтегральна крива спектральної чутливості ока

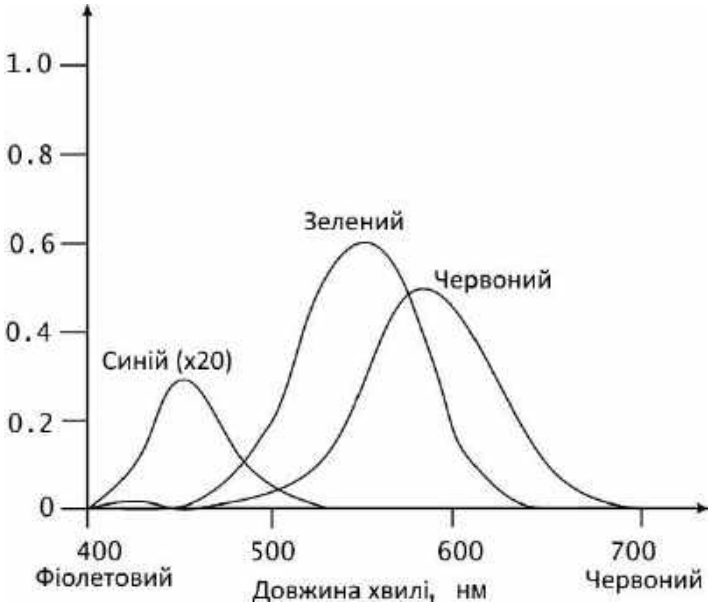


Рис. 3.3. Криві чутливості різних рецепторів

Таким чином, якщо функція $C(\lambda)$ характеризує спектральне розкладання світлового випромінювання від деякого джерела (рис. 3.4),т. е. розподіл інтенсивності по довжинах хвиль, те три типи колбочок будуть посилати в мозок сигнали R, G, B (червоний, зелений, синій), потужність яких визначається інтегральними співвідношеннями

$$R = \int C(\lambda) S_R(\lambda) d\lambda,$$

$$G = \int C(\lambda) S_G(\lambda) d\lambda,$$

$$B = \int C(\lambda) S_B(\lambda) d\lambda,$$

де S_R, S_G, S_B – функції чутливості відповідних типів колбочок.

Рис. 3.4. Характерна спектральна крива

Якщо сприймане світло містить всі видимі довжини хвиль у приблизно рівних кількостях, то він називається **ахроматичним** і при максимальній інтенсивності сприймається як білий, а при більше низьких інтенсивностях – як відтінки сірого кольору. Інтенсивність відбитого світла зручно розглядати в діапазоні від 0 до 1, і тоді нульове значення буде відповідати чорному кольору. Якщо ж світло містить довжини хвиль у нерівних пропорціях, то він є **хроматичним**. Об'єкт, що відбиває світло, сприймається як кольоровий, якщо він відбиває або пропускає світло у вузькому діапазоні довжин хвиль. Точно так само й джерело світла сприймається як кольоровий, якщо він випускає хвилі у вузькому діапазоні довжин. При висвітленні кольорової поверхні кольоровим джерелом світла можуть виходити досить різноманітні колірні ефекти.

3.2. Колірний графік МКО

Тривимірна природа сприйняття кольору дозволяє відображати його в прямокутній системі координат. Будь-який колір можна зобразити у вигляді

вектора, компонентами якого є відносні ваги червоної, зеленої й синьої квіток, обчислені по формулах

$$r = \frac{R}{R+G+B}, \quad g = \frac{G}{R+G+B}, \quad b = \frac{B}{R+G+B}.$$

Оскільки ці координати в сумі завжди становлять одиницю, а кожна з координат лежить у діапазоні від 0 до 1, те всі представлені в такий спосіб крапки простору будуть лежати в одній площині, причому тільки в трикутнику, що відтинається від її позитивним октантом системи координат (рис. 3.5а). Ясно, що при такому поданні вся безліч крапок цього трикутника можна описати за допомогою двох координат, тому що третя виражається через них за допомогою співвідношення

$$b = 1 - r - g.$$

Таким чином, ми переходимо до двовимірного подання області, тобто до проекції області на площину (рис. 3.5б).

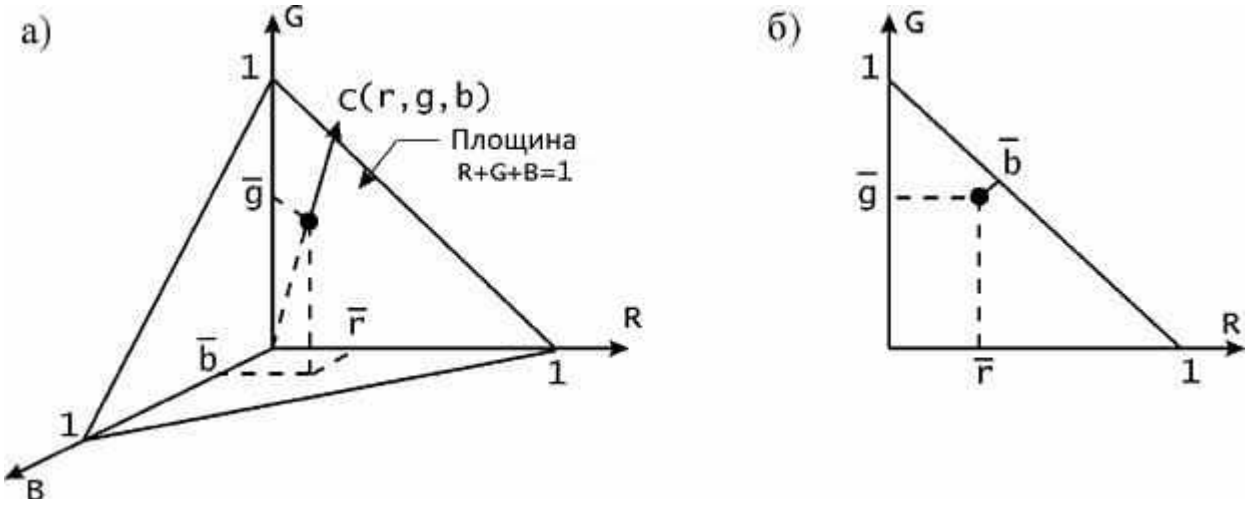


Рис. 3.5. Тривимірний колірний простір

З використанням такого перетворення в 1931 р. були вироблені міжнародні стандарти визначення й виміру кольорів. Основою стандарту став так званий двовимірний колірний графік МКО. Оскільки, як показали фізичні експерименти, додаванням трьох основних кольорів можна одержати не всі можливі колірні відтінки, то в якості базисних були обрані інші параметри, отримані на основі дослідження стандартних реакцій ока на світло. Ці параметри – X, Y, Z – є чисто теоретичними, оскільки побудовані з використанням негативних значень основних складові кольори. Трикутник основних кольорів був побудований так, щоб охоплювати весь спектр видимого світла. Крім того, рівна кількість всіх трьох гіпотетичних кольорів у сумі дає білий колір. Координати кольоровості будуються так само, як і в наведеній вище формулі:

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z}, \quad x+y+z = 1$$

При проекції цього трикутника на площину виходить колірний графік МКО. Але координати кольоровості визначають тільки відносні кількості

основних кольорів, не задаючи яскравості результуючого кольору. Яскравість можна задати координатою Y, а X, Z визначити виходячи з величин (x, y, Y), по формулах

$$X = \frac{Y}{y}x, \quad Z = \frac{Y}{y}(1 - x - y).$$

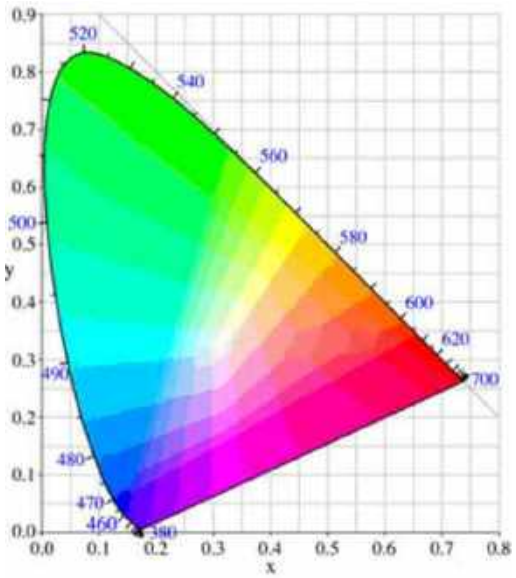


Рис. 3.6. Колірний графік МКО.
На контурі зазначені довжини хвиль у нанометрах

Колірний графік МКО наведений на рис. 3.6. Область, обмежена кривій, охоплює весь видимий спектр, а сама крива називається лінією спектральних кольорів. Числа, проставлені на малюнку, означають довжину хвилі у відповідній крапці. Крапка C, що відповідає полуденному висвітленню при суцільній хмарності, прийнята в якості опорного білого кольору.

Колірний графік зручний для цілого ряду завдань. Наприклад, з його допомогою можна одержати додатковий колір: для цього треба провести промінь від даного кольору через опорну крапку до перетинання з іншою стороною кривій (кольори є **додатковими** друг до друга, якщо при додаванні їх у відповідній пропорції виходить білий колір). Для визначення домінуючої довжини хвилі якого-небудь кольору також проводиться промінь із опорної крапки до перетинання з даним кольором і триває до перетинання з найближчою крапкою лінії кольорів.

Для змішання двох кольорів використовуються закони Грассмана. Нехай два кольори задані на графіку МКО координатами $D_1 = (x_1, y_1, Y_1)$ й $D_2 = (x_2, y_2, Y_2)$. Тоді змішання їх дає колір $D_{12} = (x_1 + x_2, y_1 + y + 2, z_1 + z_2)$. Якщо ввести позначення $t_1 = \frac{Y_1}{y_1}, t_2 = \frac{Y_2}{y_2}$, то одержимо координати кольоровості суміші

$$x_{12} = \frac{x_1 t_1 + x_2 t_2}{t_1 + t_2}, \quad y_{12} = \frac{y_1 t_1 + y_2 t_2}{t_1 + t_2}, \quad Y_{12} = Y_1 + Y_2.$$

Координати МКО є точним стандартом визначення кольору. Але в різних областях, що мають справу з кольором, є свій підхід до його моделювання. Зокрема, може використовуватися інший набір основних кольорів. Комп'ютерна графіка опирається на систему RGB, тому становить інтерес перехід між цими двома наборами кольорів (іншими словами, перетворення координат кольоровості).

3.3. Колірні моделі RGB і CMY

Колірні моделі, використовувані в комп'ютерній графіці, – це засобу опису кольорів у певному діапазоні.

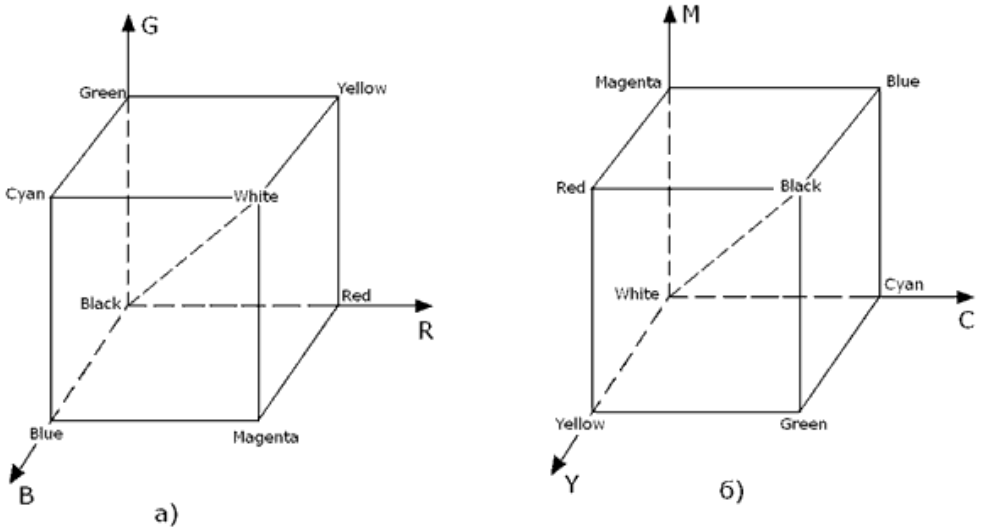


Рис. 3.7. Колірний куб для моделей RGB і CMY

На основі описаних вище фізичних подань у комп'ютерній графіці була прийнята так звана **адитивна колірна модель**, що використовує **три первинні складові кольори**. Ця модель припускає, що будь-який колір можна розглядати як зважену суму трьох основних кольорів. Проілюструвати її можна на прикладі висвітлення сцени за допомогою трьох прожекторів різного кольору. Кожний прожектор управляється незалежно, і шляхом зміни потужності кожного з них можна відтворити практично всі кольори. У моделі RGB колір можна представити у вигляді вектора в тривимірній системі координат з початком відліку в крапці (0,0,0). Максимальне значення кожної з компонентів вектора приймемо за 1. Тоді вектор (1,1,1) відповідає білому кольору. Всі колірні вектори, таким чином, укладені усередині одиничного куба, називаного колірним кубом (рис. 3.7а).

Інша модель змішання кольорів – **субстантивна колірна модель**, або модель CMY, що використовує в якості первинних складові кольори Cyan, Magenta, Yellow (блакитний, пурпурний, жовтий), які є додатковими до Red, Green, Blue. У цій моделі відтінки кольори виходять шляхом "вирахування" з падаючого світла хвиль певної довжини. Цей підхід має потребу в поясненні. У цій системі координат вектор (0,0,0) відповідає білому кольору, а вектор (1,1,1) – чорному. Відповідний колірний куб представлений на рис. 3.7б.

Зв'язок між значеннями (R,G,B) і (C,M,Y) для того самого кольору виражається формулою

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

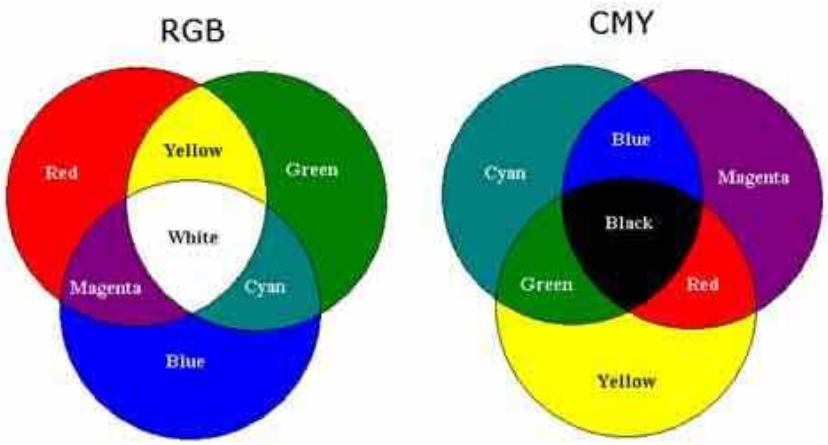


Рис. 3.8. Схема змішання кольорів для моделей RGB і CMY

Кольору однієї моделі є додатковими до кольорів іншої (додатковий колір – це колір, результатом змішування якого з даним є білий). Схема змішування кольорів для двох моделей представлена на рис. 3.8. Приклад субстантивного формування відтінків показаний на рис. 3.9. При висвітленні падаючим білим світлом у шарі блакитний (Cyan) фарби зі спектра білого кольору поглинається (віднімається) червона частина як додатковий колір, потім зі світла, що залишилося, у шарі пурпурної (Magenta) фарби поглинається зелена частина спектра, і, нарешті, від білої поверхні відбивається синій колір, що ми й бачимо. Таким чином, змішання блакитної й пурпурної квіток дає в підсумку синій колір.

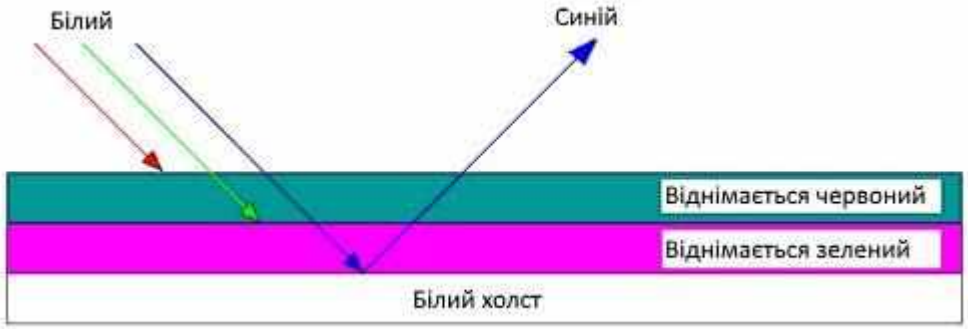


Рис. 3.9. Субстрактивне формування відтінків

Растрові дисплеї, як правило, використовують апаратно- орієнтовану модель кольорів RGB. Існують також дисплеї з **таблицею кольоровості**, що представляє собою матрицю, кожний елемент якої – деякий колір (вектор

RGB). У таких дисплеях значення кодів пікселів, що заносяться у відеопам'ять, являють собою індекси матриці кольоровості. При відображенні деякого пікселя на екран за значенням коду вибирається елемент таблиці кольоровості, що містить трійку значень R, G, B. Ця трійка й передається на монітор для завдання кольору пікселя на екрані.

У повнокольорових дисплеях для кожного пікселя у відеопам'ять заноситься трійка значень R, G, B. У цьому випадку для відображення пікселя з відеопам'яті безпосередньо вибираються значення R, G, B, які й передаються на монітор (але можуть і передаватися в таблицю кольоровості).

У моделях RGB і CMY легко задавати яскравості для одного з основних кольорів, але досить важко задати відтінок з необхідним колірним тоном і насиченістю, що відповідає якому-небудь зразку кольору. У різного роду графічних редакторах це завдання найчастіше вирішується за допомогою інтерактивного вибору з палітри кольорів і формуванням кольорів у палітрі шляхом підбора значень координат до одержання необхідного візуального результату. Іноді така палітра наочно відображає вибір вектора з колірного куба: спочатку за допомогою одного движка вибирається колірна площина, а потім на цій площині вибирається конкретна крапка. Але й таким методом не відразу вдається досягти бажаного ефекту, оскільки не так просто вибрати правильну колірну площину.

3.4. Колірні моделі HSV і HLS

Наведені моделі не охоплюють усього діапазону видимого кольору, оскільки їхній колірний охват – це лише трикутник на графіку МКО, вершинам якого відповідають базові кольори. Вони є апаратно орієнтованими, тобто відповідають технічній реалізації кольору в пристроях графічного виводу. Але психофізіологічне сприйняття світла визначається не інтенсивністю трьох первинних кольорів, а колірним тоном, насиченістю й світлістю. Колірний тон дозволяє розрізняти кольори, насиченість задає ступінь "розведення" чистого тону білим кольором, а світлість – це інтенсивність світла в цілому. Тому для адекватного нашому сприйняттю підбора відтінків більше зручними є моделі, у числі параметрів яких є присутнім тон (Hue). Цей параметр прийнятий вимірювати кутом, відлічуваним навколо вертикальної осі. При цьому червоному кольору відповідає кут 0°, зеленому – 120°, синьому – 240°, а доповнюючі один одного кольори розташовані один напроти іншого, тобто кут між ними становить 180°. Кольори CMY розташовані посередині між складовими їхніми компонентами RGB. Існує дві моделі, що використовують цей параметр.

Модель HSV (Hue, Saturation, Value, або тон, насиченість, кількість світла) можна представити у вигляді світлової шестигранної піраміди (рис. 3.10), по осі якої відкладається значення V, а відстань від осі до бічної грані в горизонтальному перетині відповідає параметру S (за діапазон зміни цих величин приймається інтервал від нуля до одиниці). Значення S дорівнює одиниці, якщо крапка лежить на бічній грані піраміди. Шестикутник, що

лежить у підставі піраміди, являє собою проекцію колірного куба в напрямку його головної діагоналі (рис. 3.11).

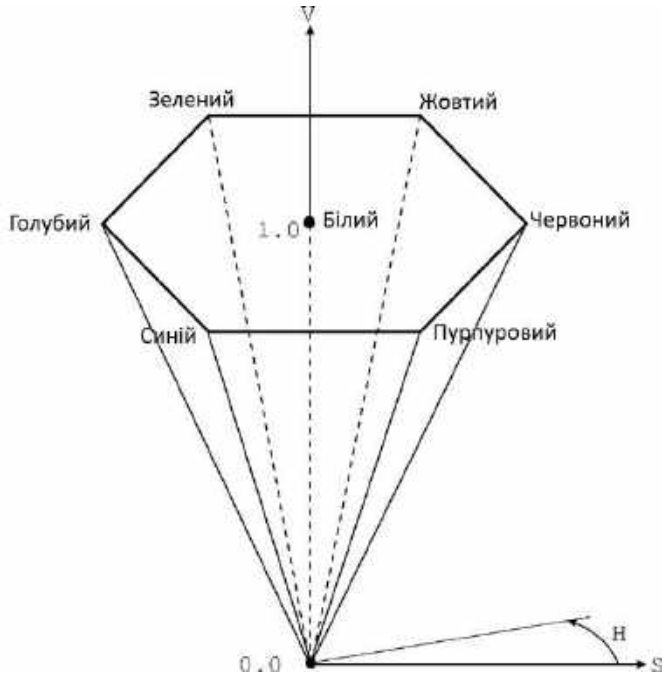


Рис. 3.10. Колірний простір HSV

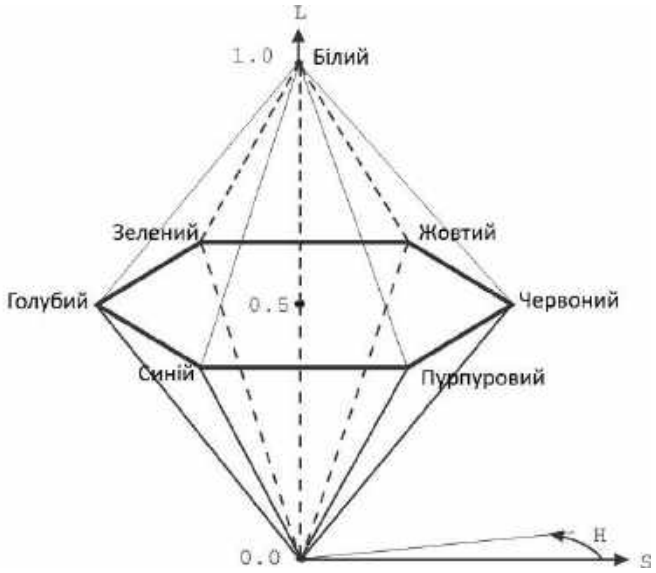


Рис. 3.11. Колірний простір HLS

Перетворення колірного простору HSV в RGB здійснюється безпосередньо за допомогою геометричних співвідношень між шестигранною пірамідою й кубом.

Колірна модель HLS (Hue, Lightness, Saturation, або тон, світлість, насиченість) є розширенням моделі HSV. Тут колірний простір уже представляється у вигляді подвійної піраміди (рис. 3.13), у якій по вертикальній осі відкладається L (світлість), а інші два параметри задаються так само, як і в

попередній моделі. У літературі ці піраміди іноді називають шестигранним конусом.

На рис. 3.12 і 3.13 наведені блок-схеми перетворення моделей HSV і HLS у модель RGB. Алгоритми зворотного перетворення пропонуються читачеві як вправа.

У першому алгоритмі використовується функція Ent, що означає цілу частину числа. Крім того, використовується операція присвоювання для векторів. Константа *ndf* (скорочене від вираження "not defined") використовується при вході в алгоритм для того, щоб з'ясувати, чи задане значення змінної H. Наприклад, за згодою *ndf* може бути деяким негативним значенням, тому що тон – це завжди позитивна величина. У другому алгоритмі застосовується допоміжна функція *Value (H, M1, M2)* для обчислення значення компоненти R, G або B залежно від ситуації.

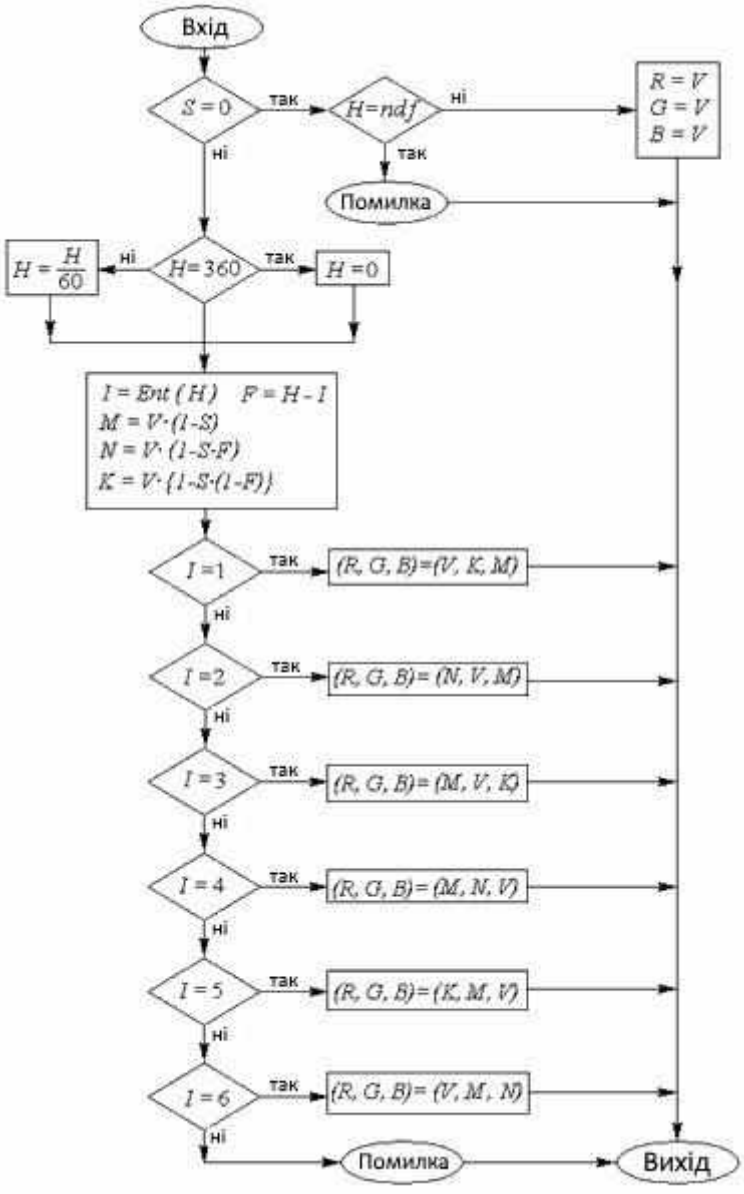


Рис. 3.12. Перетворення моделі HSV в RGB

Алгоритм перетворення:

Приведення H до заданого діапазону:

Поки H<0 H=H+360

Поки H>360 H= H-360

Визначення координат

Якщо H<60 то Value=M1+(M 2-M1)*H/60

Якщо 60<=H<180 то Value=M2

Якщо 180<=H<240 то Value=M1+(M 2-M1)*(240-H)/60

Якщо 240<=H то Value=M1

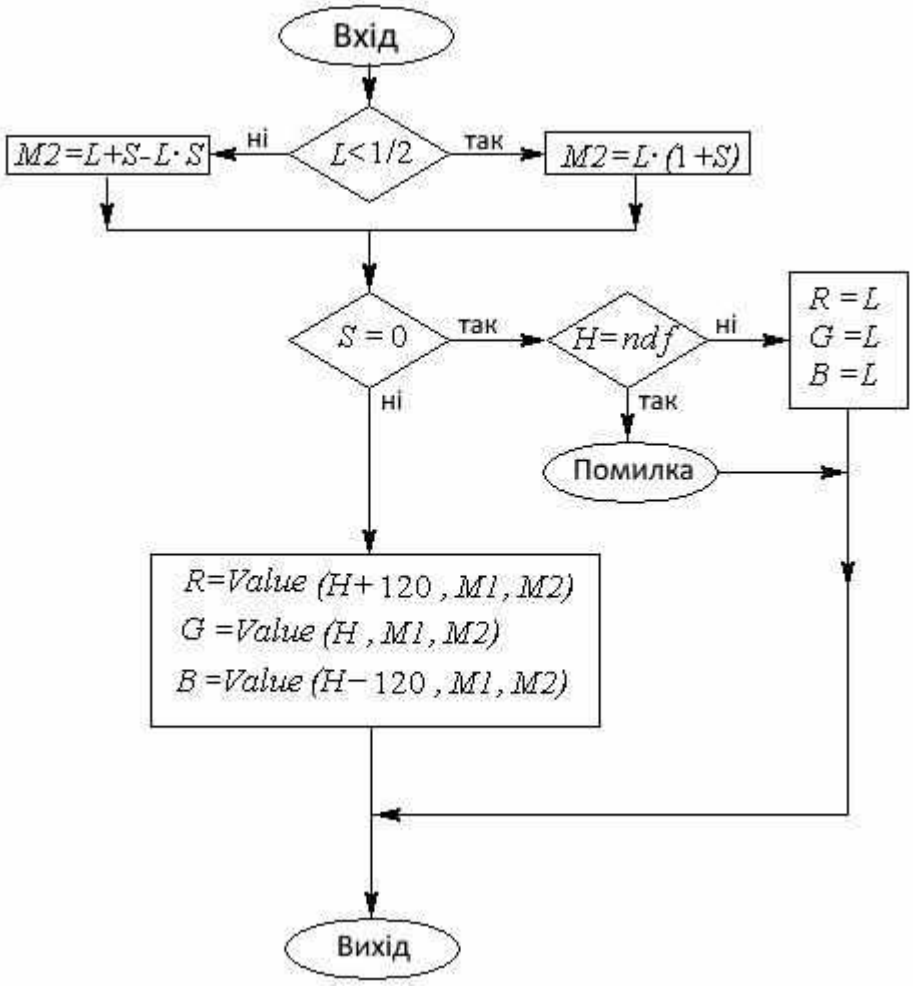


Рис. 3.13. Перетворення моделі HLS в RGB

3.5. Простір CIE Luv

Один з істотних мінусів колірного простору XYZ – це те, що воно не є перспективне (візуально) рівномірним і не може використовуватися для обчислення колірних відстаней. Тому CIE (МКО) продовжила розробку перспективної рівномірного простору. Метою комітету CIE було створення повторюваної системи стандартів передачі кольору для виробників фарб, чорнила, пігментів і інших барвників. Найважливіша функція цих стандартів – надати універсальну схему, у рамках якої можна було б установлювати відповідність кольорів.

У результаті був створений колірний простір CIE Luv, що дозволяє визначити розрізнення кольорів для людини з "усередненим" зором, (тобто різні люди неоднаково сприймають різницю між кольорами). Своя назва простір одержала завдяки його компонентам L, u і v. Параметр L відповідає яскравості кольору, u відповідає за перехід від зеленого до червоного (при збільшенні), а при збільшенні параметра v відбувається перехід від синього до фіолетового. Якщо u і v рівні 0, те, міняючи L, одержуємо кольору, що є градаціями сірого.

Цей колірний простір був розроблений для кількісного виміру розходження двох кольорів. CIE були проведені дослідження за участю великої кількості людей, результатом чого з'явилося створення простору Luv. Виміри проводилися в "гарних" умовах (достатнє висвітлення й неяскраве монотонне тло); перед випробуваним перебували два аркуші паперу, пофарбованих відповідно двома кольорами, і він повинен був дати відповідь, наскільки, на його думку, розрізняються ці кольори. У випадку реальних зображень ми повинні знаходити розходження між кольорами на більше складному тлі, при цьому не завжди при гарному висвітленні (наприклад, занадто яскравому). Але висвітлення залежить і від приміщення, і від часу доби, і від того, під яким кутом перебуває поверхня до джерела світла.

Перехід з RGB в Luv здійснюється в такий спосіб. Спочатку нормуємо R, G, B:

$$\begin{pmatrix} R^* \\ G^* \\ B^* \end{pmatrix} = \begin{pmatrix} R/255 \\ G/255 \\ B/255 \end{pmatrix}$$

Далі робимо перетворення простору RGB в XYZ:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.35758 & 0.180423 \\ 0.212671 & 0.71516 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \times \begin{pmatrix} R^* \\ G^* \\ B^* \end{pmatrix}$$

Колірний простір CIE Luv – безперервне однорідне перетворення простору CIE XYZ, описуване наступними формулами:

$$L = \begin{cases} 116 \times \sqrt[3]{\frac{Y}{Y_n}} - 16, & \frac{Y}{Y_n} > 0.008856 \\ 903.3 \times \frac{Y}{Y_n}, & \frac{Y}{Y_n} \le 0.008856 \end{cases}$$

$$u' = \frac{4X}{X + 15Y + 3Z} = \frac{4x}{-2x + 12y + 3}, \quad v' = \frac{9Y}{X + 15Y + 3Z} = \frac{9y}{-2x + 12y + 3}$$

$$u = 13L(u' - u_n), \quad v = 13L(v' - v_n)$$

Для визначення параметрів Y_n, u_n і v_n, уводиться поняття **білої крапки** (**white point**). Біла крапка – це пари параметрів кольоровості (x, y), що визначає еталон білого кольору для різних джерел світла. CIE склала таблицю білих крапок для джерел світла різної яскравості. При цьому значення компонента Y білої крапки в XYZ нормалізовано до 100 (у наведені вище формулах Y_n саме відповідає нормалізованої Y компоненту). Параметри u_n й v_n обчислюються по тимі ж формулам, що u' й v', у яких використовуються значення x і для білої крапки.

Як уже згадувалося вище, компонента L відповідає яскравості кольору, а з формул видно, що L пропорційно кубічному кореню з компонента Y простору XYZ. Однак існує думка, що людському сприйняттю більше відповідає корінь другого ступеня з освітленості. Так, наприклад, у колірному просторі Lab параметр L обчислюється з використанням квадратного кореня.

Небагато про властивості величин L, u, v:

- L міняється від 0 до 100;
- u, v лежать у межах -200, 200;
- u відповідає за перехід від зеленого до червоного (при збільшенні u);
- v відповідає за перехід від синього до фіолетового (при збільшенні v);
- якщо u і v рівні 0, міняючи L, одержуємо зображення, що містить градації сірого (grayscale).

Нарешті, найважливіше, до чого ми прагнули, переходячи в цей простір. Нам задані два кольори – L_1, u_1, v_1 і L_2, u_2, v_2. Як визначити відстань між кольорами, тобто наскільки людина помітила б розходження між ними? Виявляється, воно задається евклідовою нормою

$$D = \sqrt{(L_1 - L_2)^2 + (u_1 - u_2)^2 + (v_1 - v_2)^2}$$

При відстані між двома кольорами $D > 5$ більшість людей уже зауважують розходження, при $D > 10$ воно помітно всім. У цьому й складається головне перевага цього простору. Воно враховує сприйняття кольорів людиною, і розходження між кольорами визначається дуже простою формулою. Необхідно помітити, що ця формула застосовна в певних умовах: висвітлення, тло не повинні заважати й відволікати.

Одночасно з розробкою CIE Luv було також розроблене перцептивно рівномірний колірний простір CIE Lab. Із цих двох моделей більш широко застосовується модель CIE Lab. Структура колірного простору Lab заснована на тій теорії, що колір не може бути одночасно зеленим і червоним або жовтим і синім (рис. 3.14). Отже, для опису атрибутів "червоний/зелений" і "жовтий/синій" можна скористатися тими самими значеннями. Формули переходу від простору XYZ до простору Lab здійснюється в такий спосіб:

$$L = \begin{cases} 116 \cdot [(Y/Y_n)^{1/3}] - 16 & \text{если } (Y/Y_n) > 0.008856 \\ 903.3 \cdot Y/Y_n & \text{если } (Y/Y_n) \le 0.008856 \end{cases} \quad a = 500 \cdot [f(X/X_n) - f(Y/Y_n)] \quad b = 200 \cdot [f(Y/Y_n) - f(Z/Z_n)]$$

$$\text{где } f(t) = \begin{cases} t^{1/3} & \text{если } (Y/Y_n) > 0.008856 \\ 7.787 \cdot t + 16/116 & \text{если } (Y/Y_n) \le 0.008856 \end{cases}$$

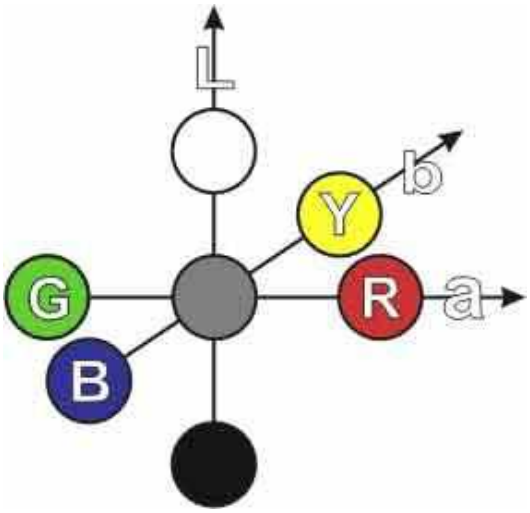


Рис. 3.14. Подання кольору в просторі CIE Lab

Рис. 3.15. Видимий стандартним спостерігачем простір Lab

Кожна колірна модель, крім переваг, також має й свої індивідуальні недоліки. Існують і інші моделі, які тут не розглядаються.

3.6. Питання й вправи

1. Розташуєте в зворотному порядку чутливість рецепторів ока до кольорів: червоний, зелений, синій.
2. Що таке хроматичний спектр?
3. Що таке ахроматичний спектр?
4. Як здійснюється проекція тривимірного колірного простору на площину?
5. Чим відрізняється колірний графік МКО від трикутної проекційної області колірного простору?
6. Що таке додатковий колір?
7. Що таке адитивна й субстантивна колірні моделі? Чим відрізняються їхні колірні куби?
8. Що є основою колірної моделі HSV і HLS?
9. Чи є колірні моделі HSV і HLS адитивними або субстантивними?
10. Побудуйте алгоритм перетворення моделі RGB в HSV.
11. Побудуйте алгоритм перетворення моделі RGB в HLS.
12. У чому складається головне перевага колірного простору Luv?
13. У чому складається головне перевага колірного простору Lab?

Тема 4. ГЕОМЕТРИЧНІ ПЕРЕТВОРЕННЯ

Системи координат і геометричні перетворення (паралельний перенос, масштабування, обертання). Завдання геометричних перетворень за допомогою матриць. Конгруентні перетворення. Перехід в іншу систему координат. Завдання обертання щодо довільної осі.

4.1. Системи координат і вектори

Для подальшого викладу нам знадобляться деякі відомості з аналітичної геометрії й лінійної алгебри. Не ставлячи перед собою завдання докладного розгляду всіх цих питань, приведемо (або нагадаємо) ті основні поняття й операції, які використовуються в алгоритмах комп'ютерної графіки.

Дві взаємно перпендикулярні пересічні прямі із заданим масштабом утворять **декартову прямокутну систему координат на площині**. Крапка перетинання *O* називається **початком координат**, прямі називаються **осями координат**. Одну з осей називають *віссю OX*, або **віссю абсцис**, іншу – *віссю OY*, або *віссю ординат*. Ці осі також називають **координатними осями.**

Візьмемо довільну крапку M на площині із заданою системою координат. Нехай M_x і M_y – проекції цієї крапки на осі абсцис і ординат відповідно, причому довжина відрізка OM_x дорівнює x , а довжина OM_y дорівнює y. Тоді пари чисел (x, y) називається декартовими координатами крапки M на площині (**абсцисою** й **ординатою** крапки).

Три взаємно перпендикулярні пересічні прямі із заданим масштабом утворять **декартову прямокутну систему координат у просторі.** Так само як і у випадку площини, крапка перетинання *O* називається **початком координат,** прямі називаються **осями координат**. Одну з осей називають *віссю OX*, або **віссю абсцис**, іншу – *віссю OY*, або **віссю ординат**, третю – *віссю OZ*, або **віссю аплікат.**

Нехай M_x, M_y і M_z – проекції довільної крапки M в просторі на осі абсцис, ординат і аплікат відповідно, причому довжина відрізка OM_x дорівнює x , довжина OM_y дорівнює y, а довжина OM_z дорівнює z. Тоді трійка чисел x, y, z називається декартовими координатами крапки M в просторі (**абсцисою, ординатою** й **аплікатою** крапки).

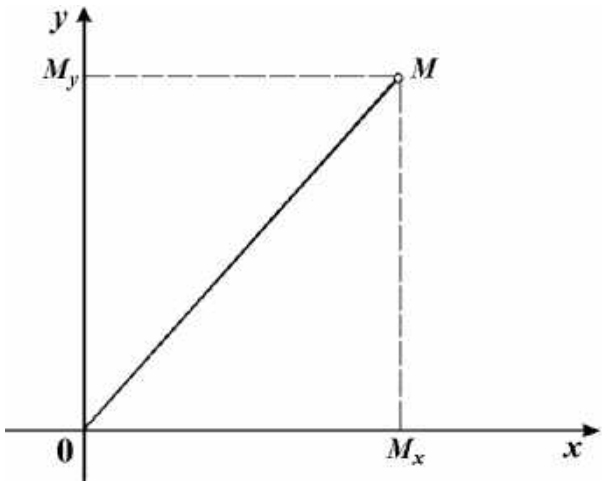


Рис. 4.1. Система координат на площині

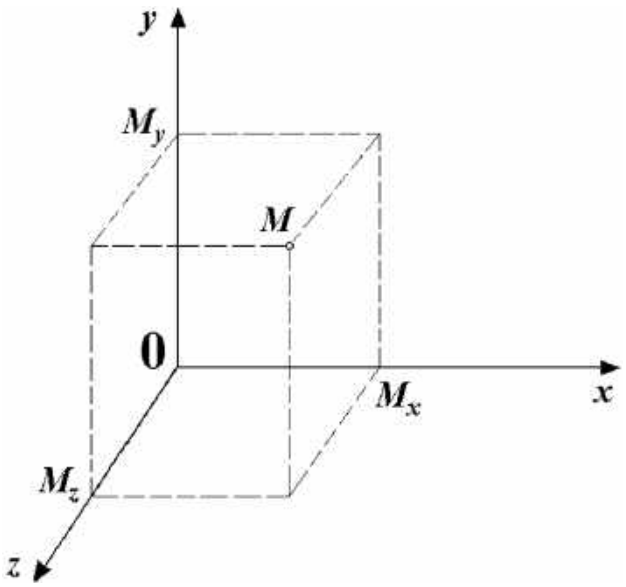


Рис. 4.2. Система координат у просторі

Нехай на площині задана декартова система координат. Візьмемо дві крапки з координатами (x_1, y_1) й (x_2, y_2) відповідно. Тоді, використовуючи терему Піфагора, можна одержати, що відстань між цими двома крапками виражається формулою $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Відстань між двома крапками в просторі з координатами (x_1, y_1, z_1) й (x_2, y_2, z_2) виражається аналогічною формулою:
$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$.

Відрізок на площині й у просторі задається за допомогою двох крапок, що вказують його границі. **Геометричним вектором**, або просто **вектором у просторі**, будемо називати відрізок, у якого зазначено, яка з його граничних крапок є початком, а яка – кінцем (тобто зазначений **напрямок** вектора). Початок вектора називають **крапкою його додатка**. Вектор називається **нульовим**, якщо його початок і кінець збігаються. Вектори називаються **колінеарними**, якщо вони лежать на паралельних прямих. Вектори вважаються **рівними**, якщо вони колінеарними, мають однакову довжину й однаковий напрямок. Таким чином, всі вектори, що виходять паралельним переносом з того самого вектора, рівні між собою. Будь-яка крапка на площині й у просторі може розглядатися як вектор, початок якого збігається з початком координат (радіус-**вектор**), а кожний вектор, перенесений у початок координат, задає своїм кінцем єдину крапку простору. Тому будь-який вектор може бути представлений сукупністю своїх координат у декартовій системі.

Лінійними операціями над векторами прийнято називати операції додавання векторів і операцію множення вектора на число.

Сумою двох векторів $\vec{a}$ і $\vec{b}$ називається вектор, що йде з початку вектора $\vec{a}$ в кінець вектора $\vec{b}$, за умови, що вектор $\vec{b}$ прикладений до кінця вектора $\vec{a}$.

Перелічимо основні властивості операції додавання векторів:

– $\vec{a} + \vec{b} = \vec{b} + \vec{a}$.

– $(\vec{a} + \vec{b}) + \vec{c} = \vec{a} + (\vec{b} + \vec{c})$.

– Існує нульовий вектор $\vec{0}$, такий, що $\vec{a} + \vec{0} = \vec{a}$ для будь-якого вектора $\vec{a}$.

– Для кожного вектора $\vec{a}$ існує протилежний йому вектор $\vec{a}'$, такий, що $\vec{a} + \vec{a}' = \vec{0}$.

Різницею двох векторів $\vec{a}$ і $\vec{b}$ називається такий вектор $\vec{c}$, що у сумі з вектором $\vec{b}$ дає вектор $\vec{a}$.

Добутком $\alpha\vec{a}$ вектора $\vec{a}$ на число α називається вектор $\vec{b}$, колінеарний вектору $\vec{a}$, що має довжину $|\alpha| \cdot |\vec{a}|$ й напрямок, що збігається з напрямком вектора $\vec{a}$ при $\alpha > 0$ й протиставлене напрямку $\vec{a}$ при $\alpha < 0$. Геометричний зміст множення вектора на число полягає в тому, що довжина вектора збільшується в $|\alpha|$ раз.

Операція множення вектора на число має наступні властивості:

– $\alpha(\vec{a} + \vec{b}) = \alpha\vec{a} + \alpha\vec{b}$ (розподільна властивість числового співмножника щодо суми векторів);

– $(\alpha + \beta)\vec{a} = \alpha\vec{a} + \beta\vec{a}$ (розподільна властивість векторного співмножника щодо суми чисел);

– $(\alpha\beta)\vec{a} = \alpha(\beta\vec{a})$ (сполучна властивість числових співмножників);

– якщо вектор $\vec{b}$ колінеарний ненульовому вектору $\vec{a}$, то існує речовинне число β, таке, що $\vec{b} = \beta\vec{a}$.

Лінійною комбінацією векторів $\vec{a}$ і $\vec{b}$ називається вектор $\vec{c} = \alpha\vec{a} + \beta\vec{b}$. При цьому числа α й β називаються **коефіцієнтами розкладання** вектора $\vec{c}$ по векторах $\vec{a}$ і $\vec{b}$.

Якщо два вектори $\vec{r}_1$ й $\vec{r}_2$ задані своїми координатами (x_1, y_1, z_1) й (x_2, y_2, z_2), то операції над ними легко виразити через ці координати:

– $\vec{r}_1 + \vec{r}_2 = \vec{r} = (\vec{x}_1 + \vec{x}_2, \vec{y}_1 + \vec{y}_2, \vec{z}_1 + \vec{r}_2)$;

– $\vec{r}_1 - \vec{r}_2 = \vec{r} = (\vec{x}_1 - \vec{x}_2, \vec{y}_1 - \vec{y}_2, \vec{z}_1 - \vec{r}_2)$;

– $\alpha\vec{r}_1 = \vec{r} = (\alpha\vec{x}_1, \alpha\vec{y}_1, \alpha\vec{z}_1)$.

Вектори $\vec{a}$, $\vec{b}$ і $\vec{c}$ називаються компланарними, якщо вони лежать в одній площині.

Вектори називаються лінійно незалежними, якщо рівність нулю їхньої лінійної комбінації можливо тільки у випадку рівності нулю коефіцієнтів α і β.

Справедливі наступні властивості:

– Які б не були неколінеарні вектори $\vec{a}$ й $\vec{b}$, для будь-якого вектора $\vec{c}$, що лежить в одній площині з ними, існують числа α й β, такі, що $\vec{c} = \alpha\vec{a} + \beta\vec{b}$, причому така пари чисел для кожного вектора єдина. Таке подання вектора $\vec{c}$ називається розкладанням по векторах $\vec{a}$ і $\vec{b}$.

– Які б не були некомпланарні вектори $\vec{a}$, $\vec{b}$ і $\vec{c}$, для будь-якого вектора $\vec{d}$ існують числа α, β і γ, такі, що $\vec{d} = \alpha\vec{a} + \beta\vec{b} + \gamma\vec{c}$, причому ця трійка чисел для кожного вектора – єдина (розкладання вектора $\vec{d}$ по векторах $\vec{a}, \vec{b}, \vec{c}$).

– Будь-які три вектори в системі координат площини є лінійно залежними.

– Будь-які чотири вектори в системі координат простори є лінійно залежними.

Говорять, що пари лінійно незалежних векторів на площині (трійка лінійно незалежних векторів у просторі) утворять **базис**, оскільки будь-який вектор може бути представлений у вигляді лінійної комбінації цих векторів. Коефіцієнти розкладання вектора по базисних векторах називаються **координатами вектора в цьому базисі**. Якщо вектори базису взаємно перпендикулярні й мають одиничну довжину, то базис називається ортонормованим, а вектори базису називаються **ортами**. Таким чином, базис із одиничних векторів, спрямованих уздовж осей декартової системи координат, є ортонормованим.

Скалярним добутком векторів $\vec{r}_1$ і $\vec{r}_2$ називається число, рівне добутку довжин цих векторів на косинус кута між ними. Будемо позначати скалярний добуток векторів символом $(\vec{r}_1 \cdot \vec{r}_2)$. Тоді скалярний добуток можна виразити формулою $(\vec{r}_1 \cdot \vec{r}_2 = |\vec{r}_1| \cdot |\vec{r}_2| \cos\alpha)$.

Нескладно довести наступні властивості даної операції.

– Скалярний добуток двох ненульових векторів дорівнює нулю тоді й тільки тоді, коли ці вектори ортогональними.

– Якщо кут між двома векторами гострий, то скалярний добуток цих векторів позитивно, якщо ж кут тупий, те скалярний добуток негативно.

– $(\vec{r}_1 \cdot \vec{r}_2) = (\vec{r}_2 \cdot \vec{r}_1)$ (властивість комутативності).

– $\alpha(\vec{r}_1 \cdot \vec{r}_2) = (\alpha\vec{r}_1 \cdot \vec{r}_2) = (\vec{r}_1 \cdot \alpha\vec{r}_2)$ (сполучне щодо числового множника властивість).

– $((\vec{r}_1 + \vec{r}_2) \cdot \vec{r}_3) = (\vec{r}_1 \cdot \vec{r}_3) + (\vec{r}_2 \cdot \vec{r}_3)$ (розподільне щодо суми векторів властивість).

– Скалярний добуток вектора самого на себе дорівнює квадрату довжини вектора.

Приведемо деякі формули, пов'язані з розкладанням вектора в декартовій системі координат.

Нехай вектори $\vec{r}_1$ й $\vec{r}_2$ задані своїми координатами (x_1, y_1, z_1) й (x_2, y_2, z_2). Тоді їхній скалярний добуток може бути обчислене по формулі

$$(\vec{r}_1 \cdot \vec{r}_2) = x_1x_2 + y_1y_2 + z_1z_2. \quad (4.1)$$

Звідси треба умова перпендикулярності векторів:

$x_1x_2 + y_1y_2 + z_1z_2 = 0.$

І, нарешті, косинус кута між векторами обчислюється по формулі

$$\cos\varphi = \frac{x_1x_2 + y_1y_2 + z_1z}{2\sqrt{x_1^2 + y_1^2 + z_1^2} \cdot \sqrt{x_2^2 + y_2^2 + z_2^2}.} \quad (4.2)$$

Тепер відстань між двома крапками з координатами (x_1, y_1, z_1)й (x_2, y_2, z_2)можна виразити через скалярний добуток відповідних векторів:

$d = \sqrt{(\vec{r}_1 - \vec{r}_2)}.$

Уведемо ще одне поняття, що стосується векторів. Три вектори називаються впорядкованою трійкою, якщо зазначено, який із цих векторів є першим, який – другим і який – третім. При записі трійки векторів будемо розташовувати ці вектор у порядку їхнього проходження. Так, запис $\vec{b}\,\vec{a}\,\vec{c}$ означає, що першим вектором трійки є вектор $\vec{b}$, другим – $\vec{a}$, третім – $\vec{c}$.

Трійка векторів називається **правої (лівої)**, якщо після приведення до загального початку вектор $\vec{c}$ розташовується по ту сторону від площини, що містить вектори $\vec{a}$, $\vec{b}$, звідки найкоротший поворот від $\vec{a}$ до $\vec{b}$ здається **проти годинникової стрілки (за годинниковою стрілкою).**

Векторним добутком вектора $\vec{a}$ на вектор $\vec{b}$ називається вектор $\vec{c}$, позначуваний символом $\vec{a} \times \vec{b}$ і задовольняючою наступною вимогами:

– довжина вектора $\vec{c}$ дорівнює добутку довжин векторів $\vec{a}$, $\vec{b}$ на синус кута між ними, тобто

– $|\vec{c}| = |\vec{a}| \cdot |\vec{b}| \sin\varphi;$

– вектор $\vec{c}$ ортогональний векторам $\vec{a}$, $\vec{b}$;

– вектор $\vec{c}$ спрямований так, що трійка векторів $\vec{a}\,\vec{b}\,\vec{c}$ є правою.

Приведемо (без доказу) основні властивості векторного добутку.

– $[\vec{a} \times \vec{b}] = -[\vec{b} \times \vec{a}]$ (антисиметричність);

– $\alpha[\vec{a} \times \vec{b}] = [\alpha\vec{a} \times \vec{b}]$ (сполучна властивість щодо множення на число);

– $[(\vec{a} + \vec{b}) \times \vec{c}] = [\vec{a} \times \vec{c}] + [\vec{b} \times \vec{c}]$ (розподільна властивість щодо додавання);

– $[\vec{a} \times \vec{a} = 0$для будь-якого вектора $\vec{a}$.

Ясно, що векторний добуток двох колінеарних векторів дає нульовий вектор. Виведемо тепер формулу для векторного добутку. Нехай базисні вектори декартової системи координат $\vec{i}, \vec{j}, \vec{k}$ утворять праву трійку. Тоді справедливі наступні співвідношення:

$$[\vec{i} \times \vec{j}] = \vec{k} = -[\vec{j} \times \vec{i}], \quad [\vec{j} \times \vec{k}] = \vec{i} = -[\vec{k} \times \vec{j}],$$
$$[\vec{k} \times \vec{i}] = \vec{j} = -[\vec{i} \times \vec{k}]$$

Якщо задані два вектори $\vec{r}_1 = x_1\vec{i} + y_1\vec{j} + z_1\vec{k}$ й $\vec{r}_2 = x_2\vec{i} + y_2\vec{j} + z_2\vec{k}$, те, з огляду на властивості векторного добутку, звідси легко вивести, що $\vec{r}_3 = [\vec{r}_1 \times \vec{r}_2] = x_3\vec{i} + y_3\vec{j} + z_3\vec{k}$, де

$$x_3 = y_1z_2 - z_1y_2, \quad y_3 = z_1x_2 - x_1z_2, \quad z_3 = x_1y_2 - y_1x_2. \tag{4.3}$$

4.2. Рівняння прямої на площині

Рівняння прямої на площині в декартовій системі координат можна задати рівнянням виду $y = kx + b$, для випадку, коли пряма не паралельна осі *OY*, і рівнянням $x = c$, для вертикальної прямої. Але пряма може бути також задана й іншим способом. Досить указати вектор напрямку цій прямій $\vec{l} = (l_x, l_y)$ і якій-небудь крапці $\vec{r}_0 = (x_0, y_0)$, що лежить на цій прямій. При цьому крапки, що лежать на прямій, можуть бути задані з використанням векторних операцій у вигляді так званого **параметричного рівняння прямій** $\vec{r} = \vec{r}_0 + t\vec{l}$, у якому параметр *t* пробігає всі значення числової прямої. Координати крапки, що відповідає деякому значенню цього параметра, визначаються співвідношеннями

$$x = x_0 + tl_x, \quad y = y_0 + tl_y. \tag{4.4}$$

Пряму в просторі теж можна задавати параметричним рівнянням, що дуже легко одержати з попереднім простим переходом від двовимірних векторів до тривимірного. Нехай $\vec{l} = (l_x, l_y, l_z)$, $\vec{r}_0 = (x_0, y_0, z_0)$. Тоді це рівняння буде визначати пряму в просторі, а координати крапок цій прямій будуть визначатися формулами

$$x = x_0 + tl_x, \quad y = y_0 + tl_y, \quad z = z_0 + tl_z, \quad -\infty < t < +\infty. \tag{4.5}$$

Як відомо з елементарної геометрії, через будь-які три крапки в просторі проходить площина. З іншого боку, через кожну крапку площини можна провести єдину пряму, перпендикулярну даної площини. При цьому всі ці прямі будуть паралельні один одному, а виходить, вони мають загальний вектор напрямку. Цей вектор будемо називати **нормаллю до площини**. Якщо довжина вектора дорівнює одиниці, ми будемо називати його **одиничною нормаллю**. У комп'ютерній графіці часто доводиться вирішувати завдання побудови нормалі до деякої площини, заданої трьома крапками, а також завдання перетинання прямій із площиною й двох площин.

Площина в просторі можна задати, указавши вектор нормалі до неї і яку-небудь крапку, що належить даної площини. Нехай $\vec{n} = (n_1, n_2, n_3)$ – вектор одиничної нормалі, а $\vec{r}_0 = (x_0, y_0, z_0)$ – деяка крапка на площині. Тоді для

будь-якої крапки $\vec{r} = (x, y, z)$, що лежить на площини, вектор $\vec{r} - \vec{r}_0$ буде ортогональне вектору нормалі, а отже, виконується рівність $((\vec{r} - \vec{r}_0) \cdot \vec{n}) = 0.$

Розкриваючи це вираження в координатному виді, одержуємо

$$n_1 x + n_2 y + n_3 z - n_1 x_0 - n_2 y_0 - n_3 z_0 = 0.$$

Тепер перепишемо це рівняння у вигляді

$$n_1 x + n_2 y + n_3 z + d = 0, \tag{4.6}$$

де $d = -n_1 x_0 - n_2 y_0 - n_3 z_0$. Це рівняння називається канонічним рівнянням площини. При цьому зовсім ясно, що якщо все це рівняння помножити на який-небудь відмінний від нуля множник, то воно буде описувати ту ж саму площину, тобто коефіцієнти n_1, n_2, n_3 для кожної площини задаються з точністю до довільного ненульового множника. Але якщо при цьому вектор $\vec{n}$ має одиничну довжину, то $|d|$ задає відстань від початку координат до даної площини.

В алгоритмах комп'ютерної графіки досить часто доводиться зіштовхуватися із завданням побудови площини, що проходить через три задані крапки. Нехай три крапки $\vec{r}_1$, $\vec{r}_2$ і $\vec{r}_3$, що не лежать на одній прямій, мають координатами $(x_1, y_1, z_1), (x_2, y_2, z_2)$ й (x_3, y_3, z_3). Для канонічного рівняння необхідно побудувати нормаль до площини, що легко можна здійснити, використовуючи операцію векторного добутку. Оскільки вектори $\vec{v}_1 = \vec{r}_2 - \vec{r}_1$ й $\vec{v}_2 = \vec{r}_3 - \vec{r}_1$ лежать у шуканій площині, то вектор $\vec{N} = \vec{v}_1 \times \vec{v}_2$ буде ортогональний цієї площини. Нехай $\vec{N} = (N_x, N_y, N_z)$, тоді рівняння площини буде мати вигляд $N_x x + N_y y + N_z z + D = 0.$

Залишається визначити значення D. Тому що крапка $\vec{r}_1$ належить цій площині, те її координати повинні задовольняти отриманому рівнянню. Підставимо їх у рівняння й одержимо $N_x x_1 + N_y y_1 + N_z z_1 + D = 0,$ отже $D = -N_x x_1 - N_y y_1 - N_z z_1,$ і після підстановки остаточно одержимо:

$$N_x(x - x_1) + N_y(y - y_1) + N_z(z - z_1) = 0 \tag{4.7}$$

У більшості алгоритмів, що використовують площини, досить знати нормаль до неї і яку-небудь крапку, що належить площини. Очевидно, що за аналогією можна вивести канонічні рівняння прямої на площині, якщо задано нормаль до неї й приналежній прямій крапка.

4.2.1. Аналітичне подання кривих і поверхонь

Нехай на площині задана декартова система координат.

Крива на площині – це геометричне місце крапок (x, y), що задовольняють рівнянню

$$F(x, y) = 0 \tag{4.10}$$

де F – функція двох змінних. Ясно, що далеко не кожна функція буде задавати лінію. Так, наприклад, рівнянню $x^2 + y^2 + 1 = 0$ не задовольняє жодна крапка площини, а рівнянням $x^2 + y^2 = 0$ задовольняє тільки одна крапка $(0,0)$.

Для аналітичного подання кривої в багатьох випадках зручніше задавати криву параметричними рівняннями, використовуючи допоміжну змінну (параметр) t:

$$x = \varphi(t), \quad y = \psi(t), t \in [a,b], \tag{4.11}$$

де φ й ψ – безперервні функції на заданому інтервалі зміни параметра. Якщо функція $\varphi(t)$ така, що можна виразити t через $x(t = \varphi^{-1}(x))$, то від параметричного подання кривої легко перейти до рівняння (4.10):

$y = \psi(\varphi^{-1}(t)) = 0.$

Систему рівнянь (4.11) можна записати у векторному виді:

$\vec{r} = \vec{f}(t), \quad \vec{r} = (x,y), \quad \vec{f}(t) = (\varphi(t), \psi(t)).$

Відрізок прямої являє собою окремий випадок кривої, причому параметричне подання його може мати вигляд $x = t, \quad y = at + b, \quad t \in [t_1, t_2]$ або $x = at + b, y = t, t \in [t_1, t_2]$.

Окружність радіуса r із центром у крапці (x_0, y_0) може бути представлена параметричними рівняннями $x = x_0 + r \cdot \cos t, \quad y = y_0 + r \cdot \sin t, t \in [0, 2\pi]$.

Перейдемо до тривимірного простору із заданої декартової системою координат.

Поверхня в просторі – це геометричне місце крапок (x,y,z), що задовольняють рівнянню виду

$$F(x,y,z) = 0. \tag{4.12}$$

Так само як і у випадку кривої на площині, не всяка функція F описує яку-небудь поверхню. Наприклад, рівнянню $X^2 + y^2 + z^2 + 1 = 0$ не задовольняє жодна крапка простору. Поверхня також може бути задана в **параметричному виді**, але на відміну від кривої для цього потрібні два допоміжні змінні (параметри):

$$x = \varphi(u,v), \quad y = \psi(u,v), \quad z = \zeta(u,v), \quad u \in [a,b], \quad v \in [c,d]. \tag{4.13}$$

Наприклад, сфера радіуса r із центром у крапці (x_0, y_0, z_0) може бути задана рівнянням $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2 = 0$ або ж параметричними рівняннями

$x = x_0 + r \cdot \cos u \cdot \cos v, \quad y = y_0 + r \cdot \sin u, \quad z = z_0 + r \cdot \cos u \cdot \sin v.$

Криву в просторі можна описати як перетинання двох поверхонь, тобто за допомогою системи рівнянь

$$F_1(x,y,z) = 0, \quad f_2(x,y,z) = 0 \tag{4.14}$$

або параметричними рівняннями виду

$$x = \varphi(t), \quad y = \psi(t), z = \zeta(t), \quad t \in [a,b]. \tag{4.15}$$

4.3. Перетинання променю із площиною й сферою

Пряма на площині й у просторі є нескінченною в обидва боки. **Променем** називається напівпряма, тобто безліч всіх крапок прямій, що лежать по одну сторону від заданої її крапки, називаної початком променю. Промінь будемо задавати в параметричному виді, як це було описано в одному з попередніх розділів. Нехай $\vec{l} = (l_x, l_y, l_z)$– напрямний вектор прямій, а $\vec{r}_0 = (x_0, y_0, z_0)$– початкова крапка. Тоді координати крапок променю будуть визначатися формулами

$$x = x_0 + tl_x, \quad y = y_0 + tl_y, \quad z = z_0 + tl_z. \tag{4.8}$$

Будемо вважати, що напрямний вектор одиничний, тобто $l_x^2 + l_y^2 + L_z^2 = 1$.

Спочатку розглянемо завдання про знаходження крапки перетинання променю із площиною, заданої канонічними рівнянням

$$n_1 x + n_2 y + n_3 z + d = 0. \tag{4.9}$$

Вектор нормалі $\vec{n} = (n_1, n_2, n_3)$теж будемо вважати одиничним. Спочатку треба визначити значення параметра t, при якому промінь перетинає площина. Для цього підставимо координати з формули (4.8) у рівняння (4.9) і одержимо $n_1(x_0 + tl_x) + n_2(y_0 + tl_y) + n_3(z_0 + tl_z) + d = 0,$ звідки легко визначити, що промінь перетинає площина в крапці зі значенням

$$t_0 = -\frac{(\vec{r}_0 \cdot \vec{n}) + d}{(\vec{l} \cdot \vec{n})}.$$

Очевидно, що така крапка існує тільки за умови $(\vec{l} \cdot \vec{n}) \neq 0$. У свою чергу, ця величина звертається в нуль тільки у випадку, коли вектори $\vec{l}$ й $\vec{n}$ ортогональні один одному.

Нехай тепер нам задана сфера із центром у крапці $\vec{r}_c = (x_c, y_c, z_c)$ й радіусом d. Тоді рівняння сфери буде мати вигляд

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = d^2.$$

Підставивши сюди координати променю з рівняння (4.9), одержимо, що параметр, при якому промінь перетинає сферу, повинен задовольняти квадратному рівнянню $at_0^2 + bt_0 + c = 0,$

де $a = |\vec{r}_c|^2,\ b = 2((\vec{r}_0 - \vec{r}_c) \cdot \vec{l}),\ c = |\vec{r}_0 - \vec{r}_c|^2 - d^2$. Визначимо корінь цього рівняння. Якщо дискримінант $D = \frac{b^2}{4} - c \geq 0$, то корінь існують. Їх може бути або два $(D > 0)$, або один $(D = 0)$. У першому випадку маємо дві крапки перетинання, у другому – одну (промінь стосується сфери). Відповідні значення параметра визначаються співвідношенням

$$t_{1,2} = -\frac{b}{2} \mp \sqrt{D}$$

4.4. Інтерполяція функцій однієї й двох змінних

Крім функцій, заданих аналітично (тобто за допомогою елементарних функцій, значення яких легко можуть бути обчислені в будь-якій крапці області визначення), на практиці часто доводиться мати справа з таблично заданими функціями. У цьому випадку функція задається своїми значеннями на деякій дискретній безлічі крапок (**вузлів**) з області визначення. Якщо необхідно одержати значення функції в якій-небудь крапці, що не збігається з вузлом, використовують різні методи наближеного обчислення, які ґрунтуються на деяких апріорних припущеннях щодо цієї функції. При цьому сама процедура обчислення називається **інтерполяцією** у випадку, коли крапка належить заданій області, і екстраполяцією, якщо вона лежить поза областю.

Як припущення про характер дискретно заданої функції найбільше часто використовуваної й простій є те, що вона кусково- лінійна, тобто що в проміжках між вузлами вона поводиться відповідно до лінійного закону. Тоді інтерполяція називається **лінійної**, і цей метод ми будемо досить часто застосовувати в алгоритмах комп'ютерної графіки.

Нехай на площині задана система координат XOY і відрізок $[x_1, x_2]$ на осі OX, на кінцях якого задані значення y_1, y_2 деякої **лінійної** функції (рис. 4.3). Тоді для будь-якої крапки x усередині заданого відрізка відповідне значення y обчислюється по формулах $y = ty_1 + (1-t)y_2, \qquad t = (x_2 - x)/(x_2 - x_1).$

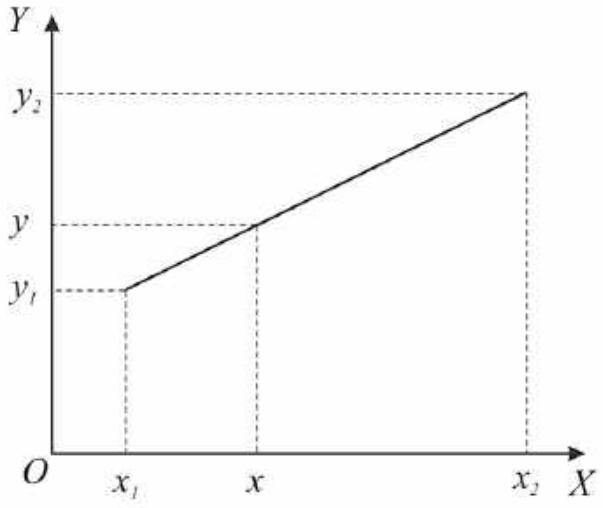


Рис. 4.3. Лінійна інтерполяція функції однієї змінної

Звернемося тепер до завдання інтерполяції функцій двох змінних. У цьому випадку найбільш простої також є інтерполяція по трьох заданих крапках знову ж за допомогою кусочно-лінійної функції. Нехай на площині заданий трикутник з вершинами $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ й задані значення функції в цих крапках z_1, z_2, z_3. Тоді три крапки (x_i, y_i, z_i) визначають у просторі трикутник, що є плоскою фігурою. Передбачається, що площа трикутника більше нуля, або, як говорять, **трикутник невироджений**. Для визначення значення функції в довільній крапці (x, y), що лежить усередині трикутника, скористаємося так

званими **барицентричними координатами** (α, β, γ) цієї крапки. Геометричний зміст цих координат полягає в тім, що вони дорівнюють відношенню площ трикутників, зображених на рис. 4.4:

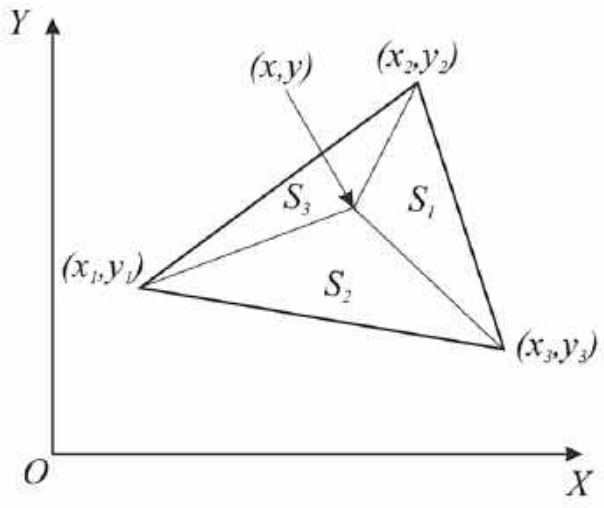


Рис. 4.4. Лінійна інтерполяція функції двох змінних

$$\alpha = \frac{S_1}{S},\ \beta = \frac{S_2}{S},\ \gamma = \frac{S_3}{S},\ S = S_1 + S_2 + S_3.$$

Ці числа ненегативні й задовольняють наступним співвідношенням:

$$\left.\begin{aligned} &\alpha + \beta + \gamma = 1 \\ &\alpha x_1 + \beta x_2 + \gamma x_3 = x \\ &\alpha y_1 + \beta y_2 + \gamma y_3 = y \end{aligned}\right\}.$$

Ці співвідношення будемо розглядати як рівняння для знаходження чисел (α, β, γ).

Визначник цієї системи рівнянь є

$$\Delta = \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix} = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1),$$

і він по модулі дорівнює подвоєної площі трикутника, тому $\Delta \neq 0$, отже, система має єдине рішення при будь-якій правій частині. Скористаємося формулами Крамера й випишемо вид цього рішення:

$$\alpha = \frac{\Delta_1}{\Delta}, \quad \beta = \frac{\Delta_2}{\Delta}, \quad \gamma = 1 - \alpha - \beta,$$

де

$$\Delta_1 = \begin{vmatrix} 1 & 1 & 1 \\ x & x_2 & x_3 \\ y & y_2 & y_3 \end{vmatrix} = (x_2 y_3 - x_3 y_2) + x(y_2 - y_3) + y(x_3 - x_2),$$

$$\Delta_2 = \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x & x_3 \\ y_1 & y & y_3 \end{vmatrix} = (x_3 y_1 - x_1 y_3) + x(y_3 - y_1) + y(x_1 - x_3).$$

Після того як отримані барицентричними координати крапки (x, y), значення функції в ній розраховується по формулі $z = \alpha z_1 + \beta z_2 + \gamma z_3$.
Існують добре розроблені методи гладкої інтерполяції функцій. Особливо часто при інтерполяції кривих і поверхонь використовуються сплайн-**функції**, які гладко "склеюються" з поліномів. Серед них варто виділити **кубічні сплайни**, які будуються з поліномів третього ступеня. Вони широко використовуються в інженерній геометрії завдяки простоті їхнього обчислення й інших корисних властивостей. Ми їх розглянемо докладніше в наступних главах.

4.4. Матриці

Для виконання перетворень векторів у просторі ми будемо використовувати матричний метод. **Матрицею** розмірності $n \times n$ називається таблиця чисел виду

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}.$$

Надалі будемо використовувати скорочений запис для матриці: $A = (a_{ij})$. Рядка матриці $A_i = (a_{i1}, a_{i2}, \dots, a_in)$ будемо називати вектор-**рядками** (оскільки кожна з них визначає вектор), а стовпці A^j - **векторами-стовпцями**. Матриці є ефективним інструментом для виконання перетворень на площині й у просторі. У цих випадках застосовуються матриці розмірності 2×2 й 3×3.

Спочатку введемо ряд операцій над матрицями й векторами.

Нехай задані матриці $A = (a_{ij})$ й $B = (b_{ij})$. **Сумою матриць** називається матриця $C = (c_{ij})$, елементами якої є $c_{ij} = a_{ij} + b_{ij}$.

Визначимо також операцію **множення матриці на число**. Результатом множення матриці $A = (a_{ij})$ на число α є матриця $B = (b_{ij})$, елементи якої $b_{ij} = \alpha a_{ij}$.

Добутком двох матриць $A = (a_{ij})$ і $B = (b_{ij})$ називається матриця $C = (c_{ij})$, елементи якої визначаються в такий спосіб:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

Добуток матриць некомутативно, тобто в загальному випадку $A \cdot B \neq B \cdot A$.

Попередні визначення ми вводили для матриць довільної розмірності. Наступні операції будуть пов'язані з векторами, і ми будемо мати на увазі, що $n = 2$ або $n = 3$. Нехай задана матриця $A = (a_{ij})$ й вектор $\overrightarrow{r} = (x_1, \dots, x_n)$. Результатом **множення матриці на вектор** є вектор $\overrightarrow{r}_0 = (x_1^0, \dots, x_n^0)$, координати якого обчислюються як скалярний добуток рядка матриці на вектор:

$$x_i^0 = \sum_{k=1}^{n} a_{ik} x_k \equiv (A_i \cdot \vec{r}).$$

Якщо матриця $B = (b_{ij})$ отримана з матриці $A = (a_{ij})$ шляхом заміни всіх вектор-рядків на вектори-стовпці, тобто $b_{ij} = a_{ji},\ i,\ j = 1, \ldots, n$, то неї називають транспонованою матрицею A й позначають A^T.

Аналогічним образом визначається **множення вектора на матрицю**, тільки в цьому випадку вектор скалярно множиться на вектор-рядка матриці. Матриця виду

$$E = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}$$

називається **одиничної** й має наступні властивості:

– $A \cdot E = E \cdot A = A$ для будь-якої матриці A.

– $\vec{r} = A \cdot \vec{r}$ для будь-якого вектора $\vec{r}$.

– Якщо для матриці A існує матриця B, така, що $A \cdot B = E$, то B називається зворотною матрицею до A і позначається A^{-1}. При цьому $A \cdot A^{-1} = A^{-1} \cdot A = E$, і для будь-якого вектора $\vec{r}$ одержуємо співвідношення: якщо $\vec{r}' = A \cdot \vec{r}$, те $\vec{r} = A^{-1} \cdot \vec{r}'$.

– Якщо для матриць A і B існують зворотні матриці, то існує й зворотна матриця для їхнього добутку й $(a \cdot B)^{-1} = B^{-1} \cdot A^{-1}$.

Завдяки операції множення матриці на вектор будь-яка матриця визначає перетворення в просторі, по якому кожному вектору зіставляється деякий інший по цілком певному законі.

Відзначимо, що для геометричних перетворень зручно використовувати матриці розмірністю на одиницю більше, ніж розмірність простору, але про це докладніше мова йтиме в наступній главі.

4.5. Геометричні перетворення (перенос, масштабування, обертання)

Геометричні об'єкти на площині й у просторі можна піддавати ряду різних перетворень. Найбільш уживаними в завданнях комп'ютерної графіки є:

– переміщення (паралельний перенос);

– зміна розмірів (масштабування);

– повороти навколо деякої крапки на площині або деякій осі в просторі (обертання).

Надалі ми часто будемо ототожнювати крапки простору з радіус-вектором, обумовленим цією крапкою.

Спочатку розглянемо перетворення на площині, або **двовимірні перетворення**.

Паралельний перенос об'єкта зводиться до переміщення всіх його крапок на те саме відстань d у тому самому напрямку, заданому певним вектором $\vec{v}$. Якщо цей вектор має довжину d, то операція переносу може бути реалізована

шляхом додавання всіх крапок об'єкта з вектором $\vec{v}$. Досить просто довести, що при такій операції зберігаються відстані між крапками й, як наслідок, кути між відрізками. Зрозуміло також, що відрізки прямих перейдуть у відрізки прямих. Тому при переносі багатокутника немає необхідності піддавати цієї операції нескінченна безліч крапок, досить просто перенести вершини, а потім з'єднати їхніми відрізками.

Масштабування об'єкта можна реалізувати шляхом множення координат всіх його крапок на деяке число. Нехай є крапки з координатами (x_1, y_1) й (x_2, y_2), над якими виконується таке перетворення. Результатом будуть нові крапки з координатами $(S_x x_1, S_y y_1)$ й $(S_x x_2, S_y y_2)$. Якщо $S_x = S_y = S_0$, то нескладно довести, що обидві крапки перемістяться уздовж прямих, що проходять через саму крапку й початок координат, тобто в напрямку свій же радіус-вектора (рис. 4.5). При цьому відстань між новими крапками буде в S_0 раз відрізнятися від колишнього, але кути між відрізками збережуться (це можна показати, якщо виразити косинус кута через скалярний добуток векторів). Ясно, що якщо коефіцієнт масштабування S_0 більше одиниці, що відповідає відрізок розтягується, а якщо менше, те стискується. Крім того, при такому перетворенні об'єкт зміщається.

У випадку, коли $S_x \neq S_y$, відстані між крапками зміняться нерівномірно, оскільки розтягання в горизонтальному й вертикальному напрямках будуть різними. Кути між відрізками також не збережуться (рис. 4.6).

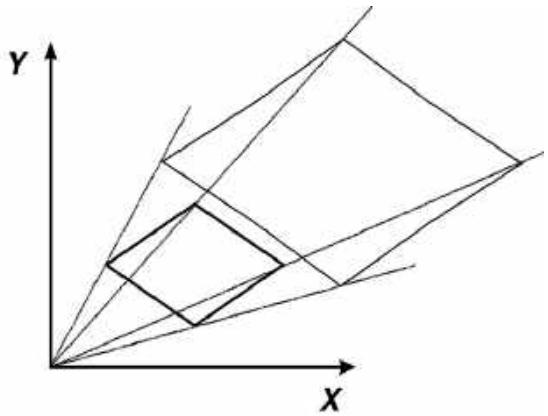


Рис. 4.5. Масштабування зі збереженням кутів

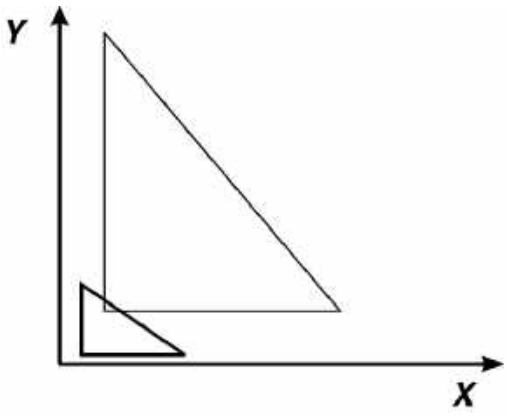


Рис. 4.6. Масштабування з перекручуванням кутів

Обертання в площині переміщають крапки по дузі кола, центр якої перебуває на початку координат. Розглянемо спочатку рух однієї крапки при повороті на кут α (позитивним є напрямок проти годинникової стрілки), тобто поворот радіус-вектора на кут (рис. 4.7). Нехай крапка розташовувалася на відстані r від початку координат, а її радіус-вектор становив кут β з віссю абсцис. Тоді координати крапки визначаються формулами $x = r\cos\beta, \quad y = r\sin\beta.$

Після повороту вектор буде становити кут $\beta + \alpha$, а нові координати крапки будуть визначатися співвідношеннями

$$x' = r\cos(\beta+\alpha) = r\cos\beta\cos\alpha - r\sin\beta\sin\alpha = x\cos\alpha - y\sin\alpha$$
$$y' = r\sin(\beta+\alpha) = r\sin\beta\cos\alpha + r\cos\beta\sin\alpha = x\sin\alpha + y\cos\alpha \quad (4.7)$$

Можна показати, що при такому перетворенні зберігаються відстані між крапками, а отже, і кути між відрізками.

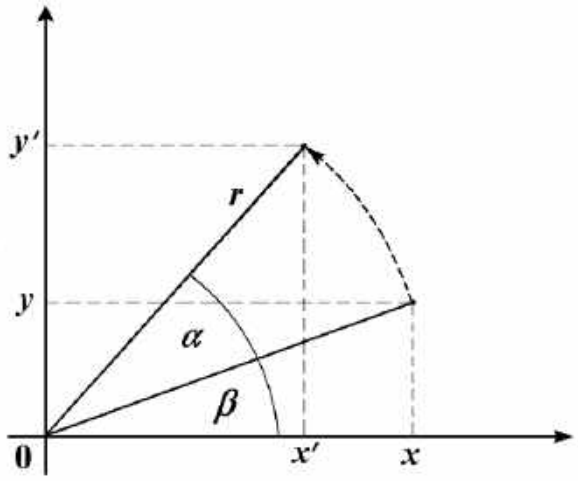


Рис. 4.7. Поворот на площині

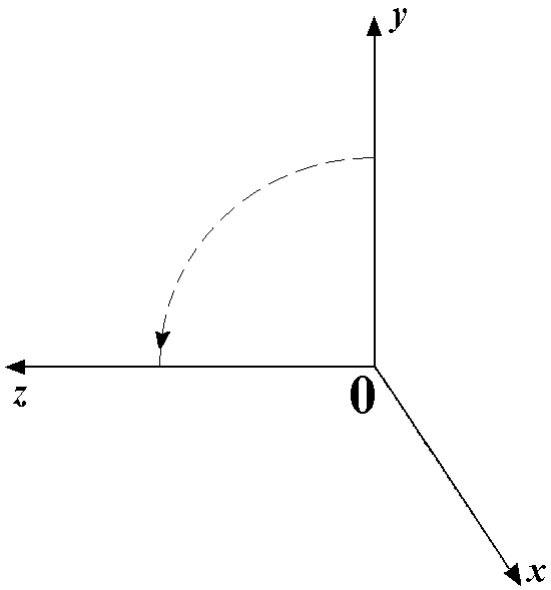


Рис. 4.8. Поворот у просторі

У випадку тривимірного простору міркування, що стосуються переносу й масштабування, повністю аналогічні, тільки вони поширюються на третю координату крапок. З обертанням же інша справа, оскільки тут обертовий рух є переміщення уздовж поверхні сфери й поворот на якийсь кут щодо крапки не можна визначити однозначно. Але переміщення з однієї крапки сфери в іншу завжди можна здійснити послідовністю поворотів щодо осей координат, тому виведемо формули для цих трьох обертань.

При повороті щодо осі OX на кут α у всіх крапок координата x залишається незмінної. Якщо дивитися на площину YOZ з боку кінця осі OX, то осі будуть розташовані так, як показано на рис. 4.8. Позитивним уважається поворот від осі OY до осі OZ. Якщо скористатися формулами для плоских поворотів, то координати y' й z' нової крапки визначаються вираженнями

$$y' = y\cos\alpha - z\sin\alpha$$
$$z' = y\sin\alpha + z\cos\alpha.$$

Формули повороту щодо осі OZ повністю збігаються з тими, які минулого виведені для плоского випадку, а поворот щодо осі OY виглядає так:

$$x' = x\cos\alpha + z\sin\alpha$$
$$z' = -x\sin\alpha + z\cos\alpha.$$

У всіх цих формулах варто звернути увагу на знаки, тому що вони залежать від того, який поворот уважається позитивним (у цьому випадку ми маємо справу із правою трійкою базисних векторів).

Перетворення масштабування й повороту на площині й у просторі можна виразити за допомогою матриць. Якщо задані коефіцієнти масштабування

S_x, S_y, S_z, то перетворення крапки здійснюється за допомогою множення матриці на її радіус-вектор,

$$\vec{r}' = S \cdot \vec{r} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & s_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} S_x x \\ S_y y \\ S_z z \end{pmatrix}. \tag{4.8}$$

Двовимірний випадок виглядає подібним же чином.

Поворот на площині можна здійснити за допомогою матриці

$$R = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}. \tag{4.9}$$

І нарешті, повороти в просторі щодо осей координат можна виконати за допомогою трьох матриць обертання

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}, \quad R_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix},$$
$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.10}$$

Неважко перевірити, що для матриць обертання справедливе співвідношення

$R(-\alpha) = R^{(}-1)(\alpha)$

Для виконання послідовних поворотів навколо осей на кути α, β, γ можна створити матрицю перетворення шляхом перемножування трьох матриць:

$R = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha).$

Використання цієї матриці дасть помітну економію в обчисленнях у порівнянні з послідовними множеннями на кожну із трьох матриць обертання.

4.6. Перехід в іншу систему координат

Ми розглянули перетворення геометричних об'єктів, заданих у певній декартовій системі координат. Але в багатьох випадках зручно розглядати ті ж об'єкти в іншій системі координат, оскільки їхній опис може стати більше простим. Найпростіший приклад – завдання координат паралелепіпеда: простіше всього це зробити в системі координат, сполученої з однієї з його вершин з осями, спрямованими уздовж ребер. У зв'язку із цим зупинимося на питанні, як зміняться координати крапки при переході від однієї декартової системи координат до іншої.

