

Міністерство освіти і науки України

**Чернівецький національний університет
імені Юрія Федьковича**

В. Г. Маценко

Комп'ютерна графіка

Рекомендовано Міністерством освіти і науки України
як навчальний посібник
для студентів вищих навчальних закладів

Чернівці
«Рута»
2009

ББК 32.973.2 –018я73

М-367

УДК 004.92(075.8)

Рекомендовано

Міністерством освіти і науки України
як навчальний посібник для студентів вищих навчальних закладів
(лист про надання грифу № 1.4/18-Г-633 від 14.03.08)

Рецензенти:

Бомба А. Я., доктор технічних наук, професор кафедри інформатики та прикладної математики Рівненського державного гуманітарного університету;

Сопронюк Ф. О., доктор фізико-математичних наук, професор, декан факультету комп'ютерних наук Чернівецького національного університету імені Юрія Федьковича;

Турчак Л. І., доктор фізико-математичних наук, професор, завідувач відділом обчислювальної фізики Обчислювального центру імені Дородніцина РАН (Москва);

Косак О.С., доцент кафедри прикладної математики Львівського національного університету імені Івана Франка.

М-367 **Маценко В.Г. Комп'ютерна графіка:** Навчальний посібник. – Чернівці: Рута, 2009 – 343 с.
ISBN 966-568-846-4

У посібнику викладено математичні основи комп'ютерної графіки, наведено широкий спектр основних понять та алгоритмів, розглянуто фізичні основи сприйняття кольору та особливості колірних моделей, описані технічні та програмні засоби комп'ютерної графіки. Значна увага приділяється математичному моделюванню геометричних об'єктів і побудові їх проєкцій. Розглянуто методи комп'ютерної 2D- і 3D-графіки та засоби програмування графіки. Наведено завдання до лабораторного практикуму, вправи та задачі для самостійного розв'язування, питання для самоконтролю, завдання для розробки комп'ютерних проєктів.

Для студентів вищих навчальних закладів, які навчаються за напрямами „Прикладна математика”, „Інформатика” та ін.

ББК 32.973.2 –018я73
УДК 004.92(075.8)

ISBN 966-568-846-4

© В.Г. Маценко, 2009
© „Рута”, 2009

ЗМІСТ

Передмова	7
Тема 1. Вступ до комп'ютерної графіки	9
1.1. Основні поняття	9
1.2. Простіші моделі графічних об'єктів	11
1.3. Математичні моделі об'єктів графічних сцен	16
1.4. Застосування комп'ютерної графіки.	20
Тема 2. Технічне та програмне забезпечення комп'ютерної графіки	29
2.1. Технічне забезпечення комп'ютерної графіки	29
2.1.1. Пристрої введення графічної інформації	29
2.1.2. Пристрої виведення графічної інформації	31
2.2. Програмне забезпечення	36
2.2.1. Основні поняття	36
2.2.2. Найвідоміші графічні редактори	37
2.2.3. Мови програмування графіки	40
2.2.4. Графічні засоби відеосистем.	41
2.3. Види комп'ютерної графіки	44
2.3.1. Растрова графіка	44
2.3.2. Векторна графіка	45
2.3.3. Фрактальна графіка	47
2.4. Графічні файлові формати	48
Тема 3. Колір. Моделі кольору	54
3.1. Природа кольору	54
3.2. Моделі кольорів	59
3.2.1. Адитивна модель кольору RGB	59
3.2.2. Субтрактивна модель кольорів CMY/CMYK	61
3.2.3. Суб'єктивна модель кольорів HSB (HSV)	64
3.3. Баланс кольорів	65
3.4. Кодування кольору. Палітра кольорів	66
3.5. Оптимальне поєднання кольорів при побудові зображень	69
Тема 4. Відеоадаптер. Програмування відеоадаптера	72
4.1. Основні поняття	72
4.2. Режими відеоадаптерів	76
4.3. Програмування відеоадаптерів у простіших режимах	79
4.4. Програмування відеоадаптерів у режимах ОЕН, 10Н, 12Н	82
Тема 5. Растрові алгоритми генерування кривих ліній	89
5.1. Числові методи	89
5.2. Інкрементні алгоритми	91
5.2.1. Алгоритм Брезенхема для відрізка	92
5.2.2. Алгоритм Брезенхема для кола	95
5.2.3. Інкрементний алгоритм виведення еліпса	97
5.2.4. Інкрементний метод Жордана	99

Тема 6. Растрові алгоритми зафарбовування і заповнення областей . . .	102
6.1. Основні поняття	102
6.2. Рекурсивні алгоритми заповнення областей	104
6.3. Пострічковий алгоритм зафарбовування з затравкою	105
6.4. Алгоритм заповнення області за критерієм парності	106
6.5. Зафарбовування полігонів. УХ-алгоритм	108
6.6. Заповнення фігур. Текстури	110
Тема 7. Побудова інтерполяційних та згладжуючих кривих	119
7.1. Основні поняття.	119
7.2. Поліноміальна інтерполяція	119
7.2.1. Інтерполяційний многочлен Лагранжа	120
7.2.2. Інтерполяційні сплайни	120
7.3. Згладжуючі кубічні сплайни	124
7.4. Сплайнові криві	125
7.5. Криві Безьє	127
7.5.1. Основні поняття	127
7.5.2. Властивості кривих Безьє	128
7.5.3. Складені криві Безьє	130
7.5.4. Геометричний алгоритм для кривої Безьє	131
7.6. В-сплайнові криві	132
7.7. Інтерполяційні кубічні криві Ерміта	140
7.8. ТСВ-сплайни	142
Тема 8. Математичні моделі поверхонь	146
8.1. Білінійна та лінійчаста поверхні	146
8.2. Інтерполяційні бікубічні сплайни	148
8.3. Сплайнові поверхні	148
8.3.1. Поверхні Безьє	149
8.3.2. В-сплайнові поверхні	151
Тема 9. Основні алгоритми комп'ютерної геометрії	153
9.1. Тести орієнтації	153
9.1.1. Орієнтація нормального вектора	153
9.1.2. Розміщення точки відносно прямої	153
9.1.3. Тест напрямку обходу трьох точок	154
9.2. Тест опуклості полігона	155
9.3. Тести орієнтації точки відносно полігона	155
9.3.1. Габаритний тест	155
9.3.2. Тест, що визначає орієнтацію точки відносно кожного ребра	156
9.3.3. Променевий тест	156
9.3.4. Кутовий тест	157
9.4. Тести перетину на площині	159
9.4.1. Тест перетину прямої з полігоном	159
9.4.2. Тест перетину відрізків	161

9.5. Алгоритми відсікання	161
9.5.1. Двовимірний алгоритм Сазерленда-Коена	162
9.5.2. Відсікання відрізка опуклим полігоном	163
9.5.3. Перетин та об'єднання опуклих полігонів	165
9.6. Інші алгоритми відсікання відрізків.	165
9.6.1. Двовимірний FC-алгоритм	166
9.6.2. Алгоритм Кіруса-Бека	169
9.6.3. Відсікання полігонів. Алгоритм Вейлера-Азертонна	172
9.7. Побудова опуклої оболонки масиву точок	175
9.7.1. Метод загортання подарунка	176
9.7.2. Метод обходу Грехема	177
9.8. Тріангуляція полігонів	179
9.8.1. Тріангуляція опуклих полігонів	179
9.8.2. Тріангуляція неопуклих полігонів	180
9.8.3. Тріангуляція Делоне	181
Тема 10. Фрактали в комп'ютерній графіці	184
10.1. Поняття фрактала	184
10.2. Конструктивні фрактали	186
10.2.1. Крива Коха	186
10.2.2. Зіркові фрактали	188
10.3. Аналіз конструктивних фракталів	190
10.4. Динамічні фрактали	195
10.4.1. Множини Жуліа і Мандельброта	195
10.4.2. Фрактали Жуліа	197
10.4.3. Фрактали Мандельброта	198
10.4.4. Фрактали Ньютона	201
10.5. Застосування фракталів	202
Тема 11. Моделювання 2D/3D-перетворень	204
11.1. Афінні перетворення на площині	204
11.2. Приклади афінних перетворень на площині	210
11.3. Афінні перетворення в просторі.	213
11.4. Приклади складніших 3D-перетворень	216
11.5. Методи задання складних афінних перетворень	220
Тема 12. Моделювання проєкцій	226
12.1. Класифікація проєкцій	226
12.2. Ортографічна проєкція	227
12.3. Аксонометрична проєкція	229
12.4. Косокутна проєкція	234
12.5. Одноточкова (однофокусна) перспективна проєкція	239
12.6. Двоточкове та триточкове перспективні перетворення	242
12.7. Методи створення перспективних видів	244

Тема 13. Системи координат та їх перетворення	247
13.1. Системи координат	247
13.2. Видове перетворення	250
13.3. Перспективне проектування	253
13.4. Відображення у вікно виведення	257
Тема 14. Усунення невидимих ліній і граней	259
14.1. Основні поняття	259
14.2. Алгоритм поточного горизонту	260
14.3. Алгоритм Робертса	266
14.4. Метод Z-буфера	274
14.5. Огляд деяких інших методів	276
14.5.1. Метод відсікання нелицьових граней	276
14.5.2. Алгоритм Варнока розбиття картинної площини	277
14.5.3. Метод сортування за глибиною. Алгоритм художника	278
Тема 15. Зафарбовування видимих поверхонь	283
15.1. Моделі відбиття світла	284
15.1.1. Дзеркальне відбиття світла	284
15.1.2. Дифузне відбиття	286
15.2. Обчислення нормалей до поверхні відбиття світла	289
15.3. Зафарбовування поверхонь	291
15.3.1. Зафарбовування з постійною інтенсивністю	292
15.3.2. Метод Гуро	292
15.3.2. Метод Фонга	294
15.4. Методи трасування променів	295
Тема 16. Програмування графіки на OpenGL	301
16.1. Основні поняття	301
16.2. Вершини і примітиви	304
16.3. Перетворення координат і проекцій	309
16.4. Функції виведення тривимірних об'єктів	313
16.5. Функції визначення видимих поверхонь	314
16.6. Моделювання освітлення	314
16.7. Область виведення	318
Завдання до лабораторних робіт	327
Комп'ютерні проекти	339
Список літератури	341

ПЕРЕДМОВА

Стрімкий розвиток комп'ютерного апаратного забезпечення і різноманітного програмного забезпечення сприяє якісним змінам у традиційних технологіях обробки інформації. З'являються нові нетрадиційні технології, в тому числі й такі, що змінюють сам стиль використання комп'ютерів. Серед них найбільш динамічно розвиваються мережні технології, комп'ютерна графіка, тривимірне моделювання й анімація, користувацький інтерфейс тощо.

Комп'ютерна графіка – складова всіх сучасних комп'ютерних технологій. Графічний інтерфейс став стандартом для програмного забезпечення різних класів, починаючи з операційних систем. Комп'ютерна графіка привертає увагу багатьох спеціалістів із різних галузей знань – програмістів, проектувальників засобів візуалізації, інженерів, фізиків, математиків та ін. Знання основ комп'ютерної графіки необхідне кожному вченому чи інженеру.

Комп'ютерна графіка – це галузь знань, в якій, з одного боку, накопичено значний багаж знань, з іншого боку, здійснюється постійний розвиток методів, алгоритмів та практичних застосувань, це складна і різноманітна дисципліна. Комп'ютерна графіка є важливою компонентою освіти сучасного спеціаліста. В багатьох випадках потреби в графіці можуть бути забезпечені різними існуючими графічними бібліотеками та системами. Однак, постійно виникає необхідність створювати спеціальні графічні програмні засоби. Зробити це можна, якщо оволодіти практичними навичками розв'язування типових задач комп'ютерної графіки та відповідними теоретичними знаннями. Крім цього, слід зауважити, що навіть ефективне оволодіння існуючими графічними засобами вимагає знань теоретичних основ комп'ютерної графіки.

Метою курсу з комп'ютерної графіки для напрямів „Прикладна математика”, „Інформатика” (для студентів, що вивчають вищу математику і програмування) є оволодіння алгоритмами та методами, які можна було б використовувати при створенні нових реальних систем машинної графіки в поєднанні з застосуванням існуючих програмних засобів.

Фундаментом більшості програм машинної графіки є складний та розгалужений математичний апарат, а саме методи аналітичної та нарисної геометрії, векторної алгебри, числових методів, математичної логіки, методів оптимізації та ін. Математичним забезпеченням комп'ютерної графіки служать геометричні моделі разом із методами та алгоритмами перетворення цих моделей.

Глибоке оволодіння базовими математичними концепціями та мистецтвом програмування є головним ключем до розуміння подальшого

розвитку машинної графіки. Не розглядаючи складних математичних задач, потенціал машинної графіки часто реалізується не повністю внаслідок неадекватного використання математичного апарату. Тому в посібнику основна увага приділяється математичним методам, основним алгоритмам, технологіям та концепціям комп'ютерної графіки, а також методам процедурного характеру.

Мета цього посібника – забезпечити студентів навчальним матеріалом з різних аспектів комп'ютерної графіки, тому посібник складається з двох частин: теоретичної та практичної.

В теоретичній частині курсу даються основні означення, зроблено опис технічних та програмних засобів комп'ютерної графіки, описано організацію графічних режимів, наведено базові растрові алгоритми, основні алгоритми обчислювальної геометрії, методи моделювання координатних перетворень, методи й алгоритми тривимірної графіки, методи рендерінга, технології програмування графіки, приклади розв'язування типових задач. Більшість з наведених алгоритмів лежить в основі роботи багатьох сучасних графічних редакторів.

Практична частина містить лабораторний практикум, що базується на викладеному теоретичному матеріалі курсу. Він призначений для вироблення вмінь та навичок самостійної розробки програмного забезпечення. Тут наведено завдання для лабораторних робіт та більш складніші завдання для розробки комп'ютерних проектів.

Зростання частини самостійної роботи студентів потребує відповідних вправ, тому після кожної теми дається список контрольних питань та формулюються вправи і задачі для самостійного розв'язування. Це буде сприяти кращому розумінню і більш глибокому засвоєнню одержаних теоретичних знань, а також розвитку у студентів творчого і просторового мислення.

Зміст посібника значною мірою побудований на основі курсу "Комп'ютерна графіка", який викладається на факультеті прикладної математики Чернівецького національного університету імені Юрія Федьковича. Посібник може використовуватись студентами як денної, так і заочної форм навчання, викладачами вищої школи та спеціалістами, які працюють у галузі програмування графіки.

Побажання і пропозиції щодо покращення структури та змісту навчального посібника надсилайте за адресою *vgm2006@mail.ru*.

Тема 1. Вступ до комп'ютерної графіки

1.1. Основні поняття

Формування комп'ютерної графіки (надалі – КГ) як самостійного напрямку інформаційних технологій відноситься до 60-х років ХХ ст., коли Сазерлендом був створений спеціалізований пакет машинної графіки. Вперше подання інформаційних даних на екрані комп'ютера в графічному вигляді було продемонстровано на початку 50-х років спеціалістами Массачусетського технологічного інституту і згодом стало використовуватися в наукових і військових дослідженнях. За цей час комп'ютерна графіка пройшла шлях від окремих експериментів до одного з найважливіших інструментів сучасності. Розвиток комп'ютерної графіки проходить жваво: дещо швидко відживає, а водночас з'являється багато нового. Нині не можна собі уявити розвиток будь-якої галузі людської діяльності, пов'язаної з наукою, чи технікою без використання КГ. Вона є багатофункціональною складовою усіх інформаційних технологій і важливою компонентою для взаємодії людини з комп'ютером.

На першому етапі свого розвитку КГ була пасивною, однак це вже забезпечувало наочну форму сприйняття інформації, а, як відомо, більшу частину інформації (до 75%) про навколишній світ людина сприймає візуально, тобто за допомогою органів зору. З цього приводу можна згадати прислів'я: „краще раз побачити, ніж сто разів почути”, або його китайський аналог „одна картинка коштує 1000 слів”. Зображення не тільки місткий, але і доступний вид інформації, оскільки для сприйняття візуальної інформації від користувача вимагається менше зусиль. Інформація, що міститься у зображеннях, є найбільш концентрованою, найлегше сприймається та найшвидше обробляється.

При пасивній графіці ЕОМ формує і виводить зображення на графічний пристрій під управлінням прикладних програм. Користувач не може безпосередньо керувати формуванням зображення, перш ніж воно не з'явиться на екрані. Пасивний контроль не дає можливості втрутитися в процес проектування конструкції (система працює в пакетному режимі без діалогу). Для модифікації вихідного зображення необхідний повторний запуск пакета прикладних програм. Тому виникла необхідність у розвитку діалогових систем КГ, тобто систем інтерактивної графіки, щоб користувач оперативно міг вносити зміни в зображення безпосередньо в процесі його відтворення (в реальному масштабі часу). Нині всі сучасні графічні програми є системами інтерактивної КГ і вони можуть відтворювати всі особливості реальних зображень. Дамо ряд визначень.

Графіка – вид мистецтва, що включає малюнок і художнє зображення (походить від грецьк. слова графо, що означає „пишу, креслю, малюю”).

Комп'ютерна графіка – це галузь знань, що вивчає та розробляє методи і засоби синтезу, збереження й перетворення цифрових зображень за допомогою ЕОМ та інших технічних пристроїв.

Цифрове зображення – це модель реального або синтезованого (створеного штучно) зображення, що зберігається в пам'яті ЕОМ у вигляді сукупності цифрових кодів.

Режим, при якому користувач у реальному часі може впливати на весь процес формування зображення, тобто зміни в зображення можна вносити безпосередньо в процесі його відтворення (в режимі діалогу), називається **інтерактивним**.

Інтерактивною КГ називають тоді, коли до складу графічної системи входять технічні і програмні засоби, що дозволяють динамічно формувати зображення.

Історично першими інтерактивними системами вважаються системи автоматизованого проектування (САПР), які з'явилися в 60-х роках ХХ ст.

КГ – це область інформатики, у сферу інтересів якої входять всі аспекти формування зображення за допомогою комп'ютера. Цифрові зображення немислимі без комп'ютерної обробки графічної інформації, їх простіше зберігати, тиражувати, компоувати тощо. Цифрові зображення можна створити сканером, цифровим фотоапаратом, а потім відредагувати в програмі обробки зображень (наприклад, в Adobe Photoshop), а можна створити цифрові малюнки з нуля, застосувавши спеціальні програми (наприклад, у Corel Draw) або написавши відповідну програму на мові програмування (наприклад, Visual C++).

Основними задачами КГ є візуалізація інформації, тобто створення зображень різних об'єктів і сцен (у загальному випадку тривимірних) на деякому двовимірному екрані (наприклад, на екрані монітора або на аркуші паперу), виконання різних дій із зображеннями, зберігання та передавання графічної інформації.

Під **синтезованим зображенням** розуміють певне довільне візуальне подання інформації, яке отримується в комп'ютері на основі математичного опису (моделі) об'єкта згідно з директивами користувача.

Таке зауваження вилучає з предмета КГ задачі обробки та розпізнавання зображень, отриманих шляхом фотографування, оскільки в цьому випадку ми не маємо справи з побудовою графічних об'єктів за певними правилами.

При обробці зображень на основі початкового зображення одержують зображення, яке володіє іншими властивостями. Прикладом методів обробки зображень можуть бути підвищення контрасту, корекція кольорів, реставрація зображень.

При розпізнаванні образів ставиться задача за початковим зображенням знайти математичний опис об'єктів, наприклад розпізнати текст.

Тому обробка та розпізнавання зображень віднесені в самостійні напрями у сфері обробки графічної інформації.

Отже, питаннями роботи з зображеннями на ЕОМ займаються три напрями інформатики: комп'ютерна графіка, обробка зображень, розпізнавання образів. Ми розглядатимемо в основному лише задачі першого напрямку.

1.2. Простіші моделі графічних об'єктів

Графічний об'єкт – це певний малюнок, одержаний за допомогою комп'ютера. Сукупність графічних об'єктів називають *зображенням*. Графічні зображення часто мають ієрархічну структуру, яка виникає в результаті процесу побудови знизу-вверх.

Простіші базові графічні елементи, з яких будуються графічні об'єкти довільної складності, називаються *графічними примітивами*. Вони використовуються як будівельні блоки для об'єктів більш високого рівня. Графічні примітиви розглядаються як прості неподільні елементи зображення і генеруються однією окремою командою (підпрограмою). Вони можуть мати також апаратний та апаратно-програмний рівень формування. Як правило, множина примітивів містить такі елементи [19]:

- точку,
- відрізок,
- ламану лінію (замкнену і незамкнену),
- трикутник,
- трикутний стріп (рис. 1.1, *а*),
- трикутний фен (рис. 1.1, *б*),
- чотирикутник,
- чотирикутний стріп (рис. 1.1, *в*),
- опуклий багатокутник (полігон).

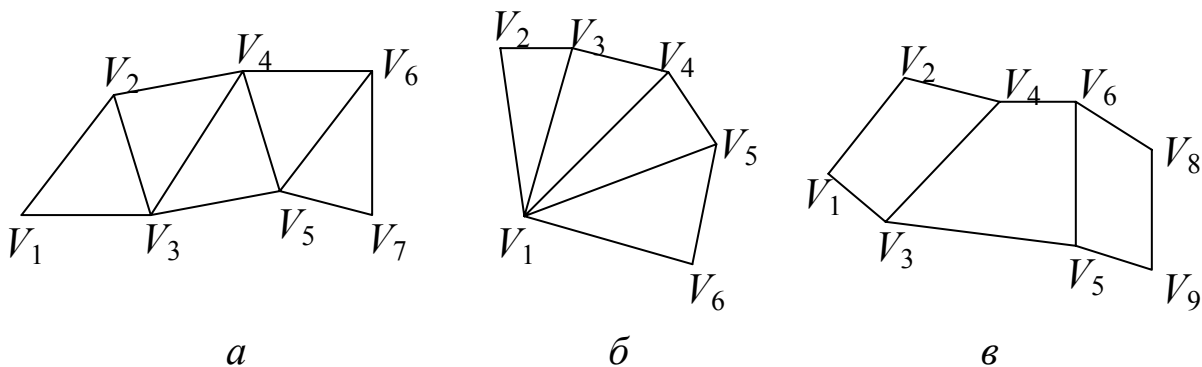


Рис. 1.1. Елементи множини примітивів: *а* – трикутний стріп, *б* – фен, *в* – чотирикутний стріп

Трикутний стріп – це множина трикутників $T = \{V_i V_{i+1} V_{i+2}\}$, що утворюється послідовністю вершин $V_1, V_2, V_3, \dots, V_{n+2}$. Перший трикутник стріпа $V_1 V_2 V_3$. Другий трикутник $V_2 V_3 V_4$ містить уже відоме ребро $V_2 V_3$ і тому для його задання необхідно вказати тільки одну вершину і т.д. Тому стріп із 5 трикутників (рис. 1.1, а) задається послідовністю V_1, V_2, \dots, V_7 із семи вершин (замість п'ятнадцяти у випадку задання кожного трикутника окремо). Оскільки обробка кожної вершини в графічній системі займає певний час, то зменшення кількості обчислювальних вершин прискорює візуалізацію примітива.

Трикутний фен (віяло) – це множина трикутників $T = \{V_1 V_{i+1} V_{i+2}\}$, утворених послідовністю вершин V_1, V_2, \dots, V_{n+2} (рис. 1.1, б), тобто трикутників, які задаються першою і кожною наступною парою вершин (пари не перетинаються). При заданні фену спочатку вказується загальна вершина, а потім решта вершин у порядку обходу спільних ребер.

Чотирикутний стріп – це множина чотирикутників $Q = \{V_{2i-1} V_{2i} V_{2i+2} V_{2i+1}\}$, утворених послідовністю вершин $V_1, V_2, \dots, V_{2n+2}$. Як видно з рис. 1.1, в, при заданні чотирикутного стріпа послідовність перерахування вершин інша, ніж при заданні незв'язаних чотирикутників.

Полігоном називається замкнена крива на площині, утворена відрізками прямих ліній. Відрізки прямих називаються ребрами або сторонами, кінцеві точки – вершинами. Дві вершини, що належать одному й тому ж ребру, називають *суміжними*. Відрізок прямої лінії, що лежить між двома несуміжними вершинами, називається *діагоналлю*.

Полігон називається **простим**, якщо він не має самоперетинів ребер. Простий полігон однозначно охоплює неперервну частину площини, яка є внутрішньою частиною полігона.

Для тривимірних додатків головною характеристикою полігона є не кількість вершин, а їх тип. Вершини полігона поділяються на опуклі і неопуклі (ввігнуті). Вершина називається **опуклою**, якщо внутрішній кут при вершині менше 180° , в протилежному випадку вершина **неопукла**.

Нагадаємо, що область на площині називається **опуклою**, якщо для будь-яких двох точок області відрізок, що їх з'єднує, повністю лежить всередині цієї області. Будь-яка вершина в опуклому полігоні є опуклою (в неопуклому полігоні принаймні одна вершина неопукла). З цього випливає той факт, що при обході границі опуклого полігону в напрямку за годинниковою стрілкою в кожній вершині наступне ребро або продовжується по прямій лінії, або повертається вправо. В комп'ютерній графіці полігони особливо важливі як засоби задання поверхонь.

В сучасних системах комп'ютерної графіки інформація про графічні об'єкти поділяється на дві категорії: параметри та атрибути. Примітиви

теж характеризуються параметрами та атрибутами. **Параметри** примітивів характеризують форму, розміри, місце розташування примітива. **Атрибути** примітивів – це властивості, які визначають їх вигляд при візуалізації, тобто спосіб зображення заданого примітиву.

Для кожного типу об'єктів визначають свій набір атрибутів:

- для точки – розмір та колір;
- для лінії – колір, товщина і тип (лінія, яка використовується для малювання відрізків може бути суцільною або штриховою);
- для многокутників – режим контуру чи заповнення. Цей параметр вказує спосіб задання многокутників. Можна малювати тільки вершини або ребра, а можна зображати зафарбовані многокутники, при цьому контур може малюватись одним кольором, а внутрішня область заливатись іншим.

У просторі простішими графічними примітивами є поліедри. **Поліедри** – це замкнені просторові об'єкти, що обмежені площинами. Серед об'ємних тіл поліедри займають таке ж важливе місце, як і полігони серед плоских фігур. Один полігон задає одну грань об'ємного об'єкта. Декілька граней у просторі складають об'ємний об'єкт у вигляді замкненої полігональної поверхні (многогранника), або незамкненої поверхні (полігональної сітки).

Модель поліедра $H = \{P, G\}$ складається з двох масивів:

- координатного списку вершин $P = \{P_1, \dots, P_n\}$ пронумерованих у довільному порядку;
- масиву списків граней $G = \{G_1, \dots, G_m\}$. Елемент $G_i = \{g_{i1}, \dots, g_{in_i}, g_{i1}\}$ масиву G є списком номерів вершин полігона i -тої грані, перерахованих у порядку обходу їх по замкненому контуру.

Полігональні моделі найбільш широко розповсюджені в сучасних системах тривимірної графіки, хоча вони мають і ряд недоліків, наприклад, приводять до складних алгоритмів візуалізації реалістичних зображень. Завдяки планарності всіх граничних поверхонь поліедра істотно спрощується розрахунок його перетинів із різними геометричними примітивами. Крім цього, часто криволінійні граничні поверхні об'єктів апроксимуються системою полігональних граней, що перетворює ці об'єкти на поліедри.

У природі існує цілий ряд поліедральних об'єктів. Серед них правильні многогранники, призми, піраміди тощо.

Правильні многогранники (платонові тіла) – це опуклі многогранники, всі грані яких є правильними многокутниками і всі многогранні кути при вершинах рівні між собою.

Існує 5 правильних многогранників (це довів ще Евклід): тетраедр,

гексаедр (куб), октаедр, додекаедр, ікосаедр. Префікс перед грецьким коренем 'едр' означає кількість граней поліедра.

Для повного опису правильних многогранників унаслідок їхньої опуклості достатньо вказати спосіб знаходження всіх його вершин. Розглянемо деякі способи їх побудови.

Вершинами тетраедра є чотири вершини куба, попарно несуміжні з жодним із його ребер (рис. 1.2, *a*).

Вершини октаедра знаходяться в центрах граней куба (рис. 1.2, *б*).

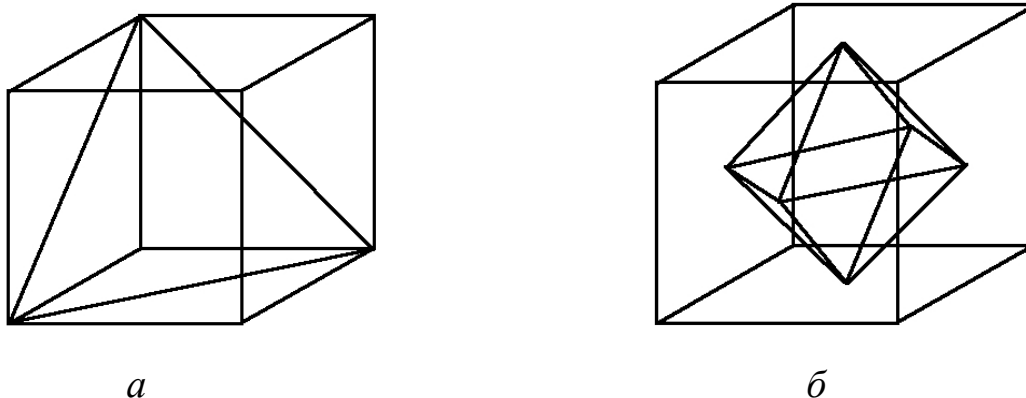


Рис. 1.2. Приклади простіших поліедрів: *a* – тетраедр, *б* – октаедр

Опишемо спосіб конструювання ікосаедра. Побудуємо циліндр одиничного радіуса, вісь якого збігається з віссю апікат z , а основи лежать у площинах $z = 0,5$ і $z = -0,5$. Розбиваємо кожне з кіл, що лежать в основах циліндра, на 5 рівних частин, як це зображено на рис. 1.3.

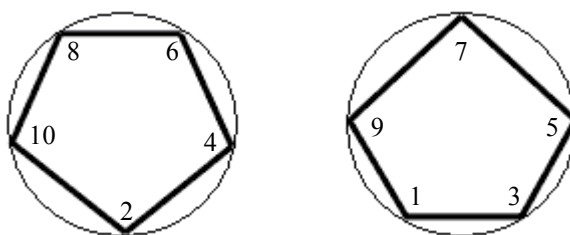


Рис. 1.3. Послідовність з'єднання вершин

Далі нумеруємо ці точки проти годинникової стрілки парними та непарними числами і потім послідовно, відповідно до нумерації, з'єднуємо ці точки відповідними відрізками (рис. 1.4). Одержуємо десять середніх трикутних граней ікосаедра.

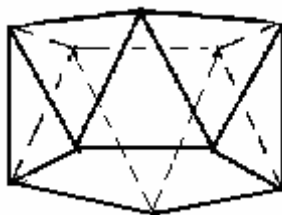


Рис. 1.4. Середні грані ікосаедра

Для завершення побудови ікосаедра вибираємо на осі z дві точки так, щоб довжини бокових ребер у п'ятикутних пірамідах із вершинами в цих точках і основами, що співпадають з побудованими п'ятикутниками, дорівнювали довжинам сторін побудованих правильних трикутників. Такими точками є точки з аплікатами $\pm \sqrt{5/2}$.

У результаті описаних побудов одержуємо 12 точок (вершин), а опуклий многогранник із вершинами в цих точках буде мати 20 граней, кожна з яких є правильним трикутником (рис. 1.5, *a*). Декартові координати вершин побудованого ікосаедра легко обчислюються.

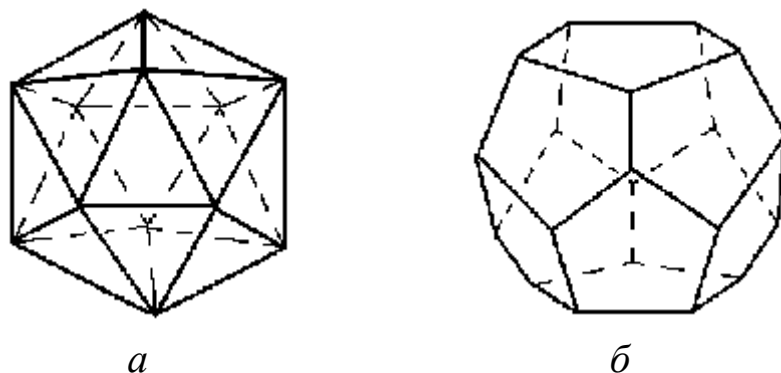


Рис. 1.5. Складніші поліедри: *a* – ікосаедр, *б* – додекаедр

Залишається побудувати додекаедр (кількість граней – 12) – найбільш складне платонове тіло. Вершини додекаедра є центрами трикутних граней ікосаедра. Отже, координати кожної вершини додекаедра можна знайти шляхом обчислення середнього арифметичного координат вершин ікосаедра. Далі необхідно з'єднати центри суміжних граней ребрами (рис. 1.5, *б*).

Множина видимих на екрані об'єктів, упорядкованих згідно з просторовими відношеннями (наприклад, за відношенням даліше – ближче, лівіше – правіше), разом з атрибутами елементів поверхонь (колір, текстура, оптичні властивості матеріалів) і з атрибутами оточення (рівень освітлення, туман, димка тощо) утворюють **сцену**.

Графічні об'єкти, з яких складається сцена, називаються **елементами сцени**. Для повного опису сцени необхідна, як мінімум, така інформація:

- морфологічна інформація – це зведення про форму, структуру кожного елемента незалежно від його розміщення та розмірів і місцезнаходження спостерігача. Методи моделювання змін форми об'єкта, тобто поступового перетворення одного образу в інший (анімація форми) називаються **морфінгом** (від грецьк. *morphe* – форма). Зазвичай перетворення форми об'єкта визначається за допомогою множини опорних точок і подальшої інтерполяції;
- геометрична інформація, яка включає характеристики розмірів еле-

- ментів сцени, параметри взаємного розміщення елементів (віддаль між ними, кути), розміщення елементів відносно спостерігача і т.д.;
- інформація про джерела світла, про їх розміщення, про властивості матеріалу об'єктів.

1.3. Математичні моделі об'єктів графічних сцен

Для систем машинної графіки джерелом вхідної інформації є не самі фізичні процеси та об'єкти, а математичні моделі. *Математична модель* – це сукупність математичних співвідношень, за допомогою яких описуються основні характеристики реального об'єкта. При побудові об'єктів і сцен різної природи у комп'ютерній графіці найчастіше використовують геометричну версію математичного моделювання, тобто кожний елемент сцени розглядається як геометричний об'єкт із певними властивостями, а сцена складається з множини геометричних об'єктів.

За способом представлення геометричних об'єктів можна виділити дві найчастіше використовувані моделі елементів графічних сцен:

- графічні примітиви;
- математичні моделі просторових об'єктів, які задаються рівняннями ліній і поверхонь.

У першій моделі вважається, що сцена формується з відомих геометричних фігур (примітивів): двовимірних (ліній, многокутників тощо) і тривимірних (кубів, пірамід, куль, циліндрів тощо). Примітиви розглядаються як прості неподільні елементи зображення. Тому програмна реалізація побудови сцени полягає в підборі відповідних фігур, які вибираються з бібліотеки примітивів. При використанні бібліотеки примітивів маємо справу з двома рівнями примітивів: двовимірними та тривимірними.

Щоб побудувати елемент сцени, який відсутній в бібліотеці примітивів, потрібно сконструювати його з тих примітивів, які наявні в бібліотеці, використовуючи різні операції та композиції елементів. Це добре відомі операції, які застосовуються до суцільних тіл: повороти різного роду, зсув, перетворення подібності, симетричне відображення, розтяг, сегментація об'єкта (виділення тієї частини об'єкта, над якою буде проводитися перетворення), об'єднання (сума) об'єктів – $Pr1 \cup Pr2$ (графічний об'єкт, кожна точка якого належить хоча б одному з примітивів), перетин (добуток) примітивів $Pr1 \cap Pr2$ (тіло, кожна точка якого належить одночасно двом примітивам), різниця двох примітивів $Pr1 \setminus Pr2$ (тіло, яке складається з точок $Pr1$, що не належать до $Pr2$).

У другому типі моделей геометричні елементи сцени розглядаються як функціональні залежності яскравості точок сцени від координат цих точок, тобто сцену можна побудувати за допомогою аналітичних рівнянь,

які описують лінії та поверхні, що задають тіла сцени. Аналітичні моделі широко застосовуються при проектуванні геометричних об'єктів складної форми, при підготовці програм управління для станків з ЧПУ, при розкроюванні матеріалу тощо. Наведемо рівняння деяких ліній і поверхонь.

Пряма лінія на площині задається неявним рівнянням $ax + by + c = 0$, або явним рівнянням $y = kx + b$. Відрізок, що визначається двома точками r_0, r_1 , задається векторно-параметричним рівнянням вигляду $r = r_0(1 - t) + r_1t$, де $r = (x, y)$, $t \in [0, 1]$.

Загальна явна форма задання кривої на площині має вигляд $y = f(x)$, неявна форма $-f(x, y) = 0$. Явний вигляд кривої $y = f(x)$ має ряд недоліків. По-перше, не можна описати замкнені криві одним виразом. Такі криві необхідно розбивати на ряд сегментів і шукати аналітичне задання для кожного з них. По-друге, такий опис кривих не володіє інваріантністю відносно перетворення повороту. Щоб задати повернуту криву, необхідно виконати певну обчислювальну роботу. По-третє, при обчисленнях значень $f(x)$ виникають істотні обчислювальні складності, наприклад для кривих із великими кутами нахилу. Задання кривої в неявній формі $f(x, y) = 0$ теж викликає ряд проблем.

Параметрична форма задання кривої $x = x(t), y = y(t), t \in [\alpha, \beta]$ усуває недоліки функціонального і неявного способів задання кривої.

Параметрична форма рівнянь є найбільш універсальною і найбільш розповсюдженою для опису і побудови геометричних об'єктів завдяки різноманітності вибору функцій, що задають рух точки вздовж напрямів степенів вільності. Параметрична форма дозволяє задати замкнені й багатозначні криві.

Поверхні можна задати явним рівнянням $z = f(x, y)$, або неявним $-F(x, y, z) = 0$, чи в параметричній формі: $x = x(\tau, t), y = y(\tau, t), z = z(\tau, t), (\tau, t) \in D$.

Поверхні першого порядку (площини), що задаються рівнянням $ax + by + cz = d$, та поверхні другого порядку, наприклад еліпсоїд $x^2/a^2 + y^2/b^2 + z^2/c^2 = 1$ – найбільш широко використовувані поверхні в КГ, оскільки площина є основою наближення поверхні довільної форми, а квадратичні поверхні в багатьох випадках добре передають візуальні образи криволінійних поверхонь.

Зазначимо, що серед поверхонь 2-го порядку тільки еліпсоїд (сфера як частковий випадок) локалізований у просторі, всі інші простягаються в нескінченність, тому вимагають для свого задання обмежуючих площин.

Параметричне рівняння сфери має вигляд

$$x(\tau, t) = R \sin t, y(\tau, t) = R \cos t \sin \tau, z(\tau, t) = R \cos t \cos \tau, \quad t \in [0, \pi], \tau \in [0, 2\pi].$$

Для побудови моделі сфери використовують побудований раніше ікосаедр. Зауважимо, що ікосаедр уже є моделлю сфери (всі вершини

ікосаедра лежать на сфері), єдиний недолік – мала кількість граней для зображення поверхні сфери. Для підвищення реальності зображення сфери діють так:

- кожна грань ікосаедра розбивається на чотири частини (трикутники). Додаткові вершини нових трикутників беруться на серединах сторін грані, як це показано на рис 1.6;
- нові вершини проектується на поверхню сфери. Для цього з центра сфери через вершини проводяться промені і відшукуються точки перетину променів із поверхнею сфери. На цих точках будуються нові грані які краще апроксимують сферу;
- указані дії повторюються доти, поки не буде досягнуто задовільного зображення сфери.

Аналогічним чином можна побудувати моделі циліндра, тора тощо.

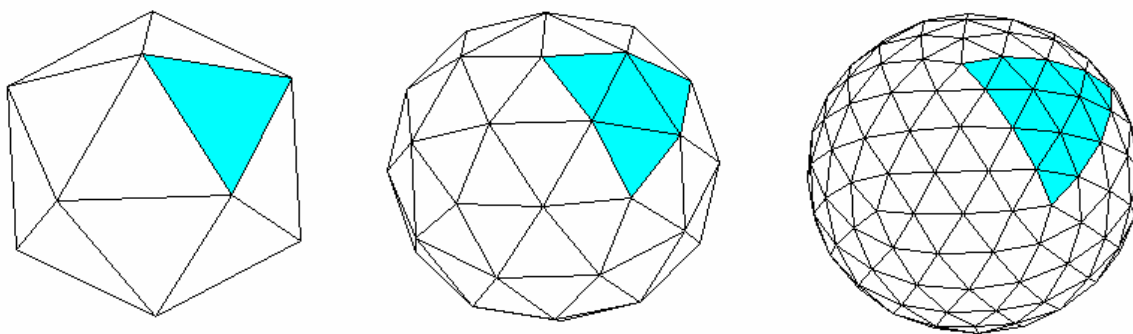


Рис 1.6. Побудова зображення сфери

Фрагмент білінійної поверхні, розташованої між чотирма точками P_0, P_1, P_2, P_3 , описується векторно-параметричним рівнянням

$$r = P_0(1 - u)(1 - v) + P_1u(1 - v) + P_2(1 - u)v + P_3uv, \quad r = (x, y, z), \quad u, v \in [0, 1].$$

За допомогою білінійних поверхонь описують складні 3D-поверхні.

Лінійчаста поверхня загального вигляду, що утворюється внаслідок руху прямолінійної твірної вздовж двох напрямних ліній $r_1(v)$ та $r_2(v)$ задається рівнянням $r = r_1(v)(1 - u) + r_2(v)u$.

Лінія в просторі є або перетином двох поверхонь, або слідом руху деякої точки, тому маємо дві форми моделі просторової кривої: неявна форма $\{F_1(p) = 0\} \cap \{F_2(p) = 0\}$, параметрична форма $x = x(t), y = y(t), z = z(t)$.

Прикладом параметричної просторової кривої є циліндрична гвинтова лінія (рис. 1.7), що задається рівнянням $x(t) = r \cos t, y(t) = r \sin t, z(t) = ht, r, h > 0, t \in [0, \infty)$. Ця крива лежить на поверхні циліндра радіусом r . При зміні параметра t на 2π змінні x та y повертаються до свого попереднього значення, а z збільшується на $2\pi h$. Ця величина називається кроком спіралі.

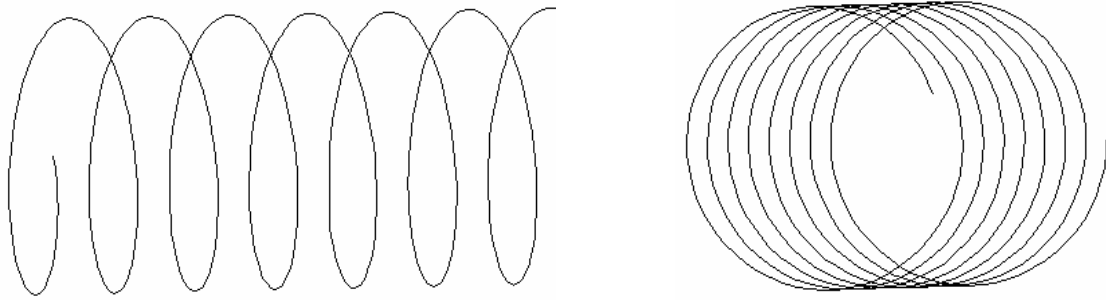


Рис. 1.7. Циліндрична гвинтова лінія

Іншим прикладом параметричних кривих є *циклічні криві*. Простіші циклічні криві на площині утворюються додаванням відносного руху точки по колу з переносним рухом центра кола по кривій $p_u(t)$ і описуються рівняннями

$$p(t) = p_u(t) + rc(wt), \text{ де } p = (x, y), \text{ } c(wt) = (\sin wt, \cos wt).$$

Випишемо рівняння деяких циклічних кривих на площині.

Циклоїда – це траєкторія руху точки кола радіусом r , яке котиться по прямій $y = 0$ (рис. 1.8). Координати (рівняння) циклоїди мають вигляд

$$x(t) = r(t - \sin t), \quad y(t) = r(1 - \cos t), \quad t \geq 0.$$

Епіциклоїда – це траєкторія руху точки кола радіусом r , якщо воно котиться зовні кола радіусом r_0 (рис. 1.9, *а*). Її параметричне рівняння:

$$p(t) = (r_0 + r)(\sin t, \cos t) - r(\sin wt, \cos wt), \quad w = (r_0/r + 1).$$

Гіпоциклоїда – це траєкторія руху точки кола радіусом r , яке котиться всередині кола радіусом r_0 (рис. 1.9, *б*). Рівняння гіпоциклоїди має вигляд:

$$p(t) = (r_0 - r)(\sin t, \cos t) + r(-\sin wt, \cos wt), \quad w = (r_0/r - 1).$$

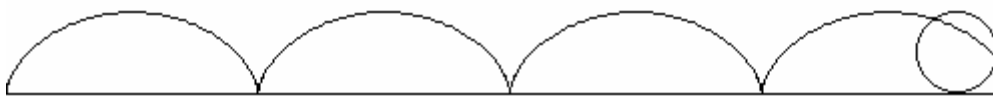


Рис. 1.8. Циклоїда

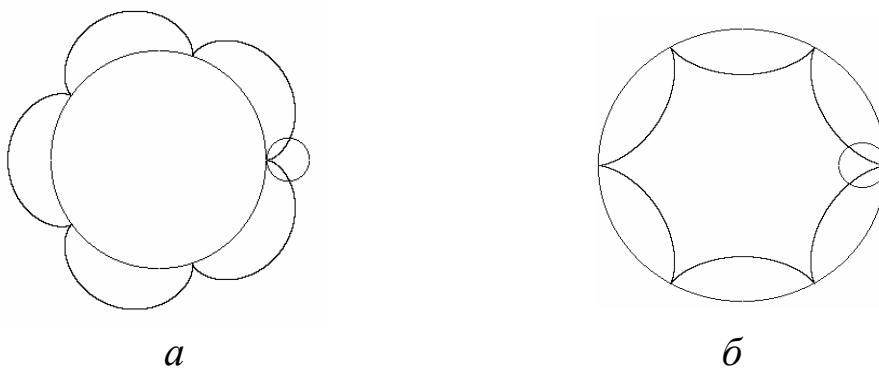


Рис. 1.9. Циклічні криві: *а* – епіциклоїда, *б* – гіпоциклоїда

1.4. Застосування комп'ютерної графіки

У наш час комп'ютерна графіка застовується майже в усіх галузях людської діяльності. Нині графічний інтерфейс користувача став основним засобом спілкування людини з ЕОМ, усі сучасні операційні системи використовують графічні елементи управління. Комп'ютерна графіка має найрізноманітніші застосування в:

- 1) системах автоматизації проектування та конструювання (інженерна графіка);
- 2) автоматизованих системах наукових досліджень;
- 3) інформаційних системах;
- 4) системах ілюстративної та ділової графіки;
- 5) системах машинної геометрії;
- 6) анімаційних задачах;
- 7) комп'ютерних іграх;
- 8) відеотренажерах (для тренування льотчиків, диспетчерів, військових, шоферів);
- 9) мистецтві, видавничій та рекламній діяльності, засобах масової інформації, криміналістиці, медицині та в інших галузях людської діяльності.

1. *Інженерна графіка* – це створення і корегування графічної документації (креслень) в електронній формі. *Сучасна інженерна графіка* – це складний програмний комплекс, що дозволяє вести наскрізне проектування, від постановки задачі та математичного моделювання до випуску всієї необхідної виробничо-технологічної документації, й використовується в усіх галузях зв'язаних із проектуванням. Основними об'єктами інженерної комп'ютерної графіки є складні структури: інтегральні схеми; промислове і побутове обладнання; хімічні, енергетичні та космічні установки; кузови автомобілів; фюзеляжі літаків; корпуси суден; складні механізми; будівлі та інженерні споруди; комунікації та мережі тощо. Головне практичне втілення інженерної комп'ютерної графіки – це системи автоматизованого проектування (САПР). Системи типу САПР, активно використовуються в багатьох галузях (наприклад, у машинобудуванні, електроніці, будівництві, архітектурі). Одними із перших були створені САПР для проектування літаків, автомобілів, електронних інтегральних схем. Такі системи спершу функціонували на великих машинах, згодом із ростом продуктивності персональних комп'ютерів САПР почали використовувати дешеві комп'ютери, а це привело до широкого розповсюдження САПР. Застосування САПР дало незаперечні переваги. Вони дозволили накопичувати й передавати знання та досвід конструкторів, що зумовлює скорочення часу розробки і підвищення

якості виробів. Без САПР неможливо виробляти сучасну складну техніку. Рівень розвитку САПР, кількість робочих місць САПР визначають рівень розвитку суспільства та стратегічний потенціал нації. В системах САПР конструктор сприймає на екрані зображення деякого об'єкта і може інтерактивно вносити зміни в зображення цього об'єкта. Такими змінами можуть бути введення і редагування окремих елементів, а також задання числових значень деяких параметрів для елементів бази даних. СУБД дають можливість одержувати тривимірне зображення будь-якої деталі, її проекції з усуненням невидимих ліній, проводити повний аналіз цих деталей. Провідні інженерні компанії прагнуть до повного цифрового представлення різноманітних конструкцій.