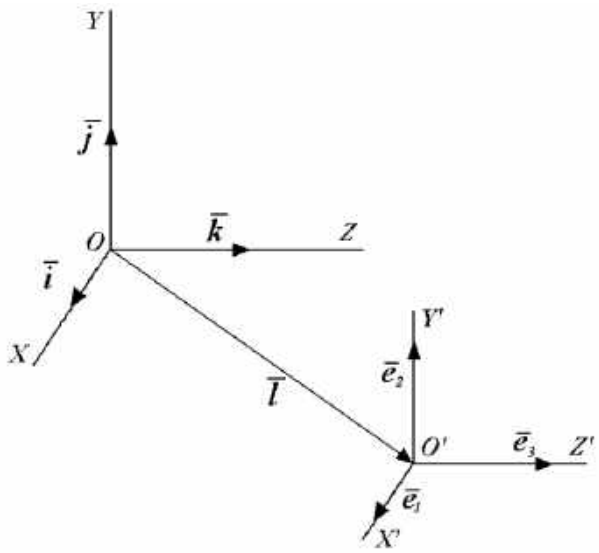


Рис. 4.9. Дві системи координат у просторі

Нехай одиничні орти першої системи координат позначаються $\vec{i}, \vec{j}, \vec{k}$, а осі координат – OX, OY, OZ. Уведемо ще одну систему координат, одиничні орти якої позначимо $\vec{e}_1, \vec{e}_2, \vec{e}_3$, а осі координат – $O'X', O'Y', O'Z'$. Ця система має свій початок координат і свої напрямки осей. Уважаємо, що в обох системах координат орти утворять праву трійку (рис. 4.9).

Спочатку розглянемо ситуацію, коли крапка O' збігається із крапкою O. Вектори $\vec{e}_1, \vec{e}_2, \vec{e}_3$ можна задати в першій системі координат, розклавши їх по векторах $\vec{i}, \vec{j}, \vec{k}$:

$$\left.\begin{aligned} \vec{e}_1 &= e_x^1\vec{i} + e_y^1\vec{j} + e_z^1\vec{k} \\ \vec{e}_2 &= e_x^2\vec{i} + e_y^2\vec{j} + e_z^2\vec{k} \\ \vec{e}_3 &= e_x^3\vec{i} + e_y^3\vec{j} + e_z^3\vec{k} \end{aligned}\right\}.$$

Якщо в першій системі крапка $\vec{M}$ має координати (x, y, z), а в другій системі – (x', y', z'), те, мабуть, $x'\vec{e}_1 + y'\vec{e}_2 + z'\vec{e}_2 = x\vec{i} + y\vec{j} + z\vec{k}$.

Множачи скалярно це співвідношення на вектори $\vec{e}_1, \vec{e}_2, \vec{e}_3$, одержимо зв'язок між значеннями координат у різних системах:

$$\left.\begin{aligned} x' &= e_x^1 x + e_y^1 y + e_z^1 z \\ y' &= e_x^2 x + e_y^2 y + e_z^2 z \\ z' &= e_x^3 x + e_y^3 y + e_z^3 z \end{aligned}\right\}.$$

Ці співвідношення можна записати в матричному виді

$$(x'//y'//z') = (e_x^1 \quad e_y^1 \quad e_z^1//e_x^2 \quad e_y^2 \quad e_z^2//e_x^3 \quad e_y^3 \quad e_z^3) \cdot (x//y//z \qquad (4.11)$$

або у векторному записі $\vec{M}' = A \cdot \vec{M}$.

Припустимо, що друга система координат отримана з першої шляхом повороту на кут φ щодо осі OY. Тоді

$$\vec{e}_1 = R_y(\varphi)\cdot\begin{pmatrix}1\\0\\0\end{pmatrix} = \begin{pmatrix}\cos\varphi\\0\\-\sin\varphi\end{pmatrix}, \quad \vec{e}_2 = R_y(\varphi)\cdot\begin{pmatrix}0\\1\\0\end{pmatrix} = \begin{pmatrix}0\\1\\0\end{pmatrix},$$
$$\vec{e}_3 = R_y(\varphi)\cdot\begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}\sin\varphi\\0\\\cos\varphi\end{pmatrix},$$

отже

$$A = \begin{pmatrix}\cos\varphi & 0 & -\sin\varphi\\0 & 1 & 0\\\sin\varphi & 0 & \cos\varphi\end{pmatrix} = R_y(-\varphi) = R_y^{-1}(\varphi).$$

Таким чином, при поворотах системи координат нові координати крапок виходять шляхом множення матриці повороту на протилежний кут на вектор вихідних координат.

Якщо нова система координат отримана зі старої шляхом зрушення на вектор $\vec{l} = (l_x, l_y, l_z)$, то очевидно, що нові координати крапки задаються формулами $x' = x - l_x, \quad y' = y - l_y, \quad z' = z - l_z.$

Тепер можна розглянути композицію двох перетворень системи координат – переносу й обертання. Тоді координати крапок перетворяться по формулі

$$\vec{M}' = A\cdot(\vec{M}' - \vec{l}). \tag{4.12}$$

4.7. Завдання обертання щодо довільної осі

Обертання щодо довільної осі також можна реалізувати за допомогою множення матриці на вектор, але попередньо цю матрицю треба побудувати. Припустимо, що пряма проходить через початок координат і задана одиничним вектором $\vec{l} = (l_x, l_y, l_z)$, і потрібно виконати поворот крапки на кут α щодо її. Для цього скористаємося наступним алгоритмом:

1. Сполучимо пряму з віссю OZ за допомогою повороту системи координат щодо осі OX на кут φ, а потім повороту щодо осі OY на кут ψ.
2. Виконаємо поворот щодо осі OZ на кут α.
3. Виконаємо повороти системи спочатку щодо осі OY на кут $-\psi$, а потім щодо осі OX на кут $-\varphi$ (у зворотному порядку стосовно перших поворотів), тим самим повертаючи її у вихідне положення.

Підсумкова матриця перетворення, таким чином, є добутком декількох матриць, а саме $R = R_x(\varphi)\cdot R_y(\psi)\cdot R_z(\alpha)\cdot R_y(-\psi)\cdot R_x(-\varphi).$

Матриці $R_y(-\psi), R_x(-\varphi)$ є матрицями перетворення координат при поворотах системи координат, як було показано в попередньому розділі. Визначимо спочатку кут φ, що є кутом між віссю OZ і його проекцією вектора $\vec{l}$ на площину YOZ $\vec{l}_1 = (0, l_y, l_z)$. Нехай $d = \sqrt{l_y^2 + l_z^2}$ – довжина цієї проекції. Тоді $\cos\varphi = \frac{l_z}{d}$, $\sin\varphi = -\frac{l_y}{d}$, (синус негативний, оскільки поворот іде

від осі OZ до осі OY, тобто в негативному напрямку). Після повороту системи координат новими координатами вектора $\overrightarrow{l}$ будуть $l_x, 0, d$. Кут ψ – це кут між векторами $\overrightarrow{l}_1$ й $\overrightarrow{l}$, тому $\cos\psi = d, \sin\psi = l_x$. Тепер ми можемо виписати вид матриць перетворення координат для кожного кроку алгоритму, зважаючи на те, що матриці перетворення координат при повороті системи координат оборотні стосовно відповідних матриць обертання:

$$R_x(\varphi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{l_z}{d} & -\frac{l_y}{d} \\ 0 & \frac{l_y}{d} & \frac{l_z}{d} \end{pmatrix}, \quad R_y(\psi =) \begin{pmatrix} d & 0 & -l_x \\ 0 & 1 & 0 \\ l_x & 0 & d \end{pmatrix},$$

$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Неважко переконатися, що послідовне множення матриць $R_x(-\varphi)$ і $R_y(-psi)$ на вектор $\overrightarrow{l}$ дадуть у результаті вектор $(0,0,1)$, тобто цей вектор дійсно стане віссю аплікат.

Залишається тільки виписати остаточний вид матриці R (для скорочення запису введемо наступні позначення: $c = \cos\alpha,\ s = \sin\alpha,\ c_1 = 1 - \cos\alpha$):

$$R = \begin{pmatrix} l_x^2 + c & l_x l_y c_1 - l_z s & l_x l_z c_1 + l_y s \\ l_x l_y c_1 + l_z s & l_y^2 c_1 + c & l_y l_z c_1 - l_x s \\ l_x l_z c_1 - l_y s & l_x l_z c_1 + l_x s & l_z^2 c_1 + c \end{pmatrix}. \qquad (4.13)$$

Нагадаємо, що l_x, l_y, l_z є напрямними косинусами прямої, щодо якої виконується поворот. Неважко переконатися, що якщо як осі обертання взяти осі координат, то ми в точності одержимо формули (4.10).

4.8. Питання й вправи

1. Дайте визначення декартової системи координат.
2. Що таке вектор?
3. Які вектори вважаються рівними?
4. Які вектори називаються лінійно незалежними?
5. Як виразити довжину вектора, використовуючи операцію скалярного добутку?
6. Як визначити косинус кута між векторами, використовуючи операцію скалярного добутку?
7. Доведіть, що векторний добуток задовольняє співвідношенню $\alpha[\overrightarrow{r}_1 \times \overrightarrow{r}_2] = [\overrightarrow{r}_1 \times (\alpha\overrightarrow{r}_2)]$.
8. Як з довільного вектора $\overrightarrow{r}$ одержати одиничний вектор, що збігається з ним по напрямку? (Ця операція називається **нормування вектора**).
9. Яке максимальне число лінійно незалежних векторів у просторі?
10. Що таке орти?
11. Як побудувати параметричне рівняння прямої, що проходить через дві задані крапки площини або простору?

12. Доведіть, що якщо у формулі (4.7) замінити координати (x_1, y_1, z_1) координатами будь-якої іншої крапки площини, то рівняння буде описувати ту ж саму площину. **Вказівка**: візьміть довільну крапку, що задовольняє рівнянню (4.7), напишіть нове рівняння площини й покажіть, що будь-яка крапка другої площини належить першої й навпаки.
13. У яких випадках промінь із площиною не перетинаються?
14. У яких випадках промінь перетинає сферу тільки в одній крапці?
15. Виходячи з визначення множення матриці на вектор, доведіть, що для будь-яких двох векторів $\vec{r}_1, \vec{r}_2$ і будь-якої матриці A справедливе співвідношення $A \cdot (\vec{r}_1 + \vec{r}_2) = A \cdot \vec{r}_1 + A \cdot \vec{r}_2$.
16. Доведіть, що для будь-якого вектора $\vec{r}$, числа α й матриці A справедлива співвідношення $A \cdot (\alpha \vec{r}) = \alpha(A \cdot \vec{r}) = (\alpha A) \cdot \vec{r}$.
17. При якій умові масштабування зберігає кути між відрізками?
18. Яку траєкторію описують крапки об'єкта при повороті?
19. Навколо чого здійснюється поворот на площині?
20. Навколо чого здійснюється поворот у просторі?
21. Які кроки виконуються в алгоритмі повороту щодо довільної осі в просторі?
22. Доведіть, що якщо матриця A є матрицею повороту, те $A \cdot A^T = E$.

Тема 5. ПОДАННЯ ГЕОМЕТРИЧНОЇ ІНФОРМАЦІЇ

Геометричні примітиви. Системи координат: світового, об'єктна, спостерігача й екранна. Однорідні координати. Завдання геометричних перетворень в однорідних координатах за допомогою матриць

5.1. Геометричні примітиви

Під геометричними примітивами розуміють той базовий набір геометричних фігур, що лежить в основі всіх графічних побудов, причому ці фігури повинні утворювати "базис" у тому розумінні, що жоден із цих об'єктів не можна побудувати через інші. Однак питання про те, що включати в набір геометричних примітивів, не можна вважати остаточно вирішеним у комп'ютерній графіці. Наприклад, кількість примітивів можна звести до якогось мінімуму, без якого не можна обійтися, і цей мінімум зводиться до апаратно реалізованих графічних об'єктів. У цьому випадку базисний набір обмежується відрізком, багатокутником і набором літер (символів).

Інша точка зору полягає в тому, що в набір примітивів необхідно включити гладкі криві різного роду (кола, еліпси, криві Безьє), деякі класи поверхонь і навіть суцільні геометричні тіла. У якості тривимірних геометричних примітивів у такому випадку пропонуються просторові криві, паралелепіпеди, піраміди, еліпсоїди. Але якщо такий розширений набір примітивів пов'язаний з апаратною реалізацією, то виникає проблема перенесення програмних додатків з одного комп'ютера на іншій, оскільки така апаратна підтримка існує далеко не на всіх графічних станціях. Крім того, при створенні тривимірних геометричних примітивів програмісти зіштовхуються із проблемою їхнього математичного опису, а також розробки методів маніпулювання такими об'єктами, оскільки ті типи об'єктів, які не потрапили в список базових, треба вміти наближати за допомогою цих примітивів.

У багатьох випадках для апроксимації складних поверхонь використовуються багатогранники, але форма граней може бути різної. Просторовий багатокутник із числом вершин більше трьох не завжди буває плоским, а в цьому випадку алгоритми зображення багатогранників можуть привести до некоректного результату. Тому програміст повинен сам подбати про те, щоб багатогранник був описаний правильно. У цьому випадку оптимальним виходом з положення є використання трикутників, оскільки трикутник завжди є плоским. У сучасній графіці це, мабуть, найпоширеніший підхід.

Але існує й альтернативний напрямок, що називається **конструктивною геометрією тіл.** У системах, що використовують цей підхід, об'єкти будуються з об'ємних примітивів з використанням теоретико- множинних операцій (об'єднання, перетинання).

Будь-яка графічна бібліотека визначає свій набір примітивів. Так, наприклад, широко розповсюджена інтерактивна система тривимірної графіки OpenGL включає до списку своїх примітивів крапки (вершини), відрізки, ламані, багатокутники (серед яких особливо виділяються трикутники й

чотирикутники), смуги (групи трикутників або чотирикутників із загальними вершинами) і шрифти. Крім того, у неї входять і деякі геометричні тіла: сфера, циліндр, конус і ін.

Зрозуміло, що для зображення таких примітивів повинні бути розроблені ефективні й надійні алгоритми, оскільки вони є конструктивними елементами. Історично зложилося так, що перші дисплеї були векторними, тому базовим примітивом був відрізок. Але, як уже було відзначено в першому розділі нашого курсу, найперша інтерактивна програма Sketchpad А.Сазерленда в якості одного із примітивів мала прямокутник, після чого цей об'єкт уже традиційно входив у різні графічні бібліотеки.

Тут ми розглянемо такі примітиви, як **вершина**, **відрізок**, **воксель** і моделі, що будуються на їхній основі, а також **функціональні моделі.**

5.2. Полігональні моделі

Для цих просторових моделей використовуються як примітиви вершини (крапки в просторі), відрізки прямих (вектори), з яких будуються **полілінії, полігони** й **полігональні поверхні**. Головним елементом опису є вершина, всі інші є похідними. У тривимірної декартовій системі координати вершини визначаються своїми координатами (x,y,z), лінія задається двома вершинами, полілінія являє собою незамкнуту ламану лінію, полігон – замкнуту ламану лінію. Полігон моделює плоский об'єкт і може описувати плоску грань об'ємного об'єкта. Кілька граней становлять цей об'єкт у вигляді полігональної поверхні – багатогранник або незамкнута поверхня ("полігональна сітка").

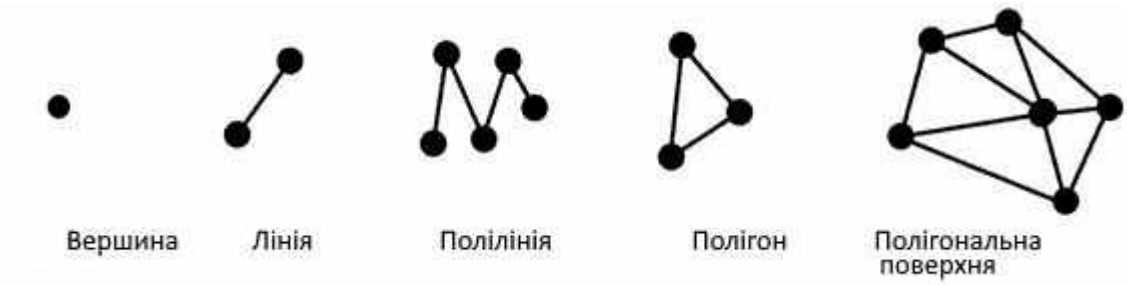


Рис. 5.1. Полігональні моделі

У сучасній комп'ютерній графіці векторно-полігональна модель є найпоширенішою. Вона застосовується в системах автоматизованого проектування, комп'ютерних іграх, тренажерах, ГИС, САПР і т.д. Переваги цієї моделі полягають у наступному:

- Зручність масштабування об'єктів.
- Невеликий обсяг даних для опису простих поверхонь.
- Апаратна підтримка багатьох операцій.

До числа недоліків полігональних моделей можна віднести те, що алгоритми візуалізації виконання топологічних операцій (наприклад, побудова перетинів) досить складні. Крім того, апроксимація плоскими гранями

приводить до значної погрішності, особливо при моделюванні поверхонь складної форми.

5.3. Воксельні моделі

Воксельна модель – це подання об'єктів у вигляді тривимірного масиву об'ємних (кубічних) елементів. Сама назва "воксель" складено із двох слів: volume element. Так само як і піксель, воксель має свої атрибути (колір, прозорість і т.п.). Повна прозорість вокселя означає порожнечу у відповідній крапці обсягу. Чим більше вокселів у певному обсязі й менше їхній розмір, тим точніше моделюються тривимірні об'єкти.

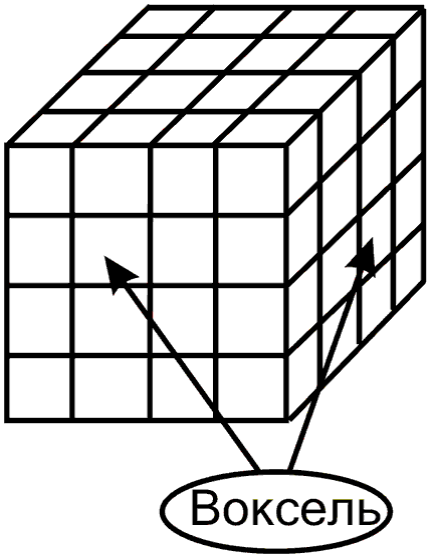


Рис. 5.2. Воксельна модель

Позитивними рисами воксельної моделі є:

– Можливість представляти внутрішність об'єкта, а не тільки зовнішній шар; проста процедура відображення об'ємних сцен.

– Просте виконання топологічних операцій; наприклад, щоб показати перетин просторового тіла, досить вокселі зробити прозорими.

До її недоліків ставляться:

– Велика кількість інформації, необхідне для подання об'ємних даних.

– Значні витрати пам'яті, що обмежують розв'язну здатність, точність моделювання.

– Проблеми при збільшенні або зменшенні зображення; наприклад, зі збільшенням погіршується роздільна здатність зображення.

5.4. Поверхні вільних форм (функціональні моделі)

Характерною рисою пропонованого способу завдання поверхонь є те, що основним примітивом тут є поверхня другого порядку – **квадрик**. Він визначається за допомогою речовинної безперервної функції трьох змінних (x, y, z)у вигляді нерівності $F(x, y, z) \geq 0.$

Таким чином, квадрик є замкнута підмножина евклідова простору, всі крапки якого задовольняють зазначеній нерівності. Рівняння $F(x,y,z)=0$ описує **границю** цієї безлічі. Безліч крапок, що задовольняють нерівності $F(x,y,z)<0,$ утворить **зовнішню область** квадрика.

Вільна форма – це довільна поверхня, що володіє властивостями гладкості, безперервності й нерозривності. На базі квадриків будуються вільні форми, які описують функціональні моделі. Вільна форма, побудована на цих принципах, має ряд достоїнств, до яких, у першу чергу, треба віднести наступні:

– Легка процедура розрахунку координат кожної крапки.
– Невеликий обсяг інформації для опису досить складних форм.
– Можливість будувати поверхні на основі скалярних даних без попередньої тріангуляції.

Цей підхід буде більш докладно викладений у наступних главах.

У нашім курсі передбачається розглянути растрові алгоритми для зображення таких геометричних примітивів, як відрізки, багатокутники, кола й еліпси. Але спочатку ми займімось тим геометричним апаратом, що дозволить адекватно описувати об'єкти в просторі, працювати з ними й формувати зображення.

5.5. Системи координат: світового, об'єктна, спостерігача й екранна

Однієї з розповсюджених завдань комп'ютерної графіки є зображення двовимірних графіків у деякій системі координат. Ці графіки призначені для відображення залежності між змінними, заданої за допомогою функції. Наприклад, у другому розділі справжнього курсу наведений ряд графіків, що характеризують сприйняття світла оком людини. Щоб одержати такий графік, прикладна програма повинна описати різні вихідні примітиви (крапки, лінії, ланцюжка символів), указавши їхнє місце розташування й розміри в прямокутній системі координат. Одиниці виміру, у яких задаються ці об'єкти, залежать від їхньої природи: зміна температури, наприклад, можна відображати в градусах за годину, переміщення тіла в просторі – у кілометрах у секунду, і т.д. Ці прикладні (або орієнтовані на користувача) координати дозволяють задавати об'єкти у двовимірному або тривимірному світі користувача, і їх прийнято називати **світовими координатами.**

Зображення тривимірних об'єктів сполучено із цілим рядом завдань. Насамперед треба пам'ятати, що зображення є плоским, тому треба домогтися адекватної передачі візуальних властивостей предметів, дати досить наочне подання про глибину. Надалі групи тривимірних об'єктів, призначених для зображення, будемо називати **просторовою сценою**, а її двовимірне зображення – **образом.**

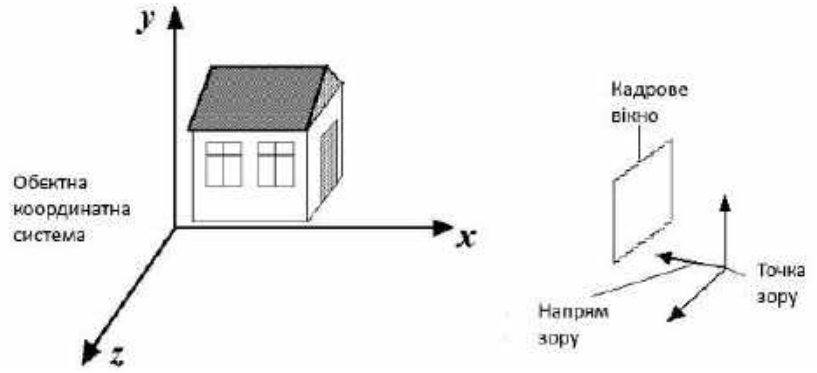


Рис. 5.3. Об'єктна система координат і система координат спостерігача

Як і у випадку із двовимірними об'єктами, першим кроком побудови є введення інформації про об'єкти. Сцена займає якесь певне місце в просторі, а її опис прив'язується до тривимірного декартовой системі координат, пов'язаної з нею, – **об'єктній координатній системі**. Координати об'єктів, що становлять сцену, визначаються на основі їхніх реальних розмірів і взаємного розташування. Залежно від крапки, з якої розглядається сцена, можна одержати безліч різних її образів. Якщо побудовано досить багато таких образів, то по них можна відновити об'ємну структуру предмета. Вибір крапки й напрямку зору теж можна описати математично, увівши декартову **систему координат спостерігача**, початок який перебуває в крапці огляду, а одна з осей збігається з напрямком зору (рис. 5.3). Перехід від об'єктних координат до координат спостерігача математично реалізується так, як це було описано в третьому розділі. На цьому етапі перетворень зберігаються реальні розміри об'єктів.

Видимий образ формується на деякій площині, що надалі будемо називати **картинною площиною**. Способи перетворення тривимірного об'єкта у двовимірний образ (**проекції**) можуть бути різними. Так чи інакше, але отриманий образ також повинен бути описаний у деякій двовимірній системі координат. Залежно від способу його одержання реальні розміри образи також можуть бути різні. Різні види проектування будуть докладно розглянуті в наступних главах.

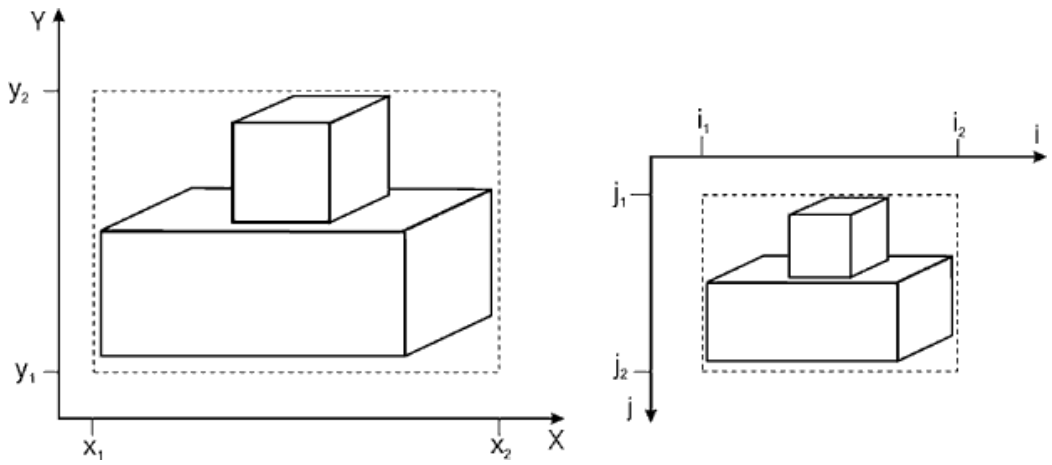


Рис. 5.4. Картинна площина й екран

Оскільки нашою кінцевою метою є одержання зображення на екрані, то перенесення образа супроводжується зміною масштабу відповідно до розмірів екрана. Звичайно початком координат у системі координат образа вважається лівий нижній кут аркуша із зображенням. На екрані дисплея початок координат традиційно перебуває в лівому верхньому куті. Відображення малюнка з картинної площини на екран повинне вироблятися з мінімальним перекручуванням пропорцій, що саме по собі вносить обмеження на область екрана, займану малюнком. Зміна масштабу повинне здійснюватися зі збереженням пропорцій області (рис. 5.4).

Об'єкти в системі координат картинної площини задаються в яких- або одиницях виміру, причому масштаб однаковий по обох осях координат. На екрані одиницею виміру є піксель, якому варто розглядати як прямокутний, тому масштаби по горизонтальній і вертикальній осях можуть бути різні, що необхідно враховувати при завданні коефіцієнтів масштабування.

Розглянемо ситуацію, коли зображення займає на картинній площині прямокутну область $x_1 \le x_2,\ y_1 \le y \le y_2$. При відображенні малюнка на екран кожна крапка вихідного прямокутника з координатами (x, y) перейде в деяку крапку із цілочисловими координатами (i, j). Уведемо позначення:

$$\Delta x = x_2 - x_1, \quad \Delta y = y_2 - y_1, \quad \Delta i = i_2 - i_1, \quad \Delta j = j_2 - j_1,$$

$$S_x = \frac{\Delta i}{\Delta x}, \quad S_y = \frac{\Delta j}{\Delta y}$$

(передбачається, що зображення займе на екрані прямокутник $i_1 \le i \le i_2,\ j_1 \le j_2$). Визначимо перетворення координат образа (x, y) в екранні координати (i, j) формулами $i = i_1 + S_x(x - x_1), \quad j = j_2 - S_y(y - y_1).$

Ясно, що при такому відображенні прямокутна область образа в точності перейде у відповідний екранний прямокутник, як показано на малюнку. Тепер треба визначити сам екранний прямокутник так, щоб його пропорції відповідали прямокутнику образа, тобто

$$\frac{\Delta x}{\Delta y} = \frac{\Delta i_x}{\Delta j \cdot l_y} \equiv \kappa \frac{\Delta i}{\Delta j},$$

де l_x, l_y - горизонтальний і вертикальний розмір одного пікселя. Ці параметри легко встановити, знаючи розміри екрана й дозвіл. Звідси одержуємо $\Delta j = \kappa \cdot S_x \cdot \Delta y, \quad S_y = \kappa \cdot S_x$

Тепер досить задати на екрані початок відліку й горизонтальний розмір вікна, а інші параметри легко обчислюються.

5.6. Однорідні координати. Завдання геометричних перетворень в однорідних координатах за допомогою матриць

У попередній главі описувалися геометричні перетворення на площині й у просторі, а також було показано, як можна використовувати апарат матриць для таких завдань. Для перетворень на площині застосовувалися двовимірні вектори й матриці розмірністю 2×2. У просторі, відповідно, із цією же метою використовувалися тривимірні вектори й матриці 3×3. Але такий підхід не дозволяє задавати за допомогою матриць перетворення переносу й проекції. У

зв'язку із цим у проективній геометрії був розроблений апарат, що дозволяє уніфікувати всі геометричні перетворення шляхом введення так званих **однорідних координат**.

Для пояснення такого підходу спочатку розглянемо випадок двовимірного простору. Кожна крапка площини з координатами (x, y) може одночасно розглядатися як крапка тривимірного простору з координатами $(x, y, 1)$, тобто як крапка, що лежить на площині $z = 1$. З іншого боку, кожній крапці тривимірного простору (x, y, z) за умови $z \neq 0$ відповідає єдина крапка цієї ж площини $\left(\frac{x}{z}, \frac{y}{z}, 1\right)$. При цьому виходить, що кожній крапці площини $(x, y, 1)$ відповідає пряма, що проходить через початок координат, тобто встановлюється взаємно однозначна відповідність між крапками площини й безлічами (tx, ty, t), $-\infty < t < \infty$, $t \neq 0$.

Якщо тепер розглядати крапку площини як приналежному **тривимірному** простору, то її **двовимірні** перетворення можна буде описувати за допомогою матриць 3×3, причому можна буде задавати таким способом не тільки повороти й масштабування, але й зрушення й проекції (як ортографічні, так і центральні).

Поворот на кут α відносно початку координат можна здійснити за допомогою нової матриці повороту:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Операція масштабування може бути записана у вигляді

$$\begin{pmatrix} ax \\ by \\ 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Перенос на вектор (x_0, y_0) також можна задати за допомогою матриці:

$$\begin{pmatrix} x + x_0 \\ y + y_0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Проекції крапки на осі координат визначаються за допомогою матриць проекції:

$$P_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad P_y = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Перейдемо тепер до тривимірного простору. Кожній крапці (x, y, z) будемо ставити у відповідність крапку чотирьохмірного простору $(x, y, z, 1)$, а для виконання основних перетворень будемо використовувати матриці розмірністю 4×4. Будуються вони зовсім аналогічно тому, як це робилося у двовимірному випадку. Матриця зрушення на вектор (x_0, y_0, z_0) має вигляд

$P = \begin{pmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, матриця масштабування теж очевидним образом будується із тривимірної матриці:

$$S = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Проекції крапок на координатні площини здійснюються за допомогою матриць (більш докладно проекції і їхніх видів будуть розглянуті пізніше):

$$P_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad P_{yz} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad P_{zx} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Множення цих матриць на вектор приводить до того, що обнулюється одна з координат, і в результаті одержуємо проекцію крапки на відповідну площину.

Матриця повороту щодо осі OX на кут α виглядає в такий спосіб:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Звідси легко зрозуміти, як будуються матриці повороту щодо інших координатних осей, а також матриця повороту щодо довільної осі. Просто беремо матриці, побудовані в третьому розділі, і розширюємо їх шляхом додавання вже відомих одиничних вектора-рядка й вектора- стовпця:

$$\begin{pmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Шляхом об'єднання наведених елементарних перетворень можна побудувати й більше складні. У третьому розділі ми використовували добуток простих матриць обертання для побудови матриці повороту щодо довільної осі. Приведемо один приклад.

Нехай у просторі задані два відрізки – $\overline{AB}$ і $\overline{CD}$. Будемо будувати матрицю перетворення, що переводить перший відрізок у другий. Це перетворення розкладемо на наступні елементарні дії.

1. Зрушення, що переміщає крапку $A = (A_x, A_y, A_z)$ в крапку $C = (C_x, C_y, C_z)$.
2. Зрушення початку координат у цю же крапку.
3. Якщо відрізки неколінеарні:

– будується вектор нормалі до площини, у якій лежать відрізки (для цього можна використовувати векторний добуток вихідних векторів);

– поворот щодо вектора нормалі, що сполучає два відрізки по напрямку (кут повороту можна визначити за допомогою скалярного добутку вихідних векторів).

4. Масштабування з метою вирівнювання довжини відрізків.
5. Повернення початку координат у вихідну точку.

Кожне із цих перетворень реалізується за допомогою матриці, а повне перетворення можна виконати, використовуючи добуток матриць.

Використання матриць дуже зручно для виконання перетворень у просторі, хоча в деяких випадках це приводить до надлишкового числа виконуваних операцій. Наприклад, поворот однієї крапки в просторі щодо координатної осі OZ за допомогою матриць в однорідних координатах вимагає 16 операцій множення й 12 операцій додавання. У той же час він легко може бути виконаний за допомогою формул перетворення

$$x' = x\cos\alpha - y\sin\alpha$$
$$y' = x\sin\alpha + y\cos\alpha$$

тобто за допомогою всього лише чотирьох множень і одного додавання й одного вирахування. Операції зрушення також набагато більш економічно виконувати без використання матриць. Але коли мова йде про суперпозицію багатьох перетворень (як, наприклад, у випадку повороту щодо довільної осі), те доцільно застосовувати відповідну матрицю повороту. Ефективність матричного підходу дуже сильно зростає, якщо матричні операції реалізовані апаратно. Питання про те, у яких випадках використовувати матриці, а в яких ні, багато в чому залежить від можливостей обчислювальної техніки, рівня складності завдання й вимог до тимчасових характеристик процесу візуалізації.

5.7. Питання й вправи

1. Які геометричні об'єкти вважаються примітивами?
2. Які вимоги пред'являються до набору геометричних примітивів?
3. У якій програмі вперше як геометричний примітив використовувався прямокутник?
4. Що таке об'єктна система координат?
5. Що таке система координат спостерігача?
6. Чи відповідають розміри об'єктів у системі координат спостерігача їхнім реальним розмірам?
7. Що таке картинна площина?
8. Як називається операція переходу від тривимірної системи координат до двовимірного?
9. Що відбувається при перенесенні зображення з картинної площини на екран?
10. Чим відрізняються однорідні координати крапки від звичайних декартових координат?
11. З якою метою вводяться однорідні координати?
12. Скільки елементарних дій потрібно для сполучення двох відрізків у просторі?
13. Чи завжди використання матриць для виконання перетворень у просторі ефективніше, ніж інші способи їхньої реалізації?

Тема 6. АЛГОРИТМИ РАСТЕРИЗАЦІЇ ВІДРІЗКІВ, КІЛ І ЕЛІПСІВ

Введення в растеризацію кривих. Зображення відрізка із цілочисловими координатами кінців. Цифровий диференціальний аналізатор. Алгоритм Брезенхема. Алгоритм Кастла-Пітвея. Зображення відрізка з нецілочисловими координатами кінців. Зображення окружностей. Алгоритм Брезенхема. Зображення еліпсів. Побудова по неявній функції. Побудова шляхом стиску кола.

Цей розділ присвячений растеризації найпростіших геометричних об'єктів, таких як відрізки, кола й еліпси, на площині.

6.1. Введення в растеризацію кривих

Нехай у нас є деяка крива, і ми хочемо побудувати її зображення на растрових ґратах. Виникає питання: які з найближчих пікселів варто зафарбовувати? У даній і наступній лекціях ми розглянемо випадок побудови на монохромному растрі, коли можливі тільки два рівні інтенсивності зафарбування пікселя – "повністю зафарбований" або "повністю не зафарбований". Якщо ж припустимі кілька рівнів інтенсивності, то можна растеризувати більш акуратно, зменшуючи ефекти аліасинга (тобто ступінчастості).

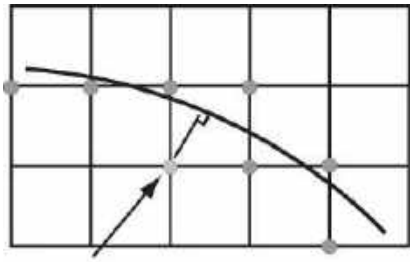


Рис. 6.1. Зображення кривих на растрі

Нехай (x_0, y_0) – фіксований піксель, а (x, y) – деякий інший піксель на площині. Тоді для визначення їхньої близькості вводяться наступні поняття:

1. 4-зв'язність
$|x-x_0|+ |y-y_0|=1$

2. 8-зв'язність
$\max\{ |x-x_0|+ |y-y_0|\}=1$

У подальших міркуваннях відстань будемо вважати заданим стандартною евклідовою метрикою[1)].

6.2. Зображення відрізка із цілочисловими координатами кінців

Нехай наш відрізок – це AB. Перейдемо від системи координат Oxy до Ax'y' (див. рис. 6.2, етап 1). Відрізок може лежати в кожному з 8 октантів, але завжди існують симетрії щодо осей, що розділяють ці октанти, симетрії визначаються матрицями

$$\begin{pmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{pmatrix}$$

і

$$\begin{pmatrix} 0 & \pm 1 \\ \pm 1 & 0 \end{pmatrix},$$

що дозволяють звести завдання до випадку відрізка, що лежить у першому октанті (приклад див. на рис. 6.2, етап 2, у ньому матриця має вигляд

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Назвемо такий випадок канонічним, надалі будуть розглянуті алгоритми для цього випадку. У канонічному випадку процес малювання 8-зв'язної лінії можна закодувати послідовністю виду: sdssd... (див. рис. 6.3), де

s – горизонтальний зсув;

d – діагональний зсув.

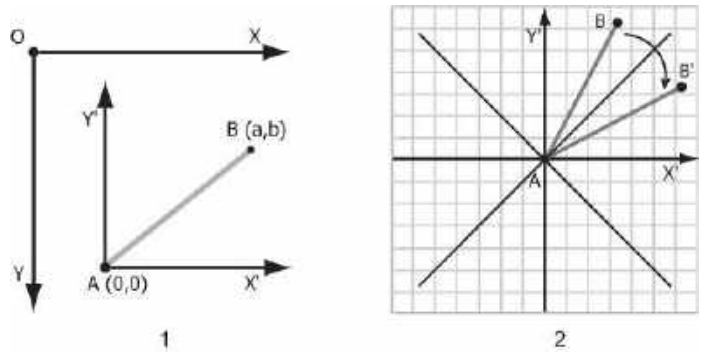


Рис. 6.2. Перехід до канонічного випадку у два етапи

Еквівалентно цієї послідовності можна зіставити бінарний код, де 0 відповідає s, а 1 відповідає d. Такий код для малювання відрізка $(0,0) \to (p,q), p > q \in N$ називається кодом Ротштейна [46] для $\frac{p}{q}$.

Нехай plot(x,y) – функція, що зафарбовує крапку растра з координатами (x,y).

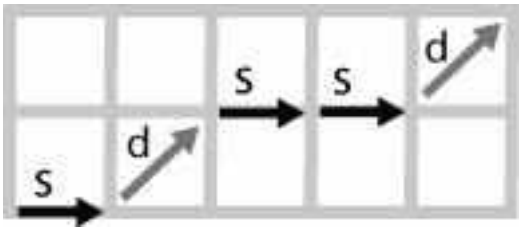


Рис. 6.3. Кодування зафарбовування відрізка (або код Ротштейна)

6.2.1. Цифровий диференціальний аналізатор

Алгоритм **Цифровий диференціальний аналізатор** (англ. DDA – Digital Differential Analyzer) будує 8-зв'язну лінію.

Для початку, нехай $P_1 = (0, 1)$; $P_2 = (1, 1)$. Для визначення того, який з пкселів, – P_1 або P_2, – варто зафарбувати, зрівняємо відстані до них. У силу подоби трикутників, утворених перетинанням відрізка, що малюється, прямій $x = 1$ і перпендикулярами з $P1$ і $P2$ на відрізок (див. рис. 6.4), досить зрівняти e (ординату перетинання відрізка с прямій $x = 1$) с. $\frac{1}{2}$ Далі, для наступного кроку алгоритм працює аналогічно з урахуванням зміни e – ординати перетинання відрізка з наступної вертикальної прямої $x = k, k \in N$.

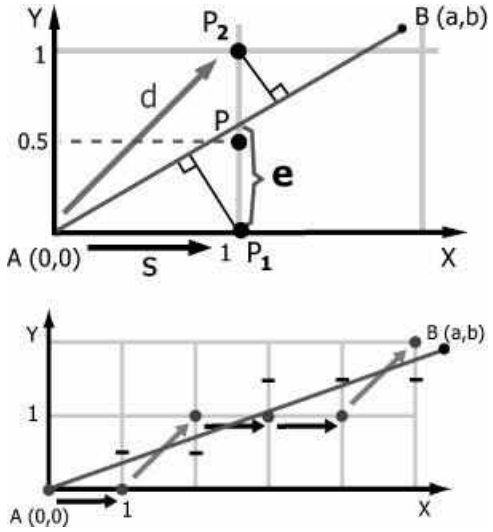


Рис. 6.4. Цифровий диференціальний аналізатор

```
// Координати кінців відрізка – (0,0) і (a,b)

e = b/a; // Поточна ордината
?e = b/a; // Збільшення ординати

// (x,y) – Координати поточної крапки
x = 0; y = 0;

while( x < a )
{
    plot(x, y);

    if( e > 1/2 )
    {
        // d : діагональний зсув
        x++; y++;

        // т.к. відбувся зсув по y на 1 нагору
        e += ?e - 1;
    }
```

```
    else
    {
        // s : горизонтальний зсув
        x++;
        e += ?e;
    }
}
```

Лістинг 6.1. Цифровий диференціальний аналізатор

Недоліком даного алгоритму є те, що він працює із числами із плаваючою крапкою.

6.2.2. Алгоритм Брезенхема

Брезенхем [16] модифікував алгоритм DDA, щоб він працював у цілих числах. Модифікуємо алгоритм у такий спосіб:

– зменшимо скрізь e на $\frac{1}{2}$, щоб порівнювати з 0;
– домножимо e і Δe на 2a: $e_0 = 2b - a$, $\Delta e = 2b$.

Приходимо до наступного алгоритму:

```
// Координати кінців відрізка - (0,0) і (a,b)
e = 2b - a;
?e = 2b;
?e = 2b - 2a;

x = 0; y = 0; // (x,y) – Координати поточної крапки

while(x < a)
{
    plot(x, y);

    if(e > 0)
    { // d : діагональний зсув
        x++; y++;
        e += ?e;
    }
    else
    { // s : горизонтальний зсув
        x++;
        e += ?e;
    }
}
```

Лістинг 6.2. Алгоритм Брезенхема для відрізка

Алгоритм Брезенхема був створений їм для виводу відрізків на цифрових інкрементальних графобудівниках, які могли здійснювати лише прості одиничні зрушення друкуючої головки. Подальша оптимізація може бути зроблена, якщо помітити, що відрізок симетричний відносно прямій, що

проходить перпендикулярно йому через його середину; у цьому випадку можна починати малювати відразу із двох кінців, що скоротить число ітерацій циклу в алгоритмі вдвічі.

6.2.3. Алгоритм Кастла-Пітвея

Цей алгоритм набагато менш ефективний з обчислювальної крапки зрені, чим алгоритм Брезенхема, однак має гарну математичну структуру. Він заснований на ідеї, схожої з відомим алгоритмом Евкліда знаходження Найбільшого Загального Дільника двох натуральних чисел [19].

Будемо працювати з рядками, що кодують послідовність зафарбування (тобто складаються із символів s і d, певних вище).

Визначимо дві операції над такими рядками:

– $\oplus$ – конкатенація рядків, наприклад $"ssds" \oplus "sddd - "ssdssddd"$

– $\sim$ – "переворот" рядка, наприклад $\sim ("ssdds") = "sddss"$

```
// Координати кінців відрізка – (0,0) і (a,b)
y = b;
x = a - b;
m1 = "s";
m2 = "d";

while( x \ne y )
{
    if( x > y )
    {
        x = x - y;
        m2 = m1 ⊕ ~ m2;
    }
    else
    {
        y = y - x;
        m1 = m2 ⊕ ~ m1;
    }
}
```

Лістинг 6.3. Алгоритм Кастла-Пітвея

Після завершення роботи алгоритму $m = m_2 \oplus \sim m_1$ задає потрібну послідовність зрушень. Доказ коректності роботи цього алгоритму ми опустимо через його громіздкість.

6.3. Зображення відрізка з нецілочисловими координатами кінців

Для відрізка з нецілочисловими координатами кінців будемо будувати відповідну 4-зв'язну лінію на растрі.

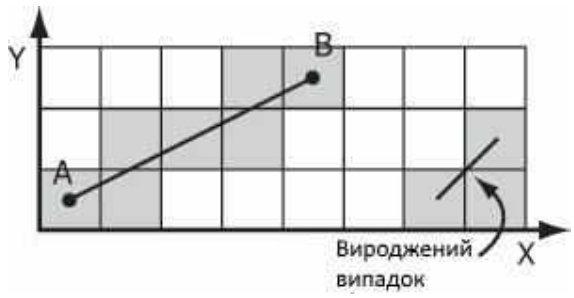


Рис. 6.5. Малювання відрізка з нецілочисловими координатами кінців

Існує два підходи.

1. Округлити координати кінців до цілочислених і скористатися алгоритмом для цілочислового случаючи. Недолік: може викликати істотні перекручування (особливо у випадку відрізків невеликої довжини).
2. Перейдемо до нашого канонічного випадку, що тепер характеризується тим, що відрізок лежить у першому октанті, але координати $A(x_A, y_A)$ в цьому випадку: $x_A \in [0,1), y_A \in [0,1)$. Параметризуємо наш відрізок стандартним образом:

$$(x, y) = A + t \cdot \frac{B - A}{c}, t \in [0, c],$$

де A і B – кінцеві крапки, c > 0 – якийсь масштабний коефіцієнт. Зробимо c досить більшим цілим числом, щоб зменшити помилки округлення. Тоді розглянемо

$$\Delta h = \frac{c}{x_B - x_A}$$

- збільшення t, при зрушенні на 1 піксель по x;

$$\Delta v = \frac{c}{y_B - y_A}$$

- збільшення t, при зрушенні на 1 піксель по y.

Будемо порівнювати поточні значення h і v, а потім, залежно від цього, робити крок по x або y і надавати відповідні збільшення h і v. Алгоритм закінчиться, коли h або v перевищить c.

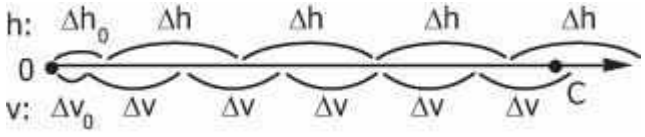


Рис. 6.6. Зміна параметрів h і v

```
x = 0; y = 0; // Канонічний випадок: початкова крапка
// лежить в [0, 1) ⊕ [0, 1)

/* Збільшення t, що відповідають зсувам від початкової
крапки до границь першого пікселя. */

h = ?h * (1 - x); // ?h0
```

```
v = ?v * (1 - y); // ?v0

while( (h < c) AND (v < c) )
{
      plot(x, y);

      if( h < v )
      {
          // Зрушення по горизонталі
          x++; h += ?h;
      }
      else if( h > v )
      {
          // Зрушення по вертикалі
          y++; v += ?v;
      }
      else
      {
          // h = v : Вироджений випадок (див. рис. 6.5)
          // малюємо довільний із двох можливих пікселів,
          // наприклад, верхній:
          plot(x,y+1);
          x++; y++;
          h += ?h; v += ?v;
      }
}
```

Лістинг 6.4. Алгоритм відображення відрізка з нецілочисловими координатами кінців

Зауваження. Наведений вище алгоритм легко узагальнюється на n-мірний випадок.

6.4. Зображення кіл

Для початку перейдемо до канонічної системи координат, у якій центр кола збігається з початком координат. Тоді можна помітити, що в силу симетрії кола відносно прямих, що розділяють октанти, досить побудувати растрове подання в одному октанті, а потім за допомогою симетрій одержати зображення в інших октантах (див. рис. 6.7). Будемо користуватися завданням кола у вигляді неявної функції: $x^2 + y^2 - R^2 = 0$.

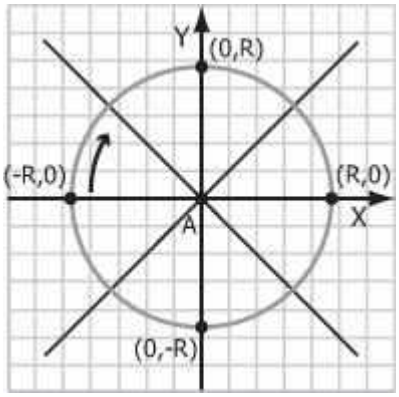


Рис. 6.7. Симетрії при зображенні кола

Нехай $f(x, y) = x^2 + y^2 - R^2$. Будемо малювати частину кола в 4-м октанті, починаючи із крапки (-R, 0) (див. рис. 6.7, показано стрілкою).

Нехай $R \in N$, тоді $f(x,y) \in Z$ для $x \in Z, y \in Z$. Нехай функція plot8(x, y) відображає на растрі всі 8 крапок, отриманих з (x, y) за допомогою симетрій.

6.4.1. Алгоритм Брезенхема

Будемо міркувати подібно алгоритму Брезенхема для відрізків (з відповідними виправленнями на 4-й октант) [17]. Із двох можливих пікселів в 4-му октанті (відповідних вертикальному й діагональному зсувам, які позначаються аналогічно колишнім s і d, див. рис. 6.10) будемо вибирати той, відстань від кола до якого менше.

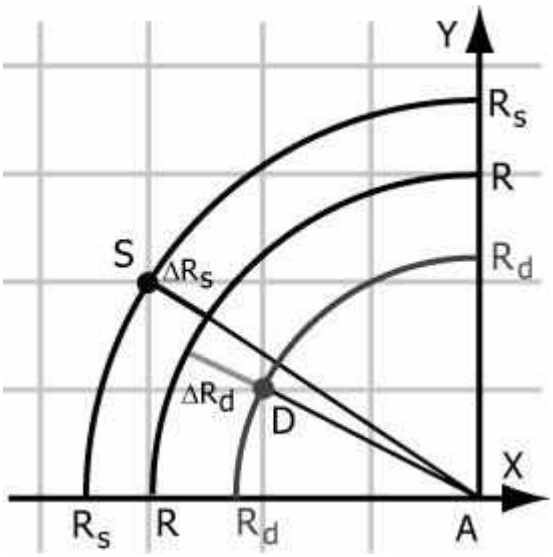


Рис. 6.8. Відстані до кола

Для того щоб вибрати один із двох можливих пікселів, будемо порівнювати відстані від них до кола: де відстань менше – той піксель і буде знайденим. У прикладі на рис. 6.8 рівняються відстані від крапок $S(x_s, y_s)$ і $D(x_d, y_d)$ до кола з радіусом R. З евклідової метрики одержуємо:

$$\Delta R_s = \sqrt{x_s^2 + y_s^2} - R;$$
$$\Delta R_d = R - \sqrt{x_d^2 + y_d^2}.$$

Але обчислення квадратного кореня - трудомістка операція, тому при досить більших R ми будемо заміняти порівняння відстаней порівнянням наближених значень їхніх квадратів (див. рис. 6.9):

$$(\Delta R_s)^2 - (\Delta R_d)^2 = x_s^2 + y_s^2 - 2R\sqrt{x_s^2 + y_s^2} + R^2 - x_d^2 - y_d^2 + 2R\sqrt{x_d^2 + y_d^2} - R^2.$$

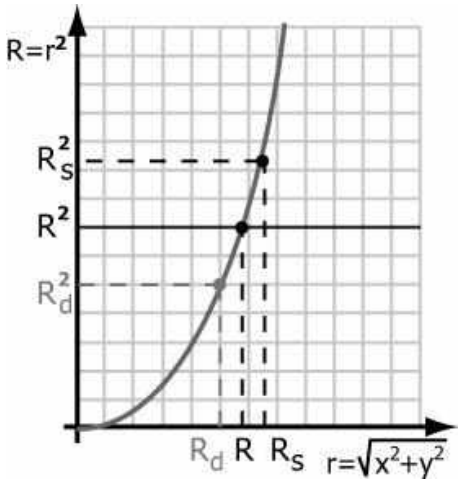


Рис. 6.9. Наближене порівняння відстаней

Зменшимо два доданки на приблизно однакові величини:

$-2R\sqrt{x_s^2 + y_s^2}$ замінимо на $-2R \cdot R$,

$+2R\sqrt{x_d^2 + y_d^2}$ замінимо на $+2\sqrt{x_d^2 + y_d^2} \cdot \sqrt{x_d^2 + y_d^2}$

одержимо

$$(\Delta R_s)^2 - (\Delta R_d)^2 \approx x_s^2 + y_s^2 - 2R \cdot R + R^2 -$$
$$- x_d^2 - y_d^2 + 2\sqrt{x_d^2 + y_d^2} \cdot \sqrt{x_d^2 + y_d^2} - R^2 =$$
$$= x_s^2 + y_s^2 + x_d^2 + y_d^2 - 2R^2.$$

Таким чином, приблизно

1. D ближче до кола, чим S $\Leftrightarrow x_s^2 + y_s^2 + x_d^2 + y_d^2 - 2R^2 > 0;$
2. S ближче до кола, чим D $\Leftrightarrow x_s^2 + y_s^2 + x_d^2 + y_d^2 - 2R^2 < 0.$

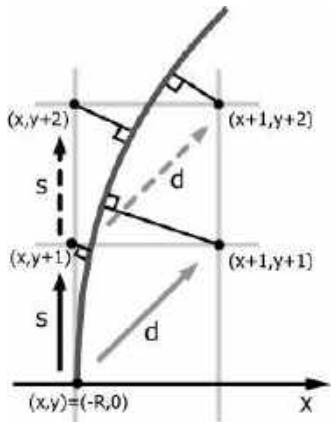


Рис. 6.10. Алгоритм Брезенхема для кола

Нехай (x, y) – поточний піксель. Позначимо

$$F_s = f(x, y+1) = x^2 + (y+1)^2 - R^2 > 0,$$
$$F_d = f(x+1, y+1) = (x+1)^2 + (y+1)^2 - R^2 < 0,$$
$$F = F_s + F_d;$$
$$\Delta F(s) = (\Delta F \text{ при переходе } s\colon (x,y) \to (x, y+1)) =$$
$$= f(x, y+2) + f(x+1, y+2) - f(x, y+1) -$$
$$- f(x+1, y+1)$$
$$= 4y + 6,$$
$$\Delta F(d) = (\Delta F \text{ при переходе } d\colon (x,y) \to (x+1, y+1)) =$$
$$= f(x+1, y+2) + f(x+2, y+2) - f(x, y+1) -$$
$$- f(x+1, y+1)$$
$$= 4x + 4y + 10.$$

Тоді із двох можливих зсувів d і s виберемо.

1. $F = F_s + F_d < 0 \Leftrightarrow F_s < -F_d$, тобто (x + 1, y + 1) ближче до кола, чим (x, y + 1):

d: Переходимо в (x + 1, y + 1) і надаємо відповідні збільшення F, ΔF(s), ?F(d):

F = F +?F(d); ?F(s) = ?F(s)+4; ?F(d) = ?F(d)+8.

2. $F = F_s + F_d < 0 \Leftrightarrow F_s < -F_d$, тобто (x, y+1) ближче до кола, чим (x + 1, y + 1):

s: Переходимо в (x, y + 1) і надаємо відповідні збільшення F, ΔF(s), ΔF(d):

F = F +?F(s); ?F(s) = ?F(s)+4; ?F(d) = ?F(d)+4.

Якщо ми починаємо з (-R, 0), то початкові значення будуть наступними:

$$F = F_s + F_d = ((-R)^2 + 1^2 - R^2) + ((-R+1)^2 + 1^2 - R^2)$$
$$= 1 + (-2R + 1 + 1) = 3 - 2R,$$
$$\Delta F(s) = 4 \cdot 0 + 6 = 6,$$
$$\Delta F(d) = 4 \cdot (-R) + 4 \cdot 0 + 10 = 10 - 4R.$$

Легко бачити, що в алгоритмі всі величини, пов'язані з F, крім F початкового, будуть кратні 2. Але, якщо ми поділимо всі ці величини на 2 (надалі значення всіх величин уже розуміються в цьому змісті), те $F_{\text{нач}} = \frac{1}{2} + 1 - R$. Тому що збільшення F можуть бути тільки цілочисловими, те $F = \frac{1}{2} + T$, де $T \in Z$; тобто якщо відняти $\frac{1}{2}$ від всіх значень, то знак F не зміниться для всіх T, крім T = 0. Для того щоб результат порівняння залишився колишнім, будемо вважати, що F = 0 тепер відповідає зсуву s.

```
x = -r; y = 0;

F = 1-r;
?Fs = 3;
?Fd = 5-2*r;

while( x + y < 0)
{
    plot8( x, y );

    if( F > 0)
```

```
    {
        // d: Діагональний зсув
        F += ?Fd;
        x++; y++;
        ?Fs += 2;
        ?Fd += 4;
    }
    else
    {
        // s: Вертикальний зсув
        F += ?Fs;
        y++;
        ?Fs += 2;
        ?Fd += 2;
    }
}
```

Лістинг 6.5. Алгоритм Брезенхема для кола

Розмірність обчислень цього алгоритму (тобто відношення максимальних модулів значень величин, з якими він оперує до модулів вихідних даних (у цьому випадку R)) дорівнює 2.

6.5. Зображення еліпсів

Насамперед відзначимо, що в еліпса, на відміну від кола, усього 2 осі симетрії, тому по крапках прийде будувати вже 2 октанти (див. рис. 6.11).

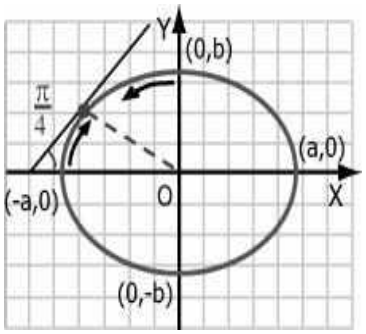


Рис. 6.11. Зображення еліпса

6.5.1. Побудова по неявній функції

Будемо міркувати подібно алгоритму Брезенхема для окружностей.

Неявна функція, що задає еліпс, має вигляд

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 - 1 = 0, a > b$$

$$b^2x^2 + a^2y^2 - a^2b^2 = 0$$

Уведемо $f(x, y) = b^2x^2 + a^2y^2 - a^2b^2$.

Аналогічно алгоритму для кола можна порівнювати f для двох можливих варіантів. Докладний вивід залишаємо читачеві як вправа.

6.5.2. Побудова шляхом стиску кіл

Скористаємося тим, що еліпс із параметрами a, b (нехай a < b) виходить із кола радіуса a стиском по осі y в a/b раз. Побудуємо алгоритм, що є якоюсь комбінацією алгоритмів Брезенхема для кола й для відрізка (див. рис. 6.12).

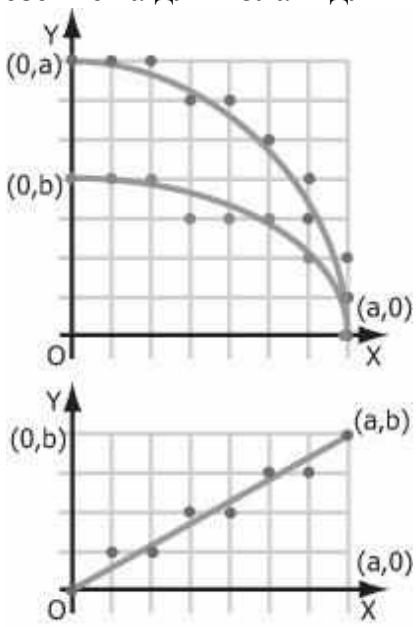


Рис. 6.12. Побудова еліпса шляхом стиску кола

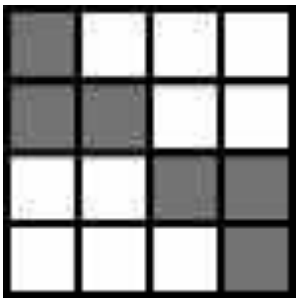


Рис. 6.13. Змішана зв'язність

Почнемо із крапки (a, 0) на колі й із крапки (0, 0) на відрізку. Будемо будувати еліпс точно так само, як окружність, але зміщати поточну крапку по y тільки в тому випадку, коли такий зсув відбувається в поточному кроці вже для відрізка, тобто побудова відрізка саме і є реалізацією стиску в a/b раз (точніше, його дискретною апроксимацією). Цей алгоритм теж має недолік: можлива змішана зв'язність отриманої лінії (див. рис. 6.13).

1) dist(P1,P2) = ((x1 - x2) + (y1 - y2)2)1/2

Тема 7. ВІДСІКАННЯ (КЛіпування) ГЕОМЕТРИЧНИХ ПРИМІТИВІВ

Алгоритм розподілу відрізка навпіл. Коди Сазерланда-Коена. Кліпування багатокутників. Штрихування багатокутної області. Перехід до тривимірного кліпування пірамідою видимості.

У комп'ютерній графіці часто доводиться вирішувати завдання виділення деякої області зображуваної сцени, причому завдання ця може вирішуватися як у застосуванні до плоскої області (якщо сцена вже спроектована на картинну площину), так і до тривимірного. Алгоритми відсікання застосовуються для видалення невидимих поверхонь і ліній, для побудови тіней, при формуванні текстур. Область, що відтинається, може бути як правильної форми (прямокутник або паралелепіпед зі сторонами, паралельними осям координат або координатних площин), так і неправильної (довільний багатокутник або багатогранник). Для того щоб ці алгоритми можна було використовувати в завданнях зображення динамічних сцен, вони повинні бути ефективними відносно часу обчислень. Ми розглянемо трохи найбільше часто застосовуваних алгоритмів.

7.1. Алгоритм Сазерленда-Коена відсікання прямокутною областю

Розглянемо плоску сцену, що складається з відрізків різної довжини й напрямків, у якій треба виділити частину, що перебуває усередині прямокутника. Прямокутник заданий списком ребер: **t<op>, b<ottom>, l<eft>, r<ight>**, відрізки також задаються координатами кінцевих крапок. Область, що відтинається вікном (з урахуванням його границі), складається із крапок (x, y), що задовольняють співвідношенням

$$x_l \le x \le x_r, \quad y_b \le y \le y_t. \tag{7.1}$$

Нехай кінці відрізка задані крапками (x_1, y_1) й (x_2, y_2). Перший крок алгоритму націлений на те, щоб виявити повністю видимі й повністю невидимі відрізки. Відрізок цілком належить виділюваної (кліпуємої) області, якщо обоє його кінця задовольняють умовам (7.1).

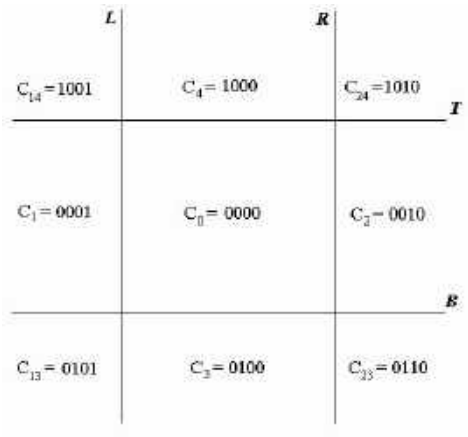


Рис. 7.1. Коди Сазерленда-Коена для областей

Відрізок повністю не бачимо, якщо обоє його кінця лежать

1. праворуч від ребра $r(x_1 > x_r, \quad x_2 > x_r)$;
2. ліворуч від ребра $l(x_1 < x_l, \quad x_2 < x_l)$;
3. знизу від ребра $b(y_1 < y_b, \quad y_2 < y_b)$;
4. зверху від ребра $t(y_1 > y_t, \quad y_2 > y_t)$.

У всіх інших випадках відрізок може (але не зобов'язаний) перетинати прямокутне вікно.

Для виконання аналізу повної видимості або невидимості відрізка А.Сазерленд і Д.Коен запропонували наступний алгоритм. Прямі, яким належать ребра прямокутника, розбивають площина на дев'ять областей, кожної з яких привласнюється чотирьохрозрядний код. Кожний біт цього коду "відповідає" за одну із прямих: 1-й (старший) біт – за пряму t, 2-й – за пряму b, 3-й - за l, 4-й - за r. Якщо в коді області який-небудь біт установлений в 1, то це означає, що вона відділена від вікна відповідної прямої. Схема ідентифікації областей наведена на рис. 7.1.

Кінцевим крапкам відрізків сцени тепер можна привласнити коди залежно від розташування крапок. Ясно, що якщо коди обох кінців відрізка дорівнюють нулю, то відрізок повністю лежить усередині вікна. Для подальшого аналізу скористаємося операцією логічного множення кодів (порозрядне логічне "І"). Тоді таблиця істинності для кодів, відповідно до схеми на рис. 7.1, буде виглядати в такий спосіб:

Таблиця 7.1.

Значення істинності для логічного множення кодів областей

	C_1	C_2	C_3	C_4	C_{13}	C_{14}	C_{23}	C_{24}
C_1	T	F	F	F	T	T	F	F
C_2	F	T	F	F	F	F	T	T
C_3	F	F	T	F	T	F	T	F
C_4	F	F	F	T	F	T	F	T
C_{13}	T	F	T	F	T	T	T	F
C_{14}	T	F	F	T	T	T	F	T
C_{23}	F	T	T	F	T	F	T	T
C_{24}	F	T	F	T	F	T	T	T

Із зіставлення таблиці з малюнком видно, що якщо добуток кодів кінців відрізка приймає значення **T<rue>**, те відрізок цілком лежить по одну сторону якоїсь із прямих, причому зовнішню сторону стосовно вікна, отже, він повністю невидимий. У всіх інших випадках відрізок може частково лежати усередині вікна, тому для визначення їхньої видимої частини треба вирішувати завдання про перетинання відрізків з ребрами вікна. При цьому бажано по можливості скоротити число пар, що перебираються, "відрізок-ребро".

У самому загальному випадку існує дві крапки перетинання відрізка з ребрами, і ці дві крапки приймаються за нові кінцеві крапки зображуваного відрізка. Але спочатку можна виділити деякі більше прості окремі випадки,

пошук перетинань для яких є більше ефективним. Насамперед це горизонтальні й вертикальні відрізки, для яких пошук крапки перетинання тривіальний. Далі, якщо код одного з кінців відрізка дорівнює нулю, то існує тільки одне перетинання цього відрізка з ребром (або із двома ребрами, якщо відрізок проходить через кутову крапку вікна). На рис. 7.2 наведена загальна блок-схема алгоритму відсікання для одного довільно спрямованого відрізка.

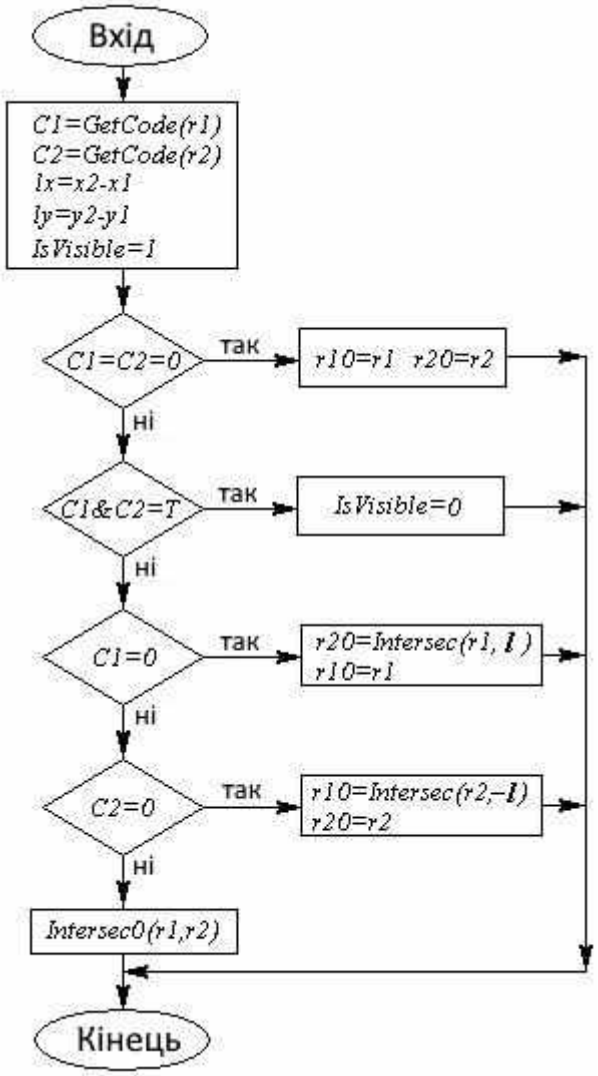


Рис. 7.2. Блок-схема алгоритму Сазерленда-Коена

У блок-схемі використовуються наступні угоди й позначення:

– вхідними даними є крапки $\vec{r}_1 = (x_1, y_1)$, $\vec{r}_2 = (x_2, y_2)$, масив координат вікна $w = \{L, R, B, T\}$; $\vec{l} = (x_2 - x_1, y_2 - y_1) \equiv (l_x, l_y)$;

– на виході одержуємо нові кінцеві крапки $\vec{r}_{10} = (x_{10}, y_{10})$, $\vec{r}_{20} = (x_{20}, y_{20})$, а також значення змінної *IsVisible* (0 – відрізок видимий);

– використовуються наступні допоміжні функції:

GetCode(r) – визначення коду крапки;

Intersec0(r1,l) – пошук перетинання відрізка зі сторонами вікна за умови, що обидві крапки лежать поза вікном; якщо перетинання ні, установлює змінну *IsVisible* в 0;

Intersec(r1,l) – пошук перетинання відрізка зі сторонами вікна за умови, що крапка *r1* лежить у вікні;

C1, C2 – коди крапок *r1, r2*.

У наведеному алгоритмі тепер залишається тільки деталізувати функції *Intersec0* і *Intersec*, ефективність роботи яких є ключовим моментом. Розглянемо один з методів пошуку перетинань, що використовує параметричне рівняння прямої, що проходить через крапки $\vec{r}_1 = (x_1, y_1)$, $\vec{r}_2 = (x_2, y_2)$:

$$\vec{r} = \vec{r}_1 + s(\vec{r}_2 - \vec{r}_1), \quad s \in [-\infty, +\infty],$$

або в координатному виді

$$x = x_1 + sl_x, \quad y = y_1 + sl_y, \quad l_x = x_2 - x_1, \quad l_y = y_2 - y_1.$$

Спробуємо визначити крапку перетинання відрізка з верхньою границею вікна. Оскільки ця границя описується співвідношеннями $L \le x \le R, \quad y = T,$ та умова перетинання з нею кліпованого відрізка виглядає в такий спосіб:

$$s_T = \frac{T - y_1}{l_y}, \quad L \le x_1 + s_T l_x \le R.$$

Аналогічно виглядають формули й для інших границь вікна. Якщо крапка $\vec{r}_1$ розташована усередині вікна, те, залежно від знака l_y, варто шукати перетинання або з верхньої, або з нижньою границею вікна. При відсутності таких відшукуються перетинання з лівою або правою стороною вікна. Але перш ніж перебирати ці варіанти, необхідно виключити випадки горизонтальних $(l_y = 0)$ і вертикальних $(l_x = 0)$ напрямків відрізка. У першому випадку крапками перетинання із правою або лівою границею (залежно від знака l_x) можуть бути (L, y_1) або (R, y_1), а в другому – (x_1, B) або (x_1, T).

Цей алгоритм реалізований у вигляді функції *Intersec*, блок-схема якої наведена на рис. 7.3.

Тепер розглянемо випадок, коли жоден з кінців відрізка не лежить усередині вікна. Тут ми можемо знайти першу крапку перетинання, замінити нею один з кінців відрізка й використовувати попередній алгоритм знаходження другої крапки. Помітимо, що в цьому випадку перший крок не завжди кінчається успішно, оскільки попередній аналіз не дозволяє повністю виключити всі невидимі відрізки. Як і в попередньому алгоритмі, спочатку виконується перевірка на горизонтальне й вертикальне напрямки відрізка. Оскільки частина відрізків ми вже виключили з розгляду завдяки попередньому аналізу, то для цих простих ситуацій залишаються тільки дві можливості, наведені на рис. 7.4. Тут крапки перетинання визначаються зовсім очевидним образом, причому відразу обидві.

Потім послідовно розглядаються чотири випадки розташування крапки $\overrightarrow{r}_1$ відносно прямих, що обмежують вікно. Якщо в якімсь із варіантів буде знайдена крапка перетинання, то аналіз припиняється, крапка $\overrightarrow{r}_1$ заміняється новою крапкою й викликається функція *Intersec*. У випадку ж, коли всі чотири варіанти не дали позитивного результату, змінної *IsVisible* привласнюється значення 0. Алгоритм реалізується функцією *Intersec0* (блок-схема на рис. 7.5).

Запропонована реалізація алгоритму відсікання не є єдиною у своєму роді. Існують і інші підходи, зокрема використання методу розподілу відрізка навпіл для пошуку крапок перетинання й виявлення видимої частини відрізка. Такий варіант алгоритму був запропонований Сазерлендом і Спрулом для апаратної реалізації, але він може бути реалізований і на програмному рівні, хоча при цьому ефективність його буде нижче, ніж у попередні. У ньому немає прямого обчислення координат нової крапки по явних алгебраїчних співвідношеннях. Пошук здійснюється ітераційним методом, у якому на кожному кроці для відрізка, "підозрюваного" у частковій видимості, перебуває його середня крапка, визначається її код, потім із двох відрізків залишаються або обоє (якщо вони обоє не виявляться повністю невидимими), або тільки один, після чого операції тривають із новими відрізками. Ситуація, коли подальшому дробленню піддаються відразу два знову отриманих відрізки, може виникнути тільки на першому ітераційному кроці в тих випадках, коли вихідний відрізок має дві крапки перетинання із границями вікна. На наступних ітераційних кроках кількість аналізованих відрізків уже не буде збільшуватися. Процес триває доти, поки довжина чергового відрізка не стане менше наперед заданій точності. Після цього знайдена крапка перевіряється на предмет перетинання зі стороною вікна. На рис. 7.6 наведені три варіанти відрізків, попередній аналіз яких не класифікує їх як повністю невидимі, і показаний перший ітераційний крок. Відрізок a після першого розподілу дає два частково видимих відрізки, після чого шукаються дві крапки перетинання. В інших випадках залишається лише один із двох відрізків, причому у випадку відрізка c крапка перетинання зі стороною вікна відсутній, тобто виявляється повна невидимість відрізка. По суті справи загальний алгоритм, показана на рис. 7.3, зберігається, змінюється лише метод пошуку крапки перетинання.

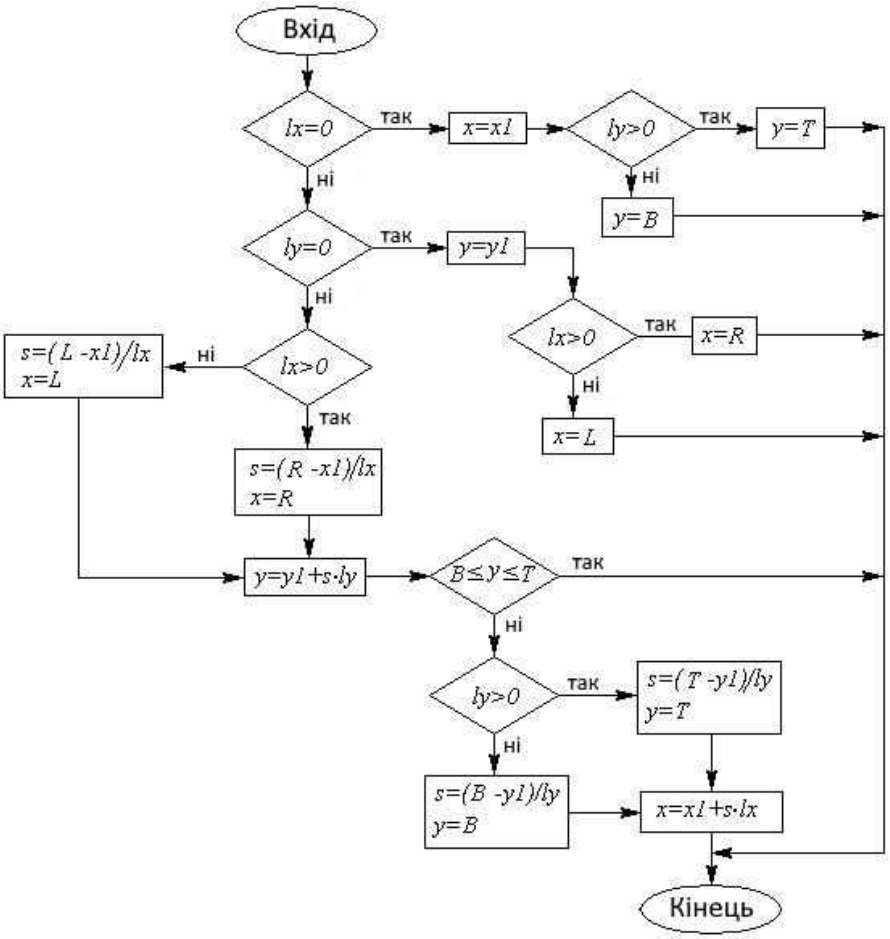


Рис. 7.3. Блок-схема функції Intersec

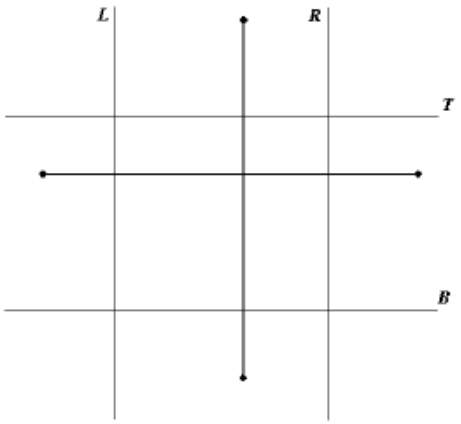


Рис. 7.4. Відрізки паралельні сторонам вікна

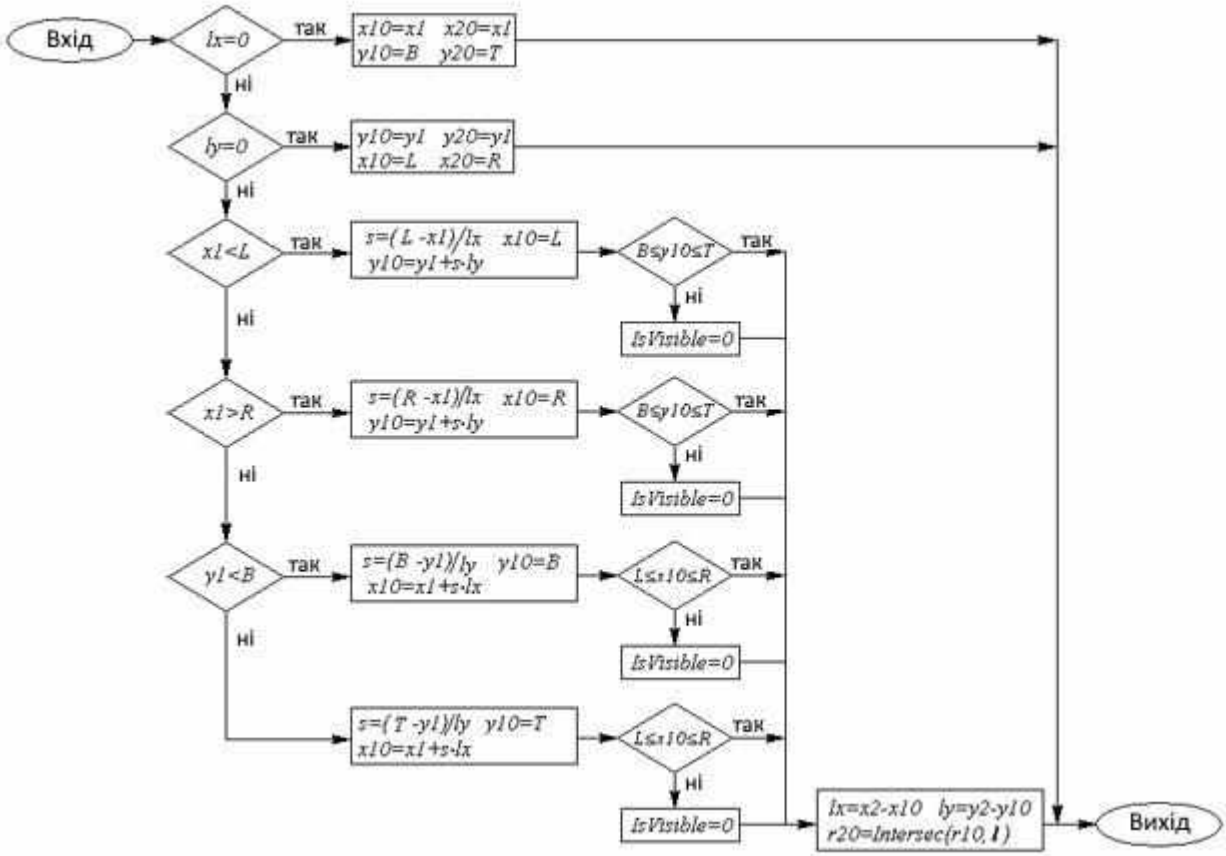


Рис. 7.5. Блок-схема функції Intrsec0

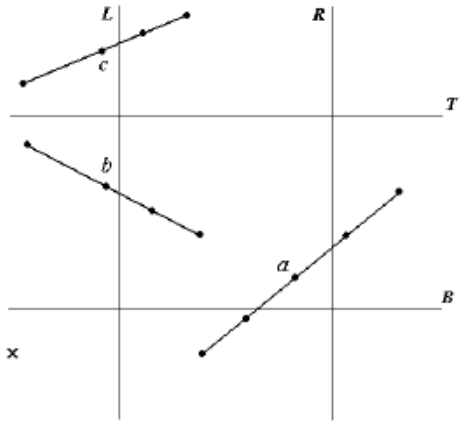


Рис. 7.6. Довільне розташування відрізків

7.2. Відсікання опуклим багатокутником

У багатьох завданнях комп'ютерної графіки часто доводиться мати справа з відсіканням не тільки простим прямокутним вікном, але й вікном досить довільної геометрії. Зокрема, такі завдання можуть виникнути при використанні перспективних проекцій тривимірних сцен, але не тільки в цих випадках. Тому актуальної є завдання відсікання опуклим багатокутником. Ясно, що простий аналіз за допомогою кодів Сазерленда-Коена в такій ситуації не застосуємо. Тут потрібний надійний і досить ефективний алгоритм знаходження крапки перетинання двох довільно орієнтованих відрізків, а також

алгоритм визначення місця розташування крапки щодо багатокутника (усередині, зовні або на границі).

Розглянемо завдання про перетинання відрізка з кінцями $\vec{r}_1 = (x_1, y_1),\ \vec{r}_2 = (x_2, y_2)$ з опуклим багатокутником, заданим списком ребер. Ребро може бути задане у вигляді пари крапок з безлічі вершин багатокутника $R = (\vec{p}, \vec{q}),\ \vec{p} = (p_x, p_y),\ \vec{q} = (q_x, q_y)$ (рис. 7.7). Та обставина, що багатокутник опуклий, є дуже істотним: це дозволяє використовувати досить простий алгоритм, що використовує внутрішні нормалі до його сторін. Під внутрішньою нормаллю розуміється вектор, перпендикулярний стороні й спрямований усередину багатокутника. Як і в попередньому алгоритмі, скористаємося параметричним рівнянням прямої, що проходить через кінці відрізка: $\vec{r} = \vec{r}_1 + s(\vec{r}_2 - \vec{r}_1)$. Якщо при деякому значенні параметра s_0 ця пряма перетинається із прямої, що проходить через крапки $\vec{p}, \vec{q}$, то вектор, що з'єднує довільну крапку ребра із крапкою $\vec{r}_0 = \vec{r}_1 + s_0(\vec{r}_2 - \vec{r}_1)$, буде перпендикулярний вектору нормалі. Отже, скалярний добуток векторів $\vec{n}$ і $\vec{p} - \vec{r}_0$ буде дорівнює нулю. Звідси шляхом нескладних викладень одержуємо $s_0 = \frac{((\vec{p} - \vec{r}_1) \cdot \vec{n})}{((\vec{r}_2 - \vec{r}_1) \cdot \vec{n})}$.

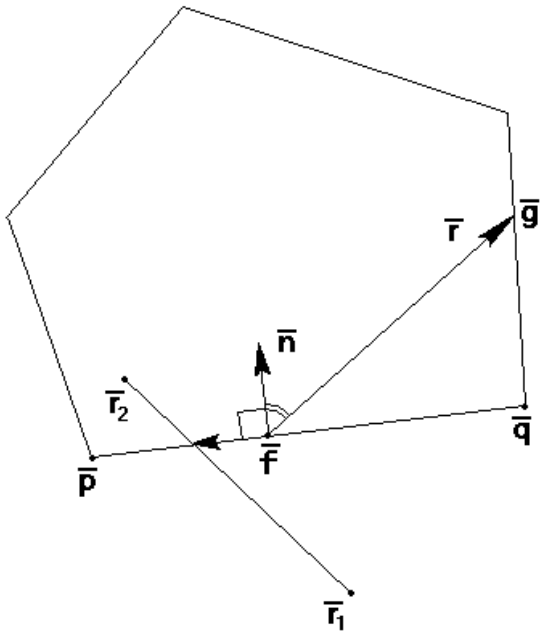


Рис. 7.7. Перетинання відрізка багатокутником

Звичайно, використання цієї формули припускає, що $d \equiv ((\vec{r}_2 - \vec{r}_1) \cdot \vec{n}) \cdot \neq 0$, тобто що відрізок не паралельний стороні багатокутника, але цей випадок розглядається окремо. Знайдена крапка належить відрізку за умови $0 \le s_0 \le 1$. Умова приналежності цієї крапки ребру багатокутника також можна виразити через скалярний добуток, тому що

вектори $\overrightarrow{p} - \overrightarrow{r}_0$ й $\overrightarrow{r}_0 - \overrightarrow{q}$ у цьому випадку повинні бути однаково спрямованими, тобто $((\overrightarrow{p} - \overrightarrow{r}_0) \cdot (\overrightarrow{r}_0 - \overrightarrow{q})) \geq 0$.

Для кожного відрізка можливі три випадки взаємного розташування з багатокутником:

– крапок перетинання немає;
– існує одна крапка перетинання;
– існують дві крапки перетинання.

У кожному із цих варіантів для знаходження перетинання відрізка з вікном необхідно вміти визначати приналежність крапки опуклому багатокутнику. З рис. 7.7 видно, що якщо для будь-якої крапки $\overrightarrow{g}$, що належить багатокутнику (або його границі), і довільної крапки ребра $\overrightarrow{f}$ побудувати вектор $\overrightarrow{m} = \overrightarrow{g} - \overrightarrow{f}$, то виконується умова $(\overrightarrow{m} \cdot \overrightarrow{n} \geq 0)$, оскільки кут між векторами не може перевищувати 90°. Таким чином, якщо дана умова виконується для всіх ребер багатокутника, то крапка є внутрішньою.

Таким чином, алгоритм відсікання відрізка починається з аналізу розташування кінців відрізка стосовно вікна. Якщо обидві крапки лежать усередині вікна, то відрізок повністю видимий, і подальший пошук припиняється. Якщо тільки одна із крапок лежить усередині вікна, то має місце випадок II, і має бути знайти одну крапку перетинання. І, нарешті, якщо обидві крапки лежать поза вікном, то існують або дві крапки перетинання (відрізок перетинає дві границі вікна), або ні однієї (відрізок повністю не бачимо). Втім, дві крапки перетинання можуть збігатися (якщо відрізок проходить через вершину багатокутника), але цей випадок у додатковому аналізі не має потреби.

Далі виконується цикл по всіх ребрах багатокутника з метою знаходження крапок перетинання. Для кожного ребра перед початком пошуку перетинання необхідно перевірити, чи не паралельно воно з відрізком. Якщо це так, то можна обчислити відстань від одного з кінців відрізка до прямої, що проходить через ребро $d = ((\overrightarrow{r}_1 - \overrightarrow{p}) \cdot \overrightarrow{n})$. При $d = 0$ відрізок лежить на прямій, і залишається визначити взаємне розташування кінців відрізка й кінців ребра, що можна зробити простим покоординатним порівнянням. При $d \neq 0$ відрізок не має загальних крапок з даним ребром.

7.3. Кліпування багатокутників

Від завдання відсікання відрізків можна перейти до більше складного: кліпування довільних багатокутників. Якщо багатокутник неопуклий, то в результаті перетинання навіть із прямокутним вікном може вийти трохи не зв'язаних між собою фігур, як це показано на рис. 7.8.

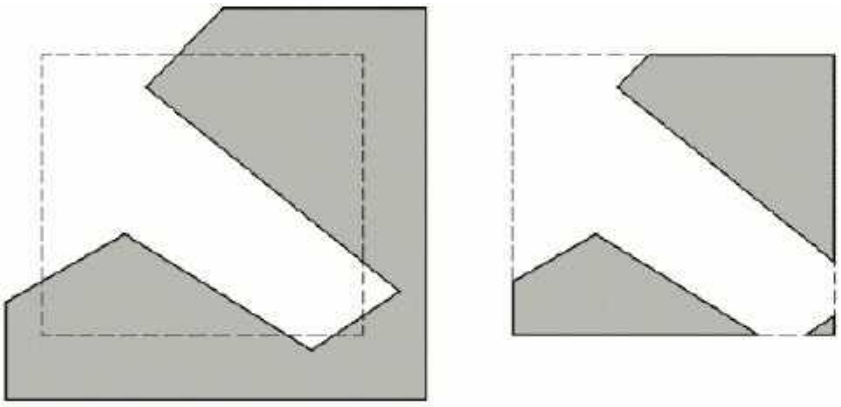


Рис. 7.8. Відсікання неопуклого багатокутника

Коли коштує завдання штрихування замкнутої області одночасно із завданням відсікання, те важливо правильно визначити приналежність знову отриманих фігур внутрішньої або зовнішньої частини вихідного багатокутника. Нехай вихідний багатокутник заданий упорядкованим списком вершин $\{\overrightarrow{p}_1, \overrightarrow{p}_2, \ldots, \overrightarrow{p}_n$, що відповідають ребрам $(\overrightarrow{p}_1, \overrightarrow{p}_2), (\overrightarrow{p}_2, \overrightarrow{p}_3), \ldots, (\overrightarrow{p}_{n-1}, \overrightarrow{p}_n), (\overrightarrow{p}_n, \overrightarrow{p}_1)$. Для відсікання такого багатокутника прямокутним вікном можна застосувати алгоритм, запропонований Сазерлендом і Ходжменом. Ідея його полягає в послідовному відсіканні частини багатокутника прямими, що відповідають сторонам вікна. Результатом його роботи є впорядкований список вершин, що лежать у видимій частині вікна. На кожному кроці алгоритму утвориться деяка проміжна фігура, також представлена впорядкованим списком вершин і ребер. Приклад таких послідовних відсікань показаний на рис. 7.9. У процесі відсікання послідовно обходиться список вершин, причому кожна чергова крапка за винятком першої розглядається як кінцева крапка ребра, початковою крапкою якого є попередня крапка зі списку. Порядок, у якому розглядаються сторони вікна, не має значення. У процесі обходу формується список нових вершин багатокутника.

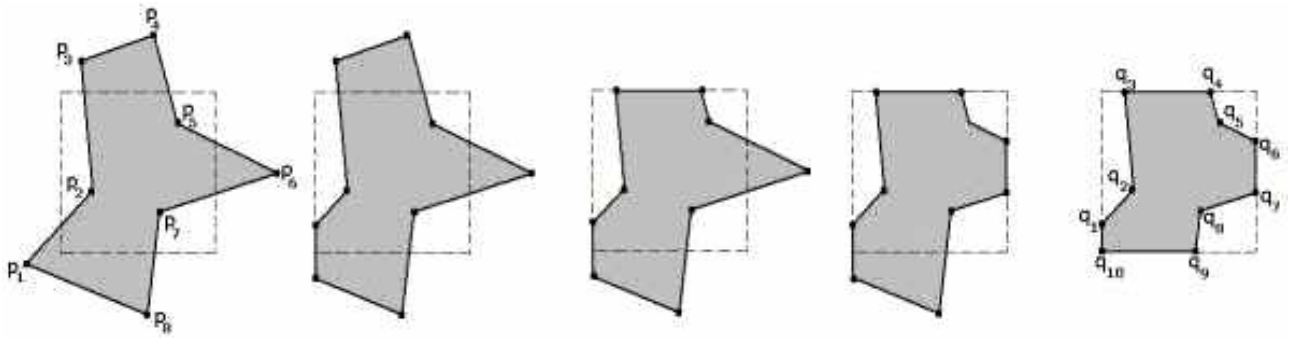


Рис. 7.9. Послідовні кроки клішування довільного багатокутника

На першому кроці для першої вершини в списку визначається її приналежність видимої області. Якщо вона видима, то вона стає першою крапкою першого оброблюваного ребра й заноситься в список нових вершин.

Якщо ж вона невидима, то в список нових вершин не заноситься, але однаково стає першою крапкою ребра.

Для аналізованого ребра можливі чотири випадки розташування щодо вікна.

1. Ребро повністю видиме. Чергова крапка заноситься в список нових вершин (що чекає уже повинна перебувати в цьому списку, оскільки ребро повністю видиме).
2. Ребро повністю невидимо. Ніяких дій не виробляється.
3. Ребро виходить із області. Перебуває крапка перетинання ребра зі стороною вікна й заноситься в список нових ребер.
4. Ребро входить в область. Також відшукується крапка перетинання зі стороною вікна й заноситься в список нових вершин. Кінцева крапка теж заноситься в список нових вершин.

У цьому алгоритмі постійно доводиться визначати видимість крапки стосовно конкретного ребра вікна, що відтинає. Вікно також можна задати у вигляді впорядкованого списку вершин. Якщо обхід вершин вікна здійснюється за годинниковою стрілкою, то його внутрішня область буде розташована праворуч від границі. При цьому розташування крапки $\vec{p}$ відносно прямій, який належить ребро, можна встановлювати різними способами:

1. Вибирається початкова крапка $\vec{s}$ даного ребра й будується вектор $\vec{r} = \vec{p} - \vec{s}$ і вектор внутрішньої нормалі n до границі (ребру) вікна. Обчислюється скалярний добуток $d = (\vec{r} \cdot \vec{n})$. Якщо $d \geq 0$, то крапка $\vec{p}$ є видимою.
2. Будується просторовий вектор $\vec{r}$ (третю координату можна покласти рівної нулю). Вектор $\vec{l}$ (також просторовий, лежачий у тій же площині, що й $\rightarrow$) спрямований уздовж ребра (з урахуванням напрямку обходу). Обчислюється векторний добуток $\vec{v} = [\vec{r} \times \vec{l}]$. Якщо координата z у вектора $\vec{v}$ позитивна, то крапка $\vec{p}$ лежить праворуч від ребра (є видимою).
3. Виписується канонічне рівняння прямої, що проходить через ребро: $f(x, y) + by + c = 0$
4. Для довільної внутрішньої крапки вікна (x_0, y_0) обчислюється значення $d = f(x_0, y_0)$, а також для крапки $\vec{p} = (x_1, y_1)$ обчислюється $d_1 = f(x_1, y_1)$. Якщо числа d й d_1 мають однаковий знак, то крапка $\vec{p}$ є видимою.

Найбільше просто ці алгоритми реалізуються у випадку відсікання прямокутним вікном зі сторонами, паралельними осям координат.

7.4. Питання й вправи

1. Що таке кліпування?
2. Якщо кінці відрізків мають коди 1000 і 0100, скільки сторін вікна він може перетинати?

3. При якому значенні коду одного з кінців відрізка він обов'язково буде частково видимим?
4. Якщо обидва кінці відрізка лежать поза вікном, то при яких кодах кінців він може проходити уздовж діагоналі вікна?
5. Який з алгоритмів відсікання відрізків ефективніше: наведений у блок-схемі 7.3 або заснований на розподілі відрізка навпіл?
6. За допомогою якої умови можна визначити приналежність крапки опуклому багатокутнику?
7. Чи буде ця умова застосовна у випадку довільного багатокутника? (підтвердите свою відповідь прикладами).
8. Які випадки розташування ребра щодо вікна розглядаються в алгоритмі клішування довільного багатокутника?

Тема 8. ВИДАЛЕННЯ НЕВИДИМИХ ПОВЕРХОНЬ І ЛІНІЙ

Історичний екскурс. Методи переборного типу. Метод Z-Буфера. Методи видалення нелицьових граней багатогранника. Алгоритми Варнака й Вейлера-Азертона. Методи пріоритетів (художника, що плаває обрію). Метод двійкової розбивки простору. Алгоритми порядкового сканування для криволінійних поверхонь. Алгоритм визначення видимих поверхонь шляхом трасування променів

Завдання видалення невидимих ліній і поверхонь є однієї з найцікавіших і складних у комп'ютерній графіці. Алгоритми видалення полягають у визначенні ліній ребер, поверхонь або обсягів, які видимі або невидимі для спостерігача, що перебуває в заданій крапці простору.

Необхідність видалення невидимих ліній, ребер, поверхонь або обсягів проілюстрована на рис. 8.1. Малюнок наочно демонструє, що зображення без видалення невидимих ліній сприймається неоднозначно.

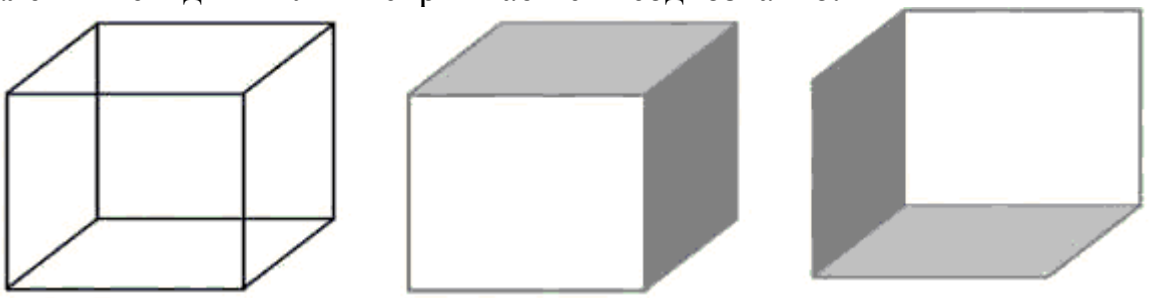


Рис. 8.1. Неоднозначність сприйняття зображення куба

Складність завдання видалення невидимих ліній і поверхонь привела до появи великої кількості різних способів її рішення. Багато хто з них орієнтовані на спеціалізовані додатки. Єдиного (загального) рішення цього завдання, придатного для різних випадків, природно, не існує: для кожного випадку вибирається найбільш підходящий метод. Наприклад, для моделювання процесів у реальному часі потрібні швидкі алгоритми, у той час як для формування складного реалістичного зображення, у якому представлені тіні, прозорість і фактура, що враховують ефекти відбиття й переломлення кольору в дрібних відтінках, фактор часу виконання вже не так явний. Подібні алгоритми працюють повільно, і найчастіше на обчислення потрібно кілька мінут або навіть годин. Існує тісний взаємозв'язок між швидкістю роботи алгоритму й детальністю його результату. Жоден з алгоритмів не може досягти гарних оцінок для цих двох показників одночасно. У міру створення усе більше швидких алгоритмів можна будувати усе більше детальні зображення. Реальні завдання, однак, завжди будуть вимагати обліку ще більшої кількості деталей.

Всі алгоритми такого роду так чи інакше містять у собі сортування, причому головне сортування ведеться по геометричній відстані від тіла, поверхні, ребра або крапки до крапки спостереження або картинної площини. Основна ідея, покладена в основу сортування по відстані, полягає в тім, що чим далі розташовано об'єкт від крапки спостереження, тим більше ймовірність, що

він буде повністю або частково закритий одним з об'єктів, більше близьких до крапки спостереження. Після визначення відстаней або **пріоритетів** по глибині залишається провести сортування по горизонталі й по вертикалі, щоб з'ясувати, чи буде розглянутий об'єкт дійсно закритий об'єктом, розташованим ближче до крапки спостереження. Ефективність будь-якого алгоритму видалення значною мірою залежить від ефективності процесу сортування.

Алгоритми видалення невидимих ліній або поверхонь можна класифікувати по способі вибору системи координат або простору, у якому вони працюють. Алгоритми, що працюють в об'єктному просторі, мають справа зі світовою системою координат, у якій описані ці об'єкти. При цьому виходять досить точні результати, обмежені, загалом кажучи, лише погрішністю обчислень. Отримані зображення можна вільно масштабувати. Алгоритми, що працюють в об'єктному просторі, особливо корисні в тих додатках, де необхідна висока точність. Алгоритми ж, що працюють у просторі зображення, мають справа із системою координат того екрана, на якому об'єкти відображаються. При цьому точність обчислень обмежена розв'язною здатністю екрана.

Ми приведемо деякі з алгоритмів, що працюють як в об'єктному просторі, так і в просторі зображення, кожний з яких ілюструє одну або кілька основних ідей теорії алгоритмів видалення невидимих ліній і поверхонь.

8.1. Алгоритм Робертса

Цей алгоритм, запропонований в 1963 р., є першою розробкою такого роду й призначений для видалення невидимих ліній при штриховому зображенні об'єктів, складених з опуклих багатогранників. Він ставиться до алгоритмів, що працюють в об'єктному просторі, і дуже елегантний з математичної точки зору. У ньому дуже вдало сполучаються геометричні методи й методи лінійного програмування.

Опуклий багатогранник однозначно визначається набором площин, що утворять його грані, тому вихідними даними для алгоритму є багатогранники, задані списком своїх граней. Грані задаються у вигляді площин, заданих у канонічній формі (див. лекцію 3) в об'єктній системі координат: $ax + by + cz + d = 0$.

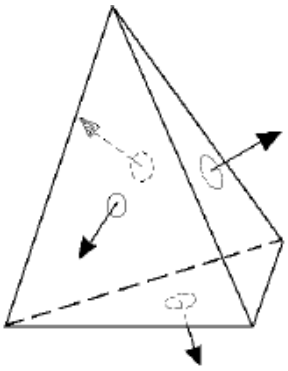


Рис. 8.2. Зовнішні нормалі тетраедра

Таким чином, кожна площина визначається чотирьохмірним вектором $\overrightarrow{P}$, а кожна крапка $\overrightarrow{r}$, задана в однорідних координатах, також являє собою чотирьохмірний вектор:

$$\overrightarrow{P} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}, \overrightarrow{r} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Приналежність крапки площини можна встановити за допомогою скалярного добутку, тобто якщо $(\overrightarrow{P} \cdot \overrightarrow{r}) = 0$, те крапка належить площини, якщо ж ні, те знак добутку показує, по яку сторону від площини ця крапка перебуває. В алгоритмі Робертса площини будуються таким чином, що внутрішні крапки багатогранника лежать у позитивній напівплощині. Це означає, що вектор (A, B, C) є **зовнішньою нормаллю** до багатогранника (рис. 8.2). З векторів площин будується прямокутна матриця порядку $4 \times n$, що називається **узагальненою матрицею опису багатогранника**:

$$M = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ b_1 & b_2 & b_3 & \dots & b_n \\ c_1 & c_2 & c_3 & \dots & c_n \\ d_1 & d_2 & d_3 & \dots & d_n \end{pmatrix}.$$

Множачи стовпці матриці на вектор $\overrightarrow{r}$, одержимо n-*мірний* вектор, і якщо всі його компоненти не негативні, то крапка належить багатограннику. Ця умова будемо записувати у вигляді $(\overrightarrow{r} \cdot M) \geq 0$(мається на увазі множення вектор-рядка на матрицю).

У своєму алгоритмі Робертс розглядає тільки відрізки, що є перетинанням граней багатогранника.

З узагальненої матриці можна одержати інформацію про те, які грані багатогранника перетинаються у вершинах. Дійсно, якщо вершина $\overrightarrow{v} = (x, y, z, 1)$належить граням $\overrightarrow{P}_1$, $\overrightarrow{P}_2$, $\overrightarrow{P}_3$, то вона задовольняє рівнянням

$$\left.\begin{aligned} (\overrightarrow{v} \cdot \overrightarrow{P}_1) &= 0 \\ (\overrightarrow{v} \cdot \overrightarrow{P}_2) &= 0 \\ (\overrightarrow{v} \cdot \overrightarrow{P}_3) &= 0 \\ (\overrightarrow{v} \cdot \overrightarrow{e}_4) &= 0 \end{aligned}\right\}, \quad \text{где} \quad \overrightarrow{e}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Цю систему можна записати в матричному виді:

$$\overrightarrow{v} \cdot Q = \overrightarrow{e}_4,$$

де Q – матриця, складена з вектор-стовпців $\overrightarrow{P}_1, \overrightarrow{P}_2, \overrightarrow{P}_3, \overrightarrow{e}_4$. Виходить, координати вершини визначаються співвідношенням $\overrightarrow{v} = \overrightarrow{e}_4 \cdot Q^{(-1)}$, тобто вони становлять останній рядок зворотної матриці. А це означає, що якщо для яких-небудь трьох площин зворотна матриця існує, то площини мають загальну вершину.

Алгоритм насамперед видаляє з кожного багатогранника ті ребра або грані, які екрануються самим тілом. Робертс використовував для цього простий тест: якщо одна або обидві суміжні грані звернені своєю зовнішньою поверхнею до спостерігача, то ребро є видимим. Тест цей виконується обчисленням скалярного добутку координат спостерігача на вектор зовнішньої нормалі грані: якщо результат негативний, то грань видима.

Потім кожне з видимих ребер кожного багатогранника рівняється з кожним з багатогранників, що залишилися, для визначення того, яка його частина або частини, якщо такі є, екрануються цими тілами. Для цього в кожну крапку ребра проводиться відрізок променю, що виходить із крапки розташування спостерігача. Якщо відрізок не перетинає жодного з багатогранників, то крапка видима. Для рішення цього завдання використовуються параметричні рівняння прямої, що містить ребро, і променю.

Якщо задані кінці відрізка $\vec{r}$ й $\vec{s}$, а спостерігач розташований у крапці $\vec{u}$, то відрізок задається рівнянням

$$\vec{v} = \vec{r} + t \cdot (\vec{s} - \vec{r}) \equiv \vec{r} + t \cdot \vec{d}, \quad 0 \le t \le 1,$$

а пряма, що йде в крапку, що відповідає параметру t, – рівнянням

$$\vec{w} = \vec{v} + \tau \cdot \vec{g} = \vec{r} + t \cdot \vec{d} + \tau \cdot \vec{g}.$$

Для визначення тої частини відрізка, що закривається яким-небудь тілом, досить знайти значення t й τ, при яких добуток вектора $\vec{w}$ на узагальнену матрицю позитивно. Для кожної площини $\vec{P}_i$ записується нерівність

$$q_i = (\vec{v} \cdot \vec{P}_I) + t(\vec{d} \cdot \vec{P}_I) + \tau(\vec{g} \cdot \vec{P}_I) > 0.$$

Ці умови повинні виконуватися для всіх площин.

Думаючи $q_i = 0$, одержуємо систему рівнянь, рішення якої дають нам крапки "зміни видимості" відрізка. Результат можна одержати шляхом спільного рішення всіляких пар рівнянь із цієї системи. Число всіляких рішень при N площинах дорівнює $N \cdot (N-1)/2$.

Тому що обсяг обчислень росте зі збільшенням числа багатокутників, те бажано в міру можливості скорочувати їхнє число, тобто якщо ми апроксимуємо деяку поверхню багатогранником, то як грані можна використовувати не трикутники, а більше складні багатокутники. При цьому, зрозуміло, встає проблема, як побудувати такий багатокутник, щоб він мало відхилявся від плоскої фігури.

8.2. Алгоритм Варнока

На відміну від алгоритму Робертса, Варнок в 1968 р. запропонував алгоритм, що працює не в об'єктному просторі, а в просторі образа. Він також націлений на зображення багатогранників, а головна ідея його заснована на гіпотезі про спосіб обробки інформації, що втримується в сцені, оком і мозком людини. Ця гіпотеза полягає в тім, що витрачається дуже мало часу й зусиль на обробку тих областей, які містять мало інформації. Більша частина часу й праці затрачається на області з високим інформаційним умістом. Так, наприклад,

розглядаючи приміщення, у якому є тільки картина на стіні, ми швидко оглядаємо стіни, підлогу й стеля, а потім вся увага зосереджуємо на картині. У свою чергу, на цій картині, якщо це портрет, ми швидко відзначаємо тло, а потім більш уважно розглядаємо особу зображеного персонажа, особливо ока, губи. Як правило, досить детально розглядаються ще й руки й з ледве меншою увагою – одяг.

В алгоритмі Варнока і його варіантах робиться спроба скористатися тим, що більші області зображення однорідні. Таку властивість називають **когерентністю**, маючи на увазі, що суміжні області (пікселі) уздовж обох осей *x* и *в* мають тенденцію до однорідності.

У просторі зображення розглядається вікно й вирішується питання про те, чи порожньо воно, або його вміст досить просто для візуалізації. Якщо це не так, то вікно розбивається на фрагменти доти, поки вміст фрагмента не стане досить простим для візуалізації або його розмір не досягне необхідної межі розширення. В останньому випадку інформація, що втримується у вікні, осереднюється, і результат зображується з однаковою інтенсивністю або кольором.

Конкретна реалізація алгоритму Варнока залежить від методу розбивки вікна й від деталей критерію, використовуваного для того, щоб вирішити, чи є вміст вікна досить простим. В оригінальній версії алгоритму кожне вікно розбивалося на чотири однакових підвікна. Багатокутник, що входить у зображувану сцену, стосовно вікна будемо називати (рис. 8.3)

- **зовнішнім**, якщо він цілком перебуває поза вікном;
- **внутрішнім**, якщо він цілком розташований усередині вікна;
- **що перетинає**, якщо він перетинає границю вікна;
- **що охоплює**, якщо вікно цілком розташоване всередині нього.

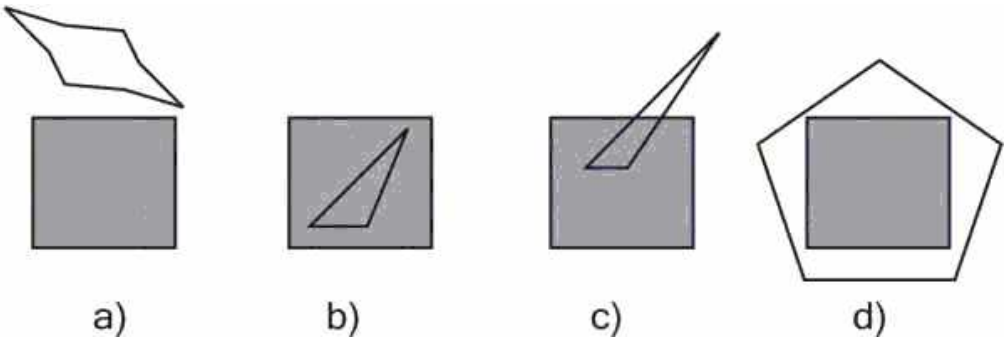


Рис. 8.3. Варіанти розташування багатокутника стосовно вікна

Тепер можна в самому загальному виді описати алгоритм.

Для кожного вікна:

1. Якщо всі багатокутники сцени є зовнішніми стосовно вікна, то воно пусте; зображується фоновим кольором і подальшою розбивкою не підлягає.
2. Якщо тільки один багатокутник сцени має загальні крапки з вікном і є стосовно нього внутрішнім, то вікно заповнюється фоновим кольором, а сам багатокутник заповнюється своїм кольором.

3. Якщо тільки один багатокутник сцени має загальні крапки з вікном і є стосовно нього що перетинає, то вікно заповнюється фоновим кольором, а частина багатокутника, що належить вікну, заповнюється кольором багатокутника.
4. Якщо тільки один багатокутник охоплює вікно й немає інших багатокутників, що мають загальні крапки з вікном, то вікно заповнюється кольором цього багатокутника.
5. Якщо існує хоча б один багатокутник, що охоплює вікно, то серед всіх таких багатокутників вибирається той, котрий розташований ближче всіх багатокутників до крапки спостереження, і вікно заповнюється кольором цього багатокутника.
6. У противному випадку виробляється нова розбивка вікна.

Кроки 1-4 розглядають ситуацію перетинання вікна тільки з одним багатокутником. Вони використовуються для скорочення числа розбивок. Крок 5 вирішує завдання видалення невидимих поверхонь. Багатокутник, що перебуває ближче всіх до крапки спостереження, екранує всі інші.

Для реалізації алгоритму необхідні функції, що визначають взаємне розташування вікна й багатокутника, які досить легко реалізуються у випадку прямокутних вікон і опуклих багатокутників. Для визначення, чи є багатокутник що охоплює, зовнішнім або внутрішнім, можна скористатися, наприклад, зануренням багатокутника в прямокутну оболонку. Для визначення наявності перетинань можна використовувати опорні прямі (так само, як використовувалися площини в алгоритмі Робертса). Якщо ж багатокутник неопуклий, то завдання ускладняється. Методи рішення такого роду завдань будуть розглянуті в главі, що ставиться до геометричного пошуку.

Варто помітити, що існують різні реалізації алгоритму Варнока. Були запропоновані варіанти оптимізації, що використовують попереднє сортування багатокутників по глибині, тобто по відстані від крапки спостереження, і інші.

8.3. Алгоритм Вейлера-Азертона

Вейлер і Азертон спробували оптимізувати алгоритм Варнока відносно числа виконуваних розбивок, перейшовши від прямокутних розбивок до розбивок уздовж границь багатокутників (1977). Для цього вони використовували ними ж розроблений алгоритм відсікання багатокутників. Алгоритм працює в об'єктному просторі, і результатом його роботи є багатокутники. У самому загальному виді він складається із чотирьох кроків.

1. Попереднє сортування по глибині.
2. Відсікання по границі найближчого до крапки спостереження багатокутника, називане сортуванням багатокутників на площині.
3. Видалення багатокутників, екранованих більше близькими до крапки спостереження багатокутниками.
4. Якщо потрібно, то рекурсивна розбивка й нове сортування.

У процесі попереднього сортування створюється список приблизних пріоритетів, причому близькість багатокутника до крапки спостереження визначається відстанню до найближчої до неї вершини. Потім виконується

відсікання по найпершому з багатокутників. Відсіканню піддаються всі багатокутники зі списку, причому ця операція виконується над проекціями багатокутників на картинну площину. При цьому створюються списки зовнішніх і внутрішніх фігур. Всі попавші в список зовнішніх не екрануються багатокутником, що відтинає. Потім розглядається список внутрішніх багатокутників і виконується сортування по відстані до багатокутника, що відтинає. Якщо всі вершини деякого багатокутника виявляються далі від спостерігача, чим сама вилучена з вершин що екранує, то вони невидимі, і тоді вони віддаляються. Після цього робота алгоритму триває із зовнішнім списком.

Якщо якась із вершин внутрішнього багатокутника виявляється ближче до спостерігача, чим найближча з вершин багатокутника, що екранує, то такий багатокутник є частково видимим. У цьому випадку попередній список пріоритетів некоректний, і тоді в якості нового багатокутника, що відтинає, вибирається саме цей "порушник порядку". При цьому використовується саме вихідний багатокутник, а не той, що вийшов у результаті першого відсікання. Такий підхід дозволяє мінімізувати число розбивок.

Цей алгоритм надалі був узагальнений Кетмулом (1974) для зображення гладких бікубічних поверхонь. Його підхід полягав у тім, що розбивці піддавалася поверхня. Коротко цей алгоритм можна описати так:

1. Рекурсивно розбивається поверхня доти, поки проекція елемента на площину зображення не буде покривати не більше одного пікселя.
2. Визначити атрибути поверхні в цьому пікселі й зобразити його.

Ефективність такого методу, як і алгоритм Варнока, залежить від ефективності розбивок. Надалі цей алгоритм був розповсюджений на сплайнові поверхні.

8.4. Метод Z-Буфера

Це один з найпростіших алгоритмів видалення невидимих поверхонь. Уперше він був запропонований Кетмулом в 1975 р. Працює цей алгоритм у просторі зображення. Ідея Z-Буфера є простим узагальненням ідеї про буфер кадру. Буфер кадру використовується для запам'ятовування атрибутів кожного пікселя в просторі зображення, а Z-Буфер призначений для запам'ятовування глибини (відстані від картинної площини) кожного видимого пікселя в просторі зображення. Оскільки досить розповсюдженим є використання координатної площини XOY як картинна площина, то глибина дорівнює координаті z крапки, звідси й назва буфера. У процесі роботи значення глибини кожного нового пікселя, якому потрібно занести в буфер кадру, рівняється із глибиною того пікселя, що вже занесений в Z-Буфер. Якщо це порівняння показує, що новий піксель розташовано спереду пікселя, що перебуває в буфері кадру, те новий піксель заноситься в цей буфер і, крім того, виробляється коректування Z-Буфера новим значенням глибини. Якщо ж порівняння дає протилежний результат, то ніяких дій не виробляється. По суті, алгоритм є пошуком по x й y найбільшого значення функції $z(x,y)$.

Головна перевага алгоритму – його простота. Крім того, цей алгоритм вирішує завдання про видалення невидимих поверхонь і робить тривіальної

візуалізацію перетинань складних поверхонь. Сцени можуть бути будь-якої складності. Оскільки габарити простору зображення фіксовані, оцінка обчислювальної трудомісткості алгоритму не більш ніж лінійна. Оскільки елементи сцени або картинки можна заносити в буфер кадру або в Z-Буфер у довільному порядку, їх не потрібно попередньо сортувати по пріоритеті глибини. Тому заощаджується обчислювальний час, затрачуване на сортування по глибині.

Основний недолік алгоритму – великий обсяг необхідної пам'яті. Останнім часом у зв'язку зі швидким ростом можливостей обчислювальної техніки цей недолік стає менш лімітує. Але в той час, коли алгоритм ще тільки з'явився, доводилося винаходити способи створення буфера як можна більшого обсягу при наявному ресурсі пам'яті.

Наприклад, можна розбивати простір зображення на 4, 16 або більше прямокутників або смуг. У граничному варіанті можна використовувати буфер розміром в один рядок розгорнення. Для останнього випадку був розроблений **алгоритм порядкового сканування**. Оскільки кожний елемент сцени обробляється багато разів, то сегментування Z-Буфера, загалом кажучи, приводить до збільшення часу, необхідного для обробки сцени.

Інший недолік алгоритму складається в трудомісткості реалізації ефектів, пов'язаних з напівпрозорістю, і ряду інших спеціальних завдань, що підвищують реалістичність зображення. Оскільки алгоритм заносить пікселі в буфер кадру в довільному порядку, те досить складно одержати інформацію, що необхідна для методів, що ґрунтуються на попередньому аналізі сцени.

У цілому алгоритм виглядає так:

1. Заповнити буфер кадру фоновим значенням кольору.
2. Заповнити Z -буфер мінімальним значенням z (глибини).
3. Перетворити зображувані об'єкти в растрову форму в довільному порядку.
4. Для кожного об'єкта:

4.1. Для кожного пікселя (x, y) образа обчислити його глибину $z(x, y)$.

4.2. Зрівняти глибину $z(x, y)$ зі значенням глибини, що зберігається в Z-Буфері в цій же позиції.

4.3. Якщо $z(x, y) > Z - \text{буфер}(x, y)$, те занести атрибути пікселя в буфер кадру й замінити $Z - \text{буфер}(x, y) \text{на} z(x, y)$. У противному випадку ніяких дій не робити.

Алгоритм, що використовує Z-Буфер, можна також застосовувати для побудови перетинів поверхонь. Зміниться тільки оператор порівняння:

$$z(x, y) > Z - \text{буфер}(x, y) \text{ и } z(x, y) = z \text{ сечения}$$

де z сечения – глибина шуканого перетину.

8.5. Методи пріоритетів

Тут ми розглянемо групу методів, що враховують специфіку зображуваної сцени для видалення невидимих ліній і поверхонь.

При зображенні сцен із суцільним зафарбовуванням поверхонь можна скористатися **методом художника**: елементи сцени зображуються в

послідовності від найбільш вилучених від спостерігача до більше близького. При екрануванні одних ділянок сцени іншими невидимі ділянки просто зафарбовуються. Якщо обчислювальна трудомісткість одержання зображення для окремих елементів досить висока, то такий алгоритм буде не найкращим по ефективності, але зате ми уникнемо аналізу (і цілком можливо, теж дорогого), що дозволяє встановити, які ж з елементів зображувати не треба в силу їхньої невидимості. Наприклад, при зображенні правильного багатогранника ми досить легко можемо впорядкувати його грані по глибині, але таке сортування для довільного багатогранника можливі далеко не завжди. Ми розглянемо застосування цього методу на прикладі зображення поверхні, заданої у вигляді однозначної функції двох змінних.

Нехай поверхня задана рівнянням $z = f(x,y), \quad a \le x \le b, \quad c \le y \le d.$

Як картинна площина виберемо площину XOY. В області завдання функції на осях координат побудуємо сітку вузлів:

$$a = x_0 < x_1 < \ldots < x_{n-1} = b, \quad c = y_0 < y_1 < \ldots < y_{m-1} < y_m = d.$$

Тоді $z_{ij} = f(x_i, y_i)$ являють собою набір "висот" для даної поверхні стосовно площини XOY. Поверхня будемо апроксимувати трикутниками з вершинами в крапках $\overrightarrow{r}_{ij} = (x_i, y_i, z_{ij})$ так, що кожному прямокутнику сітки вузлів будуть відповідати два трикутники: $tr^1_{i\,j} = \{\overrightarrow{r}_{ij}, \overrightarrow{r}_{i+1\,j}, \overrightarrow{r}_{i+1\,j+1}\}$ і $tr^2_{i\,j} = \{\overrightarrow{r}_{ij}, \overrightarrow{r}_{i\,j+1}, \overrightarrow{r}_{i+1\,j+1}\}$. Для побудови наочного зображення поверхні повернемо її на деякий кут спочатку щодо осі OX, а потім щодо осі OY, причому напрямок обертання виберемо таким чином, що крапки, що відповідають кутам координатної сітки, розташуються в наступному порядку по далекості від картинної площини: $\overrightarrow{r}_{nm}, \overrightarrow{r}_{n0}, \overrightarrow{r}_{0m}, \overrightarrow{r}_{00}$, тобто крапка $\overrightarrow{r}_{nm}$ виявиться найбільш близької до картинної площини (і найбільш вилученої від спостерігача). Передбачається, що спосіб зафарбовування трикутників уже визначений. Тоді процес зображення поверхні можна коротко записати так:

Для $i = n, \ldots, 1$

Для $j = m, \ldots, 1$

Нарисовать $tr^1_{i\,j}$; нарисовать $tr^2_{i\,j}$.

При такій послідовності виводу зображення ми просуваємося від самого вилученого трикутника до усе більше близьких, частково зафарбовуючи вже зображені ділянки поверхні.

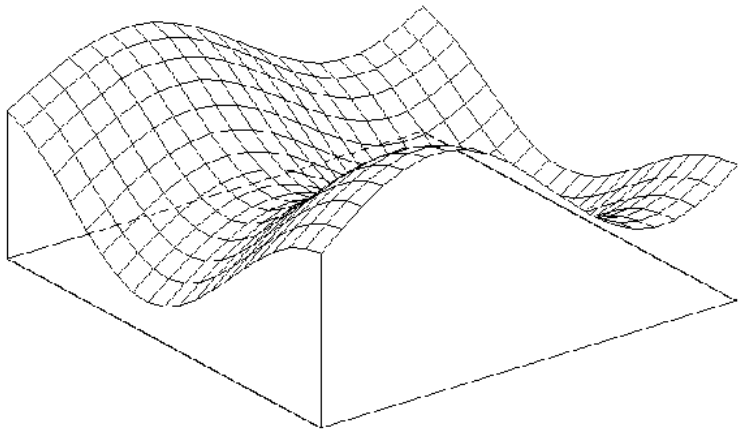


Рис. 8.4. Просте каркасне зображення з поверхні

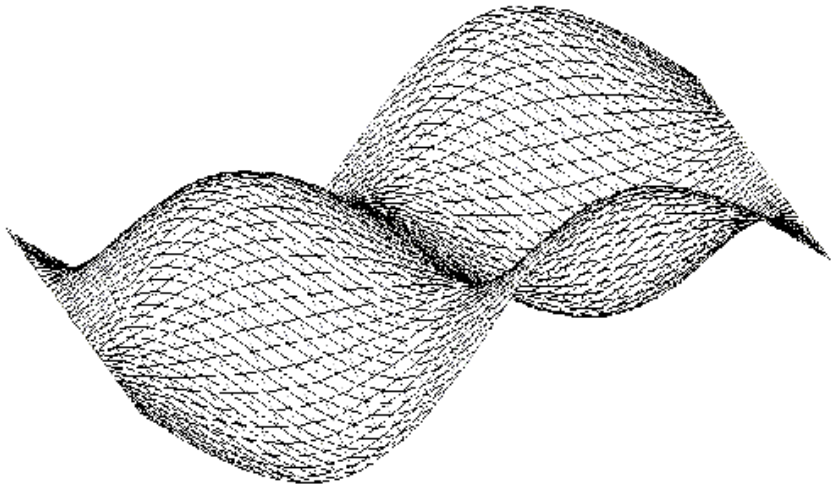


Рис. 8.5. Каркасне зображення діагональними ребрами

Алгоритм художника можна застосовувати для повністю зафарбованої сцени, а для каркасного зображення, коли об'єкт представляється у вигляді набору кривих або ламаних ліній, він непридатний. Для цього випадку запропонований ще один метод, досить ефективний – **метод плаваючого обрію**. Повернемося до попереднього прикладу зображення поверхні. Каркасне зображення виходить шляхом зображення кривих, одержуваних при перетинанні цієї поверхні площинами $x = x_i$ й $y = y_i$ (рис. 8.4).

Насправді ми будемо малювати чотирикутник і одна діагональ. У процесі малювання нам знадобляться два цілочислених масиви: $LHor$ (нижній обрій) і $HHor$ (верхній обрій) розмірністю, що відповідає горизонтальному розміру екрана в пікселях. Вони потрібні для аналізу видимості ділянок зображуваних відрізків. Спочатку ми ініціалізуєм верхній обрій нулем, а нижній – максимальним значенням вертикальної координати на екрані. Кожна виведена на екран крапка може закривати інші крапки, які "ховаються за обрієм". У міру малювання нижній обрій "опускається", а верхній "піднімається", поступово залишаючи усе менше незакритого простору. На відміну від методу художника, тут ми просуваємося від ближнього кута до далекого. Тепер опишемо алгоритм докладніше.

Функція $segment$ в цьому фрагменті призначена для виводу на екран відрізка прямій, причому в момент ініціалізації чергового пікселя (i, j) вона виконує наступні дії:

Если $(HHor[i] < j)$, то $HHor[i] = j$; вывести пиксель;

Иначе если $(LHor[i] > j)$, то $LHor[i] = j$; вывести пиксель.

Таким чином, піксель виводиться тільки в тому випадку, якщо він вище верхнього або нижче нижнього обрію, після чого його координати вже самі стають одним з обріїв. А в цілому алгоритм буде виглядати так:

Для $i = 0, \ldots, n-1$

Для $j = 0, \ldots, m-1$

$segment(x_i, y_i, x_{i+1}, y_j)$; $segment(x_i, y_i, x_{i+1}, y_{j+1})$;

$segment(x_i, y_i, x_i, y_{j+1})$.

На рис. 8.5 наведений приклад зображення поверхні з використанням цього алгоритму.

8.6. Алгоритми порядкового сканування для криволінійних поверхонь

Ідея порядкового сканування, запропонована в 1967 р. Уайли, Ромни, Евансом і Ердалом, полягала в тім, що сцена обробляється в порядку проходження скануючій прямій. В об'єктному просторі це відповідає проведенню січної площини, перпендикулярної простору зображення. Строго говорячи, алгоритм працює саме в просторі зображення, відшукуючи крапки перетинання скануючій прямій з ребрами багатокутників, що становлять картину (для випадку зображення багатогранників). Але при перетинанні чергового елемента малюнка виконується аналіз глибини отриманої крапки й порівняння її із глибиною інших крапок на скануючій площини. У деяких випадках можна побудувати список ребер, упорядкований по глибині, але найчастіше це неможливо виконати коректно. Тому сканування сполучать із іншими методами.

Один з таких методів ми вже розглядали – метод Z-Буфера, у якому буфер ініціюється заново для кожної скануючого рядка. Інший метод **називають інтервальним алгоритмом порядкового сканування**. У ньому скануючий рядок розбивається проекціями крапок перетинання ребер багатокутників на інтервали, потім у кожному з інтервалів вибираються видимі відрізки. У цій ситуації їх уже можна відсортувати по глибині. Ми зупинимося ледве докладніше на методі порядкового сканування для криволінійних поверхонь.

В описі методу пріоритетів поверхня задавалася у вигляді функції двох змінних, тут ми будемо задавати поверхню параметричними рівняннями:

$$x = x(u, v), \quad y = y(u, v), \quad z = z(u, v).$$

Перетинання скануючої площини $y = y_1$ з поверхнею дає нам так звану **лінію рівня**, або **ізолінію**. Ця крива може бути неоднозв'язної, тобто складатися з декількох окремих кривих. Щоб одержати цю криву, ми повинні вирішити рівняння $y(u, v) = y_i$ і, визначивши значення параметрів u, v, знайти крапки кривої. Для одержання рішення можна скористатися чисельними

ітераційними методами, але це вносить додаткові проблеми (наприклад, при поганому виборі початкового наближення ітераційний процес може не зійтися). Вибір підходящого методу рішення лежить поза завданнями нашого курсу, тому перейдемо до опису алгоритму, уважаючи, що він уже обраний і надійно працює.

Для каждой сканирующей строки со значением ординаты y_i :
  Для каждого значения абсциссы x_j :
    Для всех решений уравнений $u = u(x_j, y_j)$, $v = v(x_j, y_j)$
    вычислить глубину z=z(u,v).
      Определить точку с наименьшим значением глубины
      и изобразить.

Другий крок цього алгоритму припускає, що рішення відшукується тільки для тих елементів поверхні (або групи поверхонь, якщо мова йде про більше складну сцену, що містить складені об'єкти), які при даному значенні ординати перетинають скануючу площину. Попередній аналіз у деяких випадках дозволяє оптимізувати алгоритм.

8.7. Метод двійкової розбивки простору

Тепер розберемо один спосіб використання методу художника при зображенні просторових сцен, що містять кілька об'єктів або складові об'єкти. Це так званий **метод двійкової розбивки простору площинами**. Площини, як звичайно, будуть задаватися за допомогою вектора нормалі $\vec{n}$ й відстані до початку координат d (з точністю до знака). Нехай зображувана сцена складається з набору непересічних граней $F_1, F_2, \ldots, F_n$ (вони можуть мати загальні прямолінійні ділянки границі). Проведемо площину P_1, що розбиває весь простір на два півпростори, в одному йз яких перебуває спостерігач. Припустимо, що площина при цьому не перетинає ні одну із граней (але може містити ділянку її границі). Тоді грані, що перебувають в одному півпросторі зі спостерігачем, можуть заслоняти від нього частина граней із другого півпростору, але не навпаки. Це означає, що вони повинні зображуватися пізніше. Розіб'ємо площиною P_2 другий півпростір і знову визначимо, яка група граней з нього повинна зображуватися раніше. Продовжуючи цей процес до того рівня, коли весь простір буде розбито площинами на секції, у кожній з яких буде перебувати тільки одна грань, ми одержимо впорядкований набір граней. Цей порядок можна зобразити у вигляді двійкового дерева. У контексті розглянутого алгоритму це дерево являє собою структуру даних T, елементами якої є покажчик на грань зображуваної сцени, площина, що відокремлює цю грань, покажчики на ліве й праве піддерево TL й TR. Такий елемент називається вузлом дерева.

У кожному вузлі дерева ліве піддерево буде містити грані, відділені площиною, а праве – не відділені. Малювання сцени здійснюється за допомогою рекурсивного алгоритму наступного виду:

Рисуем дерево (T) :
Если наблюдатель находится в положительной полуплоскости, то:
Если правое поддерево TR не пусто, рисуем дерево (TR).
Рисуем корневую грань.
Если левое поддерево TL не пусто, рисуем дерево (TL).
Иначе
Если левое поддерево TL не пусто, рисуем дерево (TL).
Рисуем корневую грань.
Если правое поддерево TR не пусто, рисуем дерево (TR).

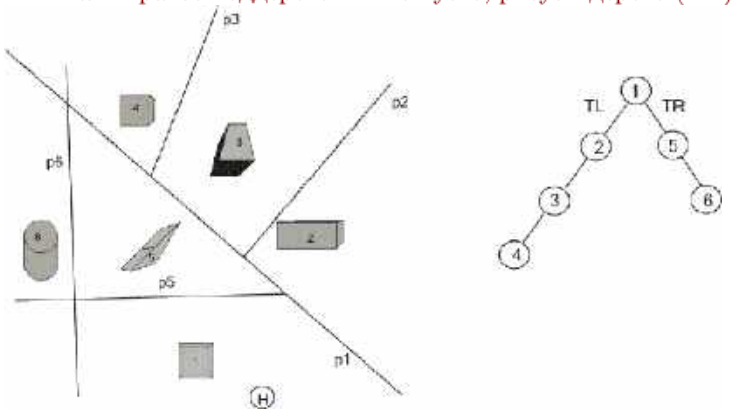


Рис. 8.6. Розбивка простору й відповідне йому дерево

Побудова площин і дерева в цьому випадку здійснюється "вручну". Для ефективності роботи алгоритму треба прагнути до того, щоб дерево було збалансованим. Якщо якісь грані не вдається відокремити, то їх перетинають площинами й малюють як два об'єкти. Спосіб визначення, по яку сторону площини перебуває спостерігач, а по яку – грань, дуже простий. Параметр площини d для кожної грані будемо задавати так, щоб грань перебувала в позитивній напівплощині. Тоді якщо при підстановці координат спостерігача в це рівняння одержуємо позитивне значення, то він перебуває в одній напівплощині із гранню, якщо ні, те в різні.

Алгоритм може застосовуватися не тільки до багатогранників, але й взагалі до будь-якої сцени за умови, що є алгоритм зображення складових її об'єктів. На рис. 8.6 зображена проекція сцени, розбитої вертикальними площинами, і відповідне їй дерево. Положення спостерігача відзначене кружком з буквою **H.** При цій точці зору об'єкти будуть зображуватися в послідовності 5, 6, 1, 2, 3, 4.

8.8. Метод трасування променів

Головна ідея цього алгоритму була запропонована в 1968 р. А. Аппелем, а перша реалізація була виконана в 1971 р.

Спостерігач бачить будь-який об'єкт за допомогою світла, що випускається якимсь джерелом, що падає на цей об'єкт, відбивається або переломлюється відповідно до законів оптики й потім якимось шляхом доходить до ока спостерігача. З величезної безлічі променів світла, випущених джерелом, лише невелика частина дійде до спостерігача. Отже, відслідковувати шляхи променів у такому порядку неефективно з погляду обчислень. Аппель запропонував

відслідковувати (трасувати) промені у зворотному напрямку, тобто від спостерігача до об'єкта. У першій реалізації цього методу трасування припинялося, як тільки промінь перетинав поверхню видимого непрозорого об'єкта; тобто промінь використовувався тільки для обробки схованих або видимих поверхонь. Згодом були реалізовані алгоритми трасування променів з використанням більше повних моделей висвітлення з урахуванням ефектів відбиття одного об'єкта від поверхні іншого, переломлення, прозорості й затінення.

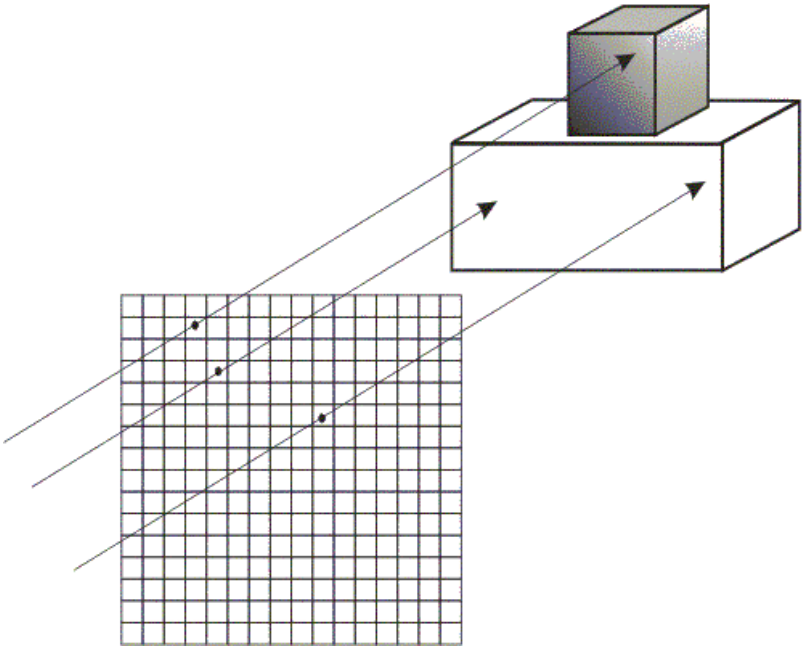


Рис. 8.7. Трасування паралельними променями

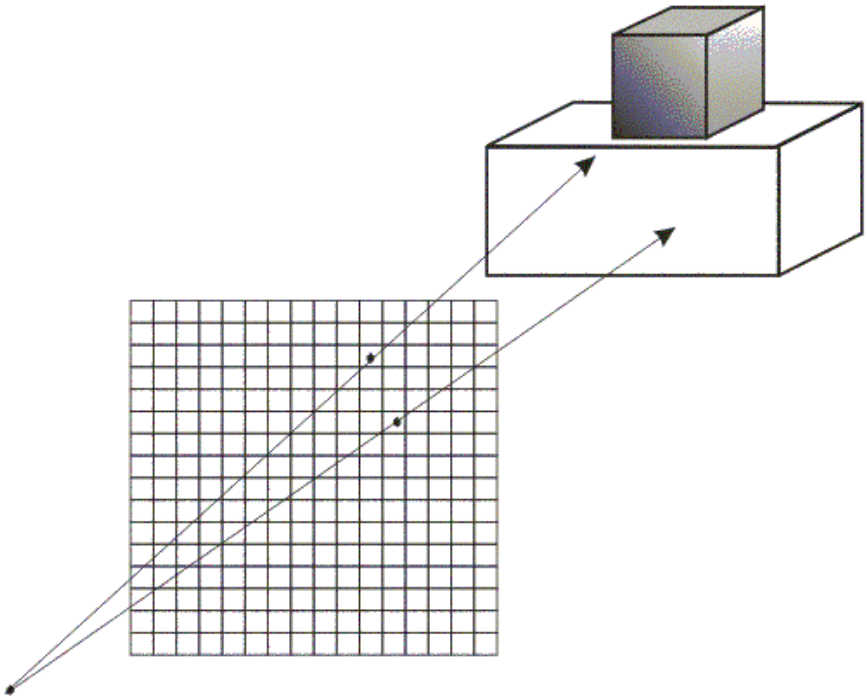


Рис. 8.8. Трасування із центральною крапкою

У цьому алгоритмі передбачається, що сцена вже перетворена в простір зображення. Якщо використовується ортографічна проекція, то точка зору або спостерігач перебуває в нескінченності на позитивній пів осі OZ. У цьому випадку всі світлові промені, що йдуть від спостерігача, паралельні осі (рис. 8.7). Кожний промінь проходить через піксель растра до сцени. Траєкторія кожного променю відслідковується, щоб визначити, які саме об'єкти сцени, якщо такі існують, перетинаються з даним променем. Необхідно перевірити перетинання кожного об'єкта сцени з кожним променем. Якщо промінь перетинає об'єкт, то визначаються всі можливі крапки перетинання променю й об'єкта. Можна одержати велику кількість перетинань, якщо розглядати багато об'єктів. Ці перетинання впорядковуються по глибині. Перетинання з максимальним значенням z представляє видиму поверхню для даного пікселя. Атрибути цього об'єкта використовуються для визначення характеристик пікселя.

Якщо точка зору перебуває не в нескінченності (перспективна проекція), алгоритм трасування променів лише незначно ускладняються. Тут передбачається, що спостерігач як і раніше перебуває на позитивній пів осі OZ. Картинна площина, тобто растр, перпендикулярна осі OZ, як показано на рис. 8.8.

Найбільш важливим і трудомістким елементом цього алгоритму є процедура визначення перетинань, оскільки це завдання віднімає найбільшу частину часу всієї роботи алгоритму. Тому ефективність методів пошуку особливо важлива. Об'єкти сцени можуть складатися з набору плоских багатокутників, багатогранників або тіл, обмежених замкнутими параметричними поверхнями. Для прискорення пошуку важливо мати ефективні критерії, що дозволяють виключити із процесу свідомо зайві об'єкти.

Одним зі способів скорочення числа пересічних об'єктів є занурення об'єктів в опуклу оболонку – сферичну або у формі паралелепіпеда. Пошук перетинання з такою оболонкою дуже простий, і якщо промінь не перетинає оболонки, те не потрібно більше шукати перетинань цього об'єкта із променем.

Особливо просто виконується тест на перетинання зі сферичною оболонкою (у лекції 3 були розглянуті завдання про перетинання променю зі сферою й площиною). Трохи більшого обсягу обчислень вимагає завдання про перетинання із прямокутним паралелепіпедом, оскільки необхідно перевірити перетинання променю щонайменше із трьома нескінченними площинами, що обмежують прямокутну оболонку. Оскільки крапки перетинання можуть виявитися поза гранями цього паралелепіпеда, для кожної з них треба, крім того, зробити перевірку на влучення усередину. Отже, для трьох вимірів тест із

прямокутною оболонкою виявляється більше повільним, чим тест зі сферичною оболонкою.

Після виконання цих первинних тестів починається процес пошуку перетинань із об'єктами, що потрапили в список потенційно видимих. При цьому завдання формування зображення не вичерпується знаходженням самої крапки перетинання: якщо ми враховуємо ефекти відбиття й переломлення, необхідно відслідковувати подальший шлях променю, для чого, як правило, потрібно відновити нормаль до поверхні, а також визначити напрямок відбитого або переломленого променю. У зв'язку з усіма цими завданнями важливо вибрати досить зручні апроксимації поверхонь, що становлять сцену. Визначення атрибутів пікселя, виведеного в остаточному підсумку на екран, залежить від вибору моделі висвітлення, про що більш докладно буде розказано в наступних лекціях.

Алгоритм трасування променів для простих непрозорих поверхонь можна представити в такий спосіб.

Створити список об'єктів, що містить:

– повний опис об'єкта: тип, поверхня, характеристики, тип оболонки й т.п.;

– опис оболонки: центр і радіус для сфери або шість значень для паралелепіпеда $(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$.

Для кожного трасуючого променю:

Виконати для кожного об'єкта тривимірний тест на перетинання з оболонкою. Якщо промінь перетинає цю оболонку, то занести об'єкт у список активних об'єктів.

Якщо список активних об'єктів порожній, то зобразити даний піксель із фоновим значенням кольору й продовжувати роботу. У противному випадку для кожного об'єкта зі списку активних об'єктів:

– Знайти перетинання з усіма активними об'єктами.

– Якщо список перетинань порожній, то зобразити даний піксель із фоновим значенням кольору.

– У противному випадку в списку припинень знайти найближче до спостерігача (з максимальним значенням z) і визначити атрибути крапки.

– Зобразити даний піксель, використовуючи знайдені атрибути пересіченого об'єкта й відповідну модель освітленості.

У цей час алгоритм трасування, незважаючи на обчислювальну складність, став дуже популярний, особливо в тих випадках, коли час формування зображення не дуже істотно, але хочеться домогтися як можна більшої реалістичності зображення.

8.9. Питання й вправи

- У чому полягає суть видалення невидимих ліній і поверхонь?
- У якому просторі працює алгоритм Робертса?
- Для яких об'єктів приміриться алгоритм Робертса?
- Що являє собою вектор-стовпець узагальненої матриці опису багатогранника?
- Як інтерпретується вираження $(\overrightarrow{r} \cdot M) \geq 0$(M – узагальнена матриця) в алгоритмі Робертса?
- У якому просторі працює алгоритм Варнока?
- Які типи розташування багатокутника щодо вікна розглядаються в алгоритмі Варнока?
- Який із шести кроків алгоритму вирішує завдання про видалення невидимих поверхонь?
- У якому просторі працює алгоритм Вейлера-Азертона?
- У чому принципова відмінність алгоритму Вейлера-Азертона від алгоритму Варнока?
- Яке узагальнення алгоритму Вейлера-Азертона запропонував Кетмул?
- Ким запропонований алгоритм Z-Буфера?
- У чому недоліки алгоритму Z-Буфера?
- На чому засновані методи пріоритетів?
- Для якого виду зображення розроблений метод художника?
- Для якого виду зображення розроблений метод плаваючого обрію?
- Що загального між алгоритмом порядкового сканування й методом Z-Буфера?
- У чому складається ідея методу трасування?
- Які бувають види трасування?
- Які прийоми використовуються для підвищення ефективності алгоритму трасування?

Тема 9. ПРОЕКТУВАННЯ ПРОСТОРОВИХ СЦЕН

Основні типи проекцій. Пряма й перспективна проекція. Спеціальні картографічні проекції. Екзотичні проекції земної сфери.

9.1. Основні типи проекцій

У математичному змісті проекції – це перетворення крапок простору розмірності *n* у крапки простору розмірності меншої, чим *n*, або, як ще говорять, **на підпростір вихідного простору**. У комп'ютерній графіці розглядаються переважно проекції тривимірного простору образа на двовимірну картинну площину. Проекція тривимірного об'єкта, представленого у вигляді сукупності крапок, будується за допомогою прямих променів, що проектують, які називаються проекторами і які виходять із центра проекції, проходять через кожну крапку об'єкта й, перетинаючи картинну площину, утворять проекцію.

Певний у такий спосіб клас проекцій називають **плоскими геометричними проекціями**, оскільки проектування в цьому випадку виробляється на **проекційну площину** і як проектори використовуються прямі. Існують і інші проекції, у яких проектування здійснюється на криволінійні поверхні або ж проектування здійснюється не за допомогою прямих (такі проекції використовуються, наприклад, у картографії).

Слід зазначити, що, приводячи ілюстрації до даної глави, ми змушені використовувати ті ж самі проекції, методи побудови яких збираємося описати. Хочеться сподіватися, що матеріал через це не буде виглядати більше неясним, чим при відсутності малюнків.

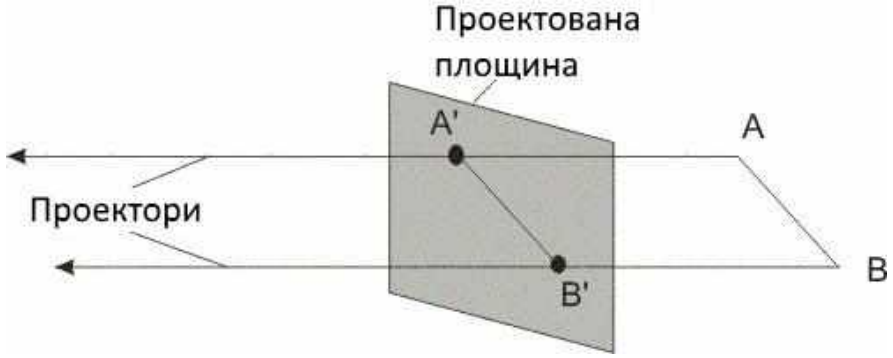


Рис. 9.1. Проекція паралельним пучком променів

Рис. 9.2. Центральна проекція

Плоскі геометричні проекції підрозділяються на два основних класи: центральні й паралельні. Розходження між ними визначається співвідношенням

між центром проекції й проекційною площиною. Якщо відстань між ними звичайно, то проекція буде центральної, якщо ж воно нескінченно, то проекція буде паралельною. Паралельні проекції названі так тому, що центр проекції нескінченно вилучений і й всі проектори паралельні. При описі центральної проекції ми явно задаємо її центр проекції, у той час як, визначаючи паралельну проекцію, ми вказуємо напрямок проектування. На рис. 9.1 і 9.2 показані дві різні проекції того самого відрізка, а також проектори, що проходять через його кінцеві крапки. Оскільки проекція відрізка сама є відрізком, то досить спроектувати одні лише кінцеві крапки й з'єднати їх.

Центральна проекція породжує візуальний ефект, аналогічний тому, до якого приводять фотографічні системи або зорова система людини, і тому використовується у випадках, коли бажано досягти певного ступеня реалістичності. Цей ефект називається **перспективним укорочуванням**: у міру збільшення відстані від центра до об'єкта розмір одержуваної проекції зменшується. Це, з іншого боку, означає, що хоча центральна проекція об'єктів є реалістичної, вона виявляється непридатної для подання точної форми й розмірів об'єктів: із проекції не можна одержати інформацію про відносні відстані; кути зберігаються тільки на тих гранях об'єкта, які паралельні проекційної площини; проекції паралельних ліній у загальному випадку не паралельні. Так, при центральній проекції куба в більшості випадків ми одержуємо картину, що взагалі не має паралельних відрізків.

Паралельна проекція породжує менш реалістичне зображення, оскільки відсутнє перспективне укорочування, хоча при цьому можуть мати місце різні постійні укорочування уздовж кожної з осей. Проекція фіксує щирі розміри (з точністю до скалярного множника), і паралельні прямі залишаються паралельними. Як і у випадку центральної проекції, кути зберігаються тільки на тих гранях об'єкта, які паралельні проекційної площини.

9.2. Паралельні проекції

Паралельні проекції розділяються на два типи залежно від співвідношення між напрямком проектування й нормаллю до проекційної площини. Якщо ці напрямки збігаються, тобто напрямок проектування є нормаллю до проекційної площини, то проекція називається **ортографічною**. Якщо ж проектори не ортогональний до проекційної площини, то проекція називається **косокутної.**

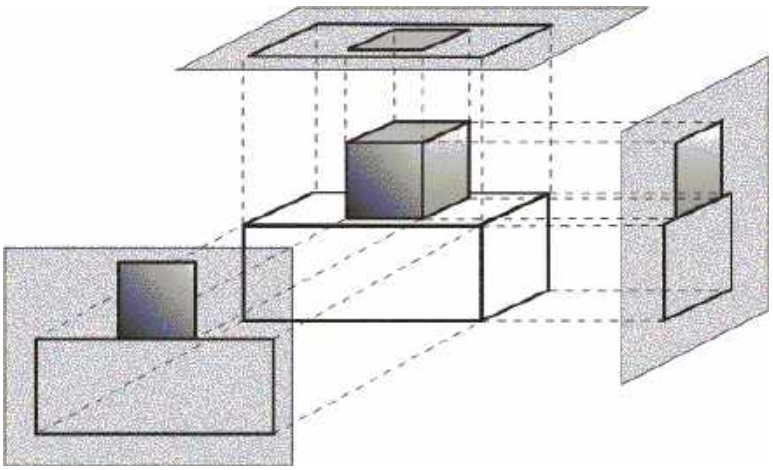


Рис. 9.3. Ортографічні проекції

В інженерній графіці найбільше широко використовуваними видами ортографічних проекцій є вид попереду, вид зверху (план) і вид збоку, у яких проекційна площина перпендикулярна головним координатним осям, що збігаються внаслідок цього з напрямком проектування (рис. 9.3). Оскільки кожна проекція відображає лише одну сторону об'єкта, часто зовсім непросто уявити собі просторову структуру проектованого об'єкта, навіть якщо розглядати відразу кілька проекцій того самого об'єкта. Але проте такі креслення дозволяють визначати реальні розміри об'єкта.

У випадку аксонометричних ортографічних проекцій використовуються проекційні площини, не перпендикулярні головним координатним осям, тому на них зображується відразу кілька сторін об'єкта, так само як і при центральному проектуванні, однак в аксонометрії укорочування постійно, тоді як у випадку центральної проекції воно пов'язане з відстанню від центра проекції. При аксонометричному проектуванні зберігається паралельність прямих, а кути змінюються; відстані ж можна виміряти уздовж кожної з головних координатних осей (у загальному випадку з різними масштабними коефіцієнтами).

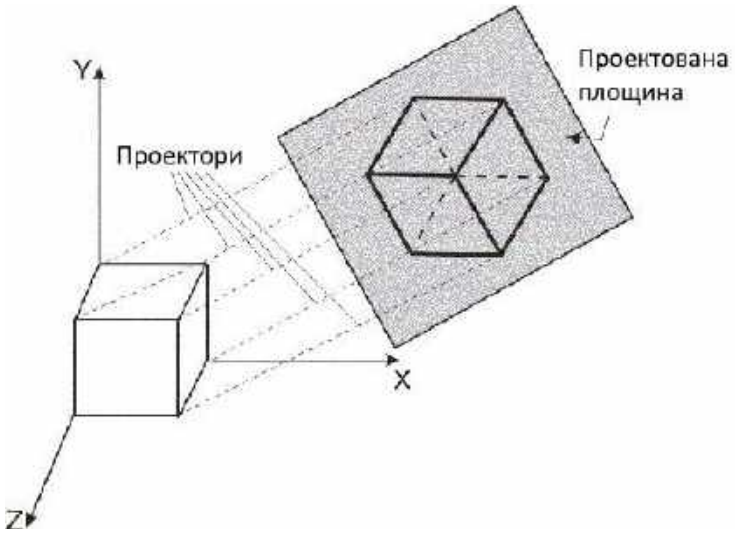


Рис. 9.4. Ізометрична проекція

Аксонометричні проекції підрозділяються на три групи відповідно до розташування проекційної площини стосовно осей координат. Якщо нормаль до проекційної площини утворить три різних кути з осями, то проекція називається **триметричною** (**триметрією**). Якщо два із цих кутів однакові, то одержуємо **диметричну** проекцію (**диметрію**). І, нарешті, якщо всі три кути рівні між собою, то проекція називається **ізометричної** (**ізометрією**). Ізометрична проекція володіє тим властивістю, що всі три головні координатні осі однаково коротшають. Тому можна проводити виміру уздовж напрямку осей з тим самим масштабом (звідси назва: "з", що означає "дорівнює", і "метрія" – "вимір"). Крім того, головні координатні осі проектуються так, що їхньої проекції становлять рівні кути один з одним (рис. 9.4).

Косокутні проекції також є паралельними, причому проекційна площина перпендикулярна головної координатної осі. Сторона об'єкта, паралельна цієї площини, проектується так, що можна вимірювати кути й відстані. Проектування інших сторін об'єкта також допускає проведення лінійних вимірів (але не кутових) уздовж головних осей. Ми торкнемося тільки двох найбільше часто використовуваних косокутних проекцій: проекції **каваль**е (**cavalier**) і **кабіні** (**cabinet**). У вітчизняній практиці ці проекції називають **горизонтальною косокутною ізометрією** й **кабінетною проекцією**.

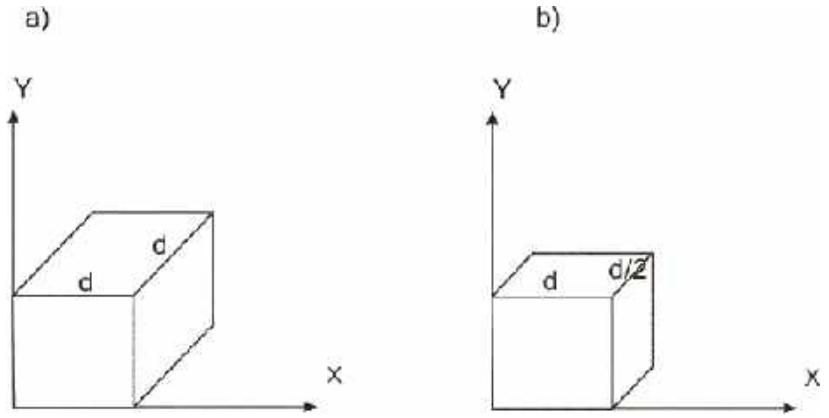


Рис. 9.5. Горизонтальна косокутна ізометрія (а) і кабінетна проекція (б)

У проекції горизонтальної косокутної ізометрії напрямок проектування становить із площиною кут 45°. У результаті проекція відрізка, перпендикулярного проекційної площини, має ту ж довжину, що й сам відрізок, тобто укорочування відсутнє (рис. 9.5а). Кабінетна проекція має напрямок проектування, що становить із проекційною площиною кут $\operatorname{arctg}(1/2)$. При цьому відрізки, перпендикулярні проекційної площини, після проектування становлять 1/2 їхньої дійсної довжини, що більше відповідає нашому візуальному досвіду, тому зображення виглядає більш реалістичною (рис. 9.5б).

9.3. Центральні проекції

Коли пучок проекторів виходить із заданого центра проекції, те паралельні відрізки на площині проекції вже не будуть паралельними, за винятком

случаючи, коли вони лежать у площині, паралельної проекційної. При проектуванні декількох паралельних прямих їхньої проекції перетинаються в так званій **крапці сходу**. Якщо сукупність прямих паралельна однієї з координатних осей, то їхня крапка сходу називається **головної**. Таких крапок може бути не більше трьох. Наприклад, якщо проекційна площина перпендикулярна осі OZ, те лише на цій осі буде лежати головна крапка сходу, оскільки прямі, паралельні як осі OX, так і OY, паралельні також і проекційної площини й тому не мають крапки сходу.

Центральні проекції класифікуються залежно від числа головних крапок сходу, якими вони володіють, а отже, і від числа координатних осей, які перетинає проекційна площина. На рис. 9.6 наведені три різні одно крапкові проекції куба, причому дві з них мають одну крапку сходу, а третя – дві крапки.

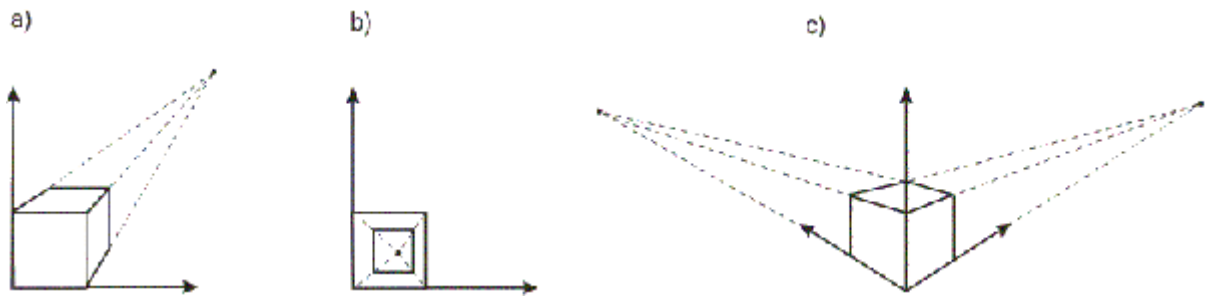


Рис. 9.6. Однокрапкові й двохкрапкові проекції

Двохкрапкова центральна проекція широко застосовується в архітектурному, інженерному й промисловому проектуванні й у рекламних зображеннях, у яких вертикальні прямі проектуються як паралельні й, отже, не сходяться. Трьохкрапкові центральні проекції майже зовсім не використовуються, по-перше, тому, що їх важко конструювати, а по-друге, через те, що вони додають мало нового з погляду реалістичності в порівнянні із двохкрапковою проекцією.

9.4. Математичний апарат

Для виконання проективних перетворень будемо використовувати однорідні координати й матриці перетворень, розглянуті раніше в лекції 4. Проекція виконується в системі координат спостерігача.

9.5. Ортогональні проекції

Спочатку розглянемо математичний опис паралельних проекцій як більше простих. Випадок, коли картинна площина перпендикулярна осі OZ й задається рівнянням $z = 0$(тобто ортографічна проекція), фактично вже розглядався в лекції 4, де був наведений вид матриць проекції на координатні площини.

Випадок аксонометричної проекції зводиться до послідовності перетворень, подібно тому як здійснювався поворот у просторі щодо довільної осі. Нехай площина задається одиничним вектором нормалі $\vec{n} = (n_x, n_y, n_z)$ й

відстанню від початку координат $d \geq 0$. Канонічне рівняння площини, таким чином, має вигляд $n_x \cdot x + n_y \cdot y + n_z \cdot z - d = 0.$

Вектор, спрямований по нормалі від початку координат до перетинання із площиною, є $\overrightarrow{N} = d \cdot \overrightarrow{n} = (n_x d, n_y d, n_z d) = (N_x, N_y, N_z).$

Координати вектора одиничної нормалі є її напрямними косинусами.

Проектування в просторі однорідних координат здійснюється наступною послідовністю кроків.

– Зрушення на вектор $-\overrightarrow{N}$ за допомогою матриці

$$S_1 = \begin{pmatrix} 1 & 0 & 0 & -N_x \\ 0 & 1 & 0 & -N_y \\ 0 & 0 & 1 & -N_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

–

– Поворот, що сполучає напрямок нормалі з напрямком осі OZ. Як було показано в лекції 4, цей поворот можна реалізувати у вигляді двох поворотів: перший (щодо осі OZ) переводить нормаль у площину YOZ, а потім – поворот щодо осі OY до сполучення нормалі з віссю OZ. Відповідну матрицю обертання, що є добутком двох матриць, позначимо R.

– Проекція на площину XOY за допомогою матриці

$$P_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

–

– Поворот за допомогою матриці R^{-1}.

– Зрушення на вектор $\overrightarrow{N}$ за допомогою матриці S_1^{-1}.

Повне перетворення, таким чином, визначається матрицею

$\Pr = S_1^{-1} \cdot R^{-1} \cdot P_{xy} \cdot R \cdot S_1.$

9.6. Косокутні проекції

Розглянемо косокутну проекцію на площину XOY, при якій орт $\overrightarrow{e}_3 = (0,0,1)$ переходить у вектор $\overrightarrow{r}_0 = (a,b,0)$, тобто напрямок проекції задається вектором $\overrightarrow{p} = \overrightarrow{r}_0 - \overrightarrow{e}_3 = (a,b,-1)$. Таке перетворення в просторі однорідних координат можна задати за допомогою матриці

$$P = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

У проекції кавальє вектор $\overrightarrow{e}_3$ переходить у вектор $(\cos(\pi/4), \cos(\pi/4), 0)$, а в кабінетній проекції – у вектор $(0.5 \cdot \cos(\pi/4), 0.5\cos(\pi/4), 0)$, причому в обох проекціях $a = b$.

9.7. Центральні проекції

Припустимо, що центр проекції перебуває в крапці $\overrightarrow{c} = (c_x, x_y, c_z)$, а картинна площина збігається із площиною XOY. Візьмемо довільну крапку зображуваного об'єкта $\overrightarrow{M} = (x, y, z)$ й визначимо її проекцію на обрану площину (рис. 9.7).

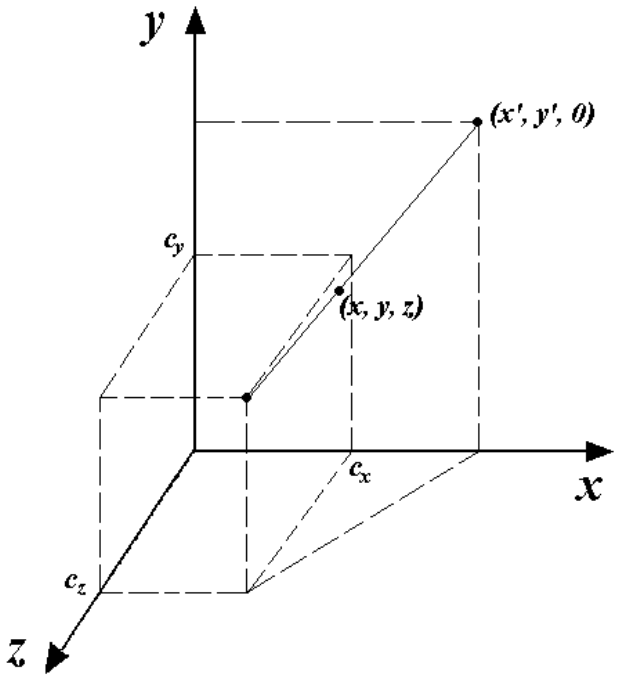


Рис. 9.7. Центральна проекція на площину XOY

Пряму, що проходить через крапки $\overrightarrow{c}$ й $\overrightarrow{M}$, задаємо в параметричному виді:

$$\overrightarrow{r}(t) = \overrightarrow{c} + t \cdot (\overrightarrow{M} - \overrightarrow{c}) = (c_x + t \cdot (x - c_x), c_y + t \cdot (y - c_y), c_z + t \cdot (z - c_z)). \quad (9.1)$$

Тепер знайдемо крапку перетинання цій прямій з картинною площиною. Вона визначається з умови рівності нулю третьої координати:

$$c_z + t \cdot (z - c_z) = 0,$$

звідки визначаємо значення параметра t, при якому крапка прямої належить координатній площині:

$$t^* = \frac{c_z}{c_z - z} = \frac{1}{1 - \frac{z}{c_z}}.$$

Підставляючи це значення у формулу (9.1), ми одержимо координати проекції крапки $\overrightarrow{M}$:

$$x^* = c_x + \frac{x - c_x}{1 - \frac{z}{c_z}}, \quad y^* = c_y + \frac{y - c_y}{1 - \frac{z}{c_z}}. \quad (9.2)$$

Фактором, що впливає на перспективну зміну розмірів, є наявність координати z в знаменнику. Чим ближче виявляється крапка до центра проекції, тим більше знаменник, а відповідно й координати крапки.

Ми будемо розглядати ситуацію, коли центр проекції лежить на осі OZ, а сама вісь спрямована від спостерігача до проекційної площини, тобто $c_x = x_y = 0, \quad c_z = -d, \; d > 0$. Тоді формули (9.2) здобувають вид

$$x^* = \frac{x}{1 + \frac{z}{d}}, \quad y^* = \frac{y}{1 + \frac{z}{d}}. \tag{9.3}$$

В однорідних координатах таке перетворення можна записати за допомогою двох операцій. Спочатку множимо матрицю проективного перетворення P_z на вихідну точку й одержуємо крапку в чотирьохвимірному просторі:

$$P_z \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 + z/d \end{pmatrix}. \tag{9.4}$$

Потім проектуємо цю крапку в простір однорідних координат шляхом розподілу на четвертий компонент:

$(x^*, y^*, 0, 1) = (x/p, y/p, 0, 1), \quad p = 1 + z/d.$

Подивимося тепер, що відбувається з пучком паралельних прямих під дією матриці проекції. Нехай заданий пучок прямих, паралельних вектору $\vec{v} = (a, b, c)$. Тоді параметричне рівняння прямої, що належить цьому пучку, має вигляд $\vec{r} = (x + at, y + bt, z + ct)$.

З формули (9.4) треба, що в результаті проектування одержимо безліч крапок

$$\left(x + at, y + bt, 0, 1 + \frac{z + ct}{d} \right).$$

Переходячи до однорідних координат і помноживши чисельник і знаменник кожного дробу на d, одержимо крапки $\vec{r}_0$ виду

$$\vec{r}_0 = \left(d\frac{x + at}{d + z + ct}, d\frac{y + bt}{d + z + ct}, 0, 1 \right).$$

Тепер у кожному компоненті вектора чисельник і знаменник поділимо на t:

$$\vec{r}_0 = \left(d\frac{x/t + a}{(d + z)/t + c}, d\frac{y/t + b}{(d + z)/t + c}, 0, 1 \right).$$

Переходячи до межі при $t \to \infty$, одержимо крапку

$$\vec{r}_\infty = \left(\frac{da}{c}, \frac{db}{c}, 0, 1 \right).$$

Таким чином, одержуємо, що після проектування пучок паралельних прямих перетинається в точці сходу $\vec{r}_\infty$. Зрозуміло, що в кожного пучка своя крапка сходу. Якщо пучок прямих паралельний площини XOY, тобто $c = 0$, то крапка сходу виявляється на нескінченності, а виходить, прямі залишаються паралельними.

Для побудови перспективної проекції з декількома крапками сходу використовується матриця перспективного перетворення без проектування:

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/d_1 & 1/d_2 & 1/d_3 & 1 \end{pmatrix}.$$

Тепер крапки простору спочатку піддаються перспективному перетворенню, а потім здійснюється проекція.

Визначимо крапки сходу для прямих, паралельних осям координат. Для прямих $\overrightarrow{r} = (x, y, z + ct)$ результатом проективного перетворення буде безліч крапок $(x/f, y/f, (z + ct)/f, 1)$, де $f = x/d_1 + y/d_2 + (z + ct)/d_3 + 1$. При $t \to \infty$ одержимо крапку з координатами $(0, 0, d_3, 1)$. При проекції на площину XOY одержимо крапку $(0, 0)$. Пучок прямих $\overrightarrow{r} = (x + ct, y, z)$ перейде в $((x + ct)/f_1, y/f_1, z/f_1, 1)$, $f_1 = (x + ct)/d_1 + y/d_2 + z/d_3 + 1$, а крапкою сходу для нього буде $(d_1, 0, 0, 1)$, що при проектуванні перейде в крапку, що лежить на осі OX $(d_1, 0)$. Аналогічно для пучка прямих, паралельні осі OY, одержимо крапку сходу на осі OY $(0, d_2)$. Ці три крапки на площині є головними крапками сходу.

9.8. Спеціальні картографічні проекції. Екзотичні проекції земної сфери

З розвитком торгівлі й подорожей придбала більшу важливість досить непросте геометричне завдання: як перенести на площину частина земної поверхні, щоб відстані між будь-якими двома крапками на ній залишилися неспотвореними? Різні вчені протягом багатьох століть намагалися розв'язати цю проблему по замовленнях своїх урядів, великих комерсантів або просто мандрівників, для яких така карта була б справжньою знахідкою, тому що істотно полегшила б навігацію, як морську, так і повітряну.

У цілому це завдання виявилося нерозв'язною. Поверхня циліндра або конуса можна без перекручувань перенести на площину (такі поверхні називаються що **розгортаються**), відобразити ж на площину поверхня сфери, зберігши відстань між будь-якими двома крапками, неможливо. Справа в тому, що навіть малу область сферичної поверхні (на відміну від циліндра або конуса) неможливо розгорнути на площині без тріщин, складок або перекручувань. Будь-яка плоска карта Землі або якоїсь її частини неодмінно буде спотворювати які-небудь властивості. Тому в цей час так необхідні карти з мінімальними (ще краще нульовими) перекручуваннями тих властивостей, для передачі яких призначається карта. Бажано, щоб і інших властивостях деформувалися якнайменше. Усього існує чотири основних типи перекручувань:

– перекручування довжин (лінії, однакові на поверхні Землі, зображуються на карті відрізками різної довжини);

– перекручування кутів (кути на карті між узятими напрямками не дорівнюють горизонтальним кутам між тими ж напрямками на поверхні земного еліпсоїда);

– перекручування форм (форма ділянки або зайнятий об'єктом території на карті відмінна від їхньої форми на поверхні Землі);

– перекручування площ (пов'язане з масштабом площі: при сталості величини масштабу площі по всій поверхні карти перекручування площ на ній немає).

Крім класичних карт, більша частина яких була розроблена в середні століття, фантазія вчених надавала навігаторам досить незвичайні способи проекції земної поверхні, але перш ніж описати способи складання незвичайних карт, розглянемо деякі класичні методи картографії.

Центр проекції може бути довільним стосовно проектованої сфери; таким чином, існує нескінченна безліч усіляких різних проекцій. Якщо проводити промені з деякої крапки, узятої на прямій, що проходить через центр кулі перпендикулярно деякої площини, то одержимо на цій площині перспективну проекцію. Розглянемо деякі із цих проекцій, найбільш корисні з погляду картографії.

9.9. Стереографічна проекція

Важлива властивість будь-якої карти – збереження кутів (кут між будь-якими двома лініями на карті повинен бути таким же, як кут між прообразами цих ліній на земній поверхні). Збереження кутів особливо важливо для мореплавання й аеронавтики, тому що воно означає, що спостережуваний кут між будь-якими двома орієнтирами дорівнює куту, вимірюваному на карті за допомогою транспортира. Крім того, на такій карті залишаються незмінними й площі малих областей. Карти, що зберігають кути, називаються **конформними.** Найпростіше побудувати конформну карту за допомогою **стереографічної** проекції.

На рис. 9.8 показано, як поверхня сфери в крапці X проектується із крапки A (приналежній сфері) на площину, дотичну до сфери в діаметрально протилежній крапці (**антипод** крапки A). Проекція називається екваторіальною, полярної або косою залежно від того, де перебувають антиподи: на екваторі, полюсах або в якій-небудь іншій крапці земної поверхні відповідно. На жаль, конформовість викликає перекручування масштабу, що зростає зі збільшенням відстані від центра карти.

Рис. 9.8. Три проекції

Позначимо за довготу й доповнення до широти крапки на сфері буквами λ й θ відповідно, $0 \le \lambda \le 360°$, $0 \le \theta \le 180°$, а через x і y – координати проекцій цієї крапки в деякої декартової системі координат, заданої на площині проекції. Відповідні формули проектування для Північної півкулі мають такий вигляд:

$$x = \operatorname{tg}\theta/2 \cos(\lambda - 135°);$$

$$y = \operatorname{tg}\theta/2 \sin(\lambda - 135°)$$

Тут і надалі радіус уважається рівним одиниці. Вісь X спрямована уздовж меридіана 135°.