Використання КГ у проектуванні дозволяє вивчити властивості та поведінку модельного об'єкта, а потім уже після детального вивчення приступити до його реального виготовлення. Так, усередині 90-х років на суперкомп'ютері фірми "Фольксваген" був спроектований найбільш економічний і екологічно чистий автомобіль "Лупо ТДІ".

Важливим досягненням КГ є можливість імітувати роботу складних механізмів та здійснення перевірки на віртуальному макеті правильності функціонування всіх його елементів, проведення аналізу фактичних положень, що займають деталі механізмів у різні моменти часу.

Архітектура та будівництво є другою важливою сферою використання графічних систем проектування. Наприклад, фірма McDonalds з 1987р. використовує комп'ютерну графіку для архітектурного дизайну, розміщення посадкових місць, планування приміщень тощо. Відомо ряд ефективних приміненнь у проектуванні стадіонів, спортивних закладів тощо.

Отже, системи інженерної комп'ютерної графіки призначені для

- автоматизації креслярсько-графічних та конструкторських робіт,
- моделювання в реальному часі 2D-, 3D- об'єктів;
- формування різних поверхонь та їх перетинів;
- забезпечення промислового дизайну;
- наочної візуалізації компонентів та систем механічних, електричних та інших конструкцій, пристроїв і приладів;
- підвищення якості та швидкості проектування.

Типовими прикладами таких систем є САД-системи (Computer Aided Design – системи конструювання):

- AutoCAD – система для автоматизації розробки та виконання проектно-конструкторських, розрахунково-графічних робіт (графічне ядро більшості САПР);
- ProEngineer, Engineering Geometry Assistant – системи для розв'язування геометричних проблем;

- VariCAD, ArchiCAD ArcView – системи для розробки архітектурних проектів тощо.

2. Системи наукової комп'ютерної графіки призначені для проведення наукових досліджень та експериментів, для візуалізації наукових результатів у вигляді статичних або динамічних зображень, що інтерпретують великі масиви даних, для моделювання поведінки різних об'єктів, фізичних процесів (наприклад, швидкоплинних фізичних процесів). Такий реальний фізичний процес може відбуватися впродовж кількох мікросекунд і безпосереднє їх спостереження в реальному часі неможливе. Натурний експеримент часто дорогий і дає мало інформації. Тому розширити можливості дослідників можна шляхом проведення якісного модельного експерименту з екранною візуалізацією на комп'ютері.

Основними класами систем наукової комп'ютерної графіки є системи розв'язування науково-технічних задач (MatCAD, MatLab, Mathematica, Maple, Statistica, StatGraphics Plus та ін.). Спеціалізованими програмами наукової графіки для побудови кривих і поверхонь на основі таблиць із числовими даними є програми Grapher і Surfer.

У процесі розв'язування наукових задач із використанням ілюстративної КГ розв'язки одержують у вигляді наочних графічних ілюстрацій (наприклад, картину розподілу температури для нерівномірно нагрітої пластини). Візуалізація числових даних, що описують розподіл забруднення території, дозволяє в цілому уявити собі екологічну ситуацію або картину розповсюдження забруднення від деякого джерела забруднення. Візуалізація хімічних і атомних реакцій, поведінки плазми дозволяє краще зрозуміти механізм і закони протікаючих процесів. Засобами комп'ютерного відображення даних хіміки вивчають складні молекули білків. Візуалізація різних характеристик є ефективним (а іноді єдиним) методом розв'язування задач (наприклад, знаходження особливих розв'язків диференціальних рівнянь). Дедалі більшу роль починає відігравати інтерактивна комп'ютерна графіка у фундаментальних наукових дослідженнях. Людське пізнання користується двома механізмами мисленнями. Один із них – це можливість працювати з ланцюгом символів (алгебраїчне мислення), другий спосіб – це працювати з образами. Вони володіють більшою інтегрованістю, ніж символічні представлення. Без образних представлень ми не змогли б уявити собі світ в його повноті. Одна із функцій КГ – це використання одного з важливих механізмів людського пізнання – образного мислення, тобто КГ властива когнітивна функція (когнітивна – від англійського *cognitive*, що означає сприяти пізнанню). Когнітивна графіка відрізняється від ілюстративної тим, що її основною задачею є створення моделей представлення знань (когнітивних моделей).

Когнітивна функція графіки полягає в тому, щоб за допомогою формування зображень отримати нове знання, для якого не існує не тільки текстового символного опису, але воно не існує навіть у голові дослідника. Потім на основі спостережуваних образів картин можна перейти до формулювання відповідної гіпотези про механізми і процеси, які сховані за динамікою цих картин. Когнітивна графіка тільки зараз формується, вона зумовлює появу людино-машинної технології пізнання для розв'язування складних слабо формалізованих задач.

3. При великих обсягах інформації потрібно мати швидкий доступ до неї та легку форму її розуміння. Один із важливих аспектів інформаційної технології – розв'язування інформаційних задач методами машинної графіки. Графічна форма подання інформації має переваги – це наглядність, ємність, висока швидкість сприйняття. Більшу частину інформації про навколишній світ людина сприймає візуально. Інформаційні пошукові системи при роботі з базами даних використовують засоби машинної графіки. Існують графічні бази даних, де зберігаються графічні образи (малюнки, ілюстрації, сліди фізичних частин, дактилоскопічні відбитки), геометричні бази даних, де зберігаються геометричні об'єкти (описи деталей, структури молекул), картографічні бази даних, які містять карти місцевості. Для роботи з графічними базами даних створені відповідні СУБД (наприклад, SDMS). Ці СУБД забезпечують огляд інформації, можливість інтерактивної роботи з графічними об'єктами. У наш час стають популярними геоінформаційні системи (ГІС) для роботи з картографічними базами даних (із географічною інформацією) [27]. Типовими для кожної ГІС є такі операції: введення та редагування об'єктів з урахуванням їх розміщення на поверхні Землі, формування цифрових моделей, запис у бази даних, виконання різноманітних запитів до баз даних, аналіз множини об'єктів, розміщених на деякій території, візуалізація об'єктів, що знаходяться на поверхні Землі, причому візуалізацію треба виконувати з різним ступенем деталізації як для Землі загалом, так і в межах окремих ділянок.

4. Ілюстраційна комп'ютерна графіка дозволяє використовувати персональний комп'ютер як інструмент для художника. Спеціальні програми перетворюють комп'ютер у віртуальну майстерню художника. Ілюстраційна графіка має практичні застосування в усіх галузях людської діяльності: промисловості, сільському господарстві, бізнесі, менеджменті, телебаченні, пресі, освіті, поліграфії, шоу-бізнесі тощо. Системи ілюстраційної графіки призначені для створення та художньої обробки різних ілюстрацій (малюнків, фотографій, карт тощо). Користувач систем ілюстраційної графіки має можливість

- вводити зображення;
- будувати власні зображення шляхом компоювання інших зображень;
- редагувати зображення;
- моделювати форму, розташування та властивості об'єктів;
- анімувати зображення;
- створювати ефекти об'ємності тощо.

Типовими прикладами таких систем є Illustrator, Corel Draw, Photoshop, Painter, 3DStudio MAX, Bryce 3D, Poser, PowerPoint, Hyper Method, Director, Free Hand, Premiere тощо. Системи ділової комп'ютерної графіки призначені для наочного графічного зображення табличних даних (наприклад, для зображення даних у вигляді графіків, діаграм). Прикладами таких систем є MS Excel, Quattro Pro, SuperCalc, PowerPoint тощо.

5. Геометричне моделювання – це розв'язування різних геометричних задач у 2D-вимірному, 3D-вимірному та n -вимірному просторах. Поява комп'ютерів сприяла автоматизації геометричного моделювання з використанням засобів машинної графіки. Користувач має можливість, застосовуючи геометричну мову програмування, оперувати поняттям геометричного об'єкта та здійснювати операції над геометричними об'єктами, тобто користувач оперує категоріями, а результати роботи програми одержуються як у символічній, так і в графічній формах. Для розв'язування задач із геометричними об'єктами створені спеціальні інструментальні засоби – інтегровані системи машинної геометрії і графіки. Ці системи є сукупністю процедур формування та перетворення графічних образів, визначення їх параметрів. Прикладами таких інструментальних засобів є системи ГЕОМАЛ, СИМАК.

6. Комп'ютерна анімація (від англ. *animation* – оживлення) – це сукупність засобів та методів, зорієнтованих на одержання на екрані ЕОМ динамічних графічних зображень, тобто з пересуванням та зміною форми зображень. Для створення анімації генеруються послідовність кадрів, що дещо відрізняються один від одного, і в такий спосіб створюється ілюзія руху та зміна форми об'єктів.

Методи анімації базуються на модифікаціях такого алгоритму:

- виведення спрайта на екран;
- стирання спрайта;
- виведення з деяким зсувом іншого варіанту спрайта.

Спрайт – це логічно завершений елемент зображення на екрані ЕОМ, який рухається.

Часто анімацію реалізують на основі використання відеосторінок. Виведення "зображень" на різні сторінки відеопам'яті забезпечує більш плавний рух спрайта та усуває миготіння екрана. Для цього застосовують

такий алгоритм:

- вивести зображення на сторінку 0, яка є видимою за замовчуванням;
- сформувати нове зображення на невидимій сторінці 1;
- зробити видимою сторінку 1;
- сформувати нове зображення на невидимій сторінці 0 і т.д.

Використовуючи чотири сторінки відеопам'яті, будь-яку об'ємну фігуру можна розглядати в плавному русі без миготіння екрана. Для цього достатньо коректно побудувати відповідну фазу руху фігури на чітко визначеній сторінці.

Системи комп'ютерної анімації надають користувачу можливості

- підготовки окремих статичних зображень (програмно або інтерактивно за допомогою графічних редакторів);
- опису динаміки графічних зображень;
- якісного виведення комп'ютерних фільмів на екран дисплея;
- редагування фільмів.

Вони застосовуються для автоматизації підготовки мультфільмів, відеокліпів, художніх фільмів із 3D-спецефектами. Один із перших суперкомп'ютерних центрів був створений саме на студії Діснея і використовувався для створення мультфільмів. Це пояснюється тим, що при створенні мультфільмів необхідно виконувати великий обсяг обчислювальної роботи для побудови зображень, що відповідають проміжним фазам руху персонажа.

Створення тривимірних світів активно входить у кіноіндустрію, особливо у сфері фантастики. Одним із перших фільмів був фільм „Зоряні війни”. Він був створений за допомогою комп'ютера GRAY. Далі були фільми „Термінатор-2”, „Вавилон-5” та десятки інших. У фільмі „Титанік” більша частина інтер'єру лайнера – не реальні декорації, а графічні зображення. До недавнього часу технології комп'ютерної графіки використовувалися для імітації сцен, створення екзотичних чудовиськ, нереальних фантастичних зображень і ефектів та інших елементів, які були лише фоном для гри живих артистів. У 2001 р. вийшов на екрани повнометражний фільм „Фінальна фантазія”, в якому все, включаючи людей, синтезовано комп'ютером – живі артисти лише озвучили ролі за кадром. Прикладом комп'ютерних анімаційних систем є системи ANIMA, ANIMATOR, МОНТАЖ, ASAS, 3DStudio MAX.

GIF Animator – це найбільш популярна програма для створення професійної анімації. Вона містить ряд ефектів і переходів, має зручний інтерфейс та можливість налаштування швидкості і розмірів анімаційного зображення, володіє широкими можливостями роботи з текстом.

Macromedia Flash – прийнятий стандарт для використання в анімації векторних зображень, дозволяє створювати високоякісну векторну анімацію (фільми, рекламні ролики для розміщення в Інтернеті).

3DStudio MAX – професійне програмне забезпечення для тривимірного моделювання й анімації, розробки комп'ютерних ігор та роликів.

7. Індустрія розваг активно використовує методи комп'ютерного синтезу для створення численних комп'ютерних ігор, в яких віртуальні персонажі діють у віртуальних комп'ютерних світах. Наступним кроком є системи віртуальної реальності, які створюють ілюзію присутності й участі людини в житті віртуального світу, який може бути моделлю існуючого або видуманого простору. *Віртуальна реальність* – це технологія, що забезпечує реалістичне моделювання оточуючої дійсності (3D-простору) та підтримує інтерактивну взаємодію з користувачем.

Основа систем віртуальної реальності – високопродуктивні графічні робочі станції, що володіють великою швидкістю та можливостями побудови високореалістичних зображень. Засоби відображення можуть бути різними – від звичайних моніторів із високою роздільною здатністю до екранів на всю стіну, чи стереоскопічних систем відображення. Розвиток засобів віртуальної реальності почався з досліджень по створенню авіаційних тренажерів. Зараз системи віртуальної реальності широко використовують для імітації функціонування складних систем. Важливими характеристиками віртуальних систем є вплив не тільки на зір і на слух, а й на інші органи відчуття за рахунок використання спеціальних шлемів, костюмів, сенсорів на тілі людини. Комп'ютерні графічні системи на базі моделювання відповідних середовищ, об'єктів та ефектів використовуються для тренування льотчиків, шоферів, диспетчерів, операторів складних хімічних, енергетичних установок, військових тощо. Наприклад, в кінці 70-х років для космічних кораблів “Шатл” з'явилися льотні тренажери, що базуються на КГ. Перші кроки в цьому напрямку були зроблені ще в 60-х роках. Ці графічні системи дають можливість тренувати швидкість реакції пам'яті, логічне мислення. Прикладом імітаційного тренажера є парний тренажер повітряного бою. Місце кожного пілота оснащено імітаційним екраном і ричагами управління за допомогою яких можна змінити умови спостереження. В пам'яті ЕОМ закладені необхідні дані. Під час бою, змінюючи положення ричагів, інформація поступає в ЕОМ і в режимі реального часу переформатовується видима картина.

8. Графічні системи використовуються для створення творів мистецтва, музейної та реставраційної діяльності, оздоблення інтер'єрів, підготовки рекламних роликів та відеокліпів, різних заставок і телеефектів, оформлення Web-сторінок, створення художніх відеофільмів та спец-

ефектів в кіно, спрощення та скорочення часу підготовки друкованих матеріалів різного призначення, підвищення якості друкованої продукції, оформлення обложок книг і журналів.

9. У медицині КГ використовується, наприклад, для автоматизованого проектування імплантів (особливо кісток і суглобів), що дозволяє скоротити час проведення операцій. Анатомічні моделі КГ використовуються в медичних дослідженнях і в хірургічній практиці. Широке розповсюдження отримала КГ в економіці, рекламному бізнесі, криміналістиці та інших сферах людської діяльності. У наш час розвиток Internet істотно розширює сфери застосування КГ. Із цією метою вдосконалюються методи передачі візуальної інформації, розробляються нові графічні формати.

Контрольні питання та завдання

1. Який режим роботи називається інтерактивним?
2. Назвіть основні задачі КГ.
3. Що таке примітив? Наведіть приклади примітивів.
4. Що характеризують параметри та атрибути примітивів?
5. Які є способи задання геометричних об'єктів?
6. Що таке поліедр? Як задати модель поліедра?
7. Назвіть платонові тіла. Які їхні властивості?
8. Що таке сцена? Яка необхідна інформація для задання сцени?
9. В чому полягають недоліки явної та неявної форм задання кривої?
10. Які переваги має параметрична форми задання кривих і поверхонь?
11. Які лінії називаються циклічними/гвинтовими? Наведіть приклади.
12. Яку роль відіграють полігональні сітки в комп'ютерній графіці?
13. Що таке морфінг?
14. Назвіть області/приклад застосування КГ.
15. Коротко охарактеризуйте можливості найбільш відомих графічних пакетів.
16. Дайте визначення інженерній, ілюстраційній та діловій комп'ютерній графіці. Опишіть напрямки їх практичного застосування. Наведіть приклади задач для яких вони використовуються.
17. Наведіть приклади програм, які спеціалізовані на науковій комп'ютерній графіці. Які основні задачі вони розв'язують?
18. Охарактеризуйте поняття комп'ютерної анімації. Опишіть іструментарій мов програмування високого рівня для створення анімаційних зображень.
19. Що таке віртуальна реальність?
20. Назвіть основні задачі когнітивної графіки.
21. Наведіть приклади криволінійних поверхонь 2-го порядку.

Вправи і задачі для самостійного виконання

1. Розробіть алгоритм аналізу простоти/опуклості многокутника.
2. Задайте алгоритм визначення внутрішньої області непростого многокутника.
3. На прямій задано n точок. Знайти пару найбільш близьких між собою точок.
4. Задано n точок на площині. Знайти пару найбільш близьких між собою точок.
5. Знайдіть координати всіх вершин тетраедра, якщо початок системи координат знаходиться в центрі описаної навколо нього сфери, а одна з вершин тетраедра знаходиться на осі z .
6. Знайдіть на поверхні сфери чотири рівновіддалених між собою точки. *Вказівка. Ці точки – вершини вписаного тетраедра.*
7. Розробіть алгоритм визначення видимих та невидимих граней платонових тіл.
8. Виведіть рівняння тора (тор утворюється обертанням кола відносно осей координат).
9. Виведіть рівняння траєкторії руху точки кола радіусом r_0 , якщо воно котиться зовні/всередині еліпса з півосями a та b .
10. Розробіть алгоритм все більш точнішої апроксимації сфери многогранною трикутною поверхнею. Для цього візьміть на початку тетраедр і кожну його грань середніми лініями розбийте на чотири трикутники. Вершини нових трикутників із центра сфери спроектуйте на сферу і на цих точках побудуйте нові трикутні грані, які краще апроксимують сферу і т.д.
11. Побудуйте сферу мінімального радіуса, що обмежує n точок в просторі?
12. За матеріалами комп'ютерних періодичних видань проаналізуйте останні досягнення в галузі практичних застосувань комп'ютерної графіки. Наведіть декілька прикладів.

Тема 2. Технічне та програмне забезпечення комп'ютерної графіки

Швидке зростання функціональних можливостей комп'ютерної техніки та розробка відповідного програмного забезпечення створило базу для подальшого розвитку комп'ютерної графіки.

2.1. Технічне забезпечення комп'ютерної графіки

До апаратного забезпечення комп'ютерної графіки належать: засоби введення і виведення графічної інформації, обчислювальні засоби, засоби зберігання графічної інформації тощо. Більшість пристроїв КГ растрові. Растрові пристрої формують зображення з окремих точок, тому важливою характеристикою растрових пристроїв є роздільна здатність, яка визначається розмірами точок і можливостями їх розміщення.

2.1.1. Пристрої введення графічної інформації

Основні класи пристроїв введення графічної інформації – це дигітайзери, пристрої сканувального введення, клавіатура, маніпулятори тощо.

Дигітайзери. Назва *дигітайзер* походить від англійського слова *digit* (цифра) і означає пристрій, який здійснює аналогово-цифрове перетворення.

Дигітайзери (або графічні планшети) застосовуються для поточного координатного введення зображень у комп'ютер. Уведення інформації відбувається за допомогою спеціального пера (стилусу) або координатного пристрою з прицілом (його під'єднують кабелем до планшета). Графічні планшети чутливі до натиску, швидкості ведення і нахилу стилуса.

У найдосконаліших п'єзоелектричних дигітайзерів робоча поверхня планшета має тактильну чутливість на базі п'єзоелектричного ефекту. Завдяки цьому введення інформації відбувається без спеціальних пір'їн. П'єзоелектричні дигітайзери дозволяють креслити так само, як на креслярській дошці.

Для перенесення зображення з плівки в комп'ютер використовують спеціальні дигітайзери. За допомогою таких дигітайзерів у кіностудіях можна вводити в комп'ютер фотографії акторів, пейзажів тощо.

Пристрої сканувального введення (сканери, цифрові відеокамери та фотоапарати). Сканувальні пристрої здійснюють оптичне введення інформації та автоматичне перетворення її в цифрову форму.

Сканери – це пристрої для введення в комп'ютер чорно-білих або кольорових зображень безпосередньо з паперового документа, тобто сканер перетворює зображення в його цифрову форму (по точках) і передає цей образ у комп'ютер.

Сканування – це переведення паперових документів в електронну цифрову форму, тобто це процес оцифрування зображень. У результаті сканування документа створюється растровий графічний файл, в якому

зберігається зображення вихідного документа. Цей набір пікселів ще не є документом в електронній формі, – це файл графічного формату (наприклад bmp, tiff, gif, psx, jpeg). Якщо оригінал містив текст, то відсканований файл не може бути прочитаний текстовим редактором. Текст відсканованих документів потрібно ще розпізнати. Це здійснюють програми розпізнавання тексту. Прикладом програми, що забезпечує високу якість розпізнавання тексту на різних мовах, є програма FineReader.

Принцип дії сканерів базується на освітленні паперового документу та перетворенні відбитого світла в цифрову форму. Найуживанішими зараз є планшетні сканери. У планшетних сканерах папір кладуть на спеціальну поверхню. Лінійка фотоприймачів переміщується відносно паперу і так здійснюється сканування документа та введення його в комп'ютер. Розрізняють чорно-білі (для введення тексту і рисунків, виконаних контуром), напівтонові, в яких кольори замінюються різним тоном сірого кольору, і кольорові сканери.

Якість сканера суттєво залежить від його *роздільної здатності* (максимальна кількість точок, яку здатний розрізнити сканер), яка має два показники: горизонтальна роздільна здатність та вертикальна. Горизонтальна роздільна здатність залежить від густини фоторецепторів на лінійці фотоприймачів, вертикальна роздільна здатність визначається мінімальним кроком зсуву каретки вздовж оригінала, наприклад, 600×800 dpi. Роздільну здатність сканера вимірюють кількістю точок на дюйм – dpi (dots per inch). Першу величину ще інколи називають оптичною роздільною здатністю, а другу – механічною. У сучасних звичайних моделях сканерів ці характеристики досягають 1200×2400 dpi, а в професійних – 2400×4800 dpi і більше. Це основна характеристика сканера. Часто використовується інша характеристика – лінійна роздільна здатність. Під *лінійною роздільною здатністю* розуміють максимальну кількість точок однакового розміру, які можна розмістити на горизонтальному чи вертикальному відрізку зображення одиничної довжини.

Яка роздільна здатність потрібна сканеру? У випадку тексту для подальшого розпізнавання в програмі FineReader – 300 dpi в монохромному режимі, для простого кольорового друку – 300 dpi, для фотодруку – 600 dpi, для збереження зображень і перегляду їх тільки на комп'ютері – 200 dpi.

За точність передачі кольорів відповідає другий показник – *розрядність сканера* (глибина кольору), що вимірюється в бітах. Глибина кольору вказує на кількість кольорів, яку сканер може розпізнати. Наприклад, розрядність 8 бітів відповідає тому, що сканер може розпізнати 256 кольорів, або 256 градацій сірого кольору, 10 бітів – уже 1024

градації, 24 біти відповідають 16,7 млн. кольорів (зрозуміло, що в побуті в такій кількості кольорів немає потреби, хоча виробники сканерів уже заявили про 48-бітну розрядність домашніх сканерів). Але, як правило, додаткові біти необхідні тільки сканеру, а в комп'ютер передаються лише 24 біти.

Для зв'язку сканера з комп'ютером використовують спеціальні 8- або 16-розрядні плати, що під'єднані до шини ISA, або стандартний інтерфейс LPT. Більшість сучасних сканерів постачається з інтерфейсом USB, який має велику швидкість передачі даних. Взаємодія сканера з комп'ютером забезпечується спеціальним індивідуальним набором драйверів або через стандартні драйвери TWAIN-інтерфейса. Стандарт TWAIN – це стандарт обміну між прикладною програмою та зовнішнім пристроєм. Сканер формує інформацію в растровому вигляді і далі проводить векторизацію графічної інформації, оскільки зберігання інформації в растровому вигляді вимагає багато пам'яті.

Цифрові камери призначені для введення зображення безпосередньо з оригіналу в комп'ютер. Побутові цифрові фотоапарати з'явилися на початку 1990-х років (це були чорно-білі пристрої), і зараз цифрові технології розвиваються настільки стрімко, що цифрові фотокамери за якістю зображень та ціною наздоганяють традиційні (плівкові). До цифрових камер належать і Web-камери, які використовуються для організації відеоконференцій в Інтернеті.

В цифровому фотоапараті зображення через об'єктив проектується не на фотоплівку, а на світлочутливу матрицю. Далі значення електричних зарядів оцифровується і формується матриця пікселів. Якість зображення залежить від роздільної здатності. У професійних камерах роздільна здатність може досягати 3264×2448 пікселів і більше, тобто це десятки мегапікселів (1 Мрпх = 1 мільйону пікселів). Потім цифрове зображення записується в FLASH-пам'ять і зберігається в графічному форматі JPEG, TIFF. Цифрові фотокамери обладнані LCD-екранами, які дають можливість одразу переглянути відзняті кадри. Цифрові фотокамери можуть бути під'єднані до ПК або до фотопринтера. Файл із цифровим зображенням на комп'ютері можна редагувати, ретушувати, споряджувати спецефектами, кадрувати, передавати каналами зв'язку тощо.

2.1.2. Пристрої виведення графічної інформації

Існує багато різноманітних пристроїв для виведення графічних зображень. Найбільш типові – електронно-променеві трубки (монітори), дисплеї на рідких кристалах, графобудівники, матричні, струменеві та лазерні принтери тощо.

Монітор є основним пристроєм оперативного виведення інформації. Найбільш розповсюджені протягом багатьох років монітори на базі електронно-променевої трубки (CRT-монітори). Сьогодні випускаються монітори стандарту SVGA. Якість зображення на екрані монітора визначається як можливостями самого монітора, так і можливостями контролера SVGA (відеоконтролера).

Екран дисплейного монітора подається як набір дискретних точок, що утворюють регулярну прямокутну решітку – растр (від лат. *gastrum* – граблі). Точки растру формують зображення.

Основними параметрами монітора є розмір екрану, розмір зерна, швидкість оновлення зображень (частота кадрової розгортки), ступінь плоскості екрану (вища реалістичність зображення, менше відблисків).

Існує декілька стандартних розмірів діагоналі екрану: 15 дюймів (39 см), 17 дюймів (44 см), 19 дюймів (49 см), 21 дюйм (54 см) і т.д. Створення і редагування продукції комп'ютерної графіки вимагає, як мінімум, діагоналі екрану не меншої 17 дюймів. Інакше робоче середовище більшості графічних редакторів не поміщатиметься на екрані. Для професійної роботи з графікою використовують монітори з діагоналями 21 чи 22 дюйми.

Ще одним параметром, що визначає якість зображення (і відповідно, ціну монітора) є розмір зерна (0,21 – 0,29 мм). Чим менше зерно, тим якісніше зображення на екрані. Зерно – це мінімальна точка люмінофора (піксель), яка вимірюється в десятих частках міліметра. Як правило, для 15-дюймових моніторів розмір зерна складає від 0,25 мм до 0,28 мм. Величина зерна на 17-дюймових моніторах коливається в діапазоні 0,24 – 0,27 мм. Для сучасних CRT-моніторів величина зерна часто вимірюється віддалю між сусідніми точками люмінофора.

Близьким параметром до розміру зерна є *роздільна здатність*. Ця величина показує, скільки пікселів може вміститися на екрані. Роздільну здатність описують два числа, перше з яких показує кількість точок по горизонталі, друге – кількість горизонтальних ліній, наприклад 320×200.

Наведемо приклади деяких стандартних режимів:

640 × 480 для 14-дюймових моніторів;

800 × 600 для 15-дюймових моніторів;

1024 × 768 для 17-дюймових моніторів;

1280 × 1024 для 19-дюймових моніторів.

На практиці будь-який з цих моніторів може підтримувати і вищі роздільні здатності.

Іншою характеристикою моніторів є *частота вертикальної розгортки*, тобто частота оновлення кадрів, яку ще часто називають *частото-*

тою регенерації. Для комфортної роботи необхідно, щоб частота вертикальної розгортки складала не менше 85 Гц. Менша частота шкідлива для очей (миготіння на екрані швидко стомлює очі). Якщо частота вертикальної розгортки вища за 110 Гц, то людина вже не помічає ніякого мерехтіння зображення.

Горизонтальна частота розгортки показує кількість ліній, яку можна вивести на екран за 1 с. Для сучасних моніторів вона складає від 15 кГц до 100 кГц. Параметри моніторів пов'язані між собою, наприклад, якщо зменшити роздільну здатність, то зростає частота розгортки.

Істотними характеристиками для роботи з моніторами є ергономічні характеристики і характеристики безпеки. Ці характеристики для пристроїв комп'ютера визначаються стандартами ТСО.

Дисплеї зазвичай рекомендується розглядати з відстані, не меншої 0,5 м. Тому окремі пікселі стануть непомітні (людина не буде сприймати зображення як растрове) при роздільній здатності 200 dpi, оскільки на цій відстані пікселі вже не сприймаються як окремі точки, тому що відстань між ними $dp < 0,14$ мм. Зазначимо, що в сучасних дисплеях роздільна здатність становить 100 – 120 dpi. Для зображення на папері, щоб на відстані 30 см не були помітні окремі пікселі, цей параметр повинен бути ще вищим (до 300 dpi), що відповідає відстані 0,085 мм між точками.

Різновиди моніторів. Розрізняють два класи моніторів: світловипромінювальні (монітори з електронно-променевою трубкою) і світлопропускні (монітори на рідких кристалах). Монітори на базі електронно-променевих трубок досі залишаються неперевершеними в основних вимогах, що пояснюється високою якістю зображення, відсутністю інерційності зображення, більш високою швидкістю зміни зображення, більшим кутом огляду, дешевою технологією виготовлення тощо.

CRT-монітори отримують зображення від пучка електронів, що випускається електронною гарматою і управляється електромагнітним полем. В результаті попадання електронного пучка на люмінофор на ньому утворюється світла точка – піксель. Яркість пікселя залежить від енергії електронного пучка. Електронний промінь пробігає екран зліва направо і зверху вниз із певною швидкістю, послідовно формуючи множину пікселів, які, зливаючись, сприймаються як єдине ціле.

Для створення кольорового зображення в конструкцію монітора входять три електронні гармати („червона”, „зелена”, „синя”). Керування вертикальною і горизонтальною розгортками у них спільне, а керування яскравістю кольору – роздільне. Конструктивно електронні гармати налаштовані так, що їх промені попадають в позицію одного пікселя

одночасно. На поверхню монітора наносяться три види люмінофора, кожний з яких випромінює свій колір. Випромінювання трьох кольорів сприймається як суміш кольорів. Для точного попадання в задану точку люмінофора необхідно електронний промінь сфокусувати до заданих розмірів у площині екрану, тому перед люмінофором ставиться спеціальна маска-решітка, яка звужує пучок і зосереджує його на одній із ділянок люмінофора. Без решітки зображення було б розпливчастим (у сучасних моніторах використовуються кілька типів решіток).

Монітори на базі CRT поділяються на векторні графічні дисплеї з регенерацією зображення (рисування відрізками) і растрові скануючі дисплеї з поточковим виведенням зображення.

Монітори на основі дисплеїв із рідкими кристалами (LCD-монітори). В основі роботи LCD-моніторів лежать оптичні властивості рідких кристалів. Як і в звичайних моніторах, у LCD-моніторах зображення створюється за допомогою матриці пікселів (LCD-комірок), яка формується рідкими кристалами. Рідкі кристали знаходяться між двома скляними панелями і володіють оптичними властивостями, тобто під дією електронів їх молекули можуть змінювати свою орієнтацію і внаслідок цього змінювати інтенсивність світлового променя, що проходить через них. Роздільна здатність LCD-монітора фіксована.

У наш час характеристики LCD-моніторів значно поліпшилися. Це монітори з відмінною чіткістю та ідеальною якістю геометрії зображень. Вони компактні, мають гарний дизайн, не генерують електромагнітне випромінювання і безпечні для здоров'я людини.

Одна з основних переваг LCD-моніторів – відсутність миготіння та мала енергоємність. До недоліків LCD-моніторів відносять обмеженість діапазонів кутів зору (вертикального та горизонтального), тобто варто трохи повернути дисплей, як помітно зміниться яскравість і кольори. Випускають такі монітори компанії LG, BenQ.

Зовсім недавно з'явилися *плазмові дисплеї (PDP)*. Технологія PDP базується на світловому розряді, що утворюється при рекомбінації іонізованого газу. Заряджений газ, що називається плазмою, випромінює світло в ультрафіолетовому діапазоні, який, потрапляючи на люмінофор, змушує його частинки світитися, але вже у видимому для людини діапазоні. Плазмову панель спрощено можна уявити як матрицю з маленьких неонових лампочок, кожна з яких вмикається незалежно і може світитися з різною інтенсивністю.

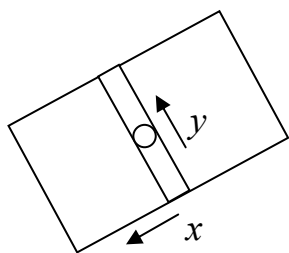


Рис. 2.1. Схема графобудівника

Принтери, перові графобудівники, плотери.

Принтер – це пристрій для одержання твердих копій цифрових зображень, для друку на папері тексту, графіків, зображень, креслень тощо. Нині найрозповсюджені лазерні, струменеві, матричні принтери. Лазерні принтери володіють широким діапазоном роздільної здатності: 300dpi, 600 dpi, 1200 dpi (для видавничих систем, що забезпечує досить високу якість зображення). Роздільна здатність принтера вимірюється в точках на дюйм. Наприклад, 300 dpi означає здатність принтера надрукувати на відрізок довжиною один дюйм 300 окремих точок. Сучасна технологія виготовлення друкувальних головок для струменевих принтерів забезпечує роздільну здатність до 300 dpi. Роздільна здатність матричних принтерів визначається розміром штирків, що формують зображення. Найбільш сучасні 48-голчаті матричні принтери мають роздільну здатність до 300 dpi.

Графобудівники – це пристрої траєкторного типу, в яких зображення на папір наноситься рухомим вузлом за допомогою пера з високою якістю зображень. Розрізняють планшетні та барабанні (рулонні) графобудівники. У планшетних носій зображення нерухомо закріплюється на площині графобудівника. Переміщення писального вузла на площині відбувається при русі кронштейна по осі x і при русі вузла з пером по кронштейну (по осі y) з високою точністю (до сотих долей мм) (рис. 2.1).

Графобудівники випускаються різних розмірів: від мініатюрних до великих (5–8 метрів у ширину і довжину), тому корпус автомобіля вимальовується в натуральну величину. Носієм зображення можуть виступати папір, фотопапір, тканина, лист металу тощо. Різним носіям відповідає множина різних пристроїв для рисування: кулькові ручки, олівці, фломастери, грифельні олівці, чорнильні пера, лазерні промені, інструменти для гравіювання, різні різачки тощо.

Таке різноманіття типів носіїв та писальних вузлів забезпечує багато сфер застосування планшетних графобудівників. Наприклад, різні модифікації планшетних графобудівників використовують для розкрою одягу, матеріалу. Якщо замість писального вузла використовується ніж, то такі різучі плотери називаються *каттерами* (від англ. cut – різати).

Графобудівники оснащені своїми електронними блоками для переміщення головки з пером у двох напрямках. Більш складні блоки мають лінійні і навіть кругові інтерполятори, які викреслюють дуги. Графобудівники мають власну буферну пам'ять і навіть власні процесори. Тому зображення, підготовлене на ПК, можна повернути або масштабувати безпосередньо на самому графобудівнику.

Основні характеристики графобудівників: розмір носія зображення,

параметри пам'яті, дискретність руху писального вузла, параметри точності, швидкість креслення (швидкість – до десятків м/с, тому навіть чорнило в перо подається під тиском), прискорення руху пера (досягає 4g).

На планшетних графобудівниках отримують досить високоякісні високоточні зображення та гарну передачу кольорів.

Барабанні графобудівники більш складні в механічному плані, ніж планшетні. Перо в них пересувається вздовж нерухомого кронштейна, а носій зображення закріплюється на барабані, який обертається (інколи може обертатися і сам носій за допомогою притискувальних роликів). Точність виведення інформації барабанними плотерами нижча порівняно з планшетними, але відповідає вимогам більшості застосувань. Ці плотери компактніші та можуть відрізати автоматично від рулону паперу аркуш потрібного розміру.

Крім перових плотерів, виділяють плотери, які використовують растровий спосіб створення зображення. Наприклад, струменеві плотери формують зображення шляхом розпилення чорнила на папір за допомогою дрібних форсунок друкувального вузла. Ці плотери можуть бути і кольоровими, в них використовується стандартна для поліграфії схема СМУК.

На ринку плотерів є ще клас плотерів, які використовують електростатичну та лазерну технологію створення електронного зображення.

Проектори. Для демонстрації комп'ютерних зображень на великих екранах, наприклад під час проведення лекцій чи презентацій, використовують відеопроєктори. Проектор під'єднується до комп'ютера паралельно з монітором або замість нього і сигнал на нього подається безпосередньо з відеокарти комп'ютера. Основними характеристиками проєкторів є світловий потік, контрастність, роздільна здатність, діапазон розмірів екрану, вхідні та вихідні інтерфейси тощо.

2.2. Програмне забезпечення

2.2.1. Основні поняття

Програмне забезпечення КГ дуже різноманітне – від простіших програм до професійних графічних редакторів, які різняться сферою застосування, мовами програмування, різними режимами роботи (пакетний, діалоговий) тощо.

До програмного та технічного забезпечення графічної системи висуваються досить високі вимоги. Щоб побачити проблеми, які виникають при створенні графічних зображень, що працюють у реальному часі, розглянемо задачу обертання об'ємного зображення, що складається з 1000 ліній зі швидкістю 15об/с.

Об'єкт з 1000 ліній подається матрицею 1000×4 однорідних координат крайніх точок відрізків. Обертання об'єкта задається шляхом множення цієї матриці на матрицю обертання розміром 4×4 . Для проведення такого матричного множення потрібно 16 тис. операцій множення, 12 тис. операцій додавання і 1 тис. операцій ділення дійсних чисел. Звичайний універсальний комп'ютер таке множення матриць може виконати за 0,1 с. Оскільки для того, щоб картинка на екрані була рухомою, потрібно перерисовувати її не менше ніж 30 разів за секунду, то в цьому випадку картинка не зможе неперервно рухатись. Отже, потрібно використовувати або більш потужний комп'ютер, або більш детальне програмування множення матриць для зменшення часу виконання програми.

Для створення прикладного графічного ПЗ застосовують такі класи інструментальних засобів [5]:

- стандартні графічні пакети (графічні редактори, системи автоматизованого проектування);
- стандартні програмні системи, що не спеціалізуються на графіці, але підтримують певний набір графічних функцій (наприклад, офісні пакети);
- мови програмування високого/низького рівня;
- синтаксичні графічні розширення універсальних мов;
- спеціалізовані графічні мови;
- графічні засоби операційних систем;
- бібліотеки графічних функцій.

Жоден із вказаних класів інструментальних засобів не є універсальним, а зорієнтований на свої класи задач і відповідних користувачів.

Системи, що належать до перших двох класів, пришвидшують процес створення графічного ПЗ. Вони зорієнтовані на користувача, що не має професійної підготовки з програмування.

2.2.2. Найвідоміші графічні редактори

Графічні редактори – це прикладні програми, призначені для створення й обробки графічних зображень на комп'ютері в діалоговому режимі. Професійний графічний діалог передбачає, крім уведення даних, вибір ліній, графічних елементів, інструментів із таблиць меню із зображеннями типових елементів. Вказати на пункт меню – означає вибрати відповідну команду, яка виконає програму побудови зображення елемента конструкції. Наприклад, щоб зобразити певну деталь, користувач не рисує цю деталь, а тільки вказує місце її знаходження та параметри.

Нині існує велика кількість графічних програм для роботи з графікою. Усі програми обробки графіки можна розділити на дві групи:

растрові та векторні. Зробимо невеликий огляд цих програм.

Деякі графічні редактори для роботи з растровою графікою орієнтовані безпосередньо на процес створення зображень (наприклад, Paint). У них акцент зроблений на застосуванні зручних інструментів для створення зображень.

Інший клас растрових редакторів спрямований на професійну обробку готових растрових рисунків із метою поліпшення їхньої якості. До таких програм належить, зокрема, найдовершеніший і популярний редактор **Adobe Photoshop**. Цей пакет програм має потужні можливості для обробки готових зображень і їх модифікацій. Цьому сприяє величезний набір різноманітних фільтрів та ефектів. Пакет володіє засобами відновлення пошкоджених зображень, ретушування фотографій тощо. Компанія Adobe поновлює графічний редактор Photoshop приблизно раз в два роки. Одна з останніх версій Photoshop CS3 Extended містить всі можливості стандартної версії графічного редактора і, крім цього, додаткові інструменти. В CS3 Extended можна працювати не тільки з 2D-графікою, а й із тривимірними моделями, накладати текстури, створювати анімацію, відкривати відеофайли, працювати як у відеоредакторі тощо.

Corel PhotoPaint – не менш відомий графічний редактор для обробки растрової графіки, який конкурує з Adobe Photoshop. Ця програма також має різноманітні інструменти для роботи з графікою, різниця лише в інтерфейсі та швидкості виконання графічних операцій.

Photostyler – ще один приклад редактора для роботи з растровою графікою. Ці програми використовують практично всі художники і фотографи для досягнення різних художніх ефектів. Вихідний матеріал для обробки в цих редакторах може бути отриманий різними шляхами: скануванням кольорової ілюстрації, введенням зображення з цифрової відео- або фотокамери. Крім того, звичайний фотоапарат теж можна використовувати для одержання електронних зображень, придатних для обробки в графічному редакторі, оскільки на пунктах проявлення фотоплівки введена нова послуга – запис фотознімків на компакт-диски.

Особливим класом програм для роботи з растровими зображеннями є програми-каталізатори. Вони дозволяють проглядати графічні файли різних форматів, створювати на диску зручні альбоми. Зручною програмою цього класу є програма **ACDSee32**.

У тих випадках, коли основними вимогами до зображень є висока точність і чіткість форми або коли необхідно створювати штрихові рисунки застосовують редактори, призначені для роботи з векторною графікою. Для роботи з векторною графікою використовуються програми Adobe Illustrator, CorelDraw, Macromedia FreeHand та ін.

Adobe Illustrator призначений для підготовки документів, що містять векторні та растрові зображення, тобто логотипів, плакатів, буклетів, брошур, Web-зображень тощо.

CorelDraw – лідер серед професійних векторних графічних редакторів. Цей багатофункціональний редактор призначений для обробки і створення векторної та растрової графіки, для підготовки файлів для графобудівників. Він є головною програмою для дизайнерів, зайнятих у сфері поліграфії та реклами.

Одним із найпопулярніших і зручних пакетів у галузі комп'ютерної анімації та 3D-графіки є професійний векторний пакет 3DStudio MAX. Він призначений для моделювання як окремих реалістичних зображень так і високоякісних 3D-анімаційних сцен. Досить задати лише геометрію сцени, тип матеріалу, джерела світла, розміщення камери, а пакет сам побудує зображення сцени. Програма володіє широкими можливостями моделювання освітленості, поступового перетворення форм об'єктів NURBS-моделювання, застосування різноманітних фільтрів, алгоритмами розрахунку тіні, динамічних властивостей рухомих об'єктів, спец-ефектів тощо. Більша частина сучасної TV-реклами створюється саме засобами 3DStudio.

Macromedia Flash – відомий у світі векторний редактор для анімації графіки. Основною сферою його застосування є графіка для Інтернет і комп'ютерні ігри.

Maya – система тривимірного моделювання (аналогічна 3DStudio MAX) має поліпшену фрактальну графіку, яка дозволяє більш ефективно моделювати складні поверхні.

Bryce 3D – одна з перших програм для моделювання ландшафту (рельєфу, води, рослин).

Для автоматизації проектно-конструкторських робіт призначена векторна програма **AutoCAD**. Цей програмний комплекс розроблений фірмою Autodesk на сучасному рівні розвитку машинної графіки. AutoCAD є основою для реалізації високоефективних технологій у проектуванні і потужним засобом моделювання складних каркасних полігональних і об'ємних конструкцій. Ця програма реалізовує багато функцій інженерної графіки, автоматизує розв'язування задач, які виникають в процесі проектування і є світовим лідером тривимірної векторної графіки.

Найрозповсюдженішим пакетом САПР у країнах СНГ є комплекс програм **КОМПАС** (нині КОМПАС-5). КОМПАС являє собою САПР загального призначення. Ця програма зорієнтована насамперед на підприємства машинобудівної галузі і їй надали перевагу біля 1500 промислових підприємств СНД.

2.2.3. Мови програмування графіки

Найуніверсальнішим, хоча і трудомістким, способом програмування графіки є програмування графіки засобами мов високого рівня: Pascal, C, C++, Visual Basic, Java, Delphi, C++Builder, Visual C++ та інші. Вони мають широкі можливості для створення графічних зображень. Ці мови програмування містять стандартні бібліотеки візуальних графічних компонентів, за допомогою яких можна швидко створити привабливий графічний інтерфейс до будь-якої прикладної програми, графічний редактор тощо. При цьому залежно від використаних розробником засобів, що надають мови програмування, графічні програми можна класифікувати на такі, що

- базуються на використанні процедур та функцій графічних бібліотек;
- створюються засобами BIOS;
- реалізують пряме звернення до портів відеоадаптера і комірок відеопам'яті;
- використовують асемблерні вставки для виконання графічних дій.

Мови програмування є гнучким інструментарієм, який дозволяє створити ефективне та швидкісне графічне ПЗ. При цьому формування зображення на екрані може бути поєднане з різними розрахунками.

Програми для створення графічних зображень будуються з процедур та функцій графічних бібліотек, призначених для побудови окремих графічних елементів. Наприклад, при побудові графічного зображення деякої машини основна програма використовує підпрограми зображення деталей, які в свою чергу, використовують підпрограми зображення елементів (наприклад, болтів), які використовують зображення прямих, дуг тощо. Іншими словами, ми маємо ланцюжок програм і тому можемо говорити про різні рівні програмного забезпечення (3, 4 або більше).

На нижньому рівні знаходяться програми, пов'язані з графічними пристроями. Наприклад, ті, що прямо звертаються до портів та комірок відеопам'яті. З геометричної точки зору ці програми прості, але від них вимагається висока раціональність у роботі (наприклад, програма побудови відрізка). Оскільки ці програми пов'язані з графічними технічними засобами, то пишуться вони мовою Асемблер або мовами, наближеними до машинних. Використання цих мов дозволяє підвищити продуктивність графічних програм завдяки збільшенню швидкості роботи програм.

Програми другого рівня розв'язують ширше коло задач машинної графіки і геометрії, використовуючи підпрограми нижнього рівня. Ці програми не прив'язані до конкретних марок графічних пристроїв.

До вищого рівня належать підпрограми, пов'язані вже з предметною

галуззю, зокрема підпрограми викреслювання фасадів, створення машинобудівних деталей.

Окрім універсальних мов, для програмування графіки ще застосовуються графічні мови високого рівня. Мовні засоби графіки – проблемно зорієнтовані мови програмування. Виділяють такі групи графічних мов: процедурні, автономні, мови розширення.

Процедурні мови – це пакети (бібліотеки) графічних підпрограм, які доступні з програм на різних мовах. До процедурних мов належать ГРАФОР, СМОГ, ДИГРАФ, АТЛАНТ, OpenGL, DirectX. Викликаючи відповідні процедури, графічні зображення будуються з окремих елементів.

Автономні мови – це спеціалізовані мови, що мають власну граматику та засоби опису геометричної і графічної інформації. Основною областю застосування цих мов є автоматизація програмування для устаткування з ЧПУ, систем автоматизації проектно-конструкторських робіт, систем геометричного моделювання. Для автономних геометричних мов існують спеціальні транслятори. Такі мови зорієнтовані на лексику конструктора. Як приклад можна навести мови СИМАК, AutoLisp.

Мови розширення будуються на основі універсальних алгоритмічних мов за рахунок їх графічного доповнення. Наприклад, мова CINEMIRA на основі Pascal, системи GALA, Автокод для мови Fortran, система DIGOS для мови ALGOL тощо. В цих мовах визначені графічні типи даних (наприклад, тип figure для задання статичних графічних образів, тип animated для задання залежних від часу фігур) та відповідні операції над ними (наприклад, об'єднання, перетин, масштабування, обертання тощо). Такий підхід дозволяє використати всі наявні в базовій мові потужні засоби обробки даних.

2.2.4. Графічні засоби відеосистем

Використання програмістами графічних можливостей відеосистеми може здійснюватися різними способами.

1. Шляхом прямого звернення до портів відеоадаптера та комірок відеопам'яті. Це найнижчий рівень програмування, тобто це програмування на фізичному рівні. Такі прийоми дозволяють створити найбільш оптимальні за швидкодією програми, а інколи одержувати ефекти, які інакше не можуть бути одержані. Однак рекомендувати такі прийоми можна лише в особливих випадках або при створенні спеціалізованих бібліотек графічних функцій. Основні проблеми тут пов'язані з трудоемкістю програмування та переносимістю програм.

2. Наступним, більш високим рівнем програмування графічних задач є використання можливостей відео-BIOS для виконання простіших графічних операцій (наприклад, визначення графічного режиму, виве-

дення пікселя на екран тощо). При цьому трудоемкість програмування зменшується, ступінь переносимості програм збільшується і програмісту вже не потрібно оперувати регістрами, бітами, фізичними адресами. Функції BIOS викликаються як програмне переривання. Всі функції відео-BIOS обслуговує програмне переривання 10H. Викликаються переривання з програми за допомогою функцій:

```
int 10H //мовою Assembler,  
intr($10, R) // мовою Pascal,  
genintrrupt(0×10) // мовою C++ .
```

3. Використання спеціалізованих графічних інтерфейсів та бібліотек (наприклад, OpenGL), які підтримуються апаратними можливостями сучасних графічних процесорів.