9.10. Гномонічна проекція

Відображення крапки X із центра земної кулі B на площину карти в крапку B' породжує **гномоничну** проекцію (рис. 9.8). Проекція одержала таку назву, тому що вона нагадує конструкцію сонячних годин із гномоном. Будь-яка дуга великого кола на поверхні земної кулі переходить у пряму на гномонічній карті. **Більшим колом** називається окружність на сфері, площина якої проходить через її центр. Така карта не володіє комфортністю, але навігатори цінують її за одна важлива властивість, відсутнє у всіх інших проекцій сфери на площину: пряма між будь-якими двома крапками на гномонічній карті є геодезичної, або найкоротшою дугою між цими двома крапками, і відповідає дузі великого кола на поверхні Землі.

9.11. Ортографічна проекція

Якщо центр проекції перебуває в нескінченності (всі промені, що проектують, паралельні), то проекція буде ортографічної (рис. 9.8). Наприклад, дивлячись на Місяць із Землі, спостерігач бачить Місяць практично в ортографічної проекції. У краю ортографічної карти відстані сильно перекручені. Ортографічна карта не зберігає ні площ, ні кутів, але, виконана досить мистецьки, створює сильну ілюзію кулястої Землі. Карти, накреслені з

погляду спостерігача, що перебуває над земною поверхнею, не точні в передачі багатьох її властивостей, але найбільше вірно відповідають нашому зоровому сприйняттю сфери.

Ця проекція виходить при проектуванні на площину, дотичну до сфери в центрі зображуваного явища (λ_0, θ_0), за допомогою променів, перпендикулярних цієї площини. Формули цієї проекції наступні:

$$x = \sin\theta \sin(\lambda - \lambda_0)$$

$$y = \sin\theta_0 \cos\theta - \cos\theta_0 \sin\theta \cos(\lambda - lambda_0)$$

9.12. Проекції на циліндр

Поверхня сфери також можна проектувати на циліндри й конуси, "надягнуті" на сферу. Після побудови циліндричної або конічної проекції поверхня розрізається й розгортається на площину.

Промені, що проектують земна куля на циліндр, вибираються так, щоб вони були паралельні площини, що висікає окружність, по якій сфера й циліндр стикаються (рис. 9.9). Якщо циліндр стосується Землі уздовж екватора, то всі меридіани й паралелі на карті переходять у прямі, що перетинаються під прямими кутами.

Рис. 9.9. Метод циліндричної проекції зі збереженням площ

Циліндрична карта не завжди володіє комфортністю й може сильно спотворювати відстані й форму областей. Відзначимо, що жодна карта не може одночасно бути конформної й зберігати площі. Було запропоновано величезне число інших проекцій, що зберігають площу; у сучасних атласах найчастіше зустрічаються площі, що зберігають, карти, побудовані за допомогою циліндричної проекції, що була запропонована Карлом Б.М ольвейде в 1805 р.

9.13. Проекція Меркатора

В XVI столітті фламандський картограф Герхард Меркатор створив знамениту циліндричну проекцію, що володіє властивістю комфортності. Комфортність у проекції Меркатора досягається за рахунок розтягування циліндра за полюси, при цьому у верхній і нижній частині цього циліндра масштаб стає дуже перекрученим. Незважаючи на це дана проекція володіє однією чудовою властивістю, дуже потрібним для навігаторів: пряма, проведена через будь-які дві крапки на карті, є локсодромою, або лінією постійного румба. Локсодрома на сфері або якій-небудь іншій поверхні обертання перетинає всі меридіани під постійним кутом (рис. 9.10).

Проекція Меркатора задається наступними формулами:

$$x = \begin{cases} \lambda/180^\circ \text{ при } 0^\circ \le \lambda \le 180^\circ, \\ \lambda/180^\circ - 2 \text{ при } 180^\circ \le \lambda \le 360^\circ \end{cases}$$

$$y = \ln(\mathrm{tg}(90^\circ - 0.5\theta))/\pi.$$

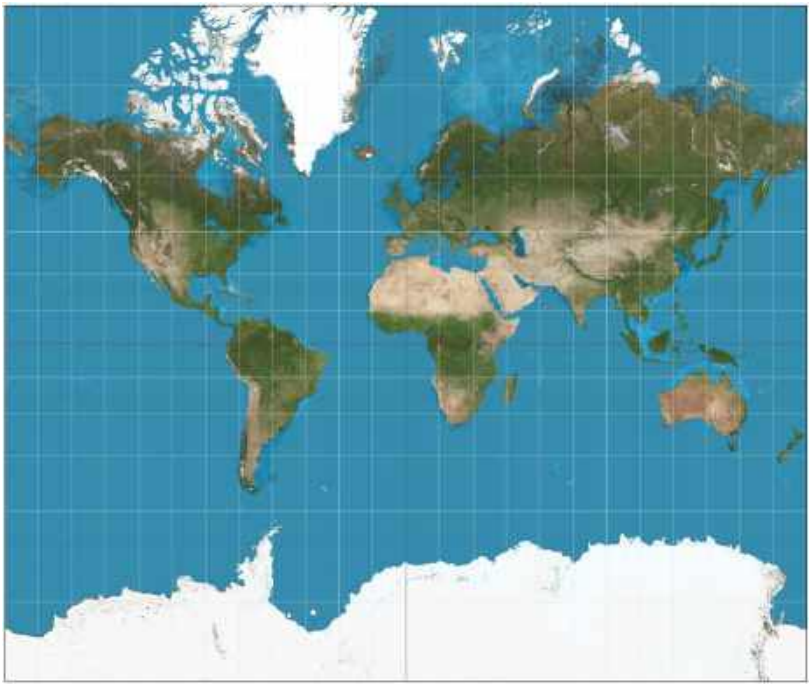


Рис. 9.10. Комфортна проекція Меркатора.
На карту нанесені локсодроми з Нью-Йорка

9.14. Проекції на багатогранник

Будемо називати **розрізану** карту миру, спроектовану на той або інший візерунок з яких-небудь багатокутників. Після складання цих фрагментів утвориться карта з розривами будь-якої частини земної кулі. Одну таку конформну карту склав філософ і математик Ч. Пірс. Земна поверхня спроектована на цій карті на вісім рівнобедрених трикутників, які можна розглядати як грані октаедра, що сплющується доти, поки довжина його просторової діагоналі не звернеться в нуль. Вершинам нульової діагоналі на карті Пірса відповідають північний і південний полюси.

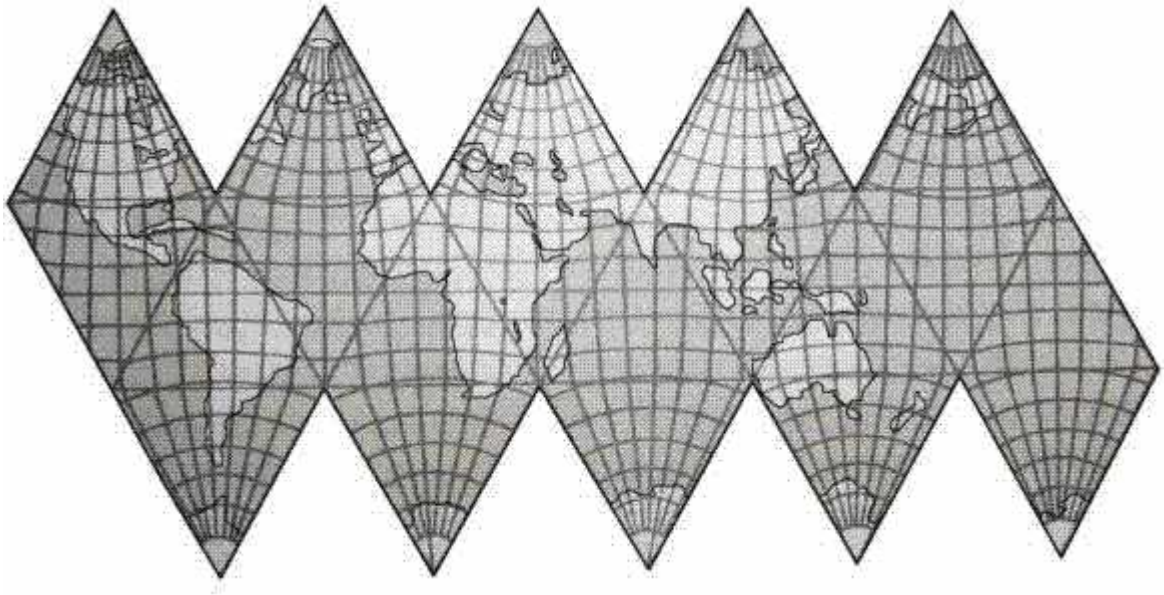


Рис. 9.11. "Лінкаглобус" И. Фишера, що складається в ікосаедр

Приблизно в той же час аналогічна ідея спала на думку видатному економістові з Єльського університету Ірвінгу Фішеру: він задумав здійснити гномонічну проекцію поверхні Землі на 20 трикутних гранях ікосаедра (рис. 9.11). Ікосаедр є найбільш близьким до ідеалу багатогранником для розрізування "на карти".

9.15. Незвичайні проекції

Картографи придумували найрізноманітніші види проекцій, що часом дивно виглядають і при цьому володіють досить непоганими властивостями. Одну з таких карт у формі кардіоїди (серця) придумав Іоганн Вернер. Ця карта зберігає площі. Вона користувалася широкою популярністю в XVI в., але зараз нею вже давно не користуються. У звіті про картографічні курйози, написаному для внутрішнього користування фірми "Лабораторії Белла", математик Едгар Н. Гілберт пише: "...незаслужено забута. Сильно перекручені частини карти лежать далеко від основних мас суші. Скривлені паралелі надають карті приємну ілюзію округлості... Паралелі являють собою дуги окружностей, розташовані на однаковій відстані друг від друга, із центром у північному полюсі. Меридіани, проведені для вказівки відстаней уздовж паралелей, такі ж як на сфері".

Скупчення материків на карті Вернера відбиває нерівномірний розподіл суші по поверхні Землі. Тихий океан настільки великий, що якщо дивитися на Землю із крапки, розташованої над протокою Ла-Манш, то погляду відкриється близько 80% всієї суші, а протилежна півкуля буде майже суцільно покрито водою

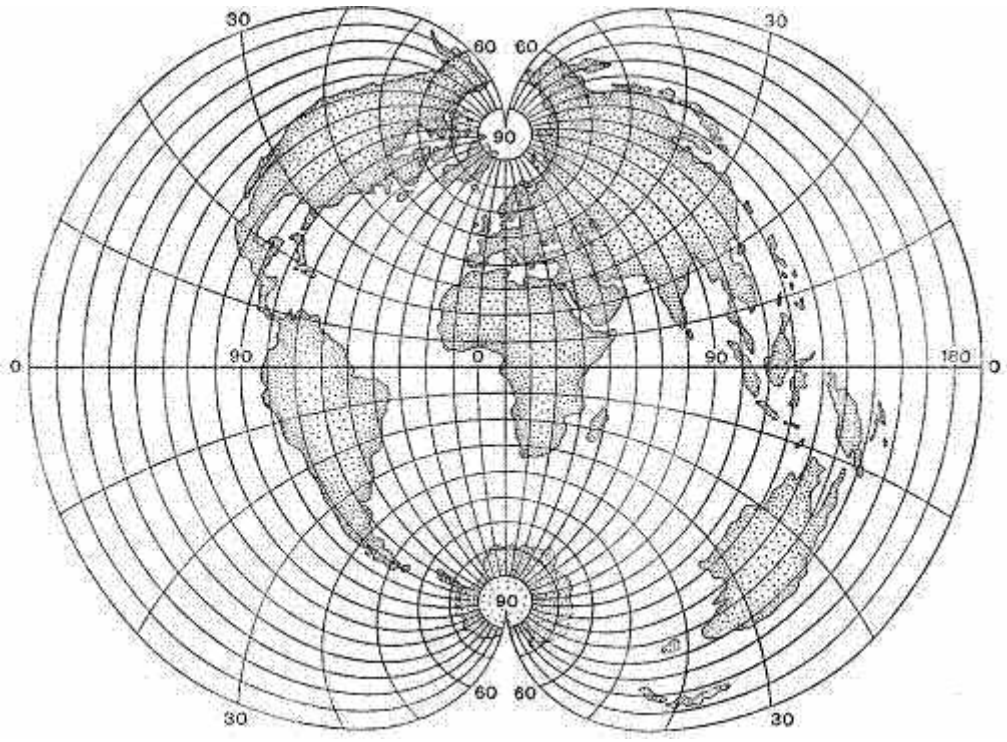


Рис. 9.12. Поліконічна картографічна проекція

Досить незвичайна поліконічна картографічна проекція – картографічна проекція, що будується за допомогою ряду конусів, дотичних до земного еліпсоїда (кулі). У поліконічній картографічній проекції паралелі нормальної сітки – дуги ексцентричних окружностей, осьовий меридіан – пряма, на якій розташовані центри паралелей, а інші меридіани – криві (рис. 9.12).

9.16. Питання й вправи

1. Назвіть два основних види проекцій, обумовлених типом пучка променів.
2. Назвіть чотири види паралельних проекцій.
3. Скільки кроків в алгоритмі ортогональної проекції на довільну площину?
4. Який вид має матриця косокутної проекції на площину XOY, що переводить вектор $(0,0,1)$ у вектор $(x,y,0)$?
5. Напишіть формули перетворення координат при центральній проекції на площину XOY із центром у крапці $(0,0,c)$. Як виглядає матриця такої проекції в однорідній системі координат?
6. Що таке перспективне укорочування?
7. Що таке крапка сходу?
8. Як реалізується проекція із трьома крапками сходу?
9. Якою властивістю володіє конформна проекція?
10. Якою властивістю володіє циліндрична проекція?
11. У чому цінність проекції Меркатора?
12. Який багатогранник найбільш зручний для побудови розрізаних карт?

Тема 10. РАСТРОВЕ ПЕРЕТВОРЕННЯ ГРАФІЧНИХ ПРИМІТИВІВ

Алгоритми Брезенхема растрової дискретизації відрізка. Алгоритми Брезенхема растрової дискретизації кола й еліпса. Алгоритми заповнення внутрішніх областей

Екран растрового дисплея можна розглядати як матрицю дискретних елементів, або пікселів. Процес визначення пікселів, щонайкраще апроксимуючу деяку геометричну фігуру, називається розкладанням у растр, або побудовою растрового образа фігури. Порядкова візуалізація растрового образа називається растровим розгорненням даної фігури.

10.1. Алгоритм Брезенхема растрової дискретизації відрізка

При побудові растрового образа відрізка необхідно, насамперед, установити критерії "гарної" апроксимації. Перша вимога полягає в тому, що відрізок повинен починатися й кінчатися в заданих крапках і при цьому виглядати суцільним і прямим (при досить високому дозволі дисплея цього можна домогтися). Крім того, яскравість уздовж відрізка повинна бути однакової й не залежати від нахилу відрізка і його довжини. Ця вимога виконати складніше, оскільки горизонтальні й вертикальні відрізки завжди будуть яскравіше похилих, а постійна яскравість уздовж відрізка знову ж досягається на вертикальних, горизонтальним і нахилених під кутом в 45° лініях. І, нарешті, алгоритм повинен працювати швидко. Для цього необхідно по можливості виключити операції з речовинними числами. З метою прискорення роботи алгоритму можна також реалізувати його на апаратному рівні.

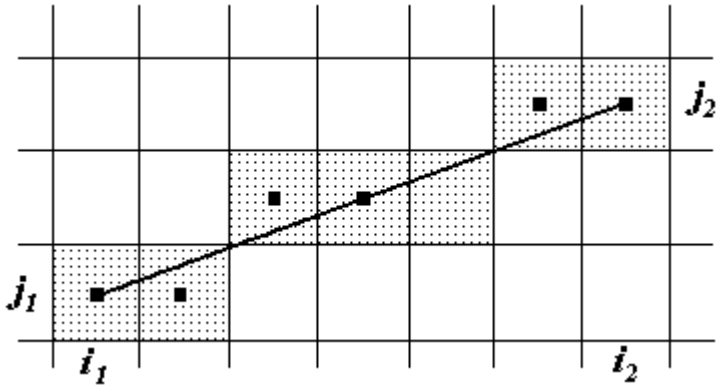


Рис. 10.1. Растровий образ відрізка

У більшості алгоритмів використовується покроковий метод зображення, тобто для знаходження координат чергової крапки растрового образа нарощується значення однієї з координат на одиницю растра й обчислюється збільшення іншої координати.

Завдання полягає в побудові відрізка, що з'єднує на екрані крапки з координатами $(i_1, j_1), (i_2, j_2)$ (будемо вважати, що $i_1 \neq i_2$). Для побудови відрізка прямій на площині з речовинними координатами можна скористатися рівнянням прямої, що проходить через дві задані крапки, що має вигляд

$j = j_1 + k(i - i_1), \quad k = (j_2 - j_1)/(i_2 - i_1).$

Тепер, уважаючи, що $0 \le k \le 1$, (i, j) – координати поточної крапки растрового образа, а y – точне значення координати крапки відрізка, можна побудувати наступну крапку: $i' = i + 1, \quad j' = y + k$

Варто помітити, що цілочислова координата j зміниться тільки в тому випадку, якщо у перевищить величину $j + 0.5$ (j' є найближче до y ціле число, отримане в результаті операції округлення). Наведений приклад включає операції з речовинними числами, які виконуються істотно повільніше, ніж відповідні цілочислові операції, а при побудові растрового образа відрізка бажаний алгоритм, по можливості звертається тільки до цілочислової арифметики. Крім того, алгоритм повинен працювати при будь-якому взаємному розташуванні кінців відрізка.

Алгоритм Брезенхема побудови растрового образа відрізка був споконвічно розроблений для графобудівників, але він повністю підходить і для растрових дисплеїв. У процесі роботи залежно від кутового коефіцієнта відрізка нарощується на одиницю або i, або j, а зміна іншої координати залежить від відстані між дійсним положенням крапки й найближчою крапкою растра (зсуву). Алгоритм побудований так, що аналізується лише знак цього зсуву.

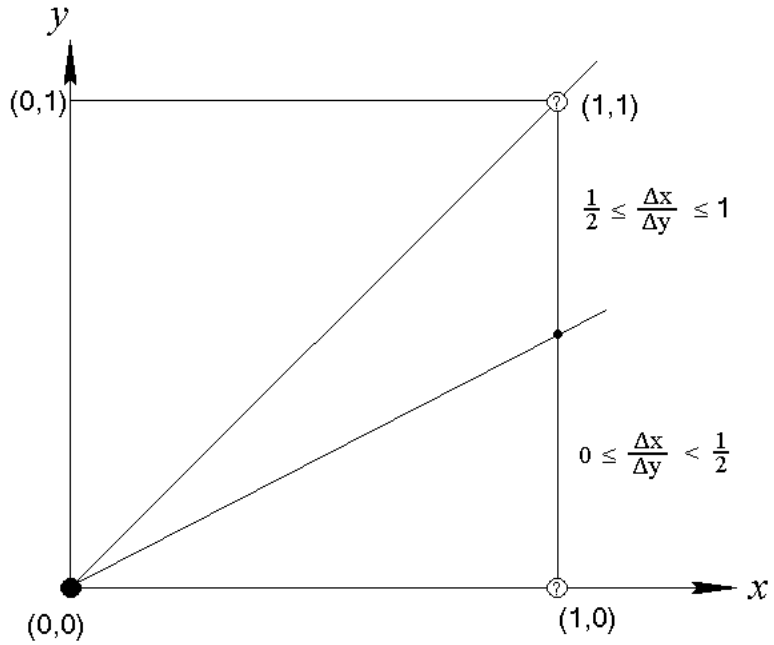


Рис. 10.2. Зв'язок кутового коефіцієнта з вибором пікселя

На рис. 10.2 це ілюструється для відрізка з кутовим коефіцієнтом, що лежить у діапазоні від нуля до одиниці. З малюнка можна помітити, що якщо кутовий коефіцієнт $k \ge \frac{1}{2}$, то при виході із крапки $(0,0)$ перетинання із прямої $x = 1$ буде ближче до прямої $y = 1$, чим до прямої $y = 0$. Отже, крапка растра $(1,1)$ краще апроксимує проходження відрізка, чим крапка $(1,0)$. При $k < \frac{1}{2}$ вірно зворотне.

На рис. 10.3 показано, яким образом будуються крапки растра для відрізка з тангенсом кута нахилу $3/8$, а на рис. 10.4 – графік зсуву. На початку побудови зсув покладається рівним $k-1/2$, а потім на кожному кроці воно нарощується на величину k, і якщо при цьому вертикальна координата крапки растра збільшується на одиницю, то зсув у свою чергу зменшується на одиницю.

На рис. 10.5 наведена блок-схема алгоритму для випадку $i_2 > i_1, \quad j_2 > j_1, \quad k > 0$. Неважко зрозуміти, як від цього алгоритму перейти до цілочислового: досить замість величини зсуву e перейти до величини $\tilde{e} = 2\Delta i \cdot e$.

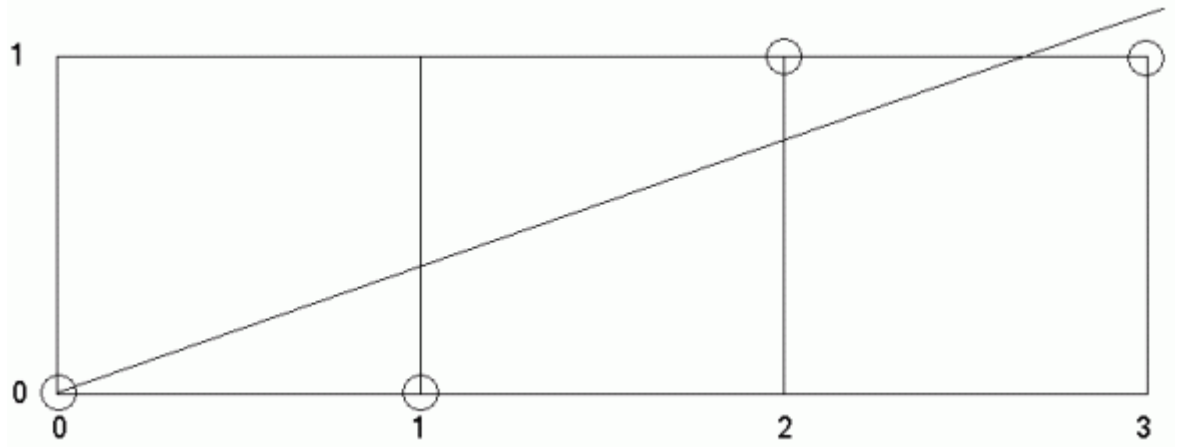


Рис. 10.3. Пікселі, що належать розгорненню відрізка

Рис. 10.4. Графік зміни відхилення

Приведемо загальний алгоритм Брезенхема, що враховує всі можливі випадки напрямку відрізка, розглянутого як вектор на координатній площині (на рис. 10.6 виділені чотири області й зазначені особливості алгоритму в кожній з них).

В описі алгоритму використовуються наступні функції:
swap (a, b): обмін значень змінних a, b;
abs (a): абсолютне значення a;
sign (a): 0, якщо a= 0, 1, якщо a>0, -1, якщо a<0;
point (i, j) - ініціалізація крапки (i, j).

Передбачається, що кінці відрізка $(i_1, j_1), (i_2, j_2)$ не збігаються й що всі використовувані змінні є цілими.

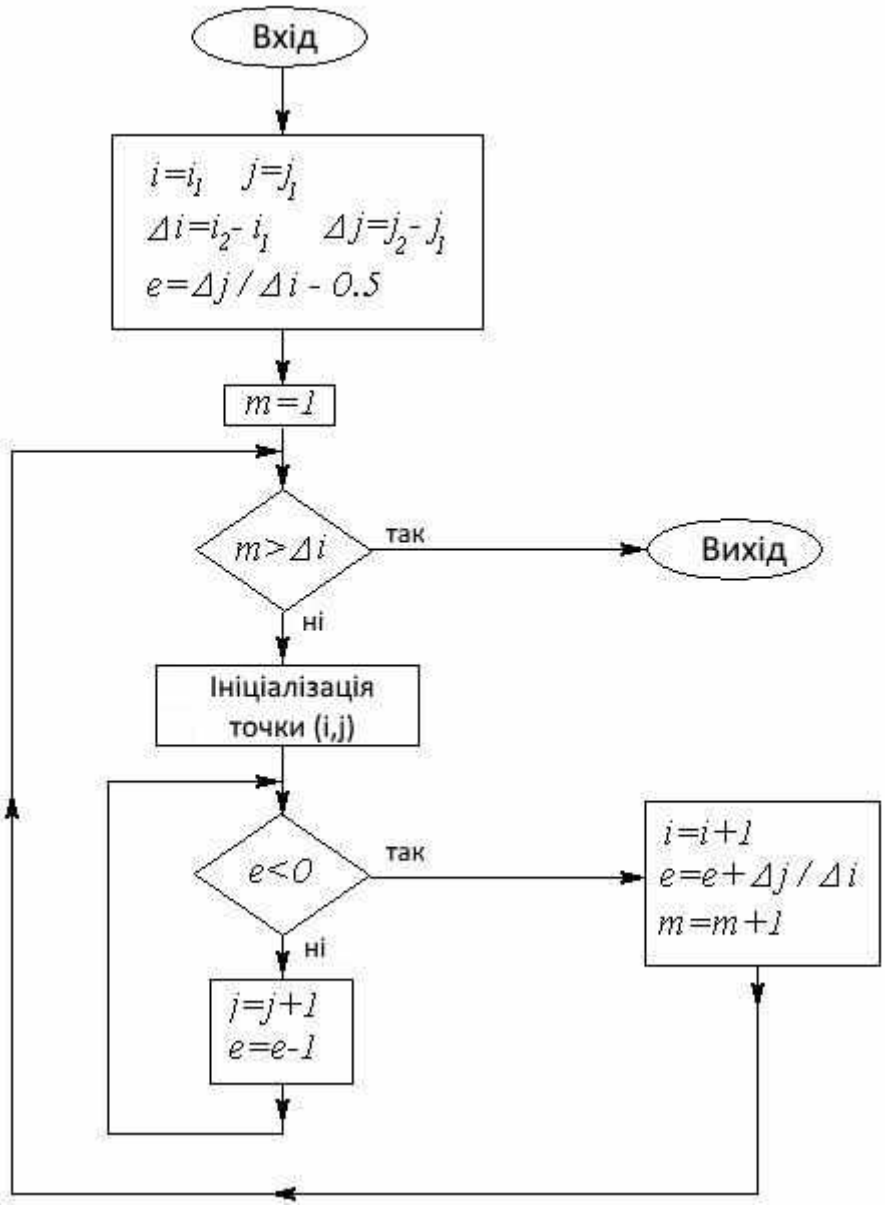


Рис. 10.5. Блок-схема однієї галузі алгоритму Брезенхема

```
i=i1;
j=j1;
di=i 2-i1;
dj=j 2-j1;
s1=sign(i 2-i1);
s2=sign(j 2-j1);
di=abs(di);
dj=abs(dj);
if (dj>di)
{
 swap(di,dj); c=1;
}
else c=0;
e=2* dj-di;        // Ініціалізація зсуву
```

```
// Основний цикл
for (l=0; l<di; l++)
{
  point(i,j);
  while (e>=0)
  {
    if (c==1) i=i+s1;
    else j=j+s2;
    e= e-2*di;
  }
  if (c==1) j=j+s2; else i=i+s1;
  e=e+2*dj;
  }
```

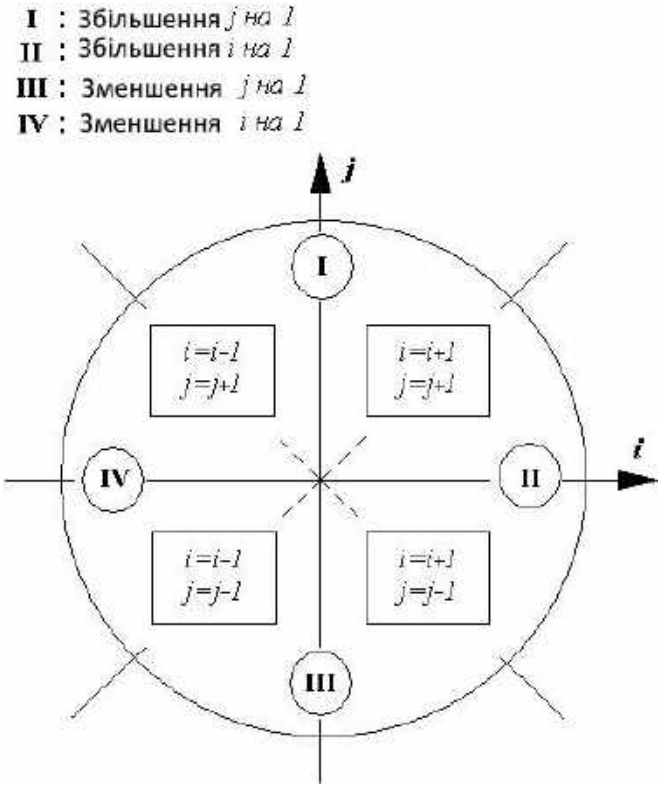


Рис. 10.6. Чотири можливих напрямки відрізка

10.2. Алгоритми Брезенхема растрової дискретизації кола й еліпса

Алгоритм зображення кола трохи складніше, ніж побудова відрізка. Ми розглянемо його для випадку кола радіуса r із центром на початку координат. Перенесення його на випадок довільного центра не становить праці. При побудові растрового розгорнення кола можна скористатися її симетрією щодо координатних осей і прямих $y = \pm x$. Необхідно згенерувати лише одну восьму частину кола, а інші її частини можна одержати шляхом відображень симетрії. За основу можна взяти частина кола від 0 до 45° у напрямку за годинниковою стрілкою з вихідною точкою побудови

$(r,0)$. У цьому випадку координата кола x є монотонно убутною функцією координати y.

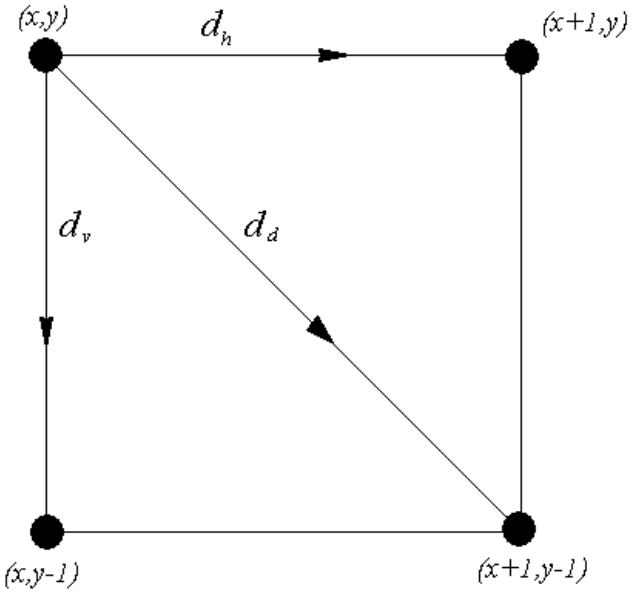


Рис. 10.7. Найближчий піксель при русі по колу

При обраному напрямку руху по кола є тільки три можливості для розташування найближчого пікселя: на одиницю вправо, на одиницю долілиць і по діагоналі долілиць (рис. 10.7). Вибір варіанта можна здійснити, обчисливши відстані до цих крапок і вибравши мінімальне з них:

$$d_h = |s_h|, \quad d_v = |s_v|, \quad d_d = |s_d|,$$

$$s_h = (x+1)^2 + y^2 - r^2, \quad s_v = x^2 + (y-1)^2 - r^2,$$

$$s_d = (x+1)^2 + (y-1)^2 - r^2.$$

Алгоритм можна спростити, перейшовши до аналізу знаків величин s_h, s_v, s_d. При $s_d < 0$діагональна крапка лежить усередині кола, тому найближчими крапками можуть бути тільки діагональна й права. Тепер досить проаналізувати знак вираження $\Delta = d_v - d_d$. Якщо $\Delta \le 0$, вибираємо горизонтальний крок, у противному випадку - діагональний. Якщо ж $s_d > 0$, то визначаємо знак $\Delta^1 = d_d - d_v$, і якщо $\Delta^1 \le 0$, вибираємо діагональний крок, у противному випадку - вертикальний. Потім обчислюється нове значення s_d, причому бажано мінімізувати обчислення не тільки цієї величини, але й величин Δ, Δ^1на кожному кроці алгоритму. Шляхом нескладних перетворень можна одержати для першого кроку алгоритму, що $\Delta = 2(s_d + y) - 1, \quad \Delta^1 = 2(s_d + x) - 1$.

Після переходу в крапку $(x', y'),\ x' = x+1,\ y' = y+1$по діагоналі нове значення s_dобчислюється по формулі $s'_d = s_d + 2x' - 2y' + 2$, при горизонтальному переході $(x' = x+1, y' = y)$ $s'_d = s_d + 2x' + 1$, при вертикальному $(x' = x, y' = y-1)$ $s'_d = s_d - 2y' + 1$.

Таким чином, алгоритм малювання цієї частини кола можна вважати повністю описаним (блок-схема його наведені на рис. 10.8). Всі її частини, що залишилися, будуються паралельно: після одержання чергової (x', y') крапки можна ініціювати ще сім крапок з $(-x', y'),\ (-x', -y'),\ (x', -y'),\ (y', x'),\ (y', -x'),\ (-y', -x'),\ (-y', x')$ координатами .

Для побудови растрового розгорнення еліпса з осями, паралельними осям координат, і радіусами a, b скористаємося канонічним рівнянням

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1,$$

яке перепишемо у вигляді $f(x, y) \equiv b^2x^2 + a^2y^2 - a^2b^2 = 0.$

На відміну від кола, для якої було досить побудувати одну восьму її частину, а потім скористатися властивостями симетрії, еліпс має тільки дві осі симетрії, тому прийде будувати одну чверть всієї фігури. За основу візьмемо дугу, що лежить між крапками $(0, b)$ й $(a, 0)$ у першому квадранті координатної площини.

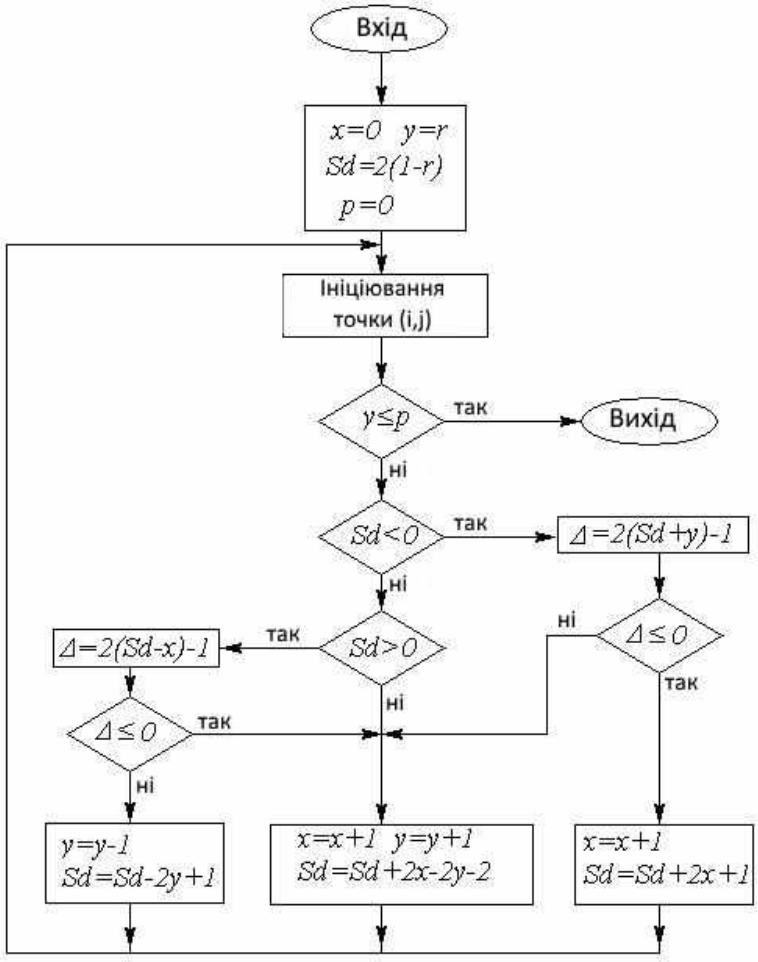


Рис. 10.8. Блок-схема побудови восьмої частини кола

У кожній крапці (x, y) еліпса існує вектор нормалі, що задається градієнтом функції f. Дугу розіб'ємо на дві частини: перша – з кутом між

нормаллю й горизонтальною віссю більше 45° (тангенс більше 1) і друга – з кутом, меншим 45° (рис. 10.9). Рух уздовж дуги будемо здійснювати в напрямку за годинниковою стрілкою, починаючи із крапки $(0, b)$. Уздовж всієї дуги координата є монотонно убутною функцією від x, але в першій частині вона убуває повільніше, ніж росте аргумент, а в другий – швидше. Тому при побудові растрового образа в першій частині будемо збільшувати x на одиницю й шукати відповідне значення y, а в другий – спочатку зменшувати значення y на одиницю й визначати відповідне значення x.

Напрямок нормалі відповідає вектору

$$grad(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (2b^2x, 2a^2y).$$

Звідси знаходимо тангенс кута нахилу вектора нормалі: $t = a^2y/b^2x$. Дорівнюючи його одиниці, одержуємо, що координати крапки розподілу дуги на вищевказані частини задовольняють рівності $b^2x = a^2y$. Тому критерієм того, що ми переходимо до другої області в цілочислових координатах, буде співвідношення $a^2(y - 1/2) \le b^2(x + 1)$, або, переходячи до цілочислових операцій, $a^2(2y - 1) \le 2b^2(x + 1)$.

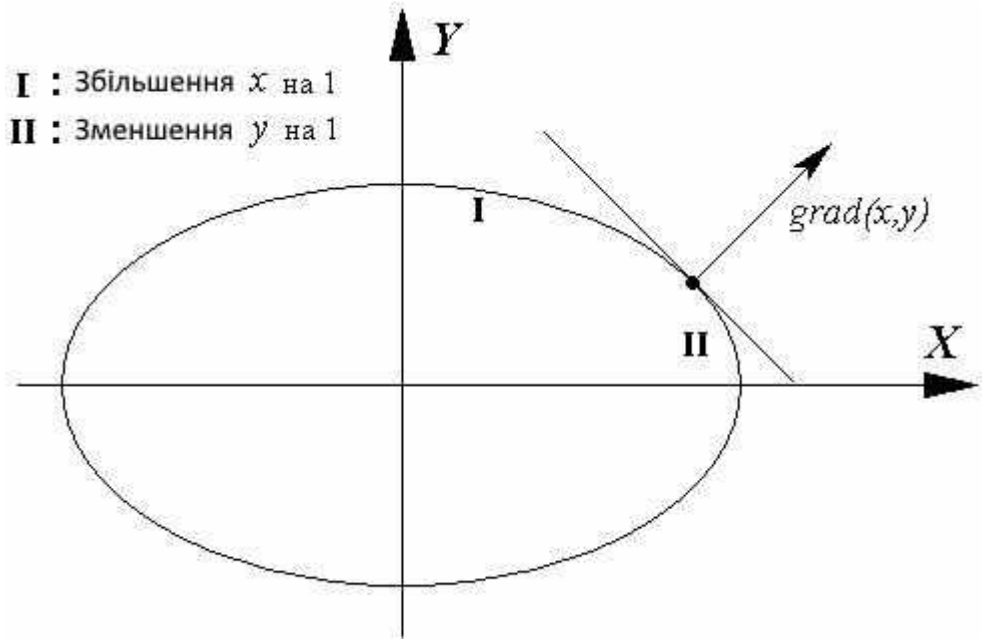


Рис. 10.9. Дві області на ділянці еліпса

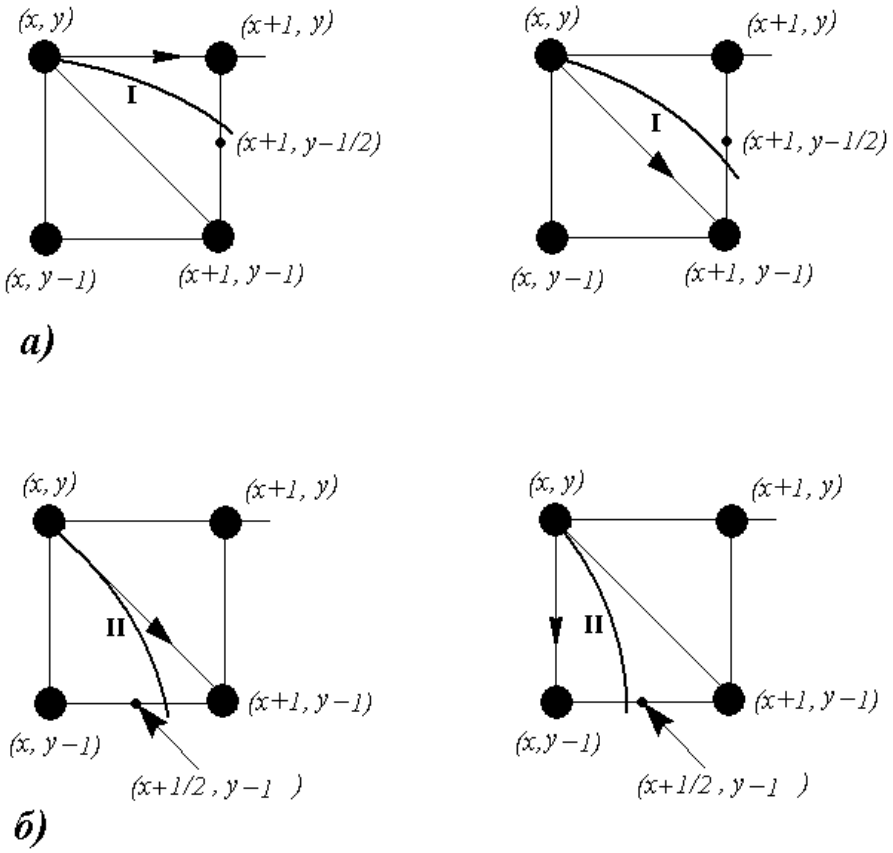


Рис. 10.10. Схема переходу в першій і другій областях дуги еліпса

При переміщенні уздовж першої ділянки дуги ми з кожної крапки переходимо або по горизонталі, або по діагоналі, і критерій такого переходу нагадує той, котрий використовувався при побудові растрового образа кола. Перебуваючи в крапці (x,y), ми будемо обчислювати значення $\Delta = f(x+1, y-\frac{1}{2})$. Якщо це значення менше нуля, то додаткова крапка $(x+1, y-\frac{1}{2})$ лежить усередині еліпса, отже, найближча крапка растра є $(x+1,y)$, у противному випадку це крапка $(x+1, y+1)$ (рис. 10.10а).

На другій ділянці дуги можливі перехід або по діагоналі, або по вертикалі, тому тут спочатку значення координати y зменшується на одиницю, потім обчислюється $\Delta = f(x+\frac{1}{2}, y-1)$ й напрямок переходу вибирається аналогічно попередньому випадку (рис. 10.10б).

Залишається оптимізувати обчислення параметра Δ, помноживши його на 4 і представивши у вигляді функції координат крапки. Тоді для першої половини дуги маємо

$$\widetilde{\Delta}(x,y) \equiv 4\Delta(x,y) = 4b^2(x+1)^2 + a^2(2y-1) - 4a^2b^2,$$

$$\widetilde{\Delta}(x+1,y) = \widetilde{\Delta}(x,y) + 4b^2(2x+3),$$

$$\widetilde{\Delta}(x+1,y-1) = \widetilde{\Delta}(x,y) + 4b^2(2x+3) - 8a^2(y-1).$$

Для другої половини дуги одержимо

$$\widetilde{\Delta}(x,y) \equiv 4\Delta(x,y) = b^2(2x+1)^2 + 4a^2(y-1) - 4a^2b^2,$$
$$\widetilde{\Delta}(x+1,y) = \widetilde{\Delta}(x,y) + 8b^2(x+1),$$
$$\widetilde{\Delta}(x+1,y-1) = \widetilde{\Delta}(x,y) + 8b^2(x+1) - 4a^2(2y+3).$$

Всі дуги, що залишилися, еліпса будуються паралельно: після одержання чергової (x', y')крапки , можна ініціювати ще три крапки з $(-x', y'),\ (-x', -y'),\ (x', -y')$координатами . Блок-схему не приводимо через прозорість алгоритму.

10.3. Алгоритми заповнення областей

Для заповнення областей, обмежених замкнутою лінією, застосовуються два основних підходи: затравочне заповнення й растрове розгорнення.

Методи першого типу виходять із того, що задано деяку крапку (запал) усередині контуру й заданий критерій приналежності крапки границі області (наприклад, заданий колір границі). В алгоритмах шукають крапки, сусідні із затравочною і розташовані усередині контуру. Якщо виявлено сусідню крапку, що належить внутрішньої області контуру, то вона стає затравочний і пошук триває рекурсивно.

Методи растрового розгорнення засновані на скануванні рядків растра й визначенні, чи лежить крапка усередині заданого контуру області. Сканування здійснюється найчастіше "зверху долілиць", а алгоритм визначення приналежності крапки заданої області залежить від виду її границі.

Спочатку розглянемо простий алгоритм заповнення із запалом з використанням стека. Під стеком у цьому випадку ми будемо розуміти масив, у який можна послідовно поміщати значення й послідовно витягати, причому витягають елементи не в порядку надходження, а навпаки: за принципом "першим прийшов – останнім пішов" ("first in – last out"). Алгоритм заповнення виглядає в такий спосіб:

Помістити затравочний піксель у стек
Поки стек не порожній:
  Витягти піксель зі стека
  Ініціювати піксель
  Для кожного із чотирьох сусідніх пікселів:
    Перевірити, чи є він граничним і чи був він ініційованим
    Якщо ні, то помістити піксель у стек

Алгоритм можна модифікувати таким чином, що сусідніми будуть уважатися вісім пікселів (додаються елементи, розташовані в діагональному напрямку).

Методи растрового розгорнення розглянемо спочатку в застосуванні до заповнення багатокутників. Найпростіший метод побудови полягає в тому, щоб для кожного пікселя растра перевірити його приналежність внутрішності багатокутника. Але такий перебір занадто неекономічний, оскільки фігура може займати лише незначну частину екрана, а геометричний пошук – завдання трудомістка, сполучена з довгими обчисленнями. Алгоритм стане більше

ефективним, якщо попередньо виявити мінімальний прямокутник, у який занурений контур багатокутника, але й цього може виявитися недостатньо.

У випадку, коли багатокутник, що обмежує область, заданий списком вершин і ребер (ребро визначається як пара вершин), можна запропонувати ще більш ощадливий метод. Для кожної скануючої рядки визначаються крапки перетинання з ребрами багатогранника, які потім упорядковуються по координаті x. Визначення того, який інтервал між парами перетинань є внутрішній для багатогранника, а який ні, є досить простим логічним завданням. При цьому якщо скануючий рядок проходить через вершину багатогранника, те це перетинання повинне бути враховане двічі у випадку, коли вершина є крапкою локального мінімуму або максимуму. Для пошуку перетинань скануючого рядка з ребрами можна використовувати алгоритм Брезенхема побудови растрового образа відрізка.

На закінчення як приклад приведемо алгоритм зафарбування внутрішньої області трикутника, заснований на складанні повного впорядкованого списку всіх відрізків, що становлять цей трикутник. Для запису горизонтальних координат кінців цих відрізків будемо використовувати два масиви X_{min} й X_{max} розмірністю, рівної числу пікселів растра по вертикалі (рис. 10.11).

Побудова починається з ініціалізації масивів X_{min} і X_{max}: масив X_{max} заповнюється нулями, а масив X_{min} – числом N, рівним числу пікселів растра по горизонталі. Потім визначаємо значення j_{min}, J_{max}, що обмежують трикутник у вертикальному напрямку. Тепер, використовуючи модифікований алгоритм Брезенхема, занесемо границі відрізків у масиви X_{min} й X_{max}. Для цього щораз при переході до чергового пікселю при формуванні відрізка замість його ініціалізації будемо порівнювати його координату i із умістом j-й осередку масивів. Якщо $X_{min}[j] > i$, то записуємо координату i в масив X_{min}. Аналогічно за умови $X_{max}[j] < i$ координату i записуємо в масив X_{max}.

Якщо тепер послідовно застосувати алгоритм Брезенхема до всім трьох сторонам трикутника, то ми одержимо потрібним образом заповнені масиви границь. Залишається тільки проініціювати пікселі усередині відрізків $\{(X_{min[j],j}),\ (X_{max}[j], j)\}$.

Цей алгоритм можна легко поширити на випадок довільного опуклого багатокутника.

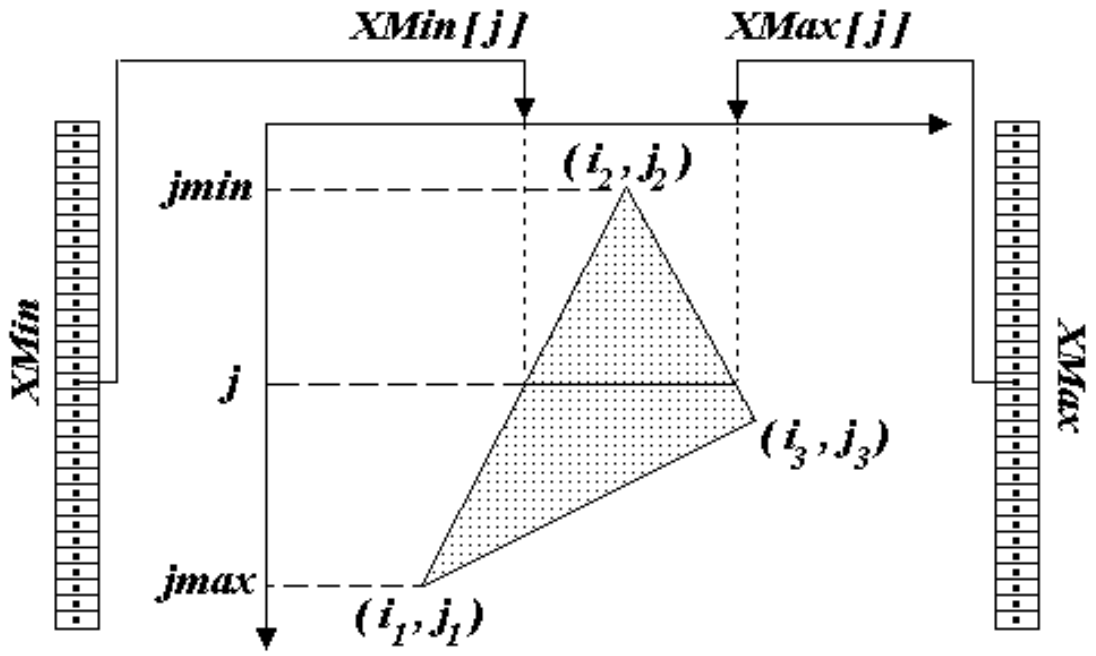


Рис. 10.11. Схема побудови растрового розгорнення трикутника

10.4. Питання й вправи

1. Що таке розкладання в растр?
2. Яка математична основа растрового розкладання в алгоритмі Брезенхема?
3. За яким критерієм ініцілізується піксель у цьому алгоритмі?
4. Чим відрізняються галузі алгоритму при кутах нахилу <45° і >45°?
5. Яку частину кола досить побудувати, щоб потім шляхом відбитків одержати окружність цілком?
6. Яку частину еліпса досить побудувати, щоб потім шляхом відбитків одержати еліпс цілком?
7. Назвіть два типи алгоритмів заповнення областей.
8. Яка структура даних використовується в алгоритмах із запалом?
9. Які дані використовуються при побудові растрового розгорнення трикутника?

Тема 11. ЗАФАРБОВУВАННЯ. РЕНДЕРИНГ ПОЛІГОНАЛЬНИХ МОДЕЛЕЙ

Моделі висвітлення. Зафарбування граней: плоске зафарбовування, метод Гуро, метод Фонга. Усунення ступінчастості (антиелайзінг).

Візуальне сприйняття об'єктів навколишньої дійсності являє собою складний процес, що має як фізичні, так і психологічні аспекти. У другому розділі ми вже обговорювали деякі особливості сприйняття світла й кольору оком людини. До того, що вже було сказано про спектральну чутливість ока, треба додати ще кілька моментів.

Рис. 11.1. Ефекти сприйняття зображення

Очей адаптується до середньої яскравості розглянутої сцени, тому при зміні тла змінюється сприйняття сцени. Наприклад, однорідно пофарбована область на більше темному тлі буде здаватися більше яркою, чим на світлому. Крім того, вона буде сприйматися як більше велика (рис. 11.1).

Ще одна особливість сприйняття полягає в тім, що границя рівномірно освітленої області здається більше яскравої в порівнянні із внутрішніми частинами. Це явище було виявлено Ернстом Махом, тому воно одержало назву **ефекту смуг Маху**. Такі особливості необхідно враховувати, якщо ми прагнемо до створення реалістичних зображень сцен.

При формуванні зображення сцен, що містять дзеркальні й напівпрозорі поверхні, варто використовувати закони геометричної оптики, що переломлюють властивості матеріалів, ефекти змішання кольорів і т.д.

11.1. Проста модель висвітлення

Об'єкти навколишнього простору стають видимими для ока завдяки світловій енергії, що може випромінюватися поверхнею предмета, відбиватися або проходити крізь неї. У свою чергу, відбиття світла від поверхні залежить від фізичних властивостей матеріалу, з якого вона виготовлена, а також від характеру й розташування джерела світла. Яскравість (або інтенсивність) висвітлення залежить від енергії світлового потоку, що обумовлює, по-перше, потужністю джерела світла, а по-друге, що відбивають і пропускають властивостями об'єкта.

Спочатку ми розглянемо модель висвітлення, що враховує тільки відбиття. Властивості відбитого світла залежать головним чином від напрямку променів і характеристик поверхні, що відбиває.

Відбиття може бути двох видів: **дифузійне** й **дзеркальне**. Перше з них виникає в ситуації, коли світло як би проникає під поверхню об'єкта, поглинається, а потім рівномірно випромінюється у всіх напрямках. Поверхня в цьому випадку розглядається як ідеальний розсіювач. При цьому виникає ефект матового світла, а видима освітленість тої або іншої ділянки поверхні не залежить від положення спостерігача. Дзеркальне відбиття, навпаки, походить від зовнішньої поверхні, інтенсивність його неоднорідна, тому видимий максимум освітленості залежить від положення ока спостерігача.

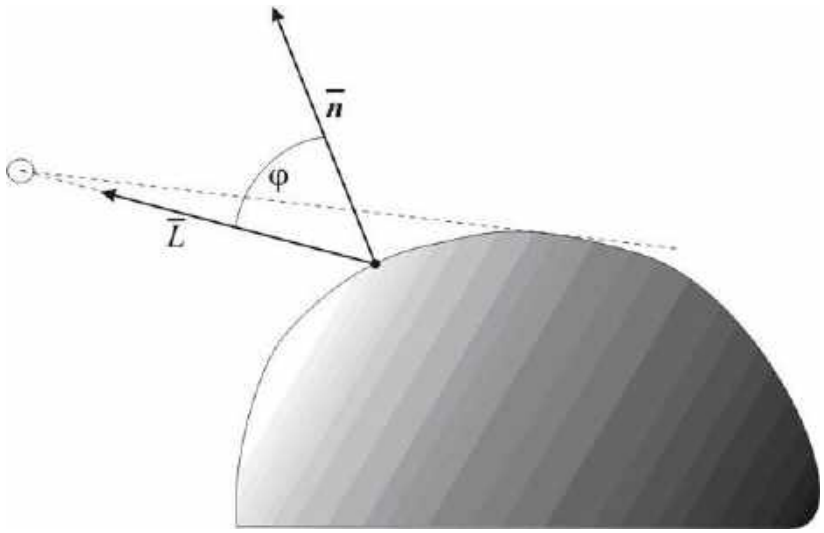


Рис. 11.2. Висвітлення крапковим джерелом

Світло крапкового джерела відбивається від поверхні розсіювача за законом Ламберта: інтенсивність відбиття пропорційна косинусу кута між зовнішньою нормаллю до поверхні й напрямком до джерела світла (рис. 11.2). Якщо I_S – інтенсивність джерела світла, φ – кут між вектором зовнішньої нормалі до поверхні й напрямком до джерела світла, то інтенсивність відбитого світла визначається формулою

$$I = \begin{cases} I_S \cos\varphi \text{ при } 0 \le \varphi \le \pi/2 \\ \\ 0 \text{ в противном случае} \end{cases} \qquad (11.1)$$

При такому розрахунку інтенсивності вийде дуже контрастна картина, тому що ділянки поверхні, на які промені від джерела не попадають прямо, залишаться абсолютно чорними. Для підвищення реалістичності необхідно враховувати розсіювання світла в навколишньому просторі. Тому вводиться **фонова освітленість**, що залежить від інтенсивності розсіяного світла I_F, і інтенсивність відбитого світла визначається вираженням

$$I = \begin{cases} I_F k_F + k_s I_S \cos\varphi \text{ при } 0 \le \varphi \le \pi/2 \\ \\ I_F k_F \text{ в противном случае} \end{cases} \tag{11.2}$$

де k_S – коефіцієнт дифузійного відбиття розсіяного світла, k_F – коефіцієнт дифузійного відбиття падаючого світла, $0 \le k_S \le 1,\ 0 \le k_F \le 1$.

В описаній моделі поки ніяк не враховувалася далекість джерела світла від поверхні, тому по освітленості двох об'єктів не можна судити про їхнє взаємне розташування в просторі. Якщо ми хочемо одержати перспективне зображення, то необхідно включити загасання інтенсивності з відстанню. Звичайно інтенсивність світла обернено пропорційна квадрату відстані від джерела. Як відстань до джерела у випадку перспективного перетворення можна взяти відстань до центра проекції, і якщо він досить вилучений, то зображення буде досить адекватним. Але якщо цей центр розташований близько до об'єкта, то квадрат відстані міняється дуже швидко, і в цьому випадку краще використовувати лінійне загасання. У цьому випадку інтенсивність відбитого світла від безпосередньо освітлених ділянок поверхні буде задаватися формулою

$$I = I_F k_F + \frac{k_S I_S \cos\varphi}{d + C} \tag{11.3}$$

де d – відстань до центра проекції, а C – довільна постійна. Якщо центр проекції перебуває на нескінченності, тобто при паралельному проектуванні, то в якості d можна взяти відстань до об'єкта, найбільш близького до спостерігача.

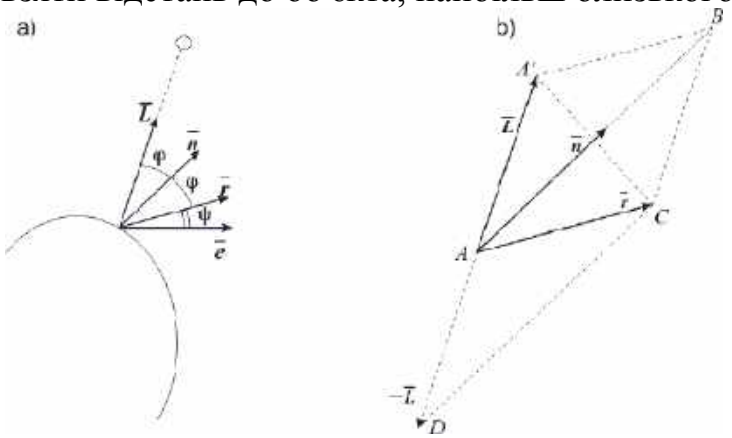


Рис. 11.3. Дзеркальне відбиття

На відміну від дифузійного, дзеркальне відбиття є спрямованим. Ідеальне дзеркало відбиває промені за принципом "відбитий і падаючий промені лежать в одній площині, причому кут падіння дорівнює куту відбиття" (мається на увазі кут між напрямком променю й нормаллю до поверхні). Якщо поверхня не ідеально дзеркальна, то промені відбиваються в різних напрямках, але з різною інтенсивністю, а функція зміни інтенсивності має чітко виражений максимум. Оскільки фізичні властивості дзеркального відбиття досить складні, то в комп'ютерній графіці використовується емпірична модель Фонга. Суть її полягає в тім, що для ока спостерігача інтенсивність дзеркально відбитого

променю залежить від кута між ідеально відбитим променем і напрямком до спостерігача (рис. 11.3а). Крім того, оскільки дзеркальне відбиття залежить ще й від довжини хвилі, це також будемо враховувати у формулі для обчислення інтенсивності. Модель Фонга описується співвідношенням

$$I_Z = \omega(\varphi, \lambda) \cdot I_S \cos^n \psi \qquad (11.4)$$

де $\omega(\varphi, \lambda)$ – функція відбиття, λ – довжина хвилі. Ступінь, у яку зводиться косинус кута, впливає на розміри світлового відблиску, спостережуваного глядачем. Графіки цієї функції наведені на рис. 11.4, і вони саме є характерними кривими поводження функції зміни інтенсивності залежно від властивостей поверхні.

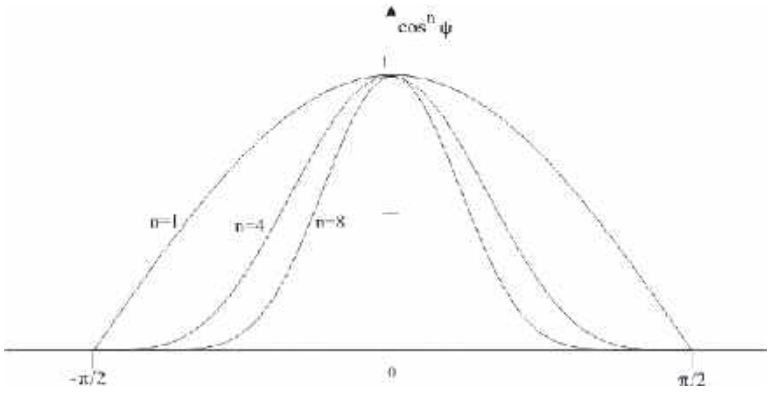


Рис. 11.4. Дзеркальне відбиття

Тепер модель освітленості, що враховує дзеркальне й дифузійне відбиття, можна описати формулою

$$I = I_F k_F + \frac{I_S(k_s \cos\varphi + \omega(\varphi, \lambda) \cdot \cos^n \psi)}{d + C}. \qquad (11.5)$$

Використовуючи одиничні вектори $\vec{L}$ (напрямок до джерела) і $\vec{n}$ (зовнішня нормаль), косинус кута φ можна обчислити через скалярний добуток: $\cos\varphi = (\vec{L} \cdot \vec{n})$. Для розрахунку інтенсивності дзеркального відбиття спочатку треба визначити відбитий вектор $\vec{r}$. З рис. 11.3b видно, що $\vec{r} = |AB| \cdot \vec{n} - \vec{L}$. З іншої сторони AB є діагоналлю ромба $AA'BC$, тому $|AB| = 2 \cdot (\vec{L} \cdot \vec{n})$. З огляду на всі ці співвідношення, одержуємо формулу

$$I = I_F k_F + \frac{I_S\{k_s(\vec{L} \cdot \vec{n}) + \omega(\varphi, \lambda) \cdot [2 \cdot (\vec{L} \cdot \vec{n}) \cdot (\vec{e} \cdot \vec{n}) - (\vec{e} \cdot \vec{L})]^n\}}{d + C}. \qquad (11.6)$$

В алгоритмах зафарбовування з використанням колірних моделей інтенсивність розраховується для кожного з базових кольорів, оскільки зміна інтенсивності при дзеркальному відбитті залежить від довжини хвилі.

11.2. Зафарбування граней

11.2.1. Плоске зафарбовування

Якщо припустити, що джерело світла перебуває на нескінченності, то промені світла, що падають на поверхню, паралельні між собою. Якщо до цього додати умова, що спостерігач перебуває в нескінченно вилученій крапці, то ефектом ослаблення світла зі збільшенням відстані від джерела також можна зневажити. Крім того, таке положення спостерігача означає ще й те, що вектори, спрямовані від різних крапок поверхні до спостерігача, також будуть паралельні. При виконанні всіх цих умов, як треба з формули (11.6), плоска грань у всіх крапках має однакову інтенсивність висвітлення, тому вона зафарбовується одним кольором. Таке зафарбовування називається **плоским.**

Якщо ми апроксимуємо деяку гладку поверхню багатогранником, то при плоскому зафарбовуванні неминуче виявляться ребра, оскільки сусідні грані з різними напрямками нормалей мають різний колір. Ефект смуг Маху додатково підсилює цей недолік. Для його усунення при використанні цього способу зафарбовування можна лише збільшити число граней багатогранника, що приводить до збільшення обчислювальної складності алгоритму.

11.2.2. Зафарбування методом Гуро

Один зі способів усунення дискретності інтенсивності зафарбовування був запропонований Гуро. Його метод полягає в тім, що використовуються не нормалі до плоских граней, а нормалі до апроксимованої поверхні, побудовані у вершинах багатогранника. Після цього обчислюються інтенсивності у вершинах, а потім у всіх внутрішніх крапках багатокутника виконується білінійна інтерполяція інтенсивності.

Метод сполучається з алгоритмом порядкового сканування. Після того як грань відображена на площину зображення, для кожної скануючої рядки визначаються її крапки перетинання з ребрами. У цих крапках інтенсивність обчислюється за допомогою лінійної інтерполяції інтенсивностей у вершинах ребра. Потім для всіх внутрішніх крапок багатокутника, що лежать на скануючої рядку, також обчислюється інтенсивність методом лінійної інтерполяції двох отриманих значень. На рис. 11.5 показаний плоский багатокутник з обчисленими значеннями інтенсивностей у вершинах.

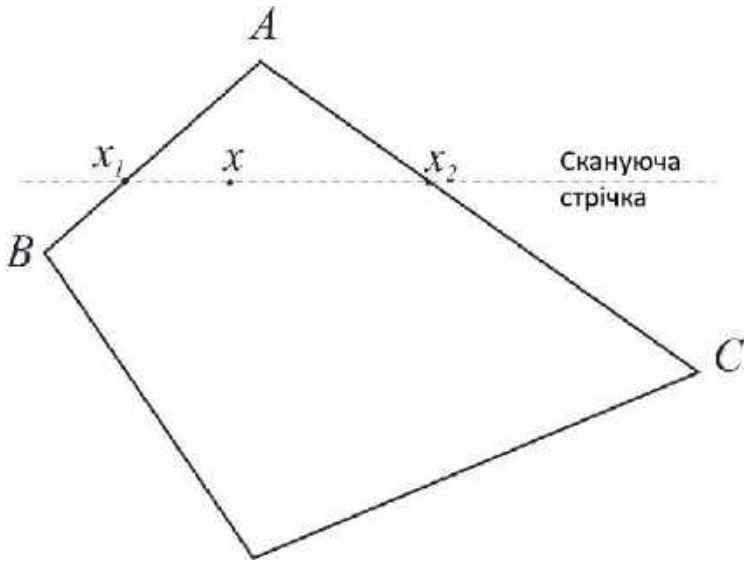


Рис. 11.5. Інтерполяція інтенсивності

Нехай I_A, I_B, I_C – інтенсивності у вершинах A, B, C, x_A, x_B, x_C – горизонтальні координати цих крапок. Тоді в крапках перетинання скануючого рядка з ребрами багатокутника інтенсивності можна обчислити по формулах інтерполяції:

$$\begin{aligned} I_1 &= t_1 I_A + (1 - t_1) I_B, \quad t_1 = \frac{x_1 - x_B}{x_A - x_B}, \\ I_2 &= t_2 I_A + (1 - t_2) I_C, \quad t_2 = \frac{x_2 - x_C}{x_A - x_C}. \end{aligned} \tag{11.7}$$

Після цього інтенсивність у крапці x одержуємо шляхом інтерполяції значень на кінцях відрізка:

$$I = t I_1 + (1 - t) I_2, \quad t = \frac{x_2 - x}{x_2 - x_1} \tag{11.8}$$

До недоліків методу Гуро варто віднести те, що він добре працює тільки з дифузійною моделлю відбиття. Форма відблисків на поверхні і їхнє розташування не можуть бути адекватно відтворені при інтерполяції на багатокутниках. Крім того, є проблема побудови нормалей до поверхні. В алгоритмі Гуро нормаль у вершині багатогранника обчислюється шляхом усереднення нормалей до граней, пов'язаним із цією вершиною. Така побудова сильно залежить від характеру розбивки.

11.2.3. Зафарбування методом Фонга

Фонг запропонував замість інтерполяції інтенсивностей зробити інтерполяцію вектора нормалі до поверхні на скануючому рядку. Цей метод вимагає більших обчислювальних витрат, оскільки формули інтерполяції (11.6)-(11.7) застосовуються до трьох компонентів вектора нормалі, але зате дає кращу апроксимацію кривизни поверхні. Тому дзеркальні властивості поверхні відтворюються набагато краще.

Нормалі до поверхні у вершинах багатогранника обчислюються так само, як і в методі Гуро. А потім виконується білінійна інтерполяція в сполученні з

порядковим скануванням. Після побудови вектора нормалі в черговій крапці обчислюється інтенсивність.

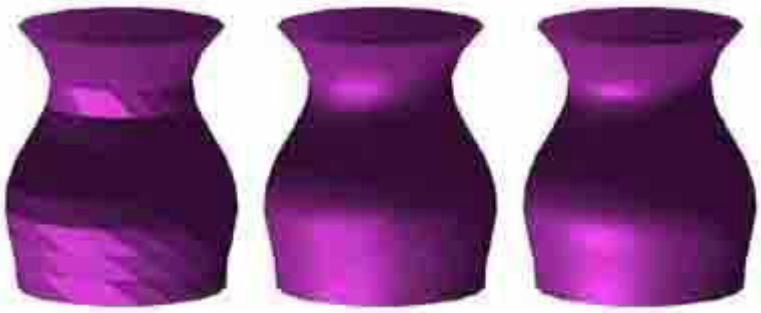


Рис. 11.6. Три способи зафарбовування

Цей метод дозволяє усунути ряд недоліків методу Гуро, але не все. Зокрема, ефект смуг Маху в окремих випадках у методі Фонга буває навіть сильніше, хоча в переважній більшості випадків апроксимація Фонга дає кращі результати. На рис. 11.6 наведені результати зафарбовування поверхні обертання, апроксимованої багатогранником, що складений із трикутних граней: a) – плоске зафарбовування, b) – зафарбовування по методу Гуро, c) – зафарбовування по методу Фонга. Перший з варіантів дає зображення ребристої поверхні з дуже контрастними переходами від однієї грані до іншої. Друга модель дає більше гладке зображення, але в районі відблисків чітко спостерігаються лінії ребер, хоча й згладжені. Третій варіант вийшов найбільш гладкі, дзеркальні відблиски мають досить реалістичну форму.

11.3. Більше складні моделі висвітлення

Коли ми розглядали алгоритми видалення невидимих ліній, передбачалося, що сцена включає тільки непрозорі об'єкти. У простої моделі висвітлення теж мова йшла про непрозорі поверхні. Тепер можна ускладнити завдання, включивши в модель не тільки відбиття світла, але й переломлення.

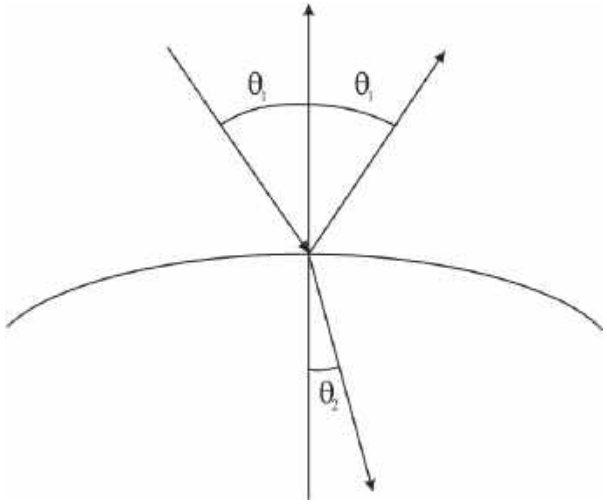


Рис. 11.7. Переломлений і відбитий промені

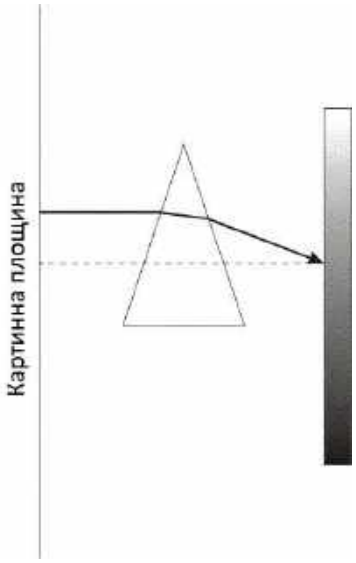


Рис. 11.8. Переломлення в призмі

При переході променю з одного середовища в іншу його напрямок змінюється відповідно до закону Синеліуса: переломлений промінь лежить у площині, утвореною нормаллю до площини й падаючим променем, а кути, утворені променями з нормаллю, зв'язані формулою $\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$, де η_1, η_2 – показники переломлення двох середовищ (рис. 11.7). Пропущення світла також може бути дифузійним (якщо частина енергії світла розсіюється середовищем) або спрямованим. У першому випадку ми маємо справу з напівпрозорими тілами, які змінюють фарбування видимих крізь них об'єктів. У другому випадку тіло є прозорим, і воно візуально виявляється тільки завдяки перекручуванням об'єктів за рахунок переломлення променів.

При наявності в просторовій сцені прозорих або напівпрозорих об'єктів треба враховувати, що зображення інших об'єктів буде відрізнятися від звичайної проекції на картинну площину (рис. 11.8). Ці ефекти добре знайомі всім, хто зіштовхувався з різними лінзами. Для побудови зображення таких сцен доцільно використовувати алгоритми зі зворотним трасуванням променів.

Для зображення напівпрозорих поверхонь без обліку переломлення можна ввести так званий коефіцієнт прозорості κ, що дозволяє змішувати інтенсивності для видимої поверхні й тієї, що розташовано за нею:

$$I = \kappa I_1 + (1-\kappa) I_2, \quad 0 \le \kappa \le 1$$

При $\kappa = 1$ поверхня непрозора, при $\kappa = 0$ – повністю прозора. Для напівпрозорих тіл необхідно враховувати їхню об'ємну структуру.

Методи побудови зображень сцен із прозорими й напівпрозорими об'єктами будуть більш докладно розглянуті в наступній лекції.

11.4. Усунення ступінчастості (антиелайзінг)

При побудові растрового образа ліній (див. лекцію 8) ми зіштовхуємося з ефектом ступінчастості, пов'язаним з дискретизацією безперервного об'єкта. Перекручування ідеального образа відбувається тому, що із усього безлічі

крапок ми вибираємо тільки ті, які виявляються ближче всього до центра елемента растра, і ініціюємо цей елемент.

Рис. 11.9. Розподіл ваг при збільшенні розширення в 4 рази

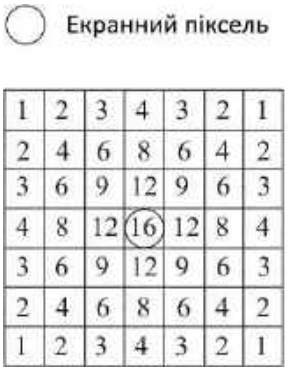


Рис. 11.10. Розподіл ваг при збільшенні розширення в 16 разів

Для запобігання сильних перекручувань у цьому випадку можна, по-перше, підвищувати дозвіл растра, що дозволяє відображати усе більше дрібні деталі об'єктів. Але в цього підходу є свої чисто фізичні обмеження. Другий підхід полягає в тім, що растр розраховується з більше високим дозволом, а зображується з більше низьким – шляхом усереднення атрибутів пікселів першого більше детального растра з певними вагами. Якщо ваги однакові, то ми одержуємо рівномірне усереднення, як показано на рис. 11.9. Кращих результатів можна досягти, якщо використовувати різні ваги в пікселів першого растра. На рис. 11.10 показаний розподіл ваг при деталізації пікселя екранного растра.

Інший метод усунення ступінчастості полягає в тому, щоб розглядати піксель не як крапку, а як деяку кінцеву область. В алгоритмах побудови растрового розгорнення піксель уважається приналежної області зафарбовування, якщо його центр перебував усередині ідеального образа області. Якщо малюнок чорно-білий, то усунути ефект ступінчастості растра практично неможливо. Але при наявності відтінків півтонів можна задати інтенсивність кольору пікселя залежно від площі його перетинання з областю.

Розглянемо застосування цього методу на прикладі розфарбування багатокутника. Ребро багатокутника будується з використанням алгоритму Брезенхема, описаного в лекції 8. Тут у цей алгоритм будуть внесені зміни, що

включають параметр максимального числа рівнів інтенсивностей. Визначаючи приналежність пікселя багатокутнику, ми будемо використовувати як помилка е частку площі, що належить ідеальній фігурі (рис. 11.11).

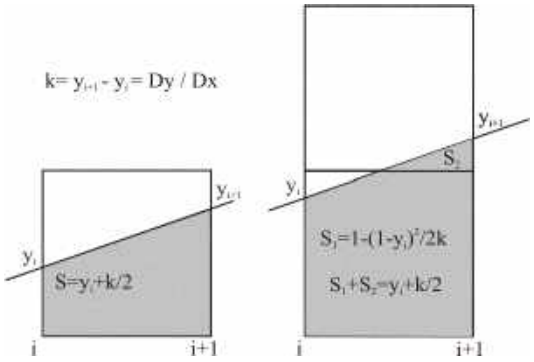


Рис. 11.11. площа, Що Відтинається відрізком, пікселя

Розглянемо знову випадок, коли відрізок спрямований у позитивний квадрант координатної площини під кутом, меншим $\pi/4$. Ідеальний відрізок при заданому значенні цілочислової координати i може перетинати один або два пікселя. У попередній версії алгоритму вибирався піксель, центр якого розташовувався ближче до відрізка. Тепер інтенсивність для обох пікселів буде задаватися залежно від **ступеня близькості** кожного з них. Ініціалізація пікселя буде використовувати інтенсивність як параметр. Передбачається, що відрізок починається з кута першого пікселя, виходячи із чого й задається початкова інтенсивність. Блок-схема алгоритму наведена на рис. 11.12.

Усунення ефекту ступінчастості з математичної точки зору є завданням **згладжування**. Наведений тут алгоритм, що використовує площі перетинання растра й ідеального образа, можна описати за допомогою операції згортки функції. Спочатку дамо необхідні визначення. **згорткою** функції $f(x)$ називається інтеграл виду

$$C(\xi) = \int_{-\infty}^{+\infty} K(\xi - t) f(t) dt. \qquad (11.9)$$

Функція $K(x)$ називається **ядром згортки**. Як ядро згортки звичайно використовується або функція з кінцевим носієм (тобто відмінна від нуля лише на деякому кінцевому інтервалі), або швидко убутна на нескінченності функція (це може бути необхідною умовою існування інтеграла).

Розглянемо як згортається функція і ядра наступні функції:

$$f(x) = \begin{cases} x \text{ при } 0 \le x \le 1 \\ 0 \text{ при } x < 0 \text{ и } x > 1 \end{cases}, \qquad K(x) = \begin{cases} 1 \text{ при } 0 \le x \le 1 \\ 0 \text{ при } x < 0 \text{ и } x > 1 \end{cases}.$$

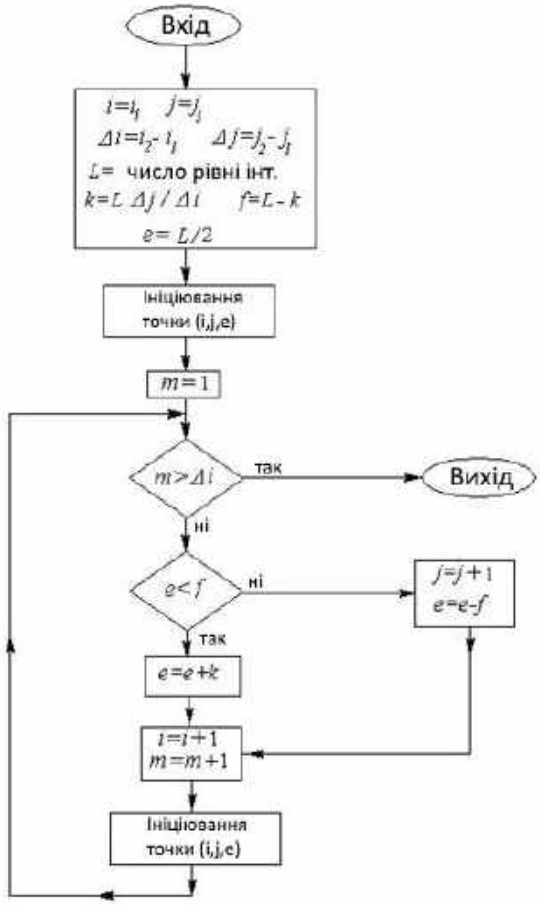


Рис. 11.12. Блок-схема модифікованого алгоритму Брезенхема

Тоді, у силу того, що підінтегральне вираження звертається в нуль при $\xi - t < 0$ й при $\xi - t > 0$, одержуємо

$$C(\xi) = \int_{\xi-1}^{\xi} f(t)dt.$$

З огляду на вид функції $f(x)$, одержуємо, що згортка буде відмінна від нуля тільки на інтервалі $0 < \xi < 2$. Значення згортки в деяких крапках наведені в таблиці 11.1.

Таблиця 11.1.

Значення згортки у вузлах

ξ	0	1/2	1	3/2	2
$C(\xi)$	0	1/8	1/2	3/8	0

Очевидно, що наша згортка дає площа перетинання трикутника, утвореного згортається функцією, що, із квадратом, підстава якого є відрізок $[\xi, xi+1]$ на осі OX .

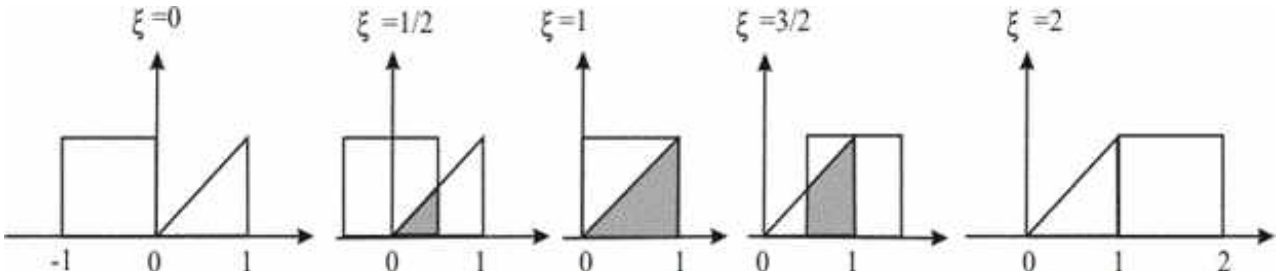


Рис. 11.13. Фігури, що відповідають значенням згортки з таблиці 11.1

На рис. 11.13 наведений вид перетинання для всіх п'яти випадків з таблиці 11.1. Якщо зрівняти ці результати з рис. 11.11, те видно, що значення згортки при $k \le 1$ дають площа тої частини пікселя, що перебуває усередині багатокутника (якщо вважати $y_i = 0$), а при $k > 1$ – суму площ двох пересічних пікселів.

На закінчення проілюструємо результат застосування алгоритму усунення ступінчастості на прикладі зображення, отриманого за допомогою програми CorelDraw. Ця програма являє собою розвитий графічний редактор, що дозволяє будувати об'єкти векторної графіки. На рис. 11.14 показане зображення простих графічних примітивів, попередньо переведене в растрову форму, на якому при великому збільшенні помітне згладжування із застосуванням відтінків сірого кольору.

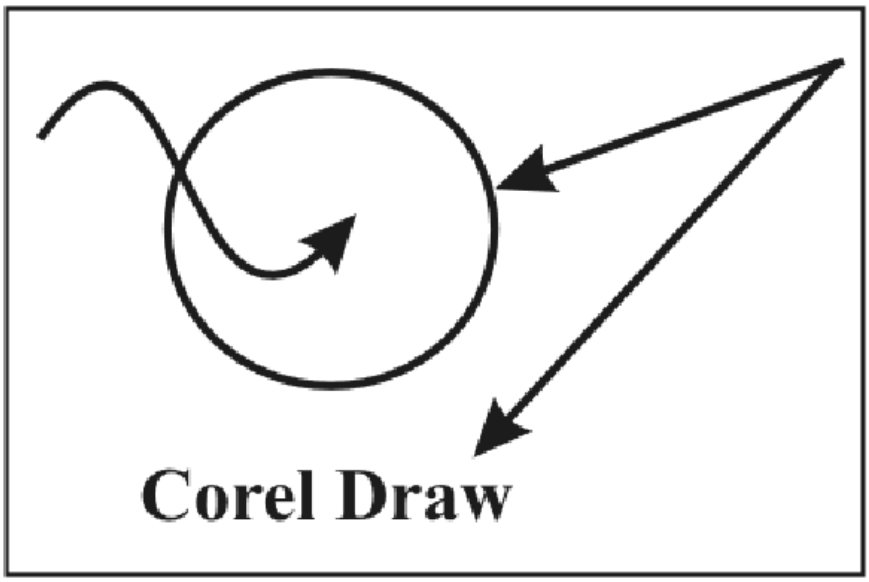


Рис. 11.14. Згладжені зображення

11.5 Питання й вправи

1. Що таке ефект смуг Маху?
2. Чим відрізняється дифузійне відбиття від дзеркального?
3. Від чого залежить інтенсивність висвітлення крапки поверхні при дифузійному відбитті?

4. Від чого залежить інтенсивність висвітлення крапки поверхні при дзеркальному відбитті?
5. Які параметри враховує модель дзеркального відбиття, запропонована Фонгом?
6. Чи міняється інтенсивність висвітлення при плоскому зафарбовуванні грані?
7. Який параметр інтерполюється при зафарбовуванні методом Гуро?
8. Який параметр інтерполюється при зафарбовуванні методом Фонга?
9. У чому складається ідея алгоритмів антиелайзінга, заснованих на рівні деталізації растра?
10. Який параметр використовується в алгоритмі антиелайзінга, що враховує розміри пікселя?

Тема 12. ВІЗУАЛІЗАЦІЯ ПРОСТОРОВИХ РЕАЛІСТИЧНИХ СЦЕН

Світло-Тіньовий аналіз. Метод випромінювання. Глобальна модель висвітлення із трасуванням променів. Алгоритм зворотного трасування.

12.1. Світло-Тіньовий аналіз

Тіні виникають як результат екранування джерела світла предметами, що становлять сцену. При цьому існують повні тіні й півтіні, що пов'язане з фізичною природою світла: повна тінь утвориться в центральній частині затінення, а півтінь – поблизу її границь. У нашім курсі ми будемо стосуватися тільки повної тіні, хоча й існують технології зображення напівтіней, що супроводжуються більшим обсягом обчислень.

Якщо вважати, що спостерігач перебуває в одній крапці із джерелом світла, то тіні в спостережуваній сцені не виникають: затінюється область, що, є невидимою. У всіх інших випадках тіні видні. Джерело світла може перебувати на нескінченності або на кінцевій відстані, причому в другому випадку він може виявитися в поле зору спостерігача.

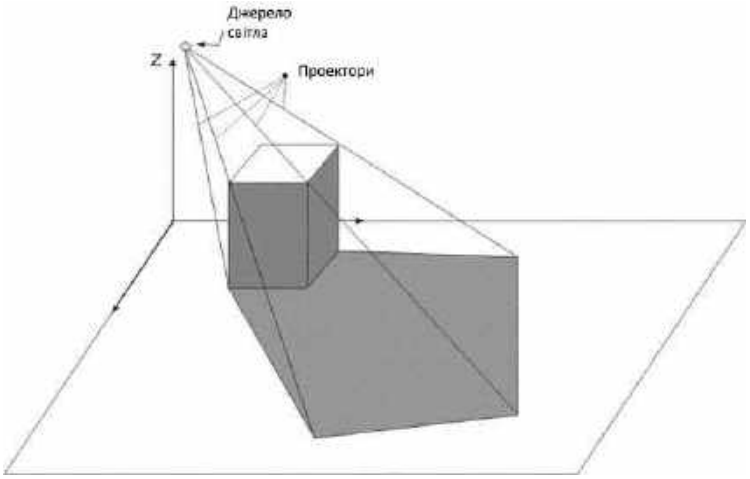


Рис. 12.1. Утворення тіней при кінцевій відстані до джерела світла

Для нескінченно вилученого джерела світла тіні на картинній площині виходять у результаті паралельної проекції об'єктів, а для близького джерела - центральної проекції. Об'єкти і їхні частини стають невидимими, якщо вони попадають в область тіні, тому при побудові зображення завдання про видалення невидимих областей вирішується двічі: щодо спостерігача в процесі проектування й щодо джерела світла. При визначенні розташування тіней будуються проекції невидимих з позиції джерела світла граней (неосвітлених) на картинну площину, у результаті чого виходять тіньові багатокутники. Ці багатокутники будуються для всіх об'єктів сцени й заносяться в список. Відзначимо, що тіньові багатокутники не залежать від положення спостерігача, тому при огляді сцен з різних точок зору вони будуються тільки один раз. Неосвітлені області визначаються тими ж методами, які були описані раніше в лекції 7, а для побудови проекцій використовуються методи, деякі з яких

описані в лекції 8. Зокрема, можна застосовувати матриці проекцій в однорідній системі координат.

Уперше ідея сполученого аналізу видимості й затемнення була запропонована в 1968 р. Аппелем. Як приклад розглянемо один алгоритм на основі порядкового сканування, що складає із двох основних етапів.

І. Аналіз сцени стосовно джерела світла. Для всіх багатокутників, отриманих у результаті проектування сцени, визначаються неосвітлені (затінені) ділянки й тіньові багатокутники (проекційні тіні), причому багатокутники утворять пронумерований список. Для цих багатокутників формується матриця $A = (a_{ij})$, що дозволяє визначити, чи відкидає багатокутник тінь і які з багатокутників він може закривати. Якщо для деяких значень $i, j \quad a_{ij} = 1$, то це означає, що багатокутник з номером i може відкидати тінь на багатокутник з номером j.

Таким чином, на цьому етапі основним є питання про ефективний алгоритм побудови такої матриці. Якщо проекція включає N багатокутників, то необхідно розглянути "взаємини" $N \cdot (N-1)$ пар багатокутників. Скоротити перебір можна за рахунок занурення об'єктів у прямокутні або сферичні оболонки або шляхом використання сортування по глибині.

ІІ. Аналіз сцени стосовно спостерігача. Виконуються два процеси сканування. Перший – для визначення відрізків, видимих з позиції спостерігача (про нього розповідалося раніше в лекції 7). Другий – для визначення перетинань відрізків з тіньовими багатокутниками зі списку. Рекурсивний алгоритм другого сканування складається із чотирьох основних кроків.

Для кожного видимого відрізка

1. Якщо немає жодного тіньового багатокутника, то відрізок зображується з основною інтенсивністю.
2. Якщо багатокутник, що містить видимий відрізок, не перетинається з тіньовими багатокутниками, то відрізок зображується з основною інтенсивністю.
3. Якщо відрізок повністю закривається деякими тіньовими багатокутниками, то інтенсивність його зображення визначається з урахуванням затінення всіма цими багатокутниками.
4. Якщо кілька тіньових багатокутників частково закривають відрізок, то він розбивається на ряд відрізків крапками перетинання з тіньовими багатокутниками.

Цей алгоритм припускає, що затінення не абсолютне, тобто затінені ділянки все-таки є видимими, тільки їхня освітленість падає залежно від кількості й освітленості багатокутників, що затінюють. При повнім затіненні в третьому пункті алгоритму відрізок стає повністю невидимим, а в четвертому подальшому аналізу піддаються тільки незатінені відрізки.

Способи розрахунку інтенсивності при неповнім затіненні можуть бути різні. У цьому випадку всі затінені багатокутники мають свою інтенсивність залежно від обраної моделі освітленості. При цьому можна враховувати відстань затіненої ділянки від поверхні, що відкидає тінь.

Ще один алгоритм, часто застосовуваний при побудові тіней, зветься методу тіньового буфера. Він будується на основі методу Z-Буфера, описаного в лекції 7. Тіньовий буфер – це той же Z-Буфер, тільки з погляду джерела світла. Таким чином, використовуються два буфери: один – для відстані від картинної площини до крапок зображуваної сцени, а іншої – для відстаней від цих же крапок до джерела світла. Алгоритм дозволяє зображувати сцени з повним затіненням і зводиться до двох основних етапів:

1. Сцена розглядається із крапки розташування джерела світла у відповідній системі координат. Підсумком побудови є повністю заповнений тіньовий буфер.
2. Сцена розглядається з погляду спостерігача, застосовується звичайний метод Z-Буфера з невеликим доповненням. Якщо крапка (x, y, z) є видимою в цій системі координат, то обчислюються її координати в системі, пов'язаної із джерелом світла – (x', y', z'), потім перевіряється, чи є крапка видимої із цієї позиції. Для цього значення z' рівняється зі значенням, що втримується в тіньовому буфері для цієї крапки, і у випадку видимості значення інтенсивності заноситься в буфер кадру в крапці (x, y).

Обоє наведених алгоритму працюють у просторі зображення, тобто мають справа із проекціями на площину й деяку додаткову інформацію про крапки сцени, що відповідають цим проекціям. Існують алгоритми, що працюють у тривимірному об'єктному просторі. Зокрема, для побудови тіней використовуються модифікації алгоритму Вейлера-Азертона, описаного в лекції 7. Модифікація полягає в тім, що, як і у випадку тіньового буфера, завдання видалення невидимих граней вирішується спочатку з позиції джерела світла, а потім отримана інформація про об'єкти використовуються при побудові зображення з позиції спостерігача. Загалом кроки алгоритму можна описати так:

1. Визначаються грані, видимі із крапки розташування джерела світла. З метою підвищення ефективності запам'ятовується інформація тільки про видимі грані. Оскільки аналіз виконується в системі координат, пов'язаної із джерелом світла, те отримані видимі багатокутники потім заново приводяться до вихідної системи координат. Багатокутники зв'язуються із гранями, яким вони належать (у результаті затінення одна грань може містити кілька багатокутників).
2. Сцена обробляється з положення спостерігача. При зображенні видимої грані враховуються тільки ті багатокутники, які входять у список, отриманий на першому етапі, тобто грань розглядається як сукупність таких багатокутників.

При наявності декількох джерел світла кількість освітлених ділянок природно збільшується.

12.2. Метод випромінювання

У цій лекції вже говорилося, що освітленість поверхні визначається власним випромінюванням тіла й відбитих променів, що падають від інших тіл

(джерел). Модель випромінювання включає обоє ці факторів і заснована на рівняннях енергетичного балансу. При цьому виконувані розрахунки враховують тільки взаємне розташування елементів сцени й не залежать від положення спостерігача.

Представимо сцену з N елементів (ділянок поверхонь). Освітленість будемо моделювати як кількість енергії, випромінювана поверхнею. Для кожного елемента ця кількість енергії складається із власної енергії (E_k) й відбитої частки енергії, отриманої від інших об'єктів. Передбачається, що для кожної пари елементів з номерами i, j можна визначити, яка частка енергії одного попадає на інший (w_{ij}). Нехай α_i – коефіцієнт відбиття енергії $i-$ м елементом. Тоді повна енергія, випромінювана цим елементом, буде визначатися рівнянням $U_i = E_i + \alpha_i \sum_{j=1}^{N} w_{ij} U_j$.

Таким чином, ми одержуємо систему рівнянь для знаходження значень U_i, що у матричному виді виглядає в такий спосіб: $(I - W) \cdot U = E$ де I – одинична матриця, U і T – вектори випромінюваної й власної енергій, а матриця W складається з елементів $(\alpha_i w_{ij})$. Оскільки частина випромінювання елемента може не попадати ні на один із що залишилися, те

$$\sum_{i=1}^{N} w_{ij} \leq 1,$$

а ця умова в сполученні з тим, що $\alpha_i < 1$ (відбиття не є повним), приводить до того, що матриця системи має так звану **діагональну перевагу**, тобто діагональний елемент по абсолютній величині більше, ніж сума інших елементів рядка. У такому випадку система рівнянь має рішення, яке можна знайти за допомогою чисельних методів.

Отже, кроки алгоритму зображення сцени зводяться до наступним:

1. Сцена розбивається на окремі ділянки, для кожного з яких визначаються значення $E_i, \alpha_i, w_{ij}, \quad j = 1, 2, \ldots, N$.
2. Перебувають значення U_i для кожного із трьох основних компонентів кольору.
3. Для обраної крапки спостереження стоїть проекція з видаленням невидимих граней і здійснюється зафарбовування, що використовує значення U_i для завдання інтенсивності. При цьому можуть використовуватися які-небудь алгоритми, що дозволяють згладити зображення.

Складним моментом у моделі випромінювання є розрахунок коефіцієнтів w_{ij}.

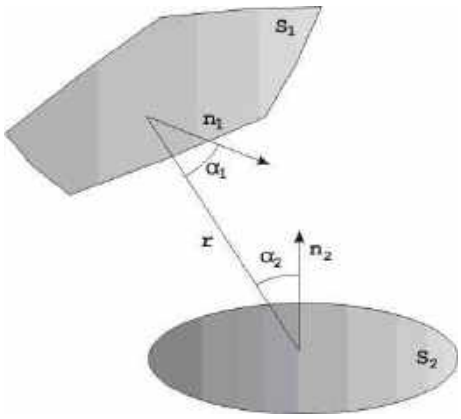


Рис. 12.2. Два елементи сцени

Розглянемо один приклад. Нехай є два елементи сцени S_1 й S_1 (рис. 12.2). Оскільки використовується дифузійна модель висвітлення, то частка енергії малої ділянки dS_1 з нормаллю $\vec{n}_1$, випромінювана під кутом α_1 до цієї нормалі, пропорційна косинусу кута. Отже, у напрямку елементарної ділянки dS_2 йде частка енергії, пропорційна косинусу кута між $\vec{n}_1$ і відрізком, що з'єднує ці ділянки. Відповідно, одержувана другою ділянкою частка цієї енергії буде пропорційна косинусу кута між нормаллю $\vec{n}_2$ й цим же відрізком. Отже, частка енергії, одержувана елементом dS_2 від елемента , dS_1 – $dw_{21} = \cos(\alpha_1) \cdot \cos(\alpha_2)/\pi r^2$, де r – відстань між елементами. Крім того, необхідно врахувати, що випромінювана елементарною ділянкою енергія рівномірно розподілена в усіх напрямках. І, нарешті, у кожній сцені одні об'єкти можуть частково екранувати інші, тому треба ввести коефіцієнт, що визначає ступінь видимості об'єкта з позиції іншого. Далі отримане вираження інтегрується по S_1 й S_2, що також може бути складним завданням.

Звідси видно, наскільки трудомісткої може виявитися процедура обчислення коефіцієнтів w_{ij}. Тому, як правило, використовуються наближені методи їхнього обчислення. Зокрема, можна розглядати поверхні об'єктів як багатогранники, тоді елементами сцени будуть плоскі багатокутники, для яких формули трохи спрощуються.

12.3. Глобальна модель висвітлення із трасуванням променів

Ми вже стосувалися раніше поняття трасування променів при описі алгоритмів видалення невидимих граней. Тепер розглянемо аналогічну процедуру в застосуванні до моделей висвітлення. У попередній главі були описані моделі освітленості від деякого джерела світла без обліку того, що самі об'єкти сцени висвітлюють один одного за допомогою відбитих променів. Метод випромінення, розроблений для дифузійної моделі освітленості, уже враховує цей фактор.

Глобальна модель освітленості здатна відтворювати ефекти дзеркального відбиття й переломлення променів (прозорість і напівпрозорість), а також

затінення. Вона є складовою частиною алгоритму видалення невидимих поверхонь методом трасування.

Якщо розглянути сцену, що містить у числі інших дзеркальні й напівпрозорі поверхні (рис. 12.3), то зображення буде включати, по-перше, проекції самих об'єктів, освітлених одним або декількома джерелами світла. У деяких своїх частинах ці об'єкти будуть перекручені за рахунок переломлення променів у прозорих і напівпрозорих тілах. По-друге, частина об'єктів буде відбиватися дзеркальними поверхнями, і ці відбиття з'являться на проекціях дзеркальних об'єктів. У зображеній на рис. 12.3 сцені крапки на поверхні призми C, D видні на картинній площині двічі: один раз – крізь напівпрозорий паралелепіпед у вигляді крапок $\tilde{C}$, $\tilde{D}$, а другий раз – як двічі відбиті невидимою поверхнею паралелепіпеда й дзеркалом C'', D''. Паралелепіпед у цьому випадку частково має дзеркальні властивості.

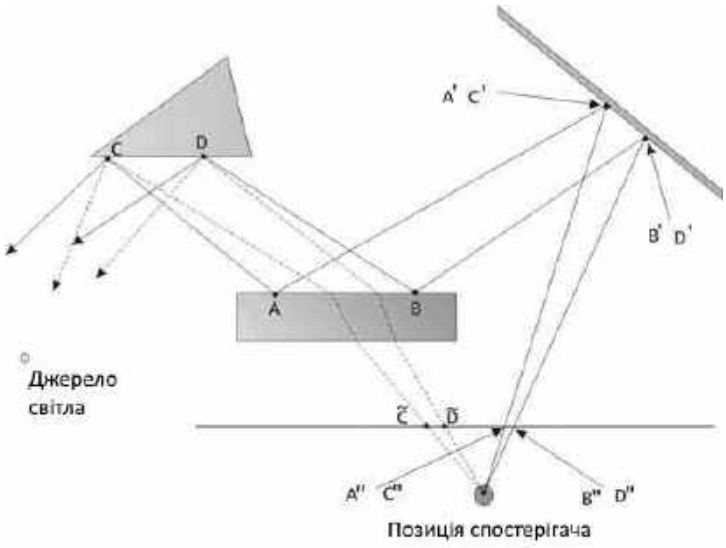


Рис. 12.3. Сцена, що містить дзеркальні й напівпрозорі поверхні

Глобальна модель висвітлення для кожного пікселя зображення визначає його інтенсивність. Будемо для простоти вважати, що всі джерела світла – крапкові. Спочатку визначається безпосередня освітленість джерелами без обліку відбиттів від інших поверхонь (вторинна освітленість): відслідковуються промені, спрямовані до всіх джерел. Тоді спостережувана інтенсивність (або відбита крапкою енергія) виражається наступним співвідношенням:

$$I = k_0 I_0 + k_d \sum_j I_j(\vec{n} \cdot \vec{l}_j) + k_r \sum_j I_j(\vec{s} \cdot \vec{r}_j)^\beta + k_r I_r + k_t I_t,$$

де k_0 – коефіцієнт фонового (неуважного) висвітлення;
k_d – коефіцієнт дифузійного відбиття;
k_r – коефіцієнт дзеркального відбиття;
k_t – коефіцієнт пропущення;
$\vec{n}$ – одиничний вектор нормалі до поверхні в крапці;

$\vec{l}_j$ – одиничний вектор, спрямований до j -му джерела світла;

$\vec{s}$ – одиничний локальний вектор, спрямований у крапку спостереження;

$\vec{r}_j$ – відбитий вектор $\vec{l}_j$;

I_0 – інтенсивність фонового висвітлення;

I_j – інтенсивність j -го джерела світла;

I_r – інтенсивність, що приходить по дзеркально відбитому промені;

I_t – інтенсивність, що приходить по переломленому промені.

В алгоритмі видалення невидимих ліній трасування променю тривала до першого перетинання з поверхнею. У глобальній моделі висвітлення цим справа не обмежується: здійснюється подальше трасування відбитого й переломленого променів. Таким чином, відбувається розгалуження алгоритму у вигляді двійкового дерева. Процес триває доти, поки чергові промені не залишаться без перетинань. Відбиття й переломлення розраховуються за законами геометричної оптики, які вже розглядалися в попередній главі.

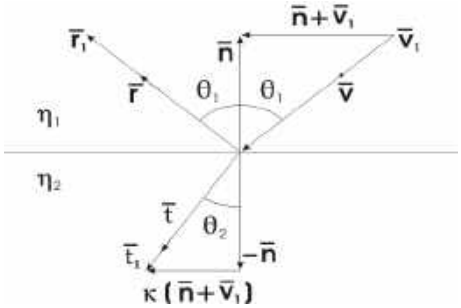


Рис. 12.4. Дзеркальне відбиття й переломлення

Нехай $\vec{v}, \vec{r}, \vec{t}$ - напрямку падаючих, відбитих і переломленого променів (рис. 12.4), $\vec{v}_1 = \vec{v}/\cos(\theta_1)$, $\vec{n}$ – одинична зовнішня нормаль, η_1, η_2 – коефіцієнти переломлення середовищ, розділених поверхнею. Тоді можна показати, що

$$\vec{r}_1 = \vec{v}_1 + 2 \cdot \vec{n}, \quad \vec{t}_1 = \kappa(\vec{n} + v_1) - \vec{n},$$
$$\kappa = (k_\eta^2 \cdot |\vec{v}_1|^2 - |\vec{v}|_1 + \vec{n}^2)^{-1/2}, \quad k_\eta = \frac{\eta_2}{\eta_1}.$$

Відповідні одиничні вектори одержати неважко.

Двійкове дерево променів можна будувати за принципом "ліве піддерево відповідає відбитому променю, а праве – переломленому". Після того як воно побудовано, можна обчислити інтенсивність у крапці. Для цього здійснюється зворотний прохід від вершин до кореня, і при проходженні вузлів інтенсивність убуває.

Теоретично дерево може виявитися нескінченним, тому при його побудові бажано задати максимальну глибину, щоб уникнути переповнення пам'яті комп'ютера.

Оскільки значна частина променів, що виходить від джерел світла й інших поверхонь, не попадає в поле зору спостерігача, то відслідковувати їх усе не

має змісту. Тому для формування зображення використовується зворотне трасування, тобто промені відслідковуються у зворотному порядку: від положення спостерігача через всі крапки картинної площини до об'єктів і далі – по відбитих і переломлених променях.

12.4. Текстури

Текстура поверхні - це деталізація її будови, що враховує мікрорельєф і особливості фарбування. По-перше, гладка поверхня може бути покрита яким-небудь візерунком, і тоді при її зображенні вирішується завдання відображення цього візерунка на проекції фрагментів поверхні (багатокутники). По-друге, поверхня може бути шорсткуватої, тому потрібні спеціальні прийоми імітації такого мікрорельєфу при зафарбовуванні.

Спочатку розглянемо методи відображення візерунків. Найчастіше візерунок задається у вигляді зразка, заданого на прямокутнику в декартовій системі координат η, ξ у просторі текстури. Фрагмент поверхні може бути заданий у **параметричному виді** в тривимірної декартовій системі координат:

$$x = f(u,v), \quad y = g(u,v), \quad z = h(u,v), \quad a \le u \le b, \quad c \le v \le d.$$

Тепер досить побудувати відображення області в просторі текстури в область параметрів поверхні $u = \varphi(\eta,\xi), \quad v = \psi(\eta,\xi),$ або

$\eta = \chi(u,v), \quad \xi = \theta(u,v),$ і тим самим кожній крапці поверхні буде відповідати крапка зразка текстури. Нехай, наприклад, поверхня являє собою один октант сфери одиничного радіуса, заданий формулами

$$x = \sin\alpha \cdot \sin\beta, \quad y = \cos\beta, \quad z = \cos\alpha \cdot \sin\beta,$$
$$0 \le \alpha \le \pi/2, \quad \pi/4 \le \beta \le \pi/2$$

а зразок текстури заданий на квадраті $0 \le \eta \le 1, \quad 0 \le \xi \le 1$. Тоді можна скористатися лінійним відображенням виду $\alpha = a\eta + b, \quad \beta = c\eta + d.$

Якщо покласти $a = \pi/2, \quad b = 0, \quad c = -\pi/4, \quad d = \pi/2$, то кути зразка відобразяться в кути криволінійного чотирикутника, як це показано на рис. 12.6.

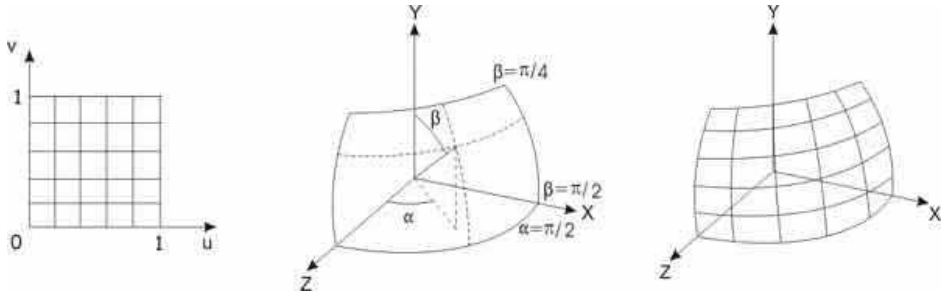


Рис. 12.5. Текстура на сферичній поверхні

Зворотне відображення має вигляд

$$\eta = \frac{\alpha}{\pi/2}, \quad \xi = \frac{\pi/2 - \beta}{\pi/4},$$

отже, вертикальні й горизонтальні лінії зразка відобразяться на фоні великого кола сфери.

Нехай тепер потрібно нанести текстуру при перспективному проектуванні довільно орієнтованої прямокутної грані. Грань задана в просторі набором своїх вершин A, B, C, D. Побудуємо вектори $\vec{e}_1 = B - A$ й $\vec{e}_2 = D - A$, спрямовані уздовж сторін прямокутника. Будь-яку крапку прямокутника можна єдиним образом представити у вигляді $P = A + u\vec{e}_1 + v\vec{e}_2$.

Будемо вважати, що використовується найпростіший випадок перспективного перетворення, що задається формулами

$$x' = \frac{x}{z}, \quad y' = yz.$$

Знайдемо образ крапки P при такому перетворенні:

$$x' = \frac{A_x + u\vec{e}_{1x} + v\vec{e}_{2x}}{A_z + u\vec{e}_{1z} + v\vec{e}_{2z}}, \quad y' = \frac{A_y + u\vec{e}_{1y} + v\vec{e}_{2y}}{A_z + u\vec{e}_{1z} + v\vec{e}_{2x}},$$

або

$$\begin{cases} u(x'\vec{e}_{1z} - \vec{e}_{1x}) + v(x'\vec{e}_{2z} - \vec{e}_{2x}) = A_x - A_z x' \\ \\ u(y'\vec{e}_{1z} - \vec{e}_{1y}) + v(y'\vec{e}_{2z} - \vec{e}_{2y}) = A_y - A_z y' \end{cases}.$$

Якщо тепер розглядати ці співвідношення як систему рівнянь для знаходження параметрів u, v, те, вирішивши її, одержимо необхідне зворотне перетворення. Для рішення можна скористатися, наприклад, правилом Крамера:

$$u = \Delta_u/\Delta, \quad v = \Delta_v/\Delta$$

де

$$\Delta = (x'\vec{e}_{1z} - \vec{e}_{1x}) \cdot (y'\vec{e}_{2z} - \vec{e}_{2y}) - (x'\vec{e}_{2z} - \vec{e}_{2x}) \cdot (y'\vec{e}_{1z} - \vec{e}_{1y});$$

$$\Delta_1 = (A_x - A_z x') \cdot (y'\vec{e}_{2z} - \vec{e}_{2y}) - (x'\vec{e}_{2z} - \vec{e}_{2x}) \cdot (A_y - A_z y');$$

$$\Delta_2 = (x'\vec{e}_{1z} - \vec{e}_{1x}) \cdot (A_y - A_z y') - (A_x - A_z x') \cdot (y'\vec{e}_{1z} - \vec{e}_{1y}).$$

Знайдені параметри будуть визначати крапку текстури, що відповідає крапці проекції.

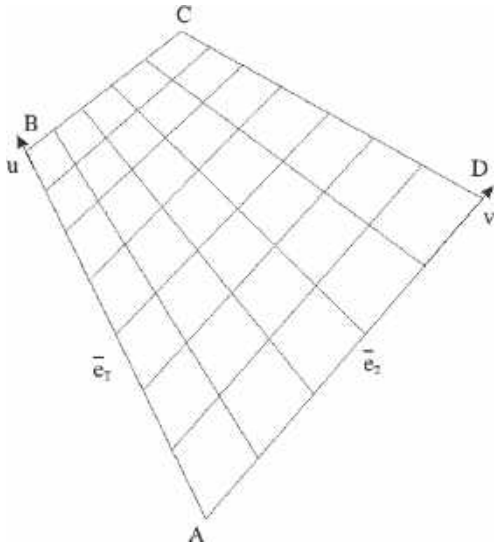


Рис. 12.6. Текстура при перспективній проекції

Можна розглянути більше загальний випадок перспективної проекції, що задається співвідношеннями

$$x' = \frac{x}{1+\frac{z}{d}}, \quad y' = \frac{y}{1+\frac{z}{d}}$$

Тоді рівняння для визначення u, v небагато ускладняться:

$$\begin{cases} u(x'\vec{e}_{1z}/d - \vec{e}_{1x}) + v(x'\vec{e}_{2z}/d - \vec{e}_{2x}) = A_x - (1 + A_z/d)x' \\ \\ u(y'\vec{e}_{1z}/d - \vec{e}_{1y}) + v(y'\vec{e}_{2z}/d - \vec{e}_{2y}) = A_y - (1 + A_z/d)y' \end{cases}.$$

Відповідно, зміниться й рішення:

$$\Delta = (x'\vec{e}_{1z}/d - \vec{e}_{1x}) \cdot (y'\vec{e}_{2z}/d - \vec{e}_{2y}) -$$
$$- (x'\vec{e}_{2z}/d - \vec{e}_{2x}) \cdot (y'\vec{e}_{1z}/d - \vec{e}_{1y});$$
$$\Delta_1 = (A_x - (1 + A_z/d)x') \cdot (y'\vec{e}_{2z}/d - \vec{e}_{2}y) -$$
$$- (x'\vec{e}_{2z}/d - \vec{e}_{2x}) \cdot (A_y - (1 + A_z/d)y')$$
$$\Delta_2 = (x'\vec{e}_{1z}/d - \vec{e}_{1x}) \cdot (A_y - (1 + A_z/d)y') -$$
$$- (A_x - (1 + A_z/d)x') \cdot (y'\vec{e}_{1z}/d - \vec{e}_{1y}).$$

У розглянутих прикладах ми мали справу із гладкими поверхнями. Можна імітувати шорсткість шляхом вибору підходящого зразка нерегулярної текстури, але однаково зображення буде виглядати так, немов неоднорідності нанесені на **гладкій** поверхні. Для моделювання мікрорельєфу Дж.Млинець запропонував метод, заснований на збурюванні нормалі до поверхні.

Нехай, як і раніше, поверхня задана в параметричному виді за допомогою векторної функції $\vec{F}(u,v)$. У кожній її крапці можна побудувати вектор нормалі, скориставшись частками похідними цієї функції. Відомо, що похідні $\vec{F}_u$ і $\vec{F}_v$ являють собою вектори, що лежать у дотичної площини даної поверхні. Тоді вектор нормалі може бути отриманий як векторний добуток цих двох векторів $\vec{n} = \vec{F}_u \times \vec{F}_v$. Після цього крапку поверхні можна відхилити від первісного положення в напрямку нормалі на деяку малу величину, що задається за допомогою функції збурювання $P(u,v)$:

$$\vec{F}'(u,v) = \vec{F}(u,v) + P(u,v) \cdot \vec{n}(u,v).$$

Можна показати, що нормаль до нової обуреної поверхні буде визначатися вираженням

$$\vec{n}' = \vec{n} + \frac{P_u \cdot (\vec{n} \times \vec{F}_u)}{|\vec{n}|} + \frac{P_v \cdot (\vec{n} \times \vec{F}_v)}{|\vec{n}|}.$$

Застосовуючи в моделі висвітлення нову нормаль, можна одержати ефект шорсткості поверхні. Як функція збурювання можна використовувати довільну диференційовану по кожній зі змінних функцію.

12.5 Питання й вправи

1. Які етапи виділяються у світло-тіньовому аналізі?

2. До якого типу ставиться алгоритм Аппеля: ітеративному або рекурсивному?
3. Чи можливо використання алгоритму Аппеля для сцен з неповним затіненням?
4. Що таке тіньовий буфер? Чим він відрізняється від традиційного Z-Буфера?
5. У чому складається модифікація алгоритму Вейлера-Азертона для виконання світло-тіньового аналізу?
6. У якій моделі освітленості можна використовувати метод випромінення?
7. Чим відрізняється трасування променів у глобальній моделі освітленості від методу видалення невидимих граней?
8. Які складові інтенсивності розглядаються в методі трасування?
9. Яким образом можна використовувати двійкові дерева в алгоритмі трасування?
10. Який спосіб завдання поверхні найбільш зручний для текстурування?
11. У чому складається ідея моделювання мікрорельєфу при нанесенні текстур?

Тема 13. АЛГОРИТМИ СТИСКУ ЗОБРАЖЕНЬ БЕЗ ВТРАТ

Необхідність стиску зображень. Неіснування ідеального алгоритму. Алгоритми кодування довжини повторення (RLE): RLE – бітовий рівень, RLE – байтовий рівень. Словникові алгоритми: алгоритм LZ77, алгоритм LZW. Алгоритми статистичного кодування: Алгоритм Хаффмена. Арифметичне кодування.

13.1. Необхідність стиску зображень

Типове зображення, отримане цифровою фотокамерою, має дозвіл порядку 3000×2000, тобто близько 6 мегапікселів; для передачі кольору звичайно використовується 24 біта на піксель. Таким чином, обсяг вихідних даних становить порядку 17 мегабайт. Для професійних пристроїв введення зображень розмір одержуваного растра може бути значно більше, а глибина кольору - досягати 48 біт на піксель (див. лекцію 2). Відповідно, розмір одного зображення може бути більше 200 мегабайт. Тому досить актуальними є алгоритми **стиску** зображень, або, іншими словами, алгоритми, які дозволяють зменшити обсяг даних, що представляють зображення.

Існують два основних класи алгоритмів:

1. A називається алгоритмом **стиску без втрат** (англ. lossless compression), якщо існує алгоритм A-1 (зворотний до A) такий, що для будь-якого зображення I $A(I) = I_1$ і $A\text{-}1(I_1) = I$. Зображення I задане як безліч значень атрибутів пікселів; після застосування до I алгоритму A одержуємо набір даних I_1. Стиск без втрат застосовується в таких графічних форматах подання зображень, як: GIF, PCX, PNG, TGA, TIFF[1)],безліч власних форматів від виробників цифрових фотокамер, і т.д.);
2. A називається алгоритмом **стиску с втратами** (англ. lossy compression), якщо він не забезпечує можливість точного відновлення вихідного зображення. Парний до A алгоритм, що забезпечує зразкове відновлення, будемо позначати як A^*: для зображення I $A(I) = I_1$, $A^*(I_1) = I_2$ і при цьому отримане відновлене зображення I_2 не обов'язково точно збігається з I. Пари A, A^* підбирається так, щоб забезпечити більші коефіцієнти стиску й проте зберегти візуальну якість, тобто домогтися мінімальної різниці в сприйнятті між I і I_2. Стиск із втратами застосовується в наступних графічних форматах: JPEG, JPEG2000 і т.д.

Ця лекція присвячена стиску без втрат, що потрібно у випадках, коли інформація була отримана великою ціною (наприклад, медичні зображення або знімки із супутників), або в інших випадках, коли навіть найменші перекручування небажані [2].

13.2. Неіснування ідеального алгоритму

Як уже було згадано в попередньому пункті, зображення I розглядається як безліч (послідовність) значень атрибутів пікселів. Надалі в цій лекції всі алгоритми й твердження ставляться як до зображень, так і до довільних послідовностей, елементи яких можуть приймати кінцеву кількість значень.

Твердження 13.2.1. Не існує алгоритму, що стискає без втрат будь-який набір даних.

Існує 2^N послідовностей розміру N бітів $\{I_k\}_{k=1}^{2^N}$ (будемо розглядати біт як мінімальний носій інформації). Нехай найдеться алгоритм A такий, що $\forall k = 1, ..., 2^N, |A(I_k)| = M_k < |I_k|$, де |I| - обсяг даних (довжина послідовності). Нехай $M = \max M_k$, тоді $M < N$. Існує $2^1 + 2^2 + ... + 2^M$ послідовностей довжини, меншої або рівної M. Але $2^1+2^2+...+2^M = 2^{M+1-2} < 2^N$. Протиріччя.

Із твердження треба, що має сенс розробляти алгоритми, які б ефективно стискали певний клас зображень; у той же час для цих алгоритмів завжди будуть існувати зображення, для яких вони не забезпечать стиски.

13.3. Алгоритми кодування довжини повторення (RLE)

Цей тип алгоритмів (алгоритми кодування довжини повторення[2](RLE[3]))базується на найпростішому принципі: будемо заміняти повторювані групи елементів вихідної послідовності на парі (кількість, елемент), або тільки на кількість.

13.3.1. RLE – бітовий рівень

Будемо розглядати вихідні дані на рівні послідовності бітів; наприклад, що представляють чорно-біле зображення. Підряд звичайно йде трохи 0 або 1, і можна пам'ятати лише кількість однакових цифр, що йдуть підряд. Т.е. послідовності 0000 11111 000 11111111111 відповідає набір чисел кількостей повторень 4 5 3 11. Виникає наступний нюанс: числа, що позначають кількість повторень, також треба кодувати бітами. Можна вважати, що кожне число повторень змінюється від 0 до 7 (тобто можна закодувати рівно трьома бітами), тоді послідовності 11111111111 можна зіставити числа 7 0 4, тобто 7 одиниць, 0 нулів, 4 одиниці. Для приклада, послідовність, що складається з 21 одиниці, 21 нуля, 3 одиниць і 7 нулів закодується так: 111 000 111 000 111 111 000 111 000 111 011 111, тобто з вихідної послідовності, що має довжину 51 біт, одержали послідовність довжиною 36 біт.

Ідея цього методу використовується при передачі факсів.

13.3.2. RLE - байтовий рівень

Припустимо, що на вхід подається напівтонові зображення, де на значення інтенсивності пікселя приділяється 1 байт. Ясно, що в порівнянні із чорно-білим растром очікування довгого ланцюжка однакових бітів істотно знижується.

Будемо розбивати вхідний потік на байти (букви) і кодувати повторювані букви парою (кількість, буква).

Тобто AABBBCDAA кодуємо (2A) (3B) (1C) (1D) (2A). Однак можуть зустрічатися довгі відрізки даних, де нічого не повторюється, а ми кодуємо кожну букву парою (цифра, буква).

Нехай у нас є фіксоване число (буква) M (від 0 до 255). Тоді одиночну букву $S \le M$ можна закодувати нею самою, – вихід = S, а якщо букв трохи або

$S : M < S \leq 255$, те вихід = CS, де C > M, а C - M – кількість букв, що підряд ідуть, S. Для прикладу приведемо тільки алгоритм декодування.

```
// M – фіксована границя
// зчитуємо символи послідовно із вхідного потоку
// in – вхід – стисла послідовність
// out – вихід – розціплена послідовність
while( in.Read( c ))
{
    if( c > M )
    {   // випадок лічильника
        n = c – M;
        in.Read( c );
        for (i = 0;  i < n;  i++)
            out.Write( c );
    }
    else // випадок просто символу
        out.Write( c );
}
```

Лістинг 13.1. Алгоритм декодування RLE – байтовий рівень

Число M звичайно дорівнює 127. Частіше використовується модифікація цього алгоритму, де ознакою лічильника служить наявність одиниць в 2 старших бітах зчитувального символу. Інші 6 бітів суть значення лічильника.

Така модифікація даного алгоритму використовується у форматі PCX. Однак модифікації даного алгоритму рідко використовуються самі по собі, тому що підклас послідовностей, на яких даний алгоритм ефективний, відносно вузький. Модифікації алгоритму використовуються як одна зі стадій конвеєра стиску (наприклад у форматі JPEG, див. розділ 13.4).

13.4. Словникові алгоритми

Головна ідея, що лежить в основі алгоритмів даного типу, полягає в тім, що замість кодування тільки по одному елементі вхідної послідовності виробляється кодування ланцюжка елементів. При цьому використовується словник ланцюжків (створений по вхідній послідовності) для кодування нових.

13.4.1. Алгоритм LZ77

Даний алгоритм (алгоритм LZ77[4])був одним з перших, що використовують словник [55]. Як словник використовуються N останніх уже закодованих елементів послідовності. У процесі стиску словник-підпослідовність переміщається ("сковзає") по вхідній послідовності. Ланцюжок елементів на виході кодується в такий спосіб: положення співпадаючої частини оброблюваного ланцюжка елементів у словнику – зсув (щодо поточної позиції), довжина, перший елемент, що випливає за частиною, що збіглася, ланцюжка. Довжина ланцюжка збігу обмежується зверху числом n. Відповідно, завдання полягає в знаходженні найбільшого ланцюжка зі словника, що збігає з оброблюваною послідовністю. Якщо ж збігів ні, то

записується нульовий зсув, одинична довжина й тільки перший елемент незакодованої послідовності – {0, 1, e}.

Описана вище схема кодування приводить до поняття **ковзного вікна** (англ. sliding window), що складає із двох частин:

1. підпослідовність уже закодованих елементів довжини N – словник – **буфер пошуку** (англ. search buffer);
2. підпослідовність довжини n з ланцюжка елементів, для якої буде зроблена спроба пошуку збігу – **буфер попереднього перегляду** (англ. look-ahead buffer).

У термінах ковзного вікна алгоритм стиску описується так: якщо $e_1, \ldots, e_i$ – уже закодована підпослідовність, те $e_{i-N+1}, \ldots, e_i$ – суть словник або буфер пошуку, а $e_{i+1}, \ldots, e_{i+n}$ – буфер попереднього перегляду. Аналогічно, завдання полягає в пошуку найбільшого ланцюжка елементів з буфера попереднього перегляду, починаючи з елемента e_{i+1}, що збігає з ланцюжком з буфера пошуку – цей ланцюжок може починатися з будь-якого елемента $e_k : i - N + 1 \le k \le i$ й закінчуватися будь-яким елементом $e_l : k \le l \le i + n$, тобто виходити за межі буфера пошуку, вторгаючись у буфер попереднього перегляду. Природно, що виходити за межі ковзного вікна не можна. Нехай у ковзному вікні знайдена максимальна по довжині ланцюжок, що збігся, елементів $e_{i-p}, \ldots, e_{i+q}$, тоді вона буде закодована в такий спосіб: $\{p+1, q+p+1, e_{i+p+q+2}\}$ – зсув відносно початку буфера попереднього перегляду (e_{i+1}), довжина ланцюжка, що збіглося, елемент, що випливає за ланцюжком, що збігся, з буфера попереднього перегляду. Якщо в результаті пошуку знайдено два збіги з однаковою довжиною, то вибирається найближче до початку буфера попереднього перегляду. Після цього ковзне вікно зміщається на p + q + 2 елементів уперед, і процедура пошуку повторюється.

Вибір чисел N і n є окремою важливою проблемою, тому що чим більше N і n, тим більше місця потрібно для зберігання значень зсуву й довжини. Природно, що час роботи алгоритму також зростає з ростом N і n. Відзначимо, що звичайно N і n розрізняються на порядок.

Приведемо приклад стиску даним алгоритмом. Стиснемо рядок "TOBEORNOTTOBE" з параметрами N = 10 і n = 3:

крок	ковзне вікно	макс. ланцюжок, що збігся	вихід
1	""+"TOB"	T	0,1,T
2	"T"+"OBE"	O	0,1,O
3	"TO"+"BEO"	B	0,1,B
4	"TOB"+"EOR"	E	0,1,E
5	"TOBE"+"ORN"	O	3,1,R
6	"TOBEOR"+"NOT"	N	0,1,N
7	"TOBEORN"+"OTT"	O	3,1,T
8	"TOBEORNOT"+"TOBE"	TOB	9,3,E

Якщо виділити для зберігання зсуву 4 біти, довжини – 2 біти, а елементів – 8 біт, то довжина закодованої послідовності (без обліку позначення кінця послідовності) складе 4 × 8 + 2 × 8 + 8 × 8 = 112 біт, а вихідної – 102 біта. У

цьому випадку ми не зжали послідовність, а, навпаки, збільшили надмірність подання. Це пов'язане з тим, що довжина послідовності занадто мала для такого алгоритму. Але, наприклад, малюнок кодового дерева Хаффмена на рис. 13.1, що займає 420 кілобайт дискового простору, після стиску має розмір близько 310 кілобайт.

Нижче наведений псевдокод алгоритму стиску.

```
// M - фіксована границя
// зчитуємо символи послідовно із вхідного потоку
// in - вхід - стисла послідовність
// n - максимальна довжина ланцюжка
// pos - положення в словнику, len - довжина ланцюжка
// nelem - елемент за ланцюжком, str - знайдений ланцюжок
// in - вхід, out - вихід
// SlideWindow - буфер пошуку
while( !in.EOF() ) //поки є дані
{
    // шукаємо максимальний збіг і його параметри
    SlideWindow.FindBestMatch( in, n, pos, len, nelem );
    // пишемо вихід: зсув, довжина, елемент
    out.Write( pos );
    out.Write( len );
    out.Write( nelem );
    // зрушимо ковзне вікно на len + 1 елементів
    SlideWindow.Move( in, len + 1 );
}
```

Лістинг 13.2. Алгоритм стиску методом LZ77

Декодування стислої послідовності є прямою розшифровкою записаних кодів: кожного запису зіставляється ланцюжок зі словника і явно записаного елемента, після чого виробляється зрушення словника. Очевидно, що словник відтвориться в міру роботи алгоритму декодування.

Видно, що процес декодування значно простіше з обчислювальної точки зору.

```
// n - максимальна довжина ланцюжка
// pos - положення в словнику, len - довжина ланцюжка
// nelem - елемент за ланцюжком, str - знайдений ланцюжок
// in - вхід, out - вихід
// Dict - словник

while( !in.EOF() ) //поки є дані
{
    in.Read( pos );
    in.Read( len );
    in.Read( nelem );
    if( pos == 0 )
    { //новий окремий символ
```

```
        //видаляємо зі словника перший (далекий) один елемент
        Dict.Remove( 1 );
        //додаємо в словник елемент
        Dict.Add( nelem );
        out.Write( nelem );
    }
    else
    {
        //скопіюємо відповідний рядок зі словника
        str = Dict.Get( pos, len );
        //видалимо зі словника len + 1 елементів
        Dict.Remove( len + 1 );
        //Додаємо в словник ланцюжок
        Dict.Add( str + nelem );
        out.Write( str + nelem );
    }
}
```

Лістинг 13.3. Алгоритм

Даний алгоритм є родоначальником цілого сімейства алгоритмів, і сам по собі в споконвічному виді практично не використовується. До його достоїнств можна віднести пристойний ступінь стиску на досить більших послідовностях, швидке розпакування, а також відсутність патенту[5] на алгоритм. До недоліків відносять повільну швидкість стиску, а також меншу, чим в альтернативних алгоритмів, ступінь стиску (модифікації алгоритму борються із цим недоліком). Сполучення алгоритмів Хаффмена (Huffman) (див. розділ 13.5) і LZ77 називають методом DEFLATE[6].Метод DEFLATE використовується в графічному форматі PNG, а також в універсальному форматі стиску даних ZIP.

13.4.2. Алгоритм LZW

Даний алгоритм (алгоритм LZW[7])є модифікацією іншого методу від Абрахама Лемпеля (Abraham Lempel) і Якоба Зіва (Jacob Ziv) – LZ78 [56]. Автор модифікації – Террі Уелч (Terry Welch) [51]. Словник у даному алгоритмі являє собою таблицю, що заповнюється ланцюжками елементів у міру роботи алгоритму. Причому під час декодування словник буде побудований автоматично, отже, немає потреби зберігати його разом зі стислою послідовністю. Звичайно таблиця ініціалізується так, що перші рядки заповнені всіма різними ланцюжками з одного елемента. У процесі стиску відшукується найбільш довгий ланцюжок, уже записана в словник. Щораз, коли новий ланцюжок елементів не знайдений у словнику, вона додається в словник; при цьому записується код ланцюжка, для якої є збіг зі словником. У теорії на розмір таблиці не накладається обмежень, однак обмеження на розмір дозволяє поліпшити ступінь стиску, тому що накопичуються непотрібні (можливо, що більше не зустрічаються) ланцюжка. Чим більше входжень має таблиця, тим більше інформації потрібно виділяти для зберігання кодів. Відповідно

резервується спеціальний код, що позначає очищення таблиці (точніше, приведення її до вихідного стану).

```
// elem - елемент, chain - ланцюжок елементів
// in - вхід, out - вихід
// Table - таблиця ланцюжків
// Table.Find - повертає код, що відповідає
// ланцюжку (0 інакше)

str = ""; // порожній ланцюжок
while( in.Read( elem ) ) // поки є дані
{
    // перевіряємо, є чи вже в таблиці новий ланцюжок
    if( Table.Find( chain + elem ) )
    {   // є
        // збережемо новий ланцюжок для майбутніх перевірок
        chain = chain + elem;
    }
    else
    {   // немає
        // запишемо код, що відповідає chain
        out.Write( Table.Find( chain ) );
        // додаємо в словник
        Table.Add( chain + elem );
        chain = elem;
    }
}
//залишається один неопрацьований ланцюжок
out.Write( Table.Find( chain ) );
```

Лістинг 13.4. Алгоритм стиску LZW

Розглянемо приклад стиску алгоритмом. Будемо, як і в попередньому пункті, стискати рядок "TOBEORNOTTOBE". Нехай таблиця ініціалізована 256 символами коду ASCII. Припускаємо, що таблиця може містити 512 входжень, тобто нам потрібно зберігати 9 біт для кожного коду; для зручності нехай вихідний словник виглядає так:

рядок	код
T	1
O	2
B	3
E	4
R	5
N	6
...	...
CLT	256

Тут CLT означає "очистити (ініціалізувати) таблицю" самий довгий ланцюжок, що збігся, – "T", вихід – {1}, словник:

рядок	код
T	1
O	2
B	3
E	4
R	5
N	6
...	...
CLT	256
TO	257

самий довгий ланцюжок, що збігся, – "O", вихід – {2}, словник має такий вигляд:

рядок	код
...	...
OB	258

Далі подібним чином у вихід записується послідовність {3}{4}{1}{5}{6}{2}, після цього самий довгий ланцюжок, що збігся, – "T", вихід: {1}, словник має такий вигляд:

рядок	код
...	...
TO	257
OB	258
BE	259
EO	260
OR	261
RN	262
NO	263
OT	264
TT	265

самий довгий ланцюжок, що збігся, – "TO", вихід: {257}, словник має такий вигляд:

рядок	код
...	...
TOB	266

самий довгий ланцюжок, що збігся, – "BE", вихід: {259}.

У результаті закодована послідовність виглядає так: {CLT}{1}{2}{3}{4}{1}{5}{6}{2}{1}{257}{259} (звичайно на початку закодованої послідовності записують код очищення таблиці для спрощення декодера). Отримана послідовність має обсяг 108 біт (без обліку позначення кінця послідовності), що більше, ніж вихідна (104 біта). У цьому випадку це обумовлено занадто малою довжиною вихідної послідовності. Однак уже згадуваний рис. 13.1 кодового дерева Хаффмена стискується за допомогою даного алгоритму з вихідних 420 кілобайт до290 кілобайт.

```
// chain - ланцюжок незакодованих елементів
// in - вхід, out - вихід
// lcode, code - елемент закодованої послідовності
// Table - таблиця ланцюжків
// Table.Get(code) - повертає ланцюжок з кодом code
// Table.IsOccupied(code) - є чи запис у словнику для code
// Firstelem(chain) - повертає перший елемент ланцюжка

while( in.Read( code ) ) // поки є дані
{
    if( code == CLT ) //прочитаний код очищення таблиці?
    {    // да
         Table.Init();
         If ( !in.Read(code) ) break; // більше немає інформації
         out.Write( Table.Find( code ) );
         lcode = code;
    }
    else
    {   // немає
         if( Table.IsOccupied( code ) )
         {
              out.Write( Table.Find( code ) );
              // заповнюємо словник новим ланцюжком
              Table.Add( Table.Find( lcode ) +
                             Firstelem(Table.Find( code ) ) );
              lcode = code;
         }
         else
         {    // code не має відповідного ланцюжка
              chain = Table.Find( lcode ) +
                             Firstelem( Table.Find( lcode ) );
              out.Write ( chain ); // запишемо ланцюжок
              Table.Add ( chain ); // і додамо в таблицю
              lcode = code;
         }
    }
}
```

Лістинг 13.5. Алгоритм декодування LZW

Декодування полягає в прямій розшифровці кодів, тобто в побудові словника й виводу відповідних ланцюжків. Словник ініціюється так само, як і в кодері.

До достоїнств алгоритму можна віднести високий ступінь стиску й досить високу швидкість як стиску, так і розгортанні. До недоліків донедавна

відносили патентну захищеність алгоритму, однак із середини 2004 року цей недолік не актуальний.

Наведений вище алгоритм має безліч модифікацій, наприклад: різні варіанти подання таблиць і пошуку в них, змінна довжина кодів і т.д. Модифікації LZW використовуються в безлічі архіваторів загального призначення, а також у таких форматах як GIF і TIFF.

13.4.3. Алгоритми статистичного кодування

Алгоритми статистичного кодування ставлять у відповідність кожному елементу послідовності код так, щоб його довжина відповідала ймовірності появи елемента. Таким чином, стиск відбувається за рахунок заміни елементів вихідної послідовності, що мають однакові довжини (кожний елемент займає однакову кількість біт), на елементи різної довжини, пропорційної негативному логарифму від імовірності, тобто елементи, що зустрічаються частіше, ніж інші, будуть мати код меншої довжини. Відповідно до теореми Шеннона про кодування джерела без перешкод [5], оптимальна довжина такого коду є – $\log_s p$, де p – імовірність появи елемента у вхідній послідовності, а s – підстава системи числення, у якій представляється закодований елемент (код елемента).

13.4.4. Алгоритм Хаффмена

Алгоритм Хаффмена[8)] використовує особливий вид подання елементів – **префіксний код**. Префіксний код – це код змінної довжини, що володіє особливою властивістю – властивістю префікса: менш короткі коди не збігаються із префіксом (початковою частиною) більше довгих. Такий код дозволяє здійснювати взаємо-однозначне кодування. Відповідно, потрібно побудувати спосіб стиску, що використовує префіксний код і при цьому забезпечуючий максимально можливий ступінь стиску. Формалізуємо завдання.

Дано на вході:

Алфавіт $A = \{a_1, \ldots, a_n\}$.

Безліч $P = \{p_1, \ldots, p_n\}$ – розподіл імовірностей появи або таблиця кількостей елементів з A, $p_i = Prob(a_i)$, $1 \le i \le n$. Далі $p_i \in P$ будемо називати **вагою** a_i.

Необхідно на виході:

Код (алфавіт) $H(A, P) = \{h_1, \ldots, h_n\}$ – набір кодів (далі бінарних), так що $h_i = Code(a_i)$, $1 \le i \le n$.

Завдання:

Нехай $S(H) = \sum_{i=1}^{n} p_i \cdot l(h_i)$, де $l(h_i)$ – довжина коду h_i. $S(H)$ – зважена довжина шляху (пояснення даного терміна див. нижче).

Потрібно, щоб H було оптимально, тобто

$$S(H) \le S(T) \quad \forall T(A, P).$$

Алгоритм Хаффмена є рішенням даного завдання [36], [37].

Побудова набору кодів звичайно здійснюється за допомогою так званих **кодових дерев**. Припустимо, що використовуються бінарні коди, тоді дерево

буде бінарним (у загальному випадку ступінь розгалуження залежить від підстави системи числення подання кодів). Термінальні вузли дерева містять сам елемент алфавіту, його вагу, посилання на батьківський вузол (втім, це залежить від конкретної реалізації). Внутрішні вузли містять суму ваг своїх вузлів-нащадків, посилання на вузли-нащадки й, можливо, посилання на батьківський вузол. Ребрам дерева зіставляються нуль і одиниця, для лівих і правих відповідно. Повністю добудоване дерево має n термінальних вузлів і n-1 внутрішніх. Роблячи прохід від кореня до термінального вузла й виписуючи по шляху 0 і 1 для ребер, що зустрічаються, одержимо код елемента в термінальному вузлі.

Тепер став зрозумілий зміст назви S(H), тому що довжина коду $l(h_i)$ суть довжина шляхи від кореня до відповідної термінальної вершини.

Опишемо алгоритм побудови **кодового дерева Хаффмена.**

1. Створити n термінальних вузлів по числу елементів в алфавіті. У кожний вузол записати відповідні ваги.
2. Створити батьківський вузол і з'єднати з ним два вільних (тобто не батьки, що мають) вузла з мінімальними вагами, записавши в нього суму ваг безпосередніх нащадків.
3. Якщо кількість вільних вузлів більше одного, то перейти до пункту 2. Інакше даний вузол – корінь дерева Хаффмена.

Процес стиску полягає в заміні кожного елемента вхідної послідовності його кодом. Стиснемо наступний рядок "TOBEORNOTTOBE".

1. Складемо таблицю ваг:

елемент алфавіту						
вага						

2. Побудуємо дерево Хаффмена (див. рис. 13.1):

– проініціюєм дерево термінальними вузлами,
– створимо батьківський вузол N1 для вузлів елементів "N" і "R" з вагою 2;
– створимо батьківський вузол N2 для вузла елемента "E" і вузла N1 з вагою 4;
– створимо батьківський вузол N3 для вузлів елементів "T" і "B" з вагою 5;
– створимо батьківський вузол N4 для вузла елемента "O" і вузла N2 з вагою 8;
– створимо батьківський вузол N5 для вузлів N3 і N4 с вагою 13.

3. Проведемо кодування. Складемо для зручності таблицю кодів:

елемент алфавіту				E	R	N
код	0	0	1	010	0110	0111

4. Отже, одержимо послідовність "10 00 11 010 00 0110 0111 00 10 10 00 11 010".

Отримана послідовність має довжину 32 біта. Вихідна, якщо вважати алфавітом набір ASCII – 104 біта, якщо вважати алфавітом тільки набір "TOBERN" – 39 біт.

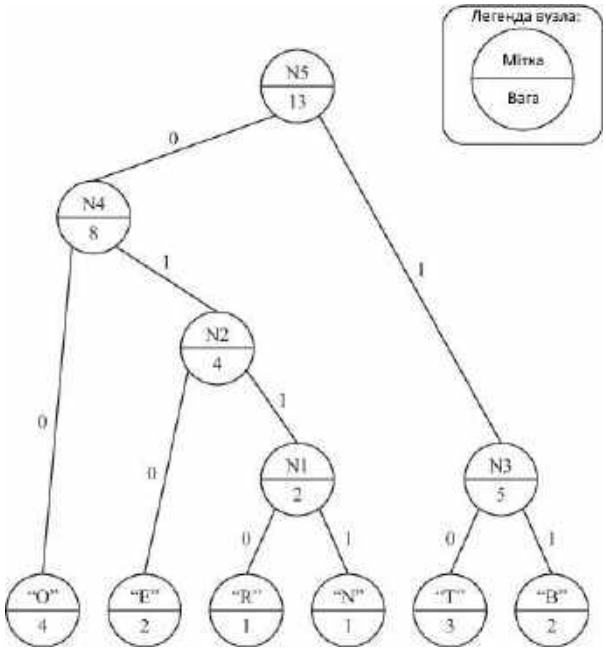


Рис. 13.1. Приклад дерева Хаффмена

Помітимо, що в процесі побудови дерева виникала неоднозначність у виборі двох вузлів для обробки. У нашім прикладі вибір був зроблений на користь наочності подання, однак ніякого впливу на якість стиску воно не робить.

Декодування здійснюється прямою заміною коду на відповідний елемент. І в силу того, що використано префіксний код, не потрібно відзначати початок і кінець чергового коду.

Наведений вище алгоритм є двохпрохідним. Перший прохід по зображенню створює таблицю кількостей (ваг) елементів, а під час другого відбувається кодування. Існують реалізації алгоритму з фіксованою таблицею (CCITT Group 3 – використовується для передачі факсимільних зображень). Часто буває, що нам невідомо апріорний розподіл імовірностей елементів алфавіту, тому що нам не доступна вся послідовність відразу. Або, наприклад, бажано позбутися від необхідності зберігати таблицю ваг для декодування. Відповідно, були запропоновані адаптивні модифікації алгоритму Хаффмена.

Розглянемо модифікацію алгоритму, у якій дерево будується в міру надходження нових елементів так, щоб зберігалася впорядкованість по вагам вузлів. Упорядкованість наступного виду: при проході кожного рівня дерева, починаючи з нижнього ліворуч праворуч, ваги вузлів не убувають. Побудова дерева виконується в такий спосіб.

- На першому кроці будується звичайне дерево Хаффмена, уважаючи, що ваги всіх вершин рівне (таке дерево буде впорядкованим у зазначеному вище змісті).
- При надходженні чергового елемента виконується наступне:

- елемент кодується;
- збільшується значення ваги у відповідному термінальному вузлі;
- збільшуються значення ваг у відповідних вершинах по шляху від цього вузла до кореня;
- якщо після цього впорядкованість дерева порушена, то:
  - для такого вузла (А) серед сусідів праворуч шукається крайній вузол (Б), вага якого менше;
  - вузли А и Б міняються місцями, при цьому ваги вузлів на шляхах від переставлених до кореня коректуються відповідним чином;
- якщо дерево знову з, то знову виконується попередній крок.

Декодування виконується за аналогічною схемою: ініціалізація дерева, одержання коду, розшифровка, відновлення дерева, перебудова дерева (якщо необхідно), і т.д.

Алгоритм Хаффмена використовується на одній зі стадій стиску у форматі JPEG, як складова частина методу DEFLATE, а також у багатьох інших універсальних кодерах. Хоча даний метод уступає по ступені стиску арифметичному кодуванню, його простота, швидкість і термін, що закінчився, дії патентів забезпечують широку поширеність.

13.4.5. Арифметичне кодування

Класичні статистичні алгоритми зіставляють елементу кодуючої послідовності якийсь код. Алгоритми **арифметичного кодування** кодують відразу ланцюжок елементів у число-дріб $f(0 \le f < 1)$ [40]. При цьому враховується розподіл імовірностей появи елементів у послідовності.

Нехай $a_1, \ldots, a_n$ - алфавіт, всі можливі одноелементні ланцюжки; $p_1, \ldots, p_n$ – імовірності появи елементів (ваг) ($p_j = Prob(a_j)$). Розіб'ємо напівінтервал [0, 1) на n непересічних напівінтервалів $I_1, \ldots, I_n$, що відповідають елементам $a_1, \ldots, a_n$, причому довжина I_j пропорційна p_j.

Далі будується дріб, що кодує: виробляється побудова системи вкладених напівінтервалів так, що кожний наступний напівінтервал займає в попереднє місце, що відповідає положенню елемента у вихідній розбивці напівінтервалу [0, 1).

Коротко процес виглядає так:

– зчитування чергового елемента;

– вибір відповідного напівінтервалу з розбивки поточного напівінтервалу (на першому кроці – [0, 1)).

По закінченні цього процесу можна взяти будь-яке число з напівінтервалу, що вийшов. Звичайне число вибирають так, щоб його запис у двійковому виді була самої короткою. Також часто накладають обмеження на довжину двійкового дробу, що вийшло. При перевищенні цього обмеження поточний дріб виводиться, і починається спочатку формування нової, що доповнює попередню(ие).

// elem - елемент

```
// in - вхід, out - вихід
// cleft - поточна ліва границя напівінтервалу
// cright - поточна права границя напівінтервалу
// interv - масив границь у вихідній розбивці

cleft = 0;
cright = 1;
while( in.Read( elem ) ) // поки є дані
{
    // довжина поточного напівінтервалу
    d = cright - cleft;
    // обчислимо границі нового напівінтервалу
    cright = cleft + d*interv[elem].right;
    cleft = cleft + d*interv[elem].left;
}
// шукаємо дріб усередині [left, right)
frac = Between( left, right );
out.Write( frac );
```

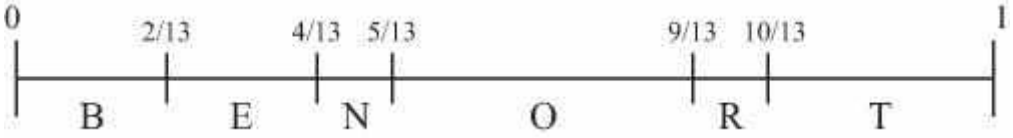
Лістинг 13.6. Алгоритм арифметичного кодування

Стиснемо наступну послідовність "TOBEORNOTTOBE".

Складемо таблицю ваг:

елемент алфавіту						
вага						

Побудуємо розбивку напівінтервалу [0, 1):



– На вході T, тоді права границя напівінтервалу буде $\frac{10}{13}$, ліва - 1, довжина - $\frac{3}{13}$;

– На вході O, тоді права границя напівінтервалу буде $\frac{145}{169}$, ліва - $\frac{157}{169}$, довжина - $\frac{12}{169}$;

– На вході B, тоді права границя напівінтервалу буде $\frac{1885}{2197}$, ліва - $\frac{1909}{2197}$, довжина - $\frac{24}{2197}$;

– На вході E, тоді права границя напівінтервалу буде $\frac{24553}{28561}$, ліва - $\frac{24601}{28561}$, довжина - $\frac{48}{2197}$;

– ...

– На вході E, тоді права границя напівінтервалу буде $\frac{260680817264869}{302875106592253}$, ліва - $\frac{260680817375461}{302875106592253}$, довжина - $\frac{110592}{302875106592253}$.

Двійковий дріб можна взяти таку: 0.11011100010101100000010000000101, тому що двійковий запис границь підсумкового напівінтервалу наступна: ліва - 0.11011100010101100000010000000100111010011000101101, права - 0.11011100010101100000010000000101101100100100100001.

Закодована послідовність має довжину 31 біт. Вихідна, якщо вважати алфавітом набір ASCII – 104 біта, якщо вважати алфавітом тільки набір "TOBERN" – 39 біт.

Декодування полягає в розшифровці дробу по відомому розподілі ймовірностей появи елементів у послідовності. Отже, потрібно зберігати або передавати інформацію про розподіл імовірностей появи елементів. Для того щоб обійти ця вимога, а також для того щоб позбутися від необхідності робити два проходи по послідовності, були запропоновані адаптивні модифікації алгоритму.

```
// elem - елемент, in - вхід, out - вихід
// interv - масив границь у вихідній розбивці
// frac - дріб, що кодує
// cleft - поточна ліва границя напівінтервалу
// cright - поточна права границя напівінтервалу
// nelem - кількість елементів, закодована дробом

i = nelem;
in.Read( frac );
while( i > 0 ) // поки треба декодувати
{
    // шукаємо елемент по дробі й розбивці напівінтервалу
    elem = GetElem( interv, frac );
    out.Write( elem );
    // обчислимо границі напівінтервалу
    cright = interv[elem].right;
    cleft = interv[elem].left;
    // довжина напівінтервалу
    d = cright - cleft;
    // перетворимо дріб
    x = (x - cleft) / d;
    // відзначимо обробку чергового елемента
    i-i--;
}
```

Лістинг 13.7. Алгоритм арифметичного декодування

Опишемо ідею адаптивної модифікації. Споконвічно передбачається, що поява всіх елементів рівноможлива. У міру надходження нових елементів ваги

й розбивка напівінтервалу коректуються. При декодуванні також здійснюється корекція після обробки чергового елемента.

Арифметичне кодування є майже оптимальним видом кодування, тобто дає коди, майже рівні довжинам оптимальних кодів з теореми Шеннона. Однак така ефективність досягається за рахунок більших обчислювальних витрат. Існують модифікації алгоритму, що використовують тільки операції зрушення й додавання без розподілу й множення для прискорення процесу кодування. На жаль, практично всі такі модифікації захищені патентами. Частина цих патентів уже мають термін, що закінчився, дії, інша частина – немає. Таким чином, у міру витікання строків патентів ефективні реалізації модифікацій алгоритму арифметичного кодування будуть витісняти алгоритм Хаффмена в сферах, де продуктивність не є критичною вимогою.

1) Загалом кажучи, даний формат також підтримує стиск із втратами.

2) У літературі часто зустрічається й інша назва – групове кодування.

3) англ. Run Length Encoding

4) Названий по іменах авторів Абрахама Лемпеля (Abraham Lempel) і Якоба Зива (Jacob Ziv). Опублікований в 1977 році.

5) документ, що забезпечує виключне право експлуатувати винахід протягом відомого часу (звичайно 15-20 років)

6) Так називається стиск, а разжатие називається INFLATE (англ. DEFLATE – здувати, INFLATE - надувати).

7) Алгоритм названий по ім'ю автора Терри Уелч (Terry Welch) і назви вихідного алгоритму LZ78 (також від імен Лемпеля й Зива). Він був опублікований в 1984 році.

8) Huffman coding. Автор алгоритму – Давид Хаффмен (David A. Huffman). Опублікований в 1952 році.

Тема 14. СТИСК ЗОБРАЖЕНЬ ІЗ ВТРАТАМИ

Необхідність стиску із втратами. Оцінка втрат. Зображення як функція: дискретне Перетворення Фур'є, дискретне косинусне перетворення. Алгоритм стиску зображень JPEG. Вейвлет-Перетворення. Фрактальний стиск.

14.1. Необхідність стиску із втратами

Як зазначено в першому параграфі попередньої лекції, обсяг інформації, потрібної для зберігання зображень, звичайно великий. При цьому з кожним днем роздільна здатність пристроїв введення зображень росте. Якийсь час назад стало очевидно, що класичні алгоритми не забезпечують достатнього ступеня стиску. Будучи алгоритмами загального призначення, вони не враховували того, що стислива інформація є зображення. Відповідно, були розроблені алгоритми, що забезпечують стиск із втратами інформації, які дозволили істотно підняти ступінь компресії.

Стиск із втратами ґрунтується на особливостях сприйняття людиною зображення: найбільшої чутливості в певному діапазоні хвиль кольору, здатності сприймати зображення як єдине ціле, не зауважуючи дрібних перекручувань. Також використовується той факт, що зображення – двовимірний об'єкт. Головний клас зображень, на який орієнтовані алгоритми стиску із втратами, – фотографії, зображення із плавними колірними переходами.

14.2. Оцінка втрат

Нагадаємо, що A називається алгоритмом **стиску с втратами** (англ. lossy compression), якщо він не забезпечує можливість точного відновлення вихідного зображення. Т.е. підбирається пара A, A^*, де A^* приблизно відновлює зображення: для будь-якого зображення I A(I) = I_1, $A^*(I_1) = I_2$ і при цьому отримане відновлене зображення I_2 не обов'язково точно збігається з I (див. визначення в першому параграфі попередньої лекції). Виникає питання: як оцінювати втрати візуальної інформації, тобто міру відмінності I від I_2? Існує безліч гарних мір для оцінки таких втрат, однак для всіх з них можна підібрати такі два зображення, що їхня міра відмінності буде досить великий, але на око розходження будуть майже непомітними. І навпаки – можна підібрати зображення, що сильно розрізняються на око, але які мають невелику міру відмінності.

Уведемо спочатку норму для значень атрибутів пікселів. Для напівтонових зображень, де на кожне значення атрибута пікселя приділяється 8 біт:

$$\|I(i,j)\| = |\, I(i,j)\,|^2.$$

Причому $M^* = max\|\cdot\|$, тобто максимально можливе значення для такої норми дорівнює 255 × 255 = 65025. Для повноколірних зображень із трьома 8-бітними значеннями атрибута пікселя (трьома 8-бітними каналами):

$$\|I(i,j)\| = |\, R(i,j)\,|^2 + |\, G(i,j)\,|^2 + |\, B(i,j)\,|^2,$$

отже $M^* = 3 \times 255 \times 255 = 195075$.

Стандартною мірою відмінності є міра **відносини сигналу до шуму** (PSNR – англ. Peak Signal-to-Noise Ratio), обумовлена так:

$$PSNR(I_1, I_2) = 10log_{10}\left(\frac{M^*}{MSE(I_1, I_2)}\right),$$

де MSE(I1, I2) – інша міра – **середньоквадратична помилка** (L 2-міра, MSE – англ. Mean Squared Error), обумовлена так:

$$MSE(I_1, I_2) = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\|I_1(i,j) - I_2(i,j)\|.$$

Проте найважливішою "мірою" оцінки втрат є думка спостерігача. Чим менше розходжень (а краще – їхня відсутність) виявляє спостерігач, тим вище якість алгоритму стиску.

Відзначимо важливий факт, що алгоритми стиску із втратами часто надають користувачеві можливість вибирати обсяг "" даних, що втрачаються. Таким чином, існує вибір між якістю й розміром стислого зображення. Природно, що чим краще візуальна якість при більшому коефіцієнті стиску, тим алгоритм краще.

Рис. 14.1. **8**-бітне напівтонове зображення

14.3. Зображення як функція

Будемо розглядати зображення як функцію двох змінних, певну в крапках кінцевого растра (мається на увазі крапкова модель растра, див. визначення в I(x, y) – значення атрибута пікселя (наприклад номер у палітрі, інтенсивність), що залежить від колірної моделі подання зображення (див. рис. 14.1 і рис. 14.2). Безліч таких функцій на крапках фіксованого кінцевого растра утворять кінечномірний евклідовий простір $R^{X,Y}$ розмірності m × n (|X|=m,|Y|= n) зі скалярним добутком

$$(I_1, I_2) = \sum_{i,j=0}^{m,n} I_1(i,j) \cdot I_2(i,j).$$

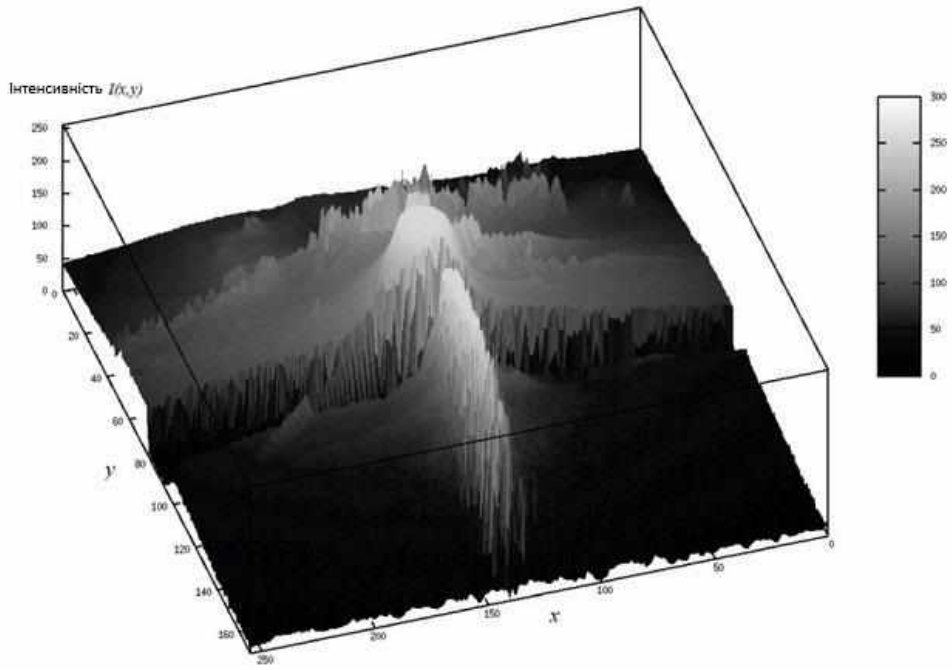


Рис. 14.2. Зображення як функція двох змінних

Будемо ототожнювати з таким простором $L_2(X \times Y)$. У такому просторі існує базис (див. [3]), тобто така система елементів $\{e_k\}_{k=1}^{k=m\times n}$ з $R^{X,Y}$ і такий набір не рівних одночасно нулю коефіцієнтів $\{C_k\}_{k=1}^{k=m\times n}$, що для будь-якої функції I із цього простору виконане

$$I = \sum_{k=0}^{m\cdot n} C_k e_k.$$

Якщо додатково припустити ортонормальність базису, тобто

$$(e_p, e_q) = \begin{cases} 0, & p \neq q \\ 1, & p = q \end{cases},$$

те виконується наступне співвідношення: $C_k = (I, e_k)$.

14.4. Дискретне Перетворення Фур'є

Нагадаємо визначення двовимірного дискретного перетворення Фур'є функції I:

$$F(k,l) = \sum_{p=0}^{m}\sum_{q=0}^{n} I(p,q) e^{-\frac{2\pi i p}{m} k e - \frac{2\pi i q}{n} l}$$

для всіх k = 1 . . .m, l = 1 . . . n. Зворотне перетворення визначається так:

$$I(p,q) = \frac{1}{mn}\sum_{k=0}^{m}\sum_{l=0}^{n} F(k,l) e^{-\frac{2\pi i k}{m} p e - \frac{2\pi i l}{n} q}.$$

Система функцій $\left\{e^{2\pi\left(p\frac{k}{m}+q\frac{l}{n}\right)}\right\}_{k,l=0}^{m,n}$ утворить базис у просторі функцій-зображень (см. [3]).

Таке перетворення дуже популярно в області обробки зображень, однак у загальному виді практично не використовується для стиску зображень через погану частото-просторову локалізацію.

14.5. Дискретне косинусне перетворення

Розглянемо визначення **дискретного косинусного перетворення** (ДКП) [43]. Нехай зображення має розміри N × N. Пряме перетворення записується так:

$$t(u,v) = c(u)c(v)\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} I(k,l)\cos\frac{(2k+1)u\pi}{2N}\cos\frac{(2l+1)v\pi}{2N},$$

$$i,j = 0,...,N-1, c(i) = \begin{cases} \sqrt{\frac{1}{N}}, & i=0 \\ \sqrt{\frac{2}{N}}, & i\neq 0 \end{cases}.$$

Зворотне перетворення має такий вигляд:

$$I(k,l) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} c(u)c(v)t(u,v)\cos\frac{(2k+1)u\pi}{2N}\cos\frac{(2l+1)v\pi}{2N},$$

$$i,j = 0,...,N-1, c(i) = \begin{cases} \sqrt{\frac{1}{N}}, & i=0 \\ \sqrt{\frac{2}{N}}, & i\neq 0 \end{cases}.$$

Дискретне перетворення має властивості.

– **Некорелювання коефіцієнтів.** Коефіцієнти незалежні друг від друга, тобто точність подання одного коефіцієнта не залежить від будь-якого іншого.

– **"Ущільнення" енергії** (англ. energy compaction). Перетворення зберігає основну інформацію в малій кількості коефіцієнтів. Дана властивість сильніше всього проявляється на фотореалістичних зображеннях.

Коефіцієнти t(u, v) – це амплітуди просторових частот зображення. У випадку зображень із плавними переходами більша частина інформації втримується в низькочастотному спектрі.

Відзначимо, що застосування дискретний косинус-перетворення еквівалентно застосуванню дискретного перетворення Фур'є приблизно подвійної довжини до дійсного (некомплексним) і парно симетричним даним (еквівалентність випливає з того, що перетворення Фур'є парної дійсної функції парно й дійсно).

14.6. Алгоритм стиску зображень JPEG

Алгоритм стиску, використовуваний у форматі зберігання зображень JPEG[1)][49],побудований на використанні дискретного косинусного перетворення. Схема стиску в алгоритмі являє собою конвеєр, де дане перетворення – лише одна зі стадій (хоча, можливо, і найважливіша). Опишемо

алгоритм, припускаючи, що на вхід дані 24-бітне зображення, значення атрибутів пікселів якого є елементами колірного простору RGB.

1. **Переклад у колірний простір YCbCr** (докладніше див. лекцію 1). Тут Y – компонента яскравості, Cb і Cr – компоненти кольоровості. Людське око більше чутливе до яскравості, чим до кольору. Тому важливіше зберегти більшу точність при передачі Y, чим при передачі Cb і Cr. Переклад здійснюється по наступній формулі:

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0,299 & 0,587 & 0,114 \\ -0,1687 & -0,3313 & 0,5 \\ 0,5 & -0,4187 & -0,0813 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix};$$

зворотний переклад:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1,402 \\ 1 & -0,34415 & -0,71414 \\ 1 & 1,772 & 0 \end{pmatrix} \begin{pmatrix} Y \\ Cb-128 \\ Cr-128 \end{pmatrix}.$$

2. **Субдискретизація компонент кольоровості.** Після перекладу в колірний простір YCbCr здійснюється субдискретизація по наступних співвідношеннях: 4:4:4 (відсутність передискретизації), 4:2:2 (компоненти кольоровості міняються через одну по горизонталі), 4:2:0 (компоненти кольоровості міняються через одну по горизонталі; при цьому по вертикалі вони міняються через рядок). Проілюструємо докладніше дані співвідношення на прикладах. Будемо перетворювати блок 4 × 4 пікселя зображення:
   1. $Y_{00}Cb_{00}Cr_{00}$ $Y_{01}Cb_{01}Cr_{01}$ $Y_{02}Cb_{02}Cr_{02}$ $Y_{03}Cb_{03}Cr_{03}$
   2. $Y_{10}Cb_{10}Cr_{10}$ $Y_{11}Cb_{11}Cr_{11}$ $Y_{12}Cb_{12}Cr_{12}$ $Y_{13}Cb_{13}Cr_{13}$
   3. $Y_{20}Cb_{20}Cr_{20}$ $Y_{21}Cb_{21}Cr_{21}$ $Y_{22}Cb_{22}Cr_{22}$ $Y_{23}Cb_{23}Cr_{23}$
   4. $Y_{30}Cb_{30}Cr_{30}$ $Y_{31}Cb_{31}Cr_{31}$ $Y_{32}Cb_{32}Cr_{32}$ $Y_{33}Cb_{33}Cr_{33}$

   тоді:

   4:4:4. Субдискретизований блок буде таким же.

   4:2:2. Субдискретизований блок буде таким:

   $Y_{00}Cb_{00}Cr_{00}$ $Y_{01}Cb_{00}Cr_{00}$ $Y_{02}Cb_{02}Cr_{02}$ $Y_{03}Cb_{02}Cr_{02}$
   $Y_{10}Cb_{10}Cr_{10}$ $Y_{11}Cb_{10}Cr_{10}$ $Y_{12}Cb_{12}Cr_{12}$ $Y_{13}Cb_{12}Cr_{12}$
   $Y_{20}Cb_{20}Cr_{20}$ $Y_{21}Cb_{20}Cr_{20}$ $Y_{22}Cb_{22}Cr_{22}$ $Y_{23}Cb_{22}Cr_{22}$
   $Y_{30}Cb_{30}Cr_{30}$ $Y_{31}Cb_{30}Cr_{30}$ $Y_{32}Cb_{32}Cr_{32}$ $Y_{33}Cb_{32}Cr_{32}$

   4:2:0. Субдискретизований блок буде таким:

   $Y_{00}Cb_{00}Cr_{00}$ $Y_{01}Cb_{00}Cr_{00}$ $Y_{02}Cb_{02}Cr_{02}$ $Y_{03}Cb_{02}Cr_{02}$
   $Y_{10}Cb_{00}Cr_{00}$ $Y_{11}Cb_{00}Cr_{00}$ $Y_{12}Cb_{02}Cr_{02}$ $Y_{13}Cb_{02}Cr_{02}$
   $Y_{20}Cb_{20}Cr_{20}$ $Y_{21}Cb_{20}Cr_{20}$ $Y_{22}Cb_{22}Cr_{22}$ $Y_{23}Cb_{22}Cr_{22}$
   $Y_{30}Cb_{20}Cr_{20}$ $Y_{31}Cb_{20}Cr_{20}$ $Y_{32}Cb_{22}Cr_{22}$ $Y_{33}Cb_{22}Cr_{22}$

   Надалі компоненти обробляються й зберігаються окремо друг від друга.

Таким чином, в останніх двох випадках ми відразу забрали $\frac{1}{3}$ й $\frac{1}{2}$ інформації відповідно. Вибір того або іншого способу передискретизації впливає на зміну ступеня стиску. Очевидно, що при відсутності передискретизації ступінь стиску погіршиться, а при схемі 4:2:0 буде найбільшою. Якщо зображення не ділиться націло на блоки 4 × 4, то воно доповнюється по безперервності, тобто у

випадку, якщо розмір по вертикалі не ділиться на 4, то додається ще від однієї до трьох рядків, що збігаються з останньої знизу. Аналогічно робиться, якщо розмір по горизонталі не ділиться на 4 – додаються стовпці, що збігаються із самим правим.

3. **Застосування дискретного косинуса-перетворення.** Зображення (точніше, отримані після субдискретизації компоненти) розбивається на блоки 8 × 8; до кожного блоку застосовується дискретне косинус-перетворення (окремо для компонентів Y, Cb і Cr). Якщо зображення не ділиться націло на блоки 8 × 8, то додається відповідна кількість рядків і стовпців по безперервності.
4. **Квантування.** Людське око практично не зауважує зміни у високочастотних складових, отже, коефіцієнти, відповідальні за високі частоти, можна зберігати з меншою точністю. Квантування здійснюється за допомогою множення матриці коефіцієнтів ДКП на так звану **матрицю квантування**:

$$\begin{pmatrix} t11 & \dots & t18 \\ \vdots & \ddots & \vdots \\ t81 & \dots & t88 \end{pmatrix} \otimes \begin{pmatrix} q11 & \dots & q18 \\ \vdots & \ddots & \vdots \\ q81 & \dots & q88 \end{pmatrix} = \begin{pmatrix} T11 & \dots & T18 \\ \vdots & \ddots & \vdots \\ T81 & \dots & T88 \end{pmatrix},$$

де $\otimes$ означає покомпонентне множення й узяття цілої частини, тобто $T_{ij} = [t_{ij}q_{ij}]$, де t_{ij} – вихідні коефіцієнти ДКП, q_{ij} – компоненти матриці квантування, [] – операція узяття цілої частини. Таким чином, відбувається квантування області визначення коефіцієнтів вихідної матриці. Матриці квантування різні для компонентів кольоровості і яскравості.

На даній стадії можна задавати ступінь стиску шляхом зміни матриць квантування. Чим ближче до нуля елементи матриці квантування, тим менше буде діапазон значень елементів матриці T_{ij}, а виходить, їх можна закодувати, затративши меншу кількість інформації. Наприклад, якщо елементи q_{ij} досить близькі до нуля, то більшість елементів T_{ij} буде нулями.

Матриці квантування застережені в стандарті, а для зміни ступеня стиску їх множать на певний коефіцієнт. Очевидно, що втрати на цій стадії самі більші; якщо забрати занадто багато інформації з низькочастотних компонентів (тобто занадто сильно огрубіти компоненти), те з'являться артефакти: розпадання на квадрати 8 × 8, ефект Гиббса (виникнення ореола поруч із місцями різких колірних переходів) (див. рис. 14.3).

5. **Зиґзаґ-впорядкування.** До кожної квантованої матриці застосовується так зване **зиґзаґ-в**порядкування. Це особливий прохід матриці для одержання послідовності (див. рис. 14.4). Спочатку йде елемент T_{00}, потім T_{01}, T_{10}, T_{11} . . . Причому для типових фотореалістичних зображень спочатку будуть іти ненульові коефіцієнти, що відповідають низькочастотним компонентам, а потім – безліч нулів.
6. **Стиск методом RLE.** Отримана послідовність кодується за допомогою модифікованого алгоритму групового кодування. Виводяться пари чисел: перше – число нулів, друге – значення після підпослідовності нулів. Наприклад, закодуємо таку послідовність: 122 0 125 0 0 44 0 0 0 0 -1.

Одержимо: (0, 122) (1, 125) (2, 44) (4, -1). Також існує спеціальний код для позначення того факту, що значення, що залишилися, у послідовності суть нулі.

7. **Стиск** методом **Хаффмена.** Проводиться кодуванням методом Хаффмена зі спеціальною фіксованою таблицею.

Рис. 14.3. Артефакти JPEG. Угорі – вихідне зображення, унизу – зображення, стисле в 30 разів алгоритмом JPEG

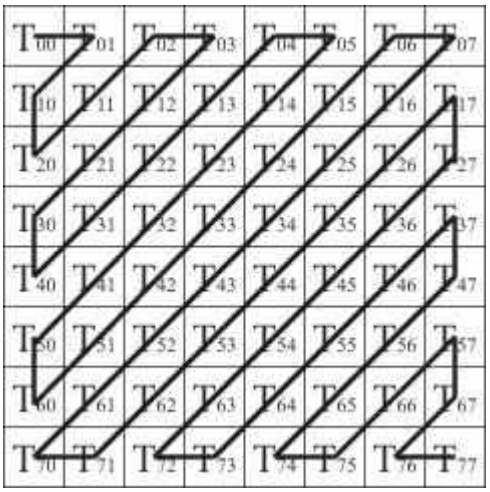


Рис. 14.4. Зигзаг-упорядкування

Алгоритм відновлення зображення суть інверсія вищенаведених дій. Ступінь стиску від 5 до 100 і більше раз (варіюється за допомогою матриць квантування й завдання методу субдискретизації). При цьому візуальна якість для більшості фотореалістичних зображень залишається на гарному рівні при стиску до 15 разів.

Даний алгоритм і формат є найпоширенішими для передачі й зберігання повноколірних зображень. Настільки широке поширення пояснюється декількома причинами: щодо невисокою обчислювальною складністю (що було дуже актуально до 1996 року), достатнім ступенем стандартизованості формату й алгоритму, а також відсутністю необхідності платити які-небудь ліцензійні відрахування (тому що відсутні патентовані алгоритми). До недоліків алгоритму відносять виникнення згадуваних вище артефактів, які занадто помітні для людського ока.