4. Використання функцій операційної системи. Різні операційні системи в цьому плані мають різні можливості. В MS DOS практично немає графічних функцій, однак програміст має вільний доступ до апаратних засобів, зокрема до відеопам'яті. Операційна система Windows забороняє прикладним програмам безпосередній доступ до відеопам'яті, але містить кілька сотень графічних функцій – інтерфейс API (Application Programming Interface – інтерфейс прикладного програмування). Функції Windows API повністю забезпечують програміста набором засобів для розробки будь-яких програм. Вони можуть бути викликані з будь-якої мови програмування високого рівня, для якої існує компілятор для Windows. Підключення функцій Windows API здійснюється через інтерфейсний модуль Windows.pas.

Основою графічної підсистеми Windows є прикладний інтерфейс GDI та GDI+ (Graphic Device Interface, інтерфейс графічного пристрою), що являє собою набір стандартних функцій графіки, які забезпечують доступ до різних установлених у Windows графічних пристроїв, як-от екран, принтер, плотер. Інтерфейс GDI апаратно-незалежний, тобто дозволяє програмісту однаково працювати з будь-яким графічним пристроєм, а прикладній програмі при виведенні графіки використовувати одні й тіж самі функції GDI для малювання на різних пристроях.

В основі роботи GDI з графічними пристроями лежить поняття контексту пристрою (DC). Контекст пристрою (контекст відображення) – це логічний об'єкт системи Windows, який зв'язаний із фізичним пристроєм і замінює його у функціях виведення. З погляду прикладної програми, контекст пристрою – це віртуальний екран із необхідними атрибутами. Фактично це посилання на пристрій виведення, тобто ідентифікатор, присвоєний ОС даному пристрою. Коли функція GDI будує зображення в контексті пристрою, зв'язаний із ним драйвер (графічна система

Windows працює з зовнішніми пристроями через драйвери) перетворює операції малювання в команди конкретного фізичного пристрою, який інтерпретує вхідні команди і відтворює зображення. Найрозповсюдженішим пристроєм виведення є дисплей. Тобто контекст пристрою – це міст між програмою і драйвером пристрою. Контекст дисплея дозволяє інтерпретувати кожне вікно прикладної програми як окремий пристрій відображення.

Функції GDI вимагають як один зі своїх аргументів посилання на контекст пристрою. Тип такої величини – `hdc` (Handle Device Context, посилання на контекст пристрою). Після одержання посилання на контекст пристрою, графічне виведення в GDI здійснюється шляхом виклику з Windows-додатків функцій, що малюють графічні примітиви. GDI підтримує сотні графічних функцій, більшість з яких знаходиться в бібліотеці `gdi32.dll`. Наведемо деякі групи цих функцій:

- растри (функції створення растрів, пікселів, заливок);
- колір (функції, що управляють палітрою кольорів);
- координати (функцій перетворення координат);
- лінії і криві (функції виведення прямих ліній, кривих Безьє);
- прямокутники (функції побудови прямокутників);
- пера (функції для роботи з атрибутами виведення ліній);
- відсікання (функції, що визначають межі області виведення в контексті пристрою) та ін.

Проілюструємо приклади деяких функцій графічного виведення в GDI:

- `function Rectangle(DC:HDC;x1,y1,x2,y2:integer):bool` – виведення прямокутника з координатами $(x1, y1)$, $(x2, y2)$;
- `function Ellipse(DC:HDC;x1,y1,x2,y2:integer):bool` – виведення еліпса в прямокутнику з координатами $(x1, y1)$, $(x2, y2)$.

Дескриптор DC є ідентифікатором контексту пристрою і визначає пристрій для виведення зображення. В такий спосіб одні й ті самі функції виводять графічні зображення на пристрої різної фізичної природи.

Хоча для більшості прикладних програм вистачає можливостей і швидкодії GDI, для програмування, наприклад, комп'ютерних ігор необхідна швидка графіка, для якої GDI не підходить. Тому для програмування швидкої графіки та мультимедійних додатків створюються сучасні графічні інтерфейси. Одним із найбільш відомих графічних інтерфейсів є OpenGL. Цей інтерфейс являє собою бібліотеку графічних функцій. Він підтримується багатьма операційними системами (в тому числі й Windows) і виробниками графічних акселераторів. Другим відомим графічним інтерфейсом є бібліотека DirectX із підсистемою Direct3D для створення тривимірної графіки. Цей стандарт призначений лише для

операційної системи Windows. Відомі також розробки інших інтерфейсів (наприклад, інтерфейс Glide для графічних відеоадаптерів Voodoo).

2.3. Види комп'ютерної графіки

Незважаючи на те, що для роботи з КГ існує багато різних програм, розрізняють всього три види комп'ютерної графіки: растрова, векторна та фрактальна. Вони відрізняються принципами формування зображення при відображенні його на екрані монітора та при друкуванні на папір.

2.3.1. Растрова графіка

У растрових системах зображення розбивається на рядки, базовим елементом яких є точка. Растрова модель цифрового зображення – це прямокутна матриця елементів, кожен з яких у закодованому вигляді зберігає значення яскравості (кольору) відповідної точки зображення.

Растр – це сукупність точок (матриця елементів), заданих на дискретній плоскій прямокутній решітці. Якщо зображення екранне, то ці точки називають пікселями (від англ. picture element – елемент картини).

Піксель – це неділиме зерно зображення, тобто мінімальний елемент зображення на екрані, який може бути згенерований комп'ютером. Кожний піксель має свій колір. Сукупність пікселів різного кольору утворює зображення, тобто растрове зображення в пам'яті комп'ютера – це набір даних про колір пікселів, упорядкованих за рядками. Для опису розташування пікселів використовують різноманітні системи координат, найчастіше – систему цілих координат (x, y) із координатами $(0,0)$ у лівому верхньому куті екрану, де x – номер рядка, y – номер пікселя в рядку.

Відображення довільного об'єкта на цілочислову решітку називається розкладом його в растр. Наприклад, лінія в растровій графіці задається матрицею дискретних точок (пікселів). Тому на екрані можна зобразити лише апроксимацію лінії, причому кількість елементів лінії залежить від роздільної здатності екрану. Растровий алгоритм – це алгоритм, який враховує властивість растру для роботи програми, що реалізує графічні примітиви.

Відзначимо основні проблеми растрової графіки:

1. У растровій графіці для створення та збереження якісного зображення використовують великі масиви окремих точок. Наприклад, стандартна фотографія повинна містити приблизно 1000×1500 пікселів (1,5 млн точок) і, якщо на кодування однієї точки відводиться 3 байти, то такій фотографії відповідає масив даних, більший 4 Мб.

2. Другий недолік растрової графіки пов'язаний із масштабуванням зображення. Оскільки зображення складається з точок, то збільшення зображення приводить до збільшення розмірів пікселів, що

спотворює графічну ілюстрацію. Жодних додаткових деталей при збільшенні растрового зображення розглянути неможливо.

До переваг растрової графіки належить висока реалістичність зображень, тонка передача кольорів, можливість впливу на кожний піксель зображення. Растрову графіку застосовують здебільшого не для створення графічних зображень, а при їх обробці. Наприклад, при корекції, ретушуванні фотографій чи ілюстрацій, одержаних на сканері чи цифровому фотоапараті. Всі растрові програми умовно можна поділити на статистичні та динамічні. До статистичних редакторів належать Adobe Photoshop, Canvas, Adobe Photo Paint, Paint Shop Pro, до динамічних – AnimagicGIF, Photo GifAnimator та ін.

2.3.2. Векторна графіка

Растрові і векторні зображення суттєво відрізняються за способом представлення графічної інформації. *Векторні зображення* – це зображення, які побудовані з графічних примітивів, що задаються своїми характерними параметрами (для кожного примітиву свої параметри).

У растровій графіці основним елементом зображення є точка, а у векторній графіці – лінія (контур), яку ще називають і вектором. Контур може бути прямою або кривою лінією, замкненим або відкритим. Кожний контур має дві або більше опорних точки (вузли). Між двома вузлами міститься сегмент контуру. Форму контуру задають через опорні точки. Над контуром можна виконувати операції комбінування та об'єднання. У векторній графіці все складається з ліній. У растровій графіці теж є лінії, але вони розглядаються як сукупність точок, і чим довша растрова лінія, тим більше пам'яті вона займає. У векторній графіці обсяг пам'яті, яку займає лінія, не залежить від розмірів лінії, оскільки лінія подається у вигляді формули (кількох параметрів).

Оскільки лінія є основним об'єктом векторної графіки, а складніші об'єкти у векторній графіці складаються з простіших (ліній), то векторну графіку називають ще об'єкто-зорієнтованою графікою. У ній зображення створюється шляхом комбінації різних примітивів. Файли векторної графіки для створення зображення містять тисячі різних команд типу “рисує лінію від *A* до *B*”, набори параметрів, інформацію про колір, дані про шрифти, що можуть бути включені в рисунок.

Ключовим моментом векторної графіки є використання для опису об'єктів комбінації комп'ютерних команд та математичних формул. Для кожного об'єкта (або класу об'єктів) визначається набір параметрів, який визначає його зовнішній вигляд. Але хоча об'єкти векторної графіки зберігаються в пам'яті як набір параметрів, зображення об'єктів на екран виводяться у вигляді точок-пікселів, оскільки такою є будова екрану.

Перед виведенням кожного об'єкта на екран відбувається процес розкладання векторного зображення в растр тобто програма обчислює координати екранних точок зображення об'єкта. Тому векторну графіку ще називають і обчислювальною графікою.

Крім геометричних параметрів векторні об'єкти мають параметри кольору контура, кольору заповнення, текстури тощо. Для векторних об'єктів (коло, квадрат, лінія) колір стосується об'єкта в цілому. Колір об'єкта зберігається в його векторному описі. У растрових об'єктах потрібно зберігати інформацію про колір для кожного пікселя. Векторна графіка не має зазначених недоліків растрової графіки, тому векторна графіка “економна” в плані обсягів файлів, оскільки зберігає не саме зображення, а тільки параметри зображення. Так, для збереження лінії другого порядку у векторній графіці необхідно приблизно 10 параметрів (20 байтів). Складні композиції, які складаються з 1000 об'єктів, займають всього десятки або сотні Кб.

Найважливіший аспект векторної графіки полягає в тому, що можна змінювати розміри векторного рисунка без втрати його якості. У векторній графіці легко розв'язується питання масштабування. При збільшенні зображення можна розглядати і деякі деталі зображення, тобто маємо можливість змінювати розміри векторного рисунка без втрати якості. Це найбільша перевага векторної графіки.

Недоліком векторної графіки є її програмна залежність. Кожна програма зберігає дані в своєму форматі, тому зображення, створене в одному редакторі, як правило, не конвертується в формат іншої програми без похибок. Виключення складає лише файловий формат AI програми Adobe Illustrator. Формат AI підтримується практично усіма програмами векторної графіки.

Крім цього, оскільки основним об'єктом векторної графіки є лінія, то засобами векторної графіки складно створювати художні ілюстрації, вони не здатні показати оригінал так реалістично, як це дозволяє зробити растровий рисунок. Тому векторна графіка використовується не для створення художніх композицій, а для оформлювальних та проектно-конструкторських робіт. Векторні засоби широко застосовуються в рекламних агентствах, редакціях, видавництвах, студіях дизайну. Разом із цим векторна графіка знайшла широке застосування і в комп'ютерній анімації. Для збереження або передачі мультфільмів по лініях зв'язку не потрібна обробка всіх кадрів фільма, достатньо відправити лише перший та останній кадр фільму, форму траєкторії, швидкість показу кадрів тощо, а браузер користувача згенерує всі кадри для проглядання фільму.

Прикладами векторних програм можуть бути CorelDraw, Adobe Illustrator, AutoCAD, Macromedia Flash для роботи з двовимірною графікою та 3DStudio MAX, 3DImpact, Bryce3D, Xara3D, Alias Maya для тривимірної графіки.

2.3.3. Фрактальна графіка

Фрактальна графіка – це графічне зображення фрактальних множин. Під фракталом у КГ розуміють структуру, яка складається з частин, які в певному розумінні подібні цілому. Поняття фрактала, фрактальної геометрії з'явилося в кінці 70-х років і нині має широке застосування.

Фрактальна графіка, як і векторна є обчислювальною, однак базовим елементом фрактальної графіки є математична формула, тобто жодні об'єкти в пам'яті комп'ютера не зберігаються. Саме зображення зберігається у вигляді формули (програми), за якою воно будується. Одним рівнянням можна описати цілий об'єкт. Змінивши параметри, матимемо інший об'єкт (з'являється можливість ефективної анімації). Іншими словами, створення фрактальної художньої композиції полягає не в рисуванні чи оформленні, а в програмуванні. Програми фрактальної графіки автоматично генерують зображення (візерунки) шляхом математичних розрахунків. Тому саме завдяки компактності математичного апарату, необхідного для відтворення фракталів (за допомогою декількох коефіцієнтів можна задавати поверхні та лінії складної форми), вони знайшли широке застосування в комп'ютерній графіці.

Фрактальна графіка незамінна при генерації надзвичайних зображень та при ілюстрації складних неевклідових об'єктів, зразки яких схожі на природні. Фрактальними властивостями володіють багато об'єктів живої та неживої природи. Звичайна сніжинка (багатократно збільшена) є фрактальним об'єктом. Фрактальні алгоритми лежать в основі росту кристалів, коралів, рослин (кожна дочірня гілка повторює властивості гілки більш високого рівня). Фрактали використовують для генерування поверхні місцевості, штучних хмар, гір, поверхні моря, а також у фізиці (фізика твердого тіла), економіці (аналіз коливання курсу валют) тощо.

Фрактальна графіка – особливий напрямок *когнітивної графіки*. Когнітивна функція фрактальної графіки, як і графіки взагалі, полягає в тому, щоб за допомогою деякого зображення отримати нове знання, про яке дослідник навіть не здогадується. Тобто з'явилася можливість образного моделювання, а отже, й можливість впливати на творчу діяльність людини, оскільки образне мислення набагато ефективніше за мислення, що спирається на символічні перетворення.

У фрактальній математиці виникають чимраз новіші сфери застосування. Наприклад, перспективним напрямом є створення алгоритму

фрактального стиску графічної інформації (алгоритм був знайдений у 1991 р.). Фрактальний архіватор розпаковує значно швидше за ближчого конкурента JPEG не лише статичну інформацію, але й відео. У 1992 р. компанія Microsoft випустила диск-енциклопедію про квіти, дерева, живописні місця. На цей диск було записано 7 год звуку, 100 анімаційних роликів, 800 карт місцевостей, 7000 якісних фотографій.

Звичайний компакт-диск у 650 Мб без використання стиску може містити 56 хв звуку або 700 фотографій розміром 640 × 480 пікселів.

2.4. Графічні файлові формати

Для збереження графічної інформації в комп'ютерній графіці застосовуються щонайменше три десятки форматів файлів. Для ефективної роботи з графічним зображенням важливо зробити правильний вибір одного з численних графічних файлових форматів. Розмір графічного файла істотно залежить від характеру зображення та вибраного формату. Назви форматів файлів знаходяться в розширенні імені графічного файла.

Формат графічних файлів – це набір правил і методів, згідно з якими дані, що містять графічні зображення, записуються у файли. Графічна інформація у файлах кодується не так, як у пам'яті комп'ютера.

Різні графічні файлові формати реалізують різні способи опису графічної інформації у файлах та різні технології їх компактного подання. Типи форматів визначаються способом збереження і типом графічних даних. Неправильно вибраний формат може зайняти надто великий обсяг пам'яті або привести в процесі стиску графічної інформації до неприпустимої втрати якості зображення. Вибираючи формат файлів, необхідно пам'ятати, що даний формат повинен підтримуватися заданою сферою застосування. Графічні редактори, як правило, дозволяють працювати з графічними файлами кількох форматів, а також конвертувати файли з одного формату в інший.

Формати графічних файлів можна класифікувати за різними ознаками. Наприклад, їх можна розділити на [5]:

- ті, що кодують тільки одне зображення;
- ті, що можуть кодувати декілька зображень і при демонстрації з деякою частотою сприймаються як фільм.

Останні формати називаються анімаційними. Найпримітивніші анімаційні формати зберігають повні зображення, які показують одне за одним. Досконаліші анімаційні формати зберігають лише різницю між двома сусідніми зображеннями (фрейми). До анімаційних форматів, зокрема, належать ANI, DAT, FLC, FLI, MVE, VIC, SMP тощо.

Як подальший розвиток анімаційних форматів можна розглядати формати мультимедіа. Вони розроблені для того, щоб в одному файлі

зберігати дані різних форматів (графіку, звук, відеоінформацію тощо). Прикладом мультимедіаформату є формати AVI (Audio Video Interleaved – чергування аудіо та відео), MPEG, MPEG-1, MPEG-2, MPEG-4, що використовуються для роботи з зображеннями в системі Windows. У форматі AVI сектори відеоданих чергуються із секторами звукових даних. У форматі AVI підтримується ряд палітр: 8-бітна, 16-бітна, 24-бітна, 32-бітна.

Класифікацію файлів комп'ютерної графіки можна здійснити і за видами комп'ютерної графіки, тобто розрізняють растрові, векторні файлові формати та метафайлові формати:

- растрові формати (BMP, PCX, GIF, JPEG, PNG, RLE, DIC, TIFF, CAM, CLP, IMG, PSD, TGA, DCM, FIF);
- векторні формати (AutoCad DXF, AI, DWF);
- ті, що сполучають растрові та векторні зображення (EPS, PIC, CDR);
- метафайли (CGM, PDF, EMF, WMF), які, крім інформації про растрові та/або векторні зображення, містять також самі команди візуалізації (інструкції Windows), тобто метафайли є послідовністю команд інтерфейса GDI, які створюють результуючий малюнок на екрані.

Растрові формати служать для опису растрової графіки, що являє собою набір числових значень, які визначають колір окремих пікселів.

Векторні формати служать для збереження зображень у вигляді сукупності геометричних примітивів. Графічні формати цього типу складаються або зі списку примітивів, або інструкцій для побудови цих примітивів. Окрім цього, у векторному файлі зберігаються атрибути примітивів. Об'єкти складної форми утворюються з базових примітивів за допомогою різних операцій.

Розглянемо приклади деяких графічних форматів.

Формат BMP (від слова bitmap) широко використовується в ОС Windows для обміну растровими зображеннями між додатками. Цей формат досить відомий, його розуміють майже всі програми, що працюють під Windows. BMP-файл має просту структуру. Растрове зображення складається з елементарних точок. Тому в графічному файлі растрового формату BMP зберігаються координати точок зображення та значення їх кольору. В бітовому масиві послідовно записуються байти рядків растру. BMP-файл зберігає єдине зображення з 1, 4, 8 та 24 бітами на піксель. Растр тут зберігається майже в тому вигляді, в якому він записується в оперативну пам'ять для відображення та обробки, тому BMP-файли займають багато пам'яті, навіть невеликі кольорові зображення з роздільною здатністю 640 × 480 вимагають кількох мегабайтів.

Формат PCX. Цей растровий формат зручний для зберігання зображень типу ділової графіки (креслення, діаграми, схеми тощо). У

форматі PCX використаний один із варіантів алгоритму ущільнення RLE, що означає групове кодування. RLE – один із найдавніших і найпростіших алгоритмів ущільнення графіки, що базується на такій ідеї: якщо в деяких растрах трапляються ланцюжки з однакових пікселів, то у файлі замість цих ланцюжків зберігаються пари чисел – лічильник повторень та саме значення. Чим довші ланцюжки, тим більше ущільнення.

Формат GIF (Graphics Interchange Format) є одним із найвідоміших ущільнених форматів для зберігання та передачі файлів растрових зображень. Він був запропонований як незалежний від апаратного забезпечення засіб обміну растровими зображеннями в мережі Internet. Основна перевага цього формату – висока ступінь стискування без особливих втрат, що досягається застосуванням алгоритму ущільнення, який належить до класу LZW-алгоритмів. В алгоритмах класу LZW використовується словниковий метод ущільнення. Створюється словник, що містить повторювані послідовності символів (фрази), які зустрічаються в масиві, що кодується. Кожна фраза отримує код (індекс) у словнику. Кодування масиву символів виконується заміною фраз відповідними індексами зі словника.

У файлах формату GIF близько розміщені однакові за кольором точки групуються в горизонтальні лінії. Це дозволяє істотно зменшити об'єм графічного файла. GIF-формат ефективно стискує графічні малюнки з великими фрагментами однорідної заливки, але погано стискує фотографії, оскільки фотографії містять багато відтінків. Обмеження GIF полягає ще і в тому, що кольорові зображення не можуть бути записані в режимі більше ніж 256 кольорів, однак у багатьох випадках цього достатньо, наприклад для передачі графічних зображень в Internet.

Оскільки при візуалізації зображень у цьому форматі передбачено черезрядкове відображення (спочатку виводиться кожний восьмий рядок, потім – кожний четвертий і т.д.), то користувач може оцінити зображення за його частиною і перервати прийом зображення, не чекаючи виведення всіх рядків зображення. GIF-формат може містити не одне, а кілька растрових зображень, які браузері довантажують одне за одним із зазначеною у файлі частотою, тобто GIF-формат підтримує анімацію.

Формат JPEG або *JPG* (Joint Photographics Experts Group) – один з найрозповсюдженіших растрових форматів. Він застосовується для відображення фотографій та інших тонових зображень в електронних мережах. Він використовує ефективні алгоритми ущільнення, що сприяє значному скороченню обсягу файла (економить від 50% до 70% обсягу пам'яті), однак дає втрату інформації. У форматі JPG можна одержати файл у 500 разів менший за розміром ніж BMP. Це найменші за обсягом

графічні файли. Для цього реалізована ціла група алгоритмів стиску, зокрема алгоритм стиску, що збільшує розміри пікселів зображення, тобто утворює блоки з 8×8 пікселів і для кожного блока формує набір чисел. Перші кілька чисел представляють колір блока, а наступні числа відображають різницю між пікселями. Так зменшується розмір графічного файла, але при цьому губиться інформація, яка майже не відчувається оком. Оскільки під час стиску втрачаються частини інформації про колір, то в JPG-форматі не бажано зберігати зображення, для яких важливі всі особливості передачі кольорів. JPEG краще стискує растрові фотографічні зображення, ніж логотипи чи схеми. З меншими втратами стискаються зображення з високою роздільною здатністю (200–300 dpi). Більшість зображень в Internet подано форматом JPG.

Формат TIFF (Tagget Image File Format) розроблений для зберігання відсканованих зображень із високою роздільною здатністю (високою якістю) та для обміну документами між різними програмами і різними комп'ютерними платформами. TIFF дозволяє зберігати в файлі декілька зображень і може використовувати різні моделі кольорів, має найбільш широкий діапазон передачі кольорів – від монохромного до 32-бітного, підтримує багато методів ущільнення. TIFF – це підтримка швидкого доступу до окремих фрагментів зображення.

Формат CDR використовується програмою Corel Draw, яка на сьогодні є однією з найпопулярніших серед програм, що дозволяють створювати векторні зображення. CDR дозволяє записувати векторну й растрову графіку, а також текст. Одна з властивостей векторних форматів – відтворення масштабованих зображень без погіршення якості. Файли Corel Draw мають робоче місце до 45×45 м.

Формат PSD – це власний формат програми Adobe Photoshop, один з найпотужніших форматів збереження растрової графічної інформації. Підтримує 48-розрядне кодування кольору, різні колірні моделі. Але відсутність ефективного алгоритму стиску приводить до великого обсягу файлів.

Формат AI – це власний векторний формат програми Adobe Illustrator. Цей формат підтримують практично всі програми векторної графіки.

Формат DXF (Drawing Exchange Format) розроблено в 1982 р. для обміну кресленнями та іншими графічними документами в середовищі AutoCad. DXF зараз підтримується багатьма графічними програмами. Формат DXF-файла представляє собою повний опис креслення в текстовій формі коду ASCII.

Формат PDF – це формат представлення документів, він призначений для електронних публікацій і передачі графіки в мережі Internet.

У цьому форматі зберігаються документи, які можна тільки читати і не можна редагувати. Файл у форматі PDF може містити елементи, що забезпечують пошук і перегляд електронних документів, зокрема гіпертекстові посилання, посилання на мультимедійні файли, електронні заголовки. Більшість графічних пакетів дозволяють конвертувати свої документи в PDF-файли. Проглядати PDF-файли можна за допомогою програм Adobe Photoshop, Adobe Acrobat, Foxit Reader. Формат PDF апаратно-незалежний, тому виведення зображень може здійснюватися на різні пристрої.

Формат PNG є відносно новим форматом, що прийшов на зміну GIF. Цей формат використовує стиснення без втрат. Стиснуті PNG-файли, як правило, менші, ніж аналогічні GIF-файли.

Формат DjVu – це нова технологія стиснення зображення з метою розміщення в Internet відсканованих документів (книг, журналів, документації, зображень) високої якості. Зазвичай DjVu стискує в 5-10 разів краще, ніж JPEG, GIF для кольорових документів, у 3-8 разів краще, ніж TIFF для чорно-білих документів. Кольорові сторінки, відскановані з роздільною здатністю 300 dpi, можуть бути стиснені з 25 Мб до 30-100 Кб, чорно-білі – до 5-10 Кб. Зазначимо, що DjVu-плагін доступний для стандартних браузерів.

Існують програмні засоби, які дозволяють перетворювати файли з одного графічного формату в інший. Наприклад, кожного разу, коли векторний файл направляється на пристрій виведення (монітор), він підлягає операції побудови растру – перетворення зображення в окремі пікселі. При раструванні програма повинна розрізнити векторні об'єкти, а потім створити растрове зображення.

При обернених перетвореннях растрових файлів у векторні, растрові зображення трасуються за допомогою відповідних програм, які відстежують групи пікселів з однаковим або схожим кольором і створюють схожі групи векторних зображень. Прикладом програми трасування растрових зображень є програма CorelTrace.

Контрольні питання та завдання

1. Назвіть пристрої введення/виведення графічної інформації.
2. Вкажіть основні характеристики сканера, опишіть принципи роботи.
3. Порівняйте технології формування зображень на різних моніторах.
4. Назвіть основні характеристики сучасних моніторів.
5. Назвіть функції та характеристики графобудівників і принтерів.
6. Що таке роздільна здатність графічних пристроїв?
7. Коротко охарактеризуйте найпоширеніші графічні редактори.

8. Які класи інструментальних засобів використовують для створення графічного програмного забезпечення?
9. Які класи графічних мов ви знаєте? Наведіть приклади.
10. Назвіть найвідоміші API для роботи з графікою.
11. Що являє собою інтерфейс GDI? Що таке контекст пристрою?
12. Назвіть види графіки. Дайте їх порівняльну характеристику.
13. Що таке піксель, растр, роздільна здатність растру?
14. Що собою являє растрове/векторне зображення в пам'яті комп'ютера?
15. Назвіть переваги та недоліки растрової/векторної графіки.
16. Що є основним елементом векторної/ фрактальної графіки?
17. Дайте коротку характеристику програм для роботи з графікою.
18. Назвіть основні растрові та векторні графічні формати.
19. Охарактеризуйте основні методи стискання графічної інформації.

Вправи і задачі для самостійного виконання

1. Зображення з роздільною здатністю 100 dpi має розміри в 200 пікселів. Знайти фактичну висоту/ширину зображення. Які розміри матиме зображення, якщо роздільну здатність збільшити в 2 рази?
2. Сканується квадратне зображення зі стороною в три дюйми. Лінійна роздільна здатність становить 100 dpi. Скільки пікселів буде містити оцифроване зображення?
3. Зображення розміром 4×4 дюйми сканується з роздільною здатністю 300 dpi. Знайти загальну кількість точок такого оцифрованого зображення. Обчислити затрати пам'яті для різних графічних режимів:
 - а) True Color (24 біти на точку);
 - б) High Color (16 бітів на точку);
 - в) Line Art (1 біт на точку).
4. Необхідно відсканувати зображення шириною три дюйми на сканері з максимальною оптичною здатністю в 600dpi. Знайти кількість світлочутливих елементів лінійки фотоприймача, які будуть задіяні в цій процедурі. Скільки датчиків буде задіяно, якщо роздільну здатність сканера зменшити вдвоє?
5. Око людини з нормальним зором може розрізняти об'єкти з кутовим зміщенням β біля однієї хвилини. Якщо віддаль до об'єкта дорівнює d , то віддаль між об'єктами $s = d\beta$. Припустимо, що людина розрізняє пікселі відповідно до цього мінімального розміру. Оцініть роздільну здатність растру, в якому людина не побачить окремих точок растру. Обчислення виконати при 1) $d = 300$ мм, 2) $d = 500$ мм. З якої віддалі потрібно розглядати екран з роздільною здатністю 100 dpi, щоб людина не побачила окремих пікселів?

Тема 3. Колір. Моделі кольору

3.1. Природа кольору

Колір – це один із факторів світлового випромінювання. Поняття кольору в КГ є основним. Світло можна розглядати двозначно: як потік частинок різної енергії (тоді колір світла визначає енергія частинок) або як потік електромагнітних хвиль високої частоти (у цьому випадку колір визначається довжиною хвилі). Ми розглядатимемо світло як потік електромагнітних хвиль, який після взаємодії з оточуючим середовищем попадає в око, де в результаті фізичної і хімічної реакції виробляються електроімпульси, що сприймаються мозком людини. За допомогою хвильової теорії, висунутої Гюйгенсом у 1678 р., було пояснено багато властивостей світла, зокрема закони відбиття та заломлення.

Однією із хвильових характеристик світла є довжина хвилі – відстань, яку проходить хвиля впродовж одного періоду коливання. Електромагнітна хвиля характеризується також амплітудою. Вона визначає енергію хвилі (енергія пропорційна квадрату амплітуди). *Видиме світло* – це множина хвиль довжиною λ від 380 – 430 нм (фіолетовий) до 605 – 780 нм (червоний). Нагадаємо, що $1 \text{ нм} = 10^{-9} \text{ м}$. Світло належить до досить вузького діапазону електромагнітних хвиль. Поза цим діапазоном знаходяться, наприклад ультракороткі, інфрачервоні, ультрафіолетові, рентгенівські хвилі.

Електромагнітні хвилі видимого діапазону задають такі кольори:

380 – 430 нм – фіолетовий;

430 – 470 нм – синій;

470 – 500 нм – блакитний;

500 – 560 нм – зелений;

560 – 590 нм – жовтий;

590 – 605 нм – оранжевий;

605 – 780 нм – червоний.

Сама по собі електромагнітна енергія не має ніякого кольору, відчуття кольору виникає в результаті фізичних та хімічних процесів в оці та мозку людини. Різна довжина хвилі сприймається нами як різний колір. Видиме світло з найбільшою довжиною хвилі буде червоним, із найменшою – синім. При зміні довжини хвилі кольори плавно переходять один в один. Чисті кольори існують лише при певних довжинах хвилі (наприклад, чистий фіолетовий – при довжині 400 нм). Слід зауважити, що колір має і психофізичну природу, тобто сприйняття кольору носить суб'єктивний характер і залежить не тільки від фізичних властивостей світла, а й від інтерпретації світла зоровою системою людини.

На практиці рідко трапляється світло певної довжини хвилі. Як правило, всі джерела світла генерують коливання в широкому діапазоні (винятком є лише випромінювання лазера), тому світло є неперервним потоком хвиль із різними довжинами та різними амплітудами. Таке світло можна характеризувати енергетичною спектральною кривою $I(\lambda)$, де значення $I(\lambda)$ визначає вклад хвиль довжиною λ в енергію всього світлового потоку, що потрапляє на одиницю поверхні за одиницю часу (інтенсивність випромінювання). При цьому загальна енергія всього потоку світла за одиницю часу дорівнює інтегралу від спектральної функції по всьому видимому діапазону довжин хвиль

$$E = \int_E I(\lambda) d\lambda .$$

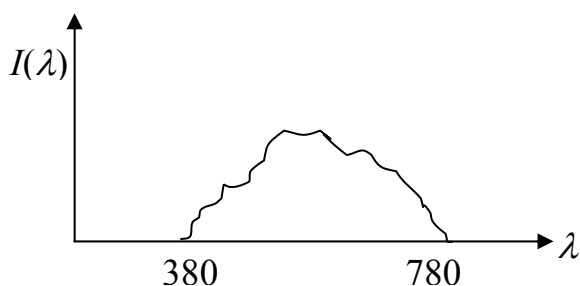


Рис. 3.1. Спектральна крива

Типова спектральна крива наведена на рис. 3.1. Отже, колір однозначно визначається спектральною функцією, але не навпаки.

Світло, що падає на поверхню об'єкта від певного джерела, частково поглинається об'єктом, частково відбивається від поверхні об'єкта і частково проходить через об'єкт. Частки поглинання, відбивання та пропускання кожної складової світла залежать від довжини хвилі λ та від властивостей матеріалу, з якого зроблений об'єкт. Саме поняття кольору пов'язане з тим, як людське око сприймає світло. Деякі предмети ми бачимо тому, що вони відбивають світло, а деякі – тому, що вони випромінюють світло. В темній кімнаті прекрасно видно предмети, які випромінюють світло, і не видно, що написано на папері.

Коли предмети випромінюють світло (такими є монітори), вони мають той колір, який ми бачимо. Коли якісь предмети, наприклад папір, відбивають світло, їхній колір визначається кольором світла, що падає на предмет, і кольором, який ці предмети відбивають.

Розглянемо, як відбувається сприйняття світла людським оком. Людське око дуже складна система. Коли очі дивляться на світ – світло попадає в око через рогівку, далі за допомогою кришталіка проектується на сітківку ока, де фоторецептори перетворюють світлову інформацію в імпульси у нервових волокнах.

Сітківка ока містить два принципово різні типи світлочутливих рецепторів: *палички*, які володіють широкою спектральною кривою чутливості, внаслідок чого вони не розрізняють довжини хвиль, а отже, і кольору та *колбочки*, які характеризуються вузькими спектральними

кривими і тому володіють чутливістю до кольору. У кожному оці знаходиться біля 6 млн. колбочок і 120 млн. паличок (приблизно 250 млн. рецепторів на два ока).

В основі трикомпонентної теорії світла лежить той факт, що в

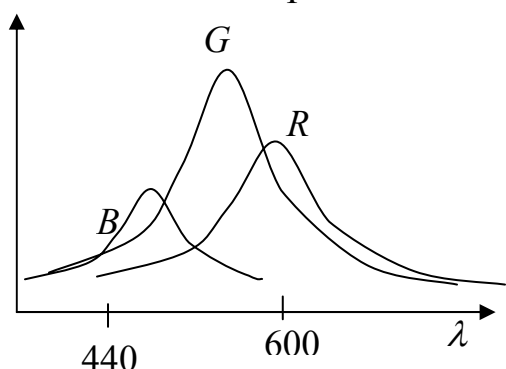


Рис. 3.2. Чутливість до кольорів

центральної частині сітківки знаходяться три типи чутливих до кольору колбочок, які відповідають за чутливість до довгих, середніх і коротких хвиль, тобто один тип колбочок реагує на зелений колір, другий — на червоний, а третій — на синій колір. Ці три кольори називаються основними (базовими). На рис. 3.2 наведено графіки функцій чутливості для всіх

трьох типів колбочок. В одних пік чутливості припадає на хвилі з короткою довжиною (448 нм — синій колір), у других — на хвилі середньої довжини хвилі (528 нм — жовто-зелений колір), у третіх — на хвилі з великою довжиною (667 нм — червоний колір). Видно, що найменша чутливість ока припадає на синій колір, а найбільша — на жовто-зелений.

Графік кривої, що відповідає за загальну чутливість ока до світла, формується в результаті додавання всіх кривих із рис. 3.2. Сумарна крива спектральної чутливості ока показана на рис. 3.3. Найбільша чутливість людського кольорового зору спостерігається для хвиль із довжиною 555 нм, що відповідають зеленому кольору. Тому з екрану ПК найкраще сприймаються жовто-зелені об'єкти. Крайні кольори важко сприймаються людським оком, яскраво-червоні та фіолетові кольори шкідливі для очей.

Якщо на всі три види колбочок діє однаковий рівень енергетичної яскравості, то світло буде білим. При низькому освітленні колбочки втрачають свою чутливість, зате зростає чутливість паличок, що забезпечує нашу здатність бачити при освітленні низького рівня, тому колбочки працюють вдень, а палички — вночі.

Монохроматичним називається випромінювання, спектр якого складається з єдиної лінії, що відповідає певній довжині хвилі.

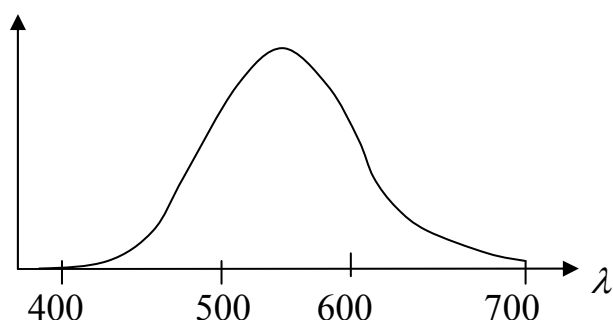


Рис. 3.3. Залежність чутливості людського зору від довжини хвилі світла

Досить якісним джерелом монохроматичного випромінювання є лазер. Колір монохроматичного випромінювання визначається довжиною хвилі.

Ньютон у 1666 р. показав, що білий колір можна подати як суміш всіх видимих монохромних кольорів, тобто рівномірним спектром суміші нескінченної кількості монохроматичних кольорів. Білий промінь світла (використовувався сонячний промінь)

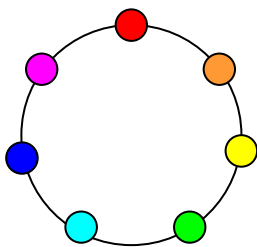


Рис. 1.4. Колірне коло Ньютона

спрямовували на скляну трикутну призму. Проходячи через призму, промінь заломлювався і на екрані давав кольорову смугу – спектр. Частинка видимого спектру має унікальне значення і називається *кольором*. У видимому спектрі наявні всі кольори, які плавно переходять один в один.

Видимий спектр складають мільйони кольорів. Ньютон розбив весь спектр на сім ділянок, які відповідали різним яскраво вираженим кольорам (червоний, оранжевий, жовтий, зелений, блакитний, синій і фіолетовий). Інша частина дослідів Ньютона показала, що біле світло можна зібрати з кольорів веселки. Ньютон довів, що будь-який колір утворюється шляхом змішування основних кольорів, узятих у певній пропорції.

Наука, що вивчає колір і його вимірювання, називається *колориметрією*. Вона описує загальні закономірності сприйняття кольору людиною. Основними законами колориметрії є закони змішування кольорів. Ці закони в найбільш повному вигляді були сформульовані в 1853 р. німецьким математиком Германом Грассманом. Наведемо їх.

1. Закон тривимірності. Колір виражається в тривимірному просторі. Це означає, що для його опису потрібні три компоненти. Довільні чотири кольори знаходяться в лінійній залежності. Іншими словами, для будь-якого кольору C можна записати рівняння:

$$C = k_1 C_1 + k_2 C_2 + k_3 C_3,$$

де C_1, C_2, C_3 – базисні, лінійно незалежні кольори, k_1, k_2, k_3 – коефіцієнти, що вказують на кількість змішуваних кольорів.

Перший закон можна трактувати і в більш широкому розумінні: необов'язково для опису кольору використовувати суміш кольорів, можна використовувати й інші величини, але їх обов'язково повинно бути три.

2. Закон неперервності. Якщо в суміші трьох кольорів один з них змінюється неперервно, а інші залишаються сталими, то колір суміші теж змінюється неперервно.
3. Закон адитивності. Колір суміші залежить тільки від кольорів ком-

понент і не залежить від їх спектральних складів, тобто змішувана компонента в свою чергу може бути отримана змішуванням інших компонент.

Для оцінки кольору людині зручно використовувати такі атрибути.

- **Тон (відтінок) кольору** асоціюється в людській свідомості з забарвленням предмета певним типом фарби. Тон кольору (відтінок) дозволяє людині відрізнити кольори (наприклад, зелений від червоного). Фізично тон кольору можна визначити переважаючою (усередненою) довжиною хвилі в спектрі випромінювання. Наприклад, світло, в якому переважає хвиля з довжиною 450 нм, буде сприйматися як відтінок синього кольору. Людське око спроможне розрізнити 350 тис. різних кольорів, хоча є й інші дані. Для характеристики відтінків вводять поняття яскравості і насиченості.
- **Яскравість (світлість)** визначається енергією, інтенсивністю випромінювання на одиницю площі і виражає кількість сприйнятого світла. Яскравість залежить від амплітуди електро-магнітних коливань. Білий колір має максимальну яскравість (100%), чорний колір дає повну відсутність яскравості (0%), тобто чим менша яскравість, тим темніший відтінок. Людське око може розрізнити близько тисячі різних рівнів яскравості.
- **Насиченість** характеризує рівень чистоти кольору і визначає кількість білого у відтінку того чи іншого кольору. Вона виражає співвідношення між основною домінуючою компонентою світла і рештою хвиль, що формують колір, тобто показує, наскільки даний колір відрізняється від білого. Чим вища насиченість, тим сильніше і ясніше відчувається тон кольору. Зниження насиченості приводить до того, що колір стає нейтральним без чітко вираженого тону. В ідеально чистому кольорі домішки білого відсутні (насиченість 100%). Якщо, наприклад, до чистого червоного кольору додати білий, то одержимо світлий блідо-червоний колір (низька насиченість). При насиченості, що дорівнює 0%, будь-який колір стає білим. Цей атрибут у людській свідомості пов'язаний із кількістю пігменту, фарби.

Вказані три атрибути дозволяють описати всі кольори. Те, що атрибутів три, вказує на тривимірність кольору.

Необхідно ще уточнити, що ми розуміємо під тоном кольору. Аналіз спектру, зображеного на рис. 3.5, а, дозволяє стверджувати, що випромінювання має світло-зелений колір, оскільки чітко виділяється одна спектральна лінія на фоні рівномірного спектру білого кольору.

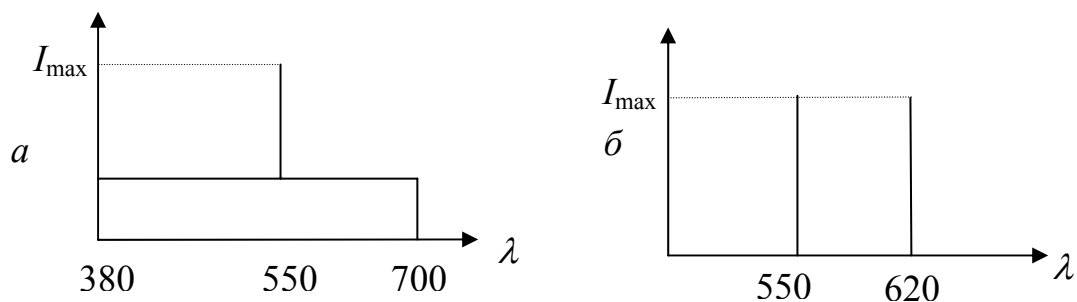


Рис. 3.5. Два спектри: *a* – явна перевага однієї складової, *б* – дві складові однакової інтенсивності

А який тон кольору відповідає спектру на рис. 3.5, *б*? Тут неможливо виділити в спектрі переважаючу складову, оскільки присутні червона та зелена лінії спектру мають однакову інтенсивність. За законами змішування кольорів це дає відтінок жовтого кольору, хоча в спектрі немає відповідної лінії монохроматичного жовтого. Тому під тоном кольору розуміють колір монохроматичного випромінювання, що відповідає сумарному кольору суміші (усередненій довжині хвилі).

3.2. Моделі кольорів

Колір може бути отриманий у процесі випромінювання та в процесі відбивання, тому існують два протилежних методи опису кольору: система адитивних кольорів і система субтрактивних кольорів. Для математичного опису кольору використовується поняття моделі кольору.

Кольори в природі рідко бувають простими. Більшість кольорових відтінків утворюється змішуванням основних кольорів. Спосіб розкладання кольору на складові компоненти називається *моделлю кольору*.

Існує багато різних моделей кольору, але в комп'ютерній графіці, як правило, застосовуються в основному три. Ці моделі відомі під назвою RGB, CMYK і HSB. Вони відрізняються базовими компонентами.

3.2.1. Адитивна модель кольору RGB

Адитивна модель кольору найпростіша для розуміння. Вона є досить штучним прийомом, оскільки продиктована технологією виготовлення електронно-променевих трубок. Її використовують для опису кольорів, отриманих за допомогою пристроїв, робота яких базується на принципах випромінювання світла, тобто для пристроїв, що світяться (монітори, побутові телевізори). Це адитивна модель кольору, оскільки для отримання потрібного кольору базові кольори в ній додаються (змішуються). Основними кольорами вибрано червоний (Red), зелений (Green) і синій (Blue), бо сприйняття кольору людиною побудовано саме на цих кольорах. Інші кольори отримуються шляхом змішування певної кількості вказаних основних кольорів, тобто

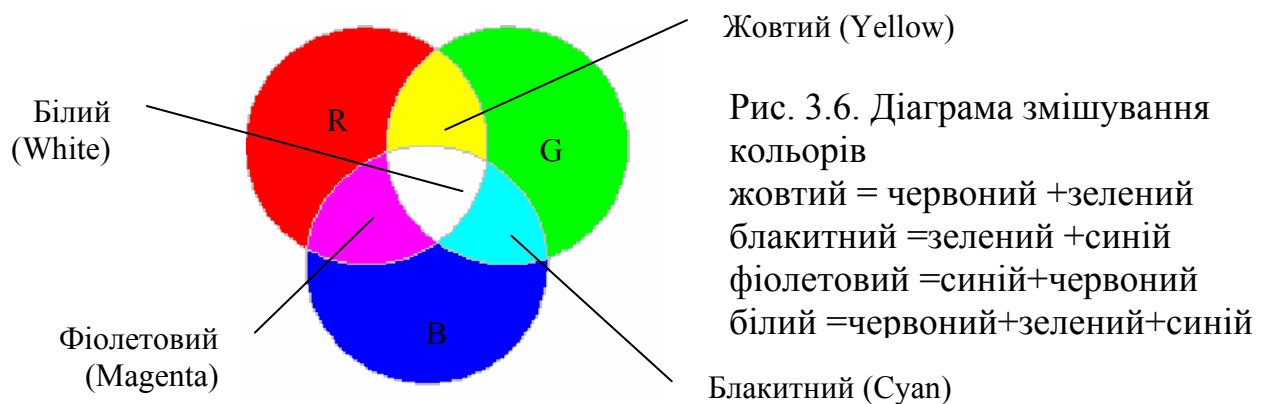
$$C = rR + gG + bB,$$

де r , g , b – відповідні кількості основних кольорів. Комп'ютер може точно управляти кількістю світла. Комбінуючи різні значення точок R, G, B, можемо одержати будь-який колір.

На сьогодні система RGB є офіційним стандартом. Рішенням Міжнародної Комісії з освітлення (МКО) в 1931 р. були стандартизовані основні кольори. Комісія рекомендувала використовувати як R, G, B такі монохроматичні кольори – випромінювання хвиль довжиною для R – 700 нм, для G – 546,1 нм, для B – 435,8 нм.

Червоний колір отримують за допомогою лампи розжарювання або криптонового лазера. Для одержання чистих зеленого і синього кольорів використовують ртутну лампу або аргонний лазер.

Діаграма змішування кольорів зображена на рис 3.6.



Колір, що створюється змішуванням трьох компонент, можна зобразити як вектор у тривимірній системі координат RGB. Точка $(0, 0, 0)$ – центр системи координат, відповідає чорному кольору (відсутність свічення екрану). Білий колір виражається максимальним значенням всіх трьох компонент. Нехай це максимальне значення вздовж кожної осі дорівнює 255, що відповідає найбільшій яскравості світла. Тоді білий колір – це вектор $(255; 255; 255)$.

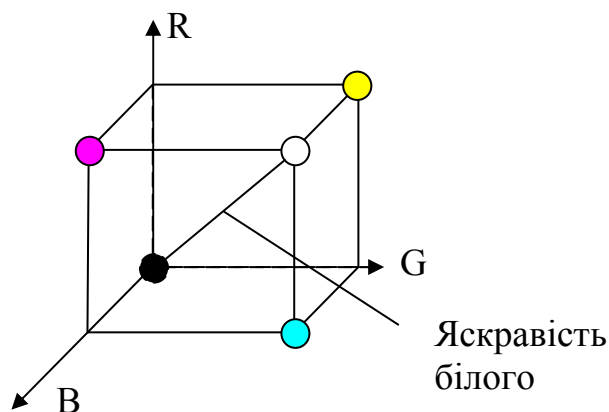


Рис. 3.7. Колірний куб у моделі RGB

Отже, наші очі й мозок, сприймаючи біле світло на екрані, говорять нам неправду, бо це комбінація червоних, зелених і синіх точок. Змішування кольорів R, G, B у різних пропорціях дає новий колір, тобто простором кольорів є колірний куб (рис. 3.7). Точки з рівними значеннями R, G, B ($R = G = B$ означає однаковий

внесок трьох базових кольорів) лежать на головній діагоналі колірнього кубу і являють собою різні градації сірого кольору (їх можна вважати білим кольором різної яскравості).

Ця модель використовується завжди при підготовці екранного зображення. Якщо зображення проходить обробку в графічному редакторі, то його теж подають у цій моделі.

3.2.2. Субтрактивна модель кольорів СМУ/СМУК

Субтрактивна модель використовується для підготовки не екранних, а друкованих зображень, тобто для пристроїв, які реалізують принцип поглинання (віднімання) кольорів. Друковані зображення відрізняються від екранних зображень тим, що їх бачать не у світлі, що проходить, а у відбитому світлі, оскільки аркуш паперу не випромінює світло. Замалований папір деякі електромагнітні хвилі з оптичного діапазону поглинає, а решту відбиває, а наше око сприймає лише відбиті хвилі. Тому для підготовки друкованих зображень використовується не адитивна модель RGB, а субтрактивна модель СМУ. На відміну від моделі RGB, біла точка в СМУ – це відсутність фарб на папері.

Назва цієї моделі складається з назв субтрактивних кольорів (протилежних до R, G, B) – блакитного (Cyan), пурпурного (Magenta) і жовтого (Yellow) (рис. 3.8), тобто, щоб отримати потрібний колір, базові кольори віднімаються від білого. Ці три кольори називаються доповнювальними, оскільки вони доповнюють основні кольори до білого, тобто змішування даного кольору і доповнювального до нього дає білий колір. Ці співвідношення можна подати у вигляді

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

тобто *доповнювальний колір = білий колір – даний колір*.

Обернене перетворення здійснюється за формулою

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

Наприклад, коли на поверхню паперу нанести блакитний (cyan) колір, тоді червоне світло, що падає на папір, повністю поглинатиметься. Отже, блакитна фарба, так би мовити, віднімає червоний колір від білого, який є сумою червоного, зеленого і синього кольорів, тобто відбивається лише зелена та синя складові світла, що і дає блакитний колір (рис. 3.9).

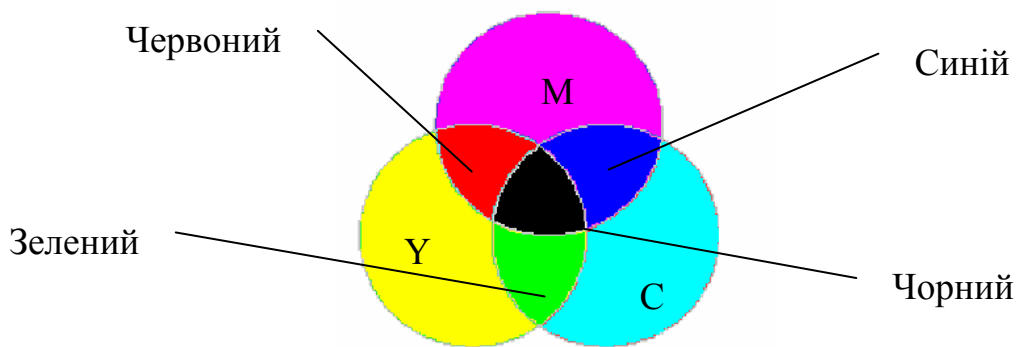


Рис. 3.8. Основні кольори системи СМУ

Аналогічно жовта фарба (Yellow) поглинає синій колір, а пурпурна (Magenta) – зелений. Білий папір виглядає білим тому, що він відбиває всі кольори і жоден не поглинає (рис. 3.9).

При освітленні білим світлом, наприклад, синьої поверхні в шарі синьої фарби зі спектру білого кольору поглинаються червона та зелена частини спектру і в результаті ми бачимо синій колір (рис. 3.10) Чорний колір відповідає поглинанню всіх кольорів при відображенні (рис. 3.10). Якщо освітити червоний папір синім або зеленим світлом, то папір буде виглядати чорним, оскільки червоний папір не відбиває синій та зелений кольори, а поглинає їх (рис. 3.9).

Істотною проблемою в поліграфії є чорний колір. Теоретично його можна отримати змішуванням трьох доповнювальних фарб, але на практиці змішування цих трьох кольорів дає невизначений темно-коричневий колір. Отримати на папері чорний колір шляхом змішування трьох фарб складно і незручно через те, що реальні фарби не є абсолютно чистими, через великі витрати дорогого чорнила та високу вологість паперу на струменевих принтерах, через небажані візуальні ефекти, тому в принтерах до базових фарб СМУ доводиться додавати ще й фарбу чорного кольору (black). Така модель кольору називається СМУК.

При друці малюнка на кольоровому принтері з чотирма кольорами драйвер принтера перетворює RGB-малюнок у модель СМУК. Однак багато відтінків, створених в кольоровій системі RGB, не вдається передати при друці на принтері. А це означає, що колірне охоплення системи СМУК менше, ніж колірне охоплення системи RGB. Водночас варто зазначити, що лише частину кольорів, які зустрічаються в природі і сприймаються людським зором, можна відтворити на екрані монітора, тобто колірне охоплення моделі RGB вужче, ніж колірне охоплення людського ока. Як видно, жодна з моделей не є повною за колірним охопленням. Під *колірним охопленням* розуміють діапазон кольорів, який може бути відтворений моделлю кольорів.

У типографіях кольорові зображення друкують у кілька етапів,

накладаючи по чергово на папір блакитний, пурпурний, жовтий і чорний відбитки. Так отримують повноколірну ілюстрацію. Тому готове зображення, отримане на комп'ютері перед друком, ділять на чотири складових одноколірових зображення. Цей процес називається діленням кольору (separations). Сучасні графічні редактори мають спеціальні засоби для виконання цієї операції. Ці програми самі визначають суміш СМУК.

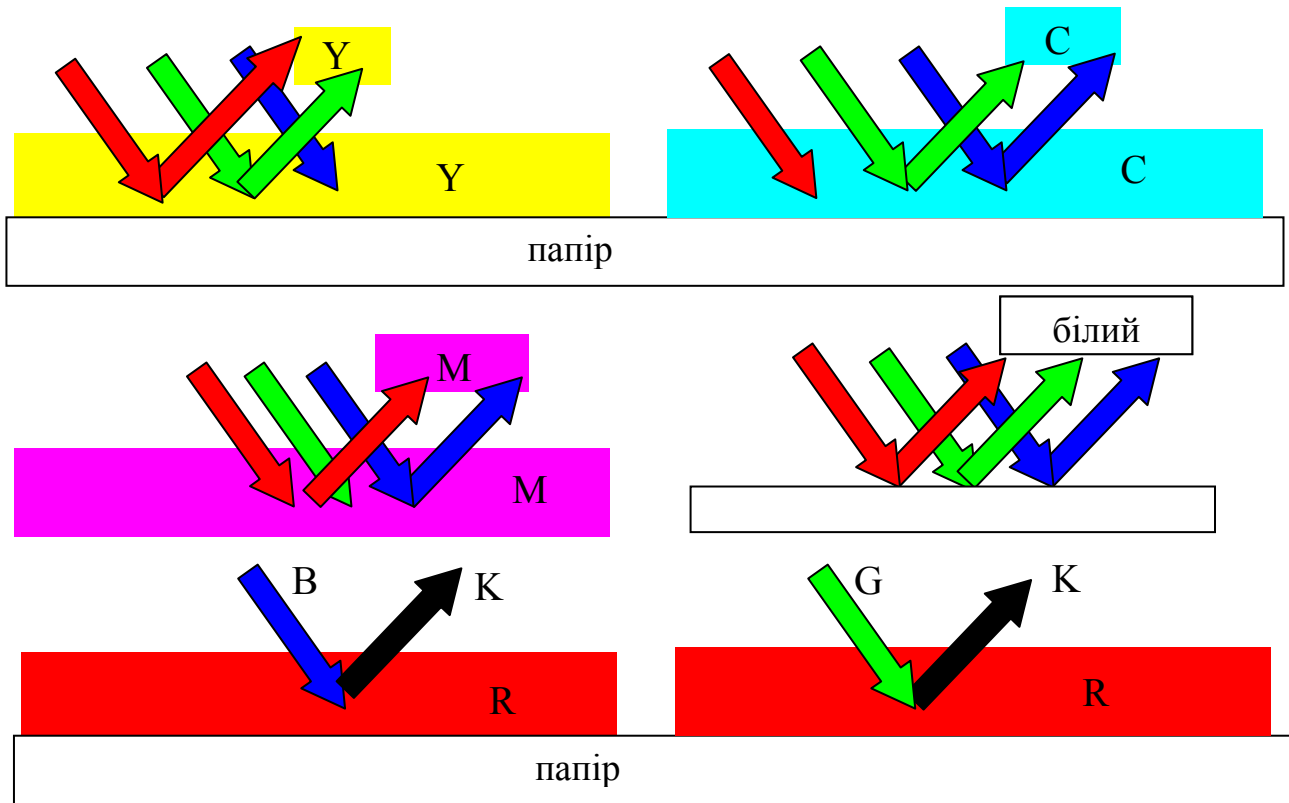


Рис. 3.9. Поглинання (віднімання) кольорів

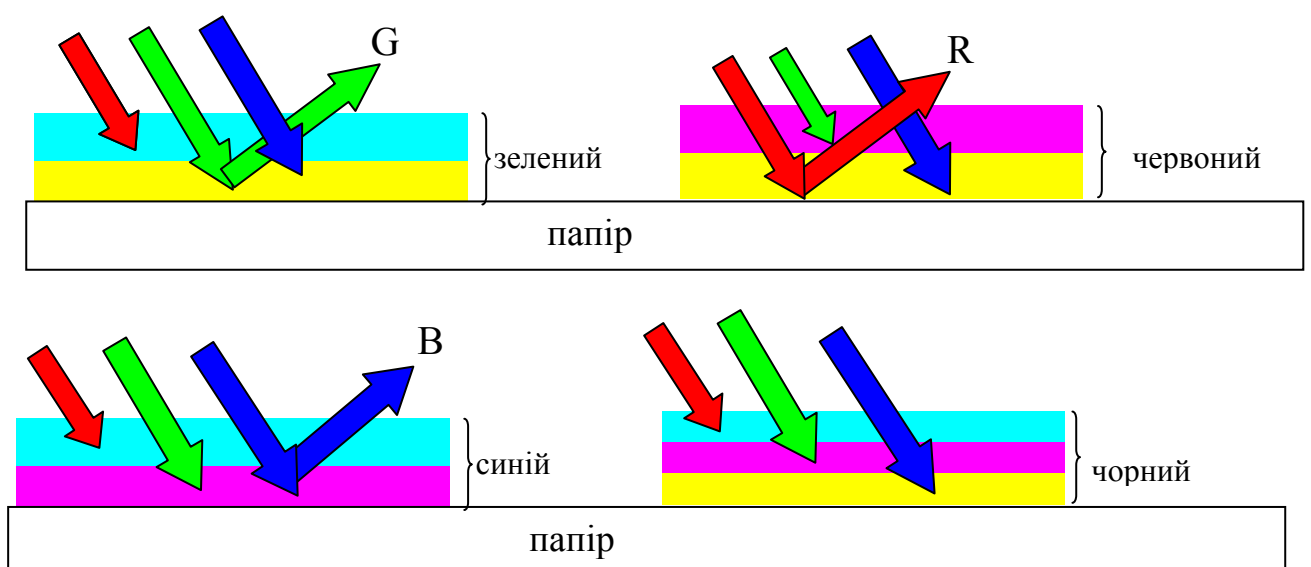


Рис. 3.10. Субтрактивність для двох і трьох кольорів

У лазерному принтері є свій “комп’ютер” із необхідними програмами, що виконують перетворення графічних даних у зображення на папері.

Кольорові принтери розділяють кольорові малюнки на 4 складові, а потім окремо друкують кожну частину на одному й тому ж аркуші паперу, тобто один аркуш проходить через принтер 4 рази. При цьому окремі кольорові точки, з яких складається малюнок, повинні бути трохи зсувеними одна від одної, щоб не було накладання кольорів.

Для переходу від моделі CMY до моделі CMYK використовують такі співвідношення:

$$K = \min(C, M, Y), \quad C = C - K, \quad M = M - K, \quad Y = Y - K.$$

3.2.3. Суб’єктивна модель кольорів HSB (HSV)

Моделі RGB, CMY, CMYK орієнтовані на роботу з технічними засобами. Якщо модель RGB найприйнятніша для комп’ютера, модель CMYK – для типографій, то модель HSB найзручніша для людини. Вона проста та інтуїтивно зрозуміла.

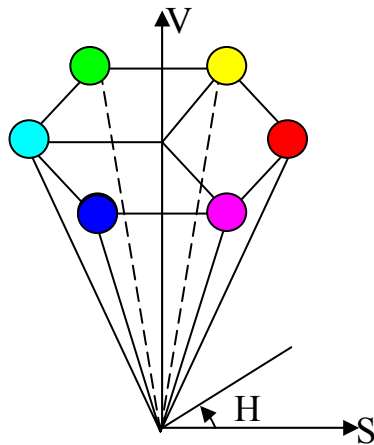


Рис. 3.11. Модель HSB

Ця модель дозволяє задавати кольори, опираючись на інтуїтивні поняття тону кольору H (Hue), насиченості кольору S (Saturation) та яскравості кольору V/V (Brightness/Value). Тому цю модель називають HSB або HSV. Регулюючи три згадані компоненти, можна отримати стільки ж кольорів, як і при роботі з іншими моделями.

Деякі графічні редактори дозволяють працювати з моделлю кольорів HSB. Модель HSB зручна для застосування в тих графічних редакторах, які зорієнтовані не на обробку готових зображень, а для створення власних художніх творів.

У цій моделі використовується циліндрична система координат, а множина всіх допустимих кольорів є конусом, покладеним на вершину (рис. 3.11). Основа конуса – яскраві кольори, що відповідають значенню $V=1$ (конус має одиничну висоту).

Значення H змінюється в градусах від 0^0 до 360^0 , оскільки кольори веселки розміщуються на колі в такому порядку: червоний, оранжевий, жовтий, зелений, блакитний, синій, фіолетовий. Червоному кольору відповідає кут 0^0 , зеленому – 120^0 і т.д. Кольори, що доповнюють один одного до білого, на колі знаходяться навпроти. Величина S змінюється від 0 до 1. Значення $V = 0$ відповідає чорному кольору. При $S = 0$ (тобто на осі V) маємо сірі відтінки. Білий колір кодується як $S = 0, V = 1$. При $S = 0$ значення H не має змісту.

Модель HSB зручна для вибору кольорів на екрані. Спочатку на екрані можна зобразити спрощену палітру, а потім збільшити або зменшити яскравість. Наприклад, так діють при моделюванні затінення об'єктів або сутінків.

Графічні редактори дозволяють працювати з кольоровими зображеннями в різних моделях. Наприклад, у Paint Brush for Windows для установки кольору використовуються дві моделі – RGB та HSV (існують відповідні формули зв'язку між параметрами R, G, B та H, S, V).

3.3. Баланс кольорів

Довільна зміна складової кольору впливає на загальний баланс кольорів, тобто зміна однієї компоненти обов'язково відбивається на інших кольорах. Тому в основі будь-якої корекції кольорів лежить налаштування не окремих кольорів, а балансу кольорів, причому одного й того ж результату можна досягти різними способами.

Щоб легше уявити взаємодію компонент кольору, необхідно розглянути спрощену схему колірної кола (рис. 3.12), яка дає можливість наочно продемонструвати взаємодію компонент балансу кольорів.

На схемі кожний колір знаходиться між двома кольорами, за допомогою яких він одержується. Наприклад, додавання зеленого і червоного кольорів дає жовтий. Щоб підсилити будь-який колір, потрібно послабити доповнювальний колір (він розміщений напроти на колірному колі). Наприклад, щоб змінити колір у бік блакитного тону, необхідно знизити в ньому вміст червоного кольору. Щоб зменшити фіолетову складову, можна зменшити її безпосередньо, але краще це зробити шляхом збільшення блакитної і жовтої складових, що дозволить зберегти насиченість зображення. Отже, щоб вплинути на фіолетовий колір, можна задіяти всі кольори кола.

Таким чином, при довільних впливах на компоненти кольору, необхідно врахувати, що це відбивається на всьому просторі кольорів. У зв'язку з цим необхідно зважати на такі закономірності:

- 1) кольори, що лежать на колі навпроти один одного, взаємно пов'язані (зменшення вмісту одного кольору збільшує вміст протилежного кольору);
- 2) вміст певного кольору можна змінити за рахунок впливу на сусідні кольори;
- 3) щоб збільшити вміст певного кольору, можна зменшити вміст кольорів, які сусідні із протилежним, і навпаки.

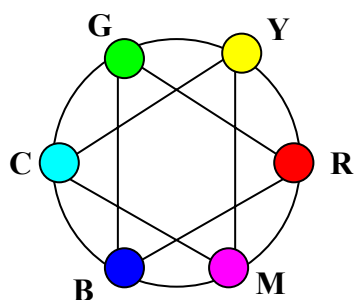


Рис. 3.12. Спрощена схема колірної кола

3.4. Кодування кольору. Палітра кольорів

Щоб комп'ютер мав можливість працювати з кольоровими зображеннями, необхідно вміти подавати кольори у вигляді чисел, тобто кодувати колір за допомогою комбінації бітів. Кількість бітів для подання кольору кожного пікселя, називається *бітовою глибиною*. Спосіб кодування кольору залежить від моделі кольорів та формату числових даних у комп'ютері.

Для моделі RGB кожен з компонент можна зобразити з допомогою чисел, обмежених певним діапазоном (наприклад, дробовими числами від 0 до 1 або цілими від 0 до деякого максимального значення). Зараз досить розповсюджений формат True Color (істинний колір), в якому під кожний піксель відводиться 24 біти, тобто кожна компонента подається у вигляді байта, що дає 256 градацій для кожної компоненти: $R = 0 - 255$, $G = 0 - 255$, $B = 0 - 255$. Це дає можливість закодувати $256 \times 256 \times 256 = 2^{24} = 16777216$ градацій кольорів, що значно перевищує кількість кольорів, які розрізняє людське око. Такий спосіб кодування кольорів називається *компонентним*. У комп'ютері коди зображень True Color подаються трьома байтами (рис. 3.13) або упаковуються в довге ціле – 32 біти (так, наприклад, зроблено в API Windows).

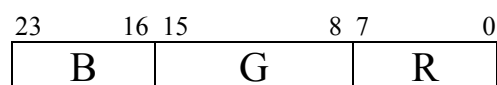


Рис. 3.13. 3-байтовий код

У 32-бітній моделі RGBA старші 8 бітів використовуються для задання компоненти маски (А-компоненти, або Alpha-каналу). Маска створюється деякими графічними програмами для спеціальних ефектів, наприклад для задання прозорості, туману.

Хоча немає видимих причин використовувати більше кольорів, реально використовуються 48-бітні і навіть 64-бітні моделі. Ці моделі використовуються в кольорових системах вищого рівня, зокрема в професійних графічних системах.

При роботі із зображеннями в системах комп'ютерної графіки часто доводиться шукати компроміс між якістю зображення (якомога більше кольорів) і ресурсами, необхідними для збереження зображення.

Якщо для кодування кольору виділити 1 біт, то можна закодувати 2 різних кольори (чорно-білий режим). Виділення одного байта дозволяє закодувати 256 різних відтінків кольорів. Два байти дозволяють визначити 65536 кольорів (рис. 3.14). Цей режим називається High Color.

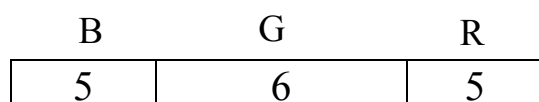


Рис. 3.14. 2-байтовий код

У випадку True Color/High Color кольорова палітра не потрібна, оскільки в трьох/двох байтах достатньо інформації про колір пікселя.

При обмеженні кількості кольорів, тобто для зменшення обсягу пам'яті використовують палітру. *Кольорова палітра* – це набір кольорів. *Палітру* можна сприймати як таблицю кольорів, у якій указано індекс (номер) кольору та сам код того чи іншого кольору. Кожен кольоровий відтінок задають одним числом, причому це число визначає не код кольору, а номер кольору в таблиці. Сам колір визначається за цим номером у таблиці-палітрі. Ця таблиця зберігається разом із графічним файлом. Програма, що здійснює візуалізацію даних, читає з файла індекси і використовує відповідні їм кольори для зображення пікселів на екрані.

Різні зображення можуть мати різні кольорові палітри. Якщо відтворити зображення з іншою палітрою, то зелена ялинка може стати рожевою (в різних палітрах під однаковим номером можуть зберігатися різні кольори). У зв'язку з цим виникає проблема відповідності кольорів при перегляді Web-графіки. Для оформлення Web-сторінок не застосовують графіку, що має кодування кольору вище 8 бітів через невисоку швидкість передачі даних в Internet.

Зважаючи на це, було прийнято рішення – всі браузері (програми перегляду Web-сторінок) завчасно налаштовувати на певну фіксовану Web-палітру. У цій палітрі не 256 кольорів, які дозволяє кодувати 8 бітів, а лише 216. Це пов'язано з тим, що в Інтернеті працюють з різними комп'ютерами і не всі комп'ютери можуть відтворити 256 кольорів. Якщо розробник Web-сторінки використовуватиме лише цю палітру при створенні ілюстрації, то користувачі побачать малюнок правильно. Найбільш часто використовуються палітри з 16 та 256 кольорів. Як приклад наведемо стандартну палітру 16-кольорових відеорежимів VGA (табл. 3.1).

При 8-бітній глибині кольору можна задати 256 кольорів. Очевидно, що тут можна виділити під кожному компоненту певну кількість бітів (наприклад, для R – 3, для G – 3, для B – 2), але така технологія надто обмежує можливості передачі кольорів. Тому для такої малої глибини кольору краще використовувати інший підхід – із множини $256 \times 256 \times 256$ кольорів вибираються довільні 256 кольорів, які нумеруються індексами від 0 до 255. Вибрані кольори записуються в таблицю кольорів (палітру), а у відеопам'яті замість коду кольорів записується 8-бітний індекс (номер кольору), який при візуалізації автоматично замінюється кольором, який у палітрі відповідає цьому індексу.

У графічних системах, наприклад OpenGL, для роботи з палітрами кольорів є спеціальні оператори. З їх допомогою можна створити палітру, вказавши для кожного індексу набір R, G, B компонент кольору і в будь-який момент можна вибрати довільний індексний колір із палітри.

Таблиця 3.1

Номер кольору	R	G	B	Назва кольору	Колір
0	0	0	0	Чорний	
1	128	0	0	Темно-червоний	
2	0	128	0	Зелений	
3	128	128	0	Коричн.-зелений	
4	0	0	128	Темно-синій	
5	128	0	128	Темно-пурпурний	
6	0	128	128	Синьо-зелений	
7	128	128	128	Сірий 50%	
8	192	192	192	Сірий 25%	
9	255	0	0	Червоний	
10	0	255	0	Яскраво-зелений	
11	255	255	0	Жовтий	
12	0	0	255	Синій	
13	255	0	255	Фіолетовий	
14	0	255	255	Блакитний	
15	255	255	255	Білий	

У реальних додатках, що працюють із палітрами кольорів, часто виникає задача створення такої палітри, яка б рівномірно охоплювала весь спектр моделі кольорів RGB.

Задача. З усіх $(256)^3$ кольорів рівномірно вибрати 256 кольорів і призначити їм відповідні індекси.

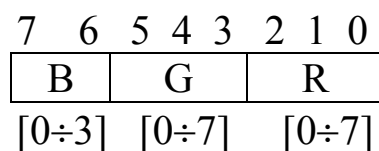


Рис. 3.15. Підіндекси компонент R,G,B

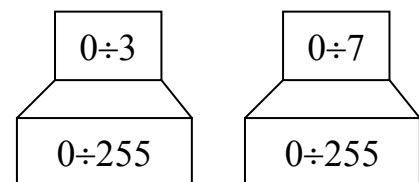


Рис. 3.16. Відображення інтервалів

Розв'язування. Розіб'ємо поле індексу справа наліво на 3 підполя довжиною 3, 3 і 2 біти, відповідно (рис. 1.15). Ці поля визначатимуть підіндекси компонент R, G, B. Для компоненти B відведемо 2 біти, оскільки зміна синього кольору менше сприймається людським оком, ніж зміна кольорів R та G (колір B має найменший діапазон електромагнітних хвиль).

Таким чином, індекс кольору в палітрі I_c можна подати у вигляді

$$I_c = I_B \cdot 2^6 + I_G \cdot 2^3 + I_R, \quad (3.1)$$

де I_R, I_G, I_B – значення підіндексів.

Для вибору кольору потрібно рівномірно відобразити інтервали підіндексів $[0, 3]$, $[0, 7]$, $[0, 7]$ на інтервал $[0, 255]$ (рис. 3.16). Для компонент R і G одержуємо відповідно значення кольору $[255 \times I_R/7]$ і $[255 \times I_G/7]$, а для компоненти B – значення $[255 \times I_B/3]$. Замість цілої частини можна брати найближче ціле.

Обчислимо, який колір буде відповідати в палітрі індексу 214. Число 214 розкладемо за формулою (3.1). Маємо $214 = 3 \cdot 2^6 + 2 \cdot 2^3 + 6$, тому $I_R = 6$, $I_G = 2$, $I_B = 3$. Тоді для компонентів кольору матимемо: R = $[255 \times 6/7] = [218,57] = 218$, G = $[255 \times 2/7] = [72,86] = 72$, B = $[255 \times 3/3] = 255$. Отже, індексу 214 в рівномірній палітрі з 256 кольорів відповідає колір (218, 72, 255).

3.5. Оптимальне поєднання кольорів при побудові зображень

Крім фізіологічного впливу кольору на нервову систему, колір характеризується психологічними та терапевтичними діями.

Розглянемо деякі аспекти психологічної дії кольору на людину. Досвід показує, що різні кольори по-різному діють на психіку людини.

Червоний і оранжевий кольори збуджують, стимулюють діяльність, сприяють короткочасному підвищенню продуктивності праці. Світло-сині тони створюють легкий настрій, темно-сині справляють холодне враження. Зелений колір – заспокійливий, приємно діє на очі, білий викликає відчуття холоду, фіолетовий колір – неспокійний (у Китаї – колір трауру), викликає негативні реакції, коричневий – м'який і затишний.

Наведені характеристики дії кольорів на психіку людини є основою для поділу кольорів на холодні та теплі. Холодні кольори (зелений, синій) заспокоюють, полегшують напруження очей, теплі (червоний, жовтий, оранжевий) потрібно розглядати як активні, динамічні. Холодні кольори створюють відчуття збільшеного простору (предмети ніби віддаляються), теплі – навпаки, зменшують зоровий простір. Предмети, зафарбовані теплими кольорами, здаються ближчими, а зафарбовані у світлий тон – більш легкими, і навпаки.

Зафарбовані об'єкти мають різний ступінь сприйняття кольорів. У різних експериментах отриманий такий порядок розрізнення кольорів (у порядку спадання):

- синій на білому;
- білий на темно-синьому;
- лимонно-жовтий на малиновому;
- чорний на білому;
- темно-синій на оранжевому (жовтогарячому);
- білий на червоному.

Найгірше розрізняють зелений колір на червоному фоні та червоний на зеленому.

Отже, перша вимога при роботі з кольором – це чіткість і зручність сприйняття зображення, що забезпечується оптимальним підбором його основних характеристик. Хоча треба зазначити, що сприйняття кольорів людиною є індивідуальним фактором.

При сучасних технічних характеристиках графічних кольорових моніторів можна істотно підвищити комфортність роботи з комп'ютером за допомогою раціонального поєднання кольорів. Сформулюємо найважливіші вимоги до роботи з кольором.

1. Не використовувати надто яскравих і насичених кольорів на великих поверхнях.
2. Не варто зафарбовувати великі поверхні яскраво-червоним кольором.
3. Уникати темних кольорів: фіолетового, коричневого, темно-сірого (часто трапляються в нашому буденному житті). Для зображення контрастних кольорів, краще вибрати світло-сірий фон.
4. Для предметів, що повинні найбільше привертати увагу, необхідно використовувати червоні, сині, зелені, жовті або білі кольори; яскраво-синій колір не годиться для зафарбовування малих графічних елементів.
5. Не слід використовувати кольори, які різко відрізняються за яскравістю, краще використати контраст тону кольорів.
6. Кольори, що використовуються, повинні викликати позитивні емоції, поліпшувати самопочуття, підвищувати працездатність. Наприклад, емоції комфорту стимулюються слабо насиченими синьо-зеленими кольорами. Темно-зелений, темно-фіолетовий кольори викликають негативні реакції.
7. Якщо яскравість об'єкта наближена до яскравості фону, то об'єкт важко розрізняється. Для фону необхідно вибирати кольори, які привертають до себе мінімум уваги. Наприклад, білі літери на синьому фоні найкраще сприймаються оком.
8. Ефект площини досягається за рахунок підбору кольорів, близьких за яскравістю та насиченістю.
9. Для динамічних зображень варто надавати перевагу основним кольорам: червоному, зеленому, блакитному. Для кодування статичних зображень бажано використовувати змішані кольори.

Використовуючи ці рекомендації у практичній діяльності, можна уникнути помилок при підборі кольорів для побудови зображень.

Контрольні питання та завдання

1. Що таке колір? Як він утворюється?
2. Дайте визначення спектральної кривої.
3. Сформулюйте основні закони колориметрії.
4. В чому полягає суть трикомпонентної теорії кольору?
5. На які основні кольори розкладається біле світло?
6. Що таке колірна модель? Які ви знаєте колірні моделі? Яке призначення кожної колірної моделі?
7. Які кольори називаються доповнювальними?
8. Що таке колірне охоплення? Порівняйте цю властивість для різних моделей кольору.
9. Які атрибути використовує людина для оцінки кольору?
10. Що таке тон кольору? У чому полягає тонова корекція зображення?
11. Що характеризує яскравість/насиченість кольору?
12. Як можна змінити вміст певного кольору?
13. Що таке глибина кольору?
14. Яка максимальна кількість кольорів отримується при глибині кольору 16/24 біти?
15. Що таке палітра кольорів? Для чого вона використовується? Як утворити рівномірну палітру?
16. Скільки різних градацій сірого кольору має режим True Color?
17. Скільки різних градацій зеленого, червоного, синього може бути одержано в режимі High Color?
18. Які правила оптимального поєднання кольорів ви знаєте?

Вправи і задачі для самостійного виконання

1. Визначити, який колір побачить людина при освітленні червоного паперу блакитним світлом.
2. Побудуйте спектральну криву для світло-червоного світла.
3. Записати формули для переходу від моделі RGB до моделі CMY в форматі True Color.
4. Записати координати HSB для світло-зеленого кольору.
5. Вивести формули для перетворення параметрів кольору RGB в HSV і навпаки.
6. Написати програму вибору кольорів, використовуючи три повзунки для задання HSV-компонент кольору.
7. Обчислити колір, який відповідатиме індексу 168 у рівномірній палітрі з 256 кольорів.
8. Нехай в повнокольоровій системі (24 біти на піксель) є буфер кадра 512×512 . Скільки кольорів можна відобразити одночасно на екрані?

Тема 4. Відеоадаптер. Програмування відеоадаптера

4.1. Основні поняття

Відеосистема комп'ютера призначена для формування графічних зображень на екрані монітора. Основна функція відеосистеми – вибір даних із відеобуфера і перетворення їх у сигнали, що створюють зображення на екрані. Центральний процесор не займається управлінням дисплея, тобто не виконує попиксельне прораховування 3D-сцени, а лише посилає спеціальному пристрою – відеоадаптеру просторове положення об'єктів, які необхідно відобразити. Ці об'єкти складаються з величезної кількості многокутників. Каркас многокутників необхідно покрити поверхнею, щоб одержати реалістичне зображення на екрані комп'ютера. Цю роботу і бере на себе відеокарта.

Отже, *відеоадаптер (відеокарта)* – це пристрій (плата), що керує роботою монітора (тобто безпосередньо передає сигнали керування блокам монітора для виведення інформації на екран монітора) та взаємодією з центральним процесором комп'ютера.

Екран дисплея – це прямокутна матриця окремих дискретних точок (пікселів), які формують зображення. Кількість пікселів визначає *роздільну здатність дисплея* і задається парою чисел. Перше число показує кількість пікселів у рядку, а друге – кількість рядків (наприклад, 640×480 , 1024×768 , 1280×1024). Чим більші ці числа, тим меншими є самі пікселі. Це приводить до того, що вони сприймаються не як окремі точки, а як єдине цільне зображення.

Кожному пікселю в деякій частині загального адресного простору, що називається *відеопам'яттю*, або *буфером кадра*, ставиться у відповідність фіксоване число бітів – *атрибут пікселя* (код кольору). Ці біти і визначають колір пікселя з заданого набору кольорів.

Кількість бітів, що відводиться для опису кольору однієї точки растра, називається *роздільною здатністю бітової глибини*, *кольоровою роздільною здатністю* або просто *глибиною кольору* чи *глибиною буфера кадра*. Глибина кольору може набувати значення 1, 2, 4, 8, 16, 24 і навіть 32 біти. Якщо глибина кольору дорівнює n бітів, то графічні зображення на екрані можуть мати на екрані 2^n відтінків.

В дисплейних адаптерах з монохромним монітором значення атрибута керує інтенсивністю одного променя, тобто яскравістю точки, а з кольоровим монітором – інтенсивністю трьох променів, що задають три компоненти кольору пікселя. Кожний кольоровий піксель утворюється трьома меншими ділянками червоного, зеленого і синього кольору, які при змішуванні визначають результуючі кольори пікселів.

Відеопам'ять є складовою частиною відеоадаптера. Фізично відеопам'ять організована у вигляді одновимірного вектора в загальному адресному просторі. Відеопам'ять реалізована у вигляді спеціальних типів мікросхем із довільним доступом (RAM), які дозволяють швидко вивести вміст буфера на екран. Як правило, адреса першого байта відеопам'яті має значення A000:0000. В усіх графічних режимах стартова адреса відеопам'яті відповідає лівому верхньому пікселю на екрані. Відеоадаптер циклічно з частотою кадрів монітора (75–120 разів за секунду) зчитує вміст відеопам'яті й постійно формує зображення на екрані монітора, при цьому колір кожного пікселя визначається поточним значенням його атрибута. Зображення на моніторі повністю відповідає поточному вмісту відеопам'яті. Ємність відеопам'яті сучасних відеокарт досягає 1024 Мб. Така ємність потрібна для зберігання й оброблення великої кількості даних (буфер кадра, буфер глибини, параметри вершин, текстури тощо), що використовуються графічним процесором для побудови тривимірних рухомих зображень на екрані монітора. Ємність графічної пам'яті визначає складність зображень, параметри якості та швидкості відображення.

Програма, що виконується на комп'ютері в графічному режимі, має доступ (читання/запис) до всіх комірок відеопам'яті. Для збереження декількох кадрів зображення в певних режимах у відеопам'яті передбачені окремі області однакової структури, кожна з яких містить атрибути пікселів екрану. Ці області називаються *відеосторінками*. Їх кількість є степенем двійки. У даний момент тільки одна сторінка зображена на екрані (програма в цей момент може працювати з неактивною сторінкою). Наявність сторінок дає можливість програмі миттєво змінювати зображення на екрані і в такий спосіб усувати ефект миготіння. Використання відеосторінок відіграє важливу роль при мультиплікації. Зміна відеосторінок створює ефект руху зображень на екрані.

Для зміни поточної сторінки можна викликати відповідну функцію BIOS, або безпосередньо змінити вміст реєстра початкової адреси, при цьому процесор записує інформацію безпосередньо у відеопам'ять. Графічна бібліотека дозволяє здійснювати роботу з будь-якою сторінкою. Активна сторінка встановлюється процедурою `setactivepage`.

Відеоадаптер включає в себе, крім відеопам'яті, в якій зберігається зображення на екрані монітора, ще й постійно запам'ятовуючий пристрій, в якому записані набори шрифтів для текстових та графічних режимів відеоадаптера і функції BIOS для роботи з відеоадаптером. Окрім цього, відеоадаптер містить складні керуючі пристрої, що забезпечують обмін даними та формування зображення.

У сучасних відеоадаптерах більша частина графічних функцій реалізована безпосередньо на апаратному рівні, що вимагає використання потужного спеціалізованого графічного процесора, який за складністю наближається до центрального процесора, причому, можливо, не одного, а декількох, кожен з яких виконує свій набір графічних функцій. Наприклад, графічний процесор ATI Radeon X1800 XT містить 320 млн. транзисторів і декілька десятків спеціалізованих процесорів.

Перші графічні процесори відеоадаптера використовувалися для виконання операцій виведення ліній, полігонів (плоских елементів). Сучасні графічні процесори (GPU, G80) виконують багато базових операцій 3D-графіки, наприклад підтримку Z-буфера, накладання текстур, формування ефектів туману, відбитого світла, прозорості тощо. Оскільки відеоадаптери виконують ці операції апаратно, що дозволяє пришвидшити їх у сотні, тисячі та мільйони разів (залежно від складності алгоритму) в порівнянні з програмною реалізацією, то нове покоління відеоадаптерів для роботи з тривимірною графікою називають ще *графічними прискорювачами* або *графічними акселераторами*.

Відеоадаптери можуть відрізнятися не тільки за швидкодією, можливостями роботи з кольором, але й за рівнем реалізації тих чи інших графічних операцій. Наприклад, відеоадаптер Matrox дозволяє створювати якісну двовимірну графіку, Nvidia GeForce – потужний ігровий 3D-акселератор, 3DLabs Wildcat використовується для професійного 3D-моделювання, для створення віртуальної реальності.

Відеоадаптер може бути оформлений у вигляді окремої плати, яка вставляється в слот розширення комп'ютера, або безпосередньо розміщуватися на системній платі.

Для виведення графічних зображень, особливо в режимі анімації, потрібна висока швидкість передачі даних для того, щоб досягти високої частоти кадрів, тому важливою рисою персонального комп'ютера з позицій графіки є те, що сучасні відеоадаптери під'єднується до системної шини за допомогою швидкісної локальної шини AGP (Accelerated Graphics Port) або до більш сучасного послідовного інтерфейсу PCI Express. Це дає можливість швидко вести обмін даними між оперативною пам'яттю та відеопам'яттю і підвищити швидкодію комп'ютера.

Розрядність шини AGP – 64 біти. На частоті 66 МГц вона забезпечує швидкість обміну 528 Мб/с. Це був стандарт AGP 2x. Зараз AGP працює на більш високих частотах. У 1999 р. з'явився режим AGP 4x (швидкість передачі даних – 1,06 Гб/с). В 2002 р. просунуті відеокарти отримали значок AGP 8x. Максимальна пропускна здатність шини PCI Express X16 становить 4000 Мб/с у двох напрямках.

Відеоадаптери можуть працювати в різних текстових і графічних режимах (відеомодах), які відрізняються роздільною здатністю, кількістю кольорів на екрані дисплея, кількістю відеосторінок тощо.

Нині існує велике різноманіття відеоадаптерів, починаючи від монохромних, які не підтримують графічні режими, і закінчуючи сучасними відеоадаптерами, які відтворюють 16,7 млн. кольорів і більше.

Можна виділити декілька типів відеоадаптерів (табл. 4.1). Як видно з табл. 4.1, перший відеоадаптер MDA не міг відображати графіки на моніторі. Крім цього, він працював із монохромним монітором. Більшість відеоадаптерів, описаних на початку табл. 4.1, сьогодні не задовольняють користувачів і не використовуються на практиці.

Із появою операційної системи Windows вимоги до відеосистеми різко зросли. Ні EGA, ні VGA не забезпечували необхідної роздільної здатності та достатньої кількості кольорів. Нині різні фірми випускають адаптери SVGA, XGA, які працюють у режимах High Color та True Color.

Таблиця 4.1

Відеоадаптер	Рік випуску	Обсяг ОЗУ (Кб)	Роздільна здатність	Кількість кольорів
MDA (Monochrome Display Adapter)	1981	4	40 × 25с.	2
CGA (Color Graphics Adapter)	1981	16	320 × 200 640 × 200	4 2
Hercules	1982	32	720 × 350	2
EGA (Enhanced Graphics Adapter)	1984	64	640 × 350	16
VGA (Video Graphics Array)	1987	256	640 × 480 320 × 200	16 256
SVGA (Super VGA)	1990	512	800 × 600 і вище	256
XGA (Extended Graphics Array)	1991	1024	1024 × 768 і вище	256

У режимі High Color відеоадаптер може одночасно відображати на екрані 32768 або 65536 різних кольорів, у режимі True Color – 16777216. Окрім перелічених стандартів, існують ще такі стандарти (адаптери): SXGA (Super XGA) – стандарт для роздільної здатності 1280 × 1024 та UXGA (Ultra XGA) – для роздільної здатності екрана 1600 × 1200 і більше із глибиною кольору 32 біти на піксель. Слід зазначити, що такі відеоадаптери вимагають і відповідних моніторів.

Здатність відеоадаптера відобразити велику кількість кольорів із високою роздільною здатністю може бути забезпечена лише відповідним обсягом відеопам'яті. Обсяг відеопам'яті – одна з найважливіших характеристик відеопам'яті. Чим більший обсяг відеопам'яті, тим більше кольорів може відобразитися на екрані і тим більшою буде максимальна роздільна здатність екрана. Мінімальний обсяг відеопам'яті (в Кб) при відповідних вимогах можна визначити з табл. 4.2.

4.2. Режими відеоадаптерів

Практично кожен відеоадаптер підтримує декілька режимів роботи (videomodes), які відрізняються один від одного роздільною здатністю і кількістю кольорів. Різні режими навіть одного адаптера мають різну організацію відеопам'яті та способи роботи з нею. Кожен режим має свій номер. У залежності від відеорежиму визначається логічна організація відеопам'яті.

Таблиця 4.2

Кількість кольорів	Роздільна здатність екрана			
	640 × 480	800 × 600	1024 × 768	1280 × 1024
16	150	235	384	640
256	300	469	768	1280
65536	600	938	1536	2560
16777216	900	1407	2304	3840

Більшість адаптерів створюється за принципом сумісності з попередніми. Так, адаптер VGA підтримує всі режими адаптера EGA і т.д.

Установкою режиму роботи керує переривання BIOS 10H, а саме функція 00H може задати будь-який стандартний режим роботи відеоадаптера. Наведемо процедуру установки режиму *v*, використовуючи процедуру *intr*:

procedure mode (v : integer);

var r : Registers;

begin

r.ax:=v; {r.ah:=0 – номер функції; r.al:= v – номер режиму}

intr(\$10, r); {звернення до переривання}

end; {mode}

Список деяких стандартних режимів відеоадаптерів наведений у табл. 4.3.

Таблиця 4.3

Номер режиму	Тип режиму	Відеоадаптер	Роздільна здатність	Кількість кольорів	Коефіцієнт масштабування	Адреса відеопам'яті
00H	текст.	CGA, EGA, VGA, SVGA	40 × 25	16	–	B000:0000H
04H	граф.	– –	320 × 200	4	1.2	B800:0000H
0DH	граф.	EGA, VGA,	320 × 200	16	1.2	B800:0000H
10H	граф.	EGA, VGA, SVGA	640 × 350	16	1.37	A000:0000H
12H	граф.	VGA, SVGA	640 × 480	16	1.0	A000:0000H
13H	граф.	VGA, SVGA	320 × 200	256	2.4	A000:0000H

Масштабний коефіцієнт визначається відношенням кількості пікселів на дюйм у горизонтальному напрямку до кількості пікселів на дюйм у вертикальному напрямку. Масштабування потрібно проводити для того, щоб зображення виглядали реальніше: інакше квадрат буде прямокутником, а коло – еліпсом.

Відеоадаптери SVGA відрізняються більш потужними графічними контролерами й організацією відеопам'яті. Всі вони підтримують стандарт VGA і можуть працювати також у режимах зі значно кращими характеристиками. Характеристики режимів відрізняються у відеоадаптерів різних фірм, оскільки різні моделі SVGA мають різні набори регістрів. А це створювало труднощі при написанні програм для SVGA.

Асоціація стандартів в області відеоелектроніки VESA (Video Electronic Association) розробила стандарт на відеорежими, що охоплював майже всі можливості сучасних відеоадаптерів, тобто створено стандарт на графічні функції BIOS, який дозволяє керувати відеоадаптерами SVGA. Цей стандарт описує розширення переривання BIOS 10H (Vesa BIOS Extension – VBE), що здійснює управління відеоадаптером. Підтримка VBE включається на заводі у BIOS відеоадаптера.

Широкі можливості для використання SVGA надає операційна система Windows. У ній використовуються спеціальні драйвери, що виконують роботу з програмування відеоадаптерів. Тому копійка робота з регістрами адаптера схована від програмістів, які пишуть програми під Windows.

Всі функції VBE доступні через функцію 4FH переривання 10H. Таких функцій у версії VESA 2.0 одинадцять із номерами від 0 до 10. Наприклад, функція 0 повертає інформацію про версію VESA та про виробника відеоплати, функція 1 повертає інформацію про конкретний

відеорежим, функція 2 встановлює відеорежим, функція 3 повертає номер поточного режиму і т.д. Перед викликом функцій VBE необхідно записати в регістр AH значення 4FH, а в регістр AL – номер функції. Якщо ця функція в VBE реалізована, то в регістр AL повертається значення 4FH, інакше – якесь інше значення. У результаті успішного виконання функції в регістр AH повертається нульове значення, якщо в регістр AH записано значення 1H, то функція завершилася з помилкою.

Наведемо текст процедури, яка встановлює VESA-режим `vmode`.

```
procedure Set_SVGA_mode (vmode : integer);
var r : Registers;
    b : integer;
begin
    r.ah:=4FH;
    r.al:=02H; {номер функції встановлення графічного режиму}
    r.bx:=vmode; {номер режиму}
    intr($10, r);
    if ((r.al = 4FH) and (r.ah = 0)) then b:=0 else b:= -1;
end; { Set_SVGA_mode }
```

У BIOS введено деякий стандартний набір розширених режимів, кожен з яких має власний номер (код) згідно зі стандартом VESA. Номер режиму є 16-бітним числом, де біти 9–15 зарезервовані і дорівнюють нулю, а біт 8 для VESA-режимів дорівнює 1.

Наведемо таблицю деяких стандартних VESA-режимів (табл. 4.4).

Таблиця 4.4

Номер режиму	Тип режиму	К-ть кольорів	Роздільна здатність
100H	граф.	256	640 × 400
101H	граф.	256	640 × 480
102H	граф.	16	800 × 600
103H	граф.	256	800 × 600
104H	граф.	16	1024 × 768
105H	граф.	256	1024 × 768
106H	граф.	16	1280 × 1024
107H	граф.	256	1280 × 1024
108H	текст.	16	80 × 60 (на 1 с.)
⋮			
10DH	граф.	32768	320 × 200
⋮			

110Н	граф.	32768	640 × 480
111Н	граф.	65536	640 × 480
112Н	граф.	16777216	640 × 480
⋮			
116Н	граф.	32768	1024 × 768
117Н	граф.	65536	1024 × 768
118Н	граф.	16777216	1024 × 768
⋮			
11АН	граф.	65536	1280 × 1024
11ВН	граф.	16777216	1280 × 1024

4.3. Програмування відеоадаптерів у простіших режимах

Фактично будь-яка графічна операція зводиться до роботи з окремими пікселями, наприклад, вивести піксель заданого кольору. Цю задачу можна розв'язувати безпосереднім доступом до відеопам'яті, але при цьому потрібно знати структуру відеопам'яті у відповідному відеорежимі. *Структура відеопам'яті* – це спосіб упаковки інформації про пікселі в адресному просторі відеопам'яті та відображення цієї інформації на прямокутний растр дисплея.

Розглянемо приклади процедур виведення пікселів у деяких простіших графічних режимах [38].

Режим 4. Це режим 320 × 200 пікселів з чотирма кольорами. Цей режим підтримується відеоадаптерами CGA, EGA, VGA, SVGA. У адаптерів EGA, VGA, SVGA дані містяться в нульовому плані (решта планів не використовуються). Кожному пікселю відповідає 2 біти відеопам'яті. Колір верхнього лівого пікселя зберігається в бітах D7, D6 нульового байта відеопам'яті, наступного – в бітах D5, D4 і т.д. (рис. 4.1).

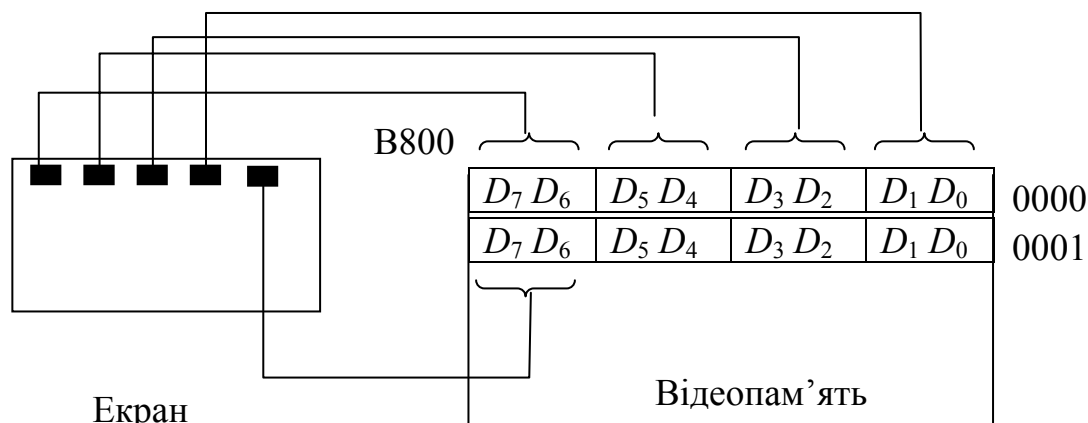


Рис. 4.1. Структура відеопам'яті в четвертому графічному режимі

Зміщення байта, що містить дані про піксель (x, y) можна знайти за формулою

зміщення = $40 * y +$ ціла частина від ділення x на 4, оскільки 320 пікселів займає 80 байтів відеопам'яті, але пікселі парних та непарних рядків знаходяться в різних місцях відеопам'яті. Всі непарні рядки записуються, починаючи з адреси B800:0000, а парні – з B800:2000.