Описана вище схема стиску є типовою для алгоритмів стиску зображень із втратами (за винятком фрактального). Відмінність в основному складається в типі перетворення на кроці 3. Далі ми розглянемо інший вид перетворення.

14.7. Вейвлет-Перетворення

Ідея вейвлет-**перетворення** складається в розкладанні сигналу (функції-зображення) по системі функцій, що мають локальний сплеск і швидко убутних на нескінченності [20]. У цьому й наступному параграфах передбачається знайомство читача з базовими поняттями функціонального аналізу (див., наприклад, [3]). Звичайно такі функції вейвлет-перетворення[2)] виводяться з так званого **материнського вейвлета**. Материнський вейвлет – це функція ψ :

$$\int_{-\infty}^{\infty} |\psi(t)|\,dt < \infty, \int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty,$$

тоді $\psi \in L_2(R) \cap L_1(R)$; також передбачається, що

$$\int_{-\infty}^{\infty} \psi(t)dt = 0, \int_{-\infty}^{\infty} |\psi(t)|^2 dt = 1.$$

Материнський вейвлет перетворюється в такий спосіб:

$$\psi_{ab}(t) = \frac{1}{\sqrt{|a|}}\psi\left(\frac{t-b}{a}\right),$$

де $(a,b) \in R\backslash\{0\} \times R$. Приклади материнських вейвлетів див. на рис. 14.5. У загальному випадку вейвлет-перетворення записується так:

$$W_\psi f(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} \overline{\psi\left(\frac{t-b}{a}\right)} f(t)dt.$$

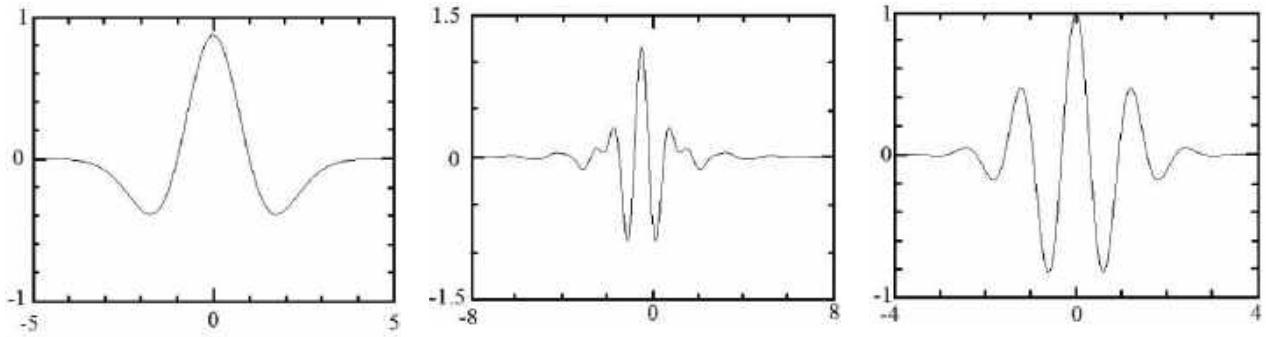


Рис. 14.5. Приклади материнських вейвлетів

При практичному застосуванні вейвлет-перетворення аналіз такого потужного (а саме континуум) безлічі коефіцієнтів неможливий. Тому (a, b) вибираються з рахункової підмножини площини $R\backslash\{0\} \times R$. Звичайно материнський вейвлет і безліч значень (a, b) вибирають так, щоб система $\{\psi_{ab}\}$ утворювала ортонормований базис у просторі $L_2(R)$. Тоді будь-яку функцію f(x) із цього простору можна розкласти по цьому базисі $f(x) = \sum_{j,k} c_{jk}\psi_{jk}(x)$.

Пояснимо як застосовувати вейвлет-перетворення до дискретного сигналу (наприклад, зображенню). Для простоти будемо розглядати одномірний випадок – послідовність кінцевої довжини $\{s_j\}_{j=0}^{N-1}$. Тоді, за умови що материнський вейвлет $\psi \in L_2(R)$, перетворення можна записати так:

$$W_s(a,b) = \frac{1}{\sqrt{a}} \sum_{j=0}^{N-1} s_j \int_j^{j+1} \psi \frac{t-b}{a} dt, a = 1...N, b = 0, ...N-1.$$

Розглянемо найпростішої вейвлет – **вейвлет Хаара** (англ. Haar wavelet). Розглянемо простір $L_2(R)$. У цьому просторі ортонормована наступна система:

$$\chi_{00}(x) = \begin{cases} 1, & x \in \left[0, \frac{1}{2}\right), \\ -1, & x \in \left[\frac{1}{2}, 1\right), \\ 0, & x \notin [0,1). \end{cases}$$

$$\chi_{qm}(x) = 2^{q/2} \chi_{00}(2^q x - m), q \in Z, m \in Z.$$

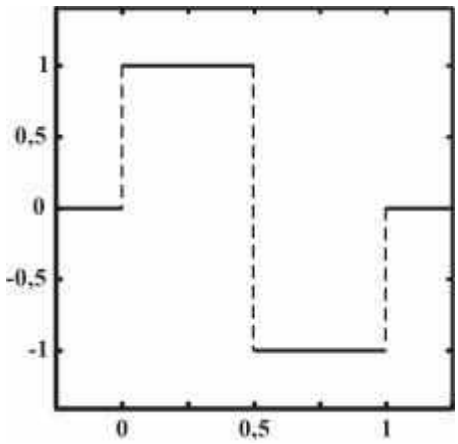


Рис. 14.6. Материнський вейвлет Хаара

Дана система називається **системою Хаара**. Для такого вейвлета материнським є χ_{00} (див. рис. 14.6). У цьому випадку (a, b) обрані із $R \setminus \{0\} \times R$ за таким законом:

$$a = 2^{-i}, \frac{b}{a} = j, i, j \in Z$$

Дія вейвлета Хаара на послідовність, що складається з 2^N елементів, можна розглядати так: елементи групуються по двох, обчислюється сума й різниця кожної пари, різниці зберігаються, а із сум формується послідовність довжини 2^{N-1}; далі перетворення виконується доти, поки не залишиться одна сума й відповідно $2^N - 1$ різниця. Таке перетворення задається так званою **матрицею Хаара**:

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Послідовність $(a_0, \ldots, a_{2N+1})$ представимо так: $((a_0, a_1), \ldots, (a_{2N}, a_{2N+1}))$.

Тепер помножимо праворуч вектори (a_i, a_{i+1}) на матрицю H_2. Одержимо послідовність сум і різниць $((s_0, d_0), \ldots, (s, d))$. Якщо послідовність має довжину, кратну 4, то можна застосувати наступну матрицю Хаара:

$$H_4 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{pmatrix}.$$

Очевидно, що дані перетворення легко оборотні. З таким вейвлет-перетворенням тісно зв'язане так зване S-**Перетворення**. Якщо ми не хочемо, щоб при перетворенні Хаара границі значень послідовності подвоювалися (через підсумовування), треба брати напівсуму. Однак при цілому розподілі на 2 ми втрачаємо точність. Рішення даної проблеми виглядає так: нехай a і b - пари значень із вихідної послідовності. Тоді

$$s = \left[\frac{a+b}{2}\right], d = a - b,$$

при цьому відновлення виглядає так:

$$a = s + \left[\frac{d}{2}\right], b = s - \left[\frac{d}{2}\right],$$

при цьому, якщо d – непарне, то при d > 0 до a додається 1, а якщо d < 0, то до b додається 1.

```
// s - сума, d - різниця
// Round(x) - узяття цілої частини
// IsOdd(x) - чи є непарним
a = s + Round( d / 2 ); b = s - Round( d / 2)
if( IsOdd( d ) )
{
   if ( d > 0)
      a++;
   else
      b++;
}
```

Лістинг 14.1. Алгоритм

В алгоритмі JPEG2000 [10] використовуються так звані вейвлети Добеши (англ. Daubechies wavelet). У матричному виді для дії на вектор A довжини 8 дане перетворення задається так:

$$D_4 = \begin{pmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 \end{pmatrix},$$

де

$$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}, h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}, h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}, h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}},$$
$$g_0 = h_3, g_1 = -h_2, g_2 = h_1, g_3 = -h - 0.$$

Як видно, матриця має розміри 8×10 через необхідність участі в підсумовуванні чотирьох компонентів. Т.е. насправді, така матриця множиться на наступний вектор: $A = (a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10})^T$.

Звичайно думають або $a_9 = a_1$, $a_{10} = a_2$, або $a_9 = a_8$, $a_{10} = a_7$. Використання вейвлет-перетворень при стиску зображень аналогічно використанню дискретний косинус-перетворення в алгоритмі JPEG, тобто саме перетворення – лише щабель конвеєра стиску.

Вейвлет-Перетворення мають дуже гарну частотно-просторову локалізацію й по цьому показнику перевершують традиційні косинус-перетворення й інші перетворення Фур'є. Таким чином, стає можливо застосовувати більше сильне квантування, поліпшуючи властивості послідовності для наступного стиску без втрат. Алгоритми стиску зображень, засновані на цьому перетворенні, при тім же ступені стиску показують кращі результати по збереженню якості зображення. До того ж обчислювальна складність дуже низька й становить O(N) (тут N – довжина послідовності, до якої застосовується перетворення). Однак поширеність форматів, що використовують стиск на основі вейвлет-перетворення, невелика через наявність патентів, а також через велику поширеність звичайного JPEG, що дає цілком прийнятні результати.

14.8. Фрактальний стиск

Фрактальний стиск ґрунтується на пошуку й кодуванні самоподібних областей у зображенні. При цьому використовуються **системи ітеруємих функцій** (англ. IFS – Iterated Function System) [31], [12].

Визначення 14.6.1. Відображення $f: R \to R$ в повному метричному просторі (R, ρ) називається **стискаючої**, якщо $\exists \alpha < 1 : \forall x, y \in R \rho(f(x), f(y)) < \alpha \rho(x, y)$.

Система зі n стискаючих відображень $F = f_1, ..., f_n$, що діють у повному метричному просторі (R, ρ), ставиться у відповідність **оператор системи ітеруємих функцій**

$$F^*(X) = \bigcup_{i=1}^{n} f_i(X),$$

де $X \subseteq R, f(X) = \{f(x) | x \in X\}$. Таке відображення є стискаючим з коефіцієнтом $\alpha = \max_{1 \le i \le n} \alpha_i$.

По теоремі про стискаючі відображення [3] існує нерухлива безліч $S: F^*(S) = S$.

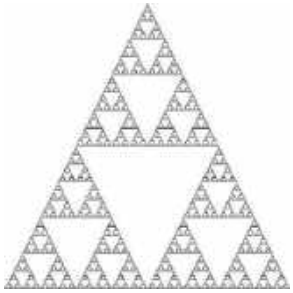


Рис. 14.7. Трикутник Серпинского

Розглянемо приклад дії системи ітеруємих функцій. Нехай

$f_1(x,y) = \left(\frac{1}{2}x, \frac{1}{2}y\right), f_2(x,y) = \left(\frac{1}{2}x + \frac{1}{2}, \frac{1}{2}y\right),$

$f_3(x,y) = \left(\frac{1}{2}x + \frac{1}{4}, \frac{1}{2}y + \frac{\sqrt{3}}{4}\right),$

діють у просторі $R = [0; 1]^2$ з евклідовою метрикою. Дані відображення є стискаючими $(\alpha_i = \frac{1}{2})$, і їхні нерухливі крапки є вершини рівностороннього трикутника (їхньої координати $(0,0), (1,0), (\frac{1}{2}, \frac{\sqrt{3}}{2})$ відповідно). Спочатку будемо послідовно застосовувати оператор F^* даних перетворень до безлічі, обмеженій рівностороннім трикутником (наприклад, з координатами вершин, зазначеними вище). Нерухлива безліч S, одержуване при дії оператора даної системи ітеруємих функцій – це так званий трикутник Серпинского (див. рис. 14.7). Процес побудови трикутника Серпинского див. на рис. 14.8. Можна почати застосовувати оператор даних перетворень не до трикутника, а до квадрата. Нерухлива безліч буде таким же (див. рис. 14.9). Неважливо, з якої області починати (потрібно тільки компактність), – одержимо трикутник Серпинского.

Іншим прикладом нерухливої безлічі S для оператора F^* для деякої системи F із чотирьох афінних перетворень є папороть Барнслі (див. рис. 14.10).

Рис. 14.8. Побудова трикутника Серпинского

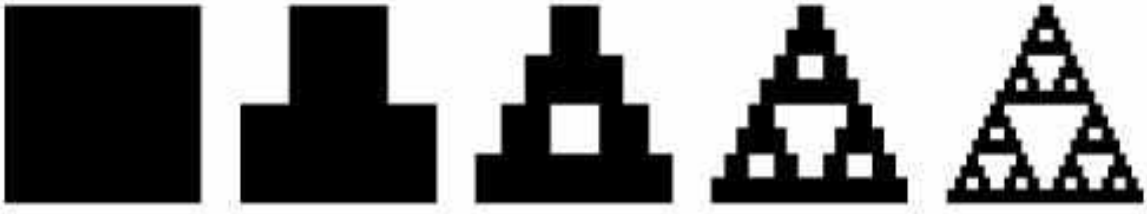


Рис. 14.9. Побудова трикутника Серпинского із квадрата

Зображення папороті може займати (у гарному дозволі), а його опис за допомогою параметрів афінних перетворення – кілька сотень байт! Відповідно, ідея стиску складається в пошуку таких систем перетворень, які б у процесі ітерування наближали бажане зображення. Завдання ставиться так: по даній безлічі S знайти систему ітеруємих функцій F так, що нерухлива безліч системи досить добре наближає S.

Складність такого завдання в загальному випадку дуже висока. Однак, якщо розглядати зображення як функцію (див. введення роздягнула 14.3) і обмежувати вид перетворень у такий спосіб:

$$f(x,y,I(x,y)) = \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & L \end{pmatrix} \begin{pmatrix} x \\ y \\ I(x,y) \end{pmatrix} + \begin{pmatrix} u \\ v \\ Q \end{pmatrix},$$

тобто афінними перетвореннями, то завдання спрощується. Проте час роботи, необхідне для пошуку коефіцієнтів перетворення, занадто велико. Для прискорення алгоритму пошуку коефіцієнтів накладають спеціальні обмеження на вид і розмір безлічей, до яких застосовуються перетворення, а також на сам вид перетворень, тобто фіксована безліч кутів повороту й коефіцієнтів масштабування. Таким чином, завдання стає дискретної й можливий прямий підбор параметрів.

Рис. 14.10. Папороть Барнслі

У реальному житті зображення, подібні до папороті Барнслі (тобто складаються тільки із самоподоби), зустрічаються рідко, тому очікувати схожого ступеня стиску не можна. Проте якість відновлених зображень при високих ступенях стиску значно перевершує JPEG. До недоліків алгоритму відносять дуже довгий час стиску в порівнянні з іншими методами, а також наявність патентів.

14.9. Список літератури

Хвальків Н.С., Жидков Е.П., Кобельков Г.М **Чисельні методи. Навчальний посібник. – 4-е видання** М. – Спб.: Физматлит, Невський діалект, Лабораторія базових знань, 2003.

8. Ватолин Д.З **Алгоритми стиску зображень** Видавничий відділ ВМиК МГУ ім. М.В. Ломоносова, 1999.

9. Колмогоров А.Н., Фомін С.В **Елементи теорії функцій і функціонального аналізу** М.: Наука, 1976.
10. Кристофидес Н **Теорія графів, алгоритмічний підхід** Мир, 1978.
11. Липкин И.А **Основи статистичної радіотехніки, теорія інформації й кодування** Сов.радіо, 1978.
12. Липский У **Комбінаторика для програмістів** Мир, 1988.
13. Препарату Ф., Шеймос М **Обчислювальна геометрія. Введення** Мир, 1989.
14. Роджерс Д., Адамс Д **Математичні основи машинної графіки** Мир, 2001.
15. Ярославський Л.П **Введення в цифрову обробку зображень** М.: Сов. радіо, 1979.
16. Adams M. D **The JPEG-2000 still image compression standard** 2000.
17. **Adobe Systems Inc. PostScript language reference. – 3rd edition** Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.
18. Barnsley M.F., Sloan A.D. **A better way to compress images** BYTE. JPEG – 1988. – 2000 Pp. 215 – 223.
19. Bayer B.E. **An optimum method for two level rendition of continuous-tone pictures** IEEE International Conference on Communication s, Conference Record. – Seattle,Washington, USA: 1973. – June 11 – 13. – Pp. 11 – 15.
20. Bezier P.E. **Emploi des machines a commande numerique** Masson et Cie., 1970.
21. Boykov Y., Jolly M.P. **Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images** Proc. IEEE Int. Conf. on Computer Vision. – Vol. 1. – Vancouver, Canada: 2001. – Pp. 105 – 112.
22. Bresenham J. **Algorithm for computer control of a digital plotter** IBM Systems Journal. – 1965. – Vol. 4, no. 1. – Pp. 25 – 30.
23. Bresenham J. **A linear algorithm for incremental digital display of circular arcs** Commun. ACM. – 1977. – Vol. 20, no. 2. – Pp. 100 – 106.
24. Canny J.**A computational approach to edge detection** IEEE Trans. Pattern Anal. Mach. Intell. – 1986. – Vol. 8, no. 6. – Pp. 679 – 698.
25. Castle C.M.A., Pitteway M.L. **An effcient structural technique for encoding best-fit straight lines** Comput. J. – 1987. – Vol. 30, no. 2. – Pp. 168 – 175.
26. Chui C.K. **An introduction to wavelets** San Diego, CA, USA: Academic Press Professional, Inc., 1992.
27. **CIE. Colorimetry. official recommendation of the international commission on illumination: Tech. Rep. 15.2** Vienna, Austria: Bureau Central de la CIE, 1986.
28. Cohen E., Lyche T., Riesenfeld R.F. **Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics** Computer Graphics and Image Processing. – 1980. – October. – Vol. 14, no. 2. – Pp. 87 – 111.
29. M. de Berg, M. van Kreveld, M. Overmars, O. Schwartzkopf. **Computational Geometry: Algorithms and Applications** Springer, 2000.
30. J. Foley, A. van Dam, S.K. Feiner, J.F. Hughes. **Computer Graphics. Principles and Practice. 2nd edition in C Addison-**Wesley, 1996.

31. Cyrus M., Beck J. **Generalized two- and three-dimensional clipping** Computers and Graphics. – 1978. – Vol. 3, no. 1. – Pp. 23 – 28.
32. de Casteljau P. **Outillages methodes calcul** Tech. rep. A. Citroen, Paris, 1959.
33. Diday E. **The dynamic clusters method in nonhierarchical clustering** International Journal of Computer and Information Sciences. – 1973. – March. – Vol. 2, no. 1. – Pp. 61 – 88.
34. Farin G. **Curves and surfaces for CAGD: a practical guide** San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.
35. Feibush E.A., Levoy M., Cook R.L. **Synthetic texturing using digital filters** SIGGRAPH 80. – New York, NY, USA: ACM Press, 1980. – Pp. 294 – 301.
36. Floyd R.W., Steinberg L. **An adaptive algorithm for spatial gray-scale** Proceedings Society Information Display. – 1976. – Vol. 17, no. 2. – Pp. 75 – 78.
37. Y. Fisher. **Fractal Image Compression: Theory and Application** New York, NY, USA: Springer Verlag, 1995.
38. Gonzalez R.C., Woods R.E. **Digital Image Processing. – 3rd edition** Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.
39. Guild J. **The colorimetric properties of the spectrum** Philosophical Trans. Royal Soc. London. – 1931. – Vol. A230. – Pp. 149 – 187.
40. Gupta S., Sproull R.F. **Filtering edges for gray-scale displays** SIGGRAPH 81. – New York, NY, USA: ACM Press, 1981. – Pp. 1 – 5.
41. Heckbert P. **Color image quantization for frame buffer display** SIGGRAPH 82. – New York, NY, USA: ACM Press, 1982. – Pp. 297 –307.
42. Huffman D. **A method for the construction of minimum-redundancy codes**.

Тема 15. АЛГОРИТМИ СТИСКУ ВІДЕО

В даній лекції мова ітиме про алгоритми стиску відео. Буде даний докладний перелік специфічних вимог, пропонованих до цих алгоритмів. У лекції докладно розглянутий алгоритм стиску MPEG, від історії його створення до послідовності кроків, виконуваних при стиску зображення

15.1. Введення

Основною складністю при роботі з відео є більші обсяги дискового простору, необхідного для зберігання навіть невеликих фрагментів. Причому навіть застосування сучасних алгоритмів стиску не змінює ситуацію кардинально. При записі на один компакт-диск "у побутовій якості", на нього можна помістити кілька тисяч фотографій, приблизно 10 годин музики й усього півгодини відео. Відео "телевізійного" формату 720x576 пікселів 25 кадрів у секунду в системі RGB вимагає потоку даних приблизно в 240 Мбіт/сек (тобто 1.8 Гб у мінуту). При цьому традиційні алгоритми стиску зображень, орієнтовані на окремі кадри, не рятують ситуації, оскільки навіть при зменшенні потоку в 10 разів він становить досить більші величини.

У результаті переважна більшість сьогоднішніх алгоритмів стиску відео є алгоритмами із втратою даних. При стиску використовується кілька типів надмірності:

– **Когерентність областей зображення** – мала зміна кольору зображення в сусідніх пікселях (властивість, що експлуатують всі алгоритми стиску зображень із втратами).

– **Надмірність у колірних площинах** – використовується більша важливість яскравості зображення для сприйняття.

– **Подоба між кадрами** – використання того факту, що на швидкості 25 кадрів у секунду, як правило, сусідні кадри змінюються незначно.

Перші два пункти знайомі вам по алгоритмах стиску графіки. Використання подоби між кадрами в найпростішому й найбільше часто використовуваному випадку означає кодування не самого нового кадру, а його різниці з попереднім кадром. Для відео типу "мовець голова" (передача новин, відеотелефони), більша частина кадру залишається незмінної, і навіть такий простий метод дозволяє значно зменшити потік даних. Більше складний метод полягає в знаходженні для кожного блоку в стисливому кадрі найменш отличающегося від нього блоку в кадрі, використовуваному в якості базового. Далі кодується різниця між цими блоками. Цей метод істотно більше ресурсномісткий.

15.2. Основні поняття

Визначимося з основними поняттями, які використовуються при стиску відео. Відеопотік характеризується **дозволом, частотою кадрів і системою подання кольорів**. З телевізійних стандартів прийшли розширення в 720x576 і 640x480, і частоти в 25 (стандарти **PAL** або **SECAM**) і 30 (стандарт **NTSC**) кадрів у секунду. Для низьких дозволів існують спеціальні назви **CIF – Common Interchange Format**, рівний 352x288 і **QCIF – Quartered Common**

Interchange Format, рівний 176x144. Оскільки **CIF** і **QCIF** орієнтовані на вкрай невеликі потоки, то з ними працюють на частотах від 5 до 30 кадрів у секунду.

15.3. Вимоги додатків до алгоритму

Для алгоритмів стиску відео характерна більшість тих же вимог додатків, які пред'являються до алгоритмів стиску графіки, однак є й певна специфіка:

– **Довільний доступ** – має на увазі можливість знайти й показати будь-який кадр за обмежений час. Забезпечується наявністю в потоці даних так званих крапок входу – кадрів, стислих незалежно (тобто як звичайне статичне зображення). Прийнятним часом пошуку довільного кадру вважається 1/2 секунди.

– **Швидкий пошук уперед/назад** – має на увазі швидкий показ кадрів, що не випливають один за одним у вихідному потоці. Вимагає наявності додаткової інформації в потоці. Ця можливість активно використовується всілякими програвачами.

– **Показ кадрів фільму у зворотному напрямку.** Рідко потрібно в додатках. При твердих обмеженнях на час показу чергового кадру виконання цієї вимоги може різко зменшити ступінь стиску.

– **Аудио-Візуальна синхронізація** – сама серйозна вимога. Дані, необхідні для того, щоб домогтися синхронності аудіо й відео доріжок, істотно збільшують розмір фільму. Для відеосистеми це означає, що, якщо ми не встигаємо дістати й показати в потрібний момент часу якийсь кадр, то ми повинні вміти коректно показати, наприклад, кадр, що випливає за ним. Якщо ми показуємо фільм без звуку, то можна дозволити собі ледве більше повільний або більше швидкий показ. У часи порівняно недосконалого німого кіно кадри йшли настільки нерівномірно, наскільки нерівномірно крутив ручку камери оператор. Показ без звуку фільму, знятого настільки недосконалими методами, сприймається нормально навіть за умови, що частота показуваних кадрів постійна (і герої фільму пересуваються карикатурно швидко, та повільно). Однак дивитися фільм (наприклад, бойовик), у якому відеосистема не встигає за звуком - стає мученням.

– **Стійкість до помилок** – вимога, обумовлена тим, що більшість каналів зв'язки ненадійні. Зіпсоване перешкодою зображення повинне швидко відновлюватися. Вимога досить легко задовольняється необхідним числом незалежних кадрів у потоці. При цьому також зменшується ступінь стиску, тому що на екрані 2-3 секунди (50-75 кадрів) може бути те саме зображення, але ми будемо змушені навантажувати потік незалежними кадрами.

– **Час кодування/декодування.** У багатьох системах (наприклад, відеотелефонах) загальна затримка на кодування-передачу-декодування повинна становити не більше 150 мс. Крім того, у додатках, де необхідне редагування, нормальна інтерактивна робота неможлива, якщо час реакції системи становить більше 1 секунди.

– **Редагованість.** Під редагованістю розуміється можливість змінювати всі кадри так само легко, як якби вони були записані незалежно.

– **Масштабованість** – простота реалізації концепції "відео у вікні". Ми повинні вміти швидко змінювати висоту й ширину зображення в пікселях. Масштабування здатне породити неприємні ефекти в алгоритмах заснованих на ДКП (дискретному косинусному перетворенні). Коректно реалізувати цю можливість для MPEG на даний момент можна, мабуть, лише при досить складних апаратних реалізаціях, тільки тоді алгоритми масштабування не будуть істотно збільшувати час декодування. Цікаво, що масштабування досить легко здійснюється в так званих фрактальних алгоритмах. У них, навіть при збільшенні зображення в кілька разів, воно не розпадається на квадрати, тобто відсутній ефект "зернистості". Якщо необхідно зменшувати зображення (що, хоч і рідко, але буває потрібно), то з таким завданням добре справляються алгоритми, засновані на wavelet перетворенні (див. опис JPEG-2000).

– **Невелика вартість апаратної реалізації.** При розробці хоча б приблизно повинна оцінюватися й ураховуватися кінцева вартість. Якщо ця вартість велика, то навіть при використанні алгоритму в міжнародних стандартах, виробники будуть пропонувати свої, більше конкурентоспроможні, алгоритми й рішення. На практиці ця вимога означає, що алгоритм повинен реалізовуватися невеликим набором мікросхем.

Вправа: Покажіть, що вимоги довільного доступу, швидкого пошуку, показу у зворотному напрямку, аудіо-візуальній синхронізації й стійкості до помилок суперечать умові високого ступеня стиску потоку.

Описані вимоги до алгоритму суперечливі. Очевидно, що високий ступінь стиску має на увазі архівацію кожного наступного кадру з використанням попередні. У той же час вимоги на аудіо-візуальну синхронізацію й довільний доступ до будь-якого кадру за обмежений час не дають можливості витягнути всі кадри в ланцюжок. І, проте, можна спробувати прийти до деякого компромісу. Збалансована реалізація, що враховує систему суперечливих вимог, може досягатися на практиці за рахунок настроювань компресора при стиску конкретного фільму.

15.4. Визначення вимог

Під процедурою визначення вимог, пропонованих до алгоритму, розуміється з'ясування класів програмного й апаратного забезпечення, на які він орієнтований і, відповідно, виробіток вимог до нього.

Носії інформації, на які орієнтований алгоритм:

– DVD-ROM – порівняно низька вартість, дуже висока щільність запису інформації роблять його найбільш перспективним пристроєм для зберігання оцифрованого відео.

– CD-ROM – низька вартість при високій щільності запису інформації роблять його найбільш привабливим пристроєм для зберігання оцифрованого відео. До недоліків ставиться порівняно малий обсяг, однак диск має рекордно низьке відношення вартості диска до обсягу.

– Жорсткий диск – найбільш швидкий і гнучкий пристрій, що володіє дуже малим часом пошуку, що необхідно для деяких додатків. Однак він має високе співвідношення вартості диска до обсягу.
– Перезаписувані оптичні диски – один з найбільш перспективних пристроїв, здатних сполучити в собі переваги CD-ROM (низька собівартість зберігання інформації, великий обсяг, довільний доступ) і жорсткого диска (можливість перезапису).
– Комп'ютерні мережі (як глобальні, так і локальні). Характеризуються можливістю швидко одержувати практично необмежені обсяги інформації. До недоліків мереж, з якими борються так звані технології Qo (Quality of Service – гарантована якість сервісу), ставляться можливі затримки пакетів, і довільна зміна пропускної здатності каналу.

Програмне забезпечення, що використовує відео-компресію, можна підрозділити на дві групи – **симетричне** й **асиметричне.**

Асиметричні додатки висувають серйозні вимоги до декодера (як правило, за часом і пам'яттю), але для них байдужні витрати ресурсів при кодуванні. Прикладом є різні мультимедіа енциклопедії, путівники, довідники, ігри й просто фільми. При такій постановці завдання з'являється можливість застосувати складні алгоритми компресії, що дозволяють одержати більший ступінь стиску даних.

Симетричні додатки пред'являють однаково тверді вимоги на час, пам'ять і інші ресурси, як при кодуванні, так і при декодуванні. Прикладами такого роду додатків можуть служити відеопошта, відеотелефон, відеоконференція, редагування й підготовка відеоматеріалів.

15.5. Огляд стандартів

В 1988 році в рамках Міжнародної Організації по Стандартизації (ISO) почала роботу група MPEG (Moving Pictures Experts Group) – група експертів в області цифрового відео (ISO-IEC/JTC1/SC2/WG11/MPEG). Група працювала в напрямках, які можна умовно назвати MPEG-Video – стиск відеосигналу в потік зі швидкістю до 1,5 Мбіт/сек, MPEG-Audio – стиск звуку до 64, 128 або 192 Кбіт/сек на канал і MPEG-System – синхронізація відео й аудио потоків [15.1]. Нас в основному будуть цікавити досягнення MPEG-Video, хоча очевидно, що багато рішень у цьому напрямку приймалися з урахуванням вимог синхронізації.

Як алгоритм, MPEG має трохи попередників. Це, насамперед, універсальний алгоритм JPEG. Його універсальність означає, що JPEG показує непогані результати на широкому класі зображень.

Якщо бути більше точним, то стандарт MPEG, як і інші стандарти на стиск, описує лише вихідний бітовий потік, неявно задаючи алгоритми кодування й декодування. При цьому їхня реалізація перекладається на програмістів-розроблювачів. Такий підхід відкриває широкі обрії для тих, хто бажає оптимально реалізувати алгоритм для конкретного обчислювального пристрою (контролера, ПК, розподіленої обчислювальної системи), операційної системи, відеокарти й т.п. [15.2]. При спеціалізованих реалізаціях можуть бути

враховані досить специфічні вимоги на час роботи, витрата пам'яті і якість одержуваних зображень. Алгоритми стиску відео досить гнучкі й найчастіше для різних підходів до реалізації, можна одержати істотну різницю по якості відео, при одній і тім же ступені стиску. Більше того - для того самого стислого файлу за допомогою різних алгоритмів декодування можна одержати істотно, що розрізняються по візуальній якості фільми. Найчастіше "проста" реалізація стандарту дає відеоряд, що смикає, з добре помітними блоками, у той час як програми відомих виробників програють цей же файл цілком плавно й без кидаючої в очі блочності. Ці нюанси необхідно добре собі представляти, коли мова заходить про порівняння різних стандартів.

У вересні 1990 був представлений попередній стандарт кодування MPEG-1. У січні 1992 робота над MPEG-1 була завершена, і почата робота над MPEG-2, у завдання якого входив опис потоку даних зі швидкістю від 3 до 10 Мбіт/сек [15.3]. Практично в той же час була почата робота над MPEG-3, що був призначений для опису потоків 20-40 Мбіт/сек. Однак незабаром з'ясувалося, що алгоритмічні рішення для MPEG-2 і MPEG-3 принципово близькі й можна безболісно розширити рамки MPEG-2 до потоків в 40 Мбіт/сек. У результаті робота над MPEG-3 була припинена.MPEG-2 був остаточно дороблений до 1995 року.

В 1991 групою експертів по відеотелефонах (EGVT) при Міжнародному консультативний комітет з телефонії й телеграфії (CCITT) запропонований стандарт відеотелефонів px64 Kbits. Запис px64 означає, що алгоритм орієнтований на паралельну передачу оцифрованого відеозображення по p каналах із пропускною здатністю 64 Кбіта/сек. Таким чином, захоплюючи кілька телефонних ліній, можна одержувати зображення цілком прийнятної якості. Одним з головних обмежень при створенні алгоритму був час затримки, що повинне було становити не більше 150 мс. Крім того, рівень перешкод у телефонних каналах досить високий, і це, природно, знайшло відбиття в алгоритмі. Можна вважати, що px64 Kbits – попередник MPEG-A для потоків даних менш 1,5 Мбіт/сек і специфічного класу відео.

У групі при CMTT (спільний комітет при CCITT і CCIR – International Consultative Committee on bRoadcasting) роботи були спрямовані на передачу оцифрованного відео по виділених каналах з високою пропускною здатністю й радіолініям. Відповідні стандарти H21 і H22 орієнтовані на 34 і 45 Мбіт/сек, і сигнал передається з дуже високою якістю.

MPEG-4 споконвічно був задуманий як стандарт для роботи з наднизькими потоками. Однак у процесі досить довгої підготовки стандарт перетерпів зовсім революційні зміни й зараз властиво стиск із низьким потоком входить у нього як одна складова частина, причому досить невелика по розмірі. Наприклад, сам формат сьогодні містить у собі такі речі, як синтез мови, рендеринг зображень і опису параметрів візуалізації особи на стороні програми перегляду.

Розробка MPEG-7 була почата в 1996. Властиво до алгоритмів стиску відео цей стандарт має ще менше відношення, чим MPEG-4, оскільки його основне завдання полягає в описі контента й керуванні ім. Опис MPEG-7 виходить за рамки це книги.

Паралельно все це час існували формати Motion-JPEG і недавно з'явився Motion-JPEG2000, призначені в основному для зручності обробки стислого відео. Розглянемо основні стандарти й лежачі в їхній основі алгоритми детальніше.

15.6. Базові технології стиску відео

15.6.1. Опис алгоритму компресії

Технологія стиску відео в MPEG розпадається на дві частини: зменшення надмірності відеоінформації в тимчасовому вимірі, заснований на тім, що сусідні кадри, як правило, відрізняються не сильно, і стиск окремих зображень.

Для того щоб задовольнити суперечливим вимогам і збільшити гнучкість алгоритму, розглядається чотири типи кадрів:

– I-Кадри – кадри стислі незалежно від інших кадрів (I-Intra pictures),
– P-Кадри – стислі з використанням посилання на одне зображення (P-Predicted),
– B-Кадри – стислі з використанням посилання на два зображення (B-Bidirection),
– DC-Кадри – незалежно стислі з великою втратою якості (використовуються тільки при швидкому пошуку).

I-Кадри забезпечують можливість довільного доступу до будь-якого кадру, будучи своєрідними вхідними крапками в потік даних для декодера. P-Кадри використовують при архівації посилання на один I-I- або P-кадр, підвищуючи тим самим ступінь стиску фільму в цілому. B-Кадри, використовуючи посилання на два кадри, що перебувають спереду й за, забезпечують найвищий ступінь стиску. Самі як посилання використовуватися не можуть. Послідовність кадрів у фільмі може бути, наприклад, такий: IBBPBBPBBPBBIBBPBB... Або, якщо ми не заощаджуємо на ступені стиску, такий (рис. 15.1):

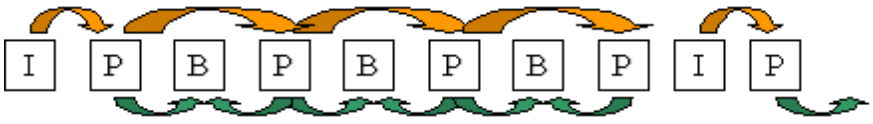


Рис. 15.1. I-Кадри – незалежно стислі (I-Intrapictures), P-Кадри – стислі з використанням посилання на одне зображення (P-Predicted), B-Кадри – стислі з використанням посилання на два зображення (B-Bidirection)

Частота I-Кадрів вибирається залежно від вимог на час довільного доступу й надійності потоку при передачі через канал з помилками. Співвідношення P-P- і B-кадрів підбирається, виходячи з вимог до величини компресії й обмежень декодера. Як правило, декодування B-Кадрів вимагає більше обчислювальних потужностей, однак дозволяє підвищити ступінь стиску. Саме варіювання частоти кадрів різних типів забезпечує алгоритму необхідну гнучкість і можливість розширення. Зрозуміло, що для того, щоб розпакувати B-Кадр, ми

повинні вже розпакувати ті кадри, на які він посилається. Тому для послідовності IBBPBBPBBPBBIBBPBB кадри у фільмі будуть записані так: 0**312645..., де цифри – номера кадрів, а зірочкам відповідають або B-Кадри з номерами -1 і -2, якщо ми перебуваємо в середині потоку, або порожні кадри (нічого), якщо ми на початку фільму. Подібний формат має досить велику гнучкість і здатний задовольняти всіляким наборам вимог.

Одним з основних понять при стиску декількох зображень є поняття макроблоку. При стиску кадр із колірного простору RGB переводиться в колірний простір YUV. Кожна із площин стисливого зображення (Y, U, V) розділяється на блоки 8x8, з якими працює ДКП. Причому площини U і V, що відповідають компоненту кольоровості беруться з дозволом у два рази меншим (по вертикалі й горизонталі), чим вихідне зображення. Таким чином, ми відразу одержуємо стиск у два рази, користуючись тим, що око людини гірше розрізняє колір окремої крапки зображення, чим її яскравість (докладніше про ці перетворення дивитеся в описі алгоритму JPEG). Блоки 8x8 групуються в макроблоки. Макроблок – це група із чотирьох сусідніх блоків у площині яркої компоненти Y (матриця пікселів 16x16 елементів) і два відповідних їм по розташуванню блоку із площин кольоровості U і V. Таким чином, кадр розбивається на незалежні одиниці, що несуть повну інформацію про частину зображення. При цьому розмір зображення повинен бути кратний 16.

Окремі макроблоки стискуються незалежно, тобто в B-Кадрах ми можемо стиснути макроблок конкретний як I-Блок, P-блок з посиланням на попередній кадр, P-Блок з посиланням на наступний кадр і, нарешті, як B-Блок.

Алгоритм стиску окремих кадрів в MPEG схожий на відповідний алгоритм для статичних зображень – JPEG. Якщо говорити коротко, то сам алгоритм стиску являє собою конвеєр перетворень. Це дискретне косинусне перетворення вихідної матриці 8x8, квантування матриці й витягування її у вектор v11,v12,v21,v31,v22,...,v88 (зигзаг-сканування), стиск вектора груповим кодуванням і, нарешті, стиск по алгоритму Хаффмана.

15.6.2. Загальна схема алгоритму

У цілому весь конвеєр перетворень можна представити так:

– Підготовка макроблоків. Для кожного макроблоку визначається, яким образом він буде стислий. В I-Кадрах всі макроблоки стискуються незалежно. В P-Кадрах блок або стискується незалежно, або являє собою різниця з одному з макроблоків у попередньому опорному кадрі, на якому посилається P-Кадр.

– Переклад макроблоку в колірний простір YUV. Одержання потрібної кількості матриць 8x8.

– Для P-Блоків і B-блоків виробляється обчислення різниці з відповідним макроблоком в опорному кадрі.

– ДКП.

– Квантування.

– Зигзаг-сканування.

– Групове кодування.

– Кодування Хаффмана.

При декодуванні весь конвеєр повторюється для зворотних перетворень, починаючи з кінця.

15.6.3. Використання векторів зсувів блоків

Найпростіший спосіб ураховувати подобу сусідніх кадрів – це віднімати кожний блок стисливого кадру з відповідного блоку попереднього. Однак більше гнучким є алгоритм пошуку векторів, на які зрушилися блоки поточного кадру стосовно попередній. Для кожного блоку в зображенні ми знаходимо блок близький по деякій метриці (наприклад, по сумі квадратів різниці пікселів) у попередньому кадрі в деякій околиці поточного положення блоку. Якщо мінімальна відстань по обраній метриці із блоками в попередньому кадрі більше обраного порога – блок стискується незалежно (рис. 15.2).

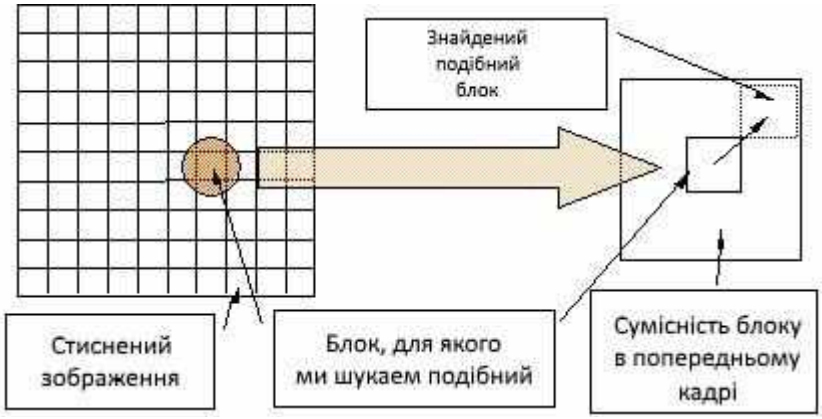


Рис. 15.2. Схема стиску блоків

Таким чином, разом з кожним блоком у потік тепер зберігаються координати зсуву максимально схожого блоку в попередньому I-I- або P-кадрі, або ознака того, що дані стислі незалежно. Ці координати задають вектор зсуву блоку (motion vector). У ситуаціях, коли камера наїжджає на об'єкт або дає панораму, використання векторів зсувів блоків дозволяє значно зменшити амплітуду різниці кадрів, і як наслідок – значно підняти ступінь стиску.

Якщо ми проаналізуємо реальні фільми, то виявиться, що часто блок зрушується не на кратне число пікселів, а, наприклад, на 10.4 пікселя (камера швидко рухається вправо, план зйомки зрушується рівномірно й проходить повний кадр розміром 352x240 за 1.35 секунди). При цьому виявляється, що для підвищення ступеня стиску вигідно будувати 4 області пошуку векторів зсувів: вихідну, зрушену на півпікселя по горизонталі, зрушену на півпікселя по вертикалі й зрушену на півпікселя по горизонталі й по вертикалі (по діагоналі), які будуються за допомогою досить швидких алгоритмів білінійної або кусочно-лінійної апроксимації. Цей прийом також дозволяє зменшити різниця між блоками й підвищити ступінь стиску при мінімальній додатковій інформації, яку треба зберігати у файл (плюс 2 біти на кожний блок). Правда будувати апроксимовані блоки прийде й при декомпресії, однак це порівняно

дешева за часом операція, що досить незначно збільшує загальний час декомпресії.

Також треба розуміти, що алгоритм пошуку оптимальних векторів зсуву полягає, загалом кажучи, у переборі. Існують різні методи зменшення цього перебору, і настроювання відео-кодеків, що регулюють швидкість стиску нерідко варіюють саме параметри методу перебору.

15.6.4. Можливості по розпаралелюваності

Навіть швидкий погляд на цей узагальнений алгоритм дозволяє помітити, що він порівняно легко розпаралелюються. Зображення 320x288 містить 330 макроблоків, які можна кодувати й декодувати незалежно. Кожний макроблок, у свою чергу, містить шість блоків даних для ДКП. Розпаралелити ДКП дуже важливо, тому що, не вважаючи пошуку векторів зсуву, це сама повільна операція. Помітимо також, що інші перетворення легко конвеєризуються. У результаті ми одержуємо паралельно-конвеєрну схему обробки потоку відеоданих.

Досить заманливо виглядає можливість Розпаралелити обробку різних кадрів, але тут ми зіштовхуємося зі складностями. Як правило, компресор будується таким чином, щоб після стиску зображення піддавалося зворотним перетворенням. Таким чином, ми одержуємо кадр із втратами й архівуємо інші кадри, відштовхуючись від нього. Це дозволяє не накопичувати помилки, одержувані ще при квантуванні. Таким чином, якщо на екрані між кадрами спостерігалися більші зміни, і якість зображення довелася понизити, то при стабілізації зображення якість швидко підвищується практично до якості вихідного відеоряду. Неприємний ефект, породжуваний цим прийомом, полягає в тім, що з'являється мерехтіння окремих крапок (або областей) зображення, значення кольору в які округляється те в більшу, то в меншу сторону.

При розпакуванні наші можливості по паралельній обробці різних кадрів досить обмежені, оскільки велико залежність між кадрами в потоці (великий відсоток P-P- і B-кадрів).

15.7. Інші шляхи підвищення ступеня стиску

Описаний вище алгоритм у цілому вкрай близький більшості застосовуваних зараз на практиці алгоритмам стиску відео. Однак нові (або добре забуті старі) ідеї з'являються щорічно. Якщо для алгоритмів стиску без втрат можна говорити про ріст ступеня стиску на 1% у рік (щодо попереднього року) для досить великого тестового масиву даних, то для алгоритмів стиску відео мова звичайно йде об 3-5% збільшення ступеня стиску для досить великого відеофрагмента при тій же візуальній якості.

Однак старе добре "правило важеля", винайдене ще Архімедом, дотримується й тут. І якщо з однієї сторони підвищується ступінь стиску, то з іншої сторони доводиться пройти, додаючи ту ж силу, набагато більший шлях. У цьому випадку росте складність програми й упаде швидкість роботи, як при компресії, так і при декомпресії.

Перелічимо основні шляхи підвищення ступеня стиску:

– **Зміна алгоритму стиску I-Кадрів.** Вище наведений алгоритм, заснована на ДКП. Сьогодні всі частіше використовуються алгоритми, засновані на вейвлетах (див. опис JPEG-2000).
– **Зміна алгоритму стиску без втрат.** Вище наведений алгоритм, що використовує стиск по алгоритму Хаффмана. Однак недавно закінчився основний патент на арифметичний стиск (даючи перевагу 2-15%) і, відповідно, всі частіше використовується саме воно.
– **Зміна алгоритму роботи з векторами зсуву блоків.** Підбор векторів зсувів блоків по найменшому середньоквадратичному зсуві не є оптимальним. Існують алгоритми даючи кращий результат при деяких додаткових витратах часу при стиску.
– **Застосування обробки коефіцієнтів.** Можна намагатися одержати більше інформації про зображення зі збережених коефіцієнтів. Наприклад, можливо швидке порівняння коефіцієнтів після ДКП у сусідніх блоках і їхнє усереднення по досить складних алгоритмах. Цей прийом помітно знижує кількість артефактів внесених у зображення ДКП, при цьому допускаючи реалізацію, що працює в реальному часі.
– **Застосування обробки кадрів, що виходять.** Відоме лихо алгоритмів стиску зображень – неминуча "блочність" (добре помітні границі макроблоків). У принципі існують алгоритми, що працюють із кадрів зовсім без застосування блоків (навіть без векторів зсуву блоків), але такий підхід поки не виправдує себе ні по ступені стиску, ні по швидкості роботи декодера. Однак можна побудувати досить швидкі алгоритми постобробки, які досить акуратно заберуть видимі границі між блоками, не вносячи істотних перешкод у саме зображення. Існують також алгоритми, що усувають на лету ефект Гіббса (див. опис JPEG) і т.п. Таким чином, істотно поліпшується візуальна якість зображення. Це також означає, що можна підвищити ступінь стиску при тій же візуальній якості.
– **Поліпшення алгоритмів масштабування зображень.** Як правило, відео на комп'ютері переглядають у весь екран. При цьому навіть застосування дуже простого й швидкого кусочно-лінійного масштабування здатна кардинально знизити швидкість програвання ролика. Тобто примітивна операція масштабування зображення буде працювати помітно довше, ніж складний алгоритм декодера. Однак швидкості сучасних комп'ютерів швидко ростуть і ті алгоритми, що викликали падіння швидкості до 7 кадрів у секунду на Celeron-300 дають живе відео – більше 30 кадрів на P 4-1200 (починає використовуватися розширений набір команд і т.п.). Т.е. з'являється можливість використовувати більше складні і якісні алгоритми масштабування на весь екран, одержуючи більше високу якість, чим для використання раніше білінійної інтерполяції (у більшості відеокарт реалізованої апаратно).
– **Застосування попередньої обробки відео.** Якщо ми хочемо одержати досить високий ступінь стиску, то можна заздалегідь пророчити, що в нашім зображенні постраждають високі частоти. Воно стане згладженим, пропадуть багато дрібних деталей. При цьому, як правило, з'являються додаткові

артефакти у вигляді смужок, ореолів у різких границь, хвиль. Значно підвищити якість зображення після кодування дозволяє передобробка з видаленням високих частот. При цьому існують алгоритми, що обробляють потік таким чином, що візуально якість зображення не змінюється, однак після декодера ми одержуємо істотно більше якісне зображення.

Вище перераховані лише окремі напрямки робіт. Фактично за 90-е роки зміни й поліпшення торкнулися всіх модулів алгоритму, побудованого за класичною схемою. Свою лепту в цей процес вносять також виробники мікропроцесорів і особливо Intel. Процесори, починаючи з P4, спеціально призначені для обробки потокових даних. Особливості архітектури процесора (зокрема невеликої в порівнянні із процесорами AMD кеш першого рівня), дають значна перевага одним алгоритмам і роблять неефективними інші. Однак загальне вдосконалювання алгоритмів стиску відео йде дуже швидко й найближчим часом можна чекати тільки збільшення швидкості появи нових розробок.

15.7.1. Motion-JPEG

Motion-JPEG (або M-JPEG) є найбільш простим алгоритмом стиску відео. У ньому кожний кадр стискується незалежно алгоритмом JPEG. Цей прийом дає високу швидкість доступу до довільних кадрів, як у прямому, так і у зворотному порядку проходження. Відповідно легко реалізуються плавні "перемотування" в обох напрямках, аудіо-візуальна синхронізація й, що саме головне – редагування. Типові операції JPEG зараз підтримуються на апаратному рівні більшістю відеокарт і даний формат дозволяє легко оперувати більшими обсягами даних при монтажі фільмів. Незалежний стиск окремих кадрів дозволяє накладати різні ефекти, не побоюючись, що взаємний вплив сусідніх кадрів внесе додаткові перекручування у фільм.

Характеристики Motion-JPEG

Потік, дозвіл (стиск): Потік і дозвіл довільні, стиск в 5-10 разів

Плюси: Забезпечує швидкий довільний доступ. Легко редагувати потік. Низька вартість апаратної реалізації.

Мінуси: Порівняно низький ступінь стиску.

15.7.2. MPEG-1

Алгоритм MPEG-1 у цілому відповідає описаній вище загальній схемі побудови алгоритмів стиску.

Характеристики MPEG-1

Потік, дозвіл: 1.5 Мбіт/з, 352x240x30, 352x288x25

Плюси: Порівняно простий в апаратній реалізації, містить перетворення, підтримувані на апаратному рівні більшою кількістю відеокарт.

Мінуси: Невисокий ступінь стиску. Мала гнучкість формату.

15.7.3. H.261

Стандарт H.261 специфікує кодування й декодування відеопотоку для передачі по каналі p*64 Кбіт, де p=1..30. Як канал може виступати, наприклад, кілька телефонних ліній.

Вхідний формат зображення – розширення CIF або QCIF у форматі YUV (CCIR 601) частота кадрів від 30 fps і нижче. Використовується зменшення розширення в 2 рази для компонентів кольоровості.

У вихідний потік записуються два типи кадрів: INTRA – стислі незалежно (відповідають I-Кадрам) і INTER – стислі з посиланням на попередній кадр (відповідають P-Кадрам). У переданому кадрі не обов'язково присутні всі макроблоки зображення, якщо блок змінився незначно передавати його звичайно нема рації. Стиск в INTRA кадрах здійснюється за схемою стиску окремого зображення. В INTER кадрах виробляється аналогічний стиск різниці кожного переданого макроблоку з "найбільш схожим" макроблоком з попереднього кадру (компенсація руху). Для згладжування артефактів ДКП передбачена можливість застосування розмиття усередині кожного блоку 8x8 пікселів. Стандарт вимагає, щоб INTRA кадри зустрічалися в потоці не рідше чим через кожні 132 INTER кадру (щоб не накопичувалася погрішність кодування й була можливість відновитися у випадку помилки в потоці).

Ступінь стиску залежить в основному від методу знаходження "схожих" макроблоків у попередньому кадрі, алгоритму рішення чи передавати конкретний макроблок, вибору способу кодування кожного макроблоку (INTER/INTRA) і вибору коефіцієнтів квантування результатів ДКП. Жоден з перераховані питання стандартом не регламентуються, залишаючи волю для побудови власних оптимальних алгоритмів.

Характеристики H.261

Потік, дозвіл: p*64 Кбіт, p=1..30, CIF або QCIF

Плюси: Простий в апаратній реалізації.

Мінуси: Невисокий ступінь стиску. Обмеження на формат.

15.7.4. H.263

Даний стандарт є розширенням, доповненням і значним ускладненням H.261. Він містить "базовий" стандарт кодування, що практично не відрізняється по алгоритмах стиску від H.261, плюс безліч опціональних його розширень. Коротко перелічимо найбільш важливі відмінності:

– **Використання арифметичного кодування** замість кодів Хаффмана. Дає можливість на 5-10% підвищити ступінь стиску.

– **Можливість завдання векторів зсуву, що вказують за межі зображення.** При цьому граничні пікселі використовуються для пророкування пікселів поза зображенням. Даний прийом ускладнює алгоритм декодування, але дозволяє значно поліпшити зображення при різкій зміні плану сцени.

– **Можливість завдання вектора зсуву для кожного блоку 8x8** у макроблоці, що в ряді випадків істотно збільшує стиск і знижує блочность зображення.

– **Поява B-**Кадрів, що дозволяє збільшити ступінь стиску, за рахунок ускладнення й збільшення часу роботи декодера.

– **Підтримка великої кількості форматів вхідних відеоданих: sub-**QCIF, QCIF, CIF, 4CIF, 16CIF і надбудовану окремо. Основна відмінність від більше універсальних форматів полягає в адаптації для декількох фіксованих дозволів, що дозволяє робити менш універсальні, але більше швидкі процедури обробки кадрів. Побудований у такий спосіб декодер працює трохи швидше.

– **Компенсація руху із субпіксельною точністю.** Можливість зрушити блок на поівпікселя також збільшує ступінь стиску, але збільшує час роботи декодера.

– **Особливий режим стиску INTRA макроблоків з посиланням на сусідні макроблоки** в оброблюваному кадрі, особливий режим квантування й спеціальна таблиця Хаффмана для поліпшення стиску I-Кадрів у ряді випадків.

– **Згладжування границь блоків декодованого зображення** для зменшення ефекту "блочності". Найчастіше при різкому русі в кадрі при стиску алгоритм виявляється змушений підвищити ступінь квантування блоків після ДКП щоб укластися у відведений на передачу бітовий потік. При цьому в кадрі виникають добре вам знайомі по JPEG блоки розміром 8x8. Як показала практика, "зрощування" границь, коли крайні пікселі блоків зрушують по яскравості так, щоб зменшити різницю, дозволяє найчастіше помітно підвищити візуальна якість фільму.

– **Зміна розширення й деформування базового кадру, що** використовується в якості базового при стиску.

– **Різні режими квантування й кодування по Хаффману.**

Характеристики H.263

Потік, дозвіл: 0.04-20 Мбіт/с, sub-QCIF, QCIF, CIF, 4CIF, 16CIF і дозволи, що набудовуються окремо.

Плюси: Алгоритм H.263 також як H.261 допускає швидку апаратну реалізацію, однак при цьому дозволяє домогтися більшого ступеня стиску при тім же якості. Підтримує стиск звуку.

Мінуси: По кількості закладених ідей перебуває між MPEG-2 і MPEG-4.

15.7.5. MPEG-2

Як уже говорилося, MPEG-2 займається стиском оцифрованого відео при потоці даних від 3 до 10 Мбіт/сек. Багато чого в ньому запозичено з формату CCIR-601. CCIR-601 являє собою стандарт цифрового відео з розміром переданого зображення 720x486 при 60 напівкадрах у секунду. Рядка зображення передаються із чергуванням, і два напівкадри становлять кадр. Цей прийом нерідко застосовують для зменшення мерехтіння. Хроматичні канали (U і V в YUV) передаються розміром 360x243 60 разів у секунду й також чергуються вже між собою. Подібний розподіл називається 4:2:2. Переклад з CCIR-601 в MPEG-I простий: треба поділити в 2 рази яркості компоненту по горизонталі, поділити потік в 2 рази в тимчасовому вимірі (забравши чергування), додати другий хроматичний компонент і викинути "зайві" рядки,

щоб розмір по вертикалі ділився на 16. Ми одержимо потік YUV кадрів розміром 352x240 із частотою 30 кадрів у секунду. Тут все просто.

Проблеми починаються, коли з'являється можливість збільшити потік даних і довести якість зображення до CCIR-601. Це не таке просте завдання, як здається. Проблема складається в чергуванні напівкадрів у вхідному форматі. Тривіальне рішення – працювати з кадрами 720x486 при 30 кадрах у секунду, як зі звичайним відео. Цей шлях приводить до неприємних ефектів при швидкому русі об'єктів на екрані. Між двома вихідними напівкадрами 720x243 зрушення стає помітним, а тому що наш кадр формується з вихідних напівкадрів через рядок, то при стиску відбувається розмивання об'єкта, що рухається. Винно в цьому ефекті ДКП, і якось виправити ситуацію, не зменшивши ступеня стиску відео, або не втративши у візуальній якості не можна. Досить розповсюдженим є застосування "деінтерлейзінга" (від англійського deinterlacing – видалення чергування рядків). Ця операція дозволяє видалити чергування, зміщаючи парні рядки в одному напрямку, а непарні в іншому, пропорційно відносному руху об'єкта в даній області екрана. У результаті ми одержуємо візуально більше якісне зображення, але трохи більше тривалу передобробку перед стиском.

Іншим рішенням є архівація парних і непарних кадрів у потоці CCIR-601 незалежно. При цьому ми, звичайно, позбудемося від артефактів, що виникають при швидкому русі об'єктів, але істотно зменшимо ступінь стиску, тому що не будемо використовувати найважливішої речі – надмірності між сусідніми кадрами, що дуже велика.

Характеристики MPEG-2

Потік, дозвіл: 3-15 Мбіт/с, універсальний

Плюси: Підтримка досить серйозних звукових стандартів Dolby Digital 5.1, DTS, висока універсальність, порівняльна простота апаратної реалізації.

Мінуси: Недостатня на сьогодні ступінь стиску, недостатня гнучкість.

15.7.6. MPEG-4

MPEG-4 кардинально відрізняється від прийнятих раніше стандартів. Розглянемо найцікавіші й корисні нововведення.

– **Розрахунок тривимірних сцен і робота із синтетичними об'єктами.** До складу декодера MPEG-4 як складова частина входить блок візуалізації тривимірних об'єктів (Animation Framework eXtension - AFX – те, що в просторіччі називають даними для тривимірного движка). Ті, хто кодував відео, знають, скільки проблем доставляють титри й взагалі будь-які об'єкти, що накладаються поверх фільму, (логотипи, заставки й т.п.). Якщо добре виглядає основний план – будуть підпорчені накладаються об’єкти, що, якщо добре дивляться вони – буде низкою загальний ступінь стиску. В MPEG-4 пропонується вирішити проблему кардинально. Об'єкти, що накладаються, розраховуються окремо й накладаються потім. Крім того, можна використовувати відеопотік навіть як текстуру, що накладається на поверхні об'єктів, що розраховуються. Така гнучка робота із тривимірними об'єктами дозволяє істотно підняти ступінь стиску при помітно кращій якості зображення.

Більше того – ніхто не заважає робити відеоролики взагалі без живого відео, а які складаються тільки з розрахованих (синтетичних) об'єктів. Розмір їхнього опису буде в рази менше, ніж розмір аналогічних фільмів, стислих просто як потік кадрів. До речі, окремо в стандарті передбачена робота з "спрайтами" – статичними зображеннями, що накладаються на кадр. При цьому розмір спрайта може бути як зовсім маленький (логотип каналу в куточку екрана), так і перевищувати розмір кадру й "прогортатися" (тобто в якості "спрайта" може бути заданий тло, а невеликі відео-об'єкти, наприклад, голову диктора, будуть на нього накладати). Це дає значну гнучкість при створенні MPEG-4 фільмів і дозволяє помітно зменшити обсяг кодованої інформації.

– **Объектно-ориентированная робота з потоком даних.** Тепер робота з потоком даних стає об’єктно-орієнтованим. При цьому дані можуть бути живим відео, звуковими даними, синтетичними об'єктами й т.д. З них створюються сцени, цими сценами можна управляти. Для простих смертних при цьому мало що зміниться, однак для програмістів об'єктне середовище означає кардинальне спрощення роботи з виникаючими складними структурами.

– **Приміщення в потік двійкового коду "З++ подібного" мови BIFS.** За допомогою BIFS у потік додаються описи об'єктів, класів об'єктів і сцен. Також на ньому можна міняти координати, розміри, властивості, поводження й реакцію об'єктів на дії користувача. У свій час Flash був названий революцією 2D графіки в Інтернеті. Аналогічний прорив в області відео робить MPEG-4.

– **Активна глядацька позиція.** Як було замічено вище, BIFS дозволяє задавати реакцію об'єктів сцени на дії користувача. Потенційно можливе видалення, додавання або переміщення об'єктів, введення команд із клавіатури. Подієва модель запозичена з мови, що вже розвивалося довгий час, моделювання віртуальної реальності VRML. Для тих, хто грав у написані на VRML гри, мабуть, що в MPEG-4 буде зовсім реально створювати "квест"-подібні (і не тільки) гри. Найширший простір відкривається для створення навчальних і розважальних програм. Представляєте, скачуєте з Інтернету один файл, що відразу в собі містить усе, що необхідно для невеликого курсу лекцій, причому ви можете прослухати його, бачити мовець голову викладача, або відключивши його, можете збільшити фрагменти ("спрайти") з матеріалами. А наприкінці – пройти короткий тест на розуміння предмета. До речі – у стандарті передбачена обробка команд на стороні сервера, тобто програма-переглядач може відіслати дані на сервер і одержати звідти оцінку. Відмінність від попередніх стандартів революційне.

– **Синтезатор осіб і фігур.** У стандарт закладений інтерфейс до модуля синтезу осіб і фігур. Наприклад, у файлі зберігаються ключові дані про профіль особи й текстури особи, а при записі фільму зберігаються тільки коефіцієнти зміни форми. Для передач типу новин, цей прийом дозволяє в десятки разів скоротити розмір файлу при чудовій якості.

– **Синтезатор звуків і мови.** Крім синтезу осіб у стандарт MPEG-4 також закладені алгоритми синтезу звуків, і навіть мови(!).

– **Поліпшені алгоритми стиску відео.** У стандарті передбачені блоки, відповідальні за потоки 4.8-65Кбіт/із із прогресивним розгорненням і більші потоки з підтримкою чересстрокового розгорнення. Для передачі по ненадійних каналах можливе використання завадостійких методів кодування (за рахунок незначного збільшення обсягу переданих даних різко знижується ймовірність перекручування зображення). При передачі відео з одночасним переглядом закладена можливість огрубіти зображення, якщо декодер через обмеження каналу зв'язку не встигає одержати всю інформацію. Усього в стандарт закладено 3 рівні деталізації. Ця можливість дозволить легко адаптувати алгоритм для трансляції відео по мережі.

– **Підтримка профілів на рівні стандарту.** Зрозуміло, що реалізація всіх можливостей стандарту перетворює декодер у досить складну й більшу конструкцію. При цьому далеко не для всіх додатків необхідні якісь складні специфічні функції (наприклад, синтез мови). Творці стандарту надійшли просто: вони обмовили набори профілів, кожний з яких містить у собі набір обов'язкових функцій. Якщо у фільмі записано, що йому для програвання необхідні такий-то профіль і декодер цей профіль підтримує, то стандарт гарантує, що фільм буде програний правильно.

Вище коротко перераховані деякі відмінності MPEG-4 від попередніх стандартів. Треба відзначити, що на момент створення стандарту гострої потреби в описані вище речах ще не було. Інакше кажучи, ми маємо справу з добре продуманою роботою з формування стандарту, що була закінчена на той час, як у ньому виникла перша необхідність.

Творцями MPEG-4 врахований досвід попередників (зокрема VRML), коли занадто рання поява стандарту й відсутність у ньому механізму профілів серйозно підірвало його масове застосування. Будемо сподіватися, що масовому застосуванню MPEG-4 такі проблеми не загрожують.

Характеристики MPEG-4

Потік, дозвіл: 0, 0048-20 Мбіт/с, підтримуються всі основні стандарти відеопотоків.

Плюси: Підтримка досить прогресивних звукових стандартів, високий ступінь універсальності, підтримка нових технологій (різні види синтезу звуку й зображення).

Мінуси: Висока складність реалізації.

15.7.7. Порівняння стандартів

Систематизуємо стандарти (табл. 15.1) і їхньої характеристики (табл. 15.2)

Таблиця 15.1.

Назва	Рік	Дозвіл і потік	Аудио	Застосування
MPEG-1	1992	352x240x30, 352x288x25, 1.5 Мбіт/з	MPEG-1 Layer II	VideoCD першого покоління
H.261	1993	352x288x30, 176x144x30, 0, 04-2 Мбіт/с (px64 Кбіт/з, де p від 1 до 30)	–	Апаратно реалізовані кодеки, відеоконференції
MPEG-2	1995	Універсальний, 3-15 Мбіт/с	MPEG-1 Layer II, Dolby Digital 5.1, DTS	DVD
H.263	1998	sub-QCIF, QCIF, CIF, 4CIF, 16CIF і набудовуються особливо	Підтримується	Апаратно реалізовані кодеки, відеотелефони, відеоконференції
MPEG-3 не прийнятий	1993-95	Телебачення високої чіткості, 20-40 Мбіт/с		HDTV
MPEG-4	1999	Універсальний, 0, 0048-20 Мбіт/с	MPEG-1 Layer II, MPEG-1 Layer III, Dolby Digital 5.1, DTS	VideoCD другого покоління

Таблиця 15.2.

Назва	За рахунок чого досягається стиск	Додаткові можливості
MPEG-1	ICT, DCT	
H.261	ICT, DCT, MC	Передача потоку даних по p-каналах із пропускною здатністю 64Кбіт (телеконференції по декількох телефонних лініях).
MPEG-2	ICT, DCT, MC	
MPEG-4	ICT, Wavelet, MC, спрайти, об'єкти із прозорим тлом, 3 d-рендеринг	Вбудована мова опису BIFS, синтезатор мови, функції анімації осіб, 3 D-Рендеринг і т.буд.

15.8. Питання для самоконтролю

1. Які параметри треба визначити, перш ніж порівнювати два алгоритми стиску відео?
2. Приведіть приклади ситуацій, коли архітектура комп'ютера дає переваги тому або іншому алгоритму стиску відео.
3. Якими властивостями відеопотоку ми можемо користуватися, створюючи алгоритм стиску? Приведіть приклади.
4. Що таке аудіо-візуальна синхронізація? Чому виконання її вимог значно знижує ступінь стиску?
5. Назвіть основні вимоги до алгоритмів стиску відео.
6. Що таке І-Кадри, Р-кадри?
7. Приведіть приклади програмно-апаратної реалізації алгоритмів стиску відео (повсякденні й досить нові).
8. Приведіть приклади областей використання відео НЕ критичних до вимоги "стійкості до помилок".

Тема 16. ЦИФРОВЕ ФОТО

16.1. Історія фотографії

В основу сучасних цифрових фототехнологій закладені результати 200-літніх пошуків в області вдосконалювання традиційної фотографії. Тому в будь-якому сучасному цифровому фотоапараті явно простежуються риси класичної вузькоплівкової камери Leica, створеної німецьким інженером Оскаром Барнаком і його послідовниками.

Ціль лекції – дати загальне подання про основні історичні віхи на шляху винаходу й удосконалювання фототехнологій. Тут же описаний пристрій "класичного" далекомірного плівкового фотоапарата.

Ми давно звикли до навколишнім нас речам, не дивуючись тому, як же чудово вони влаштовані. Візьміть у руки будь-який ілюстрований журнал. Гляньте на його обкладинку, перегорніть сторінки. Чудові кольорові фотографії – портрети, архітектурні й пейзажні знімки, жанрові сценки. Яке це чудо – зупинена мить життя! Між іншим, справжня "машина часу", що переносить нас у минуле. Поміркуєте самі – от квітка, що перецвіла роки назад. Ось дома, які сьогодні виглядають зовсім інакше. Глумливо всміхнений Хемінгуей, дотепний і уїдливий Бернард Шоу, прекрасна Грета Гарбо... Фотографія це миттєвий зріз часу й самостійний вид мистецтва, достовірний історичний документ і художній твір. Фотографія – це фотографія.

У фотографії довга історія. Камера-*Обскура* з'явилася за давніх літ. У середні століття вона була єдиним пристроєм для фіксації нерухливих зображень. Але все-таки була! Ми-Те звикли вважати, що до винаходу світлочутливих матеріалів і фотоапарата люди й уявити собі не могли, що зображення можна яким-небудь образом запам'ятати інакше, чим змалювавши його з натури. Однак камера-*обскура* використовувалася досить широко й зовсім не була інструментом для вибраних.

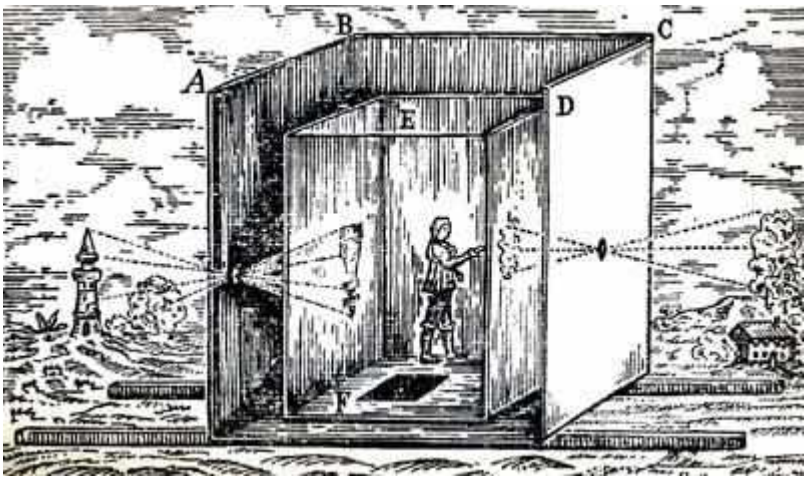


Рис. 16.1. Камера-Обскура

Влаштовано камеру-*обскура* дуже просто. Її без особливої праці можна зробити самостійно. Візьміть звичайну бляшану банку з-під консервів. У її дні

шилом проколіть невеликий отвір. На відкриту частину банки натягніть аркуш цигаркового паперу. Направте отвір банки на яскраво освітлений об'єкт і на цигарковому папері проступить тьмяне, але цілком помітне зображення. Щоб його було легше розглянути, скористайтеся шматком щільної світлонепроникної тканини – як це робили фотографи минулого. Якщо обвести олівцем контури зображення, то ми, навіть не вміючи малювати, одержимо малюнок з натури. От це і є *камера-обскура*.

Одним з винахідників фотографії був француз *Жозеф Нисефор Ньепс* (роки життя 1765-1833). До 1813 року Ньепс працював над поліпшенням способу плоского друку зображень – літографії, винайденої в 1796 році Іоханом Алоісом Зенефельдером. Вапняк, що Зенефельдер використовував для виготовлення форм, Ньепс замінив жерстиною. На цьому аркуші син Ньепса кольоровими олівцями малював картинки, що перетворювалися потім батьком у друковані форми.

Рис. 16.2. Нисефор Ньепс

Сам Нисефор Ньепс малювати не вмів (а змужнілого сина призвали на армійську службу), тому вирішив знайти спосіб змусити малювати саме світло. Зрештою він винайшов перший світлочутливий матеріал, що мав досить складний состав. Розчинивши у тваринному маслі асфальтовий лак – бітум, Ньепс наносив його на скляну, мідну або свинцево-олов'яну пластинку, що поміщав у *камеру-обскуру*. Експозиція тривала кілька годин. Коли пластичний бітумний шар затверджував і на ньому проступало видиме зображення, Ньепс виймав пластинку й у темній кімнаті обробляв її кислотою. Кислота розчиняла неекспоновані ділянки світлочутливого шару. Потім виявлена пластинка передавалася граверові, що прорисовував контури зображення й покривал пластинку чорнилом. Після цього із пластинки можна було одержати відбиток на папері – гравюру, що Ньепс назвав *геліографією* (тобто "намальованої світлом").

Перший стійкий знімок був отриманий Ньепсом в 1822 році. Але до нашого часу дійшла більше пізня геліографія, зроблена в 1826 році. Крім

геліографії Ньепс винайшов **діафрагму** – простий механізм зміни діаметра отвору камери-*обскури*, що дозволяє збільшити різкість зображення.

Рис. 16.3. Геліографія Ньепса 1826 року

Про досвід *Ньепса* довідався французький художник і винахідник *Луї Жак Манде Дагер* (роки життя 1787-1851), автор знаменитої паризької діорами – об'ємної мальовничої картини. В 1829 році він уклав з *Ньепсом* угода про розвиток геліографії. В 1837 році після багаторічних дослідів *Дагер* одержав перший знімок на поверхні полірованої срібної пластини, просоченої парами йоду. Шар солей срібла надавав пластині властивості світлочутливості. *Дагер* помістив срібну пластину в камеру-*обскуру*, піддав її експозиції (від 15, до 30 мінут) і виявив парами ртуті. Отримане нестійке зображення він промивав насиченим розчином солі й гарячою водою. У результаті з поверхні пластини вимивалися неекспоновані ділянки йодиду срібла. В 1839 році як фіксуюча речовина стала застосовуватися гіпосульфіт натрію, відкритий англійським ученим **Джоном Гершелем** (роки життя 1792-1871).

Рис. 16.4. Луї Дагер

Дагеротипія – так називався новий спосіб одержання зображень – дозволяла одержати позитивний знімок у єдиному екземплярі. Правда, зображення виходило дзеркально зверненим і розглянути його можна було лише під певним кутом.

Рис. 16.5. Стародавній дагеротип

Ще одним "батьком фотографії", поряд з *Ньепсом* і *Дагером*, уважається англійський фізик і хімік *Вільям Генрі Фокс Толбот* (роки життя 1800-1877). Його заслуга у винаході фотопаперу й *класичного негативно-позитивного процесу* одержання фотозображень. Як світлочутливий матеріал Толбот застосував просочену хлористим сріблом папір. Для цього він занурював аркуш паперу в слабкий розчин солі, а потім у розчин нітрату срібла.

Рис. 16.6. Вільям Толбот

Світлочутливий папір Толбот експонував у камері-*обскуре*, проявляв і повторно експонував, приклавши до отриманого негатива такий же аркуш паперу. У результаті виходило позитивне зображення.

Перший знімок був зроблений *Толботом* ще в 1835 році. У січні 1839 року винахідник довідається про винахід *Дагера*, публікує повідомлення про свій спосіб одержання зображень і звертається до Майклу Фарадея із проханням продемонструвати знімки на засіданні Королівського суспільства. 31 січня 1839 року доповідь *Толбота* про свій спосіб одержання зображень став надбанням наукової громадськості. Винахідник фіксуючого розчину Джон Гершель відразу придумує назву винаходу *Толбота* - фотографія, і вперше вводить в обіг терміни "негатив" і "позитив". *Толбот* працює над удосконаленням фотопроцесу й в 1840 році домагається збільшення світлочутливості паперу, що дозволило скоротити експонування до декількох мінут.

Рис. 16.7. Джон Гершель

Новий спосіб одержання фотозображень *Толбот* назвав "калотипія", а фотографи в побуті називали його "мокрим", оскільки світлочутливий папір доводилося виготовляти вручну безпосередньо перед зйомкою. "Мокрий" спосіб проіснував аж до винаходу в 1880 році американцем *Джорджем Истменом* сухої желатинової фотоемульсії, що наносилася на скляні негативні пластинки. Сам же *Толбот*, що запатентував в 1841 році свій винахід, зіграв у становленні фотографії не тільки позитивну роль. Із приводу кожного вдосконалення фотопроцесу він затівав судовий позов, уважаючи себе одноособовим автором фотографії.

Фігура *Джорджа Істмена* (роки життя 1854-1932) в історії світової фотографії коштує особняком. Його генієві належить не тільки винахід желатинової емульсії й скляної фотопластинки, але й створення великої мережі фотолабораторій, послугами яких ми користуємося й сьогодні.

Рис. 16.8. Джордж Істмен

В 1880 році *Джордж Істмен* засновує компанію Eastmen Dry Plate and Film Company, що в 1892 році перейменовується в Eastmen Kodak Company. Нова фірма займається випуском фотопластинок і світлочутливих матеріалів. В 1888 році з'являється перша масова фотокамера *Kodak*, а разом з нею й знаменитий девіз компанії – "Натисніть кнопку, а ми зробимо інше!".

Фотоапарат являв собою просту складну камеру ящикового типу, заряджену роликом світлочутливого паперу на 100 знімків. Камера була нерозбірної й заряджалася тільки у фірмових пунктах *Kodak*. Відзнявши сотню кадрів, власник камери здавав фотоапарат у лабораторію компанії й одержував його перезарядженим разом з видрукуваними знімками. Таких пунктів до початку 20 століття в Америці налічувалися тисячі й тисячі. Сьогодні вони (зрозуміло, на зовсім іншому технологічному рівні) працюють у всіх країнах миру й чи не у всіх містах.

Перша й друга моделі камер *Kodak* давали круглі знімки невеликого формату. Ця особливість обумовлювала типом застосовуваного світлочутливого матеріалу – паперу, відбиток з якої одержували контактним способом. В 1889 році *Істмен* винаходить широкоформатну прозору фотоплівку, з якої можна друкувати фотографії оптичним способом зі збільшенням. А в 1900 році Істмен випускає популярнейшую модель плівкової камери Brownie, що продавалася по рекордно низькій ціні – 1 долар. Фотоапарат розійшовся по усьому світі в мільйонах екземплярів і став першим в історії видом високотехнологічного промислового товару масового попиту. А *Джорджеві Істмену* вдалося перетворити фотографію в одне із самих популярних захоплень у світі.

Рис. 16.9. Kodak Brownie

До початку 20-го сторіччя людство впритул підійшло до двох винаходів в області фотографії, які потім у повному змісті змінили наше життя. Мова про кінематограф і вузькоплівкової 35-міліметрової фотографії.

Ім'я винахідника, що зробив, як і *Джордж Истмен*, справжню революцію у фотографії – *Оскар Барнак* (роки життя 1879-1936). В 1911 році німецький інженер *Барнак*, що служив до цього в компанії Vetzlar, влаштувався на роботу в оптичну компанію Ernst Leitz і узявся за розробку першої у світі компактної далекомірної фотокамери, у якій використовується вузька 35-міліметрова кіноплівка.

Рис. 16.10. Оскар Барнак

Знаменно саме рішення використовувати у фотографії новий формат плівки. При явному погіршенні якості знімка – 35-міліметровий кадр менше по площі кадру широкої 6-сантиметрової плівки, що у ті роки була вже в ході – фотограф одержував воістину портативну систему, що дозволяла дуже оперативно фотографувати й спростити наступну обробку плівки за допомогою нескладних пристроїв (фотобачків). Проблема принципової неможливості контактного друку фотографій з вузькоплівкового кадру вирішувалася за допомогою збільшувача – приладу оптичного друку. При цьому разом з розмірами знімальної апаратури знижувалися й витрати на матеріали – плівку й реактиви. Більше того, при масовому виробництві вузька плівка виявилася дешевше широкоформатною.

Втім, вузька 35-міліметрова фотоплівка не завжди була такий, який ми її знаємо. У перших малоформатних камерах застосовувалася неперфорована рулонна плівка, але від її швидко відмовилися, оскільки випуск таких фотоматеріалів зводив нанівець одне з головних достоїнств – високу сумісність форматів кінознімальної й фотоапаратури. Втім, набагато більше істотні схованого, невидимі ока відмінності. У перших плівках як основу (підложки) використовувався недовговічний целулоїд. Згодом він жовтіє, стає тендітним і руйнується. Застосування лавсану й сучасних полімерів усуває цю проблему.

Проста фотоемульсія на основі галогенида срібла на перших плівках закріплювалася желатином. Поверхня світлочутливого шару покривалася гігроскопічним лаком (щоб пропускати усередину шаруючи розчини). Сам світлочутливий шар був неоднорідним і мав нерівномірну товщину. Налагодження технології зайняло десятиліття. Старі фотографії виглядають так архаїчно тому, що виробництво фотоматеріалів у той час перебувало в стадії вдосконалювання. А якість фотоапаратів 30-х років було настільки високим, що цієї камери працюють сьогодні із сучасними плівками без сучка й задирки. Як і годиться справжній камері.

Ще кілька слів про фотоматеріали. Повторимо дуже коротко суть процесу одержання зображення за допомогою фотоплівки й фотопаперу на основі галогенидів срібла. Під впливом світла частки галогенида срібла

перетворюються в металеве срібло. Лужний розчин, що проявляє, відновлює галогенне срібло на засвічені при експозиції ділянках фотоемульсії плівки до металевого. Кислий фіксуючий розчин зупиняє процес прояву й вимиває з емульсії залишки які не зазнали засвітки часток галогенидів. У результаті хімічної обробки плівки ми одержуємо негативне зображення на фотоплівці. Процес друку аналогічний, за тим лише виключенням, що при експонуванні прозорого плівкового негатива на непрозорий фотопапір виходить позитивне зображення.

Фотоматеріали для монохромної (білою-білій-чорно-білої) фотографії містять один світлочутливий шар, для кольоровий – як мінімум, три (у деяких плівок може бути й більше). У свою чергу кожний світлочутливий шар покритий пофарбованим фільтруючим шаром, що пропускає світлові промені певної частини спектра – червоної, зеленої або голубий. Після проявлення й фіксування на плівці виходить негативне зображення. Емульсія фотопаперу, призначеної для кольорового друку, улаштована в такий же спосіб. При експонуванні негативного зображення світло розкладається фільтруючими шарами на ті ж базові колірні складові. У результаті зображення відновлюється до кольорового позитивного.

Це дуже спрощений опис фотографічного процесу. На практиці всі трохи складніше. Фотоматеріали містять у своїй емульсії додаткові шари – фільтруючі або захисні. А в процесі хімічної обробки застосовуються допоміжні реагенти – дубильні (зміцнювальні) емульсію, що зупиняють процес прояву, що стабілізують щільність світлочутливого шару й так далі.

Недоліки традиційної фотографії були очевидні ще в часи її становлення.

– По-перше, це висока вартість матеріалів – все-таки срібло не найдешевший метал (а фотохімічна чи промисловість не основний споживач цього металу).

– По-друге, необхідність хімічної обробки ускладнює процес одержання фотознімків.

– По-третє, якими б зробленими не були фотоматеріали, а вони піддані руйнуванню, оскільки повністю видалити з емульсії галогениди срібла неможливо.

Недоліки недоліками, а "срібна" фотографія живе другу сотню років і поки не думає здаватися!

В 1913 році *Барнак* представляє керівництву повністю готовий зразок (власник компанії Лейтц навіть бере фотоапарат із собою в Америку), але виробництво нової камери відкладається через початок Першої світової війни. В 1924 році у світло виходить перша серійна модель фотоапарата *Leica* (скорочене від Leitz Camera) з форматом кадрового вікна 24х36 мм і шторно-щілинним матер'яним затвором. Оснащена високоякісним об'єктивом зі складним тубусом, відмінно спроектована, маленька, зручна *Leica* стає загальновизнаним каноном світового фотоапаратобудування. Численні копії *Leica*, включаючи радянські "ФЕДи" і "Зіркі", наводнюють ринок, але жодна з них не може повною мірою зрівнятися із прототипом.

Рис. 16.11. Одна з перших камер Leica

Рис. 16.12. "ФЕД" – радянська копія легендарної Leica

Рис. 16.13. Найперша Leica випускається дотепер – для колекціонерів

Фотоапарат моделі, що *випускається дотепер* серійно, Leica MP зберігає *риси класичної* камери Оскара Барнака. Він як і раніше не має якої-небудь автоматики (є лише вбудований автономний експонометр), постачений механічним шторно-щілинним затвором і далекомірним механізмом фокусування об'єктива. Сьогодні, в епоху повсюдного поширення цифрових камер, *Leica* один із самих консервативних по конструкції, найдорожчих за вартістю, але в той же час один із самих зроблених фотоапаратів.

Винахід *Оскара Барнака* вплинуло на всю індустрію світового фотоапаратобудування. Навіть серед новітніх цифрових компактних і дзеркальних камер можна без особливої праці відшукати рішення, що застосовувалися в тім архаїчному на сучасний погляд прототипі першої "Лійки" 1913 року.

І вуж тим більше характерні риси, властивим першим камерам *Leica*, можна розглянути в усіх без винятку сучасних малоформатних плівкових

фотоапаратах. В історії техніки це надзвичайна рідкість – створити апарат, що відразу ж став каноном, задав основні технічні параметри цілої галузі.

Як і найперша *Leica*, будь-який сучасний *плівковий фотоапарат* складається з корпусу, затвора, механізму протягання плівки, об'єктива й видошукача. При цьому кожний вузол фотокамери має досить складний пристрій і може розглядатися, як самостійний механізм. Приміром, об'єктив фотоапарата містить у собі два механічних пристрої – діафрагмування (зміни світлосили об'єктива шляхом зміни діаметра отвору рухливими міжлінзовими шторками) і наведення на фокус. У сучасних фотоапаратах до цього додається третій пристрій – зміни фокусної відстані об'єктива, що впливає на масштаб одержуваного на плівці зображення.

Об'єктив може кріпитися до корпусу камери жорстко або бути знімним. В останньому випадку кріплення об'єктива виконується різьбовим (як у класичної *Leica*), або байонетним (спеціальний замок – байонет, називаний ще "штиковим замком", що застосовується у всіх випусках нині далекомірних і дзеркальних камерах зі змінною оптикою).