Щоб знайти номери бітів у байті пам'яті, використовують формулу:

$$S = (3 - x \bmod 4) * 2.$$

Запишемо процедуру виведення пікселя кольору c з координатами x, y .

Procedure putpixel_4(x, y, c : integer);

```
var z, m, s : integer;
begin
  if (0 <= x) and ( x <=319) and (0 <= y) and (y <= 199) then
    begin
      z:=40*y + x shr 2;
      if odd(y) then z:=z+8192;
      s:= (3 - x and 3) shl 1;
      m:=3 shl s;
      mem[$B800:z]:=mem[$B800:z] and not m or (c shl s);
    end;
  end; {putpixel_4}
```

Застосовуючи процедуру putpixel_4, досить просто можна намалювати відрізок горизонтальної лінії між точками з абсцисами x_1, x_2 , за допомогою оператора (необхідно, щоб $x_1 \leq x_2$):

```
for x:=x1 to x2 do putpixel_4(x, y, c);
```

Вертикальний відрізок зображається за допомогою оператора

```
for y:=y1 to y2 do putpixel_4(x, y, c);
```

У циклі виконуються прості операції – приріст на одиницю і перевірка менше або дорівнює, тому операція малювання здійснюється швидко. Однак виникає питання: чи не можна побудувати більш швидкодійні алгоритми, використовуючи записи відповідних бітів у пам'яті, як це було зроблено в процедурі putpixel_4?

У цьому випадку необхідно враховувати логічну організацію відеопам'яті комп'ютера, в якій зберігаються біти/байти растра. Якщо записувати в четвертому режимі одразу 4 сусідні пікселі (1 байт у пам'яті), то можна істотно пришвидшити виведення горизонталей.

Використовуючи міркування для процедури putpixel_4, можна побудувати швидкий алгоритм виведення горизонтальної лінії. Для цього за значеннями координат x_1 та x_2 потрібно визначити адресу першого та

останнього неповних байтів. Для цих байтів необхідно виробити маску запису кольору. Повні байти будуть заповнені чотирма пікселями, тобто значенням $c\ c\ c\ c$.

Процедура побудови горизонтального відрізка в четвертому графічному режимі з прямим доступом до відеопам'яті набуде такого вигляду:

```

procedure hor(x1, x2, y, c : integer);
var c1, z1, z2, M1, M2, i : integer;
begin
  if ( 0 <= x1) and (x2 <= 319) and (x1 <= x2)
    and ( 0 <= y) and (y <= 199) then
    begin
      z1:=y*40 + x1 shr 2 + 1;  {адреса першого повного байта}
      z2:=y*40 + x2 shr 2 - 1;  {адреса останнього повного байта}
      if odd(y) then
        begin
          z1:=z1+8192;
          z2:=z2+8192;
        end;
      c1 := (c shl 6) or (c shl 4) or (c shl 2) or c; {маска повного байта}
      for i:=z1 to z2 do
        mem[$B800:i]:= c1;
      z1:= z1 - 1; {адреса першого неповного байта}
      z2:= z2 + 1; {адреса останнього неповного байта}
      M1 := 255 shr ((x1 and 3) shl 1); {маска першого байта}
      M2 := 255 shl ((3 - x2 and 3) shl 1); {маска останнього байта}
      if (z1 < z2) then begin
        mem[$B800:z1]:=mem[$B800:z1] and (not M1) or (c1 and M1);
        mem[$B800:z2]:=mem[$B800:z2] and (not M2) or (c1 and M2);
      end
      else
        begin
          M1 := M1 and M2;
          mem[$B800:z1]:=mem[$B800:z1] and (not M1) or (c1 and M1);
        end;
    end; {if}
end; {hor}

```

Для вертикалі бітова маска однакова для всіх пікселів, її слід заповнювати в циклі, але кількість кроків у такому циклі буде значно більшою, тому при однаковій довжині ліній горизонталі зображаються на екрані швидше, ніж вертикалі.

Режим 13H. Цей режим забезпечує 256 кольорів для пікселів та роздільну здатність 320 × 200. Атрибут кожного пікселя визначається 1 байтом. Відеопам'ять організована лінійно.

Зміщення байта, що містить дані про піксель (x, y), від початку сторінки відеопам'яті визначається за формулою $z = 320 * y + x$.

Процедура виведення пікселя в 13H режимі має такий вигляд:

```

procedure putpixel_13(x, y, c : integer);
begin
  if (0 <= x) and ( x <=319) and (0 <= y) and (y <= 199) then
    mem[$A000:y*320 + x]:=c;
  end; { putpixel_13 }

```

4.4. Програмування відеоадаптерів у режимах 0EH, 10H, 12H

Ці режими працюють з 16 кольорами і різною роздільною здатністю: 0EH – 640×200; 10H – 640×350; 12H – 640×480 та підтримуються адаптерами VGA, SVGA. Ділянка пам'яті A000:0000H не зберігає даних про колір пікселя. Для цього VGA використовує свою внутрішню пам'ять, що не є частиною оперативної пам'яті та безпосередньо не доступна програмісту. Ця внутрішня пам'ять представляється шарами, або планами. Кожному пікселю відповідає один біт із кожного плану.

Чотири біти на піксель дозволяють відображати 16 різних кольорів (табл. 3.1). Плани мають номерацію від 0 до 3 і свою адресацію байтів (рис. 4.2). Існує жорсткий зв'язок між планами і відеопам'яттю. VGA контролює всі операції з відеопам'яттю, перехоплює їх та інтерпретує по-своєму. Наприклад, запис у відеобуфер означає запис у плани (існує запис байтами та точками). Ці всі операції виконуються через численні регістри VGA, які заповнюються через відповідні порти.

Як приклад наведемо групу регістрів, що належить складовій частині відеокарти – графічному контролеру (табл. 4.5).

Таблиця 4.5

Номер	Регістр	Стандартні значення
0	Set/Reset	00
1	Enable Set/Reset	00
2	Color Compare	00
3	Data Rotate	00
4	Read Map Select	00
5	Mode	02
6	Miscellaneous	05
7	Color Don't Care	0F
8	Bit Mask	FF

Якщо виводити інформацію на екран безпосередньо через відеопам'ять, то необхідно вміти визначати біти, що відповідають пікселю з координатами (x, y) .

Оскільки в одному рядку є 640 пікселів і на кожний піксель відводиться 1 біт з кожного плану, то всі пікселі одного рядка займають $640 : 8 = 80$ байтів, а x пікселів неповного рядка займають $[x/8]$ байтів, тобто зміщення для пікселя з координатами (x, y) дорівнює $80 * y + [x/8]$.

Номер біта обчислюємо за формулами $s = 7 - x \bmod 8$ або $s = x \text{ and } 7$. Для ідентифікації позиції пікселя всередині байта часто використовують не номер біта, а бітову маску – байт, в якому відмінний від нуля тільки той біт, що знаходиться на позиції пікселя. Бітова маска задається виразом $128 \text{ shr } s$.

Зчитування байтами. Щоб прочитати байт із плану, потрібно налаштувати цей план. Це виконує регістр вибору плану читання Read Map Select, що належить графічному контролеру. Усі регістри завантажуються через порти. Але регістри працюють у парі – індексний регістр та регістр даних. Індексний регістр завантажується через порт ЗСЕН і містить номер функції читання, тобто 4 (табл. 4.5). Регістр даних завантажується через порт ЗСФН і містить номер плану N для зчитування (рис. 4.2).

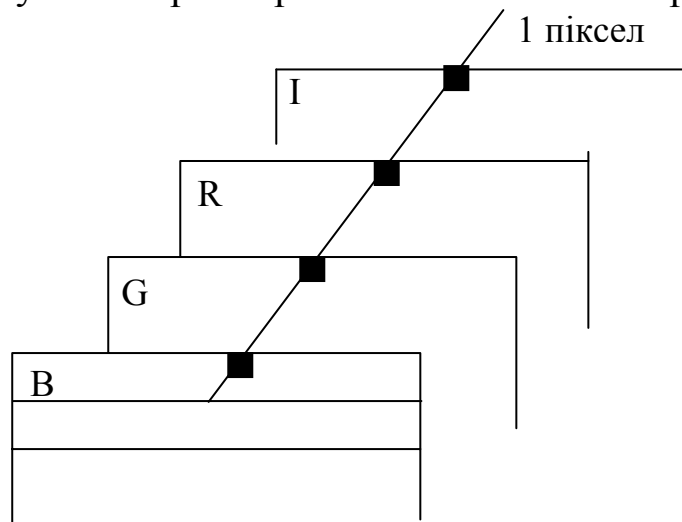


Рис. 4.2. Планарна організація відеопам'яті

Фрагмент програми для зчитування байта з плану N має вигляд

```
port[$3CE] := 4;
port[$3CF] := N;
byte:= mem[$A000:0000];
```

Запис байтами. При записі байтів у плани використовується регістри секвенсера, а саме регістр маски планів MapMask. Індексний регістр секвенсера завантажується через порт ЗС4Н і містить номер функції запису – число 2. Регістр даних секвенсера заповнюється через порт ЗС5Н і містить маску планів для запису.

Для утворення маски виставляємо 1 для тих планів, в які хочемо записувати байти, наприклад, якщо $Mask = 0101_2 = 5_{16}$, то запис відбувається у нульовий та другий плани.

Отже, фрагмент програми, що записує байт *byte* в ці плани, такий:

```
port[$3C4]:= 2;
port[$3C5]:= Mask;
mem[A000:0000]:=byte;
```

Процедура налаштування плану *N* ($0 \leq N \leq 3$) на читання і запис байтами має вигляд

```
procedure SetPlane(N : integer);
  begin
    port[$3CE]:=4; port[$3CF]:=N;
    port[$3C4]:=2; port[$3C5]:= 1 shl N;
  end;
```

Наприклад, для зафарбовування екрану в червоний колір потрібно в другий план записати байти з одиниць. Другий план має маску 4 (0100), її записуємо в порт 3C5H, у порт 3C4H заносимо номер функції 2. Отже, маємо такий фрагмент програми:

```
port[$3C4]:= 2; port[$3C5]:= 4;
for i:= 0 to 28000 do mem[A000:i]:=255; {640:8* 350=28000}
```

Запис та зчитування пікселів. Коли процесор читає байт даних із відеопам'яті, то зразу читаються 4 байти – по одному з кожного плану. При цьому прочитані байти автоматично записуються у спеціальні регістри-затвори (*latches-регістри*), які не мають прямого доступу.

При операції запису байт, який посилається процесором, накладається на вміст регістрів-затворів за правилами, що визначаються значеннями інших регістрів, і результуючі 4 байти записуються в плани.

Правила, що визначають накладання даних на значення *latches-регістрів*, при записі визначаються режимом запису. Режим читання задає правила, згідно з якими визначається значення, яке прочитає процесор.

Вибір режиму здійснюємо за допомогою регістра *Mode* графічного контролера (функція 5), тому в порт 3CEH записуємо 5. Регістр даних (порт 3CFH) заповнюється номером режиму. Номери режиму читання (0, 1) задаються у третьому біті байта *Mode*, а режими запису (00, 01, 10, 11) – в нульовому та першому бітах цього ж байта (рис. 4.3).

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	X	0	X	X

Рис. 4.3. Заповнення регістра *Mode*

Тому, задаючи режим читання, одночасно визначаємо й нульовий режим запису, і навпаки, задаючи режим запису, задаємо нульовий режим читання.

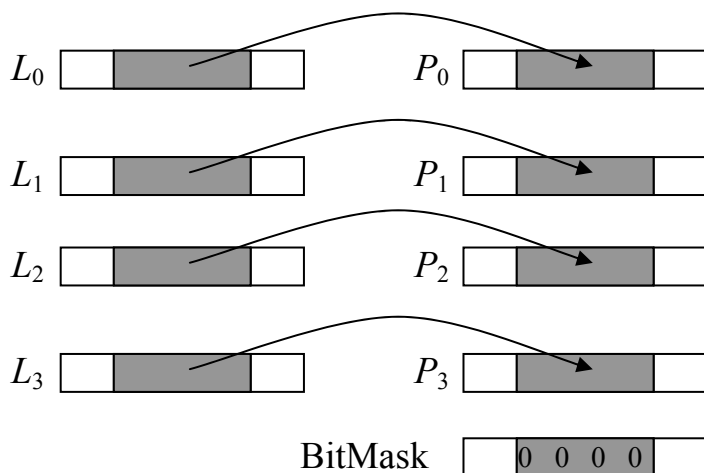


Рис. 4.4. Переміщення вмісту затворів у плани

Розглянемо нульовий режим запису. Це один із найскладніших режимів запису, але використовується він найчастіше. У цьому режимі бере участь група регістрів графічного контролера. Перший із них – регістр BitMask (його номер – 8) дозволяє захистити від зміни певні пікселі. У тих позиціях, де біти в BitMask встановлені в нуль, пікселі не змінюють свого значення, тобто у відповідні біти планів будуть записані біти затворів.

Якщо деякий біт у BitMask встановлений в одиницю, то в дію вступають ще два регістри: Set/Reset (його функція має номер 0) і Enable Set/Reset (функція – 1), який регулює дію першого регістра. У цих регістрах використовуються тільки молодші 4 біти.

Якщо біт у BitMask дорівнює 1, то у всі відповідні позиції планів записується вміст регістра Set/Reset. Але це буде зроблено для тих планів, для яких є дозвіл другого регістра Enable Set/Reset (відповідний біт цього регістра дорівнює 1). У протилежному випадку в план, закритий регістром Enable Set/Reset, буде записаний відповідний біт байта даних (рис. 4.5).

Отже, якщо Enable Set/Reset = 0FH, тоді в позиції планів, що дозволяються регістром BitMask, записується код кольору, заданий у регістрі Set/Reset. Байт, що посилається процесором, жодної ролі не відіграє. Для того, щоб нарисувати один піксель, необхідно завантажити регістр BitMask так, щоб захистити від зміни решту 7 пікселів.

Регістри BitMask і Enable Set/Reset регулюють дію байта даних на відеопам'ять і можуть замінювати окремі біти планів на вміст затворів або регістра Set/Reset.

Отже, процедура запису пікселя color із координатами x, y має вигляд

```

procedure putpixel_10(x, y, color : integer);
var s, bitmask:integer;
begin
  s:= x and 7;
  bitmask := 128 shr s;  {бітова маска}
  port[$3CE] := 5;  {встановити 0 – режим запису}
  port[$3CF] := 0;
  port[$3CE] := 1; {дозвіл на запис вмісту Set/Reset в усі плани}
  port[$3CF] := $F;
  port[$3CE] := 0; {в регістр Set/Reset завантажити код кольору}
  port[$3CF] := color;
  port[$3CE] := 8; {завантажити регістр bitmask}
  port[$3CF] := bitmask;
  mem[$A000: y*80 + (x shr 3)] := 1; {запис у пам'ять}
  {відновити регістри до стандартного значення}
  port[$3CE] := 1; port[$3CF] := 0;
  port[$3CE] := 8; port[$3CF] := $FF;
  port[$3CE] := 5; port[$3CF] := 10B;
end; {putpixel_10}

```

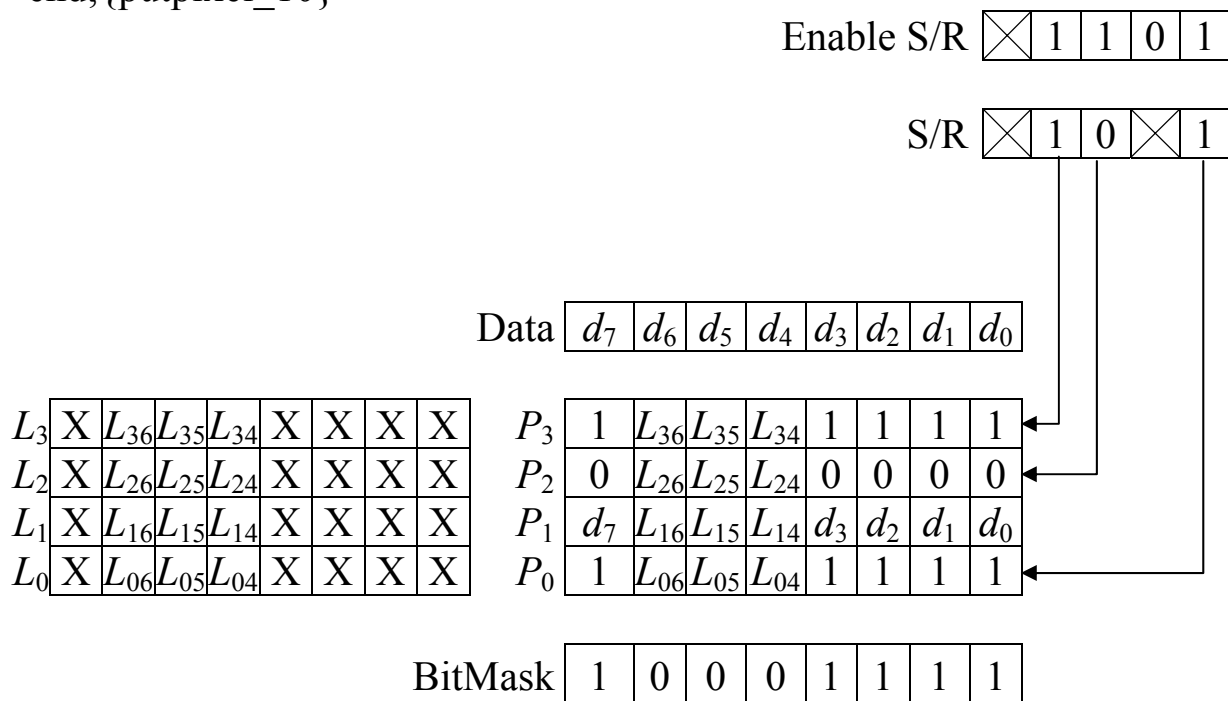


Рис. 4.5. Побітне заповнення планів

При читанні процесором байта даних із відеопам'яті, відповідні байти планів запам'ятовуються в регістрах-затворах (регістри знаходяться на платі відеоадаптера і програмісту недоступні). Якщо потім вико-

нується цикл запису, то над даними, що знаходяться в цих регістрах, і над даними, які надходять від процесора, можуть виконуватися такі логічні операції (без участі процесора):

- запис даних процесора у відеопам'ять без зміни;
- циклічний зсув байта даних, що записується процесором у пам'ять;
- виконання логічних операцій AND, OR, XOR над даними, що записуються процесором у відеопам'ять і величинами регістрів-затворів.

У відеопам'ять записується результат виконання цих операцій. Цими функціями керує регістр Data Rotate (номер 3). У бітах $D_0 - D_2$ регістра даних задаємо, на скільки розрядів потрібно зсунути байт (зсув тільки вправо і є циклічним). Біти D_3, D_4 використовуються для задання логічних операцій

- 00 – запис немодифікованих даних;
- 01 – виконання над даними логічної операції AND, 10 – OR, 11 – XOR.

У режимі зчитування пікселя Read1 в дію вступає регістр Color Compare (порівняння кольору) і VGA порівнює кожен байт в адресованому байті зі значенням, що знаходиться в цьому регістрі. Якщо є збіг цих значень, то в байті ознак у цьому розряді виставляється 1, інакше 0. Якщо тепер циклічно завантажувати регістр Color Compare різними кольорами, то, аналізуючи байт ознак, можна визначити колір пікселя.

Контрольні питання та завдання

1. Назвіть основні пристрої відеоадаптера та їх основні параметри.
2. Дайте класифікацію відеоадаптерів та назвіть їх характеристики.
3. Яке призначення мають графічні процесори? Які задачі вони розв'язують?
4. Охарактеризуйте інтерфейс відеокарти.
5. Опишіть поняття роздільної здатності бітової глибини.
6. Що таке відеосторінки, яке їх призначення?
7. Які ви знаєте стандартні відеорежими, як їх установити?
8. Як визначити обсяг відеопам'яті в різних графічних режимах?
9. Наведіть приклади VESA-режимів. Опишіть процедуру їх установки.
10. Знайдіть зміщення байта, що містить дані про піксель з координатами (x, y) у режимі 16 кольорів?
11. Як здійснюється вибір режиму запису та зчитування пікселів?
12. Яку роль відіграють регістри-затвори?
13. Назвіть регістри графічного контролера та опишіть їх призначення.
14. Яка будова внутрішньої пам'яті відеоадаптера?

Вправи і задачі для самостійного виконання

1. Записати процедуру зчитування поточного відеорежиму.
2. Графічний режим $640 \times 480 \times 256$. Яка відносна (від початку відеопам'яті) адреса атрибута пікселя з координатами (x, y) на екрані?
3. Графічний режим $640 \times 480 \times 16$. Задана відносна адреса відеопам'яті 2AFH. Для яких точок (x, y) записані атрибути за цією адресою? Скільки бітів містить атрибут пікселя? Який його формат?
4. Графічний режим True Color. Точка на екрані має координати $x = 50, y = 20$. Яка відносна адреса байта, в який буде записано значення R, G, B для цієї точки?
5. Нехай задана система з монітором 8×10 дюймів на якому можна отримати зображення 100 dpi. Якщо пам'ять складається з однобайтових слів, вихідний адрес буфера кадру 0000, а кожному пікселю відповідає один байт пам'яті, то яким буде адрес буфера кадру для пікселя з екранними координатами (x, y) ?
6. Скласти програму, яка переміщуватиме прямокутний блок пікселів з одного місця екрана в інше. Режим 13H. Функціями BIOS, VBI не користуватися.
7. Записати процедуру putpixel_10(), використовуючи функції відео-BIOS.
8. Записати процедуру getpixel_10().
9. Написати підпрограму для зчитування поточного VESA-режиму.
10. Роздільна здатність екрана 1280×1024 , частота кадрів – 72 Гц. Знайти швидкодію пам'яті буфера кадру, тобто визначити скільки часу буде займати зчитування коду кольору одного пікселя?
11. Як довго буде завантажуватися буфер кадру в режимі True Color/High Color з роздільною здатністю 1280×1024 , якщо за одну секунду може передаватися 10^7 бітів? До скількох пікселів за секунду може мати доступ контролер дисплея такої системи?
12. Знайти час доступу до одного пікселя в растровій системі з роздільною здатністю 640×480 і частотою оновлення 80 кадрів за секунду.

Тема 5. Растрові алгоритми генерування кривих ліній

Графічні пристрої, що використовуються в КГ, є в основному растровими, тому при синтезі графічних зображень, що складаються з окремих простих елементів, виникає необхідність у растрових алгоритмах.

Растровий алгоритм – це алгоритм, який для роботи програми, що реалізує графічні примітиви, враховує властивість растра.

Хоча більшість графічних бібліотек містить у собі достатню кількість простіших растрових алгоритмів, часто виникає необхідність явної побудови растрових алгоритмів.

При побудові графічних зображень вдаються до певного набору графічних примітивів, однак найчастіше в графічних системах використовуються лінії.

Зобразити криву на дискретній поверхні (екрані) означає знайти таку її цілочислову апроксимацію, яка дасть можливість відтворити неперервну криву.

Наприклад, для відрізка прямої потрібно отримати послідовність дискретних точок (пікселів), що апроксимують неперервну лінію (рис. 5.1).

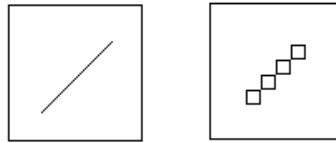


Рис. 5.1 Наближення пікселями неперервної лінії

Методи побудови ліній на дискретній поверхні поділяються на 2 класи:

- числові методи (методи прямих обчислень координат);
- інкрементні методи.

5.1. Числові методи

Числові методи базуються на числовому аналізі рівнянь кривих, тобто на обчисленні значень функцій.

Розглянемо пряму лінію загального вигляду. Для її зображення теж необхідно обчислювати координати кожного пікселя. Відомо декілька методів розрахунку координат кожного пікселя лінії.

Нехай задано координати кінців відрізка $A(x_1, y_1)$, $B(x_2, y_2)$, тоді рівняння прямої, що проходить через дві точки A і B , має вигляд $y = ax + b$, де

$$a = \frac{y_2 - y_1}{x_2 - x_1}, \quad b = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}.$$

Для простоти вважатимемо, що $0 \leq y_2 - y_1 < x_2 - x_1$, тобто цикл у програмі буде здійснюватись по x .

За цих припущень наведемо приклад запису простішого алгоритму побудови прямої лінії на мові Pascal.

```
procedure Line_1 (x1, y1, x2, y2, c : integer);  
var a, b : real, x : integer;  
begin  
    a:= (y2-y1) / (x2-x1);  
    b:= y1- a*x1;  
    for x:= x1 to x2 do  
        putpixel (x, int (a*x+b), c);  
end; {Line_1}
```

Недоліком такої програми є те, що в циклі виконується множення та використовуються операції з плаваючою крапкою. А це зменшує швидкість виведення графічного зображення.

Обчислення значень функції $y = ax + b$ можна уникнути, якщо використати рекурентне співвідношення $y = y + a$, оскільки при зміні x на одиницю значення функції y змінюється на a . Ця модифікація алгоритму має такий вигляд:

```
procedure Line_2 (x1, y1, x2, y2, c : integer);  
var a : real, x : integer;  
begin  
    a:= (y2 - y1) / (x2 - x1);  
    y:= y1;  
    for x:= x1 to x2 do  
        begin  
            putpixel (x, int y, c);  
            y:=y + a;  
        end;  
end; {Line_2}
```

Оскільки виділення цілої частини значення y не завжди може привести до коректного графічного результату, то покращити зовнішній вигляд відрізка можна за рахунок округлення значень функції до ближчого цілого, тобто це означає, що з двох пікселів, які лежать на одній вертикалі (так, що відрізок прямої проходить між ними), завжди вибирається той піксель, який лежить ближче до прямої. Для цього досить порівняти дробову частину значення y з $0,5$.

Зауваження 1. Вибір одного з двох пікселів можна здійснити шляхом визначення середини між двома пікселями-кандидатами і перевірки,

місцезнаходження (вище чи нижче середини) точки перетину відрізка прямої з даною вертикаллю. Це метод серединної точки.

У цьому алгоритмі для обчислення координати y в тілі циклу є тільки одна операція '+', тому порівнюючи наведені алгоритми, останній кращий за швидкодією.

Але незважаючи на те, що вхідні дані є цілочисловими величинами і всі результати теж мають бути цілочисловими, ця модифікація алгоритму все одно використовує дійсні числа. Окрім цього, оскільки для обчислення y в останньому варіанті використовується операція $y := y + a$, то з кожним кроком циклу будуть накопичуватись похибки за рахунок неточного представлення дробових чисел у комп'ютері та за рахунок похибки арифметичних операцій із плаваючою крапкою. Тому в результаті виконання циклу може статися так, що на останньому кроці $y \neq y_2$ (хоча з математичної точки зору тут все коректно, при $x = x_2$ маємо $y = y_2$). Це теж необхідно враховувати при використанні даного алгоритму.

Отже, числові методи мають перевагу завдяки простоті та ясності побудови алгоритму і можливості працювати з дробовими значеннями координат точок відрізка.

Однак у числових методів є й свої недоліки:

- 1) використання операцій із плаваючою крапкою та операцій множення або ділення в циклі зумовлює малу швидкість обчислень, хоча це суттєво залежить від процесора (у сучасних комп'ютерах, де процесори використовують ефективні засоби прискорення, наприклад, конвеєр арифметичних операцій із плаваючою крапкою, час виконання цілочислових операцій не набагато менший);
- 2) при обчисленні координат шляхом додавання приросту може накопичуватись похибка обчислення координат.

5.2. Інкрементні алгоритми

У 1965 році Брезенхем (Bresenham J.E.) запропонував підхід для створення інкрементних методів (тоді ці методи використовувались для графопобудовників).

Основною ідеєю інкрементних методів є теж рекурентні співвідношення для обчислення y , але в них уже не застосовуються дійсні обчислення й операції множення та ділення, а застосовуються лише цілочислові операції додавання і віднімання.

Інкрементні алгоритми виконуються як послідовність обчислень координат сусідніх точок шляхом додавання відповідних приростів так, щоб сусідня цілочислова точка була найближчою до неперервної кривої. Прирости розраховуються на основі аналізу функції похибок. У циклі

виконуються лише цілочислові операції порівняння, додавання та віднімання. У такий спосіб досягається більш висока швидкодія обчислень координат кожного пікселя порівняно з числовими методами.

На сьогодні існує багато спеціалізованих інкрементних алгоритмів для генерування кривих того чи іншого типу (наприклад відрізків, кіл, еліпсів тощо). В основному вони реалізовані апаратно.

5.2.1. Алгоритм Брезенхема для відрізка

Серед графічних примітивів найбільшу питому вагу мають відрізки прямих. У зв'язку з цим при розробці графічних засобів алгоритмам лінійної інтерполяції приділяють особливу увагу.

Розглянемо спочатку простіший випадок формування відрізка прямої, коли пряма проходить через початок координат $(0,0)$ і точку з координатами (n, m) . Рівняння цієї прямої має вигляд

$$mx - ny = 0.$$

При цьому виникає задача за відомою точкою (x, y) , що найкраще наближає пряму, визначити наступну точку $(x+1, y)$ чи $(x, y+1)$. Введемо в розгляд функцію $F(x, y) = mx - ny$. Нехай значення $F(x, y) = f \neq 0$. Знайдемо

$$F(x + 1, y) = m(x + 1) - ny = mx - ny + m = f + m,$$

$$F(x, y + 1) = mx - n(y + 1) = mx - ny - n = f - n.$$

Щоб зробити вибір між двома точками, потрібно знати, яке з двох чисел $f + m$ чи $f - n$ знаходиться ближче до нуля.

Якщо значення $f + m$ ближче до нуля, то рух із точки (x, y) відбудеться в точку $(x + 1, y)$, а якщо $f - n$ ближче до нуля, то в точку $(x, y + 1)$.

Для знаходження приросту координат (x, y) побудуємо функцію похибок $S(x, y) = F(x + 1, y) + F(x, y + 1) = 2f + m - n$ і виробимо ознаку вибору наступної точки. Для цього потрібно розглянути такі випадки:

- 1) якщо $f + m < 0, f - n < 0$, тобто $f + m$ є ближчим до нуля (рис. 5.2, а), а $S = 2f + m - n < 0$, то наступною є точка $(x + 1, y)$;
- 2) якщо $f + m > 0, f - n > 0$, тобто $f - n$ є ближчим до нуля (рис. 5.2, б), а $S = 2f + m - n > 0$, то рух відбувається в точку $(x, y + 1)$;
- 3) якщо $f + m > 0, f - n < 0$, тобто $f + m$ є ближчим до нуля (рис. 5.2, в), а $S = 2f + m - n < 0$, то надалі вибираємо точку $(x + 1, y)$;
- 4) якщо $f + m < 0, f - n > 0$, тобто $f - n$ є ближчим до нуля (рис. 5.2, г), а $S = 2f + m - n > 0$, то рух відбувається в точку $(x, y + 1)$;

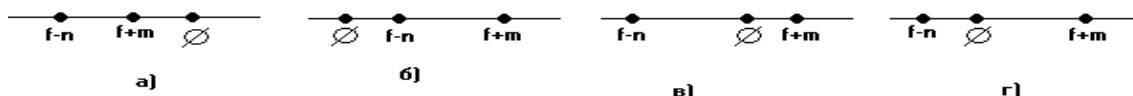


Рис 5.2. Розміщення точок

Отже, вибір наступної точки визначається знаком функції похибок S : якщо $S > 0$, то наступною є точка $(x, y + 1)$, а якщо $S \leq 0$, то $(x + 1, y)$. При переході до наступної точки змінюється значення S , тому для простого обчислення S потрібно мати рекурентну формулу.

Оскільки $S(x, y) = 2f(x, y) + m - n$, то початкове значення $S_0 = S(0, 0) = m - n$. Розглянемо

$$S(x+1, y) = 2f(x+1, y) + m - n = 2(mx - ny + m) + m - n = 2f + m - n + 2m = S(x, y) + 2m;$$

$$S(x, y+1) = 2f(x, y+1) + m - n = 2(mx - ny - n) + m - n = 2f + m - n - 2n = S(x, y) - 2n;$$

Таким чином, маємо такі рекурентні формули для S :

$$S = S + 2m, \text{ якщо } S \leq 0;$$

$$S = S - 2n, \text{ якщо } S > 0.$$

Один із варіантів алгоритму Брезенхема в простішому випадку має вигляд

```
x:= 0; y:= 0; putpixel (0, 0, c);
m2:=m+m; n2:=n+n; S:=m-n;
for i:=1 to m+n do begin
if S= < 0 then begin S:=S+m2 ; x:=x+1; end
else begin S:=S-n2 ; y:=y+1; end;
putpixel (x,y,c);
end {for}.
```

У цьому алгоритмі перехід із точки (x, y) до наступної точки може бути виконаний двома способами. Це може викликати подвійне зображення прямої лінії, тому потрібно передбачити приріст по x і по y одночасно, тобто як сусідню необхідно розглядати ще і точку $(x + 1, y + 1)$. Щоб зобразити весь відрізок між точками $(0, 0)$ та (m, n) , потрібно зробити $l = \max\{m, n\}$ кроків. Отже, приходимо до загального алгоритму Брезенхема для довільного відрізка з координатами кінців (x_1, y_1) і (x_2, y_2) .

```
procedure Line_3 (x1, y1, x2, y2, c:integer);
{Bresenham algorithm}
var k, l, dy, S, m, n : integer;
begin
if (x1 > x2) then begin S:=x1; x1:=x2; x2:=S;
S:=y1; y1:=y2; y2:=S; end;
if y1 > y2 then dy:= -1 else dy:=1;
m:=abs(y2-y1); n:=x2-x1;
S:=m-n;
if m>n then l:=m else l:=n;
m:=m+m; n:=n+n;
putpixel (x1,y1,c);
```

```

for k:=1 to l do
  begin
    if S=<0 then begin S:=S+m: x1:=x1+1; end;
    if S>0 then begin S:=S-n; y1=y1+dy; end;
    putpixel (x1,y1,c);
  end {for};
end; {Line_3}

```

Зауваження 2. Для побудови широкої лінії можна її нарисувати декілька разів із певним зміщенням початкових координат або модифікувати алгоритм Брезенхема так, щоб замість оператора виведення пікселя *putpixel (x,y,c)* виступав оператор виведення фігури так званого логічного пера з центром у точці (x, y) . Як фігуру пера можна обрати прямокутник, круг, відрізки прямої тощо. Такий підхід до побудови ліній має переваги і недоліки. Недоліком є неефективність для деяких форм пера; наприклад, якщо перо має форму квадрата, то більшість пікселів при цьому неоднократно перефарбовується (рис. 5.3).

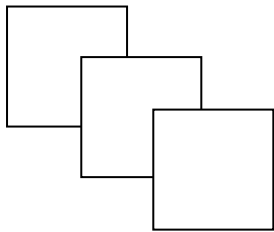


Рис. 5.3. Побудова широкої лінії

Такі алгоритми більш ефективні, коли перо має форму відрізка – найчастіше вертикального або горизонтального, але горизонтальне перо не може намалювати широку горизонтальну лінію.

Зауваження 3. Лінію можна зображати не завжди суцільною (лінії можуть мати різні структури та типи). Для цього в алгоритм рисунка лінії передають ще один аргумент (це ціле число без знака), двійковий код якого багаторазово копіюється вздовж лінії. Кожний біт при цьому циклічно аналізується. Якщо значення біта дорівнює одиниці, то піксель зображаємо, якщо біт дорівнює нулю, то піксель не зображаємо.

Зауваження 4. Для зменшення ступінчастості растрового зображення лінії використовують ефект згладжування. На дисплеях з малою роздільною здатністю екрана 320×200 згладжування не дає ефекту. Для досягнення ефекту згладжування необхідні режими з вищою роздільною здатністю, наприклад, 640×480 . Згладжування при цьому виконується за рахунок зміни яскравості пікселів, тому можливість згладжування залежить від шкали яскравості. Наприклад, для ребер многокутника можна встановити яскравість пікселів пропорційно

площі частини пікселя, що попадає в середину многокутника. Згладжування відрізків ще можна проводити шляхом розмивання різкої границі за рахунок підсвічування пікселів, які прилягають до відрізка. На дисплеях із роздільною здатністю 1280×1024 згладжування відбувається за рахунок малого зерна дисплея.

5.2.2. Алгоритм Брезенхема для кола

Для побудови кола $x^2 + y^2 = R^2$ можна реалізувати алгоритм шляхом прямого обчислення координат. Або, маючи програму побудови відрізка, коло можна апроксимувати хордами, але коло буде недостатньо гладким, поки кількість хорд не стане більшою 36. При високих роздільних здатностях потрібна ще більша кількість хорд. Ці методи вимагають багато обчислень, що зменшує швидкість побудови кола. Тому розглянемо інкрементний метод Брезенхема для кола.

Для спрощення алгоритму растрової розгортки кола можна використати його симетрію відносно координатних осей і прямих $y = \pm x$. Тому достатньо побудувати растрове зображення для $1/8$ частини кола, а решту частин побудувати за рахунок симетрії кола (якщо точка (x, y) належить колу, то і точки $(-x, y)$, $(x, -y)$, $(-x, -y)$, (y, x) , $(-y, x)$, $(y, -x)$, $(-y, -x)$ належать колу).

Із цією метою для виведення кола з центром у точці (x_c, y_c) побудуємо таку процедуру:

```
procedure circle_8 (x, y, xc, yc, c: integer);
begin
  putpixel (xc+x, yc+y, c);
  putpixel (xc+y, yc+x, c);
  putpixel (xc-x, yc+y, c);
  putpixel (xc-y, yc+x, c);
  putpixel (xc+x, yc-y, c);
  putpixel (xc+y, yc-x, c);
  putpixel (xc-x, yc-y, c);
  putpixel (xc-y, yc-x, c);
end; {circle_8}
```

Тепер розглянемо ділянку кола з другого октанта, тобто при $x \in [0, R/\sqrt{2}]$, $y \in [R/\sqrt{2}, R]$. Далі для кожної точки $P(x, y)$ вводимо змінну $D(P)$ за правилом: якщо точка P є зовнішньою до кола, то $D(P) = x^2 + y^2 - R^2$, а якщо внутрішньою, то $D(P) = R^2 - (x^2 + y^2)$. Щоб на i -му кроці визначити, яка з двох точок зовнішня S_i чи внутрішня T_i найкраще апроксимуватиме коло, потрібно порівняти величини $D(S_i)$ та $D(T_i)$.

Нехай $d_i \equiv D(S_i) - D(T_i)$. Якщо $d_i < 0$, то на i -му кроці вибирається точка S_i , оскільки $D(S_i) < D(T_i)$, якщо $d_i \geq 0$, то вибирається точка T_i .

Розглянемо перший крок алгоритму. З рівняння кола очевидно, що точка $(0, R)$ лежить на колі. На першому кроці потрібно вибрати точку серед $S_1(1, R)$ чи $T_1(1, R-1)$ (рис. 5.4), яка найкраще наближає коло. Вибір здійснюється тільки серед двох точок тому, що кутовий коефіцієнт дотичної до кола в цьому октанті належить проміжку $[-1; 0]$.

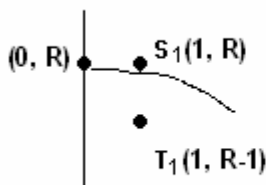


Рис. 5.4

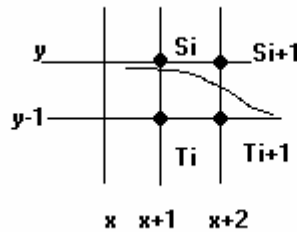


Рис. 5.5

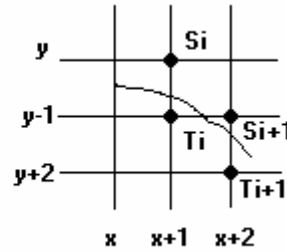


Рис. 5.6

Знайдемо $d_1 = D(S_1) - D(T_1) = 1 + R^2 - R^2 - (R^2 - (1 + (R - 1)^2)) = 3 - 2R$. Якщо $d_1 = 3 - 2R > 0$, то на першому кроці вибираємо внутрішню точку $T_1(1, R - 1)$, а якщо $d_1 = 3 - 2R < 0$, то вибираємо зовнішню точку $S_1(1, R)$.

Вибір точки на $i + 1$ -му кроці залежить від того, яка точка була вибрана на i -му кроці – S_i чи T_i (рис. 5.5).

Нехай на i -му кроці була вибрана точка S_i , тобто $d_i = D(S_i) - D(T_i) < 0$, $d_i = (x + 1)^2 + y^2 - R^2 - (R^2 - (x + 1)^2 - (y - 1)^2)$. Знайдемо $d_{i+1} = D(S_{i+1}) - D(T_{i+1}) = (x + 2)^2 + y^2 - R^2 - (R^2 - (x + 2)^2 - (y - 1)^2) = d_i + 4x + 6$.

Нехай $d_i > 0$, тобто на i -тому кроці краще апроксимує коло точка T_i (рис. 5.6), тоді $d_{i+1} = D(S_{i+1}) - D(T_{i+1}) = (x + 2)^2 + (y - 1)^2 - R^2 - (R^2 - (x + 2)^2 - (y - 2)^2) = d_i + 4(x - y) + 10$.

Запишемо алгоритм Брезенхема для кола.

```
x:= 0; y:= R; d:=3-2*R;
while (x <=y) do begin
circle_8 (x, y, xc, yc, c);
if d < 0 then d:= d+4*x+6
else begin d:= d+4*(x-y)+10; y:=y-1; end;
x:= x+1;
end; {while}
```

Зауваження 5. Щоб одержати широке коло, необхідно нарисувати декілька кіл усередині даного кола, послідовно зменшуючи радіус на одиницю (або зовні даного кола).

Зауваження 6. Кола можуть мати різну структуру. Для побудови таких кіл циклічно аналізуються біти із заданого байта або слова, як і при побудові прямої лінії.

5.2.3. Інкрементний алгоритм виведення еліпса

Цей алгоритм більш складний, ніж алгоритм побудови кола. Внаслідок симетрії еліпса відносно осей координат координати точок еліпса обчислюватимуться лише в першій чверті. Канонічне рівняння еліпса має вигляд $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$, де a та b – велика і мала півосі еліпса.

Зведемо рівняння еліпса до такого вигляду:

$$\frac{x^2}{a^2} + \frac{y^2}{(b+0.5)^2} = 1$$

Звідси маємо

$$\frac{x^2}{a^2} + \frac{(2y)^2}{(2b+1)^2} = 1, \quad \text{або} \quad (2b+1)^2 x^2 + 4a^2 y^2 = (2b+1)^2 a^2.$$

Позначимо $(2b+1)^2 = m$, $a^2 = n$, $F(x, y) = mx^2 + 4ny^2 - mn$, тоді рівняння еліпса матиме вигляд $F(x, y) = 0$.

Ідея алгоритму полягає ось у чому: для кожного y , що змінюється від b до 0, необхідно підібрати ціле значення x так, щоб $F(x, y)$ було найближчим до нуля (точка (x, y) найкраще апроксимує еліпс).

Якщо $F(x, y) < 0$, то збільшуємо x ($x := x + 1$), якщо $F(x, y) > 0$, то зменшуємо y ($y := y - 1$). Але знак функції $F(x, y)$ не може бути ознакою вибору точки (x, y) .

Для вибору точки (x, y) , що найкраще апроксимує еліпс, утворюємо функцію похибок $S(x, y)$.

$$S(x, y) = F(x, y) + F(x+1, y) = 2m(x^2 + x) + 8ny^2 - 2mn + m.$$

Тоді, якщо $S(x, y) < 0$, то збільшуємо $x := x + 1$, (рис. 5.7, а); якщо $S(x, y)$ стає більшим нуля (рис. 5.7, б), то ми підбрали потрібну точку x (цей піксель зображаємо на екрані) і здійснюємо перехід до наступного y ($y := y - 1$). Отже, точка x – це перша точка, для якої $S(x, y) > 0$.

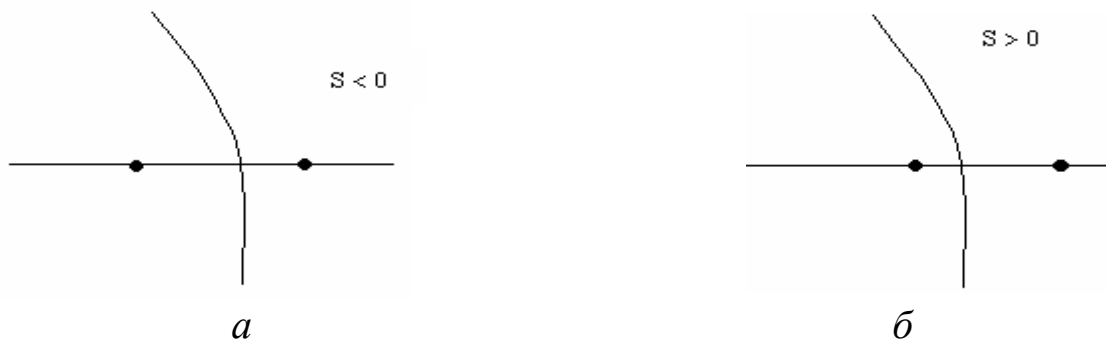


Рис. 5.7. Вибір пікселя

Для початкової точки $(0, b)$ $S_0 = 8nb^2 - 2mn + m$. Для наступних точок (x, y) потрібно мати значення $S(x, y)$, але його не можна знайти безпосередньо, тому що в цей вираз входять операції множення (втра-

чається швидкодія алгоритму). Для обчислення значень $S(x, y)$ побудуємо рекурентні співвідношення без операцій множення та ділення.

Нехай $S(x+1, y) = S(x, y) + P(x, y)$, тоді $P(x, y) = (S(x+1, y) - S(x, y)) = 2m((x+1)^2 + (x+1)) + 8ny^2 - 2mn + m - 2m(x^2 + x) - 8ny^2 + 2mn - m = 4mx + 4m$.

З останнього співвідношення видно, що при збільшенні x на одиницю, P збільшується на $4m$, тому для P маємо рекурентне співвідношення $P(x+1, y) = P(x, y) + dp$, де $dp = 4m$, $P(0, y) = 4m$.

Аналогічно при зміні y на $y - 1$ маємо

$Q(x, y) = S(x, y-1) - S(x, y) = 2m(x^2 + x) + 8n(y-1)^2 - 2mn + m - 2m(x^2 + x) - 8ny^2 + 2mn - m = -16ny + 8n$, тобто $Q(x, y-1) = Q(x, y) + dq$, $dq = 16n$, $Q(0, b) = -16nb + 8n$.

Отже, одержуємо таку процедуру побудови еліпса.

```
procedure Ellips (x0, y0, Rx, Ry, c: integer);
var S, P, Q, m, n, dp, dq: longint, a, b, x, y: integer;
begin
  b:= Ry; a:=Rx; n:=a*a; m:=sqr(2*b+1);
  x:= 0; S:=8*n*b*b-2*m*n+m;
  P:=4*m; dp:= P; Q:= -16*n*b+8*n; dq:=16*n;
  putpixel (x0, y0 + Ry, c); putpixel (x0, y0 - Ry, c);
  for y:=Ry downto 0 do
  begin
    if S < 0 then while S < 0 do
      begin
        x:= x+1; S:= S+P; P:= P+dp;
        P4(x0, y0, x, y, c);
      end;
    S:=S+Q; Q:=Q+dq; P4(x0, y0, x, y, c);
  end {for};
end; {Ellips}
```

Наведемо процедуру побудови симетричних точок відносно осей координат.

```
procedure P4(x0, y0, x, y, c: integer);
begin
  putpixel (x0 + x, y0 + y, c);
  putpixel (x0 + x, y0 - y, c);
  putpixel (x0 - x, y0 + y, c);
  putpixel (x0 - x, y0 - y, c);
end; {P4}
```

Зауваження 7. Якщо в основній процедурі замість оператора виклику процедури P4 вставити оператори

Нор ($x_0 - x, x_0 + x, y_0 + y, c$);

Нор ($x_0 - x, x_0 + y, y_0 - y, c$);

то одержуємо процедуру зафарбовування еліпса.

5.2.4. Інкрементний метод Жордана

Нехай крива задана неявним рівнянням $f(x, y) = 0$ і похідні $f_x, f_y, f_{xx}, f_{xy}, f_{yy}$ та ін. – неперервні. Припустимо, що деяка цілочислова точка $P(x, y)$ найкраще апроксимує криву $f(x, y) = 0$.

Метод Жордана полягає в тому, щоб, рухаючись уздовж кривої, з максимально можливою точністю підбирати наступні цілочислові точки. Із точки $P(x, y)$ можна перейти до сусідніх точок вісьмома способами, тобто сусідні точки можуть мати координати $(x \pm 1, y), (x, y \pm 1), (x \pm 1, y \pm 1)$ (рис. 5.9).

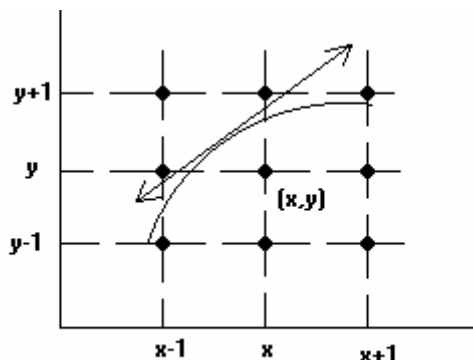


Рис. 5.9. Сусідні точки

Загальна формула сусідніх точок $(x + \Delta x, y), (x, y + \Delta y), (x + \Delta x, y + \Delta y)$, де $\Delta x, \Delta y = \pm 1$.

Знаючи знаки $\Delta x, \Delta y$, вибір серед восьми сусідніх точок зводиться до вибору серед трьох точок. Рух по кривій у точці $P(x, y)$ замінюємо рухом по дотичній. Можливі два напрямки руху вздовж дотичної $V^+ = (-f_y, f_x), V^- = (f_y, -f_x)$.

Якщо вибрати напрям V^+ , то знак Δx протилежний знаку f_y , а знак Δy збігається зі знаком f_x .

Якщо вибрати напрям V^- , то знак Δx збігається зі знаком f_y , а знак Δy протилежний знаку f_x .

Покладемо

$$d = \begin{cases} 1, & \text{якщо вибрано напрям } V^+, \\ 0, & \text{якщо вибрано напрям } V^-. \end{cases}$$

Тоді здійснюємо вибір:

$$\begin{aligned} &\text{якщо } (f_y < 0 \text{ і } d=1) \text{ або } (f_y \geq 0 \text{ і } d=0), \text{ то } \Delta x = 1; \\ &\text{якщо } (f_y \leq 0 \text{ і } d=1) \text{ або } (f_y < 0 \text{ і } d=0), \text{ то } \Delta x = -1; \\ &\text{якщо } (f_x \geq 0 \text{ і } d=1) \text{ або } (f_x < 0 \text{ і } d=0), \text{ то } \Delta y = 1; \\ &\text{якщо } (f_x < 0 \text{ і } d=1) \text{ або } (f_x \geq 0 \text{ і } d=0), \text{ то } \Delta y = -1. \end{aligned} \quad (5.3)$$

Тепер серед трьох точок вибираємо найближчу точку до кривої $f(x, y) = 0$, тобто точку для якої $|f(x, y)|$ мінімальне.

Позначимо

$$f(x + \Delta x, y) \equiv f^x, \quad f(x, y + \Delta y) \equiv f^y, \quad f(x + \Delta x, y + \Delta y) \equiv f^{xy},$$

тоді

$$\begin{aligned} \text{якщо } |f^x| < |f^y| \text{ і } |f^x| < |f^{xy}|, \text{ то } (y := y, x := x + \Delta x); \\ \text{якщо } |f^y| < |f^x| \text{ і } |f^y| < |f^{xy}|, \text{ то } (x := x, y := y + \Delta y); \end{aligned} \quad (5.4)$$

$$\text{якщо } |f^{xy}| < |f^x| \text{ і } |f^{xy}| < |f^y|, \text{ то } (x := x + \Delta x, y := y + \Delta y).$$

Для знаходження значень функції використаємо розклад у ряд Тейлора

$$f^x = f(x + \Delta x, y) = f(x, y) + f_x \Delta x + \frac{1}{2} f_x^2 \Delta x^2 + \frac{1}{6} f_x^3 (\Delta x)^3 + \dots$$

$$f^y = f(x, y + \Delta y) = f(x, y) + f_y \Delta y + \frac{1}{2} f_y^2 \Delta y^2 + \frac{1}{6} f_y^3 (\Delta y)^3 + \dots \quad (5.5)$$

$$f^{xy} = f(x + \Delta x, y + \Delta y) = f(x, y) + f_x \Delta x + f_y \Delta y + \frac{1}{2} (f_x^2 \Delta x^2 + 2 f_{xy} \Delta x \Delta y + f_y^2 \Delta y^2) + \dots$$

Частинні похідні в наступних точках у свою чергу визначаємо співвідношеннями

$$f_x(x + \Delta x, y) = f_x + f_x^2 \Delta x + \frac{1}{2} f_x^3 \Delta x^2 + \dots,$$

$$f_y(x + \Delta x, y) = f_y + f_{xy} \Delta x + \frac{1}{2} f_{xy}^2 \Delta x^2 + \dots,$$

$$f_x^2(x + \Delta x, y) = f_x^2 + f_x^3 \Delta x + \frac{1}{2} f_x^4 \Delta x^2 + \dots, \quad (5.6)$$

$$f_y^2(x + \Delta x, y) = f_y^2 + f_{xy}^2 \Delta x + \frac{1}{2} f_{xy}^3 \Delta x^2 + \dots,$$

$$f_{xy}(x + \Delta x, y) = f_{xy} + f_{x^2 y} \Delta x + \frac{1}{2} f_{x^3 y} \Delta x^2 + \dots$$

Оскільки $\Delta x, \Delta y$ можуть набувати значень $\pm 1, 0$, то обчислення значень функції та похідних у сусідніх точках виконуються тільки за допомогою операцій додавання та віднімання.

У випадку, коли $f(x, y)$ є алгебраїчною кривою у формулах (5.5), (5.6) наявна скінченна кількість доданків, тоді алгоритм Жордана буде точним, інакше – наближеним.

Отже, алгоритм Жордана має такий вигляд:

підібрати першу точку $P(x, y)$ і відобразити її на екрані;

початкова ініціалізація: знайти $f(x, y)$, частинні похідні $f_x(x, y), f_y(x, y)$

і т.д., визначити d ;

поки точка (x, y) не є кінцевою *виконувати*

початок

обчислити $\Delta x, \Delta y$ для наступної точки за формулами (5.3);

обчислити $|f^x|, |f^y|, |f^{xy}|$ за формулами (5.5);

за формулами (5.4) вибрати наступну точку і зобразити її на екрані;

для наступної точки обчислити частинні похідні за формулами (5.6);

кінець.

Остання точка на кривій не обов'язково обчислюється за допомогою цього алгоритму. Якщо відомі координати (x_k, y_k) останньої точки на кривій, то критерієм зупинки роботи цього алгоритму може бути умова $|x - x_k| \leq 1$ і $|y - y_k| \leq 1$, або $y = y_k$ і $|x - x_k| \leq 1$, або $x = x_k$ і $|y - y_k| \leq 1$.

Зауваження 8. Для відрізків прямих та кривих другого порядку більш ефективними є спеціалізовані растрові алгоритми, що були розглянуті вище, оскільки метод Жордана вимагає обчислення значення функції та її похідних і не враховує особливості кожної з кривих.

Контрольні питання та завдання

1. Які алгоритми називаються растровими?
2. Які методи використовуються для побудови растрових алгоритмів?
3. В чому полягають переваги та недоліки числових методів?
4. Розкрийте суть ідеї інкрементних методів.
5. Записати простіший алгоритм Брезенхема для відрізка.
6. Який вигляд має функція похибок в алгоритмі Брезенхема для відрізка?
7. Записати алгоритм Брезенхема для побудови кола.
8. Яка частина кола формується основним алгоритмом? Як формується решта частин кола?
9. Який вигляд має функція похибок в алгоритмі Брезенхема для кола?
10. Яка основна ідея інкрементного алгоритму побудови еліпса.
11. Записати інкрементний алгоритм Жордана.
12. В чому полягають переваги та недоліки методу Жордана?
13. Для яких кривих метод Жордана є точним?

Вправи і задачі для самостійного виконання

1. Реалізувати числовий метод побудови кола та еліпса.
2. Написати програму побудови дуги еліпса.
3. Розробити ефективний алгоритм для обчислення координат довільного прямокутника, якщо задано координати кінців однієї з його сторін та ширину прямокутника.
4. Розробити алгоритм побудови широкої лінії, яка задається двома точками і шириною лінії, використовуючи процедуру Line_3.
5. Ребро многокутника задано своїми вершинами. Розробити ефективний алгоритм визначення точок перетину рядків растра з цим ребром.
6. Розробити інкрементний алгоритм побудови гіперболи.
7. Розробити інкрементний алгоритм побудови параболи.
8. Розробити інкрементний алгоритм побудови лінії $y = ax^3$, $a > 0$.
9. Записати алгоритм Жордана для алгебраїчних кривих другого та третього порядків.

Тема 6. Растрові алгоритми зафарбовування і заповнення областей

6.1. Основні поняття

Область можна задавати двома способами: шляхом виведення ліній контуру (границі області) (рис. 6.1, *а*) або зафарбувавши всі пікселі області (рис. 6.1, *б*).



Рис. 6.1. Способи задання областей

Якщо область задається пікселями, що лежать усередині області, то всі пікселі з області мають одне й теж значення *old* і жоден піксель із границі такої області не має значення *old*. Такі області називаються *внутрішньо заданими*. Алгоритми, які працюють з такими областями так, щоб пікселі з атрибутом *old* перевести в пікселі зі значенням *new*, називаються *внутрішньо заповнюючими алгоритмами*.

Області, що задаються своїми замкнутими контурами (границями), називаються *гранично визначеними*. Пікселі, що належать границі, мають значення *bound*, а коди пікселів внутрішньої частини області відмінні від *bound*. Алгоритми заповнення таких областей називаються *гранично заповнюючими алгоритмами*. При цьому пікселі області потрібно зафарбувати в колір *new*, а колір контуру не повинен змінитися.

Користувач може інтерактивно визначати область, задавши границю із пікселів, що прилягають один до одного. Ці області потрібно зафарбувати кольором *new*. Після вибору області для рекурсивного алгоритму зафарбовування користувач повинен вказати ще довільну затравочну точку, що належить області. Далі в цьому алгоритмі зафарбовування області потрібно прочитати сусідні пікселі, визначити, чи вони належать області і не зафарбовані, якщо так, то зафарбувати їх. При цьому виникає питання, які пікселі (x_1, y_1) , (x_2, y_2) можна вважати сусідніми.

Важливим поняттям растрових областей є їх зв'язність – можливість з'єднати два пікселі області растровою лінією, тобто послідовним набором пікселів, що належать цій області. Використовуючи зв'язність можна, рухаючись від точки затравки, досягти і зафарбувати всі пікселі області.

За способом доступу до сусідніх пікселів області поділяються на 4-зв'язні та 8-зв'язні.

4-зв'язними називають ті області, в яких кожен піксель області може бути досягнутий з іншого пікселя області за допомогою комбінацій переміщень на один піксель у чотирьох напрямках: горизонтальних (вліво, вправо), або вертикальних (вверх, вниз), тобто якщо координати сусідніх пікселів задовольняють умову

$$|x_1 - x_2| + |y_1 - y_2| \leq 1.$$

Такі пікселі називають ще околom фон Неймана.

У 8-зв'язних областях до цих напрямків додаються ще і діагональні, тобто 8-зв'язними називаються області, в яких кожен піксель може бути досягнутий з іншого пікселя цієї ж області за допомогою послідовності переміщень на один піксель у восьми напрямках (рис. 6.2). У цьому випадку сусідніми вважаються пікселі, якщо їх координати відрізняються відповідно не більше ніж на одиницю, тобто

$$|x_1 - x_2| \leq 1 \text{ і } |y_1 - y_2| \leq 1.$$

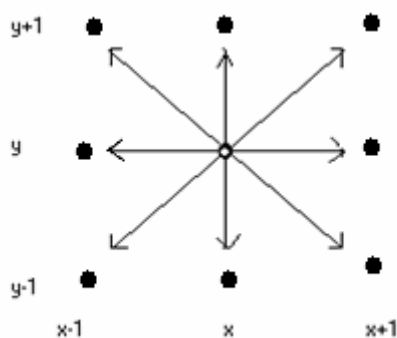


Рис. 6.2. Сусідні пікселі для 8-зв'язної області (окол Мура)

Із наведених означень випливає, що довільних два 4-зв'язних пікселі є 8-зв'язними, але не навпаки. Тому, в загальному випадку 4-зв'язну область можна заповнити як 4-зв'язними так і 8-зв'язними алгоритмами, але не навпаки, тобто 8-зв'язну область не завжди можна заповнити 4-зв'язним алгоритмом. Розглянемо два квадрати зображені на рис. 6.3. За алгоритмом заповнення для 4-зв'язних областей буде заповнено лише один квадрат, а алгоритм для 8-зв'язних областей заповнить два квадрати.

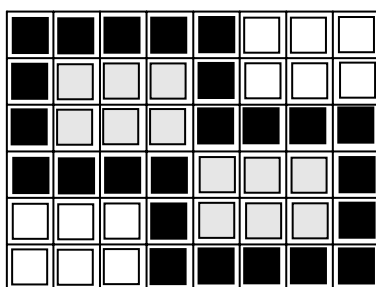


Рис. 6.3. 8-зв'язна область

Методи заповнення областей можна поділити на дві групи: методи заповнення з затравочною точкою та методи растрової розгортки.

У методах заповнення з точкою затравки заповнення починається з деякої точки, що знаходиться всередині замкнутого контуру (затравочна точка). Далі відшукуються сусідні з нею точки, які розміщені всередині контуру. Якщо така точка знайдена, то вона стає новою затравочною точкою і для неї рекурсивно ведеться пошук нових точок. Подібні алгоритми використовуються тільки для растрових пристроїв.

У методах растрової розгортки в порядку сканування рядків визначають чи точка лежить всередині контуру і якщо так, то її зафарбовують.

6.2. Рекурсивні алгоритми зафарбовування областей

Наведемо найпростіший рекурсивний алгоритм зафарбовування 4-зв'язної гранично заданої області [34].

Pixel_Fill (x, y, bound, new);

// (x,y) – координати початкової точки, з якої почнеться зафарбовування, *bound* – колір границі, *new* – новий колір області
якщо точка (x, y) не належить контуру і не зафарбована в колір *new*, то початок

Зафарбувати точку (x, y) у колір *new*;

Pixel_Fill (x + 1, y, *bound*, *new*);

Pixel_Fill (x, y + 1, *bound*, *new*);

Pixel_Fill (x, y – 1, *bound*, *new*);

Pixel_Fill (x – 1, y, *bound*, *new*);

кінець.

Аналогічний алгоритм маємо для внутрішньо заданої області [34].

Flood_Fill_4 (x, y, old, new);

//(x,y) – координати затравочної точки, *old* – значення атрибуту пікселів, що задають область, *new* – нове значення атрибуту пікселя
якщо *getpixel* (x, y) = *old* то початок

Зафарбувати точку (x, y) у колір *new*;

Flood_Fill_4 (x + 1, y, *old*, *new*);

Flood_Fill_4 (x – 1, y, *old*, *new*);

Flood_Fill_4 (x, y + 1, *old*, *new*);

Flood_Fill_4 (x, y – 1, *old*, *new*);

кінець.

Рекурсивні алгоритми досить прості. Вони абсолютно коректно зафарбовують найскладніші області, але неефективні по тій причині, що функції зафарбовування викликаються до вже зафарбованих пікселів і,

крім цього, вимагають великого стеку через значну глибину рекурсії. Практика показує, що ці алгоритми неприйнятні для областей із тисячею і більше пікселів, оскільки відбувається переповнення стеку.

Тому для розв'язання задачі зафарбування перевагу надають алгоритмам, які обробляють цілі групи пікселів.

6.3. Пострічковий алгоритм зафарбовування із затравкою

Розглянемо версію одного з найпопулярніших алгоритмів зафарбування з затравкою. Ідея цього алгоритму полягає в тому, що для заданої точки (x, y) з області визначається і заповнюється максимальний відрізок $[xl, xr]$, що лежить усередині області і містить цю точку. Після цього перевіряються відрізки області, що лежать вище і нижче знайденого відрізка. Якщо такі пікселі знаходяться, то рекурсивно викликається функція для їх обробки. Розглянемо алгоритм більш детально.

На кожній горизонталі пікселі, які потрібно зафарбувати, складають горизонтальні інтервали, що лежать усередині області. Ці інтервали відокремлені один від одного інтервалами з пікселів, які належать межі або зовнішності області.

Якщо набір пікселів утворює зв'язний інтервал, який лежить всередині області, то пікселі над і під цим інтервалом є або межовими, або лежать усередині області. Останні можуть служити затравочними пікселями для рядків, що лежать вище та нижче від активного рядка.

Нехай (x, y) – координати початкового пікселя, тоді пострічковий алгоритм має такий вигляд:

Line_Fill (x,y);

початок

Знайти крайню ліву точку xl інтервалу;

Знайти крайню праву точку xr інтервалу;

Заповнити інтервал від xl до xr ;

Для всіх x від xl до xr виконати

*якщо точка $(x, y + 1)$ не належить межі і не зафарбована,
то $Line_Fill(x, y + 1)$;*

Для всіх x від xl до xr виконати

*якщо точка $(x, y - 1)$ не належить межі і не зафарбована,
то $Line_Fill(x, y - 1)$*

кінець.

Цей алгоритм також є рекурсивним, але оскільки функція $Line_Fill$ викликається для лінії в цілому, а не для кожного пікселя, то це зменшує кількість вкладених викликів, а отже, зменшуються витрати стекової пам'яті. Алгоритм ефективно працює і для областей із дірками (рис. 6.4).

Зауваження 1. Для зафарбовування простіших фігур (прямокутник, еліпс) можна використати алгоритми виведення контуру, які розглядалися вище. У процесі виконання цих алгоритмів послідовно обчислюються координати пікселів контуру. Для зафарбовування необхідно виводити горизонталі, які з'єднують пари точок на контурі, що розміщуються симетрично відносно вертикальної осі (рис. 6.5).

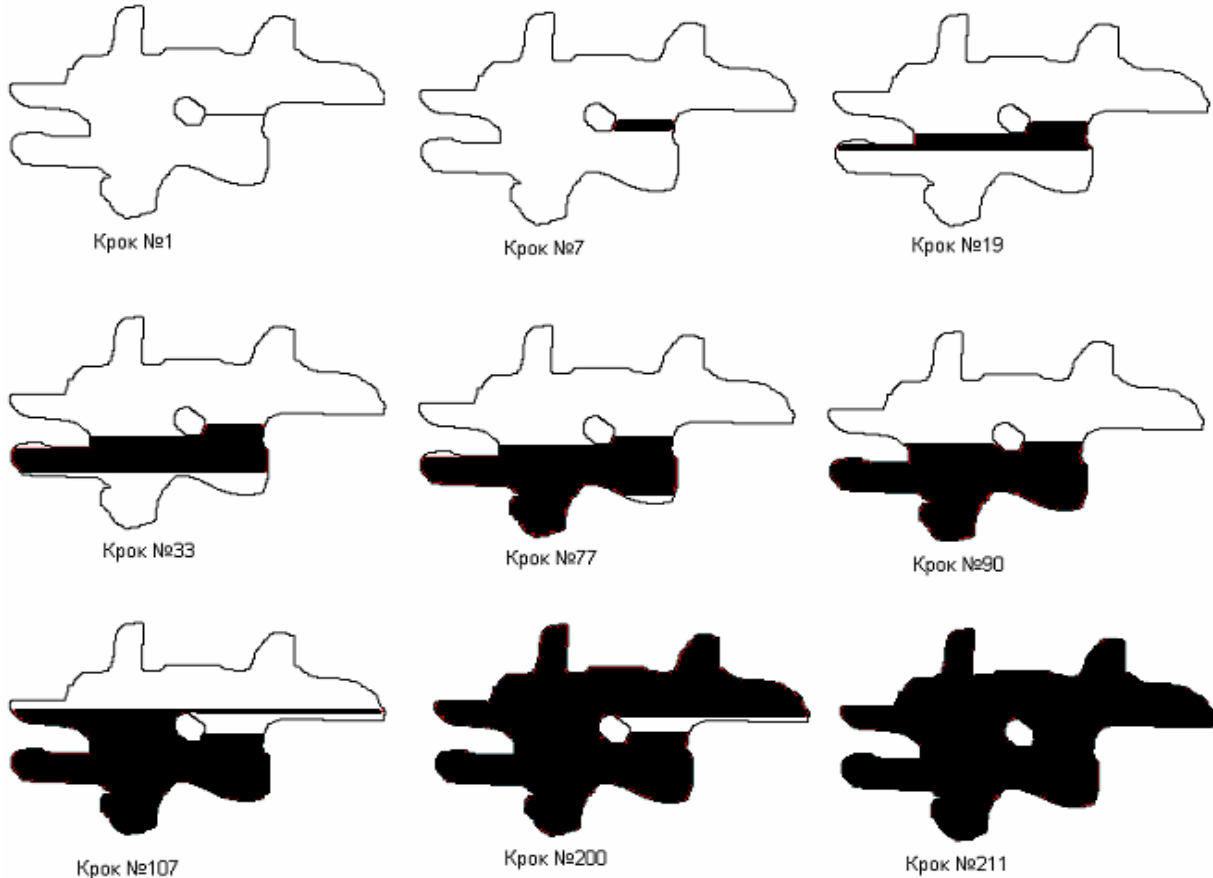


Рис. 6.4. Пострічковий алгоритм зафарбовування областей

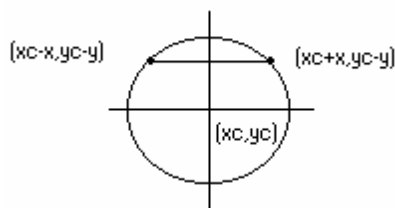


Рис. 6.5. Зафарбовування круга

6.4. Алгоритм зафарбовування області за критерієм парності

Опишемо основну ідею алгоритму. В циклі по y проводяться горизонтальні прямі. Точкам перетину дуги контуру з горизонтальною прямою ставиться у відповідність певна структура даних, що визначає характер перетину дуги з горизонтальною лінією. Ця структура даних визначає чи точки горизонталі є внутрішніми чи зовнішніми по відношенню до розглядуваної області. Так стає відомо, які пікселі потрібно зафарбовувати.

Введемо в розгляд змінні *вище*, *нижче*, які набуватимуть значень, що описують характер перетину горизонтальної прямої з дугою контуру. Точки контуру, що відповідають локальним екстремумам контуру, матимуть порядки (0, 2) або (2, 0) (рис. 6.6, а, б).

Якщо контур не містить кратних пікселів і точка контура не є екстремумом, то такій точці відповідає структура даних (1, 1) (рис. 6.6, в).

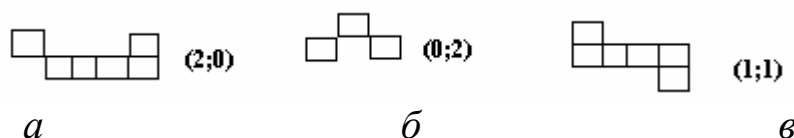


Рис. 6.6. Структури даних

Поява ознаки помилки при виконанні алгоритму сигналізує про структури даних, які відрізняються від допустимих пар значень (1, 1), (2, 0), (0, 2).

Для визначення порядків граничних точок використовується процедура *Link*. Уведемо в розгляд лічильник l , значення якого вказує на кількість перетинів контуру області з розглядуваною горизонтальною прямою. Крім цього для зменшення обчислювальних затрат визначимо прямокутну оболонку $(x_{min}, y_{min}) - (x_{max}, y_{max})$, що містить нашу область.

Алгоритм заповнення області за критерієм парності має вигляд [22].

Для кожного y від $y = y_{min}$ до $y = y_{max}$ виконати

початок

Встановити значення лічильника в 0, $l=0$;

Присвоїти x значення лівого краю сітки $x = x_{min}$;

поки $x \leq x_{max}$ виконувати

початок

якщо (x,y) не належить контуру, то

початок

якщо значення лічильника l непарне, то

Зафарбувати (x,y) ;

Збільшити x : $x:=x+1$

кінець

інакше

початок

$Link(x,y, вище, нижче)$;

якщо значення $вище, нижче = 1$, то збільшити лічильник $l:=l+1$;

якщо $вище + нижче \neq 2$ то ознака помилки

кінець

кінець

кінець.

Функцію Link визначимо наступним чином.

Link (x,y, вище, нижче)

// x,y – вхідна інформація: координати пікселя, що належить контуру
// вище, нижче – вихідна інформація (значення верхнього та нижнього порядків точки контуру)

початок

Встановити значення вище, нижче в нуль;

якщо $(x - 1, y + 1)$ належить контуру, то вище: = вище +1;

якщо $(x - 1, y - 1)$ належить контуру, то нижче: = нижче +1;

поки (x, y) належить контуру виконувати

початок

якщо $(x, y + 1)$ належить контуру, а $(x - 1, y + 1)$ не належить контуру,
то вище: = вище +1;

якщо $(x, y - 1)$ належить контуру, а $(x - 1, y - 1)$ не належить контуру,
то нижче: = нижче +1;

збільшити x

кінець

якщо $(x - 1, y + 1)$ не належить контуру, а $(x, y + 1)$ належить контуру,
то вище = вище + 1;

якщо $(x - 1, y - 1)$ не належить контуру, а $(x, y - 1)$ належить контуру,
то нижче = нижче + 1

кінець.

Зауваження 2. Наведений алгоритм спрощується для полігонів. Для цього замість процедури Link потрібно визначити характер перетину горизонтальних ліній з ребрами полігону. Якщо точка перетину є вершиною локального екстремуму, то зараховуємо дві точки перетину, інакше – одну.

6.5. Зафарбовування полігонів. YX-алгоритм

Адаптуємо наведені вище алгоритми на випадок, коли область є многокутником (полігоном). Основна ідея алгоритму – зафарбовування полігона відрізками горизонталей.

Нехай контур полігона задається множиною координат вершин $P_i(x_i, y_i)$, $i = 0, 1, \dots, n$, які послідовно з'єднані відрізками прямих.

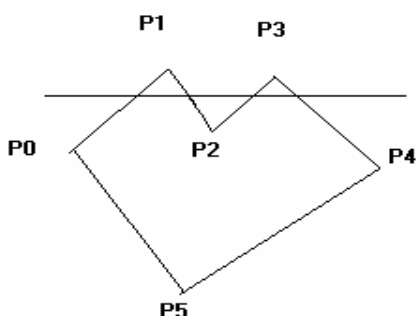


Рис. 6.7. YX-алгоритм зафарбовування полігона

В алгоритмі здійснюється цикл по y , на кожному кроці якого знаходяться точки перетину відрізків контуру полігона (ребер) з відповідними горизонталями (рис. 6.7). Цей алгоритм називається YX-алгоритмом і має такий вигляд:

початок

Знайти $\min (y_i) = y_{\min}$ і $\max (y_i) = y_{\max}$ серед всіх вершин P_i ;

Для y від $y = y_{\min}$ до $y = y_{\max}$ виконувати

початок

Знайти точки перетину усіх відрізків контуру з горизонталлю y ;

Координати точок перетину записати в масив X ;

Упорядкувати масив X по зростанню координат x_i ;

Вивести горизонтальні відрізки з координатами кінців

$(x_0, y) - (x_1, y)$; $(x_2, y) - (x_3, y)$; . . . ; $(x_{2k}, y) - (x_{2k+1}, y)$

кінець

кінець.

Довільна пряма лінія перетинає замкнений контур парну кількість разів, тобто в масив X записується парна кількість точок перетину. Опуклі фігури з довільною прямою завжди мають дві точки перетину.

При знаходженні точок перетину горизонталі з контуром необхідно брати до уваги особливі точки (вершини полігона). Якщо горизонталь має координату y , що збігається з y_i вершини P_i , то необхідно аналізувати, як горизонталь проходить через вершину P_i . Якщо вершина P_i є локальним максимумом або мінімумом, як, наприклад, P_1, P_2, P_3, P_5 (рис. 6.7), то вершина P_i не записується в масив X або записується 2 рази. Якщо горизонталь перетинає вершину P_i так, що $(y_{i-1} - y_i) * (y_{i+1} - y_i) < 0$ (наприклад, як вершини P_0, P_4 (рис. 6.7)), то в масив X ця вершина записується один раз. Якщо горизонталь зустрічається з горизонтальним відрізком полігона $P_i P_{i+1}$, то необхідно аналізувати добуток $(y_{i-1} - y_i) * (y_{i+2} - y_i)$.

Цей алгоритм можна оптимізувати, беручи до уваги той факт, що кожна горизонталь у більшості випадків перетинає не всі ребра, а лише невелику кількість ребер контуру. Тому, якщо зробити попередній відбір ребер, що перетинаються з горизонталлю, то можна досягнути зменшення кількості обчислень в алгоритмі.

Наприклад, розділимо діапазон $[y_{\min}, y_{\max}]$ навпіл точкою y_{cp} . Якщо для діапазону $[y_{\min}, y_{cp}]$ скласти список вершин (ребер), для яких y -координата належить цьому діапазону, то для горизонталей з $y \in [y_{\min}, y_{cp}]$ будемо перевіряти перетин тільки із цим списком ребер (їх буде приблизно у 2 рази менше). Аналогічно для діапазону $[y_{cp}, y_{\max}]$ також складаємо список ребер (їх теж буде приблизно вдвічі менше).

Контур можна ділити не навпіл, а на менші частини. Такий спосіб підвищення швидкодії ефективний для великої кількості вершин.

Наведені вище алгоритми можуть бути використані не тільки для зафарбовування фігур, а й для розв'язування інших задач, наприклад, для заповнення фігур, знаходження площі фігури, центру ваги та ін.

6.6. Заповнення фігур. Текстури

Задача заповнення областей полягає в тому, щоб дану область зафарбувати не суцільно, а тільки її частини (можливо, різними кольорами).

При виведенні зображення фігур можуть використовуватись різні стилі заповнення. Для визначення стилю заповнення фігур, відмінного від суцільного зафарбовування, використовують поняття текстури, хоча воно більшою мірою застосовується для тривимірних об'єктів. *Текстура* – це стиль заповнення, що імітує складну поверхню.

Для заповнення фігур певним стилем використовують ті ж самі алгоритми, що і при зафарбовуванні фігур. Суцільне зафарбовування означає, що колір кожного пікселя області однаковий ($color = const$), тобто алгоритми зафарбовування фігур обов'язково містять оператор

виведення пікселя постійного кольору $color$ із координатами (x, y) .

А для того, щоб фігуру заповнити певним візерунком, необхідно змінювати колір пікселів заповнення, тобто в алгоритмі заповнення фігури перед оператором виведення пікселя (x, y) кольору $color$ потрібно поставити оператор $color = f(x, y)$, який буде визначати колір $color$ для кожного пікселя зокрема, а отже, алгоритми заповнення будуть містити оператори: $color = f(x, y)$; *виведення пікселя (x, y) кольору $color$.*

Функція $f(x, y)$, аргументами якої є координати поточного пікселя, визначає стиль заповнення. Якщо значення функції $f(x, y)$ генерувати випадковим чином, тобто значення $color = random()$, то можна створити ілюзію матової поверхні (рис. 6.8), а якщо за $f(x, y)$ взяти функцію

$$f(x, y) = \begin{cases} C_w, & \text{якщо } (x + y) \bmod S < T; \\ C_f, & \text{для решти } x, y; \end{cases},$$

то можна заповнити область широкими штриховими лініями (рис. 6.8).

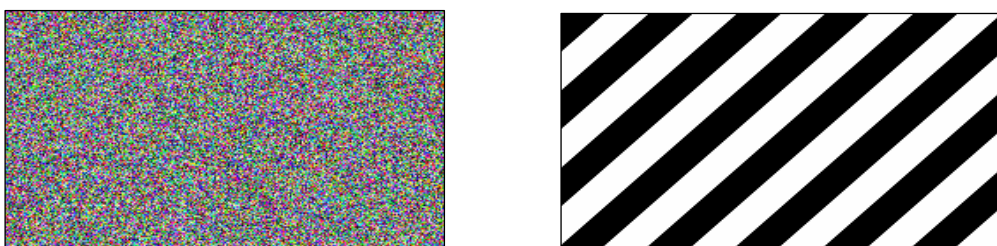


Рис. 6.8. Матова та штрихова фігури

Тут параметр T задає товщину штрихів, S – період нанесення штрихів, $C_{ш}$ – колір штрихових ліній, C_{ϕ} – колір фону. Якщо не задавати колір фону, то можна створити ілюзію напівпрозорої фігури.

Часто при заповненні фігур використовується копіювання невеликих растрових зразків-візерунків (текселів) на всю область фігури. Нехай маємо растр із $m * n$ пікселів, що складає зразок текстури. При цьому координати пікселів структури x_T, y_T можуть набувати значень $x_T = 0, 1, \dots, m - 1, y_T = 0, 1, \dots, n - 1$. Заповнення фігури можна за безпечити шляхом циклічного копіювання фрагмента зразка всередині області заповнення фігури.

Для цього потрібно оператор $color = f(x, y)$ замінити такою послідовністю операцій:

- для координат (x, y) пікселя заповнення обчислити відповідні їм координати (x_T, y_T) зразка заповнення;
- за координатами (x_T, y_T) визначити колір $color$ пікселя зразка текстури;
- вивести піксель з координатами (x, y) і з кольором $color$.

Обчислення координат (x_T, y_T) через координати (x, y) можна здійснити за формулами

$$x_T = x \bmod m, \quad y_T = y \bmod n,$$

де m, n – розміри растра-зразка, \bmod – функція, що визначає залишок від ділення. Якщо m та n є степенем 2, тоді функцію \bmod можна замінити більш швидкодіючими функціями. Приклад заповнення області текселями зображено на рис. 6.9.

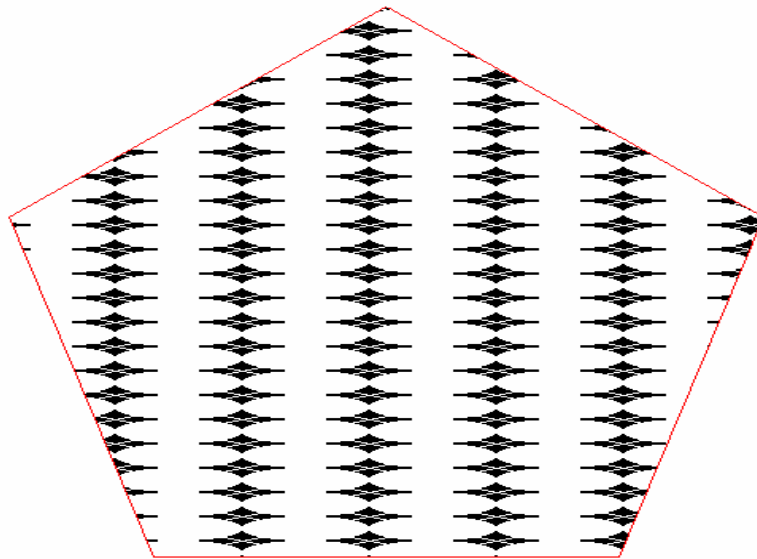


Рис. 6.9. Заповнення області деякими зразками

6.7. Обробка растрових зображень

Обробка растрових зображень передбачає перетворення растрових даних. Існує багато різних варіантів обробки растрових зображень. Можна виділити три основних класи алгоритмів обробки растрових даних [7]:

- точкові алгоритми, в яких значення пікселів змінюються на основі значень самих пікселів та їх координат;
- просторові алгоритми, в яких значення пікселів змінюється на основі вихідних значень самих пікселів і пікселів навколо них;
- алгоритми геометричних перетворень, коли розміщення пікселів в зображенні змінюється на основі геометричних перетворень.

Для кожного з цих класів розроблено цілий ряд алгоритмів, причому застосування цих алгоритмів некомутативне, тому важливим є порядок їх застосування.

Розглянемо деякі поняття, що лежать в основі цих алгоритмів. Нехай a_{ij} елемент зображення, тобто піксель з координатами (i, j) , тоді $A = \{ a_{ij} \}$, $i = 0, 1, \dots, m, j = 0, 1, \dots, n$ растрове зображення, тобто матриця пікселів.

Якщо $a_{ij} \in \{0, 1\}$, то растрове зображення називається бінарним і складається тільки з чорних та білих пікселів.

Якщо $a_{ij} \in \{0, 1, \dots, N - 1\}$, то растрове зображення називається напівтоновим і кожний піксель може приймати N відтінків сірого (градацій яскравості).

Якщо $a_{ij} \in \left\{ \begin{pmatrix} x_{ij}^1 \\ x_{ij}^2 \\ x_{ij}^3 \end{pmatrix} \right\}$, де $x_{ij}^k \in \{0, 1, \dots, N - 1\}$, то зображення називається кольоровим.

Колір кожного пікселя визначається координатами (x^1, x^2, x^3) в просторі кольорів.

Розглянемо алгоритмічні основи обробки бінарних і напівтонових зображень.

Точкові алгоритми. Точкові алгоритми прості і найбільш часто застосовуються в алгоритмах обробки зображень. Вони застосовуються для бінарних і напівтонових зображень. В цих алгоритмах для зміни яскравості пікселя в зображенні використовується тільки вихідна яскравість цього пікселя, інколи координати цього пікселя; ніякі інші значення не використовуються.

Нові значення яскравості обчислюються на основі деякого алгоритму. Точкові алгоритми сканують зображення піксель за пікселем, здійснюючи перетворення точок зображення. Якщо перетворення залежить тільки від яскравості пікселів, то процес обробки зображень краще реалі-

зувати на основі таблиць перетворення. Якщо перетворення в точкових алгоритмах враховує і розміщення пікселів, то воно задається формулою. Точкові алгоритми не можуть модифікувати деталі зображення.

Найпростіші алгоритми цього класу це алгоритми побудови негативних зображень, або алгоритми інверсії яскравості пікселів. Негативне зображення одержується шляхом віднімання вихідного значення яскравості $f(x, y)$ кожного пікселя зображення від максимально можливого значення яскравості f_{max} :

$$g(x, y) = f_{max} - f(x, y),$$

де $g(x, y)$ – нове значення яскравості пікселя (x, y) . Негативні зображення необхідні для виділення яскравих частин зображення, оскільки людське око краще сприймає деталі в темній області зображення, ніж в світлій.

Друга важлива група алгоритмів цього класу це алгоритми бінаризації – напівтонових зображень, тобто перетворення їх в чорно-біле (бінарне) зображення. Ці алгоритми базуються на пороговій обробці напівтонових зображень шляхом розділення всіх пікселів зображення на два класи за ознакою яскравості: пікселі об'єкта і фону. Формула перетворення яскравості в алгоритмах бінаризації має вигляд

$$g(x, y) = \begin{cases} f_{max}, & \text{якщо } f(x, y) \geq f_0; \\ f_b, & \text{якщо } f(x, y) < f_0, \end{cases}$$

де f_0 – деяке порогове значення яскравості (рис. 6.10).

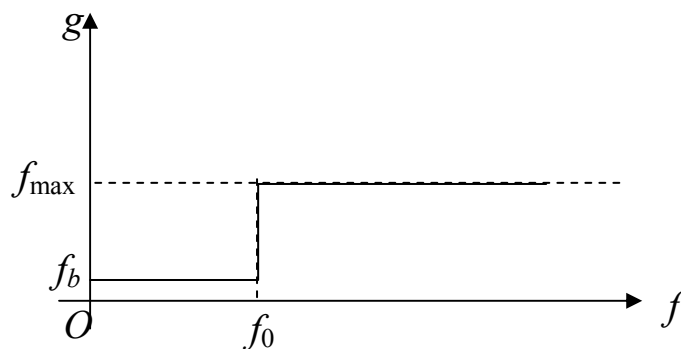


Рис. 6.10.

При цьому важливо правильно вибрати порогове значення f_0 , щоб не втратити корисну інформацію. Особливо це суттєво, коли зображення містить деякий шум. Такі перетворення виконують для того щоб в зображенні залишити тільки ту інформацію, яка необхідна для розв'язування задач, наприклад для знаходження контурів об'єкта.

За допомогою точкових алгоритмів можна розв'язати задачу збільшення контрастності зображення. Зображення з низьким контрастом можуть бути як надто світлими так і надто темними. Зображення з високим контрастом мають як темні так і світлі області, тобто використовують

весь діапазон яскравості. Зображення з низьким контрастом складається з тонів обмеженого діапазону яскравості. Задача збільшення контрасту полягає в розтягуванні діапазону яскравості вихідного зображення на всю шкалу $[f_b, f_{max}]$.

Цю задачу можна розв'язати за допомогою точкового перетворення яскравості за формулою

$$g(x, y) = af(x, y) + b$$

де a, b підібрані коефіцієнти.

Просторові алгоритми. В просторових алгоритмах використовується інформація про групи пікселів, на відміну від точкових алгоритмів у яких використовується інформація тільки про один піксель.

Група пікселів зображення, які приймають участь в просторових алгоритмах називається *областю примикання*. Область примикання визначається матрицею значень яскравості пікселів з непарною кількістю рядків і стовпців. Пікселі, в яких старе значення яскравості замінюється на нове, розміщені в центрі області притягання.

Більшість просторових алгоритмів використовують фільтри. Фільтр зручно задавати матрицею чисел малого розміру (3×3 , або 5×5), яку називають *ядром згортки*. Розміри, структура ядра згортки і значення коефіцієнтів, що містяться в ядрі визначають тип просторового алгоритму, дозволяють вплинути на значення пікселя зображення і одержати різні спецефекти.

Більший розмір матриці підвищує гнучкість процесів згортки, однак виникають труднощі з примежовими пікселями, оскільки маска згортки для них виходить за межі зображення. Тому на практиці обмежуються невеликими розмірами матриці, та додатково опрацьовують пікселі на краях зображення.

Щоб перетворити один піксель в зображенні, значення його яскравості множиться на число в центрі ядра, а яскравості пікселів, що оточують центральний піксель множаться на відповідні коефіцієнти ядра. Потім ці всі добутки сумуються і в результаті одержується нове значення яскравості для центрального пікселя.

Цей процес повторюється для кожного пікселя зображення (так фільтрується зображення). Коефіцієнти ядра визначають результат процесу фільтрування. Якщо сума коефіцієнтів більша за одиницю, то яскравість збільшується; якщо сума менша за одиницю, то яскравість зменшується.

Найбільш широко розповсюджені просторові алгоритми обробки растрових зображень – це алгоритми розмивання, збільшення чіткості, тиснення і акварельний ефект.

В алгоритмі розмивання в зображенні пом'ягшуються різкі границі за рахунок перерозподілу яскравості, тобто шляхом усереднення швидких змін яскравості (рис. 6.11, б). Ядро розмивання складають коефіцієнти менші за одиницю, а їх сума дорівнює одиниці. Це означає, що в результаті фільтрації кожний піксель вбирає інформацію від сусідніх пікселів, але повна яскравість зображення залишається незмінною. Результуюче зображення таким чином буде більш розмитим порівняно з оригіналом. Ядро розмивання, наприклад, має вигляд

$$\begin{pmatrix} 0,05 & 0,05 & 0,05 \\ 0,05 & 0,6 & 0,05 \\ 0,05 & 0,05 & 0,05 \end{pmatrix} \text{ або } \begin{pmatrix} 0,1 & 0,1 & 0,1 \\ 0,1 & 0,2 & 0,1 \\ 0,1 & 0,1 & 0,1 \end{pmatrix}.$$

Ступінь розмивання можна змінити

- збільшенням розміру ядра;
- підбором коефіцієнтів ядра зі зменшенням впливу центрального коефіцієнта;
- проведенням багатокрокової фільтрації з ядром розмивання.

Зауважимо, що у випадку кольорових зображень ядро розмивання застосовується до червоної, зеленої і синьої компонент кольору кожного пікселя зображення.

В алгоритмах збільшення чіткості застосовується інше ядро, оскільки необхідно збільшити чіткість зображення, тобто підкреслити різницю між яскравостями сусідніх пікселів і виділити непомітні деталі (рис.6.11, в).

В ядрі чіткості центральний коефіцієнт більший за одиницю, а решту елементів ядра від'ємні числа, сума яких на одиницю менша, ніж центральний коефіцієнт. Це означає, що при обробці зображення з великими змінами яскравості нове значення яскравості центрального пікселя різко збільшується, тобто великі зміни яскравості збільшуються, а області постійної яскравості залишаються незмінними. Результуюче зображення одержується більш чітким, ніж оригінал. При повторній обробці чіткість знову може бути збільшена, але при цьому ніякі нові деталі з нічого не з'являться.

При обробці пікселів в зображенні застосовуються такі ядра чіткості (високочастотні маски):

$$\begin{pmatrix} -0,1 & -0,1 & -0,1 \\ -0,1 & 1,8 & -0,1 \\ -0,1 & -0,1 & -0,1 \end{pmatrix}, \begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}, \begin{pmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{pmatrix}.$$

Знову, як і раніше у випадку кольорового зображення червона, зелена і синя складові обробляються маскою окремо, а потім об'єднуються, щоб сформувати 24-бітне значення кольору.

Алгоритми тиснення перетворюють зображення так, що об'єкти сцени виглядають видавленими на деякій поверхні (рис. 6.11, з). Тиснення виконується за допомогою ядра згортки так, як і в попередніх алгоритмах. Кожен піксель зображення обробляється ядром тиснення розміру 3×3 .

Наведемо приклади ядра тиснення

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}, \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}.$$

Зауважимо, що сума коефіцієнтів в ядрі тиснення дорівнює нулю. Це означає, що тим пікселям, які не знаходяться на границях переходу від одного кольору до іншого присвоюються нульові значення, а тим що знаходяться на таких границях ненульові значення. Тобто, якщо всі дев'ять пікселів, що знаходяться в області матриці тиснення мають однакові яскравості, то значення центрального пікселя після перетворення стане рівним нулю (чорний колір).

Після обробки пікселя ядром тиснення до одержаної яскравості (у випадку кольорового зображення до кожної складової R, G, B) додають число 128. Суми, що перевищують 255 заокруглюються до 255.

Аналогічно конструюються маски для акварельних зображень.

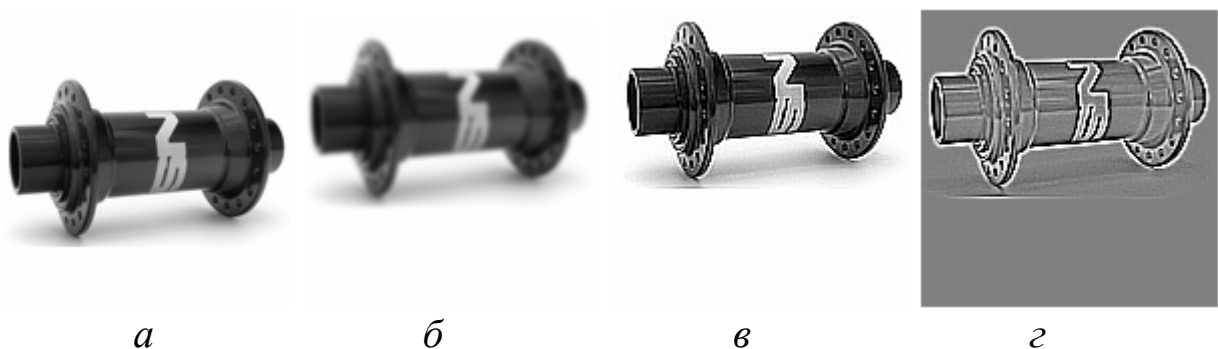


Рис. 6.11. Обробка зображень: а – вихідне зображення, б – розмивання зображення, в – збільшення чіткості, з – тиснення

До просторових алгоритмів відносяться ще *медіанні фільтри*, за допомогою яких можна відфільтрувати шумові ефекти. Ці алгоритми не працюють за принципом згортки, тобто нове значення яскравості пікселя не обчислюється шляхом матричного множення, а замість цього всі пікселі в області примикання сортуються в зростаючому порядку за яскравістю пікселів і вибирається середнє значення яскравості для нового значення розглядуваного пікселя. Наприклад, потрібно змінити значення яскравості центрального пікселя в такому вікні

$$\begin{pmatrix} 43 & 75 & 50 \\ 57 & 126 & 50 \\ 24 & 83 & 50 \end{pmatrix}.$$

Відсортувавши значення пікселів, одержимо послідовність 24, 43, 50, 50, 50, 57, 75, 83, 126 в якій 50 є середнім значенням. В результаті середнє значення яскравості 50 замінить 126 у вихідному зображенні, тобто випадковий шум, що міститься в зображенні буде усунено.

Алгоритми геометричних перетворень. Алгоритми геометричних перетворень растрових зображень змінюють місце розміщення і/або структуру пікселів зображення на основі геометричних перетворень. Геометричні перетворення не завжди змінюють яскравість елементів зображення, але вони завжди змінюють розміщення пікселів зображення, тобто значення яскравості пікселів займають нову позицію.

Геометричні перетворення растрових даних широко застосовуються, наприклад при розпізнаванні образів, накладанні текстури, забезпеченні різних ефектів у зображеннях. Геометричні перетворення растрових даних – це переміщення, масштабування, зсув і поворот зображення. Ці перетворення будуть вивчатимуться далі.

Контрольні питання та завдання

1. Які є способи задання областей?
2. Які області називаються 4-зв'язними/8-зв'язними?
3. Як працюють рекурсивні алгоритми заповнення областей?
4. В чому переваги пострічкового алгоритму?
5. Запишіть і поясніть роботу процедури Line_Fill.
6. Які структури даних використовуються в алгоритмі заповнення за критерієм парності?
7. Поясніть роботу процедури Link.
8. В чому полягає основна ідея УХ-алгоритму?
9. Чим відрізняється заповнення фігур від зафарбовування?
10. Як заповнити фігуру деяким узором?
11. Як одержати кольоровий негатив графічного зображення?

Вправи і задачі для самостійного виконання

1. Знайти небажані ефекти, які можуть виникнути при заповненні 4-зв'язної області 8-зв'язним алгоритмом?
2. Записати рекурсивні алгоритми зафарбовування 8-зв'язних областей.
3. Визначте на вашому комп'ютері максимальну кількість пікселів області, яку ще вдається заповнити рекурсивними алгоритмами.
4. Знайти порядок зафарбовування простої 4-зв'язної гранично-заданої області в рекурсивному алгоритмі з затравкою.
5. Записати пострічковий алгоритм зафарбовування полігона.
6. Оптимізувати процедуру Link для полігонів.
7. Відобразити растрове зображення з прямокутної області з протилежними вершинами $(0, 0)$, (x_0, y_0) в довільну чотирикутну область з вершинами (u_0, v_0) , (u_1, v_1) , (u_2, v_2) , (u_3, v_3) і навпаки.
Вказівка. Закон перетворення прямокутної області в чотирикутну виразити квазілінійним співвідношенням виду:

$$u = a_1x + b_1xy + c_1y + d_1$$

$$v = a_2x + b_2xy + c_2y + d_2$$

8. Розробити алгоритм створення ефекту хвиль на чорно-білому зображенні.
9. Розробити алгоритм побудови контурів предметів на чорно-білому зображенні.
10. Розробити алгоритм морфологічного розширення зображень A та B . Під морфологічним розширенням розуміють операцію

$$A(+)B = \{t \in R^2 : t = a + b, a \in A, b \in B\}.$$

Тема 7. Побудова інтерполяційних та згладжуючих кривих

7.1. Основні поняття

Часто на практиці виникає задача побудови геометричного образу Q^* , який може замінити з деяким ступенем точності початковий геометричний образ Q (задача апроксимації). Тому для побудови образу Q^* необхідно вміти розв'язувати задачі інтерполяції та згладжування. У задачах інтерполяції вимагається, щоб геометричний образ Q^* проходив через усі задані вузлові (опорні) точки, а задача згладжування полягає у відновленні геометричного об'єкта, що проходить біля вузлових точок.