Виробництво змінної оптики – ціла галузь промисловості. Світлосильний, добре спроектований і якісно зібраний об'єктив може в десятки разів перевершувати за вартістю фотоапарат, до якого він був випущений. Вартість довгофокусних об'єктивів для зйомки на великому видаленні дзеркальними фотоапаратами доходить до 40 тисяч доларів.

Інші механізми фотоапарата об'єднані в загальний конструктив і не можуть бути замінені, що не заважає їм бути досить складними по пристрої. Навіть корпус камери, ведучий походження від дерев'яної шухлядки, складається із цілого набору деталей – рами, об'єктивної дошки (до неї кріпляться деталі байонетного замка об'єктива), зовнішніх панелей. Задня частина корпусу постачена знімною або відкидною кришкою. При цьому рама звичайно виготовляється з металу або твердого пластику. А в деяких особливо міцних камерах, наприклад, в Nikon FM 3A і Leica Minilux, рама виготовлена з титанового сплаву (в Minilux з титана виконується весь корпус).

Рис. 16.14. Nikon FM 3A

Довговічність плівкової камери визначається надійністю протяжного механізму й *затвора*. Вказівка в характеристиках фотоапарата кількості спрацьовувань *затвора* – це і є ресурс камери. У професійних фотоапаратах кількість спрацьовувань до першого виходу *затвора* з ладу (тобто до явних ознак зношування) становить не менш 100 тисяч. В аматорських дзеркальних камерах середнього класу ресурс менше – 20-30 тисяч спрацьовувань *затвора*, що відповідає зйомці приблизно до тисячі 36-кадрових плівок (досить вражаючий результат).

Затвор першої *Leica* визначив архітектуру фототехніки на ціле сторіччя. До початку 20-го століття фотозатвор являло собою або рухливі пелюстки, розташовані у фокальній області об'єктива (центральний затвор), або механізовану засувку, що кріпилася перед об'єктивом. На *Leica* був застосований шторно-щілинний *затвор*. Він засвічував плівку послідовно. Швидкість руху матер'яних шторок була постійною (що спрощувало механізм затвора й збільшувало його надійність), а величина витримки регулювалася розміром щілини між шторками. При тривалих витримках відстань між світлонепроникними шторками збільшувалося, при коротких – зменшувалося.

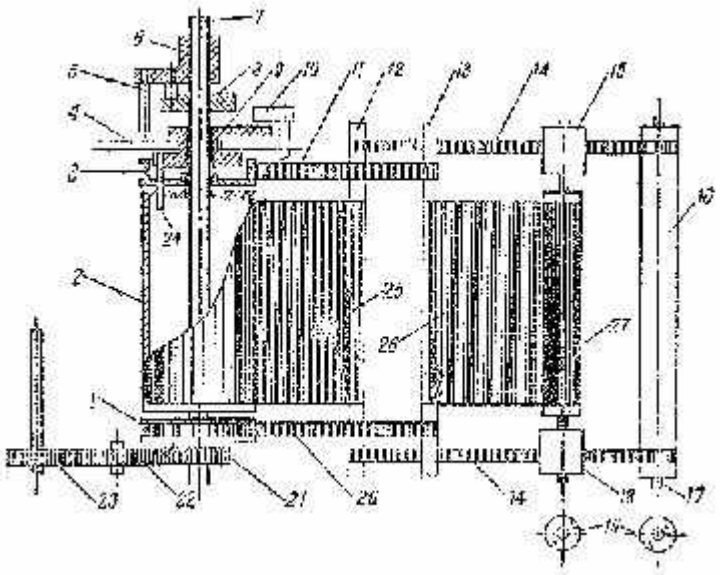


Рис. 16.15. Схема пристрою шторно-щілинного затвора

Серйозний недолік центрального пелюсткового *затвора* – неможливість реалізації дуже коротких витримок, необхідних при зйомці швидко, що рухаються об'єктів. Навіть у сучасних електронних *затворах* (мова тільки про плівкову апаратуру) пелюстки здатні відпрацьовувати витримки не коротше 1/750 секунди. Шторно-Щілинному *затвору* доступні витримки до 1/8 000 секунди (у механічних *затворах* без застосування електроніки – до 1/4 000 секунди).

Іншим недоліком центральних *затворів* є нестабільність параметрів. У пелюстковому *затворі* занадто багато вузлів тертя, а пелюстки мають велику масу й, як наслідок, зайво інертні. Крім того, у центральному *затворі* важко

розмістити потужну пружину привода. Всіх перерахованих проблем немає в шторно-щілинному *затворі*. У барабан, на який намотуються шторки, вбудована потужна циліндрична пружина зі стабільними механічними параметрами. Шторки легкі, оскільки виготовлені із прогумованої тканини, а не з металу. Шторний *затвор* легко оснастити підшипниками, що знижують тертя, (у ньому немає тертьових площин, як *у* центральному затворі).

Незручний шторний *затвор* у широкоформатних фотоапаратах. Там, де площа кадрового вікна велика, є небезпека нерівномірної засвітки різних ділянок кадру. При послідовної засвітці світлова обстановка в період експонування плівки може змінитися. Центральний же *затвор* засвічує відразу всю поверхню кадру.

Матер'яний шторно-щілинний *затвор Барнака* винайдений їм біля 1904-1910 років (за даними різних джерел), живий дотепер. У класичному виді він, як це ні дивно, є присутнім у камерах російського й білоруського виробництва – у фотоапаратах "Зеніт" і "ФЕД". У модифікованому – у всіх далекомірних і дзеркальних камерах миру.

Відмінності сучасних *затворів* від свого прабатька не так вуж великі.

– По-перше, шторки переорієнтуванні на коротку сторону кадру (рухаються не праворуч ліворуч, а зверху долілиць).

– По-друге, прогумована тканина замінена або металевої (латунної) гофрованою стрічкою, уперше застосованої в німецьких камерах Contax, або набором рухливо скріплених пелюстків – ламелей, виконаних з легенів (для зниження маси) сплавів або вуглепластика.

Модифікація шторно-щілинного *затвора* – ламельний *затвор*. Це компромісна технологія. Експозиція виробляється, як і в шторному *затворі*, щілиною, утвореної металевими ламелями. Але самі ламелі не зібрані в шторку, а закріплені в одному вузлі, тобто при спрацьовуванні повертаються, а не рухаються одна за іншою. Таким чином, ламельний *затвор* має багато загального й зі шторно-щілинним, і із центральним затвором, але по характеристиках все-таки ближче до першого.

Замість пружини, що переміщає шторки в класичному *затворі Барнака*, у сучасних камерах застосовуються електромагніти. Їхня перевага – висока точність відпрацювання витримок, недолік – неможливість роботи камери без елементів живлення. Тому в професійних фотоапаратах частіше використовують механічні *затвори*, доповнені різними механізмами "м'якого" спуска й електронного контролю витримок...

Що найбільше нагадує стара й воістину вічна *Leica*, коли ми беремо цю камеру в руки? Напевно... револьвер. І гарні механічні годинники.

З револьвером цей фотоапарат ріднить злагоджена робота механіки. Але чому – годинники? Можливо, тому, що взвод *затвора* й переклад кадру (транспортування плівки) виробляється заводною головкою, як у годин. Правда, найперші *Leica* і її послідовники мали окремі, не сполучені механізми взводу *затвора* й транспортування плівки. *Оскар Барнак* випливав устояної традиції – у популярні на той момент пластинкових камерах перемотувати було попросту нема чого. До речі, курковий взвод з'явився вже після *Барнака*.

Головка взводу довго "пручалася", але, зрештою, поступилася місцем зручному й швидкому взводу спеціальним важелем, яким фотограф одномоментно й перемотували плівку на наступний кадр, і зводив пружину *затвора*.

Нарешті, механізм фокусування, застосований *Барнаком* у фотоапараті *Leica*. Всі сучасні камери, якщо вони не постачені об'єктивом з фіксованим фокусом, успадковували механізм фокусування в цього славного німецького прабатька. Класичний ***далекомір*** – це два візирі в одному окулярі. Допоміжний візир передає зображення за допомогою дзеркала, що через важіль повертається при висуванні оправи об'єктива – під час наведення на різкість. Сполучення двох картинок в одну (усунення подвійного зображення) і є процедура фокусування. Точність забезпечується спеціальної юстировкою, у ті часи – ручний індивідуальної, у наш час – дотриманням технічних допусків при масовому виробництві.

Перші *далекоміри* були винесені за межі штатного видошукача. У розпорядженні фотографа було два окуляри. Через один він наводив різкість, через другий компонував кадр. Потім обоє оптичних пристрою були сполучені. Площа допоміжного візира була зменшена, зона наведення на фокус скоротилася до невеликої ділянки в центрі вікна видошукача. Це підвищило оперативність зйомки.

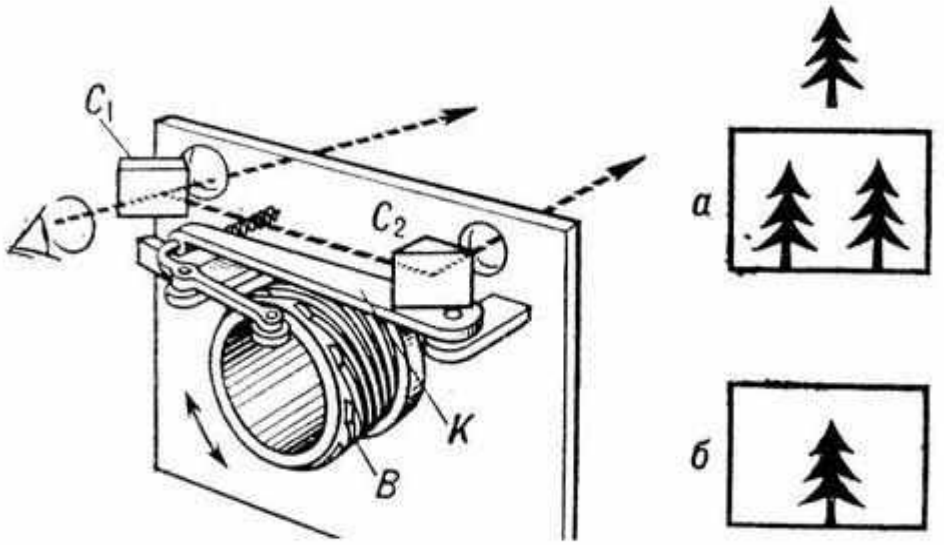


Рис. 16.16. Далекомір

Цікаво, але *далекомір Барнака* також застосовується в сучасній апаратурі, у тому числі й цифровий. Правда, не в класичному виді, але основні принципи залишаються незмінними.

Детальніше про пристрій механізмів фокусування об'єктива поговоримо окремо, поки ж як приклад нетлінності знахідок *Оскара Барнака* коротко позначимо застосовувані сьогодні системи фокусування фотоапаратів.

У сучасних цифрових і плівкових камерах використовуються дві технології - активна й пасивна. Активне фокусування – це класичний *далекомір* з моторним приводом і інфрачервоним світлодіодним прожектором. Пасивне

фокусування (контрастного типу) застосовуються в дзеркальних фотоапаратах. Кінематична схема пасивного *далекоміра* виглядає простіше, але світлочутливий елемент (звичайно не один, а ціла матриця з подібних елементів) і електронна схема керування складніше. Логічна електронна схема фотоапарата дає команду моторному приводу фокусировочного механізму об'єктива на вибір відстані фокусування. Як тільки контраст зображення на датчиках, установлених на пентопризмі, досягає максимуму (а отже, максимальним стає й струм на виводах світлочутливих елементів), фотоапарат "робить вивід", що фокус наведений. При недостатнім висвітленні на допомогу пасивному автофокусу приходять лампи підсвічування - вбудований у корпус фотокамери прожектор або вбудований спалах. Прожектор включається на короткий час, спалах дає серію малопотужних засвіток. Цього цілком достатньо для спрацьовування механізму автоматичного фокусування навіть у повній темряві.

Крім описаних схем автоматичного фокусування є безліч різних варіантів, наприклад, ультразвукове фокусування. Це та ж активна автофокусування, але замість інфрачервоного променю тут використовуються випромінювані камерою ультразвукові хвилі. У фотоапаратах для загального застосування подібні схеми використовуються рідко, оскільки точність ультразвукового *далекоміра* невисока.

До речі, принцип виміру яскравості світлового потоку використовується й в експонометричних системах автоматичних камер. Але в часи *Барнака* подібні пристрої (експонометри) тільки проектувалися...

Власники класичних *Leica* – щасливі люди. Вони можуть собі дозволити купити одну з найдорожчих малоформатних камер, або володіють екземпляром старого фотоапарата, що дотепер працює (непрацююча Leica, коли б вона не була випущена, така ж рідкість, що як не вміє плавати риба).

Навіть у наші дні, коли плівка стрімко виходить із обігу, фотоапарат *Leica* залишається мрією багатьох фотографів, предметом колекціонування й навіть поклоніння. Справа, звичайно, не тільки й не стільки у високій надійності й довговічності цієї камери. Секрет невгасаючої затребуваності фотоапарата *Оскара Барнака* в тім, що *Leica* найрідший приклад того, як спеціалізоване пристрій цілком прикладного призначення перетворилося в справжній предмет мистецтва.

16.2. Від плівки до цифрового фото

На перший погляд переваги цифрової фотографії перед фотографією традиційної не цілком очевидні. Але насправді це справжня революція у светописи, що повністю усуває саме поняття старіння оригіналів знімків. Крім цього цифрові методи фіксації зображень зробили фотографію воістину масової.

Ціль лекції – розповідь про принципові розходження між традиційною аналоговою й новою цифровий фототехнологіями. Тут же приводяться відомості про будову світлочутливих матеріалів на основі галогенидів срібла й цифрових світлочутливих матриць.

У наш час фотоапарат є не тільки в кожному будинку, але й просто в кожної людини (якщо, звичайно, уважати фотоапаратом стільниковий телефон з вбудованою камерою). Причому, мова йде тільки про цифрові фотоапарати. Ніколи повсюдно розповсюджена плівкова камера, пройшовши майже двохсотлітній шлях розвитку, зійшла зі сцени.

Нам здавалося, що плівка ще буде жити – не як інструмент професіонала (а вона в цій якості проіснує дуже довго), а як популярний, доступний, недорогий носій інформації. Але з моменту випуску на масовий ринок доступних цифрових камер пройшло всього 6-7 років. И вже на початку 2007 року ми із працею відшукаємо у своєму місті гарну лабораторію по обробці й печатці фотографій, що працює за традиційною технологією. А в магазинах майже немає гарних плівкових камер (під гарними маються на увазі доступні дзеркальні фотоапарати аматорського класу й більш-менш зроблені в технологічному змісті "мильниці"). І взагалі, століття плівки завершився. Наступило століття "цифри"...

Із цього приводу можна засмучуватися, а можна й радуватися. Якщо винахід у кінці 19 століття Джорджем Істменом, засновником компанії Kodak, міні-лабораторій і дешевих фотоапаратів зробило фотографію доступної, то винахід у кінці 20 століття цифрових камер зробило її воістину масової. При цьому під словом "масова" ми розуміємо поширеність всеосяжну й навіть абсолютну.

Дійсно, що коштувало захоплення фотографією звичайній людині, скажемо, в 70-е роки минулого століття, коли випускалися плівкові камери класичної конструкції, на ринку з'явилися інтелектуальні автоматичні фотоапарати, а по усьому світі, як гриби після дощу, з'являлися невеликі лабораторії проявлення й друк знімків? Вартість самої камери в розрахунок можна не брати – це разові витрати (навіть якщо здобувається дорогий дзеркальний фотоапарат). Плівка, проявлення, папір, печатка. Гроші начебто б і невеликі, але – гроші.

А сьогодні досить купити лише фотоапарат, а комп'ютер, цифрова "лабораторія" по обробці знімків, незабаром буде в кожній родині. І витрати на щоденну зйомку (якщо, звичайно, не роздруковувати всі кадри підряд на папері) виходять, практично, нульові.

От де роздолля захопленому фотографією людині! Хоча, чому тільки захопленому? Навіть якщо ми знімаємо лише час від часу, по святах, у туристичних поїздках – ми теж з повним на те правом можемо називати себе фотолюбителями. Нехай фотографування не перебуває в числі наших улюблених занять. Нехай фотоапарат ми використовуємо тільки як реєстратор подій, знімаємо винятково побутові сюжети й, як говоритися, "зірок з неба не вистачаємо". Зрештою, власник гарної ручки не зобов'язаний бути письменником, а власник автомобіля – гонщиком. Точно так само людина, що купила цифрову камеру, вільний знімати тоді, коли захочеться або коли буде потрібно...

Окрема розмова про камерофонах – про гібриди стільникового телефону й цифрового фотоапарата. Звичайно, це не фотоапарати в повному розумінні

слова. Це – необов'язкове мультимедійне доповнення до телефону (смартфону або комунікатору). Більшість із нас уже знає – знімати камерофоном можна, але розраховувати на гарні в технічному плані (про творчість тут говорити не будемо) результати не коштує. Тому камерофон фотоапарата не замінить. І навіть так – дуже гарний, самий дорогою камерофон не здатний конкурувати по якості знімків із самою недорогою цифровою "мильницею". У всякому разі – поки. А що буде далі, подивимося.

Так що ж – шкода плівку? Шкода. І разом з тим, бурхливий розквіт цифрової фотографії став тим подією, що здатна змінити саму картину буденного життя – так само, як змінив її стільниковий зв'язок. І, слово честі, тут немає ні найменшої причини для зневіри. Навпроти, все цікаве тільки починається...

Знову повернемося до історії.

Як це ні дивно, родоначальницею цифрової фотографії стала не "фотографічна" компанія, а електронний гігант Sony. Саме в надрах лабораторій Sony на початку вісімдесятих років двадцятого століття народився проект ***Mavica***, у якому були сформульовані, а потім і реалізовані основні принципи цифрової фотографії, в основі своєї залишившись непорушними до цих пор.

У чому принципова *відмінність цифрової фотографії від фотографії плівкової?* У способі реєстрації й зберігання зображень. Традиційна фотографія фіксує зображення в аналоговому виді. Світло, що проходить через об'єктив і сфальцьований на поверхні плівки, викликає зміна оптичної щільності солей срібла світлочутливої емульсії. Ступінь потемніння емульсії відповідає рівню засвітки. Тобто світлочутливий елемент плівки – зерно **галогеніда срібла** – змінює свої характеристики пропорційна експозиції.

Для одержання остаточного результату зйомки – готового відбитка – плівку піддають хімічній обробці, тобто прояву, закріпленню, промиванню й сушінню. Плівка в традиційній фотографії – це проміжний носій інформації. При цьому зображення на фотоплівці після проявлення стає видимим, але негативним і дзеркально зверненим.

Далі із плівкового негатива зображення переносяться на фотопапір за допомогою повторної експозиції. Через збільшувач або верстат для контактного друку негативне зображення проектується на поверхню світлочутливого фотопаперу. Потім проекспонований папір проявляється, фіксується, промивається й просушується. Цей процес практично ідентичний процесу обробки фотоплівки. Фотопапір у цьому випадку стає остаточним носієм інформації...

Ця, здавалася б, елементарно проста технологія одержання фотозображень насправді занадто складна. Багатоступінчаста обробка ще не найбільший її мінус. Основний ***недолік традиційної фотографії*** в тім, що в ній використовується різнорідні й несумісні ні із чим носії інформації.

Спробуйте, приміром, проілюструвати звичайний лист якою-небудь "паперовою" фотографією. Спосіб лише один – наклеїти фотовідбиток на аркуш паперу, на якому написане цей лист (тобто спробувати сполучити два

принципово несумісних носії інформації)... А якщо мова йде про друковане видання? А про відеофільм, якому необхідно доповнити фотозображеннями (так ще й спеціальним образом трансформованими)?

Далі – зберігання знімків. Ми звикли до того, що особистий або сімейний фотоархів зберігається у вигляді фотоальбому – паперової книги з уклеєними в нього (або закріпленими яким-небудь іншим способом) фотовідбитками. Але іноді виникає необхідність відновити втрачені або зіпсовані часом знімки. Без повторного друку тут не обійтися. Тому ми змушені зберігати ще й архів плівкових негативів.

Схоронність виявленої фотоплівки, як, втім, і паперових фотовідбитків, залежить від двох факторів – від фізичних характеристик самої плівки (або паперу) і від дотримання технології її хімічної обробки.

Старі негативні плівки мали целулоїдну підкладку. Згодом целулоїд подложки збезводнюється й стає ламким. Старі негативи дуже важко зберегти неушкодженими.

Досить непросто складається справа й зі схоронністю **емульсії.** Виготовлена на основі желатину емульсія також піддана висиханню. Крім того, емульсія старих плівок і паперу схильна до зміни кольору – з роками вона жовтіє.

У сучасних фототехнічних матеріалах проблема довгострокового зберігання негативів і готових знімків у значній мірі вирішена. Підкладка плівки виготовляється з довговічного лавсану й інших штучних матеріалів, емульсійний шар плівки захищений ласий, а паперу – полімерним покриттям. Але це не знімає проблеми механічних ушкоджень негативів. Плівку можна подряпати, забруднити, ушкодити розчинниками й так далі.

Ще одна неприємна властивість архіву плівкових негативів – його абсолютна не функціональність. Що таке архів негативів? Курні коробки зі знятими плівками, які зберігаються де-небудь у шафі й не використовуються ніяк. Негативи потрібні лише як страхова копія альбомних фотовідбитків, це єдине їхнє призначення. А місце в просторі наших будинків і офісів вони займають (і чимале, якщо фотолюбитель знімає більш-менш активно, і величезне, якщо мова йде про редакцію періодичного видання або про спеціалізований архів).

Головне ***перевага цифрової фотографії*** в тім, що зняті зображення зберігаються у вигляді цифрового коду. Цифровий фотознімок – це якийсь обсяг оцифрованої інформації, що може використовуватися разом із цифровими даними будь-якого іншого типу, наприклад, з текстовими. Комп'ютер зчитує цифрові дані з носія інформації й будує на екрані монітора зображення, ідентичне тому, що було зафіксовано світлочутливим сенсором фотоапарата в момент зйомки.

Цифрові фотографії зберігаються в пам'яті комп'ютера або на носіях у вигляді **графічних файлів** стандартних типів. Це дозволяє уніфікувати знімки, зробивши їх сумісними з будь-якими комп'ютерами, друкувальними пристроями, іншими цифровими фотоапаратами. Приміром, знімки, отримані за допомогою одного фотоапарата, можна переписати на карту флеш-**пам'яті,**

перенести на інший фотоапарат і переглянути їх на його вбудованому контрольному дисплеї. Ясно, що потреби в подібній процедурі не виникає (а сама можливість існує тільки теоретично, оскільки кожному знімку комп'ютер фотоапарата привласнює унікальне ім'я й працює тільки з ним). Набагато частіше нам доводиться переносити цифрові фотографії з комп'ютера на комп'ютер, у тому числі й на кишеньковий, або роздруковувати на різного роду друкувальних пристроях. І те, і інше не викликає проблем саме через те, що знімки зберігаються у вигляді стандартних графічних файлів.

Процес оцифровки зображення цифровим фотоапаратом виглядає в такий спосіб. Світловий потік фокусується **об'єктивом** (таким же, як і в плівковому фотоапараті) на поверхні **матриці** мікроскопічних напівпровідникових світлочутливих елементів. У момент спрацьовування затвора або запуску електронного механізму миттєвого зчитування стану матриці (останнє застосовується в камерах початкового рівня) комп'ютер фотокамери фіксує стан засвічених елементів. При цьому аналого-цифровий перетворювач (**АЦП**) фотоапарата перетворить електричні потенціали кожного елемента в набір цифрових сигналів (логічних нулів і одиниць). Інформація фіксується в дискретному виді - засвічений елемент матриці або не засвічений. Потім оцифроване АЦП зображення записується на згадку фотоапарата.

Це гранично спрощена схема роботи цифрової камери, що дає лише загальне подання про принципові розходження між традиційною плівковою й цифровою фотографією. Але й вона показує, що ускладнення технології фіксації зображення насправді приводить до значного спрощення його подальшої обробки й підвищенню точності роботи фотоапарата.

Спочатку звернемося саме до точності. Від чого залежить технічна якість плівкового знімка? Як видно, від конструкції самого фотоапарата (його об'єктива й затвора), правильної установки експозиційних параметрів і, саме головне, від характеристик і якості виготовлення застосовуваних фотоматеріалів.

Світлочутливий шар негативної фотоплівки складається із зерен галогенідів срібла, рівномірно розподілених у товщі емульсії. Однак величина й форма зерен не ідентичні – навіть при сучасному рівні технологій досягти цього неможливо. Отже, деякі зерна галогенідів мають більше високу світлочутливість (оскільки в них більше солей срібла), інші – меншої. Виходить, і ступінь фіксації засвітки на різних ділянках емульсії буде хоч незначно, але відрізнятися.

Потім наступає черга самої речовини емульсії. Полив емульсії на підкладку плівки здійснюється спеціальною машиною. Швидкість протягання прозорої лавсанової стрічки постійна, але не абсолютна. А сопла, з яких емульсія випливає на підкладку, можуть міняти пропускну здатність – засмічуватися, перебувати в різних температурних режимах (соті частки градуса, і емульсія стає ледве більше рідкої або в'язанням). І так далі... Все це укладається в норми технологічних допусків, але переконує в тім, що ніяка фотоплівка не здатна давати при експонуванні абсолютно ідентичні результати

навіть в області кадрового вікна фотоапарата, не говорячи вже про всю поверхню ролика плівки.

На відміну від фотоплівки світлочутлива матриця цифрового фотоапарата фіксує дискретні значення засвітки. Тобто там, де зерно галогеніду срібла може мати різний ступінь затемнення, елемент матриці видають точну інформацію – є засвітка чи елемента ні. І матриця, таким чином, працює набагато точніше, ніж емульсія фотоплівки. Але тільки теоретично. На практиці конкурувати з фотоплівкою світлочутливому сенсору цифрового фотоапарата поки важко.

Справа в тому, що самий мініатюрний світлочутливий елемент цифрової матриці виявляється набагато крупніший зерна галогеніда срібла самої грубозернистої фотоплівки. Сам ***елемент*** – це напівпровідниковий прилад із вмонтованими в нього провідниками. Забезпечити такий ступінь мініатюризації, щоб елемент сенсора був хоча б зрівняємо із зерном фотоемульсії по величині, сучасні технології поки не в змозі.

Це перша причина. Друга причина криється в способах одержання кольорового зображення. *На фотоплівці кольорове зображення* будується за допомогою трьох (іноді більше) світлочутливих шарів, кожний з яких відділений від попереднім пофарбованим прозорим шаром, що виконує функції світлофільтра. На світлочутливості й величині зернистості плівки це позначається мало (хоча величина зерен галогенідів у глибинних шарах емульсії повинна бути більше, ніж у шарах зовнішніх – для забезпечення рівномірної світлочутливості).

У цифровому сенсорі для одержання *кольорового зображення* використовуються, в основному, два способи. У більше зроблених цифрових камерах професійного рівня (у спеціальні фото й у більшості масових і професійних відеокамер високого рівня) застосовуються три роздільні матриці, кожна з яких постачена світлофільтром базового кольору – червоним, зеленим, синім. Матриці розташовуються під кутом друг до друга. Промені світла, що проходять через об'єктив, переломлюються спеціальною призмою й діляться на три потоки, кожний з яких експонує "свою" матрицю. На основі трьох базових колірних каналів електронікою камери й будується повноколірне зображення.

Рис. 16.17. Сенсор цифрового фотоапарата

У більшості цифрових фотоапаратів застосовується інша технологія. Потік сфальцьованого об'єктивом світла проходить через масив кольорових світлофільтрів так званої "колірної моделі Байера". У цій системі червоні,

зелені й сині фільтри розташовані в шаховому порядку, а кількість зелених фільтрів у два рази більше, ніж червоних або синіх. При цьому червоні й сині фільтри розташовані між зеленими. Більша кількість зелених фільтрів обумовлене тим, що наші очі більше чутливі до зеленого кольору.

Масив фільтрів складається з безлічі мікроскопічних світлофільтрів, кожний з яких взаємодіє з одним осередком світлочутливої матриці. Тобто можна сказати, що кожний елемент кольорового зображення (**піксель**) складається із трьох субелементів. Але при цьому дозвіл сенсора не зменшується втроє, оскільки зображення будується на основі механізмів інтерполяції, тобто з урахуванням яскравості (інтенсивності генеруючого осередками електричного сигналу) всіх елементів.

У фотоапаратах із сенсорами ПЗС (**CCD**) формування кольорового зображення будується в спеціальному пристрої після перетворення аналогового сигналу в цифровий сигнал. У фотоапаратах із сенсорами архітектури КМОП (**CMOS**) змішання кольорів може вироблятися безпосередньо в самому сенсорі. Але й у тім, і в іншому випадку, повторимо ще раз, кольорове зображення будується по спеціальних математичних алгоритмах методом інтерполяції – тобто з урахуванням яскравості сусідніх елементів кожного з базових кольорів.

Сенсори цифрових фотоапаратів програють фотоплівці по безлічі параметрів. Сенсори мають невеликий динамічний діапазон (число помітних градацій між абсолютно білим і абсолютно чорним), схильні до колірних шумів (ореолам навколо границь, що розділяють елементи зображення з більшим перепадом яскравості, наприклад, навколо фігур, знятих на освітленому тлі), мають невисоку світлочутливість (звичайно біля 50-100 одиниць ISO). Нарешті, фізичний розмір найпоширеніших сенсорів менше, ніж стандартний для вузької плівки розмір кадрового вікна 24х36 мм.

Останній параметр – фізичний розмір – має принципово важливе значення. Аргумент елементарно простій – на більшій площі кадру вміщається більша кількість світлочутливих елементів, зерен галогеніду срібла або електронних елементів сенсора. Чим більше світлочутливих елементів, тим більше дрібних деталей зображення буде зафіксовано фотоапаратом. І тем більше буде дозвіл знімка, що дозволяє збільшувати фотографії при печатці без видимої втрати якості.

В історії фотографії відомі випадки спроб введення нових стандартів на вузьку плівку. У тридцяті роки минулого століття в Латвії був випущений надкомпактний фотоапарат Minox, призначений для зйомки на 16-міліметрову неперфоровану негативну плівку. Фотоапарат мав спеціальне призначення (він застосовувався в розвідці) і вийшов дуже вдалим. У шістдесяті-сімдесяті роки в СРСР випускалися його функціональні аналоги – фотоапарати "Нарцис" і " Київ-Вега". Сучасні варіанти 16-мм фотоапаратів випускаються швейцарською фірмою Minox і сьогодні, але тільки як мініатюрні копії класичних камер Leica, призначених для колекціонерів і аматорів такого роду техніки.

Рис. 16.18. "Шпигунський" плівковий Minox

16-міліметровий формат плівки не прижився тому, що площа кадру була недостатньої для друку паперових знімків розміром більше, ніж 9x12 сантиметрів. Навіть сьогодні, коли кольорова негативна плівка найбільших світових виробників має максимальну, що наближається до теоретичної межі, що дозволяє здатність, 16-міліметровий формат для одержання відбитків нормального розміру явно недостатній (хоча подібна плівка випускається й продається в невеликих кількостях під тією же маркою Minox).

Цей приклад показує значимість такої характеристики, як *розмір кадрового вікна*, а в цифровому фотоапараті – світлочутливого сенсора.

Положення збільшується ще й тим, що матриця цифрового фотоапарата використовується не повністю. Область кадрового вікна займає більшу частину центральної області сенсора, але крайні ряди елементів сенсора у фіксації зображення не беруть участь, виконуючи службові функції (тому говорять про повні й ефективні значення розширення матриці).

Не слід забувати й про "вроджені пороки" світлочутливих сенсорів, зокрема, про "битий пікселях". У процесі масового виробництва виготовити стопроцентно працездатну матрицю практично неможливо. Кожний елемент сенсора має настільки малі розміри, що починають позначатися особливості молекулярної будови застосовуваних матеріалів. Кілька молекул газу, що проникнув в область прикордонного шару напівпровідника, здатні вивести світлочутливий елемент із ладу. У результаті сенсори всіх цифрових фотоапаратів мають кілька непрацюючих світлочутливих елементів. Але без спеціальних тестових операцій виявити ці пороки важко, оскільки одиничні непрацюючі елементи по краях кадрового вікна непомітні, а сенсори з "битими" елементами в групах відбраковуються на заводі.

Ще одна безсумнівна перевага цифрової фотографії складається в простоті обробки готових знімків. Для занять творчою фотографією власникові плівкової камери прийде обзавестися необхідним устаткуванням – проявною машиною (або набором бачків і ванночок), збільшувачем, набором реактивів і самою темною кімнатою. Що практично рівнозначно придбанню в особисте користування всього комплексу встаткування міні-лабораторії (які, як ми вже

говорили, зживають себе, поступаючись місцем цифровим міні-лабам). Або... все-таки перемінити плівкову камеру на цифровий фотоапарат і персональний комп'ютер.

Саме *персональний комп'ютер є фотолабораторією для цифрової фотографії*. Сам фотоапарат при цьому можна розглядати як черговий периферійний пристрій для ПК, тривимірний сканер з можливістю вилученого й автономного функціонування.

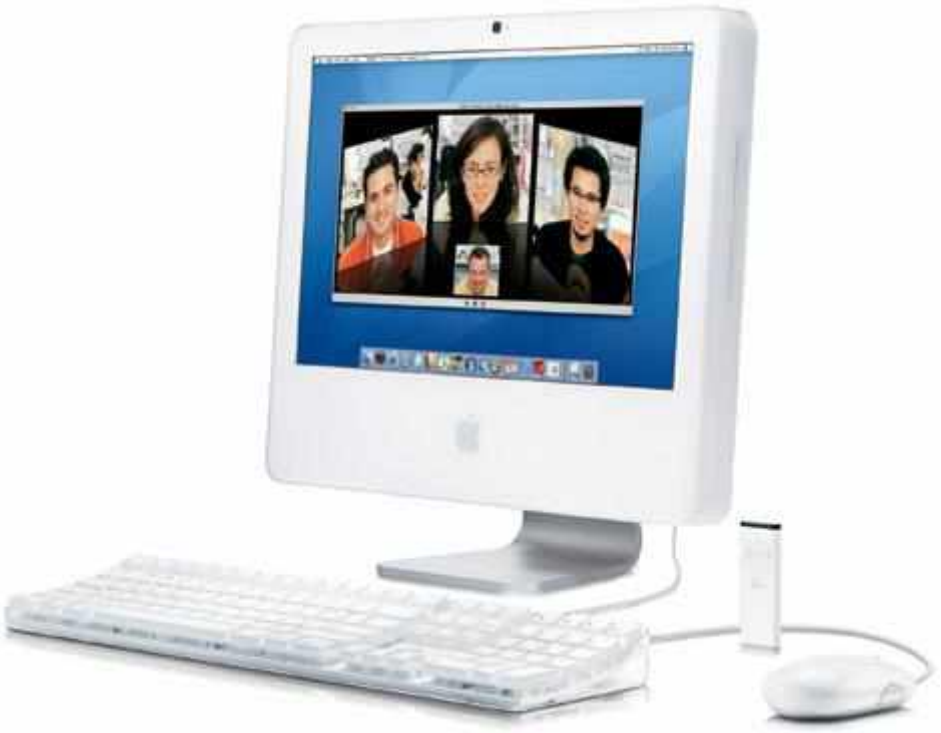


Рис. 16.19. Комп'ютер Макінтош, відмінне рішення для аматорської фотографії

Але в останні роки фотолюбителі одержали більше широкі можливості. Крім послуг спеціалізованих лабораторій або придбання персонального комп'ютера, вони можуть придбати невеликий і недорогий *принтер для прямого друку знімків*. Мова про сублімаційні і струменеві кольорові **принтери,** розрахованих на фіксований формат відбитка (звичайно 10x15 див і, як опція, менше). Втім, ці недорогі друкувальні пристрої здобуваються найчастіше не замість комп'ютера, а на додаток до нього – як портативний принтер, що дозволяє одержувати фотовідбитки, коли комп'ютер недоступний. Якщо ж використовувати тільки пару "фотоапарат-принтер" і спробувати обійтися без комп'ютера, то фотолюбитель одержить більше дорогий аналог послуг міні-лабораторії, що працює із плівковими негативами. Набір інструментів для обробки знімків буде ледве розширений (стануть доступними експокорекція при печатці, виправлення колірних перекручувань і, можливо, невелике масштабування), але для творчості залишиться явно недостатнім.

Рис. 16.20. Портативний фотопринтер

Нарешті, ще про одну очевидну перевагу цифрового фотоапарата перед плівковим. Будь-яка сучасна цифрова камера оснащена кольоровим дисплеєм. І це чудовий інструмент для навчання азам фотографії й удосконалювання майстерності, що важко переоцінити.

Тільки що знятий кадр виводиться на вбудований у камеру екран. Знімок можна збільшити, щоб розглянути деталі. Можна оцінити правильність вибору експозиційних параметрів і композиційне рішення. Якщо фотограф чимсь не вдоволений, перезняти кадр можна відразу, не чекаючи лабораторної обробки й роздруківки на папері. Погодитеся, кращого фотоапарата для навчання не знайти. Навіть із дуже якісним плівковим фотоапаратом починаючий фотолюбитель не одержить ні найменшого подання про те, чи помилився він тільки що, знявши цікавий кадр, наскільки знімок виявився вдалим, що варто змінити, щоб не допустити подібних помилок у майбутньому. Згадати обставини зйомки потім, коли міні-лабораторія віддрукує готові фотографії, дуже непросто. Особливо якщо в лабораторію здається кілька плівок скопою (як звичайно й трапляється).

Інше важливе перевага вбудованого *контрольного дисплея* – можливість його використання як електронний видошукач, функціонального аналога матового скла. Цей неабияк забутий інструмент – матове скло – застосовувався в пластинкові й дзеркальних одно- і двохоб'єктивних середньоформатних камерах для точної побудови кадру. У принципі, користуватися *контрольним дисплеєм* цифрового фотоапарата, як видошукачем, навіть зручніше, оскільки класичне матове скло дає перевернене зображення.

Дисплей здатний функціонально замінити й видошукач вузькоплівкових дзеркальних фотоапаратів з вбудованої що обертає пентапризмою. Зображення на контрольному екрані цифрового фотоапарата відповідає зображенню, що фіксується світлочутливим сенсором. Тобто фотограф бачить на екрані саме те, що буде знято. Дуже зручно для макрозйомки й зйомки через мікроскоп.

Рис. 16.21. Об'єктив фотоапарата Canon PowerShot G7

Більше того, у компактних і ультракомпактних камерах початкового й середнього рівня, у просьюмерских апаратах, призначених для творчої зйомки й навіть у таких напівпрофесійних "ультразумах", як Sony DSC-R1, електронний кольоровий дисплей використовується в якості основного і єдиного видошукача. Хоча в дзеркальних цифрових камерах (і в цьому їхнє безсумнівне перевага) використовується винятково оптичний видошукач реального зображення, а *контрольний дисплей* використовується лише для перегляду готових знімків і для керування камерою за допомогою системи екранних меню.

Рис. 16.22. Цифрова камера Sony DSC-R1

16.3. Умовна класифікація цифрових фотоапаратів

Світова промисловість випускає величезну кількість фотознімальної техніки. Але насправді гадана розмаїтість не так вуж великий, камери відмінно укладаються в рамки неофіційної, але вже устояної класифікації.

Ціль лекції – розповісти про класифікацію цифрових фотоапаратів і про модельну лінійку камер найбільших світових виробників.

Цифрові фотоапарати, як зовсім недавно фотоапарати плівкові, по своєму призначенню підрозділяються на кілька груп. У самому "низі" – дешеві й прості автоматичні "компакти" або "мильниці" (їх назвали так тому, що корпуса цих камер і справді нагадують пластмасові мильниці). Однак, паралель із плівковими автоматичними камерами тут досить умовна.

Будь-яка найпростіша плівкова "мильниця" і цифровий фотоапарат початкового рівня це далеко не те саме. Почнемо з того, що цифровий фотоапарат улаштований набагато складніше плівкового "аналога". Його світлочутливим елементом, що замінив плівку, є **сенсор** – світлочутливий датчик, що складається із сотень тисяч і мільйонів (кількість залежить від класу камери) мікроскопічних елементів, зібраних у прямокутну плоску матрицю. Кожний елемент можна зрівняти із крупинкою галогеніду срібла фотоплівки. Тільки зображення тут будується по зовсім інших фізичних принципах.

Далі – у цифровій камері працює потужний комп'ютер, що встановлює експозицію (**витримку** й **діафрагму**), перетворить електричні сигнали, одержувані з матриці, у цифровий код, управляє всіма вузлами фотоапарата.

Нарешті, оптика. У дешевих плівкових "мильницях" дуже часто встановлювали простий об'єктив, лінзи якого були виготовлені із пластмаси. Пластикова оптика зустрічається й у цифрових камерах початкового рівня. Але зовсім примітивних оптичних схем, як і оптики невисокої якості, тут не зустрінеш. Який зміст установлювати за примітивною пластиковою лінзою дорогою високотехнологічний сенсор?.

Ще чотири-п'ять років тому класифікація цифрових камер починалася з недорогих (до 100 доларів) цифрових фотоапаратів із сенсорами **CMOS**, що мають дозвіл VGA – 350 тисяч елементів (пікселів). Але цей клас апаратури зійшов зі сцени прямо на наших очах. Сьогодні він повністю витиснутий недорогими стільниковими телефонами з такими ж сенсорами VGA і такою же простою оптикою.

Ці камери – перша жертва розвитку цифрових фототехнологій. Але справа вони своє зробили, ставши стартовою площадкою для камерофонів і задавши початковий рівень цифрових фотоапаратів. Справа в тому, що ці дешеві камери (згадаємо марки – Aiptek, Agfa, Volkano і так далі) сприймалися, скоріше, як іграшки, ніж серйозна заміна плівковому фотоапарату. Так воно, у принципі, і було. Зробити технічно зроблений знімок простою камерою було неможливо (як неможливо зробити його стільниковим телефоном із сенсором VGA).

Але сьогоднішні прості камери зовсім не так прості, як здаються (особливо, якщо судити про їхню оснащеність по цінниках). Це повноцінні фотоапарати, якими можна робити дуже непогані знімки. Навіть у руках не

самого вмілого фотолюбителя цифрова камера початкового рівня буде вже не іграшкою, а самим що ні на є фотоінструментом. І технічний рівень сучасних цифрових фотоапаратів можна сміло зрівняти з рівнем далекомірних і автофокусних плівкових фотоапаратів недавнього минулого, що ставляться до середньої цінової категорії.

Про особливості пристрою цифрових "мильниць" поговоримо трохи нижче. Поки ж про відмінності усередині цієї групи. Зайшовши в магазин цифрової техніки легко помітити, що компактні цифрові фотоапарати дуже по-різному коштують. Серед них є демократичні моделі, що продаються за 120 доларів і є мініатюрні камери за 400-500 і більше доларів. Звідки ця різниця в ціні? У чому отут справа?

Може здатися, що головна причина – дозвіл сенсора. Це й так, і не так одночасно. Справа в тому, що камери всіх цінових груп прагнуть (не самі камери, звичайно, а їхні конструктори й виробники) до збільшення розширення. І сьогодні 120-доларова "мильниця" може мати сенсор дозволом в 4 або 5 мегапікселів, а камера за 500 доларів – сенсор 6-ти або 7-мегапіксельного розширення. Різниця зовсім невелика. І можна сподіватися, що в найближчому майбутньому ця різниця стане або ще менше, або зовсім зникне.

Насправді ціна камери залежить від безлічі факторів. Крім сенсора – а в дешеві камери вбудовують звичайно сенсори попереднього покоління, у той час, як дорогі компакти оснащують сенсорами новітніх розробок (а це й дозвіл, і світлочутливість, і динамічний діапазон і безліч інших характеристик, про які обов'язково поговоримо) – фотоапарат ще містить у собі високоякісну світлосильну оптику, швидкодіючий процесор обробки зображень (той самий комп'ютер, що ми вже згадували), електромеханічний затвор (найпростіші камери VGA, яких уже не зустрінеш у магазинах, і фототелефони затвора не мають зовсім). Плюс матеріал корпусу, акумулятор, плюс яскравий кольоровий дисплей. От і набігають чималі гроші. Не будемо забувати й про те, що у вартість новітніх моделей входять витрати на перспективні розробки й наукові вишукування. Цифрова фотографія одна з найбільш наукомістких галузей, що вимагає постійних фінансових уливань і інтенсивних наукових досліджень.

От і виходить, що в групу цифрових "компактів" можна включити й недорогі фотоапарати, і камери рівнем вище... У чому відмінності? Їхня безліч. Одне з основних – розміри камери. Фотоапарати початкового рівня компактні, але не мініатюрні. Це обумовлено й конструкцією об'єктива, і специфікою автономного живлення. У цих фотоапаратах як джерело живлення використовуються стандартні нікель-металгідридні акумулятори формату АА (так звані "пальчикові"). У комплект найдешевших камер вони не входять – фотоапарати комплектуються "пробними" лужними елементами, які ми називаємо "алкаліновими батарейками". Із цими елементами камера працює погано. Лужний елемент не здатний довгий час віддавати великий струм, а енергоспоживання фотоапарата таке, що в деякі моменти він споживає струм силою в 1 і навіть в 2 ампери (і більше, не будемо забувати про енергоємний

конденсатор вбудованого спалаху, яскравий дисплей з лампою підсвічування й сам сенсор). У результаті елементи розряджаються швидко й у самий невідповідний момент. Пророчити час розряду неможливо. Але після короткого "голодної непритомності" алкалінові елементи відновлюють свою енергоємність і можуть служити якийсь час джерелом живлення – до наступного непередбаченого швидкого розряду.

У більше дорогих камерах використовуються **літієві** (літій-іонні або літі-полімерні) акумулятори. І це, між іншим, один із плюсів недорогих камер, крім повсюдно розповсюдженого формату акумуляторів. А плюсом більше "просунутих цифромильниць" є те, що їхні акумулятори більше компактні й мають високу ємність. Те, що вони мають нестандартний, не універсальний формат недоліком уважати навряд чи все-таки треба, тому що усередині модельної лінійки кожної компанії-виробника акумулятори жорстко стандартизовані. Тобто до своєї камери, який уже не перший рік від роду, акумулятор можна підібрати без великої праці - якщо камера ця має ім'я відомого виробника (Canon, Nikon, Sony і так далі).

Наступна причина підвищення вартості компактних камер – зменшення розмірів корпусу до рекордних величин при одночасному збільшенні розміру матриці ЖК екрана й застосуванні в конструкції фотоапарата високоміцного легкого металу (звичайно це магнієвий сплав). Подивитеся самі – малюсінькі камери від Sony, Olympus, Canon зроблені суцільно з металу. Дисплеї покривають, практично, всю тильну частину корпусу фотоапарата. А оптичного видошукача тут не знайти... Ще один плюс бюджетних фотоапаратів? Так, напевно. Але в ультракомпактний корпус телескопічний видошукач, такий, щоб з ним можна було реально працювати, вмонтувати неможливо в принципі. Тому у всіх ультракомпактних камерах як видошукач використовується тільки рідкокристалічний дисплей.

Про не настільки явні відмінності поговоримо нижче. А зараз звернемося до наступного класу фотознімальної апаратури – до камер для ентузіастів. Їх ще називають "просьюмерками" (від Professional Consumer, можна вільно перевести як "професійний споживач"). ***Просьюмерки*** – це більші цифрові камери з жорстковбудованою (незмінної) оптикою, сенсорами більших розмірів, розвитий автоматикою, наявністю ручних режимів (у тому числі й ручному фокусуванні), але при повній відсутності оптичного видошукача. Замість телескопічного або дзеркального видошукача в цих камерах використовуються великий зовнішній і маленький внутрішній (вбудований в окуляр) дисплеї. По одному можна орієнтуватися або будувати кадр на деякій відстані від камери, по іншому – прилинув до окуляра, як у випадку із дзеркальним або класичним далекомірним фотоапаратом.

Яркий приклад відмінної просьюмерки – камера Sony DSC-R1. Дуже гарний фотоапарат. Але... Це знову такий цифровий бронтозавр, представник вимираючого класу фотоапаратури. Так, так, тільки-но з'явившись, проживши

бурхливе життя, повну новацій, відкриттів, стрімкого зростання, просьюмерки звільняють дорогу дзеркальним камерам.

Це неминучий процес. Плівкова фототехніка вдосконалювалася майже двісті років і прийшла до оптимальних форм – до дзеркальної оптичної схеми й тих пропорцій, які ми прекрасно знаємо по професійних фотоапаратах Canon, Nikon, Minolta. І цифрова знімальна техніка пройшовши короткий шлях експерименту з формою корпусу, з ергономікою кнопок і дисків керування, вертається до перевірених рішень. Все йде до того, що на ринку залишаться тільки компактні "мильниці" (які все-таки досить різноманітні й за формою корпуса й по технічних рішеннях) і близькі до класичних пропорцій дзеркальні камери.

Чим гарний дзеркальний фотоапарат і в чому його мінуси? Для недосвідченого фотолюбителя ці особливості можуть бути неочевидними. Проте те, що може дзеркальна камера, компакту не під силу в принципі. Мова про магічну абревіатуру **ГРЗП** – про глибину різко зображуваного простору. А також про пов'язаній з ним ручному фокусуванню об'єктива...

Згадаєте гарний портрет роботи відомого фотохудожника. Що відрізняє його в першу чергу? Виділення різкістю око й легеня "розмивання" другорядних деталей - волосся, вух і тим більше тла. Це і є ефект впливу ГРЗП. Використовувати цей прийом можна тільки відключивши **автофокус**, повністю відкривши діафрагму об'єктива й підбираючи експозицію установкою витримки затвора. Тобто камера працює якщо не в повністю ручному, те хоча б у напівавтоматичному режимі, а автофокусировка відключається зовсім.

Але не ГРЗПом єдиним жива творча фотографія. Дзеркальний видошукач дозволяє дуже точно будувати кадр, оскільки наше око бачить рівно те, що відображається на поверхні світлочутливого сенсора. Дзеркальна камера працює дуже швидко, прикрі затримки між двома натисканнями на спускову кнопку затвора відсутні в принципі (насправді просто дуже малі). Власники "цифромильниць" знають, яка це неприємна штука – тривала затримка між кадрами. Затримками страждають навіть дорогі просьюмерки. Приміром, у тої ж Sony DSC-R1 при збереженні кадрів в **RAW** (особливий нескомпресований формат графічного файлу, що використовується професіоналами для коректування параметрів зображення на комп'ютері) затримка може становити до 10 с. Сенсори дзеркальних камер мають самі більші фізичні розміри – вони в 1, 6-1,5 разу менше стандартного кадрового вікна плівкової камери (24х36 мм). Чим більше фізичний розмір сенсора, тим вище якість знімка.

Нарешті, із дзеркальною камерою можна використовувати весь парк змінної оптики, що випущено провідними компаніями миру за десятиліття виробництва плівкових дзеркальних фотоапаратів.

Рис. 16.23. Набір змінної оптики для дзеркального фотоапарата

А недоліки? Їх теж вистачає. Перше – ціна. Найдешевша цифрова "зеркалка" коштує близько 600 доларів. Друге – дисплей дзеркальної камери не працює як видошукач у принципі. На ЖК екрані можна тільки переглянути вже зняті кадри. Третє – дзеркальна камера не вміє знімати відео (до слова, сама зроблена просьюмерка Sony DSC-R1 – теж). І четверте, навряд чи ні саме головне... Сенсор дзеркального фотоапарата зі змінною оптикою може бути засмічений пилом. Видалити цей пил самостійно дуже складно. Елементи світлочутливої матриці мають мікронні розміри. Малюсінька порошина закриває відразу кілька десятків осередків, відповідно, виводячи з ладу кілька десятків пікселів.

У дорогі (і не дуже) камерах використовуються схеми ультразвукового струшування пилу з поверхні сенсора (наприклад, у дзеркальних камерах Olympus). Всі камери мають спеціальний режим очищення сенсора, коли дзеркало піднімається й утримується до вимикання камери. У цей момент із камери знімають об'єктив і здувають пил спеціальною гумовою грушою або за допомогою балона зі стисненим повітрям. Але, як показує практика, повне обезпилення сенсора можливо тільки в умовах добре оснащеної майстерні.

Крім описаних груп цифрової фототехніки особняком коштують фотоапарати професійні. Вони класифікуються за власними правилами, розглядати які ми тут не будемо... Найпростіше орієнтуватися за вартістю. Аматорська апаратура – дзеркальний фотоапарат з "китовим" об'єктивом (з найпростішим об'єктивом-трансфокатором) – коштує від 600 до 1200 доларів.

Професійні камери – від 1500 і до 30 і більше тисяч доларів. Але відразу помітимо, що оснащеність сучасних аматорських "зеркалок" дорівнює оснащеності професійних цифрових фотоапаратів шести-семирічної давнини. Тобто вдосконалюється професійна апаратура, а разом з нею вдосконалюється й апаратура аматорська, переймаючи в професійної всі її недавні досягнення й технології.

Детально говорити про професійну техніку змісту ні, оскільки це окрема й зовсім самостійна тема. Тут же відзначимо лише найбільш характерні особливості подібних камер.

Професійні фотоапарати оснащуються сенсорами з більшими фізичними розмірами – до 24х36 мм, тобто повністю співпадаючих з розмірами малоформатного плівкового кадру. Це дозволяє використовувати весь парк змінної оптики без обліку кроп-фактора (збільшення фокусної відстані об'єктива через зменшення фізичних розмірів сенсора). В аматорських камерах стандартний 50-мм об'єктив має фокусна відстань або 75, або 80 мм для сенсорів із кроп-фактором 1,5 і 1,6 відповідно. Тобто замість гарного нормального об'єктива виходить "портретник" і навіть "телевик". Найбільші проблеми у зв'язку із цим викликає застосування ширококутової оптики - професійні камери цих труднощів позбавлені.

Далі – зі світлочутливістю й іншими характеристиками ясно й так, у дорогу апаратуру встановлюють самі нові, самі зроблені сенсори. А от з іншими характеристиками справа складніше.

Є в професійному фотопобуті таке поняття, як ресурс камери. Ця досить суб'єктивна й приблизна характеристика прийшла з області плівкової апаратури. Раніше вважалося, що дзеркальна плівкова камера аматорського рівня (із пластмасовим фільмовим каналом, електромотором, ламельним металевим затвором, автофокусом) мала ресурс приблизно в 20-30 тисяч кадрів. Тобто життя камери до її зношування обмежувалася зйомкою від 600, до 1000 36-кадрових плівок... Скажете, мало? Але чи багато хто з нас, у кого були (і є) плівкові Nikon, Canon, Olympus або Minolta зіштовхувалися з тим, що камера розсипалася від старості? Здається, далеко не всі.

Отож, професійні плівкові камери мають ресурс на порядок вище – до 100 тисяч кадрів. До подібним до камер можна віднести в повному змісті великий механічний фотоапарат Nikon FM2. Ну а камери Nikon F3 і далекомірні фотоапарати Leica мали фантастичну довговічність у кілька сотень тисяч кадрів.

Який ресурс сучасних цифрових професійних "зеркалок", сказати важко. Зношування плівкової камери визначався більше зношуванням затвора й механізму транспортування плівки, того самого вузла, що випробовував нежартівливі перевантаження (на морозі плівка ставала твердої, у жару прилипала до напрямних і притискного столика). У цифровій камері плівки немає. Є затвор, оптика й сенсор. Ресурс затвора дуже великий – ніяк не менше 100 тисяч спрацьовувань. У цьому змісті аматорські камери від професійних відрізняються не настільки сильно, як може здатися. А от сенсор... Його довговічність поки ніхто не перевіряв. Цифрові камери Kodak, випущені на

початку 90-х років на основі топових моделей плівкових камер Nikon, працюють дотепер. Хоча світлочутливість сенсора повинна за ці роки (близько 14 років!) знизитися.

Як би там не було, але в професійній апаратурі використовуються металеві корпуси, високоточні й високонадійні ламельні затвори. І випадки поломок у цих складніших камер вкрай рідкі. Професійна техніка повинна бути бездоганної. Інакше – кому вона буде потрібна? Кому завгодно, тільки не справжнім репортерам і художникам...

В індустрії цифрової знімальної апаратури зложилася власна ієрархія брендів. На жаль, не всі класичні марки вижили. Спалахнула й згасла Minolta, що сьогодні відроджується в образі приголомшливої дзеркальної камери Sony Alpha DSLR-A100 (конструкція була куплена японським гігантом Sony у консорціуму Konica Minolta). Деякі могутні в минулому компанії, начебто Polaroid, просто збанкрутували. Але виникли й нові імена, які раніше відносини до фототехніки не мали – наприклад, Panasonic (торговельна марка компанії Matsushita).

Зробимо короткий екскурс по модельному ряді цифрових фотоапаратів найвідоміших світових виробників. При цьому будемо мати на увазі, що цей огляд ні в якій мері не може вважатися вичерпної. В області виробництва цифрової фототехніки працює величезна кількість електронних компаній...

Компанія **Canon**. Модельний ряд фотоапаратів цього найбільшого світового виробника прямо-таки вражає. Чого тут тільки немає... Помітимо, що провідної компанії в області фототехніки звичайно роблять і світлочутливі сенсори для своєї апаратури. Вони ж продають ці сенсори компаніям-конкурентам. Приміром, у фотоапаратах Nikon (у дзеркальні точно) використовуються сенсори виробництва Sony.

Отож, всі цифрові камери Canon (говоримо тільки про фото, компанія випускає ще й чудові відеокамери) підрозділяються на кілька груп. Відкриває модельну лінійку камери PowerShot Axxx. Це аматорські камери початкового й середнього рівня. Всі вони відрізняються досить великими (по мірках компактних цифрових фотоапаратів) корпусами із пластику, наявністю телескопічного видошукача (є, зрозуміло, і ЖК дисплей) і живленням від елементів (акумуляторів) формату AA. Серія A4xx включає чотири моделі - A420, A430, A450 і A460 – виконані в єдиному конструктиве, що має витягнуту форму, і оснащені 1/3-дюймовими сенсорами. Відмінності в дозволі сенсорів (від 4 мегапікселів в A420, до 5 мегапікселів в A460), в оптику (3-х і 4-х кратні зуми) і в розмірах контрольних дисплеїв (1,8 дюйма по діагоналі в молодших моделей і 2 дюйми по діагоналі в старших). Це найпростіші й самі доступні камери в сімействі Canon, які, проте, придатні для заняття творчою фотографією, оскільки наділені напівавтоматичними режимами, і дають чудові знімки.

Серія PowerShot A5xx, виконана в більше "ухватистому" корпусі, містить у собі три моделі – A530, A540 і A550 (до 7 мегапікселів). Відмінності приблизно такі ж, як і в серії A4xx. Серія PowerShot A6xx - це дві моделі A630 і A640

(дозвіл сенсора до 10 мегапікселів). Серія PowerShot A7xx – дві моделі A700 і A710 (до 7,1 мегапікселів).

Особняком коштують камери PowerShot S80, як компактне рішення в бюджетній лінійці. А також камери G7 (сенсор 10 мегапікселів) і S3 IS (сенсор 6 мегапікселів). Остання модель PowerShot S3 IS ставиться до класу просьюмерок.

Компактні камери середнього класу представлені величезною модельною лінійкою Digital IXUS, що включає ні багато, ні мало 10 моделей фотоапаратів. Їхньої відмінності – компактний металевий (комбінований, із застосуванням пластику) корпус, застосування фірмових акумуляторів (внутрішнього корпоративного стандарту). При цьому деякі моделі не обділені телескопічним видошукачем. Canon Digital IXUS перший ультракомпакт на ринку цифрової фотоапаратури. Правда, уже давно не єдиний.

З недавніх пор аматорська цифрова дзеркальна камера Canon EOS 400D перемінила популярну модель EOS 350D. Крім цього фотоапарата компанія випускає ще 4 моделі напівпрофесійних і професійних дзеркальних камер, найвідоміша з яких EOS-1D Mark II N с 8, 5-мегапіксельним сенсором і ресурсом в 200 тисяч кадрів.

Компанія Canon один зі світових лідерів виробництва цифрової фотоапаратури. У недорогі цифрові камери вона ставить сенсори CCD власного виробництва, а в дзеркальні камери унікальні сенсори CMOS, також свого виробництва. І це досить примітний факт... Про пристрій сенсорів ми ще поговоримо, а зараз відзначимо одну характерну рису цього типу світлочутливих датчиків – вони дуже стабільні й довговічні. Сенсори CCD схильні до деградації. Згодом у них знижується чутливість, знижуються характеристики, виходять із ладу окремі світлочутливі осередки. Сенсори CMOS цим порокам не піддані, обережно додаймо – теоретично. Практика показує, що дзеркальні камери із сенсорами CMOS і CCD у плані стабільності характеристик друг від друга не відрізняються.

Модельний ряд провідних світових компаній постійно міняється. Виходять нові камери, знімаються з виробництва старі. Нові фотоапарати Canon одержують сенсори більше високого розширення, нові процесори, нові контрастні, яскраві дисплеї. Так що погнатися за всіма новинками дуже й дуже непросто...

Те ж саме – істотна корекція модельного ряду цифрових фотоапаратів – відбувається й в іншій великій компанії миру, відомої своїми видатними у всіх відносинами камерами. Мова про Nikon, ім'я якої давно стало символом надійності і якості.

Nikon, як і її могутній конкурент Canon, випускає повний спектр моделей фотознімальної апаратури, починаючи із простих аматорських камер, закінчуючи складнішими моделями для професіоналів.

Моделі, призначені для фотолюбителів, Nikon назвав Coolpix. Компактні камери (цифрові "мильниці") компанія підрозділяє на три групи - просту, у яку входять моделі Coolpix Lxx, середнього рівня (так звана "стильна" серія), у яку включені моделі Sxx і премиум-класу, у яку входять моделі Coolpix Pxxx.

"Просьюмерок" Nikon більше не випускає, зробивши основний акцент на виробництві дзеркальних камер аматорського рівня. Ну й саме собою розуміє – камери професійні. Тут в Nikon конкурентів мало (крім Canon).

Початкова серія Coolpix Lxx – це 7 моделей (на лютий-березень 2007 року), починаючи з найдешевшої L2 (з 6-мегапіксельним сенсором), закінчуючи камерою L12 (з 7, 1-мегапіксельним дозволом). Основні особливості серії Lxx – пластмасовий конструктив, застосування як носії інформації карт пам'яті формату SD (Secure Digital), трикратний об'єктив-трансфокатор, відсутність оптичного видошукача (у його ролі виступає рідкокристалічний кольоровий дисплей), великий набір автоматичних режимів відпрацьовування експозиції й повна відсутність ручних режимів. Камери Nikon Coolpix Lxx – відмінний вибір для починаючих фотолюбителів і для людей, що прагнуть мати техніку найвідомішого виробника за відносно невеликі гроші.

Середня група аматорських камер – Nikon Coolpix Sxxx. У цій серії випускаються 8 компактних моделей, які значно відрізняються друг від друга (відмінностей більше, ніж у серії Lxx). Модельний ряд відкривається камерою S5. Це ультракомпакт у тонкому металевому корпусі, з оптикою особливого пристрою (оправа об'єктива не виступає за межі корпусу), із Жк-Дисплеєм як видошукач.

Досить цікава 6-мегапіксельная модель S10, виконана в поворотному конструктиве, ніколи "улюбленому" компанією Nikon. Корпус камери розділений на дві частини з'єднаних вертикальним шарніром. Об'єктивна половинка може повертатися щодо основний, що дозволяє знімати під різними кутами й з різних ракурсів.

Старша модель лінійки S500 оснащена потужним трансфокатором, оправа якого виступає за межі корпусу камери, і 7, 1-мегапіксельним сенсором.

Камери модельного ряду Coolpix Sxxx годяться й для починаючого фотолюбителя, і для фотографа-аматора середньої кваліфікації. А для поїздок і подорожей ці камери просто незамінні, оскільки гранично компактні, міцні й дозволяють домогтися високої якості карток.

Модельна лінійка Nikon Coolpix Pxxx – усього три камери. Але які камери! Сама молодша P4 наділена сенсором з дозволом в 8, 1-мегапікселя, потужною оптикою й традиційним (запозиченим у частині ергономіки в плівкових камер) корпусом. Відмінний інструмент для творчості, оскільки тут є й оптична стабілізація об'єктива (захист від мимовільних зсувів і тремтіння рук), і режим пріоритету діафрагми (можна вручну вибрати витримку), і відмінна оптика.

Ще цікавіший P3. Незважаючи на молодший індекс, вона ледве сучасніша моделі P4, оскільки має вбудований бездротовий адаптер Wi-Fi (інші характеристики ті ж). Що це дає фотографові? Можливість переписувати знімки на комп'ютер, друкувати, обмінюватися знімками з іншими портативними пристроями без кабельного підключення. Унікальна функція, що важко переоцінити.

Нарешті, модель Coolpix P5000. Самий зроблений компакт від Nikon, оснащений 10-мегапіксельним сенсором, 3, 5-кратним зумом з оптичною

стабілізацією, набором автоматичних і ручного режимів установки експозиції, що володіє рекордним для компактних камер швидкодією. Камеру P5000 можна сміло назвати спадкоємицею просьюмерок Nikon, що буде корисна й звичайному, і "професійному" фотолюбителеві (є такий умовний термін в області фотосправи).

Дзеркальні камери Nikon – особлива тема. Моделей істинно аматорського класу всього три – D40, D50 і D70s (інші моделі напівпрофесійного й професійного рівня, їх ми розглядати не будемо через високу вартість і їхнє недвозначне призначення). Nikon D40 – сама доступна дзеркальна камера для фотолюбителів, оснащена повнорозмірним сенсором формату APS (тобто зменшеним в 1,5 рази в порівнянні зі стандартним кадровим вікном плівкової камери). У порівнянні з камерою D50 у молодшої моделі немає цілого ряду допоміжних механізмів і спрощений сам фотоапарат. Приміром, у цій камері немає кнопки закриття діафрагми для оцінки ГРИП (глибини різкого зображення). У корпус камери не вбудований мотор привода автофокусування –тому з камерою в режимі автофокуса працюють тільки спеціальні об'єктиви з вбудованим мотором. Зменшено кількість датчиків (до 3-х) у сенсорі автофокуса. Nikon D40 оснащується 6-мегапіксельним CCD сенсором виробництва Sony, а новітня модифікація D40x – 10-мегапіксельної матриці.

Камера D50 займає в цьому ряді середнє положення. Від D40 її відрізняють більші розміри й наявність другого монохромного дисплея на верхній панелі фотоапарата (плюс уже відзначені відмінності). А від D70s – застосування карт пам'яті SD замість Compact Flash і відсутність захисту контрольного дисплея спеціальною кришкою. В D50 і D70s вбудовуються 6-мегапіксельні сенсори CCD (теж виробництва Sony...

З п'ятірки провідних світових виробників плівкової апаратури в цифрову епоху на плаву залишилися чотири – Canon, Nikon, **Pentax** і **Olympus**. Для російських фотолюбителів особливий інтерес представляє Olympus, оскільки ця компанія крім чудових цифрових компактів випускає доступні за ціною цифрові дзеркальні фотоапарати.

Ці камери (самий яскравий приклад Olympus E-300 і E-400) відрізняються від цифрових дзеркальних камер інших виробників тим, що компанія Olympus відмовилася від сумісності фотоапаратів з оптикою для плівкових камер свого виробництва – у цих зеркалках використовується "своя", ні із чим більше несумісна оптика. Причина такого рішення – у застосуванні сенсора формату 4/3 дюйма (тобто 22,5 мм по діагоналі). Це вдвічі менше діагоналі 35-мм вузькоплівкового кадру. Отже, застосування оптики від плівкових камер Olympus можливо тільки з урахуванням дворазового збільшення фокусної відстані об'єктивів (кроп-фактор), що найчастіше навряд чи прийнятно.

Серед інших відмінностей дзеркальних камер від Olympus – наявність спеціального ультразвукового механізму самоочищення сенсора від пилу. А в компактних цифрових камерах цієї компанії використовуються карти унікального стандарту x-Picture Card, що доводиться враховувати при покупці камери.

Цифрові камери Pentax (і компакти, і дзеркальні, що випускаються японською компанією Asahi Optical) відрізняє висока якість об'єктивів і збалансованість характеристик. Не можна сказати, що компанія зберегла повною мірою свої позиції на ринку аматорської апаратури, але своє місце вона втримує міцно. Якість і продуманість конструкції камер Pentax як і раніше залучає до них чималу частину фотолюбителів...

На ринку цифрової фотоапаратури чимало гідних гравців. Частина з них – це великі електронні компанії, що освоїли нову для себе область. Приклад – камери під маркою **Panasonic Lumix**, над розробкою яких компанія Matsushita працює разом із прославленої Leica. Ці камери відрізняє фірмова оптика (саме під маркою Leica), сенсори власної розробки й виробництва, відмінне сполучення характеристик і, як наслідок – популярність у фотолюбителів. У модельному ряді компанії є й прості моделі, і "імеджеві" ультракомпакти, і просьюмерки, і дзеркальна камера (формату 4/3, як і Olympus).

Нарешті – **Sony**. Незважаючи на "не фотографічний" імідж найбільшого світового виробника споживчої електроніки, ця грандіозна по своїх масштабах компанія в області цифрової апаратури займає особливе положення. Саме Sony подарувала нам цифровий фотоапарат таким, яким ми його знаємо сьогодні. Саме Sony випустила на початку 80-х років минулого століття перші цифрові фотоапарати (знаменитий проект **Mavica**). Тим дивовижна величезна пауза між випуском перших камер і випуском на широкий ринок дзеркального фотоапарата під цим брендом. Ми говоримо про унікальну по сполученню характеристик 10-мегапіксельной цифровий "зеркалці" Sony DSLR-A100 Alpha. Ця камера – спадкоємиця цифровий зеркалки Minolta (моделі 5D). Відповідно, до неї підходить оптика від плівкових фотоапаратів Minolta.

Що ж стосується цифрових компактних камер Sony, та їхня кількість настільки велика, що навіть короткий опис займе не одну сторінку. Ці фотоапарати відрізняються високою якістю виготовлення й застосуванням карт пам'яті фірмового стандарту Memory Stick (різновиду – Duo, Pro Duo).

Крім згаданих компаній, цифрову фотоапаратуру випускають і інші гранди виробництва електроніки, наприклад, Samsung...

Марка камери значення, звичайно, має. Але не менш важливі й характеристики конкретної моделі. Цим характеристикам ми приділимо особливу увагу, але спочатку придивимося до основних вузлів цифрового фотоапарата. Хоча б для того, щоб зрозуміти, про яких, властиво, характеристиках піде мова.

16.4. Сенсори цифрових фотоапаратів

Матриця світлочутливих елементів – основний вузол цифрового фотоапарата. Зрозуміти принцип його роботи – зрозуміти принцип самої цифрової фотографії. У цій маленькій по фізичних розмірах мікросхемі осередок сучасних високих технологій.

Ціль лекції – розповісти про пристрій і принцип дії сенсорів CMOS і CCD. Тут же докладно розглядаються найважливіші характеристики світлочутливих сенсорів.

Якісний рівень сучасного цифрового фотоапарата визначається, насамперед, технічною досконалістю встановленого в ньому сенсора – матриці світлочутливих елементів. Це найдорожча й найбільш значима деталь цифрової камери.

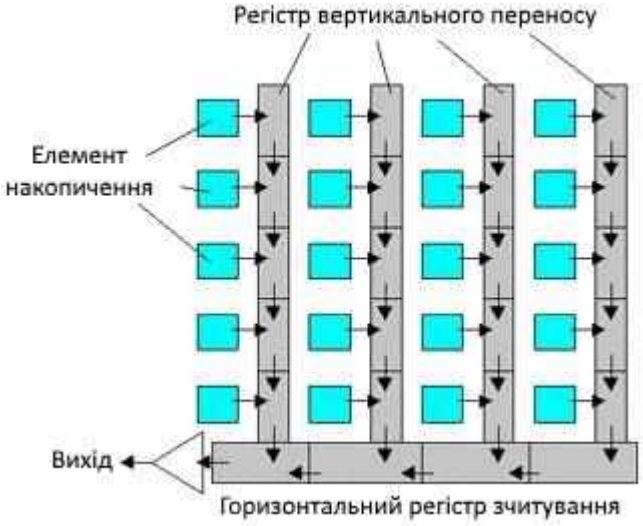


Рис. 16.23. Сенсор CCD цифрового фотоапарата

На сьогоднішній день у виробництві світлочутливих сенсорів застосовуються дві конкуруючі технології. Перша, більше проста у виробництві й по ряду ознак більше перспективна – технологія **CMOS** (**Complementary Metal-Oxide-Semiconductor**). У перекладі ця технологія називається **КМОП - комплементарний метал-окисел-напівпровідник**. У силу різних причин сенсори, побудовані за технологією CMOS, установлюються у фототелефони й у дзеркальні камери Canon і Sony.

Лідируючої на ринку цифрової фототехніки є технологія **CCD** (Charge-**Coupled Device**). У російському перекладі цей тип сенсорів називається **ПЗС – прилад із зарядовим зв'язком**. Більше трудомісткі у виробництві, сенсори CCD, проте, установлені в переважній більшості цифрових фотоапаратів аматорського й професійного класу.

У спрощеному виді принцип дії матриці світлочутливих елементів цифрового фотоапарата виглядає в такий спосіб. Сенсор CCD складається з **підкладки**, виготовленої з монокристалічного напівпровідникового матеріалу, що ізолює шару **окису**, що покриває підкладку, і набору мікроскопічних (мікронних розмірів) металевих провідників-**електродів**. До електродів матриці підводить електричний струм. Засвітка поверхні матриці приводить до того, що сила струму (заряд) на виводах електродів змінюється, тобто кожний осередок світлочутливої матриці реагує на інтенсивність засвітки. Ці зміни зчитуються електронною схемою фотоапарата, і на їхній основі будується картинка, що відповідає сфальцьованому на поверхні сенсора зображенню.

Осередку матриці, побудованої за технологією CMOS, це польові **транзистори**, які при засвітці змінюють свій стан, перешкоджаючи проходженню електричного струму через виводи осередку або, навпаки, підсилюючи сигнал. Електронна схема фотоапарата зчитує зміни стану осередків матриці й на їхній основі будує картинку.

Матриці CMOS у порівнянні з матрицями CCD відрізняються зниженим енергоспоживанням і високою технологічністю. З іншого боку, дозвіл матриць CMOS, їхня світлочутливість, динамічний діапазон і стійкість до шумів нижче, ніж у матриць CCD. Це пояснюється складністю пристрою, а також зниженою світлочутливістю польових транзисторів у порівнянні з осередками із зарядовим зв'язком.

Установлювані в стільникові камерофони сенсори CMOS виконані у вигляді великої гібридної мікросхеми, на кристалі якої змонтовані багато сервісних схем вбудованого в телефон фотоапарата. Це й аналого-цифровий перетворювач (**АЦП**), і електронний затвор (схема миттєвого зчитування стану матриці), схеми балансу білого й стиску зображень. У масовому виробництві CMOS-Сенсори виявляються дешевше, оскільки кожний елемент матриці більший, чим осередок сенсора CCD. А найпростішим камерам на основі CMOS-Сенсорів не потрібні багато допоміжних електронних механізмів. По суті недавно ще популярна, а сьогодні зійшла зі сцени дешева веб-камера з функцією автономної роботи як цифровий фотоапарат складається з корпуса, батарейного блока живлення, простого об'єктива, невеликого набору пасивних елементів (узгоджувальних резисторів, порту USB, пари кнопок), монохромного символьного дисплея й однієї мікросхеми, на яку покладена вся робота з оцифровки й обробки зображень. Звідси й надзвичайно низька ціна подібних фотокамер.

Говорячи про перспективи сенсорів CMOS, не варто забувати, що це дуже молода технологія. Вона виникла, як альтернатива трудомісткої й малоефективної технології сенсорів CCD. Досить сказати, що вихід придатної продукції при масовому виробництві матриць CCD ще шість-сім років тому перебував на рівні двох відсотків. Позначаються розміри елементів (порядку тисячних часток міліметра) і дуже високі вимоги до технологічних допусків.

У той же час, конструктори дзеркальних цифрових фотоапаратів Canon і просьюмерок Sony (приклад – камера Sony DSC-R1) установлюють у свої фотоапарати саме сенсори CMOS, доповнюючи їхніми спеціальними схемами придушення шумів. Ще одна позитивна сторона матриць CMOS – їхня стабільність і довговічність. Причина, знову ж, у застосуванні як світлочутливі елементи польових транзисторів, у більших розмірах кожного елемента й у високій технологічності масового виробництва...

Мікроскопічні осередки світлочутливої матриці здатні відреагувати тільки на силу світла, що попадає на них (на інтенсивність світлового потоку). Для того, щоб одержати зображення, що наближається по якості до плівкового фотознімка, цифровий фотоапарат повинен розпізнавати ще й колірні відтінки.

Але перш ніж говорити про технологію оцифровки кольорового зображення, варто помітити, що для збільшення точності роботи матриці

(поліпшення співвідношення сигнал/шум) і підвищення світлочутливості, кожний осередок забезпечується мікролінзами, що збирають, фокусуючих світловий потік. Особливо це стосується матриць CMOS, де без подібних лінз необхідної якості зображення домогтися важко.

Одержати кольорове зображення, і ми про цього вужа говорили, можна різними способами. У професійній знімальній апаратурі застосовується схема із трьома світлочутливими матрицями. Сфальцьоване об'єктивом зображення розщеплюється спеціальною призмою на три ідентичних світлових потоки, кожний з яких засвічує свою матрицю через світлофільтр одного з базових кольорів – червоного, зеленого й блакитного (RGB – Red, Green, Blue). Ця технологія дозволяє домогтися високої якості передачі кольору, але ускладнює конструкцію камери й відбивається на її вартості. Найчастіше три матриці встановлюються в дорогих цифрових відеокамерах.

У фотоапаратах же (крім професійних камер спеціального призначення) використовується інша технологія – з одним сенсором. Над поверхнею сенсора встановлений блок мікроскопічних світлофільтрів, розташованих у шаховому порядку відповідно до колірної моделі Байера. Цей алгоритм побудови кольорового зображення має на увазі подвоєна кількість зелених фільтрів у порівнянні із червоне й синіми, оскільки людське око більше чутливе до зеленої частини світлового спектра. Кольорове зображення будується електронікою камери вже після перетворення аналогового електричного сигналу, що знімається з осередків сенсора камери в цифровий код аналого-цифровим перетворювачем АЦП (якщо говорити про сенсори CCD, сенсори CMOS самі можуть обробляти колірну складову сигналу, оскільки звичайно це більші багатофункціональні мікросхеми).

В одержанні колірної інформації беруть участь всі експоновані елементи осередку. При обробці її застосовуються складні методи **інтерполяції**. Зокрема, ураховується колірна складова сусідніх елементів (пікселів). У результаті електроніка камери видає реалістичну повноколірну картинку максимально відповідному дійсному зображенню.

Щоб краще зрозуміти цей механізм, звернемося до іншої технології (тобто підемо від протилежного механізму), до принципів побудови кольорового зображення кольорових рідкокристалічних моніторів. Там всі елементи РК матриці, що представляють собою мікроскопічні капсули з рідкими кристалами, утворені із трьох субелементів, кожний з яких прикритий світлофільтром трьох базових кольорів – червоним, зеленим і синім. Зображення будується методом додавання кольорів. Залежно від інтенсивності світіння субелемента піксель (осередок матриці) здобуває той або інший колірний відтінок (при цьому сам осередок з рідкими кристалами не світиться, а лише перекриває або пропускає світловий потік лампи підсвічування екрана).

Приблизно так само працювали КМОП сенсори перших цифрових фотоапаратів і найдешевші веб-камери. Тобто кожний елемент матриці був прикритий індивідуальним мікроскопічним світлофільтром. Але в сучасних сенсорах ця технологія не застосовується. У побудові картинки беруть участь всі задіяні осередки (але не всі осередки сенсора – частина з них виконують

сервісні функції, тому говорять про повний і ефективний дозвіл сенсора). Це стосується й сенсорів CCD, і сенсорів CMOS – у всякому разі, сучасних, які встановлюються в цифрові фотоапарати, що випускаються сьогодні...

Ще одна важлива деталь пристрою світлочутливих матриць – спосіб **реєстрації** зображення. Матриця CCD складається із двох ідентичних наборів осередків – світлочутливих елементів, що утворять **секцію нагромадження зарядів**, і елементів, що утворять **секцію зберігання зарядів**. Електричні заряди, які виникають при опроміненні елементів сенсора світловим потоком, надходять в осередки секції нагромадження й переміщаються в осередки секції зберігання зарядів, звідки потім через **регістри вертикального зрушення** – у вихідний **підсилювач** матриці. Осередку секції зберігання прикриті світлонепроникним фільтром, а тому на світловий потік не реагують. Але при переході зарядів із секції нагромадження в секцію зберігання варто ізолювати непроникною заслінкою й світлочутливі осередки, щоб не змішувати виниклі при опроміненні нові заряди із уже збереженими (інакше зображення просто не вийде), використовуючи для цього в цифрових фотоапаратах звичайний електромеханічний затвор.

Ця особливість стосується матриць із **порядковим** переносом зарядів. У фотоапаратах з матрицями з **покадровим** переносом зарядів затвор виявляється непотрібним, оскільки заряди осередків накопичуються відразу з усією поверхні матриці й із сигналами осередків нагромадження не змішуються. При цьому регістрами вертикального зрушення, що представляють собою прості провідники, забезпечується кожний світлочутливий осередок секції нагромадження зарядів, а світлоізольована секція зберігання зарядів займає окрему область сенсора. Проблема в тім, що застосування покадрового переносу зарядів збільшує розміри сенсора й у той же час зменшує його дозвіл. Тому сьогодні застосовується технологія комбінованого построково-**кадрового** переносу. Що дозволяє, з одного боку, одержувати постійний сигнал з матриці й використовувати його для побудови зображення на вбудованому контрольному дисплеї фотоапарата, а з іншого боку - одержувати високоякісні зображення з порядковим зчитуванням зарядів і застосуванням електромеханічного затвора.

У фотоапаратах із сенсорами CMOS (у найпростіших з них і у фототелефонах) електромеханічний затвор не застосовується зовсім, оскільки інформація про стан осередків подібної матриці зчитується безпосередньо з виводів польових транзисторів, що утворять матрицю. Однак, у дзеркальних фотоапаратах із сенсорами CMOS для точного відпрацьовування експозиції електромеханічний затвор установлюється обов'язково.

Елементарні відомості про принцип дії сенсорів CCD важливі для фотолюбителя й із практичної точки зору. Справа в тому, що при покупці нового фотоапарата, поза залежністю від рівня техніки і її вартості, у фотографа, як це ні сумно, завжди їсти шанс догодити на камеру з "битими" пікселями. "Битий" піксель – це світлочутливий елемент, у силу різних причин втративши здатність реагувати на світлове опромінення. При цьому бездіяльний осередок може бути зовсім непомітна на знімку, якщо перебуває в

нижній частині матриці, на яку доводиться та частина кадру, де звичайно розташовується земля, де мало світлих ділянок, на яких одна чорна крапка може бути добре помітна. Інша справа верхня частина матриці, де зображується небо й інші світлі об'єкти.

Один-два битих пікселя річ для аматорської камери звичайна. Гірше, коли таких пікселів трохи й вони об'єднані в групу. Тоді темна крапка на знімку стає помітної навіть при зйомці з найвищим дозволом, коли в побудові зображення беруть участь всі світлочутливі осередки матриці. У магазині биті піксели матриці побачити дуже важко, а в ході практичної роботи з камерою подібна неприємність буде неодмінно виявлена...

Ми вже говорили про те, що технічна досконаліть камери визначається якістю сенсора, а як і чим визначається якість самого сенсора? Існує ряд характеристик, що мають для світлочутливої матриці цифрового фотоапарата першорядне значення.

У першу чергу цей дозвіл матриці. Чим більше дозвіл матриці світлочутливих елементів, тим вище якість кінцевого паперового відбитка або електронного фотозображення. Кількість ефективних пікселів на матриці цифрового фотоапарата визначає дозвіл оцифрованого зображення, оскільки максимальний дозвіл знімка дорівнює кількості пікселів світлочутливого сенсора.

Іноді у зведенні технічних характеристик (це стосується тільки найдешевших камер) можна виявити, що максимальний дозвіл фотознімків перевищує кількість пікселів матриці фотоапарата. Цим заявам виробників не варто довіряти. Підвищений дозвіл досягається програмною інтерполяцією, коли відсутні елементи зображення синтезуються на основі усереднених значень яскравості сусідніх пікселів. Інтерполяція збільшує кількість пікселів, але завжди за рахунок реальної якості зображення. На інтерпольованому знімку границі об'єктів втрачають різкість і погіршується деталізація. У найкращому разі дозвіл зростає без якого б те не було поліпшення зображення. Тому різного виду інтерполяційну обробку, якщо така функція присутня в камері, краще не використовувати зовсім.

Дозвіл (або роздільна здатність) світлочутливого сенсора, як і дозвіл екрана монітора (і, до речі, контрольного дисплея фотоапарата), виражається в пікселях. При цьому екранний дозвіл монітора визначається величиною світної екранної крапки – пікселя, утвореного люмінофором електронно-променевої трубки або осередком рідкокристалічної матриці (у цьому випадку світиться не елемент, а лампа за ним). Екранний дозвіл величина постійна. Воно залежить тільки від розміру пікселя екрана монітора. Стандартні значення величини пікселя 0,25, 0,28 і 0,31 мм. Стандартні значення екранного розширення комп'ютерних моніторів 72 або 96 пікселів на квадратний дюйм.

Дозвіл світлочутливих сенсорів теж залежить від розміру пікселя, але яких-небудь стандартів тут не існує. Чим менше величина кожного пікселя, тим більше пікселів (тобто світлочутливих елементів) розміщається на поверхні матриці й, як наслідок, тим більшим дозволом володіє сама матриця.

Крім екранного розширення й розширення сенсора цифрової камери існує ще одна важлива характеристика – дозвіл друкувального пристрою (у практиці аматорської цифрової фотографії, як правило, кольорового струминного принтера). Однак дозвіл принтера виміряється не в пікселях, а в крапках. А виражається дозвіл принтера в крапках на квадратний дюйм. Для фотолюбителя принципової різниці між пікселем і крапкою ні, тому обидві одиниці виміру розв'язної здатності можна вважати рівнозначними. Розходження між крапкою й пікселем носять теоретичний характер, що має значення для розроблювачів цифрової техніки. Уважається, що піксель має правильну прямокутну форму, близьку до квадрата (на практиці екранні пікселі можуть мати форму кола або витягнутого по вертикалі прямокутника залежно від типу електронно-променевої трубки – із щілинною маскою або з апертурними ґратами відповідно, а піксель матриці цифрового фотоапарата може мати форму квадрата або восьмикутника – наприклад, осередку матриці SuperCCD камер Fujifilm FinePix). Принтерна ж крапка має неправильну форму близьку до кола...