Задача інтерполяції на площині ставиться так: нехай у точках x_0, x_1, \dots, x_m таких, що $a \leq x_0 < x_1 < \dots < x_m \leq b$ (ці точки називаються *вузлами*) відомі значення функції $y_i = f(x_i)$, $i = 0, 1, \dots, m$, тобто на відрізку $[a, b]$ таблично задана функція $y=f(x)$

x	x_0	x_1	\dots	x_m
y	y_0	y_1	\dots	y_m

Необхідно відновити поведінку функції $f(x)$ для всіх $x \in [a, b]$ так, щоб графік цієї функції проходив через усі задані точки (x_i, y_i) , $i = 0, 1, \dots, m$.

Функція $\varphi(x)$ називається *інтерполяційною* для $f(x)$ на $[a, b]$, якщо її значення в заданих точках збігаються з наперед заданими значеннями, тобто $\varphi(x_0) = y_0$, $\varphi(x_1) = y_1$, \dots , $\varphi(x_m) = y_m$, а для решти $x \in [a, b]$ $f(x) \approx \varphi(x)$. Як функцію $\varphi(x)$ найчастіше використовують інтерполяційні многочлени $P_m(x)$ або інтерполяційні сплайни $S(x)$.

На практиці часто трапляються випадки, коли значення y_i задані з певною похибкою (наприклад, якщо y_i є результатами вимірювання деякої функції $y(x)$, що містить випадкову похибку). У цьому випадку не можна застосовувати інтерполяційний підхід, оскільки функція $\varphi(x)$ відобразить всі похибки експериментальних даних. Тому для задачі відновлення функції за експериментальними даними, щоб зменшити вплив випадковостей у результатах вимірювання, більш виправдане використання процедур згладжування.

Задача згладжування полягає у відновленні гладкої функції $\varphi(x)$, для якої $\varphi(x_i) \approx y_i$, $i = 0, 1, \dots, m$, тобто потрібно побудувати криву, що проходить поблизу опорних точок із заданим допустимим відхиленням. Така задача має багато розв'язків, однак, наклавши на функцію $\varphi(x)$ додаткові умови, можна досягти необхідної однозначності.

7.2. Поліноміальна інтерполяція

Задача інтерполяції алгебраїчними многочленами полягає в побудові многочлена $P_m(x) = a_0 + a_1x + \dots + a_mx_m$, значення якого в точках x_i

апроксимуюча функція лінійна на кожному з відрізків $[x_i, x_{i+1}]$, $i = 0, 1, \dots, m - 1$. В цьому випадку точки (x_i, y_i) послідовно з'єднуються прямолінійними відрізками (рис. 7.1).

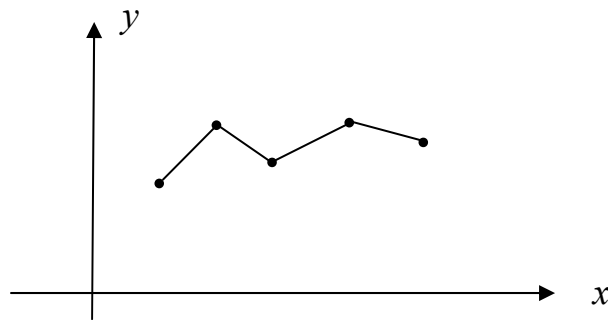


Рис. 7.1. Кусково-лінійна інтерполяція

Перевагою такою підходу є простота побудови інтерполянта. Крім цього, зміна однієї точки в масиві вимагає обчислення лише чотирьох коефіцієнтів (коефіцієнтів двох прямолінійних відрізків). Недоліком такого підходу є невисока точність побудови апроксимації та відсутність гладкості для апроксимації функції (перша похідна розривна у вузлових точках).

Уникнути цього недоліку можна шляхом побудови гладкої інтерполяційної сплайн-функції степеня p (рис. 7.2). На практиці найчастіше використовують інтерполяційні сплайни степеня три з дефектом 1 [5, 23].

Означення. Інтерполяційним кубічним сплайном $S(x)$ дефекта 1 на множині (x_i, y_i) , $i = 0, 1, \dots, m$ називається функція, яка:

1) на кожному з відрізків $[x_i, x_{i+1}]$, $i = 0, 1, \dots, m - 1$ є многочленом 3-го степеня, тобто

$$S(x) = S_3^i(x) = a_0^i + a_1^i(x - x_i) + a_2^i(x - x_i)^2 + a_3^i(x - x_i)^3, \quad x \in [x_i, x_{i+1}];$$

2) двічі неперервно диференційована на $[a, b]$ ($S \in C^2[a, b]$);

3) задовольняє інтерполяційні умови $S(x_i) = y_i$, $i = 0, 1, \dots, m$.

Оскільки на кожному відрізку $[x_i, x_{i+1}]$, $i = 0, 1, \dots, m - 1$ сплайн є многочленом 3-го степеня, то він визначається однозначно чотирма коефіцієнтами

$$a_0^i, a_1^i, a_2^i, a_3^i, \quad i = 0, 1, \dots, m - 1.$$

Всього маємо m відрізків, тому для визначення сплайна потрібно знати $4m$ коефіцієнтів.

Умова $S(x) \in C^2[a, b]$ означає неперервність $S(x)$, $S'(x)$, $S''(x)$ у внутрішніх вузлах. Таких вузлів маємо $m - 1$, а значить $3(m - 1)$ умов (рівнянь). Тому разом з 3) одержуємо всього $3(m - 1) + m + 1 = 4m - 2$ умови.

Для визначення коефіцієнтів не вистачає ще двох умов. Ці дві умови задаються у вигляді обмежень на значення сплайна та його похідних на

кінцях відрізка $[a, b]$. При побудові інтерполяційних кубічних сплайнів найчастіше використовуються такі крайові умови.

Крайові умови 1-го типу:

$$S'(a) = f'(a), S'(b) = f'(b),$$

де $f'(a)$ та $f'(b)$ задаються з певних міркувань.

Крайові умови 2-го типу:

$$S''(a) = f''(a), S''(b) = f''(b).$$

Крайові умови 3-го типу (періодичні умови):

$$S'(a) = S'(b), S''(a) = S''(b).$$

Зауваження 1. Третя похідна $S'''(x)$ у внутрішніх вузлах сітки загалом розривна (сплайн має дефект 1). Неперервність другої похідної має особливе значення, оскільки відсутній скачок прискорення. За другим законом Ньютона скачок прискорення означає скачок сили, тобто з'являється удар, який розбиває тіло, що рухається по спряженій траєкторії.

Розглянемо алгоритм побудови кубічного сплайна. Опишемо спосіб обчислення коефіцієнтів кубічного сплайна, при якому кількість величин, які необхідно знаходити, дорівнює $m + 1$, а не $4m$.

Нехай $S_3''(x_i) = M_i$, $i = 0, 1, \dots, m$. Тоді $S_3''(x)$ на кожному з відрізків $[x_i, x_{i+1}]$, $i = 0, 1, \dots, m - 1$ є многочленом першого степеня, тобто

$$S_3''(x) = M_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + M_{i+1} \frac{x - x_i}{x_{i+1} - x_i}, \quad x \in [x_i, x_{i+1}].$$

Інтегруючи $S_3''(x)$, знаходимо $S_3'(x)$ і $S_3(x)$. В результаті одержуємо

$$S_3(x) = M_{i+1} \frac{(x - x_i)^3}{6h_i} + M_i \frac{(x_{i+1} - x)^3}{6h_i} + C_1 x + C_2, \quad h_i = x_{i+1} - x_i.$$

Використовуючи умови інтерполяції

$$S_3(x_i) = y_i, \quad S_3(x_{i+1}) = y_{i+1},$$

знаходимо

$$C_1 = \frac{1}{h_i} (y_{i+1} - y_i) - \frac{M_{i+1} - M_i}{6} h_i,$$

$$C_2 = \frac{M_{i+1} h_i x_i}{6} - \frac{M_i h_i x_{i+1}}{6} - \frac{y_{i+1} x_i}{h_i} + \frac{y_i x_{i+1}}{h_i},$$

а отже,

$$S_3(x) = \frac{M_{i+1} (x - x_i)^3}{6h_i} + \frac{M_i (x_{i+1} - x)^3}{6h_i} + \left(y_i - \frac{M_i h_i^2}{6} \right) \frac{x_{i+1} - x}{h_i} +$$

$$+ \left(y_{i+1} - \frac{M_{i+1} h_i^2}{6} \right) \frac{x - x_i}{h_i}, \quad x \in [x_i, x_{i+1}]. \quad (7.1)$$

Невідомі M_i знаходимо з умов неперервності першої похідної сплайна у вузлах x_i , $i = 1, 2, \dots, m - 1$ та з крайових умов.

Для крайових умов 1-го типу для знаходження M_i одержуємо систему

$$\begin{cases} 2M_0 + M_1 = d_0, \\ \lambda_i M_{i-1} + 2M_i + \mu_i M_{i+1} = d_i, \quad i = \overline{1, m-1}, \\ M_{m-1} + 2M_m = d_m, \end{cases} \quad (7.2)$$

$$\text{де } \lambda_i = \frac{h_{i-1}}{h_{i-1} + h_i}, \quad \mu_i = \frac{h_i}{h_{i-1} + h_i}, \quad d_i = \frac{6}{h_i + h_{i-1}} \left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \right),$$

$$d_0 = \frac{6}{h_0} \left(\frac{y_1 - y_0}{h_0} - f'(a) \right), \quad d_m = \frac{6}{h_{m-1}} \left(f'(b) - \frac{y_m - y_{m-1}}{h_{m-1}} \right). \quad (7.3)$$

Оскільки $\lambda_i + \mu_i = 1$, $i = 1, 2, \dots, m - 1$, то система (7.2) має тридіагональну матрицю з переважаючою діагоналлю, а отже, ця система має єдиний розв'язок.

Якщо сітка вузлів x_i рівномірна, тобто $h_i = h$, $i = 0, 1, \dots, m - 1$, то система (7.2) має вигляд

$$\begin{cases} 2M_0 + M_1 = d_0, \\ M_{i-1} + 4M_i + M_{i+1} = \frac{6}{h^2} (y_{i+1} - 2y_i + y_{i-1}), \quad i = \overline{1, m-1}, \\ M_{m-1} + 2M_m = d_m. \end{cases}$$

Зауваження 2. Якщо у формулі (7.1) зробити заміну змінних

$$\frac{x - x_i}{h_i} = t, \quad t \in [0, 1], \quad x_i \in [x_i, x_{i+1}],$$

то на відрізку $[x_i, x_{i+1}]$ для інтерполяційного сплайна одержуємо

$$S_3(t) = y_i(1 - t) + y_{i+1}t - \frac{h_i^2}{6} t(1 - t)[(2 - t)M_i + (1 + t)M_{i+1}], \quad t \in [0, 1]. \quad (7.4)$$

Розглянемо *апроксимаційні властивості кубічного сплайна*. Точність апроксимації кубічними сплайнами залежить від гладкості функції $f(x)$. Чим вища гладкість функції $f(x)$, тим вищий порядок апроксимації.

Теорема. Нехай кубічний сплайн інтерполює у вузлах x_i , $i = 0, 1, \dots, m$ функцію $f(x) \in C^2[a, b]$, тоді справедливі такі оцінки:

$$\|f(x) - S(x)\|_C = \max_{x \in [a, b]} |f(x) - S(x)| = O(h^3),$$

$$\|f'(x) - S'(x)\|_C = O(h^2), \quad \|f''(x) - S''(x)\|_C = O(h).$$

Англійське слово *spline* перекладається як гнучка лінійка і пов'язано це з такою обставиною: бажаючи провести плавну лінію через задані точки площини, фіксують в усіх цих точках гнучку пружну лінійку; тоді

під впливом пружних сил вона набуває потрібної форми, що забезпечує мінімум потенціальної енергії. Саме функція (7.1) є математичною моделлю профіля лінійки (рис. 7.2).

Функція $S(x)$, що описує форму лінійки, між двома сусідніми опорами є многочленом 3-го степеня і двічі неперервно диференційованою на всьому відрізку $[a, b]$.

Потенціальна енергія закріпленої в $m + 1$ точці пружної лінійки характеризується величиною

$$I(f) = \int_a^b (f''(x))^2 dx,$$

де $f(x)$ – профіль лінійки.

Зазначимо, що серед всіх функцій, які проходять через опорні точки (x_i, y_i) , $i = 0, 1, \dots, m$ і належать простору $C^2[a, b]$, саме кубічний сплайн $S(x)$, що задовольняє крайові умови

$$S''(a) = S''(b) = 0,$$

доставляє екстремум (мінімум) функціоналу $I(f)$.

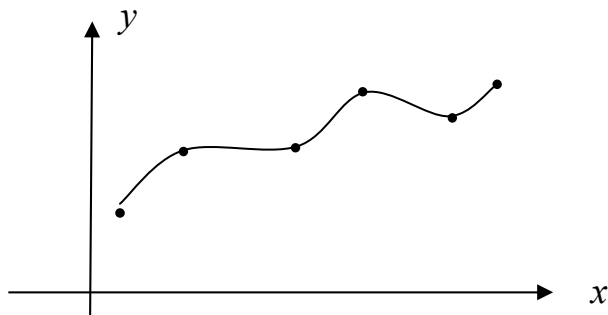


Рис. 7.2. Сплайн-функція

7.3. Згладжуючі кубічні сплайни

На практиці часто значення функції y_i вимірюються експериментально, тому вони містять похибку. А це означає, що при розв'язуванні задачі відновлення функції немає змісту використовувати інтерполяцію. Щоб зменшити ефекти випадковості у вимірах, потрібно використовувати процес згладжування. В таких задачах необхідно знайти функцію, значення якої при $x = x_i$ лише наближено дорівнює y_i й, крім того, ця функція повинна володіти певними властивостями [8, 37].

Означення. Згладжуючим кубічним сплайном $S(x)$ на $[a, b]$ називається функція, яка

- 1) на кожному відрізку $[x_i, x_{i+1}]$, $i = 0, 1, \dots, m - 1$ є многочленом 3-го степеня;
- 2) двічі неперервно диференційована на $[a, b]$;
- 3) доставляє мінімум функціоналу

$$I(f) = \int_a^b (f''(x))^2 dx + \sum_{i=0}^m p_i (f(x_i) - y_i)^2,$$

де $p_i > 0$ – задані вагові коефіцієнти, від величини яких залежить відхилення значення згладжуючого сплайна в точках x_i від значення y_i .

Якщо зменшувати деякий коефіцієнт p_k , то згладжуючий сплайн притягується точкою (x_k, y_k) . При $p_k = 0$ згладжуючий сплайн пройде через точку (x_k, y_k) . Збільшення p_k приводить до більш гладкої зміни згладжуючого сплайна.

Процедура побудови згладжуючих сплайнів складніша за побудову інтерполяційних сплайнів. Згладжуючі сплайни, як й інтерполяційні сплайни, знаходяться за формулами (7.1), але система алгебраїчних рівнянь відносно M_i вже буде п'ятидіагональною. Її розв'язування вимагає значних обчислювальних ресурсів і тому затрудняє інтерактивну роботу з апроксимаційними сплайнами.

7.4. Сплайнові криві

Вище розглядалася задача побудови кривої, що проходить через задану множину точок. Однак існує й інший клас задач, коли форма кривої повинна відповідати деяким естетичним вимогам. Наприклад, необхідно підібрати відповідний дизайн форми корабля. Крім кількісних критеріїв, у таких задачах необхідно враховувати й практичний досвід через інтерактивне втручання конструктора. Інтерполяційні методи, зокрема кубічні сплайни, незручні для інтерактивної роботи, оскільки неочевидний зв'язок між набором чисел і формою відповідної кривої. Для цієї роботи більше підходять сплайнові криві.

Задачу згладжування в загальному випадку можна сформулювати так: за заданою множиною вершин $P = \{P_0, P_1, \dots, P_m\}$ з урахуванням їх порядку необхідно побудувати гладку криву, яка б змінювалась плавно, проходила поблизу цих вершин і задовольняла певні додаткові умови.

Якщо з'єднати послідовно точки множини P , то одержуємо ламану, яку називають *опорною* або *контрольною*, а її вершини – *опорними* або *контрольними*. Ця ламана показує, як буде проходити шукана згладжуюча лінія. Вершини P_0, P_m ламаної називаються *граничними* (кінцевими), а вершини $P_i, i = 1, 2, \dots, m - 1$ – *внутрішніми*.

Розміщення точок $P_i, i = 0, 1, \dots, m$ на площині може бути довільним, деякі з них можуть навіть збігатися. Тому рівняння кривої необхідно шукати в більш загальній параметричній формі. Нагадаємо, що крива γ називається параметрично заданою, якщо декартові координати точок кривої визначаються співвідношеннями

$$x = x(t), \quad y = y(t), \quad t \in [a, b]. \quad (7.5)$$

У параметричному вигляді кожна координата точки кривої є функцією одного параметра. Значення параметра задає координату точки на кривій. Параметрична форма кривих може бути в деяких випадках зручнішою, ніж задання функції у явному вигляді. Наприклад тому, що параметрична форма дозволяє зобразити замкнені криві.

Сплайнові криві шукають у вигляді

$$R(t) = \sum_{i=0}^m a_i(t)P_i, \quad (7.6)$$

де $a_i(t)$ – деякі функціональні коефіцієнти, а

$$R(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \quad t \in [a, b].$$

Якщо кількість вершин у множині P достатньо велика, то знайти коефіцієнти $a_i(t)$, як правило, проблематично. А якщо такі коефіцієнти $a_i(t)$ знайдені, то вони можуть не володіти необхідними властивостями.

Для успішного розв'язування задачі згладжування зручно використовувати криві, які складені з елементарних фрагментів. У випадку, коли ці елементарні фрагменти будуються за єдиною схемою, такі складені криві називаються *сплайновими кривими*.

Параметричні рівняння кожного елементарного фрагмента шукаються теж у вигляді (7.6), однак для побудови фрагмента використовується лише частина вершин множини P . Відповідні коефіцієнти мають одну й ту ж природу (часто використовуються многочлени однакового степеня, раціональні дроби тощо).

На практиці для опису елементарних фрагментів використовуються многочлени невисокого степеня, найчастіше третього. Це пов'язане з тим, що многочлени нижчих порядків не забезпечують необхідної точності апроксимації кривих, а многочлени вищого степеня можуть викликати небажані коливання згладжуючої кривої. Крім цього, параметричні кубічні криві – це криві найменшого порядку, які можуть зайняти довільне положення в просторі, бо криві другого порядку визначаються трьома точками, а три точки задають площину. Многочлени 3-го степеня легко обчислюються,

Параметрична крива γ називається C^k -гладкою, якщо вектор-функція $R(t) \in C^k$ -гладкою, тобто $x(t)$, $y(t)$ мають на $[a, b]$ неперервні похідні включно до k -го порядку (в точках a , b обчислюються односторонні похідні).

Перша похідна функції $R(t)$ називається дотичним, або тангенціальним вектором параметричної кривої. Фізичний зміст дотичного вектора полягає

в тому, що його значення дорівнює швидкості зміни кривої відносно параметра t у даній точці. Друга похідна визначає прискорення.

Якщо, наприклад, камера рухається вздовж сплайнової кривої і робить знімки, то дотичний вектор задає швидкість переміщення камери вздовж кривої. В точках стику фрагментів кривої швидкість руху камери і її прискорення не повинні різко змінюватись, інакше буде порушуватись плавність знімання.

Зауважимо, що одиничний вектор дотичної до кривої γ у точці $R(t)$ обчислюється за формулою

$$T(t) = \frac{1}{|\dot{R}(t)|} \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \end{pmatrix}, t \in [a, b],$$

де $|\dot{R}(t)| = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)}$, а одиничний вектор нормалі до кривої γ в точці $R(t)$ – за формулою

$$N(t) = \frac{1}{|\dot{R}(t)|} \begin{pmatrix} -\dot{y}(t) \\ \dot{x}(t) \end{pmatrix}, t \in [a, b].$$

Існує велика кількість різних варіантів сплайнових кривих, що відрізняються за своїми властивостями. Розглянемо три основних типи сплайнових кривих – це криві Безьє, B-сплайни та інтерполяційні криві Ерміта.

7.5. Криві Безьє

7.5.1. Основні поняття

Векторні зображення складаються з контурів. Для опису контурів у програмах векторної графіки застосовують криві Безьє. Контури складаються із сегментів, обмежених вузлами. З декількох сегментів таких кривих можна скласти практично будь-яку фігуру. Такі криві розроблені математиком П'єром Безьє. Криві та поверхні Безьє були використані у 60-х роках компанією „Рено” для комп'ютерного проектування форми кузовів автомобілів. На сьогодні вони широко використовуються в комп'ютерній графіці, автоматизованих системах управління виробництвом тощо. Квадратичні криві Безьє використовуються в шрифтах TrueType.

За заданим масивом вершин $P = \{P_0, P_1, \dots, P_m\}$ крива Безьє степеня m визначається за формулою

$$R(t) = \sum_{i=0}^m B_i^m(t) P_i, t \in [0, 1], \quad (7.7)$$

де $B_i^m(t) = C_m^i t^i (1-t)^{m-i}$ – поліноми Берштейна, $C_m^i = \frac{m!}{i!(m-i)!}$, $P_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$.

У скалярній формі (7.7) має вигляд

$$x(t) = \sum_{i=0}^m C_m^i t^i (1-t)^{m-i} x_i,$$

$$y(t) = \sum_{i=0}^m C_m^i t^i (1-t)^{m-i} y_i.$$

Точки P_0, P_m називаються *кінцевими*, а точки P_1, \dots, P_{m-1} – *контрольними*, Ламана $P_0 P_1 \dots P_m$ називається *контрольною (опорною)*.

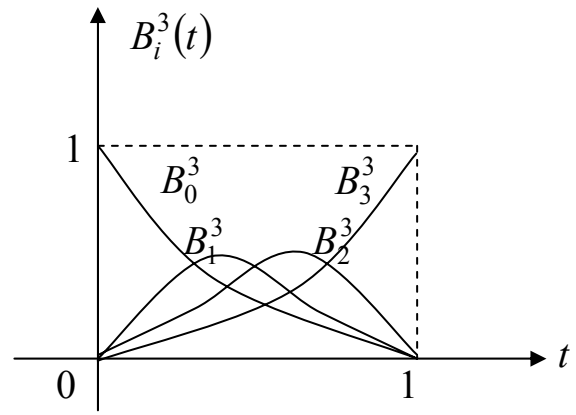


Рис. 7.3. Вагові функції Безьє/Бернштейна при $m = 3$

На рис. 7.3 зображено графіки вагових коефіцієнтів кривої Безьє $B_i^m(t)$ при $m = 3$.

Розглянемо випадок $m = 3$, тобто елементарну кубічну криву Безьє. Це особливий випадок кривих третього порядку, які часто використовуються для побудови сплайнових кривих. *Кубічна крива Безьє* визначається чотирма вершинами й описується рівнянням виду (впливає з (7.7))

$$R(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3, \quad t \in [0, 1],$$

або в матричній формі

$$R(t) = PMT,$$

$$\text{де } R(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \quad P = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \end{pmatrix}, \quad M = \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}.$$

Вектор $P = (P_0, P_1, P_2, P_3)$ називається *геометричним вектором* елементарної кубічної кривої Безьє, а матриця M – *базовою матрицею* кубічної кривої Безьє. Такі базові матриці M можна виписати для будь-якого значення m [23].

7.5.2. Властивості кривих Безьє

Для згладжувальних кривих Безьє за вагові коефіцієнти беруться *многочлени Бернштейна*

$$B_i^m(t) = \frac{m!}{i!(m-i)!} t^i (1-t)^{m-i}.$$

Властивості многочленів Бернштейна суттєво впливають на поведінку кривих Безьє. Наведемо деякі з них:

1. Многочлени Бернштейна набувають невід'ємних значень.
2. В сумі вони дають одиницю:

$$\sum_{i=0}^m B_i^m(t) = 1. \quad (7.8)$$

3. Не залежать від вершин масиву P , а залежать лише від кількості точок у масиві.

Криві Безьє володіють рядом цікавих властивостей, завдяки чому вони широко застосовуються на практиці. Розглянемо основні властивості кривих Безьє:

1. Степінь многочлена $R(t)$, що визначає криву Безьє, на одиницю менший від кількості точок масиву, тобто крива Безьє побудована на $m + 1$ точці задається многочленом степеня m .

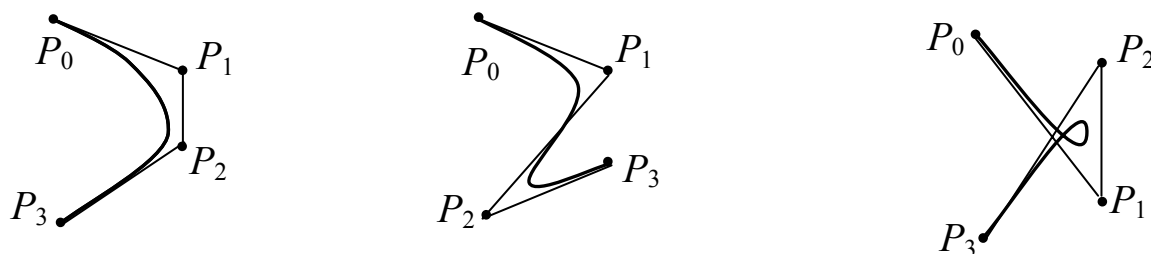


Рис. 7.4. Кубічні криві Безьє при зміні порядку точок ($m = 3$)

2. Порядок точок у масиві P суттєво впливає на вигляд кривої Безьє. Форма кривої Безьє повторює хід ламаної $P_0P_1\dots P_m$. При зміні порядку точок повністю змінюється форма кривої (рис. 7.4).
3. Перша та остання точки кривої збігаються з відповідними точками масиву P , тобто $R(0) = P_0$, $R(1) = P_m$.
4. Вектори дотичних у кінцях кривої Безьє за напрямом збігаються з першою й останньою ланками опорної ламаної, оскільки $\dot{R}(0) = m(P_1 - P_0)$, $\dot{R}(1) = m(P_m - P_{m-1})$.
5. Крива Безьє є гладкою кривою. Зокрема, першу похідну радіус-вектора $R(t)$ можна записати у вигляді

$$\dot{R}(t) = m \sum_{i=0}^{m-1} (P_{i+1} - P_i) B_i^{m-1}(t).$$

6. Крива Безьє лежить в опуклій оболонці вершин масиву P , оскільки для коефіцієнтів лінійної комбінації точок масиву виконується умова (7.8).
7. Крива Безьє інваріантна відносно афінних перетворень. Тобто побудувавши криву Безьє, над нею можна здійснити афінні перетворення, а можна вчинити інакше: спочатку здійснити афінні пере-

творення над опорними вершинами, а потім побудувати криву Безьє на нових вершинах. Якщо результати будуть однаковими, то кажуть, що крива є інваріантною відносно цього афінного перетворення. При цьому легко помітити, наприклад, що поворот опорних вершин і подальша побудова кривої займає менше часу, ніж поворот самої кривої. Нагадаємо, що афінні перетворення включають у себе поворот, розтягування/стиск, паралельне перенесення та їх можливі комбінації.

8. Якщо в масив P додати хоча б одну вершину, то необхідно повністю перерахувати параметричні рівняння кривої Безьє.
9. Зміна хоча б однієї точки в масиві P приводить до помітної зміни всієї кривої Безьє.
10. У рівнянні (7.7), що описує елементарну криву Безьє, немає вільних параметрів, тобто заданий масив однозначно визначає криву Безьє. Тому немає можливостей якимось чином впливати на форму кривої Безьє.

Останній недолік можна усунути, якщо ввести до розгляду *раціональні криві Безьє*, які визначаються формулою

$$R(t) = \frac{\sum_{i=0}^m \omega_i B_i^m(t) P_i}{\sum_{i=0}^m \omega_i B_i^m(t)}, \quad t \in [0, 1], \quad (7.9)$$

де $\omega_i \geq 0$ – вагові коефіцієнти, сума яких строго додатна.

Змінюючи параметри ω_i , можна керувати формою раціональних кривих Безьє. Якщо значення ω_i велике, то крива $R(t)$ проходить близько до точки P_i , якщо ω_i мале, то $R(t)$ проходить даліше від точки P_i . Якщо всі ω_i рівні між собою, то одержується звичайна крива Безьє.

7.5.3. Складені криві Безьє

Кількість точок множини P жорстко визначає порядок многочлена Безьє. Крім цього, довільна точка на кривій Безьє залежить від всіх опорних вершин, тому зміна однієї вершини впливає на поведінку всієї кривої. Іншими словами, локальні дії на криву Безьє неможливі, а відсутність локальної корекції може бути визначальною в деяких прикладних задачах. Вихід із цієї ситуації був знайдений у побудові складених кривих Безьє, тобто сплайнову криву складають з окремих фрагментів. Саме такі криві найчастіше застосовуються на практиці.

Зупинимось тепер на побудові складених кубічних кривих Безьє.

Складена кубічна крива Безьє – це неперервна крива γ , що є об'єднанням елементарних кубічних кривих $\gamma^0, \gamma^1, \dots, \gamma^l$, тобто

$$\gamma = \gamma^0 \cup \gamma^1 \cup \dots \cup \gamma^l.$$

Для побудови складеної кривої Безьє розбиваємо масив точок P_0, P_1, \dots, P_m (m має бути кратним 3) на l підмножин, кожна з яких містить чотири точки так, що остання точка попередньої підмножини – це перша точка наступної підмножини, тобто

$$\{P_0, P_1, P_2, P_3\}, \{P_3, P_4, P_5, P_6\}, \dots, \{P_{3l}, P_{3l+1}, P_{3l+2}, P_{3l+3}\}.$$

На кожній такій підмножині можна побудувати елементарну криву Безьє, тобто окремих фрагмент складеної кривої Безьє. Об'єднання окремих фрагментів дає складену кубічну криву Безьє, яка належить до класу неперервних функцій. Якщо тепер змінити якусь точку масиву, то складена крива Безьє зміниться тільки локально.

Для того, щоб складена кубічна крива Безьє була C^1 -гладкою необхідно, щоб трійки вершин $P_{3i-1}, P_{3i}, P_{3i+1}$ лежали на одній прямій.

Якщо у вихідному масиві немає колінеарних трійок вершин, їх завжди можна додати так, щоб побудована складена крива була C^1 -гладкою. C^2 -гладкості можна добитися для раціональних складених кривих Безьє за рахунок підбору вагових коефіцієнтів ω_i .

Зауваження 3. Метод побудови кривої Безьє в графічних пакетах базується на використанні дотичних управляючих ліній, що проведені до сегмента кривої на його кінцях (вузлах). При цьому форму кривої Безьє моделюють шляхом зміни нахилу дотичних і довжини відрізка, які визначаються контрольними точками.

7.5.4. Геометричний алгоритм для кривої Безьє

П'єр Безьє запропонував інший метод побудови цих кривих на основі геометричних міркувань. Алгоритм полягає в наступному.

1. Кожна сторона опорної ламаної ділиться у відношенні $t:(1-t)$, де $t \in [0, 1]$ – аргумент точки поділу.
2. Точки поділу з'єднуються відрізками прямих і утворюють нову ламану. При цьому кількість вузлів нового контуру на одиницю менша за кількість вузлів попереднього контуру.
3. Сторони нового контуру знову діляться у тому ж відношенні і так далі. Цей процес продовжується доти, поки не буде отримано єдину точку поділу. Ця точка й буде точкою кривої Безьє (рис. 7.5).

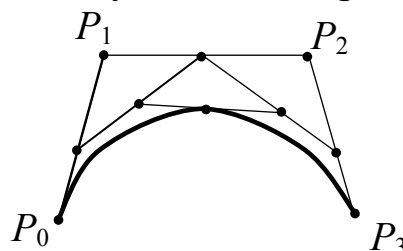


Рис. 7.5. Геометричний алгоритм для кривих Безьє

7.6. В-сплайнові криві

Крива, що визначається вершинами масиву P , залежить від способу апроксимації. Тут значну роль відіграє вибір базисних функцій. Базис Бернштейна породжує криві Безьє, однак він володіє двома недоліками.

- 1) Кількість вершин ламаної жорстко визначає порядок многочлена Безьє (див. формулу (7.7)). Наприклад, чотири вершини ламаної визначають кубічну криву Безьє, шість вершин ламаної визначають многочлен Безьє п'ятого порядку. Понизити степінь кривої Безьє можна або зменшивши кількість вершин ламаної, або ввівши до розгляду складені криві Безьє.
- 2) Друге обмеження випливає з глобальної природи базису Бернштейна. Це означає, що величина вагових коефіцієнтів (многочленів Бернштейна $B_i^m(t)$) є ненульовою для всіх значень параметра t на кривій. Тобто довільна точка на кривій Безьє залежить від всіх вершин, тому зміна однієї вершини приводить до помітної зміни всієї кривої Безьє. Локальні дії на криву Безьє, що визначається формулою (7.7), – неможливі.

Відсутність локальної корекції кривих може бути визначальною в деяких прикладних задачах. Цей недолік заважає ефективному розв'язуванню задач згладжування, тому на практиці використовується неглобальний базис, що називається базисом В-сплайна. В-сплайни не глобальні, оскільки з кожною вершиною P_i пов'язана своя базисна функція. Тому вплив кожної вершини на криву відбувається тільки при тих значеннях параметра, при яких відповідна базисна функція не дорівнює нулю, тобто зміна однієї контрольної точки впливає тільки на деякий фрагмент кривої і не вносить глобальних змін у її форму.

Сплайнові криві використовуються в анімації для задання траєкторії руху. Але при русі об'єкта по складеній кривій Безьє раптово може змінитися прискорення, оскільки об'єкт переходить на ділянку кривої з іншою другою похідною (складена крива Безьє тільки C^1 -неперервна). Цю проблему теж розв'язують В-сплайни, в яких зміна кривизни неперервна. Тому цей тип кривих оптимальний для анімації і створення 2D-форм.

Розглянемо ці специфічні сплайни, що називаються В-сплайнами [5, 19, 23]. Найпоширеніші серед них – В-сплайни третього порядку.

В-сплайни третього порядку визначаються на системі точок $x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}$. Такий сплайн позначається $N_i^3(x)$. Вважаємо, що вузли x_i рівномірно віддалені. Нехай $N_i^3(x_k) = y_k, (N_i^3)''(x_k) = M_k, k = i-2, i-1, i, i+1, i+2$. Тоді, як і для всякого кубічного сплайна, у внутрішніх точках, виконуються співвідношення

$$M_{k-1} + 4M_k + M_{k+1} = \frac{6}{h^2}(y_{k-1} - 2y_k + y_{k+1}), \quad k = i-1, i, i+1. \quad (7.10)$$

Маємо 3 рівняння і 10 невідомих $y_k, M_k, k = i-2, i-1, i, i+1, i+2$. Тому додатково вважатимемо

$$\begin{aligned} N_i^3(x_{i-2}) = y_{i-2} = 0, \quad N_i^3(x_{i+2}) = y_{i+2} = 0, \\ (N_i^3)'(x_{i-2}) = (N_i^3)'(x_{i+2}) = 0, \\ (N_i^3)''(x_{i-2}) = M_{i-2} = 0, \quad (N_i^3)''(x_{i+2}) = M_{i+2} = 0. \end{aligned} \quad (7.11)$$

З формул (7.10), враховуючи (7.11), одержуємо

$$\begin{aligned} 4M_{i+1} + M_i &= \frac{6}{h^2}(y_i - \frac{1}{3}h^2M_{i-1}), \\ M_{i-1} + 4M_i + M_{i+1} &= \frac{6}{h^2}(\frac{1}{6}h^2M_{i+1} - 2y_i + \frac{1}{6}h^2M_{i-1}), \\ M_i + 4M_{i+1} &= \frac{6}{h^2}(y_i - \frac{1}{3}h^2M_{i+1}). \end{aligned} \quad (7.12)$$

У системі (7.12) маємо чотири невідомих, тому додатково вважатимемо, що $y_i = \frac{2}{3}$, що зумовлено умовою нормалізації $\sum_{k=i-2}^{i+2} N_k^3(x) = 1$. Для

решти невідомих з (7.12) знаходимо $M_{i-1} = \frac{1}{h^2}, M_i = -\frac{2}{h^2}, M_{i+1} = \frac{1}{h^2}$.

Крім цього, з умов нормалізації випливає, що $y_{i-1} = y_{i+1} = \frac{1}{6}$.

Використовуючи формулу (7.1), одержуємо

$$N_i^3(x) = \begin{cases} \frac{1}{6} \left(\frac{x - x_{i-2}}{h} \right)^3, & x \in [x_{i-2}, x_{i-1}], \\ -\frac{1}{3} \left(\frac{x - x_{i-1}}{h} \right)^3 + \frac{1}{6} \left(\frac{x_i - x}{h} \right)^3 + \frac{x - x_{i-1}}{h}, & x \in [x_{i-1}, x_i], \\ \frac{1}{6} \left(\frac{x - x_i}{h} \right)^3 - \frac{1}{3} \left(\frac{x_{i+1} - x}{h} \right)^3 + \frac{x_{i+1} - x}{h}, & x \in [x_i, x_{i+1}], \\ \frac{1}{6} \left(\frac{x_{i+2} - x}{h} \right)^3, & x \in [x_{i+1}, x_{i+2}], \\ 0, & \text{для решти } x. \end{cases} \quad (7.13)$$

Кубічний *B*-сплайн на відрізку $[a, b]$ можна визначити як функцію

$$B(x) = \sum_{i=0}^n b_i N_i^3(x), \quad x \in [a, b],$$

де b_i – коефіцієнти, що визначаються з відповідної системи рівнянь, $N_i^3(x)$ – нормалізовані функції базису В-сплайна.

Функція $B(x)$ є поліномом третього степеня на кожному інтервалі $x_i \leq x < x_{i+1}$, похідні $B'(x)$, $B''(x)$ неперервні у кожній точці кривої.

Оскільки В-сплайн задається базисом В-сплайна, то сума базисних сплайнів у довільній точці

$$\sum_{i=0}^m N_i^3(x) = 1.$$

Елементарні В-сплайнові криві визначаються чотирма вершинами P_0, P_1, P_2, P_3 .

У нашому випадку, коли жодних обмежень на множину вершин не накладається (вони можуть бути розміщені довільно), рівняння елементарної В-сплайнової кривої потрібно шукати в параметричній формі

$$R(t) = \sum_{i=0}^3 n_i(t) P_i,$$

де $n_i(t)$ – коефіцієнти, що одержуються із формули (7.13) шляхом введення параметра t .

Наприклад, при $x \in [x_i, x_{i+1}]$

$$N_i^3(x) = \frac{1}{6} \left(\frac{x - x_i}{h} \right)^3 - \frac{1}{3} \left(\frac{x_{i+1} - x}{h} \right)^3 + \frac{x_{i+1} - x}{h}.$$

Якщо тепер ввести параметр $t = \frac{x - x_i}{h}$, $t \in [0, 1]$, то з (7.13) одержуємо

мо функцію $n_1(t) = \frac{3t^3 - 6t^2 + 4}{6}$.

Аналогічно з формули (7.13) маємо

$$n_0(t) = \frac{(1-t)^3}{6}, \quad n_2(t) = \frac{-3t^3 + 3t^2 + 3t + 1}{6}, \quad n_3(t) = \frac{t^3}{6}, \quad t \in [0, 1].$$

Графіки функцій $n_i(t)$ наведені на рис.7.5. На кінцях одиничного інтервалу три функції з чотирьох відмінні від нуля.

Отже, елементарну кубічну В-сплайнову криву можна визначити векторним рівнянням:

$$R(t) = \frac{(1-t)^3}{6} P_0 + \frac{3t^3 - 6t^2 + 4}{6} P_1 + \frac{-3t^3 + 3t^2 + 3t + 1}{6} P_2 + \frac{t^3}{6} P_3, \quad t \in [0, 1]. \quad (7.14)$$

Матричний запис параметричного рівняння елементарної кубічної В-сплайнової кривої має вигляд

$$R(t) = PMT,$$

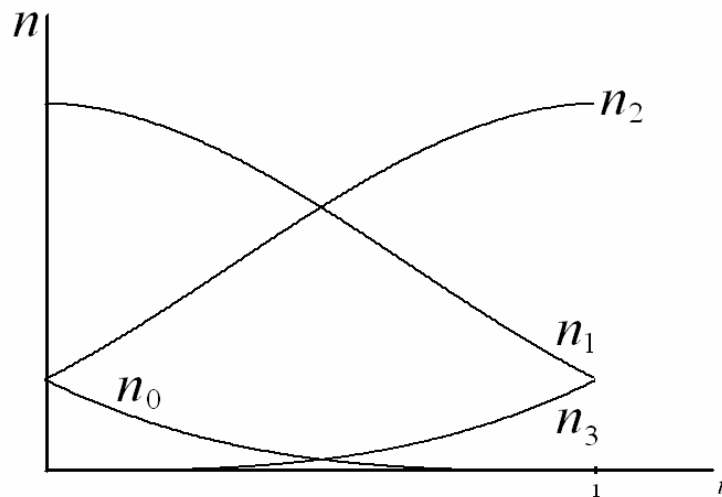


Рис 7.5. Вагові коефіцієнти елементарних В-сплайнів

де
$$R(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \quad M = \frac{1}{6} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix},$$

$$P = (P_0 P_1 P_2 P_3) = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \end{pmatrix}.$$

Матриця M називається *базовою матрицею* елементарної В-сплайнової кривої, а вектор P – *геометричним вектором* елементарної В-сплайнової кривої.

Функціональні коефіцієнти $n_i(t)$, $i = 0, 1, 2, 3$ суттєво впливають на поведінку $R(t)$. Оскільки $n_i(t) \geq 0$, $i = 0, 1, 2, 3$ і $\sum_{i=0}^3 n_i(t) = 1$, то елементарна В-сплайнова крива лежить в опуклій оболонці точок P_0, P_1, P_2, P_3 (рис. 7.6), причому $R(0) \neq P_i$, $R(1) \neq P_i$, $i = 0, 1, 2, 3$, тобто елементарна В-сплайнова крива, як правило, не проходить через жодну з точок P_0, P_1, P_2, P_3 (рис. 7.6).

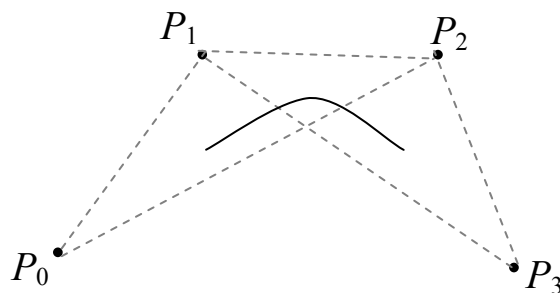


Рис. 7.6. Елементарна кубічна В-сплайнова крива

Дотична в початковій точці $R(0) = \frac{1}{6}(P_0 + 4P_1 + P_2)$ паралельна від-
 різку P_0P_1 ($\dot{R}(0) = \frac{1}{2}(P_2 - P_0)$), а в кінцевій точці $R(1) = \frac{1}{6}(P_1 + 4P_2 + P_3)$
 паралельна відрізку P_1P_3 ($\dot{R}(1) = \frac{1}{2}(P_3 - P_1)$).

Складені кубічні В-сплайнові криві

Складеною кубічною В-сплайновою кривою, що визначається масивом
 вершин P_0, P_1, \dots, P_m , називається крива γ , яку можна подати у вигляді
 об'єднання елементарних В-сплайнових кривих $\gamma^1, \gamma^2, \dots, \gamma^{m-2}$,

$$\gamma = \gamma^1 \cup \gamma^2 \cup \dots \cup \gamma^{m-2},$$

де γ^i будується як елементарний В-сплайн на точках $P_{i-1}, P_i, P_{i+1}, P_{i+2}$.

Сформулюємо *властивості складеної кубічної В-сплайнової кривої*.

1. Складена кубічна В-сплайнова крива, що породжується масивом
 вершин P_0, P_1, \dots, P_m , є C^2 -гладкою кривою.
2. Форма кривої визначається контрольними точками, що не лежать
 на кривій (рис. 7.7). За допомогою цих точок зручно управляти
 формою кривої.
3. Складена крива лежить в об'єднанні $m - 2$ опуклих оболонок, що
 породжуються точками $P_{i-1}, P_i, P_{i+1}, P_{i+2}, i = 1, 2, \dots, m - 2$.
4. Якщо всі опорні вершини лежать на одній прямій, то складена ку-
 бічна В-сплайнова крива лежить на цій прямій між точками P_0, P_m .
5. Складена В-сплайнова крива не глобальна, тому локальна зміна
 однієї точки в масиві приводить до зміни тільки частини кривої;
 при зміні вершини P_i необхідно переобчислити параметричні рів-
 ння тільки чотирьох елементарних кривих $\gamma^{i-2}, \gamma^{i-1}, \gamma^i, \gamma^{i+1}$.
6. Складена кубічна В-сплайнова крива є афінно інваріантною.

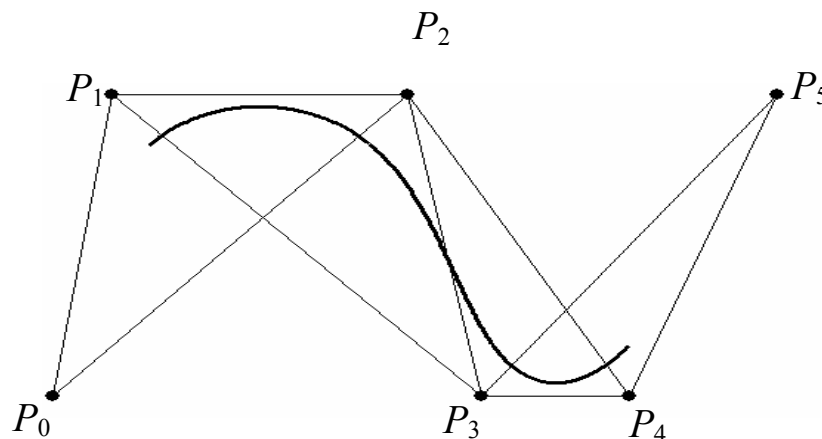


Рис. 7.7. Складена В-сплайнова крива

Зауваження 4. Складена кубічна В-сплайнова крива, як правило, не проходить через жодну з вершин P_0, P_1, \dots, P_m . Однак шляхом підбору допоміжних вершин і побудовою додаткових елементарних кривих, можна добитися, щоб початкова точка нової складеної кривої знаходилася близько до вершини P_0 і навіть збігалася з нею. Аналогічних результатів можна досягти і для кінцевої точки P_m .

Зауваження 5. На взаємне розміщення вершин у масиві не накладається жодних обмежень і вони можуть навіть збігатися. Але в таких випадках крива може втратити порядок гладкості.

Сегменти, що утворюють складену криву, визначаються за формулою (7.14) як функції аргументу $t \in [0, 1]$. Інколи їх зручніше перевизначити так, щоб вони набували значення на послідовному інтервалі. Для єдиної параметризації складеної кривої γ вводять параметр $t \in [0, m - 2]$ (оскільки на $m + 1$ вершині можна побудувати $m - 2$ елементарних фрагменти). Кожний сегмент γ^i буде задаватися на інтервалі $t \in [t_i, t_{i+1}]$, $i = 1, 2, \dots, m - 2$.

Значення t_i визначають точку стику або вузол між сусідніми сегментами. Тому t_i називаються *вузловими значеннями*. Загальне число вузлів на складеній кривій дорівнює $m - 1$.

Якщо всі вузли розміщені на рівній відстані за значенням параметра t , то такі В-сплайни називаються *однорідними*. Без втрати загальності можна вважати, що $t_1 = 0, t_{i+1} = t_i + 1, i = 1, 2, \dots, m - 2$, наприклад вектор вузлових значень взяти у вигляді $(0, 1, 2, 3, 4)$. У цьому випадку параметричне рівняння i -ї елементарної кубічної В-сплайнової кривої матиме вигляд

$$R^i(t) = (P_{i-1}, P_i, P_{i+1}, P_{i+2})M \begin{pmatrix} 1 \\ t - t_i \\ (t - t_i)^2 \\ (t - t_i)^3 \end{pmatrix}, t \in [t_i, t_{i+1}].$$

В-сплайни, в яких відстані між вузловими значеннями нерівні, але задовольняють співвідношення $t_i \leq t_{i+1}$, називаються *неоднорідними В-сплайнами*, наприклад, якщо вектор вузлових значень має вигляд $(0, 0, 0, 1, 1, 2, 3)$, або $(0; 0,18; 0,3; 0,5; 0,7; 1)$.

На відміну від однорідних В-сплайнів, для неоднорідних В-сплайнів не існує єдиного набору базових функцій. Вони залежать від вузлових значень t_i і визначаються рекурентними співвідношеннями. Вектор вузлових значень потрібно попередньо задавати.