Сучасна цифрова камера для більш-менш серйозних занять аматорською фотозйомкою – це фотоапарат із сенсором **дозволом** не менш 4 мегапікселя. Чому не 2 мегапікселя (не говорячи вже про менші значення розширення)? Справа в тім потенціалі, що закладено у фотографію високого розширення. Знімок з дозволом в 2272x1704 пікселя (нормальний дозвіл для сенсора в 4,23 мегапікселя) легко піддається кадруванню. Його можна без яких би те не було втрат вивести на екран у віконному й повноекранному режимі. Нарешті, фотографію можна роздрукувати на папері, одержавши відбиток 10x15 див дуже гарної якості й навіть формату A4 (стандартний альбомний аркуш) при цілком прийнятній якості. Зі знімком дозволом 1600x1200 пікселів (нормальний дозвіл для сенсора в 2 мегапікселя), як би добре він не виглядав (а виглядає він і справді чудово), подібні маніпуляції досить проблематичні.

Друга найважливіша характеристика світлочутливого сенсора – його **фізичний розмір**. Він виміряється по діагоналі й позначається в частках дюйма. При цьому форма сенсора, як правило, прямокутна зі співвідношенням сторін 4:3, хоча в магазинах уже можна купити фотоапарати із широкоформатними сенсорами зі співвідношенням сторін 16:9 (подібні камери випускає Samsung, Kodak, Matsushita і інші компанії).

Тут же криється одне історично сформоване не співвідношення. Справа в тому, що співвідношення сторін стандартного для більшості країн паперового відбитка 10x15 сантиметрів рівняється 2:3 (або 3:2, це не принципово). Чому ж сенсори мають співвідношення сторін 4:3, адже при печатці знімка на папері частина зображення, краю картинки, будуть обрізані? Так, будуть. Це неминучі витрати цифрової фотографії. Справа в тому, що розміри сенсора оптимізовані під вивід знімка на комп'ютерний монітор у повноекранному режимі. А співвідношення сторін стандартного комп'ютерного монітора саме й відповідає співвідношенню 4:3. Широкоекранні монітори мають співвідношення сторін 16:9. А фотопапір – 3:2... Правда, у деяких камерах (приклад – камери

Panasonic, Canon серії Digital IXUS і інші) дозволяють зберігати знімки зі співвідношенням сторін 3:2, але ціною зменшення розширення.

Чим більше фізичний розмір сенсора, тим він працює точней і ефективніше. Сенсор розміром в 1/1,8 дюйма краще, ніж сенсор розміром 1/3,2 дюйми, оскільки на більшій площі кристала вміщається більша кількість світлочутливих осередків (виходить, вище й дозвіл). Більше того, при однакових значеннях розширення сенсор більшого розміру краще, ніж сенсор меншого розміру. У цьому випадку осередку сенсора мають більші розміри, значить і такі параметри оцифровки зображення, як динамічний діапазон і стійкість до шумів, вище.

Розмір сенсорів дзеркальних камер виміряється в міліметрах по сторонах кадрового вікна. Справа в тому, що величина матриці цих фотоапаратів впритул наближається до стандартного розміру кадрового вікна вузькоплівкової 35-мм камери, тобто 36х24 мм, і в загальних випадках відповідає плівковому стандарту APS. Це дозволяє використовувати на цифровому фотоапараті сумісну оптику від плівкових камер того ж виробника. Але при цьому варто враховувати зміну фокусної відстані змінних об'єктивів – той самий кроп-фактор. Приміром, на фотоапаратах Canon EOS 400D із сенсором розміром 28,7 мм по горизонталі й 19,1 мм по вертикалі зміна фокусної відстані всієї лінійки об'єктивів для плівкових камер Canon буде кратна 1,6 одиниць убік збільшення – рівно настільки, наскільки матриця камери менше стандартного кадрового вікна плівкового фотоапарата. В аматорських дзеркальних камер Nikon кроп-фактор менше – 1,5, а в Olympus і Panasonic (вони ставляться до так званого стандарту 4/3, оскільки їхні сенсори менше по розмірах, чим розміри кадру APS) більше – 2. Тобто нормальний 50-мм об'єктив на цифровій дзеркальній камері Nikon D50 (приводимо для прикладу) буде мати фокусна відстань в 75 мм (виходить не нормальний універсальний об'єктив, а помірний телевик, що підходить для портретної зйомки).

Наступна вкрай важлива, але досить тяжко визначена без спеціального устаткування характеристика світлочутливих сенсорів – співвідношення **сигнал/шум**. У тім або іншому ступені шумлять будь-які сенсори, включаючи й самі на сьогоднішній день зроблені. Колірні шуми проявляються на знімку у вигляді дрібних пофарбованих крапок (артефактів) у тінях і у вигляді колірних ореолів навколо контурів фігур на границях контрастних переходів. Боротися із шумами дуже важко. Якщо буде потреба застосовуються спеціальні фільтри – утиліти, що працюють у програмному середовищі графічного редактора Adobe Photoshop. Фільтри здатні до деякої міри зм'якшити шуми, заміщаючи артефакти крапками з усередненими значеннями кольору і яскравості. Якщо фільтри не здатні забрати всі артефакти й ореоли, доводиться правити знімок вручну, сильно збільшуючи й ретушуючи елементи зображення.

Жоден спосіб виправлення зображення не дає стовідсоткового рятування від шумів. Тому фотолюбителеві, якщо якість знімків для нього справді значимо, доводиться додержуватися елементарних правил як при виборі камери, так і при практичній зйомці. Перше правило – не здобувати дешевих

камер з матрицями низької світлочутливості. Схильність до шумів найбільш властива сенсорам CMOS, які встановлюються у фототелефони, і сенсорам CCD найдешевших компактних камер, у які звичайно вбудовуються сенсори попереднього покоління, так до того ж і дуже невеликі по розмірі.

Шумлять сенсори й у більше серйозних камер. Технологія цифрової фотографії дуже молода, а тому виробництво сенсорів бурхливо вдосконалюється. Покоління матриць поміняють один одного швидше, чим морально застарівають конкретні моделі фотоапаратів (а вони застарівають досить швидко, протягом приблизно двох років). Сенсори дозволом в 4 мегапікселя, які встановлювалися в камери середньої й навіть старшої групи два-три роки тому сьогодні застосовуються в недорогих аматорських фотоапаратах. Їхнє місце займають матриці з підвищеними характеристиками, у тому числі й по стійкості до шумів. Отже, вибирати треба ту модель, що випускається не занадто довго, не більше року. Тоді у фотографа будуть підстави припускати, що сенсор його камери схильний до шумів у мінімальному ступені.

Друге правило стосується зміни світлочутливості сенсора. У більшості цифрових фотоапаратів з досить розвиненими сервісними функціями світлочутливість установлюється як автоматично, так і вручну. У режимі auto комп'ютер фотоапарата сам вибирає значення світлочутливості сенсора залежно від рівня освітленості об'єкта, що знімається, і встановленого програмного режиму роботи камери. Наприклад, у режимі "нічний портрет" світлочутливість буде обрана максимальної, а у звичайному режимі – мінімально можливої (якщо дозволяє висвітлення).

Чим вище значення світлочутливості матриці, тим вона більше шумить. Отже, краще зовсім відмовитися від застосування автоматичного режиму й виставити селектор вибору значення світлочутливості сенсора на мінімальне значення, оскільки мінімальне значення відповідає реальній світлочутливості сенсора. У цьому режимі не задіяні електронні схеми посилення сигналу, які вносять додаткові перекручування й приводять до появи артефактів у тінях. Якщо ж умови висвітлення такі, що автоматика камери встановлює занадто тривалу витримку, з якої неможливо знімати без штатива, то значення світлочутливості можна збільшити, але при цьому треба бути готовим до того, що рівень шумів істотно підвищиться.

І ще одне правило, що полягає в тім, що не слід пред'являти до цифрового фотоапарата завищених вимог. Те, що під силу високоякісній фотоплівці, цифровому фотоапарату не під силу в принципі. Цифровий фотоапарат не здатний знімати в умовах занадто низької освітленості без застосування джерел штучного світла, імпульсних фотоспалахів або ламп накалювання. А професійна плівка світлочутливістю в 3200 одиниць ISO витягне знімок навіть при світлі однієї свічі (причому, у буквальному значенні).

Дві якісні характеристики, що прямо впливають на результат зйомки – **динамічний діапазон** сенсора й **розрядність** подання **кольору**. Перша із цих характеристик відбиває здатність матриці передавати світлові відтінки, друга ставиться не тільки властиво до сенсора, але й до аналого-цифрового

перетворювача, що переводить електричні сигнали з виводів матриці в цифровий код.

Динамічний діапазон – це кількість відтінків сірого (тобто рівнів яскравості), які здатний розрізнити світлочутливий матеріал (фотоплівка або сенсор цифрової камери) між абсолютно чорним і абсолютно білим кольором. Чим вище динамічний діапазон, тим вище вірогідність зображення на експонованому носії. Найвищим динамічним діапазоном володіє негативна фотоплівка. Тому дотепер, незважаючи на досягнення цифрових технологій, для демонстрації фільмів у кінотеатрах використовуються звичайні плівкові, а не цифрові проектори.

Серед цифрових пристроїв найбільшим динамічним діапазоном володіють барабанні сканери, які застосовуються в поліграфії й коштують десятки тисяч доларів. Динамічний діапазон планшетних сканерів CCD набагато менше, але ще менше динамічний діапазон сенсорів цифрових фотоапаратів. У найдорожчих професійних фотоапаратів цей показник лише наближається до рівня фотопаперу на основі галогенідів срібла (а її динамічний діапазон, відповідно, у десять разів менше діапазону фотоплівки).

Якість передачі кольору цифрового фотоапарата виражається розрядністю кольору. Розрядність кольору – це сума значень розрядності оцифровки кожного колірного каналу. Приміром, кожний колірний канал більшості матриць цифрових фотоапаратів аматорського класу здатний зафіксувати 256 відтінків (або градацій) сірого, що становить 8 біт. У цьому випадку розрядність сенсора буде 8+8+8=24 біта, по 8 біт на кожний колірний канал (червоний, зелений, голубий). У принципі, 24-бітного подання кольору цілком достатньо для одержання якісного фотознімка, оскільки в цьому випадку АЦП камери видають знімок, що містить 16,7 млн. колірних відтінків. Але в продажі можна зустріти камери як з більше високою розрядністю кодування кольору по 10 або 12 біт на канал, так і з низкою – по 4 або 6 біти на канал. Надлишкова розрядність до 36 біт(тобто по 12 біт на канал) використовується в професійних камерах, призначених для одержання знімків з максимально достовірною передачею кольору. Хоча сьогодні сенсорами з підвищеною розрядністю колірного кодування оснащують і камери аматорського класу. А матриці зі зниженою розрядністю в 12 або 16 біт установлюють у бюджетні стільникові фототелефони.

Матриця світлочутливих елементів не тільки сама складна й найдорожча деталь цифрового фотоапарата, але й сама уразлива. Вона піддана старінню (електрохімічному зношуванню) і, як наслідок, змінам світлочутливості, а також, як видно, виходу з ладу окремих осередків. Якщо на природне старіння матриці власник фотоапарата не може вплинути ніяк, то можливість уберегти сенсор від небажаних впливів навколишнього середовища й, тим самим продовжити термін служби фотоапарата в цілому, у нього є.

Як будь-який складний електронний пристрій, що складається з безлічі мікроскопічних елементів, сенсор цифрової камери боїться різких температурних перепадів, при яких у матеріалі підкладки й плівкових шарів оптичних фільтрів виникають внутрішні деформації, а на поверхні сенсора

утвориться конденсат. Якщо плівкова камера, особливо механічна, здатна працювати при дуже низьких температурах, то цифровий фотоапарат при негативних температурах працювати не буде. По-перше, навіть на легкому морозі сенсор цифрової камери може змінити світлочутливість убік зменшення. По-друге, зображення на вбудованому контрольному дисплеї стане занадто світлим і малоконтрастним, щоб користуватися дисплеєм як видошукач. По-третє, постраждають елементи живлення (літієві акумулятори при температурі мінус 10 градусів можуть попросту вибухнути).

Якщо виникає необхідність знімати цифровою камерою при низьких температурах, варто подбати про надійний захист фотоапарата. Камеру варто тримати в теплі, під верхнім одягом, виймаючи фотоапарат для зйомки й відразу ховаючи його під шубу або пальто. Робота зі штативом або неквапливе кадрування виключаються. У крайньому випадку варто скористатися утепленим хутряним або тканевим чохлом. Але при цьому треба пам'ятати, що остигла камера при переміщенні в тепло (навіть під шубу) відразу покриється крапельками вологи. Із замерзлої камери треба негайно видалити елементи живлення або акумулятор і забрати фотоапарат у чохол до того моменту, поки температура не вирівняється. У противному випадку на поверхні сенсора й лінзах об'єктива можуть утворитися краплі вологи, які приведуть до короткого замикання електричних кіл камери й інших неприємностей.

Тема 17. ПОДАННЯ САЙТУ

У даній лекції розглядаються різні види руху: зменшення швидкості руху, додавання прискорення й коливання руху, покадровий рух. Також описуються можливості керування рухом, приводиться навчальний приклад.

17.1. Рух

Однієї з основних особливостей Flash є можливість руху об'єктів; об'єкти, що рухаються, можна додавати до всього, що створюється в Flash. У цій лекції ми розглянемо деякі зі способів створення в Flash MX різних типів руху.

Деяким читачам ці принципи вже можуть бути знайомі, однак ми також будемо вивчати нові можливості Flash MX, що дозволяють забезпечувати рух найбільш простими й ефективними методами, ніж це було колись. Щоб продемонструвати гнучкість Flash, розглянемо кілька різних типів руху, що ви, можливо, захотіли б додати у свої проекти, а також способи його застосування. Іноді ми будемо використовувати ActionScript для одержання наступних типів руху:

- зменшення: зменшення швидкості руху;
- свінг: додавання прискорення й коливання руху;
- покадровий рух: досягнення руху з використанням кадрів.

Спочатку розглянемо, як можна використовувати ці типи руху для переміщення об'єктів зі слідом від руху по осях X і Y. Також оборотний увага на те, як застосувати цей рух до інших параметрів, таким як розмір об'єкта.

Крім самого руху, ми розглянемо різні способи застосування руху за допомогою ActionScript: як викликати рух клацанням миші і як краще підтримувати рух, тобто як почати його і як завершити.

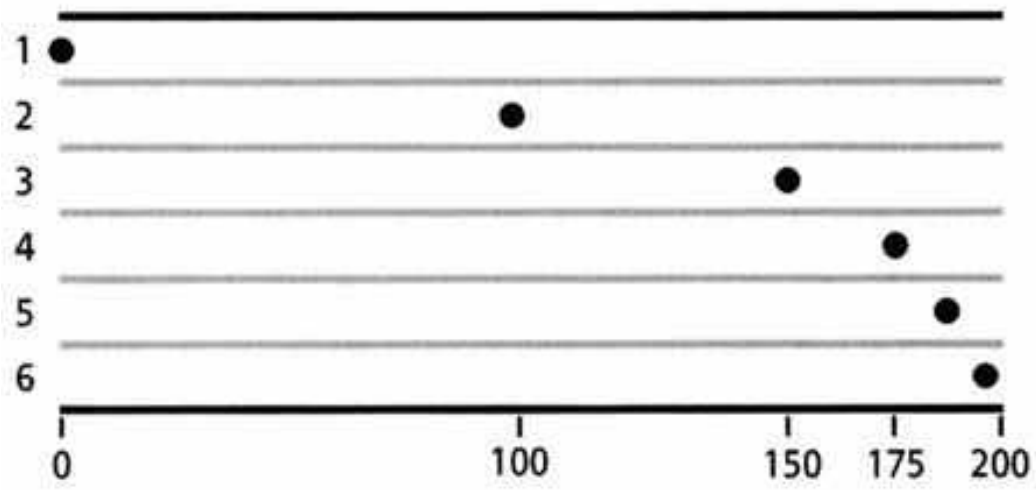
Нарешті, зупинимося на навчальному прикладі, що містить кілька картинок, що з'являються в області перегляду при клацанні на кнопці. Підходи, описані у вправі, містять у собі створення маски за допомогою ActionScript, а також динамічне застосування кнопки.

Отже, жарту убік! Почнемо з розгляду покадрового руху.

17.2. Зменшення швидкості руху

Першим типом руху, про яке ми б хотіли розповісти, є зменшення швидкості руху: об'єкт спочатку рухається досить швидко й поступово вповільнюється, поки, нарешті, не зупиняється досягши мети. Цей тип руху, імовірно, найбільш простий у реалізації – потрібно з'ясувати, як далеко перебуває об'єкт від мети й потім перемістити об'єкт до мети, розділивши дистанцію на частині. Отже, якщо об'єкт перебуває на відстані 100 пікселів від кінцевої крапки, ми можемо перемістити його на половину цієї відстані, після чого він буде в 50 пікселях від мети, далі – в 25, потім в 12,5 і так далі. Рух при цьому буде плавним; адже ми звикли спостерігати поступову з об'єктів, а не переривчасті рухи.

На малюнку нижче даний рух показаний у дії, причому кожна лінія являє собою кадр переміщення об'єкта на 200 пікселів.



З кожним кадром відстань об'єкта до мети скорочується наполовину. У кадрі 1 об'єкт переміщається на 100 пікселів, у кадрі 2 – на 50 пікселів, у кадрі 3 – на 25 пікселів, потім на 12,5 пікселів, потім на 6,25, потім на 3,125 і так далі.

Розглянемо практичне застосування цього руху, створивши об'єкт, що випливає за мишею. Нічого нового й цікавого в цьому ні, але наше завдання – розібратися в принципі руху.

17.2.1. Створення кліпу зі зменшенням швидкості руху

1. Відкрийте новий файл Flash зі значенням розширення за замовчуванням (550x400). В Property Inspector установите колір тла на білий і частоту зміни кадрів на значення 31 кадр у секунду. (Ми будемо використовувати ці настроювання завжди при створенні файлу Flash у цій частині книги.)

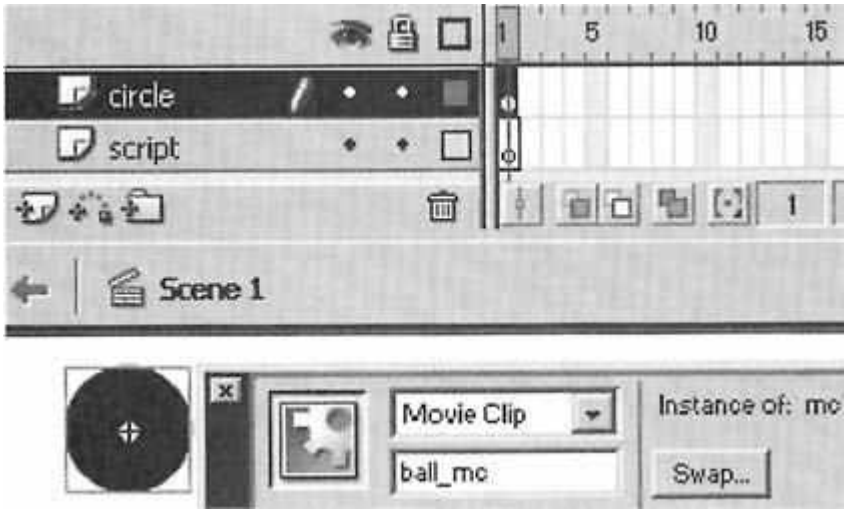
2. Намалюйте чорне коло, перетворіть його в символ кліпу (F8) і потім привласніть кліпу ім'я.

3. Відкрийте Property Inspector і привласніть інстансу* кліпу ім'я 'ball_mc'.

Ми використовуємо суфікс '_mc' в ім'ї відповідно до встановлених Macromedia перетворенням імен. Використання цього правила має дві переваги: по-перше, це допомагає відрізняти різні типи об'єктів у коді, а по-друге, у такий спосіб забезпечується поява підказки в коді потрібного типу при уведенні коду в середовищі розробки.

Ми проробили основні дії. Тепер поглибимося в процес роботи.

4. Тому що в більшості випадків краще містити дії в окремому шарі від іншого вмісту, додайте новий шар під поточним шаром. Краще завжди давати імена всім використовуваним шарам. Наші шари ми назвали просто circle і script.



5. Виберіть перший кадр першого шару й відкрийте панель Actions. Клацніть на маленькому білому значку параметрів у правому куті панелі й переконаєтеся, що відзначено параметр Expert Mode. Кращим способом швидкого ознайомлення з кодом є його введення, це може бути неприємно, але корисно.

6. Отже, перейдемо до коду. Нам потрібно застосувати подію до нашого фільму, що у кожному кадрі перевіряє, де перебуває покажчик миші, і потім переміщається за цим покажчиком. В Flash 5 це простіше всього було б зробити виділенням фільму й прикріпленням до нього події onClipEvent (enterFrame).

Завдяки новій об'єктній моделі, в Flash MX цього можна більше не робити. Ми можемо динамічно прикріплювати події clipEvents до фільмів, що дозволить всьому або майже всьому коду перебувати в одному місці, а також дозволить маніпулювати цими подіями під час виконання, наприклад, включати й виключати їх або змінювати яку-небудь функціональність.

Це досягається за допомогою присвоєння функції керуючому елементу фільму onEnterFrame. Якщо фільм має функцію, привласнену до його керуючого елемента onEnterFrame, то він щораз буде виконувати цю функцію.

Уведіть наступний код у кадр 1 шаруючи script:

```
ball_mc.onEnterFrame = function() {
  this._x += (this._parent._ xmouse-this._x) /4 ;
  this._y += (this._parent._ ymouse-this._y) /4;
};
```

Цей код спочатку створює функцію, що привласнюється керуючому елементу onEnterFrame фільму 'ball_mc'. Код цієї функції спочатку з'ясовує, наскільки далеко перебуває об'єкт від покажчика миші, і потім додає чверть цієї відстані до позиції фільму. Якщо вам потрібно було б написати цей код, так сказати, своїми словами, то вийшло б наступне: "обчислити різницю між позиціями покажчика по осях X і Y, потім збільшити позицію X на чверть цього значення".

7. Збережете ваш фільм у файлі ball motion_01.fla і протестуйте його.

Ви побачите, що об'єкт треба за покажчиком миші, так само як ми показували на вихідній діаграмі: спочатку об'єкт рухається швидко, а потім

його швидкість зменшується. На цьому етапі має сенс спробувати застосувати інші числа замість 4, щоб побачити різні результуючі особливості руху. Якщо ви збираєтеся працювати із цим типом руху, ви повинні знати, як вплинути на нього. Але, насамперед, вам варто забрати число з функції й зробити його змінної, котра дозволить застосовувати різні числа небагато швидше.

17.2.2. Додавання змінної швидкості руху

Щоб було легше міняти швидкість руху об'єкта у вашім фільмі, додайте змінну швидкості (чим менше буде значення, тим швидше буде рухатися об'єкт).

1. Використовуючи ті ж попередні дії, що й у попередніх прикладах, за допомогою панелі Actions додайте наступний код у кадр 1 вашого шару script:

```
var speed = 4;

ball_mc.onEnterFrame = function() {
  this._x += (this._parent._ xmouse-this._x)/speed;
  this._y += (this._parent._ ymouse-this._y)/speed;
};
```

Зверніть увагу на те, що усередині функції onEnterFrame слово 'this' посилається на фільм, які ви назвали ball mc, тому навіть при звертанні до змінної швидкості з функції потрібно вводити this.parent.speed, щоб виявитися на тім рівні, на якому перебуває фільм. Рівень root level є рівнем, на якому визначена змінна speed.

2. Збережете ваш фільм як ball motion 02 speed var.fla і запустите його. Рух відбувається в такий же спосіб, однак тепер легше внести зміна в код.

17.2.3. Ініціалізація руху клацанням миші

Тепер ми значно змінимо код, щоб фільм переміщався в будь-які місця, де користувач клацне мишею. Аналогічно тому, як ми привласнювали функцію onEnterFrame фільму, ми можемо привласнити йому функцію onMouseDown. Ця функція буде виконуватися при кожному натисканні кнопки миші в будь-якому місці на головному фільмі. Отже, коли користувач клацає мишею, потрібно спочатку встановити позиції цілей фільму на місце розташування клацання, а потім виконати функцію onEnterFrame для переміщення на цю позицію:

1. Використовуючи настроювання кадру, застосовані в попередніх прикладах, скористайтеся панеллю Actions для додавання наступного коду в кадр 1 шаруючи script:

```
var speed = 4;
// set up target variables
ball_mc.target = 0
ball_mc.target = 0
```

```
ball_mc.moveToMouse = function() {
  this._x += ( target-this._x)/speed;
  this._y += ( target-this._y)/speed;
};

triggerMotion = function() {
  // reset target variables on click
  target = _xmouse;
  target = _ymouse;
  // set the movieclip in motion
  this.onEnterFrame = this.moveToMouse;
}
ball_mc.onMouseDown = triggerMotion;
```

Можна бачити, що цей ActionScript кардинальним образом відрізняється від попереднього ActionScript. Якщо раніше ми приєднували функцію безпосередньо до керуючого елемента onEnterFrame, то цього разу ми спочатку визначаємо функцію moveToMouse, і при клацанні мишею вираження this.onEnterFrame указує на цю функцію й викликає її в кожному кадрі. Функція сама по собі працює так само, як і колись, за винятком того, що переміщення відбувається не на позицію покажчика миші, а на значення змінних target і target, що є змінними місця розташування покажчика миші, значення яких визначаються при кожному клацанні мишею користувачем.

Цей підхід виглядає більше ефективним, тому що потрібно лише один раз визначити функцію, а потім активізувати її, указавши для неї onEnterFrame. Якщо згодом потрібно буде виключити рух, це можна зробити, видаливши this.onEnterFrame.

2. Збережете ваш фільм у файлі ball motion_03_click to move.fla і запустите його.

Ви побачите, що об'єкт переміщається на те місце, де ви клацнули мишею, і потім, дійшовши до цього місця, припиняє свій рух, що являє гарний приклад зіставлення з тим случаємо, коли кулька постійно випливала за мишею.

17.3. Додавання сліду від руху об'єкта

Одним з доповнень, що забезпечить чіткість руху об'єкта, є його слід, що залишається на тих місцях, де тільки що перебував фільм. У нашім випадку це корисно для розуміння того, як працює сповільнення руху, однак такі сліди можуть застосовуватися також і в інших обставинах: або як, що рухається значок, під час завантаження чого-небудь, або ж у вигляді свого роду розмитості об'єкта, викликаної його рухом. Такі сліди можуть бути легко створені за допомогою дуплікації фільму в кожному кадрі з використанням функції duplicateMovieClip. Для цього потрібно лише додати рядок коду, що збільшує значення _root і потім створює дублікат поточного фільму.

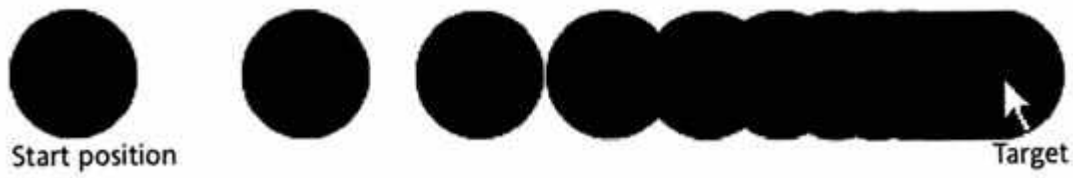
Дублікат перебуває за замовчуванням на тій же позиції, що й вихідний фільм. Насправді, дублікат має всі ті ж параметри, що й вихідний фільм, такі як alpha, xscale, yscale і т.д., однак у нього немає функцій onEnterFrame або змінних, які може містити вихідний фільм. Це одне з відмінностей від Flash 5, у якому якщо створювався дублікат фільму із привласненою подією clipEvent, також створювався дублікат цієї події, що відбувалося й на дублікаті фільму. Ми розташовуємо кожний дублікат на один рівень вище попереднього, за допомогою додавання змінної глибини для 'root', значення якої збільшується з кожним кадром.

DuplicateMovieClip можна інтерпретувати, як виробник дублікатів, зовні ідентичних вихідним фільмам, у яких, однак, не повторюються ніякі з вихідних дій. У нашім випадку, дублікати не випливають за покажчиком миші, а просто залишаються там, де вони були створені.

1. Використовуйте ті ж настроювання, як і в попередніх прикладах, і додайте наступний ActionScript у ваш фільм:

```
var speed = 4;

ball_mc.moveToMouse = function() {
  this._x += (this.target-this._x)/this._parent.speed;
  this._y += (this.target-this._y)/this._parent.speed;
  this.duplicateMovieClip("dupe"+this._parent.depth,
    this._parent.depth++);
};
ball_mc.onMouseDown = function() {
  this.target = this._parent._xmouse;
  this.target = this._parent._ymouse;
  this.onEnterFrame = this.moveToMouse;
};
```

2. Збережете фільм у файлі ball motion_04_click for trail.fla. Якщо тепер запустити фільм, ви відразу побачите, як працює ця з, і заметете, що це дуже схоже на те, що було зображено на діаграмі.



При клацанні на новому місці дублікати спочатку створюються досить далеко друг від друга, і потім стають всі ближче й ближче друг до друга доти, поки не перетворяться в монотонну картину в кінцевої крапки. Однак число об'єктів тут росте, поки фільм виконується, і, щоб уникнути "мішанини", потрібно поступово зменшувати кожний дублікат до його повного зникнення. У наступному прикладі ми реалізуємо саме це доповнення.

17.3.1. Зникнення сліду

Щоб не забивати фільм занадто більшою кількістю дублікатів того самого об'єкта, ми може привласнити кожному дублікату свою власну функцію enterFrame, що буде забезпечувати зменшення дублікатів з кожним кадром і потім видаляти їх по досягненні певного розміру.

Тому що ми збираємося працювати з кожним з дублікатів, має сенс привласнити об'єкту ball mc його власну функцію duplicate(). Приведемо код з виділеними новими частинами, що вводяться:

```
var speed = 4;

ball_mc.moveToMouse = function() {
  this._x += (this.target-this._x)/this._parent.speed;
  this._y += (this.target-this._y)/this._parent.speed;
  this.duplicate() ;
};
ball_mc.duplicate = function() {
  var dupe = this.duplicateMovieClip("dupe"+this._parent.depth,
  this._parent.depth++);
  dupe.onEnterFrame = this._parent.diminish
};
ball_mc.onMouseDown = function() {
  this.target = this._parent._xmouse;
  this.target = this._parent._ymouse;
  this.onEnterFrame = this.moveToMouse;
};
  function diminish(){
    // reduce scale gradually
    this._xscale = this._yscale-=2;
    //remove movieclip when scale goes below zero
    if (this._xscale <= 0) {
      this.removeMovieClip();
    }
  }
```

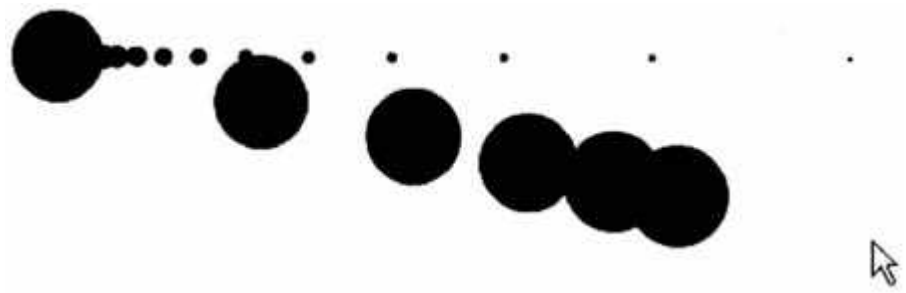
Отже, ми додали виклик функції duplicate() і значно змінили виклик duplicateMovieClip, винесувши вперед var dupe=. Незважаючи на те, що в словнику ActionScript зазначено, що duplicateMovieClip не повертає ніякого значення, насправді, вертається посилання на створений фільм.

Тут ми брали це значення й поміщали його в змінну dupe. Ми оголосили змінну за допомогою вираження var так, що це значення існує тільки під час виконання функції, і по закінченні її виконання воно зникає через непотрібність.

Тепер ми можемо використовувати dupe для посилання на наш новий дублікат фільму й для присвоєння йому функції onEnterFrame, функції diminish,

певної в _root. У функції diminish ми зменшуємо інкременти xscale і yscale з кожним кадром, і коли вони досягнуть нуля, фільм віддаляється.

Якщо запустити фільм зараз, буде набагато менше непотрібних фільмів, тому що кожний із кліпів виконує ті дії, які йому запропоновані.



17.3.2. Підвищення ефективності

Тепер все працює чітко, і кількість фільмів лежить у межах розумного. Однак ви, напевно, помітили, що фільм продовжує йти й створювати дублікати самого себе навіть по досягненні кінцевого пункту. Ми не можемо спостерігати рух, тому що воно дуже незначно, і не можна розрізнити дублікати, тому що вони перебувають на тій же позиції, що й вихідний об'єкт. Якщо потрібно переконатися в цьому, переглянете змінні або об'єкти під час перегляду фільму. Нам же потрібно з'ясувати, коли фільм досягне кінця, і потім виключити його. Той факт, що цей фільм відтворюється нескінченно, поки не є проблемою, однак якщо цей об'єкт буде використовуватися для відображення, наприклад, пункту меню, до того ж виявиться не єдиним, то число виконуваних дій може виявитися занадто більшим. Для їхньої одночасної реалізації затрачалося б занадто багато ресурсів комп'ютера, тому дуже важливо припиняти будь-які сповільнюючі роботу дії.

Отже, у міру продовження фільму, потрібно перевіряти, чи перебуває він у своїй кінцевій крапці, і, якщо це так, завершувати його. Здавалося б, легко визначити, чи є позиція фільму x рівної позиції фільму y, і т.д. Насправді, у випадку з даною зі швидкості, фільм ніколи не досягає своєї кінцевої мети, а лише нескінченно наближається до неї доти, поки відстань не стане пренебрежимо малим.

Якщо повернутися до вихідного прикладу об'єкта, що рухається від 0 до 200, з розбивкою кожного відрізка дистанції на дві частини, то для кожного кадру будуть наступні позиції: 100, 150, 175, 187,5, 193,75, 196,875, 198,4375, 199,21875, 199,609375, 199,8046875. Це називається парадоксом Зено, що полягає в тім, що якщо об'єкт завжди переміщається на половину попередньої дистанції, то він ніколи не досягне мети, всі наближаючись і наближаючись до неї. Тому що неможливо розділити на дві частини один піксель, картинка буде здаватися нерухливої, однак ActionScript цього не враховує. Замість того щоб перевіряти, чи перебуває фільм у своєму кінцевому положенні, ми будемо перевіряти, чи перебуває він поруч зі своєю метою, і чи менше заданого

відстань між об'єктом і його метою. При підході об'єкта до кінцевої крапки можна зупинити його, видаливши керуючий елемент onEnterFrame.

Почавши з нашого попереднього коду, ми додамо нову функцію checkDistance для перевірки того, чи є відстань меншим, ніж певне число. Фільм потім перейде у своє кінцеве положення, і функція, привласнена onEnterFrame, буде вилучена. Додайте наступний код після вже наявної програми:

```
ball_mc.checkDistance = function() {
  // check movieclip is within 0.2 pixels of target
  if (Math.abs(this.target-this._x)<0.2 && Math.abs(this.target-this._y)<0.2) {
    this._x = this.target;
    this._y = this.target;
    delete this.onEnterFrame;
  }
};
```

Насамперед, у нас є умовне вираження, що перевіряє, чи перебуває фільм усередині відрізка довжиною 0,2 пікселя від мети. У коді this.target-this._x представлена відстань від фільму до мети по осі X, і ми використовуємо Math.abs для перетворення цього значення в позитивне, тому що байдуже, праворуч або ліворуч перебуває фільм від кінцевої крапки, тобто між -0,2 і +0,2. Нарешті, у функції moveToMouse ми додаємо виклик функції checkDistance:

```
ball_mc.moveToMouse = function() {
  this._x += (this.target-this._x)/this._parent.speed;
  this._y += (this.target-this._y)/this._parent.speed;
  this.duplicate();
  this.checkDistance();
};
```

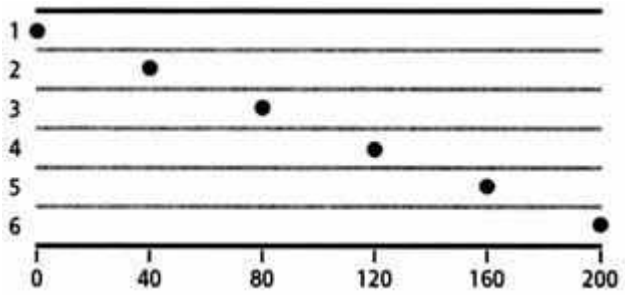
Виконання цього коду приведе до того ж результату, що й у випадку з попереднім файлом. Однак якщо вибрати команду Debug>List Variables у режимі Test Movie після закінчення фільму, ви побачите, що дублікати відсутні, і фільм ball_mc не має функції onEnterFrame. По великому рахунку, у нашім фільмі нічого не відбувається - він просто перебуває чекаючи клацання миші, щоб запуститися знову.

Давайте тепер перейдемо до розгляду того, як можна створити постійне, засноване на часі, рух.

17.4. Покадровий рух

Почнемо з одержання руху з постійною швидкістю. Вкажемо у фільмі, що необхідно перейти із крапки A у крапку B через певну кількість кадрів, переміщаючись із кожним кадром на те саме відстань. Цей тип руху корисний, коли потрібно, щоб щось відбувалося за певний проміжок часу. У цьому русі відсутня та плавність, що ви спостерігали в методі зменшення швидкості, однак

тут всі набагато простіше: нам відомо, що об'єкт досягне своєї мети після певної кількості кадрів. Даний рух виглядає більш солідно.



Відстань, на яке переміщається об'єкт, однаково у всіх кадрах. Пройдена відстань дорівнює всій відстані, діленому на число кадрів, у цьому випадку, 200 пікселів ділиться на 5 кадрів і виходить 40 пікселів на кожний кадр.

17.4.1. Постійна швидкість

Ви побачите, що в кожному кадрі переборюється те саме відстань. Для початку руху необхідно знати, наскільки повинен переміститися фільм, і протягом скількох кадрів це повинне відбутися. Обчислення відстані на кадр містить у собі лише розподіл відстані на загальне число кадрів.

1. Проробіть ті ж кроки, що й у попередньому прикладі, щоб приготуватися до додавання ActionScript у панель Actions.

2. Отже, перейдемо до програми. Нам потрібно привласнити дію нашому фільму, за допомогою якого в кожному кадрі буде виявлятися місце розташування покажчика миші, після чого фільм буде переміщатися в цю крапку. В Flash 5 це можна було б легко зробити за допомогою виділення фільму й присвоєння йому onClipEvent (enterFrame).

3. Насамперед, ми введемо змінну для зберігання числа кадрів руху. Потім ми встановимо функцію onMouseDown для ball_mc. Ця функція буде обчислювати загальну відстань по осях x і y і потім ділити це число на значення змінної кадрів для з'ясування відстані, на яке буде переміщатися фільм у кожному кадрі.

4. Ми також уведемо змінну-лічильник frameNum, що буде використовуватися для визначення того, на яку відстань перемістився фільм – ми будемо додавати до її значення одиницю в кожному кадрі фільму, і коли вона буде дорівнює значенню змінної кадрів, рух буде припинятися.

```
this.frames = 30;

ball_mc.onMouseDown = function() {
  this.frameNum = 0;
  // calculate how far to move each frame on the x and в axes
  this.xStep = (this._parent._ xmouse-this._x) /this._parent.frames ;
  this.yStep = (this._parent._ ymouse-this._y) /this._parent.frames;
  this.onEnterFrame = moveToMouse;
};
```

5. Далі нам потрібно буде написати функцію moveToMouse, що буде спочатку додавати значення xStep і yStep, додавати дублікат і потім перевіряти, чи досяг він своєї кінцевої крапки:

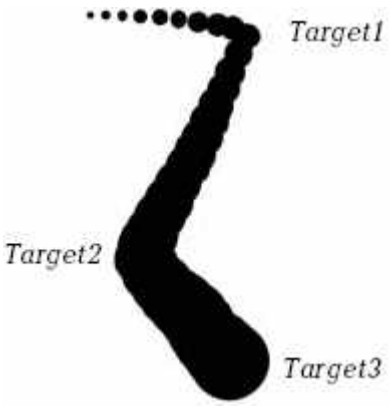
```
moveToMouse = function () {
  // add the step value each frame
  this._x += this.xStep;
  this._y += this.yStep;
  this.duplicate();
  this.checkTime();
};
```

6. Функції дубліката й зменшення ідентичні тим, які ми тільки що використовували.

7. Останнім додаванням у наш код буде функція checkTime, що додає одиницю до значення змінної frameNum і потім припиняє рух по досягненні кінцевої крапки, тобто числа кадрів, зазначеного нами на початку:

```
ball_mc.checkTime = function() {
  this.frameNum++;
  if (this.frameNum == this._parent.frames) {
    delete this.onEnterFrame;
  }
};
```

8. Збережете фільм у файлі frame based motion.fla і запустите його. При виконанні цього коду ви побачите, що дублікати створюються безупинно, і фільм безпосередньо досягає кінцевої крапки.

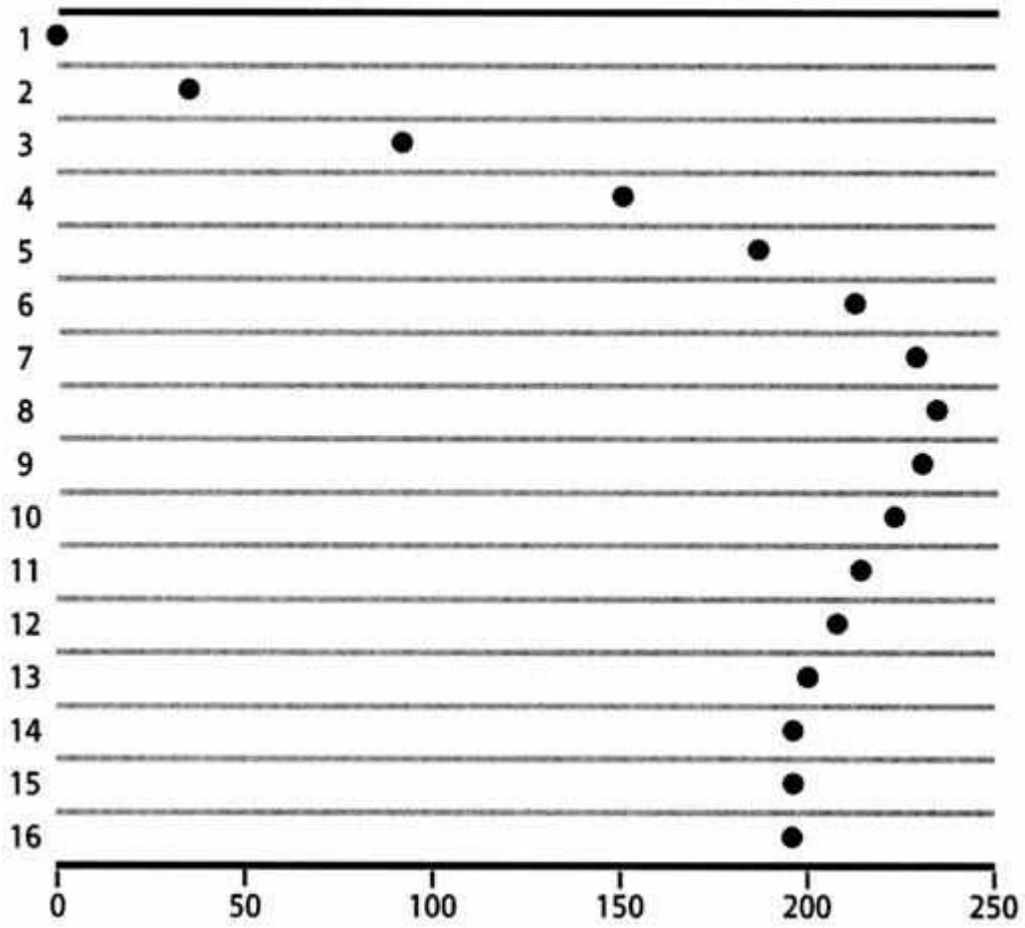


Якість руху в цьому прикладі сильно відрізняється від того, котре ми розглянемо нижче. Давайте ступнемо далі й розберемо прискорення й коливання в нескладних рухах.

17.4.2. Коливальний рух

Цей тип руху по своєму ефекті схожий на людину, що біжить, прив'язаного до еластичного троса. Уявіть, що людина біжить у напрямку до крапки, у якій закріплений гумовий канат, і, пробігши повз крапку закріплення, намагається втекти від її якнайдалі. MovieClip (тобто людина, прив'язана до каната), обчислює відстань до мети (крапка закріплення каната) і потім додає частина цієї величини до значення змінної швидкості. Фільм буде прискорюватися в міру свого наближення до кінцевої крапки (прискорення людини, що біжить), і потім, віддалившись від крапки, повернеться назад, після чого швидкість руху впаде.

У кожному кадрі ми множимо змінну speed на результат розподілу для зменшення швидкості. На графіку нижче показано, як триває рух після проходження кінцевої крапки, і наступне повернення об'єкта.



Швидкість об'єкта збільшується в пропорції з відстанню, на якому він перебуває від кінцевої крапки. З малюнка видно, що спочатку він стрімко прискорюється, проходить кінцеву крапку й потім вертається назад. Амплітуда його коливань біля мети повільно зменшується.

Існує безліч різних ефектів, який можна досягти за допомогою цього типу руху, тому що тут застосовуються два наступні параметри.

1. Розподіл відстані від кінцевої крапки, що додається до швидкості.
2. Розподіл, на яке множиться швидкість для її зменшення.

Остаточний ефект може бути у вигляді очікуваного коливання вперед та назад біля кінцевої крапки й набагато більше плавного зникнення, або ефекту різкої зупинки.

1. Ця вправа буде використовувати ті ж основні установки, які були настроєні раніше з фільмом, розміщеним на робочому місці з ім'ям ball_mc і окремим шаром для коду. Ми також можемо використовувати деякий код, що був застосований раніше (функції, що перевіряють відстань, що створюють сліди й обробну подію mouseDown), але він являє собою лише сам рух, і нам необхідно змінити його.

```
ball_mc.checkDistance = function() {
  if (Math.abs(this.target-this._x)<0.2 && Math.abs(this.target-this._y)<0.2) {
    this._x = this.target;
    this._y = this.target;
    delete this.onEnterFrame;
  }
};

ball_mc.duplicate = function() {
  var dupe = this.duplicateMovieClip ("dupe"+this._parent.depth,
       Kthis._parent.depth++);
  dupe.onEnterFrame = this._parent.diminish
};

function diminish(){
  // reduce scale gradually
  this._xscale = this._yscale-=2;
  //remove movieclip when scale goes below zero
  if (this._xscale <= 0) {
    this.removeMovieClip();
  }
}

ball_mc.onMouseDown = function() {
  this.targetx = this._parent._xmouse;
  this.target = this._parent._ymouse;
  this.onEnterFrame = this.moveToMouse;
};
```

2. Єдине, що нам потрібно додати, це функція moveToMouse, змінні швидкості й т.д. Спочатку необхідно визначити два наших параметри, які ми будемо називати acceleration (прискорення) і friction (коливання). Установимо для них значення 12 і 0,8, незважаючи на те, що в нас буде можливість змінити їх пізніше для досягнення різних ефектів. Додайте ці два рядки перед визначенням функції, у верхній частині коду:

```
var acceleration = 12;
```

var friction = 0.8;

Отже, acceleration буде значенням, на яке ми ділимо відстань перед додаванням його до швидкості нашого фільму, а friction буде значенням, що множиться на швидкість кожного кадру один по одному для зменшення швидкості руху.

Ми проробимо код руху по осі x перед додаванням його у функцію moveToMouse.

3. Насамперед, ми створимо тимчасову змінну для зберігання відстані між нашим фільмом і кінцевою крапкою. Наступний код призначений для функції moveToMouse.

var xdif=this.target-this._x;

4. Потім ми додамо результат розподілу цього значення до нашого значення xspeed (змінна, що представляє швидкість фільму по осі x):

this.xspeed+=xdif/this._parent.acceleration;

5. Потім множимо змінну speed на змінну friction для зменшення швидкості. Якщо не робити цього, наш фільм буде коливатися біля своєї кінцевої крапки, не наближаючись до неї. Це множення означає, що амплітуда зменшується з кожним разом. Якщо забрати цей рядок з вихідного коду нашого остаточного результату, фільм не зупиниться на своїй кінцевій крапці, а буде рухатися вперед-назад на те саме відстань.

this.xspeed*=this._parent.friction;

6. Нарешті, обновляємо фільм, додаючи значення його швидкості до положення на осі x:

this._x+=this.xspeed

Тепер ми поєднуємо все це в одну функцію moveToMouse, додаючи також виклики дубліката й checkDistance. Нижче наведений повний код програми, як для осі x, так і для осі y:

```
ball_mc.moveToMouse = function() {

  // calculate difference on x and в axes
  var xdif = this.target-this._x;
  var ydif = this.target-this._y;

  // increment speed values
  this.xspeed += xdif/this._parent.acceleration;
  this.yspeed += ydif/this._parent.acceleration;

  // dampen speed values
  this.xspeed *= this._parent.friction;
  this.yspeed *= this._parent.friction;

  // add speed values to x and в properties
  this._x += this.xspeed;
  this._y += this.yspeed;
```

```
  // create duplicate
  this.duplicate();
  // check if the movieclip has reached its target
  this.checkDistance();
};
```

7. Збережете ваш фільм у файлі swing.fla і запустите його. Ви побачите ефект коливань, причому об'єкт буде обертатися майже по колу навколо кінцевої крапки:

Зміна значень acceleration і friction сильно вплинуть на якість руху. Проекспериментуйте й спробуйте змінити їх, щоб наочно зрозуміти, як ці параметри можуть вплинути на рух.

Загалом кажучи, чим нижче значення acceleration, тим швидше буде спочатку рухатися фільм, і чим ближче до одиниці значення friction, тим більше фільм буде коливатися вперед та назад. Деякі значення можуть зробити цей приклад дуже схожим на випадок зі зменшенням швидкості, розглянутий нами в першу чергу. Наприклад, спробуйте ввести значення acceleration = 17; і friction = 0.4; і збережете ваш фільм у файлі swing02.fla. Різниця в тім, що в нас є змінна, що зберігається під час проходження кадрів, і фільм не змінює моментально напрямок свого руху, як це було у випадку з вихідним зменшенням швидкості руху, або рухом по кадрам, і тому ці зміни відбуваються плавнів.

Це все працює належним чином, однак є ще одна деталь, яку потрібно змінити. Беручи до уваги попередній приклад зі сповільненням руху, стає зрозуміло, що рух фільму повинне було припинитися відразу по проходженні певної відстані від кінцевої крапки, у цьому випадку фільм, приблизно, пройшов би прямо над кінцевою крапкою перед переходом на протилежну сторону. Це означає, що нам потрібно змінити нашу функцію checkDistance, щоб швидкість фільму була також нижче певного числа, щоб помилково не відбувалося раптового припинення руху фільму.

Це більше простий випадок: усе, що нам потрібно, це вставити вираження if у нашу функцію для перевірки того, чи є швидкість досить малої після влучення відстані в діапазон:

```
ball_mc.checkDistance = function() {
  // check that distance is within range
```

```
  if (Math.abs(this.target-this._x)<0.2 && Math.abs (this. target-this ._y) <0.2) {

    // check that speed is nearly zero
    if(Math.abs(this.xspeed)<0.2 && Math.abs(this.yspeed) <(0.2) {
      this._x = this.target;
      this._y = this.target;
      delete this.onEnterFrame;
    }
  }
};
```

Наведений вище сценарій, що втримується в swing3.fla, тепер захищений від будь-яких непередбачених обставин.

Ви можете використовувати ці основні принципи в різних ситуаціях, контролюючи швидкість об'єкта різними способами. Зараз на швидкість впливає положення кінцевої крапки (щось схоже на магнетизм), однак ви, наприклад, могли б натиснути кнопку правого курсору й додати 5 до значення xspeed. Це перемістить об'єкт сильно вправо перед тим, як його швидкість буде зменшуватися до повної зупинки.

Якщо вам цікаві способи створення інших типів руху, рекомендуємо вам звернути увагу на рівняння з на сайті www.robertpenner.com. Використовуючи ці рівняння, ви можете вказувати кількість кадрів, який необхідно для певної тривалості руху, і фільм буде сповільнювати свій рух від крапки A до крапки B. На додаток до цього, ви можете вивчити застосування цих рівнянь на www.gizma.com/easing. Якщо вам цікаві ці рівняння, Chris Andrade з www.fifthrotation.com опублікував документ, що описує їхній вивід, за адресою http://www.fifthrotation.com/u2/parabolic_ease.zip.

17.4.3. Зміна розмірів у русі

До цього моменту ми розглядали різні типи зі стосовно до руху по осях x і y, однак використання цього типу руху не обмежується як тільки цими параметрами. Можна переходити від одного значення до іншому й створювати рух, заснований на зміні розмірів об'єктів.

У даній вправі ми реалізуємо пружний рух, що тільки що було нами розглянуте. Ми будемо визначати координати клацання миші й використовувати їх для визначення розміру, що буде приймати наш фільм.

1. Створіть файл із фільмом у такий же спосіб, як і в попередніх прикладах, з ім'ям фільму ball_mc, але цього разу розташуєте фільм, указавши значення X = 275 і Y = 200 (біля середини робочого місця).

2. Ми почнемо з визначення функції mouseDown. Складемо координати X і Y покажчика миші, потім помножимо результат на два й використовуємо його для визначення кінцевого розміру нашого фільму. Якщо користувач клацне на лівому верхньому куті, фільм прийме розмір, дорівнює нулю, а якщо клацання буде в правому нижньому куті, тоді розмір стане рівним 1900, тобто (550+400)*2.

```
ball_mc.onMouseDown=function(){
  this.targetScale=(this._parent._xmouse+this._parent._ymouse)*2
  this.onEnterFrame=this.scaleMe
}
```

Отже, ми взяли це значення й записали його в змінну targetScale фільму.

3. Тепер нам потрібно визначити функцію scaleMe, що буде змінювати розмір фільму з поточного до кінцевого. Це робиться за допомогою досягнення певної крапки на осях x і y, аналогічного розглянутому раніше. Єдина різниця полягає в тім, що ми будемо змінювати параметри _xscale і _yscale замість _x і _y. Тому що ми хочемо дорівняти _xscale до _yscale, ми можемо проводити всі обчислення з використанням _xscale і потім установити _yscale на таке ж значення:

```
this.acceleration = 12;
this. friction = 0.8;
ball_mc.scaleMe = function() {
  var scaleDiff = this.targetScale-this._xscale;
  this.scaleSpeed += scaleDiff/this._parent.acceleration;
  this.scaleSpeed *= this._parent.friction;
  this._yscale = this._xscale += this.scaleSpeed;
  this.checkScale();
};
```

Перші три рядки функції такі ж, як і раніше – вони обчислюють різницю між поточним розміром і кінцевим розміром, і розраховують, на скільки

збільшується scaleSpeed. Потім змінна швидкості зменшується за допомогою множення на константу friction.

У наступному рядку ми робимо дві дії, еквівалентні наступним рядкам.

```
this._xscale += this.scaleSpeed;
this._yscale = this._xscale;
```

Коли Flash виявляє рядок коду, аналогічну тої, котру ми використовували в даному прикладі, він починає виконувати дії праворуч ліворуч. Спочатку до змінної швидкості додається _xscale і після цього _yscale прирівнюється до xscale.

4. Збережете ваш фільм у файлі scale.fla і запустите його – тепер об'єкт буде приймати різні розміри, залежно від того, на якому місці ви клацніть мишею.

5. Останнє, що потрібно зробити, це додати функцію checkScale. Вона більш-менш схожа на функцію checkDistance, з якої ми мали справу раніше, і призначена для перевірки того, що розмір фільму більш-менш близький до кінцевого розміру, і що його швидкість майже дорівнює нулю:

```
ball_mc.checkScale = function() {
  if (Math.abs(this.targetScale-this._xscale)<0.2) {
    if (Math.abs(this.scaleSpeed)<0.2) {
      this._xscale = this._yscale=this.targetScale;
      delete this.onEnterFrame;
    }
  }
};
```

Незважаючи на те, що ми використовували цей підхід лише для зміни розміру об'єкта, він може також застосовуватися для зміни параметрів _alpha, rotation або будь-якої іншої властивості вашого фільму.

17.5. Керування рухом

Тепер ви вже небагато розбираєтеся в тім, як працюють розглянуті різноманітні методи створення сценаріїв руху. Ми переходимо до розгляду різних способів керування рухом. Метод, що використовувався дотепер, має на увазі використання чималої кількості різних робочих місць, а ми настроїли його на роботу з одним фільмом.

Було б набагато краще брати будь-який певний фільм і створювати myMovieClip.slideTo(100,200), щоб фільм міг переміщатися до цієї крапки й зупинятися, без потреби привласнювати фільму вручну керуючий елемент clipEvent або функцію для перевірки досягнення кінцевої крапки, і т.д.

Далі ми будемо розглядати різні методи створення сценаріїв початку руху об'єкта. Ми не будемо вказувати на той або інший певний спосіб і говорити, що він є найкращим, а замість цього продемонструємо вам кілька можливостей, щоб ви могли вибирати з них потрібне для роботи з вашими проектами.

17.5.1. Ковзання

1. Відкрийте новий фільм і розмістите на робочому місці інстанс ball_mc.
2. Перше, що ми зробимо на окремому шарі сценаріїв, це додаймо виклик функції slideTo, що буде виконуватися щораз при клацанні мишею користувачем.
3. Ми будемо передавати функції slideTo три параметри: позицію по осі x, позицію по осі y і ще одне значення, що буде вказувати швидкість ковзання:

```
_root.onMouseDown = function() {
  ball_mc.slideTo(this._xmouse, this._ymouse, 4);
};
```

4. Тепер нам потрібно створити безпосередньо функцію slideTo, що повинна буде виконувати дві окремих дії:
   - установлювати у фільмі значення target і target;
   - набудовувати керуючий елемент enterFrame фільму для забезпечення його ковзання в напрямку до кінцевої крапки.

Замість того щоб визначати цю функцію для окремого фільму, як ми робили в попередніх функціях ball_mc.checkDistance, ми помістимо її в MovieClip.prototype. Всі функції усередині MovieClip.prototype доступні рівною мірою всім фільмам. Всі вбудовані методи фільмів, такі як goToAndPlay, також присутні в movieclip.prototype. При виклику функції вона поводиться так, ніби вона була усередині фільму.

```
MovieClip.prototype.slideTo = function(x, y, speed) {
  // set up targets and speed variable
  this.target = x;
  this.target = y;
  this.speed = speed;

  // create onEnterFrame function
    this.onEnterFrame = function() {
    // take care of motion
    this._x += (this.target-this._x)/this.speed;
    this._y += (this.target-this._y)/this.speed;

    // check if near target
      if (Math.abs(this.target-this._x)<0.2 && Math.abs(this.target-this._y)< 0.2)
{
      this._x = this.target;
      this._y = this.target;
      delete this.onEnterFrame;
    }
  };
};
```

Перші три рядки функції беруть параметри, які їй передаються, і зберігають їх у змінних усередині фільму. Потім привласнюється функція

onEnterFrame, що пересуває фільм, а також виконується перевірка досягнення кінцевої крапки.

- Збережете ваш файл під ім'ям issueCommand1.fla і запустите його. При клацанні мишею фільм буде сковзати до місця клацання.
- Ми також можемо спробувати застосувати той же самий підхід у двох фільмах - створити копію вихідного фільму ball_mc і дати новому інстансу ім'я ball_mc2 за допомогою Property Inspector.
- Перетягнете інстанс ball_mc2 на робоче місце й потім відкрийте панель Actions для зміни коду onMouseDown, відповідно до тексту коду, наведеному нижче:

```
_root.onMouseDown = function() {
  ball_mc.slideTo(this._xmouse, this._ymouse, 4);
  ball_mc2.slideTo(this._xmouse+19, this._ymouse, 1);
};
```

- Збережете фільм у файлі issueCommand1b.fla і запустите його. Ви заметете, що другий фільм переміщається значно правіше покажчика миші й зі значно меншою швидкістю.

Отже, наш фільм працює як треба, однак при більше детальному розгляді будуть виявлені деякі проблеми при використанні даного підходу. Найбільш помітний недолік полягає в тім, що будь-який фільм може мати тільки одну функцію, привласнену його керуючому елементу onEnterFrame: якщо фільм уже приблизно мав функцію, привласнену його керуючому елементу onEnterFrame, а ми привласнили йому іншу функцію, те друга функція буде записана поверх першої. Це може трапитися, якщо до фільму було застосовано плавне зникнення, збільшення розмірів або по яких-небудь інших причинах.

Інша проблема полягає в тому, що ми поміщаємо у фільм змінні, такі як target, target і speed, які можуть уже використовуватися у фільмі з іншими цілями. Вони будуть просто записуватися поверх старих, що відповідним чином вплине на функціональність фільму. Незважаючи на те, що ми звичайно пам'ятаємо імена, що привласнюються об'єктам у нашім ActionScript, корисно використовувати інформативні й не позбавлені змісту назви.

17.5.2. Використання порожніх фільмів

Тепер розглянемо все потенційно можливі проблеми, які можуть виникнути при створенні порожнього фільму усередині вже існуючого. Ми будемо використовувати внутрішній фільм як об'єкт, у якому перебувають всі змінні, і він буде також бути об'єктом, до якого можна привласнити керуючий елемент onEnterFrame, не побоюючись виходу з ладу якої-небудь попередньої функціональності.

Керуючий елемент mouseDown набудовуємо так само, як і колись:

```
_root.onMouseDown = function() {
  ball_mc.slideTo(this._xmouse, this._ymouse, 4);
};
```

Тепер наша функція slideTo буде виглядати майже так само, як раніше, і основною відмінністю є те, що колись фільм управляла сам собою, а тепер фільм виконує усередині себе певні дії. Це видно із заміни 'this' на 'this._parent' при доступі або зміні параметрів X і Y. Наш керуючий фільм здійснює доступ і керування параметрами свого батьківського фільму, тобто того фільму, якому необхідно переміщати.

```
MovieClip.prototype.slideTo = function(x, y, speed) {
  // create controller movieclip
  var control_mc = this.createEmptyMovieClip ("slideControl",
       Kthis.depth++);
  control_mc.target = x;
  control_mc.target = y;
  control_mc.speed = speed;
  control_mc.onEnterFrame = function() {
    // this._parent is the movieclip we're moving
    this._parent._x += (this.target-this._parent._x) /this.speed;
    this._parent._y += (this.target-this._parent._y)/this.speed;
    if (Math.abs(this.target-this._parent._x)<0.2 && Math.abs(this.target-
            Kthis._parent._y) <0.2) {
      this._parent._x = this.target;
      this._parent._y = this.target;
      this.removeMovieClip();
    }
  };
};
```

Як і відносно функції duplicate, створеної нами раніше, ми використовували посилання, що повертається функцією createEmptyMovieClip для звертання до знову створеного movieClip. Однієї з можливих "пасток" при використанні цього методу з createEmptyMovieClip є те, що користувач може записати фільм поверх уже наявного усередині. Простіше говорячи, це те, чого потрібно остерігатися при керуванні фільмом; необхідно мати якийсь механізм визначення того, на якій глибині перебувають об'єкти й символи. За допомогою введення змінної глибини для кожної тимчасової лінії можна ефективно реалізувати даний запобіжний захід, застосовуючи цей спосіб щораз при створенні фільму. З іншого боку, можна було б оголосити глобальну змінну глибини для всіх тимчасових ліній, однак все-таки краще мати одну змінну для кожної тимчасової лінії, щоб не втрачати контроль над значеннями. Залишаємо право вибору за вами.

Ви могли б використовувати addProperty для автоматичного додавання глибини щораз при спробі доступу, однак довелося б робити це для кожного фільму окремо, тому статична змінна, що збільшується на одиницю з кожним разом, буде оптимальним рішенням. Іншим можливим підходом може бути циклове проходження фільмів цілком в окремому місці, і використання getDepth() у кожному з них для з'ясування глибин, однак це досить

нераціональний спосіб, тому що тут прийде робити велика кількість непотрібних запитів на дії.

Збережете ваш фільм у файлі issueCommand2.fla. Якщо ви запустите фільм на даному етапі, ви заметете, що рух буде припинено при швидкому послідовному клацанні мишею. Якщо виконати команду Debug > List Variables, стане видно, що причиною цього є присутність більш ніж одного інстанса slideControl у нашім фільмі – всі вони працюють із різними напрямками. Таким чином, нам потрібно придумати щось таке, за допомогою чого можна буде переконуватися в присутності в будь-який момент часу лише одного інстанса цього керуючого кліпу усередині нашого фільму.

Можливим рішенням цієї проблеми є створення дублікатів slideControl щораз на одній і тій же глибині, щоб колишній інстанс автоматично перезаписувався новим. Однак краще не використовувати цей підхід, тому що він порушує схему глибини, про яку ми тільки що говорили. На додаток до цього, якщо ми використовували метод коливань для забезпечення руху нашого фільму, нам зовсім не потрібно буде перезаписувати будь-які змінні швидкості усередині фільму й переривати його рух; нам необхідне збереження цих даних до зупинки фільму.

Більше розумним рішенням буде перевірка того, чи є вже фільм slideControl. Якщо це не так, то вам потрібно буде створювати його, а якщо він уже присутній, то потрібно буде перезаписувати змінні X і Y кінцевої крапки й подія onEnterFrame.

Обидва підходи приведуть до одного й тому ж результату. Змінений текст коду виділений жирним шрифтом:

```
MovieClip.prototype.slideTo = function(x, y, speed) {
  var control_mc
  if (this.slideControl) {
    //if slideControl already exists
    control_mc = this.slideControl;
  }
  else {
    //if slideControl doesn't exist then create it
    control_mc = this.createEmptyMovieClip ("slideControl", this.depth++);
  }
  control_mc.target = x;
  control_mc.target = y;
  control_mc.speed = speed;
  control_mc.onEnterFrame = function() {
    this._parent._x += (this.target-this._parent._x)/this.speed;
    this._parent._y += (this.target-this._parent._y)/this.speed;
    if (Math.abs(this.target-this.__parent._x)<0.2 &&
    KMath.abs(this.target-this._parent._y)<0.2) {
      this._parent._x = this.target;
      this._parent._y = this.target;
      this.removeMovieClip();
```

```
  }
 };
};
```

Збережете файл під ім'ям issueCommand3.fla і при запуску фільму зверніть увагу на те, що тепер він поводиться потрібним образом.

Незважаючи на позитивний результат, даний підхід неспроможний із практичної точки зору, тому що ми не можемо визначити момент, коли фільм досягає своєї кінцевої позиції. Рух насправді припиняється, але нам часто буде вимагатися виконати деяка інша дія по закінченні руху. Принаймні, було б непогано знати, коли фільм перебуває в русі, а коли немає.

Для рішення цієї проблеми звернемося до функції зі зворотним зв'язком. Усе буде працювати в такий же спосіб, як і деякі інші компоненти – вони будуть викликати функцію при зміні значення або відновленні яких-небудь даних. При припиненні руху буде викликатися зазначена нами функція. Отже, насамперед, ми створимо функцію, що буде викликатися при закінченні руху фільму. У момент, коли це буде потрібно, можна виводити повідомлення у вікні Output із вказівкою ім'я фільму, що завершив свій рух, що ми будемо використовувати як параметр.

```
function slideDone(mc) {
  trace("movieClip "+mc+" has finished moving");
}
```

Загалом кажучи, коли рух фільму буде припинятися, буде здійснюватися виклик функції slideDone в _root. При виклику функції slideTo, нам потрібно буде додати два інших параметри – callBackObj і callBackFunc. callBackObj є місцем розташування викликуваної функції (у цьому випадку, _root), а callBackFunc – ім'ям викликуваної функції, тобто, у нашім випадку, функції slideDone. Отже, спочатку потрібно додати в нашу функцію два додаткових параметри:

```
_root.onMouseDown = function() {
  ball_mc.slideTo(this._xmouse, this._ymouse, 4,  _root, "slideDone");
};
```

Потім нам буде потрібно настроїти функцію slideTo для роботи із цими двома параметрами й для зберігання їх у фільмі slideControl (за допомогою додавання коду, що на прикладі нижче виділений жирним шрифтом):

```
MovieClip.prototype.slideTo = function(x, y, speed, callbackObj, callbackFunc)
{
  var control_mc
  if (this.slideControl) {
    control_mc = this.slideControl;
  }
  else   {
    control_mc = this.createEmptyMovieClip("slideControl", this.depth++);
  }
  control_mc.target = x;
  control_mc.target = y;
```

```
  control_mc.speed = speed;
  control_mc.callBackObj = callBackObj;
  control_mc.callBackFunc = callBackFunc;
};
```

Нарешті, нам потрібно з'ясувати, як викликати _root.slideDone з використанням двох змінних callBackObj і callBackFunc. Це насправді дуже просто й робиться практично в такий же спосіб, як здійснюється, наприклад, доступ до значення в масиві, або динамічне створення посилання на фільм:

```
this.callBackObj [this.callBackFunk] (this._parent)
```

Частина коду перед квадратними дужками є місцем розташування функції (_root). Уміст у квадратних дужках здійснює пошук значення, що відповідає даному рядку (у нашім випадку, slideDone), і потім значення в дужках передається у вигляді параметра. Передане тут значення є звертанням до фільму, що ми переміщали, тобто this._parent, our ball_mc. Наша функція цілком виглядає приблизно так:

```
MovieClip.prototype.slideTo = function (x, y, speed, callbackObj, callbackFunc) {
  var control_mc
  if (this.slideControl) {
    control_mc = this.slideControl;
  }
  else  {
    control_mc = this.createEmptyMovieClip ("slideControl", this.depth++);
  }
  control_mc.targetx = x;
  control_mc.target = y;
  control_mc.speed = speed;
  control_mc.callBackObj = callBackObj;
  control_mc.callBackFunc = callBackFunc;

  control_mc.onEnterFrame = function() {
    this._parent._x += (this.target-this._parent._x) /this.speed;
    this._parent._y += (this.target-this._parent._y) /this.speed;

    if (Math.abs(this.target-this._parent._x)<0.2 &&
    KMath. abs(this.target-this._parent._y)<0.2) {
      this._parent._x = this.target;
      this._parent._y = this.target;
      this.callBackObj[this.callBackFunc](this._parent);
      this.removeMovieClip();
    }
  };
};
```

Збережете ваш фільм у файлі issueCommand004.fla і при запуску фільму ви побачите, що у вікні Output відобразиться повідомлення "movieClip_level0.mc

has finished moving", тобто повідомлення про завершення руху фільму. Це буде відбуватися щораз при зупинці фільму.

Ви можете змінити вміст функції slideDone для здійснення будь-яких потрібних вам дій. Наприклад, у вас може бути масив, що містить дані про те, які фільми запущені, і ви можете використовувати його для забезпечення початку виконання наступної стадії проекту тільки по завершенні продовження фільму.

Функція callback дозволяє нам використовувати дуже важливу можливість зворотного зв'язка. Ми можемо ініціювати початок руху фільму до певного місця й залишити цей процес на самостійне завершення, тому що по закінченні руху від нього надійде відповідне повідомлення. Нашим наступним кроком буде настроювання функції, що зможе погодитися з будь-якими потрібними нам параметрами, зміненими у фільмі, після чого фільм буде змінювати всі ці параметри одночасно. Ми повернемося до цієї теми в одній з наступних лекцій.

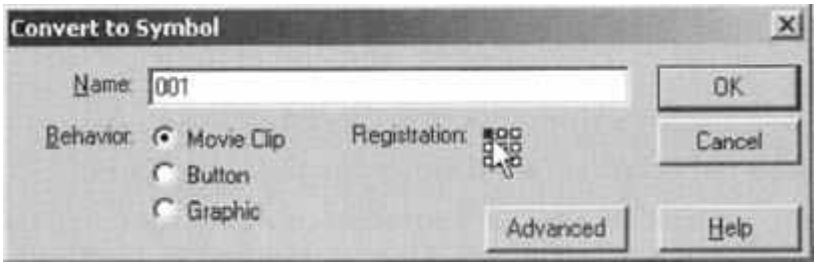
17.6. Реалізація підходів на практиці

Зараз прийшов час знову звернутися до деяким із уже розглянутих підходів. Те, про що ми говорили раніше, може здатися одноманітним, однак зараз ми приведемо нескладний приклад, що використовує деякі з вивчених нами способів. Як приклад створимо сторінку Flash.

17.6.1. Оглядач зображень

Ми плануємо створити набір з декількох фотографій так, що при клацанні на кнопці фотографії будуть переміщатися під маскою й рухатися таким чином, що в кінцевій області з'явиться потрібна фотографія.

1. Відкрийте новий фільм з дозволом (550x400 пікселів) і потім в Property Inspector установите частоту кадрів на значення 31.
2. Імпортуйте фотографії, що перебувають у файлах на компакт-диску, прикладеному до книги (001.jpg – 009.jpg), і перетворіть кожний з них у фільм з лівим верхнім кутом як крапка закріплення, назвавши їх відповідно один по одному іменами 001-009.



3. Тепер потрібно дати імена всім фільмам із зображеннями. За допомогою Property Inspector привласніть їм імена інстансів з c1 по c9.

4. Розташуєте малюнки так, щоб вони прилягали друг до друга в порядку 3x3, і, виділивши всі малюнки, перетворіть виділену область у фільм з ім'ям holder, указавши крапку закріплення в лівому верхньому куті.

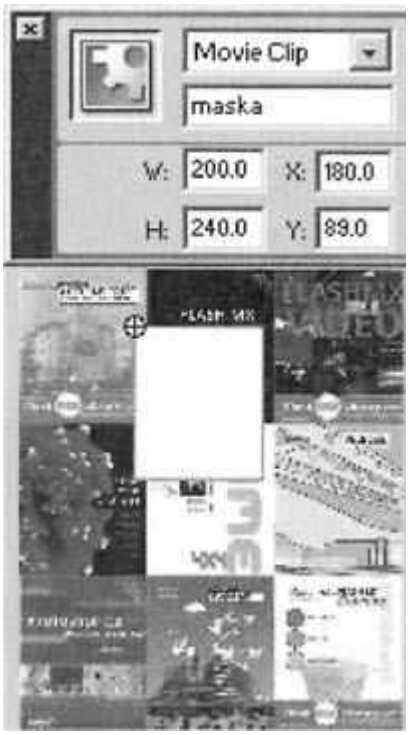
Отже, тепер у нас на робочому місці є інстанс holder у кореневій тимчасовій діаграмі. Ми додаймо рух, перемістивши фільм holder під маску так, щоб потрібний малюнок виявився на потрібній позиції.

5. Виділивши фільм holder, використовуйте Property Inspector, щоб задати йому ім'я інстанса holder і вказати його значення X і Y, установивши їх на 0,0. Після цього крапка закріплення буде розміщена в лівому верхньому куті робочого місця. Нарешті, назвіть поточний шар 'pictures'.

6. Нам потрібно буде створити маску, тому додайте новий шар над шаром pictures і позначте його як маску.

7. Намалюйте прямокутник 200x240 пікселів будь-якого кольору й за допомогою Property Inspector задайте значення X і Y, рівні 180,89.

8. Перетворіть виділений прямокутник у фільм з ім'ям maskSquare, переконавшись, що крапка закріплення перебуває в лівому верхньому куті. Нарешті, за допомогою Property Inspector дайте фільму maskSquare ім'я інстансу – maska.

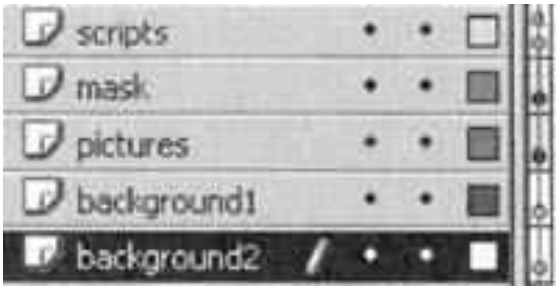


9. Тепер потрібно додати деякий код, щоб зробити прямокутник маскою. Створіть новий шар з ім'ям scripts і в новому шарі відкрийте панель Actions, щоб додати наступний код:

holder.setMask (maska);

За допомогою цього вираження фільм усередині дужок (maska) стає маскою для фільму, що був викликаний методом (holder). Якщо зараз запустити фільм, то стане видно, що під маскою будуть перебувати тільки ті частини holder (набору з дев'яти малюнків), які просвічуються, а інші будуть затемнені.

10. Зараз ми створимо кадр для прозорої області за допомогою додавання двох нових шарів, що містять прямокутники, один значно більше іншого. Додайте два нових шари під наявними у вас шарами й назвіть їх background1 і backgruond2. Шари в тимчасовій діаграмі будуть перебувати в наступному порядку:



11. Далі, на шарі background1 намалюйте ясно-сірий прямокутник 202x202 пікселя й за допомогою Property Inspector задайте його значення X і Y, рівні 179,88. Тепер на шарі background2 намалюйте інший сірий прямокутник більше темним кольором і розміром 204x244, задавши для нього значення X і Y, рівні 178,87. Це буде границею наших фільмів.

Ви можете додати будь-які інші потрібні вам зображення на фонові шари.

12. Щоб надати руху фільму holder, ми будемо використовувати функцію slide, створену раніше. Додайте наступний код у шар scripts над тільки що вставленою вами рядком коду:

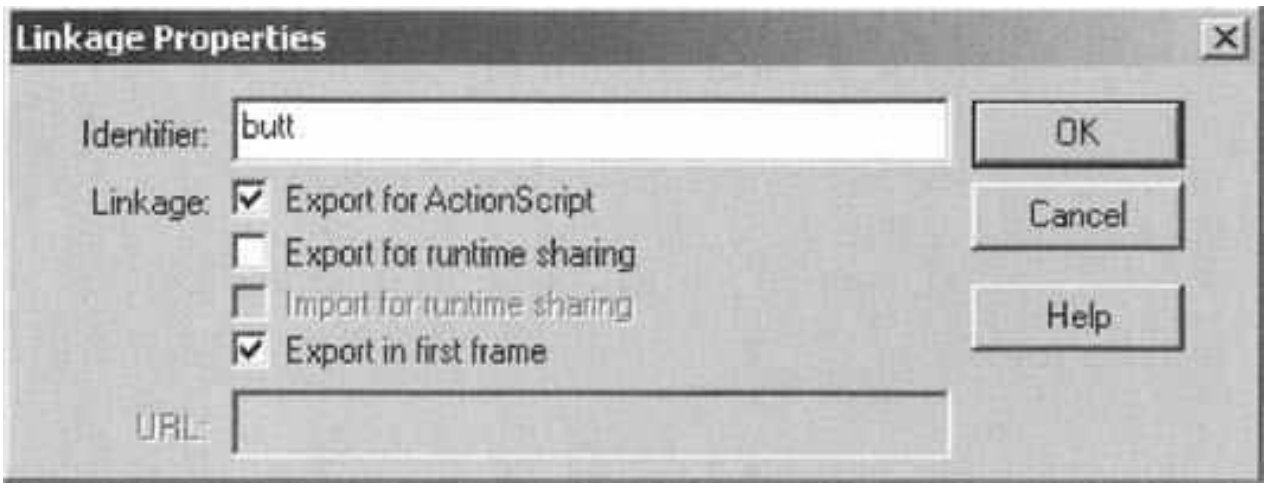
```
speed = 4;
MovieClip.prototype.slideTo = function(x, y, speed, callbackObj, callbackFunc)
{
  Var mc_control
  if (this.slideControl) {
    mc_control = this.slideControl;
  }
  else {
    mc_control = this.createEmptyMovieClip ("slideControl" , this. depth++) ;
  }
  mc_control.target = x;
  mc_control.target = y;
  mc_control.speed = speed;
  mc_control.callBackObj = callBackObj;
  mc_control.callBackFunc = callBackFunc;

  mc_control. onEnterFrame = function() {
    this._parent._x += (this.target-this._parent._x) /this.speed;
    this._parent._y += (this.target-this._parent._y) /this.speed;
    if (Math.abs(this.target-this._parent._x)<0.2 && Math.abs(this.target-
        Kthis._parent._y)<0.2) {
      this._parent._x = this.target;
      this._parent._y = this.target;
      this.callBackObj[this.callBackFunc](this._parent);
      this.removeMovieClip();
    }
  };
};
function slideDone(mc) {
```

```
  trace("movieClip "+mc+" has finished moving");
}
```

Для виконання цього коду ми створимо кілька кнопок. Замість того щоб динамічно створювати кнопки з використанням createEmptyMovieClip і createTextField (ці підходи ми розглянемо пізніше), ми створимо символ вручну й потім динамічно застосуємо його.

13. Створіть новий фільм і назвіть його butt. Виділите фільм в Library і клацніть на білому значку меню вгорі панелі заголовка Library. У меню виберіть параметр Linkage:, щоб відкрити вікно Linkage Properties.

14. Відзначте поле Export for ActionScript, привласніть символу ідентифікатор butt і натисніть клавішу OK.



15. Тепер ви будете перебувати в кореневому шарі фільму butt. Розмістите поле Dynamic Text усередині фільму й дайте йому ім'я інстанса tf за допомогою Property Inspector. Розмір тексту нашого поля дорівнює 9, шрифт – Arial, без вбудованих шрифтів.

16. Для автоматичного додавання кнопок нам потрібно додати наступний код у шар scripts нашої кореневої тимчасової діаграми (під уже наявним кодом).

```
holder.setMask(maska);
for (var i = 1; i<=9; i++) {
  var mc_control = _root.attachMovie("butt", "butt"+i, i);
  mc_control._x = 175+i*9;
  mc_control._y = 375;
  mc_control.mc = this.holder["c"+i] ;
  mc_control.tf.text = i;
}
```

Цей код циклічно виконується для кожного фільму з фотографією й додає кнопку. У ньому використовується ітератор 'i' для установки текстового значення поля 'tf', тому в поле з'являться числа від 1 до 9. На додаток до цього, використовується значення 'i' для установки кінцевого фільму для кожної

кнопки (незважаючи на те, що butt є фільмом, цей об'єкт буде кнопкою, тому що ми будемо привласнювати йому подія onPress). Число 9 наприкінці четвертого рядка коду контролює відстань між всіма фільмами-кнопками.

Перед присвоєнням події onPress нам потрібно вирішити, яке саме дія буде виконуватися при натисканні кнопки, так само як і місце, у яке повинен переміститися holder для коректного відображення кожної фотографії під маскою.

Все це можна зробити, розглянувши конкретний випадок. Наприклад, якби в нас був фільм, розташований у крапці 300,300 у фільмі holder, де повинен був би бути розташований holder, щоб правильно відображати фотографії під маскою? Якби ми розташували holder на тім же місці робочого стола, що й маску, то відображалося б усе, що було б на позиції 0,0 в holder. Ґрунтуючись на цьому, ми можемо зробити вивід, що якби нам потрібно було відображати об'єкт на позиції 300,300 в holder, ми могли б перемістити holder на 300 пікселів уліво й на стільки ж нагору. У коді ми б указали це в такий спосіб:

(maska._ x-300, maska._ y-300);

Тепер, якщо ми звернемося до координат _x і _y кожного з наших фільмів із зображеннями, то одержимо координати фільму усередині holder. Отже, можна сказати, що для того, щоб кожний фільм був розташований строго під маскою, нам необхідно переміщати holder на наступне місце:

(maska._ x-mc._x, maska._ y-mc._y);

17. Отже, тепер ми можемо побудувати функцію onPress для наших кнопок:

```
for (var 1 = 1; i<=9; i++) {
  var mc_control = _root.attachMovie("butt" , "butt"+i, i);
  mc_control._x = 175+i*9;
  mc_control._y = 295;
  mc_control.mc = this.holder["c"+i] ;
  mc_control.tf.text = i;
  mc_control.onPress = function() {
    var x = this._parent.maska._ x-this.mc._x;
    var в = this._parent.maska._ y-this.mc._y;
    this._parent.holder.slideTo(x, y, this._parent.speed,
    Kthis._parent, "slideDone") ;
  };
}
```

Виклик функції slideTo відбувається в такий же спосіб, як і раніше, з використанням показаного раніше обчислення для вказівки координат X і Y кінцевої крапки ковзання. Нам необов'язково записувати значення X і Y у

змінні, тому ми можемо забрати з коду відповідну його частину й виконувати обчислення усередині виклику функції. Зазначений код виділений жирним шрифтом:

```
for (var i = 1; i<=9; i++) {
  var mc_control = _root.attachMovie("butt", "butt"+i, i);
  mc_control._x = 175+1*9;
  mc_control._y = 295;
  mc_control.mc = this.holder["c"+i] ;
  mc_control.tf .text = i;
  mc_control.onPress = function() {
    this. parent.holder.slideTo(this._parent.maska.
    ДО_ x-this.mc._x, this._parent.maska._ y-this.mc._y,
    Kthis._parent.speed, this._parent, "slideDone");
  };
}
```

Збережете фільм у файлі pictureMask.fla. При його запуску ви побачите, що при ковзанні кожного малюнка на його позицію відбувається реєстрація виконуючого у вікні Output. Зараз для нас це не має значення, однак пізніше ми розберемося, як використовувати цю можливість для виклику якої-небудь події, наприклад, текстового ефекту. Це лише одна з можливих застосувань цього вікна. Однак цей вид керування також може бути використаний для виводу деякого пояснювального тексту, так, щоб малюнки були тлом інтерфейсу, або для запуску анімації. Необхідно помітити, що ми управляємо рухом, що самі ж і ініціювали. Ми знаємо, коли воно починається, і можемо бути сповіщені про його завершення, а це означає, що воно може стати частиною ланцюга подій.

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

Навчально-методична література

Тотосько О.В., Микитишин А.Г., Стухляк П.Д.

Навчальний посібник

«КОМП'ЮТЕРНА ГРАФІКА»

Книга 1

Для студентів спеціальності 151

«Автоматизація та комп'ютерно-інтегровані технології»

Комп'ютерне верстання *А. П. Катрич*

Формат 60х90/16. Обл. вид. арк. 14,07. Тираж 10 пр. Зам. № 2893.

Тернопільський національний технічний університет імені Івана Пулюя.
46001, м. Тернопіль, вул. Руська, 56.
Свідоцтво суб'єкта видавничої справи ДК № 4226 від 08.12.11