Позначимо через $N_i^k(t)$ базову функцію k -го порядку для контрольної

точки P_i . Нормалізовані функції базису $N_i^k(t)$ визначаються рекурентними формулами Кокса-де-Бура

$$N_i^0(t) = \begin{cases} 1, & t_i \leq t < t_{i+1}, \\ 0, & t < t_i, t \geq t_{i+1}, \end{cases}$$

$$N_i^1(t) = \frac{t - t_i}{t_{i+1} - t_i} N_i^0(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} N_{i+1}^0(t), \quad (7.15)$$

.....

$$N_i^k(t) = \frac{t - t_i}{t_{i+k} - t_i} N_i^{k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1}^{k-1}(t), \quad k = 1, 2, 3, \dots$$

Тоді В-сплайн побудований на множині точок $P = \{P_0, P_1, \dots, P_m\}$ з вектором вузлових значень $t_0, t_1, \dots, t_k, \dots, t_m, \dots, t_{m+k+1}$ має вигляд

$$R(t) = \sum_{j=i-k}^i N_j^k(t) P_j, \quad t_i \leq t \leq t_{i+1}, \quad i = k, k+1, \dots, m, \quad 1 \leq k \leq m-1. \quad (7.16)$$

Зокрема, сегмент γ^i , що задається контрольними точками $P_{i-1}, P_i, P_{i+1}, P_{i+2}$ при $k=3$ визначається рівнянням

$$R_i(t) = N_{i-1}^3(t) P_{i-1} + P_i N_i^3(t) + N_{i+1}^3(t) P_{i+1} + N_{i+2}^3(t) P_{i+2}, \quad t \in [t_{i+2}, t_{i+3}].$$

Функції $N_i^k(t)$ мають локальний характер, тобто на інтервалі (t_i, t_{i+k}) $N_i^k(t) > 0$, зовні цього інтервалу $N_i^k(t) = 0$. Тому зміна однієї вершини в масиві P не приводить до переобчислення всієї кривої.

Крім цього, для функцій $N_i^k(t)$ зберігається рівність

$$\sum_{i=k}^m N_i^k(t) = 1,$$

завдяки якій крива, задана векторним рівнянням (7.16), завжди належатиме до опуклої оболонки масиву P .

Неоднорідні В-сплайни мають ряд переваг над однорідними. Вони володіють високою чуттєвістю до зміни контрольних точок, забезпечують легке керування початковими та кінцевими точками. В будь-який момент роботи з неоднорідними В-сплайнами можна ввести додатковий вузол і змінити форму кривої.

Раціональні кубічні В-сплайнові криві. За заданим масивом точок $P = \{P_0, P_1, \dots, P_m\}$ раціональна кубічна В-сплайнова крива визначається рівнянням вигляду

$$R(t) = \frac{\sum_{i=0}^m \omega_i N_i^3(t) P_i}{\sum_{i=0}^m \omega_i N_i^3(t)}.$$

Вагові коефіцієнти $\omega_i \geq 0$, для яких $\sum_i \omega_i > 0$, як і для раціональної кривої Безьє, керують поведінкою кривої.

Вираз для $R(t)$ можна переписати у формі

$$R(t) = \sum_{i=0}^m \bar{N}_i^3(t) P_i,$$

де

$$\bar{N}_i^3(t) = \frac{\omega_i N_i^3(t)}{\sum_{i=0}^m \omega_i N_i^3(t)} > 0, \quad \sum_{i=0}^m \bar{N}_i^3(t) = 1.$$

Тому раціональні В-сплайни – це узагальнення нераціональних В-сплайнів і їх базису. І оскільки не накладається ніяких обмежень на розміщення вузлів, то ця функція належить до класу нерівномірних раціональних В-сплайнів. Нерівномірні раціональні В-сплайни в літературі з комп'ютерної графіки називаються ще NURBS-кривими (скорочено від nonuniform rational B-spline). NURBS-криві зберігають властивості В-сплайнів, зокрема вони володіють властивістю афінної інваріантності. Але, крім цього, раціональні В-сплайни інваріантні відносно перспективного перетворення, нераціональні В-сплайни зберігають інваріантність тільки відносно масштабування, повороту і зсуву.

Таким чином, для реалізації перспективного перетворення раціональних кривих достатньо застосувати це перетворення до контрольних точок, нове положення яких дозволить обчислити перетворену форму кривих.

Ще однією перевагою NURBS-кривих є їхня здатність точно описувати форми конічних перетинів, які часто використовуються в системах автоматизованого проектування. Можна показати, що квадратичні криві, які задаються аналітично в неявному вигляді, є окремим випадком зручних NURBS-кривих. Нераціональні криві можуть тільки апроксимувати ці форми, причому для збільшення точності необхідно використовувати велику кількість контрольних точок, що вимагає значних обчислювальних ресурсів.

Отже, для моделювання практично всіх типів криволінійних об'єктів, можна використовувати метод, що базується на формуванні NURBS-кривих.

Зауважимо, що нерівномірні раціональні В-сплайни є стандартом обміну проектною інформацією між системами машинного проектування, тобто вони використовуються в універсальних форматах подання геометричних даних. Раціональні В-сплайни застосовуються в системах геометричного моделювання, сучасних САД-системах й реалізовані апаратно в деяких графічних станціях.

7.7. Інтерполяційні кубічні криві Ерміта

Форма опису кубічних кривих Ерміта близька до кривих Безьє, але відрізняється від неї заданням дотичних векторів у кінцевих точках (для кривої Безьє задаються чотири точки).

Нехай задані дві точки P_0, P_1 та два напрямки Q_0, Q_1 . Необхідно побудувати криву, яка пройде через ці точки, причому напрями дотичних у цих точках до шуканої кривої повинні збігатися відповідно з Q_0, Q_1 . Такою кривою є інтерполяційна кубічна крива Ерміта. Таку назву ці криві одержали від імені французького математика Шарля Ерміта, який досліджував їх властивості.

За заданими вершинами P_0, P_1 і ненульовими векторами Q_0, Q_1 елементарна кубічна крива Ерміта визначається рівнянням

$$R(t) = H_0(t)P_0 + H_1(t)P_1 + H_2(t)Q_0 + H_3(t)Q_1, \quad 0 \leq t \leq 1,$$

де $H_i(t), i = 0, 1, 2, 3$ – ермітові базисні функції (ермітові многочлени):

$$H_0(t) = (1 + 2t)(1 - t)^2 = 2t^3 - 3t^2 + 1,$$

$$H_1(t) = (3 - 2t)t^2 = -2t^3 + 3t^2,$$

$$H_2(t) = (1 - t)^2 t = t^3 - 2t^2 + t,$$

$$H_3(t) = (t - 1)t^2 = t^3 - t^2.$$

Матричний запис параметричного рівняння кривої Ерміта має вигляд

$$R(t) = GMT, \quad 0 \leq t \leq 1,$$

де

$$G = (P_0 P_1 Q_0 Q_1) = \begin{pmatrix} x_0 & x_1 & u_0 & u_1 \\ y_0 & y_1 & v_0 & v_1 \end{pmatrix}, \quad R = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \quad M = \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}.$$

Матриця M називається *базовою матрицею кубічної кривої Ерміта*, G – геометричний вектор кривої Ерміта.

Дотичний вектор до елементарної кубічної кривої Ерміта в початковій точці $P_0 = R(0)$ збігається із заданим вектором Q_0 , а в кінцевій точці $P_1 = R(1)$ – з вектором Q_1 (рис. 7.8). Зауважимо, що якщо збільшувати тільки довжину вектора Q_0 , то крива Ерміта витягується вправо.

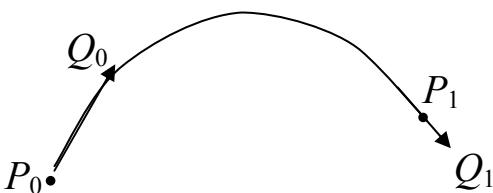


Рис. 7.8. Елементарна крива Ерміта

Складена кубічна крива Ерміта. Складена кубічна крива Ерміта, що визначається масивом точок

$$P_0, P_1, \dots, P_m, \quad m \geq 1$$

і парою ненульових векторів Q_0, Q_m , – це крива γ , яку можна зобразити у вигляді об'єднання кубічних кривих Ерміта $\gamma^1, \gamma^2, \dots, \gamma^m$, тобто

$$\gamma = \gamma^1 \cup \gamma^2 \cup \dots \cup \gamma^m.$$

При цьому i -та крива γ^i визначається параметричним рівнянням вигляду

$$R^i(t) = (P_{i-1} P_i Q_{i-1}, Q_i)MT, \quad 0 \leq t \leq 1, \quad i = 1, 2, \dots, m,$$

де M – базова матриця ермітового сплайна, а вектори Q_1, \dots, Q_{m-1} визначаються з матричного рівняння вигляду

$$\begin{pmatrix} 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & & \ddots & & \\ & & & & 1 & 4 & 1 \end{pmatrix} \begin{pmatrix} Q_0 \\ Q_1 \\ \vdots \\ Q_m \end{pmatrix} = \begin{pmatrix} -3 & 0 & 3 & & & \\ & -3 & 0 & 3 & & \\ & & & \ddots & & \\ & & & & & -3 & 0 & 3 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_m \end{pmatrix}.$$

Наведемо *властивості складеної кубічної кривої Ерміта*.

1. Складена крива є C^2 -гладкою.
2. Проходить через вершини P_0, P_1, \dots, P_m .
3. Дотичні вектори Q_i у внутрішніх точках P_i , $i = 1, 2, \dots, m - 1$ однозначно визначаються через вершини масиву P і дотичні вектори Q_0, Q_m .
4. Зміна однієї вершини або одного з дотичних векторів Q_0, Q_m приводить до зміни всієї кривої.
5. Складена кубічна крива Ерміта – афінно інваріантна, але не проективно інваріантна.

Зауваження 6. За допомогою елементарних кубічних кривих Ерміта складену криву можна побудувати за заданим масивом вершин P_0, P_1, \dots, P_m і довільним набором ненульових векторів Q_0, Q_1, \dots, Q_m . Кожна четвірка $P_{i-1}, P_i, Q_{i-1}, Q_i$ задає елементарну ермітову криву. Однак у цьому випадку, на відміну від раніше розглянутого випадку, складена крива Ерміта лише C^1 -гладка.

Зауваження 7. При побудові сплайнових кривих постійно потрібно обчислювати значення кубічних поліномів. Ефективність обчислень можна підвищити за рахунок використання схеми Горнера

$$at^3 + bt^2 + ct + d = ((at + b)t + c)t + d.$$

Існують і більш ефективні методи обчислень значень поліномів, наприклад, метод скінченних різниць.

7.8. ТСВ-сплайни

У задачах анімації при моделюванні руху за допомогою ключових кадрів задається послідовність положень об'єкта, тобто точок P_i , $i = 0, 1, \dots, m$. Задача полягає в тому, щоб провести інтерполяційну криву, що проходить через ці точки і в такий спосіб визначити положення об'єкта в довільний момент часу.

Для розв'язання цієї задачі використовують модифіковані ермітові сплайнові криві. Вони називаються *ТСВ-сплайнами*.

Розглянемо ці модифікації.

1. У точках P_i , $i = 1, 2, \dots, m - 1$ потрібно визначити вектори швидкостей, які задаватимуть напрям та швидкість зміни кривої в цих точках.

Наприклад, для напрямку можна взяти

$$\mathbf{T}_i = \frac{P_{i+1} - P_{i-1}}{2} = \frac{P_{i+1} - P_i + P_i - P_{i-1}}{2}.$$

Окрім цього, зберігаючи напрям вектора, можна регулювати його довжину. Наприклад, помноживши вектор \mathbf{T}_i на параметр $1 - t$, де $t \in [-1, 1]$, матимемо

$$\mathbf{T}_i = (1 - t) \frac{P_{i+1} - P_{i-1}}{2} = \frac{(1 - t)(P_{i+1} - P_i) + (1 + t)(P_i - P_{i-1})}{2}.$$

Швидкість \mathbf{T}_i визначає натяг кривої (tension). На рис. 7.9 зображені вектори швидкостей для різних значень t . При $t = 0$ вектор \mathbf{T}_i – середня лінія $\Delta P_{i-1}P_iP_{i+1}$. При $t \rightarrow -1$ швидкість зростає, натяг спадає, при $t \rightarrow 1$ швидкість спадає, натяг збільшується.

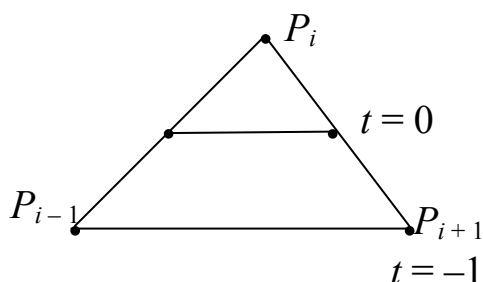


Рис. 7.9. Вектор натягу

2. Вектори $P_{i-1}P_i$, P_iP_{i+1} можуть вносити різний вклад у формування вектора швидкості \mathbf{T}_i . Для врахування цього чинника можна ввести вагові коефіцієнти $1 - \beta$, $1 + \beta$, які описують вклад кожного з векторів P_iP_{i+1} , $P_{i-1}P_i$ у напрямний вектор \mathbf{T}_i , тобто вважати, що

$$\mathbf{T}_i = ((1 - \beta)(P_{i+1} - P_i) + (1 + \beta)(P_i - P_{i-1}))/2.$$

При $\beta = 0$ вектор \mathbf{T}_i з'єднує середини відрізків P_iP_{i+1} та $P_{i-1}P_i$. При $\beta \rightarrow 1$ вклад вектора $P_{i-1}P_i$ збільшується (вектор швидкості \mathbf{T}_i наближається до вектора $P_{i-1}P_i$), а при $\beta \rightarrow -1$ збільшується вклад вектора

$P_i P_{i+1}$ (вектор швидкості наближається до вектора $P_i P_{i+1}$). Так змінюється нахил (bias) кривої при проходженні через точку P_i . Однак напрям входу в точку P_i і напрям виходу з неї збігаються (крива C^1 -гладка).

3. Для забезпечення різного напрямку входу і виходу з точки P_i необхідно розрізнити вектори T_i зліва і справа від точки P_i , тобто необхідно ввести до розгляду вектори T_i^- , T_i^+ .

Аналогічно п.2 вводимо параметр c і задаємо вектори швидкостей за формулами

$$T_i^+ = ((1 - c)(P_{i+1} - P_i) + (1 + c)(P_i - P_{i-1})) / 2,$$

$$T_i^- = ((1 + c)(P_{i+1} - P_i) + (1 - c)(P_i - P_{i-1})) / 2.$$

Параметр c визначає неперервність (continuity) вектора швидкості. При $c = 0$ вектори T_i^+ і T_i^- збігаються, тобто вектор швидкості неперервний, для інших c він буде розривним.

Об'єднуючи разом три раніше розглянуті випадки, одержуємо

$$T_i^+ = \frac{(1-t)(1-\beta)(1-c)}{2}(P_{i+1} - P_i) + \frac{(1-t)(1+\beta)(1+c)}{2}(P_i - P_{i-1}),$$

$$T_i^- = \frac{(1-t)(1-\beta)(1+c)}{2}(P_{i+1} - P_i) + \frac{(1-t)(1+\beta)(1-c)}{2}(P_i - P_{i-1}).$$

Отже, параметри t , c , β задають відповідно натяг, неперервність і загальний нахил вектора швидкості при проходженні точки P_i . Тому такі сплайни називаються *ТСВ-сплайнами*.

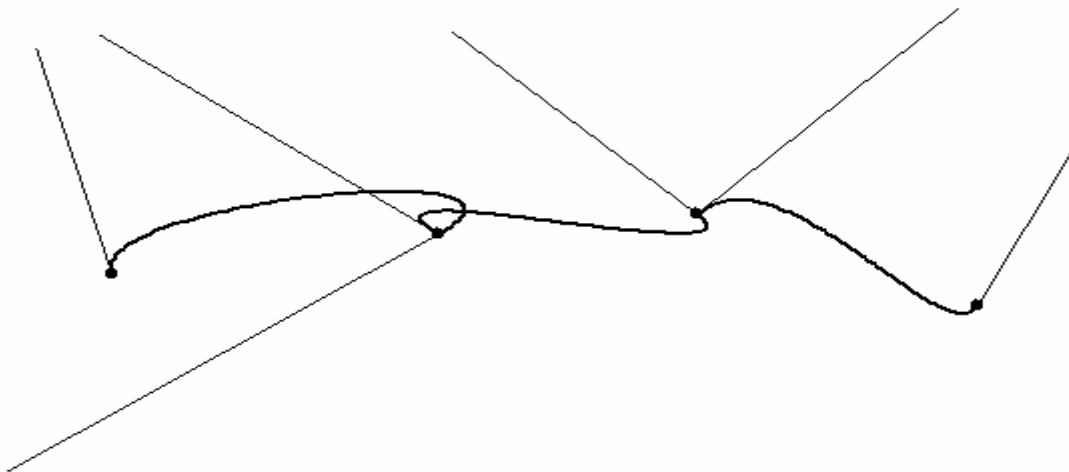


Рис. 7.10. ТСВ-сплайн

Зауваження 8. Сплайнові криві у випадку простору задаються такими ж самими формулами, тільки як $R(t)$, P_i потрібно розглядати

$$R(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}, \quad P_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}.$$

Контрольні питання та завдання

1. Як ставиться задача інтерполяції та задача згладжування?
2. Що таке інтерполяційний сплайн, які він має переваги над інтерполяційним многочленом? Назвіть його властивості.
3. Які властивості має інтерполяційний кубічний сплайн?
4. Що таке сплайнові криві, в якій формі їх розшукують?
5. Що таке контрольна ламана, контрольні точки?
6. Дайте визначення кривих Безьє.
7. Як записати елементарну криву Безьє в матричній формі?
8. Які властивості мають поліноми Бернштейна?
9. Назвіть властивості кривих Безьє?
10. Як за заданим масивом точок побудувати замкнену криву Безьє? Як визначити порядок гладкості? При яких умовах крива буде належати класу C^1 ?
11. Які криві Безьє називаються складеними? Коли ці криві є гладкими в точках стику?
12. Вкажіть переваги та недоліки кривих Безьє?
13. Як визначаються В-сплайни третього порядку?
14. Які властивості мають елементарні кубічні В-сплайнові криві?
15. Які властивості мають складені кубічні В-сплайнові криві?
16. Що таке однорідні/неоднорідні В-сплайни?
17. Як визначаються раціональні кубічні В-сплайнові криві? Як можна управляти їх формою?
18. Що таке NURBS-криві, в чому їх переваги?
19. Наведіть означення кривих Ерміта.
20. Назвіть властивості складеної кубічної кривої Ерміта.
21. Порівняйте криві Безьє, В-сплайни, криві Ерміта в плані інтерполяції контрольних точок і гладкості кривих у точках стику.
22. Що таке ТСВ-сплайни? Якими параметрами вони визначаються?

Вправи і задачі для самостійного виконання

1. Показати аналітично, що наведений геометричний алгоритм для кубічної кривої Безьє приводить до рівняння кривої вигляду $R(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2(1 - t) P_2 + t^3 P_3$.
2. Розробити ефективний метод обчислення кубічних поліномів, який базується на обчисленні правих різниць.
3. Побудувати кубічну криву Безьє, яка апроксимує четверту частину кола, що лежить в першій чверті. *Вказівка. Для побудови кубічної кривої Безьє потрібно мати чотири точки. Дві крайніх точки уже відомі – це кінці дуги, що задає чверть кола. Дві*

інші опорних точки знайдемо, використовуючи властивості кривих Безьє, симетрію відносно бісектриси координатного кута і той факт, що крива Безьє при $t=0,5$ повинна пройти через точку перетину кола і бісектриси.

4. Нехай при $t = 0$ крива Безьє проходить через точку A , при $t = 1/3$ – через точку B , $t = 2/3$ – через точку C , при $t = 1$ – через точку D . Знайти опорні точки P_0, P_1, P_2, P_3 . (альтернативне визначення кривої Безьє). Теж саме виконати для В-сплайнової кривої.
5. Ламана Безьє задана трьома точками $(0, 0), (0, 9), (18, 0)$. Визначити координати точки на кривій Безьє при $t = 1/3$.
6. Записати в матричній формі квадратурні криві Безьє.
7. Показати, що поліноми Бернштейна на інтервалі $(0, 1)$ набувають значення, менші за одиницю.
8. Використовуючи квадратурні криві Безьє, сконструювати контур гліфа для букв u, a, G .
9. Дана крива Безьє $R(t), t \in [0, 1]$, яка визначається точками P_0, P_1, P_2, P_3 . Розіб'ємо криву $R(t)$ на дві частини: ліву при $t \in [0, 1/2]$ і праву при $t \in [1/2, 1]$. Знайти точки L_0, L_1, L_2, L_3 , що визнають криву Безьє, яка збігається з лівою частиною та точки R_0, R_1, R_2, R_3 , що визначають криву Безьє, яка збігається з правою частиною.
10. Вивести параметричні рівняння кривої Ерміта.
Вказівка. Рівняння необхідно шукати у вигляді

$$R(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} a_1 t^3 + b_1 t^2 + c_1 t + d_1 \\ a_2 t^3 + b_2 t^2 + c_2 t + d_2 \end{pmatrix}.$$

11. Записати формули переходу від форми кривої Ерміта до форми кривої Безьє.
12. Побудувати рівняння кривої Ерміта, що проходить через точки з координатами $(0, a), (1, b)$, а похідні в початковій та кінцевій точках дорівнюють $y'(0) = u, y'(1) = v$.
13. Які властивості має елементарна кубічна В-сплайнова крива побудована на масиві точок P_0, P_1, P_2, P_3 у випадку, якщо а) $P_1 = P_2$, б) $P_1 = P_2 = P_3$?
14. Де буде починатися/закінчуватися В-сплайнова крива, якщо до масиву точок P_0, P_1, P_2, P_3 додати опорні вершини P_{-2}, P_{-1}, P_4, P_5 так, що $P_{-2} = P_{-1} = P_0, P_3 = P_4 = P_5$?

Тема 8. Математичні моделі поверхонь

Поверхні та їх опис відіграють важливу роль у конструюванні та виробництві. Наглядні приклади – проектування і виробництво автомобільних кузовів, корабельних корпусів, авіаційних фюзеляжів і крил, посуду, взуття тощо. Опис поверхонь відіграє важливу роль при зображенні даних, одержаних у наукових експериментах. Сучасні засоби машинної графіки мають значні можливості для моделювання різноманітних поверхонь. Як приклад можна навести графічну систему AutoCAD, яка стала основою багатьох САПР у різних областях науки і техніки.

Для комп'ютерної графіки і автоматизованого проектування необхідно мати математичну модель поверхні. Така модель дозволяє відносно легко провести аналіз характеристик поверхні (наприклад, кривизни) та спростити процес візуалізації поверхні. Поверхні можна задавати різними способами, наприклад виконати інтерполяцію або апроксимацію по точках, шляхом переміщення твірної лінії за заданою траєкторією, аналітичною формулою тощо.

У комп'ютерній графіці використовуються багато різновидів аналітичних моделей опису поверхонь: у вигляді загального рівняння неявного вигляду $F(x, y, z) = 0$, явного $z = f(x, y)$ та параметричного.

Явний та неявний вигляд рівнянь поверхні має ряд недоліків, тому на практиці часто використовується параметрична форма опису поверхні

$$x = F_x(u, v), y = F_y(u, v), z = F_z(u, v), (u, v) \in D, \quad (8.1)$$

а також векторна $r = r(u, v)$, $(u, v) \in D$, де D – деяка область параметрів u, v .

Наведемо приклади деяких поверхонь і їхні параметричні рівняння.

Еліпсоїд:

$$\begin{aligned} x &= a \cos u \sin v, & 0 \leq u \leq 2\pi, \\ y &= b \sin u \sin v, & 0 \leq v \leq 2\pi, \\ z &= c \cos v. \end{aligned}$$

Однополлий гіперболоїд:

$$\begin{aligned} x &= a \cos u \operatorname{ch} v, & 0 \leq u \leq 2\pi, \\ y &= b \sin u \operatorname{sh} v, & -\pi \leq v \leq \pi, \\ z &= c \operatorname{sh} v. \end{aligned}$$

Основна перевага параметричного описання функцій – це можливість передачі геометричної форми достатньо складних поверхонь, які іншими рівняннями описати дуже складно. Будь-яку поверхню, що описана неявно, можна подати і в параметричній формі (наприклад, еліпсоїд), але не навпаки (наприклад, рівняння тора). Параметричні поверхні можна легко обмежити в просторі, задавши межі зміни параметрів. Параметричну форму зручно використовувати при обробці деталей на станках з числовим програмним управлінням. Різець при цьому рухається за законом, що задається в параметричній формі.

Але параметричні поверхні мають і ряд недоліків, наприклад значні обчислювальні затрати, що пояснюється необхідністю застосування числових, а не аналітичних методів. Ілюстрації параметрично заданих поверхонь, як правило, не мають тіні, не передають прозорості і дзеркального відбиття сусідніх об'єктів. Це обумовлено тим, що при параметричному описі вихідна позиція світлового променя, що будує зображення – це точка на об'єкті. А це в свою чергу ускладнює застосування алгоритмів, які припускають іншу позицію променя, наприклад методу трасування променів. Щоб уникнути цього недоліку поверхні апроксимують полігональними сітками, тобто полігональними суміжними фрагментами з поверхонь першого порядку.

Особливий клас поверхонь – це бікубічні поверхні. Вони є простішими серед форм поверхонь, за допомогою яких можна конструювати неперервні складені функції разом з її першими похідними.

8.1. Білінійна та лінійчаста поверхні

Однією з найпростіших моделей поверхонь є білінійна поверхня. Нагадаємо, що пряма лінія, яка визначається двома значеннями $r(0)$ і $r(1)$, задається рівнянням

$$r(t) = tr(0) + (1 - t)r(1), \quad t \in [0, 1].$$

Білінійна поверхня визначається аналогічно на чотирьох кутових точках. Довільна точка на поверхні є лінійною інтерполяцією між протилежними границями чотирикутника, що задається кутовими точками P_{00} , P_{01} , P_{10} , P_{11} :

$$r(u, v) = (1 - u)(1 - v) P_{00} + (1 - u)v P_{01} + u(1 - v) P_{10} + uv P_{11}$$

або в матричній формі

$$r(u, v) = (1 - u, u) \begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix} \begin{pmatrix} 1 - v \\ v \end{pmatrix}, \quad u, v \in [0, 1].$$

Якщо чотири точки P_{00} , P_{01} , P_{10} , P_{11} не лежать в одній площині, то і білінійна поверхня теж не лежить у площині. У загальному випадку білінійна поверхня може бути сильно зігнутою.

Більш загальний випадок білінійної поверхні – це лінійчаста поверхня. Такі поверхні широко застосовують у техніці. *Лінійчастою* називають поверхню, яка може бути утворена рухом прямої лінії за певним законом. В залежності від характеру руху твірної одержують циліндричні, конічні, гвинтові та інші поверхні. Лінійчаста поверхня загального вигляду, що утворюється внаслідок руху прямолінійної твірної вздовж двох напрямних ліній $r_1(v)$, $r_2(v)$, задається рівнянням

$$r(u, v) = r_1(v)(1 - u) + r_2(v)u, \quad u \in [0, 1], \quad v \in [a, b].$$

Якщо обидві криві $r_1(v)$, $r_2(v)$ є відрізками прямих, то $r(u, v)$ описує бі-

лінійну поверхню, якщо одна з кривих $r_1(v)$, $r_2(v)$ вироджується в точку, то одержуємо *секторну* поверхню. Циліндрична поверхня утворюється паралельним зміщенням прямолінійної твірної вздовж деякої кривої.

8.2. Інтерполяційні бікубічні сплайни

Нехай на площині xy задано набір з $(n + 1)(m + 1)$ точок $(x_i, y_j) \in D$, $i = 0, 1, \dots, n$, $j = 0, 1, \dots, m$ ($x_0 < x_1 < \dots < x_n$), де D – деяка прямокутна область. Нехай кожній такій точці в 3D-просторі ставиться у відповідність значення $z_{ij} = f(x_i, y_j)$ деякої функції $f(x, y)$.

Задача полягає в тому, щоб відновити поведінку функції $f(x, y)$ для всіх точок $(x, y) \in D$ шляхом побудови інтерполяційного сплайна.

Означення. Інтерполяційним бікубічним сплайном називають функцію $z = S(x, y)$, що має такі властивості:

- 1) для всіх точок $(x_i, y_j) \in D$, $i = 0, 1, \dots, n$, $j = 0, 1, \dots, m$ справедлива рівність $S(x_i, y_j) = z_{ij}$;
- 2) на кожному прямокутнику $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$, $i = 0, 1, \dots, n - 1$, $j = 0, 1, \dots, m - 1$ $S(x, y)$ є многочленом 3-го степеня, тобто

$$S^{ij}(x, y) = \sum_{l=0}^3 \sum_{k=0}^3 a_{lk}^{ij} (x - x_i)^l (y - y_j)^k ;$$

- 3) на всій області визначення $[x_0, x_n] \times [y_0, y_m]$ функція $z = S(x, y)$ має неперервні другі похідні S''_{x^2} , S''_{y^2} .

Графіком інтерполяційного бікубічного сплайна є поверхня. Її будують аналогічно до 2D-сплайна. Для визначення $16mn$ коефіцієнтів a_{lk}^{ij} розв'язують систему з $16mn$ лінійних рівнянь, для побудови яких використовують табличні значення $S(x_i, y_j) = z_{ij}$, умови неперервності та граничні умови для першої похідної сплайна $S(x, y)$.

Інтерполяційні сплайни проходять через усі опорні точки, однак зміна лише однієї точки, що зустрічається часто на практиці, вимагає перерахунку всього сплайна. Крім цього, на практиці опорні точки часто задаються не точно, тому достатньо вимагати, щоб графік функції проходив поблизу цих точок. У цьому випадку задача інтерполяції зводиться до задачі згладжування.

8.3. Сплайнові поверхні

За заданим масивом вершин побудуємо гладку поверхню, яка б змінювалась плавно, проходила поблизу цих вершин і задовольняла деякі додаткові умови. Точки цього масиву називаються *контрольними* (опорними), а поверхня, що визначається цими точками – *контрольною*.

Для опису складних поверхонь часто використовують сплайнові поверхні. Так називаються поверхні, що складені з елементарних фрагментів, якщо ці фрагменти будуються за єдиною схемою. Сплайн – це спеціальна функція, яка найбільше підходить для апроксимації окремих фрагментів поверхні. Для сплайна можна досить просто обчислювати координати точок, оскільки, як правило, використовують бікубічні сплайни. Найрозповсюдженіші сплайн-поверхні – це поверхня Безьє та В-сплайнові поверхні. Ці поверхні використовуються в тих випадках, коли потрібна поверхня естетичного вигляду, що проходить поблизу цих точок. Спосіб задання згладжувальних поверхонь є аналогічним способу задання згладжувальних кривих.

8.3.1. Поверхні Безьє

Узагальнимо криві Безьє на поверхні Безьє. *Згладжуючі поверхні Безьє* будуються у вигляді тензорного добутку. Так називаються поверхні, що описуються параметричними рівняннями вигляду

$$r(u, v) = \sum_{i=0}^n \sum_{j=0}^m a_i^n(u) b_j^m(v) P_{ij}, \quad i = 0, 1, \dots, n, \quad j = 0, 1, \dots, m, \quad (8.2)$$

де P_{ij} – опорні точки-орієнтири тривимірного простору, $r(u, v) = \text{col}(x(u, v), y(u, v), z(u, v))$. Якщо ці точки відповідним чином з'єднати прямолінійними відрізками, то одержимо контрольну (опорну) багатогранну поверхню (рис. 8.1).

Оскільки співвідношення (8.2) можна переписати у вигляді

$$r(u, v) = \sum_{i=0}^n a_i^n(u) s_i(v),$$

де $s_i(v) = \sum_{j=0}^m b_j^m(v) P_{ij}, \quad i = 0, 1, \dots, n,$

то це означає, що на тривимірний випадок можна перенести багато властивостей і результатів, які одержані для сплайнових кривих.

Якщо за функції $a_i^n(u)$ та $b_j^m(v)$ взяти базисні функції Бернштейна:

$$a_i^n(u) = B_i^n(u) = C_n^i u^i (1-u)^{n-i}, \quad C_n^i = \frac{n!}{i!(n-i)!},$$

$$b_j^m(v) = B_j^m(v) = C_m^j v^j (1-v)^{m-j}, \quad u, v \in [0, 1],$$

то формулою (8.2) задаються поверхні Безьє.

Бікубічний сплайн Безьє відповідає значенням $m = 3, n = 3$ і має вигляд

$$r(u, v) = \sum_{i=0}^3 \sum_{j=0}^3 C_3^i C_3^j u^i (1-u)^{3-i} v^j (1-v)^{3-j} P_{ij},$$

або в матричній формі

$$\begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix} = \begin{pmatrix} 1 & u & u^2 & u^3 \end{pmatrix} M^T \begin{pmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{pmatrix} M \begin{pmatrix} 1 \\ v \\ v^2 \\ v^3 \end{pmatrix}, \quad (8.3)$$

де $M = \begin{pmatrix} 1 & -3 & 3 & 1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ – базова матриця бікубічної поверхні Безьє.

Зазначимо, що для визначення бікубічної поверхні Безьє необхідно задати 16 опорних точок P_{ij} , $i, j = 0, 1, 2, 3$. Для візуалізації поверхні будують лінії рівнів $r(u, v_0)$, $v_0 = \text{const}$, $r(u_0, v)$, $u_0 = \text{const}$, де $v_0, u_0 \in [0, 1]$ (рис. 8.1).

Елементарна бікубічна поверхня Безьє успадковує багато властивостей елементарної кубічної кривої Безьє. Укажемо найосновніші з них.

Бікубічна поверхня Безьє

- лежить в опуклій оболонці опорних точок;
- проходить через точки $P_{00}, P_{01}, P_{10}, P_{11}$ і дотикається до відрізків, що виходять з цих точок і належать контрольному графу масиву P ;
- афінно інваріантна, але проєктивно неінваріантна;

З елементарних фрагментів поверхонь Безьє можна конструювати складені поверхні Безьє, які є C^0 -гладкими поверхнями, коли збігаються чотири контрольні точки двох суміжних фрагментів.

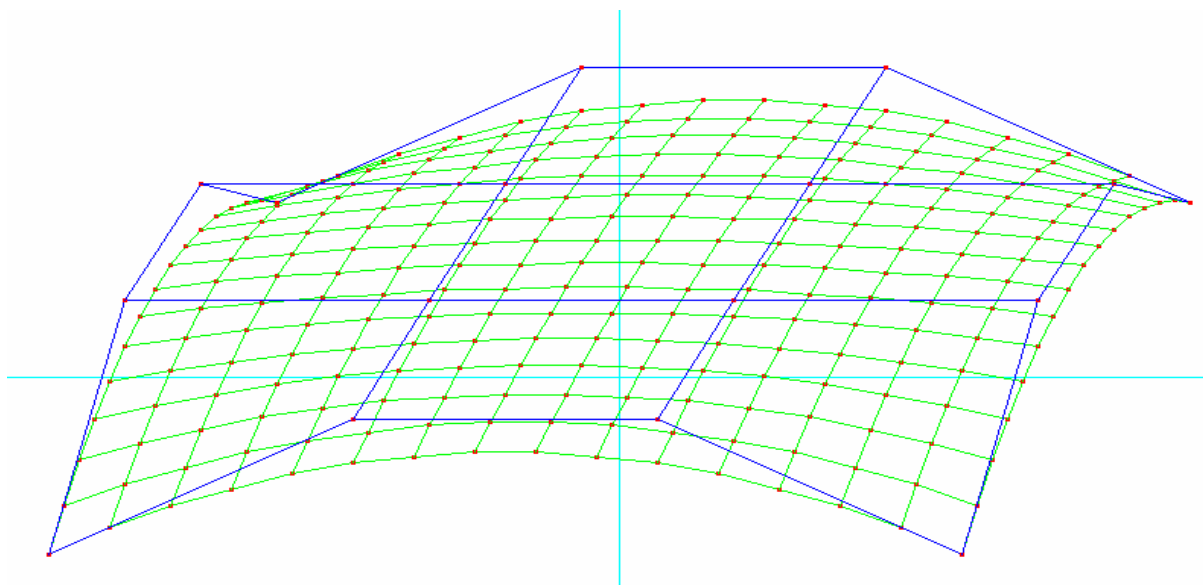


Рис. 8.1. Елементарна бікубічна поверхня Безьє та її контрольна поверхня

8.3.2. В-сплайнові поверхні

Векторне параметричне рівняння елементарного фрагмента бікубічної В-сплайнової поверхні, що задана на масиві точок P_{ij} , $i, j = 0, 1, 2, 3$

має вигляд
$$r(u, v) = \sum_{i=0}^3 \sum_{j=0}^3 n_i^3(u) n_j^3(v) P_{ij}, \quad u, v \in [0, 1],$$

де
$$n_0^3(u) = \frac{1}{6}(1-u)^3, \quad n_1^3(u) = \frac{1}{6}(3u^3 - 6u^2 + 4),$$

$$n_2^3(u) = \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1), \quad n_3^3(u) = \frac{1}{6}u^3, \quad u \in [0, 1].$$

Як і бікубічна поверхня Безьє, елементарна бікубічна В-сплайнова поверхня наслідуює багато властивостей елементарної кубічної В-сплайнової кривої. Об'єднуючи елементарні бікубічні В-сплайни, одержуємо складену В-сплайнову поверхню, для якої на границях двох фрагментів автоматично забезпечується належність класу C^2 .

Раціональна (елементарна) бікубічна В-сплайнова поверхня за заданим масивом точок P_{ij} , $i, j = 0, 1, 2, 3$ визначається рівнянням

$$r(u, v) = \frac{\sum_{i=0}^3 \sum_{j=0}^3 \omega_{ij} n_i^3(u) n_j^3(v) P_{ij}}{\sum_{i=0}^3 \sum_{j=0}^3 \omega_{ij} n_i^3(u) n_j^3(v)}, \quad u, v \in [0, 1], \quad (8.4)$$

де $\omega_{ij} \geq 0$ ($\sum_{i=0}^3 \sum_{j=0}^3 \omega_{ij} > 0$) – вагові коефіцієнти.

Раціональні В-сплайнові поверхні, які в англійській літературі називають ще NURBS-поверхнями, широко застосовуються на практиці, наприклад в інженерних системах для моделювання складних поверхонь зі складними межами та дірками, для моделювання людського тіла та тіла тварин, оскільки ними можна описувати довільні геометричні форми.

Тут ми зупинилися лише на деяких способах побудови згладжувальних поверхонь і побачили, наскільки це не проста задача. Реалістичне зображення гладкої поверхні на екрані дисплея – складна задача, вона полягає в розрахунку її форми на основі правильного визначення освітленості видимих точок поверхні, матеріалу поверхні тощо. Спрощена побудова поверхні $r = r(u, v)$ полягає в зображенні її каркаса – сукупності ліній, що передають форму поверхні. Для цього виконується дискретизація неперервних параметрів

$$u = u_0, u_1, \dots, u_n, \quad v = v_0, v_1, \dots, v_m$$

і організація вкладених циклів виведення каркасних ліній $r(u_i, v)$ і $r(u, v_j)$. Перетин ліній дає комірки трикутної або чотирикутної форми, які апроксимують поверхню.

Контрольні питання та завдання

1. Як аналітично можна задавати поверхні? В чому перевага та недоліки параметричної форми?
2. Наведіть приклади параметрично заданих поверхонь.
3. Які поверхні називаються бікубічними? Які вони мають переваги?
4. Як задати білінійну/лінійчасту поверхню?
5. Як задати секторну поверхню?
6. Назвіть властивості інтерполяційного бікубічного сплайна.
7. Як визначається елементарна бікубічна поверхня Безьє?
8. Які властивості має складена бікубічна поверхня Безьє?
9. При яких умовах на граничні контрольні точки складена поверхня Безьє належить класу C^1 ?
10. Як визначається елементарна бікубічна В-сплайнова поверхня?
11. Які властивості має складена бікубічна В-сплайнова поверхня?
12. Що таке NURBS-поверхні?

Вправи і задачі для самостійного виконання

1. Записати рівняння лінійчастої поверхні, що проходить через діагональ куба і бічне ребро, яке не має з діагоналлю спільних точок.
2. Записати в матричній формі рівняння квадратичної поверхні Безьє. Довести, що ця поверхня проходить через крайні кутові точки заданої матриці точок P_{ij} ($i, j = 0, 1, 2$).
3. Побудувати параметричне рівняння поверхні Безьє за такими контрольними точками
$$\begin{array}{cccc} P_{00}(0,0,0) & P_{01}(1,1,0) & P_{02}(2,1,0) & P_{03}(3,0,0) \\ P_{10}(0,1,1) & P_{11}(1,2,1) & P_{12}(2,2,1) & P_{13}(3,1,1) \\ P_{20}(0,1,2) & P_{21}(1,2,2) & P_{22}(2,2,2) & P_{23}(3,1,2) \\ P_{30}(0,0,3) & P_{31}(1,1,3) & P_{32}(2,1,3) & P_{33}(3,0,3). \end{array}$$
4. Розробити алгоритм обчислення вектора нормалі до поверхні Безьє в точці $P(u, v)$.
5. Записати параметричні рівняння елементарної бікубічної В-сплайнової поверхні в матричній формі.
6. Побудувати параметричне рівняння В-сплайнової поверхні за контрольними точками з прикладу 3.
7. Узагальнити рівняння кривих Ерміта на поверхні Ерміта.
8. Задати квадратичну поверхню в матричному виді. Як зміниться матриця квадратичної поверхні, якщо над нею здійснити афінні перетворення з матрицею F . Записати координати нормалі до поверхні другого порядку.

Тема 9. Основні алгоритми комп'ютерної геометрії

Комп'ютерна геометрія – це сукупність методів і алгоритмів для розв'язування на ЕОМ задач, пов'язаних з різними геометричними побудовами на площині та в просторі.

9.1. Тести орієнтації

9.1.1. Орієнтація нормального вектора

Загальне рівняння прямої має вигляд $Ax + By + C = 0$. Коефіцієнти A , B утворюють вектор нормалі $N = (A, B)$. Вектор $V = (-B, A)$ задає напрямний вектор прямої, оскільки $(N \cdot V) = 0$.

Оскільки $V = N \cdot R(90^\circ) = (A \ B) \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = (-B \ A)$, то це означає, що

при русі точки по лінії в напрямку V вектор N направлений вправо від вектора V (вектор V – вліво від N) (рис. 9.1). Тут $R(\varphi)$ є матрицею повороту точки проти годинникової стрілки на кут φ .

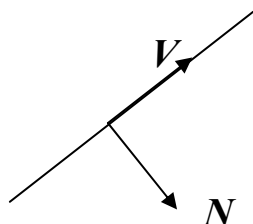


Рис. 9.1. Орієнтація нормалі

Іншими словами, якщо рухатись вздовж вектора (A, B) , то вектор $(B, -A)$ направлений вправо, тому якщо обходити полігон проти годинникової стрілки, то такі нормалі матимуть зовнішню орієнтацію щодо сторін полігона.

9.1.2. Розміщення точки відносно прямої

Функція $f(p) = Ax + By + C$, де $p = (x, y)$, дозволяє визначити орієнтацію двох точок a і b відносно прямої $Ax + By + C = 0$. Так, при $f(a)f(b) > 0$ точки лежать по один бік від прямої, при $f(a)f(b) < 0$ – по різні боки від прямої.

Параметричне рівняння прямої, що проходить через точку $p_0(x_0, y_0)$ в напрямку вектора $V=(V_x, V_y)$, має вигляд

$$x = x_0 + V_x t, \quad y = y_0 + V_y t, \quad \text{або} \quad p = p_0 + Vt, \quad \text{де} \quad p = (x, y).$$

Параметрична форма зручна для побудови відрізків, при цьому необхідно вказати границі зміни параметра t . Значення $t_0 \leq t \leq t_1$ визначають відрізок прямої між точками $p_0 + Vt_0$ та $p_0 + Vt_1$.

Завдяки правій орієнтації вектора N відносно вектора V функція

$$f1(p) = ((p - p_0) \cdot N) = ((p - p_0) \cdot (V_y, -V_x)) = \left| \begin{matrix} p - p_0 \\ V \end{matrix} \right| \text{ дозволяє визначити}$$

положення точки відносно прямої, що проходить через точку p_0 вздовж вектора V (рис. 9.2):

- при $f1(p) > 0$ точка p лежить справа від прямої (кут між векторами $p - p_0$ і N гострий);
- при $f1(q) < 0$ точка q лежить зліва (кут між векторами $q - p_0$ і N тупий).

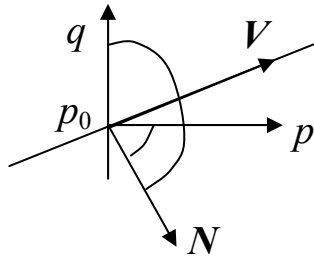


Рис. 9.2. Орієнтація точки

9.1.3. Тест напрямку обходу трьох точок

Як видно з п. 9.1.2 функція

$$f2(p) = \left| \begin{matrix} p - a \\ b - a \end{matrix} \right| = \left| \begin{matrix} x - a_x & y - a_y \\ b_x - a_x & b_y - a_y \end{matrix} \right|$$

дозволяє визначити місцезнаходження точки $p(x,y)$ відносно руху по прямій від точки $a(a_x, a_y)$ до точки $b(b_x, b_y)$:

- при $f2(p) > 0$ точка p лежить справа від прямої;
- при $f2(p) < 0$ точка q лежить зліва від прямої.

Функція $f2(p)$ може бути визначена і в такий спосіб:

$$f2(p) = - \left| \begin{matrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ x & y & 1 \end{matrix} \right|, \text{ оскільки } \left| \begin{matrix} x - a_x & y - a_y \\ b_x - a_x & b_y - a_y \end{matrix} \right| = - \left| \begin{matrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ x & y & 1 \end{matrix} \right|.$$

Отже, якщо задано три точки a, b, c , то визначник

$$D = \left| \begin{matrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{matrix} \right|$$

визначає напрям обходу цих точок:

- якщо $D > 0$, то обхід точок a, b, c здійснюється проти годинникової стрілки;
- якщо $D < 0$, то обхід точок a, b, c здійснюється за годинниковою стрілкою;

- якщо $D = 0$, то точки a, b, c лежать на прямій лінії.

Зауважимо, що визначник D визначає подвоєну орієнтовану площу трикутника, побудованого на точках a, b, c .

9.2. Тест опуклості полігона

В опуклому полігоні всі вершини P_i обходяться за годинниковою стрілкою чи проти годинникової стрілки. Щоб реалізувати тест опуклості полігона, достатньо підрахувати значення визначників

$$D_i = \begin{vmatrix} x_{i-1} & y_{i-1} & 1 \\ x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \end{vmatrix}, i = 2, 3, \dots, n-1, D_1 = \begin{vmatrix} x_n & y_n & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}, D_n = \begin{vmatrix} x_{n-1} & y_{n-1} & 1 \\ x_n & y_n & 1 \\ x_1 & y_1 & 1 \end{vmatrix}.$$

Якщо всі D_i одного знака, то полігон опуклий, інакше – неопуклий.

9.3. Тести орієнтації точки відносно полігона

Відомі кілька методів перевірки довільної точки на приналежність її полігону. На початковій стадії розв'язування цієї задачі часто застосовують габаритний тест, який іноді ще називають методом оболонки. Цей метод є досить універсальним методом прискорення обчислювальних процесів у задачах комп'ютерної графіки.

9.3.1. Габаритний тест

Суть цього методу полягає в тому, що спочатку аналізуються не самі полігони, а більш прості форми – оболонки, які охоплюють полігони (в нашому випадку – прямокутники). Використання цього методу дозволяє відразу виявити деякі випадки, в яких точка не належить полігону.

Нехай задано полігон $P = \{P_1 P_2 \dots P_n P_1\}$ та точка $Q(x, y)$. Визначимо мінімальні та максимальні координати вершин P_i ($i = 1, 2, \dots, n$), тобто мінімальну прямокутну оболонку, якій належить полігон. Позначимо $p_{x\min} = \min\{p_{ix}\}$, $p_{x\max} = \max\{p_{ix}\}$, $p_{y\min} = \min\{p_{iy}\}$, $p_{y\max} = \max\{p_{iy}\}$. Тоді умови $(x < p_{x\min}) \cup (x > p_{x\max}) \cup (y < p_{y\min}) \cup (y > p_{y\max})$ гарантують неналежність точки Q полігону P .

Повністю габаритний тест задачу розміщення точки відносно полігона не розв'язує, він розпізнає лише точки, які лежать зовні габаритного прямокутника. Якщо точка лежить усередині габаритного прямокутника, то вона не може бути однозначно ідентифікована як зовнішня, внутрішня чи гранична відносно полігона.

Проте, завдяки своїй простоті, габаритний тест застосовується в багатьох алгоритмах для швидкого розпізнавання неперетинних геометричних об'єктів. Для цього навколо кожного об'єкта описують тіло простого вигляду, наприклад прямокутник, паралелепіпед тощо, і якщо ці тіла не перетинаються, то і об'єкти, що містяться в них, теж не

перетинаються. Однак слід зауважити, що, якщо тіла перетинаються, то самі об'єкти при цьому не обов'язково перетинаються.

9.3.2. Тест, що визначає орієнтацію точки відносно кожного ребра

Цей алгоритм придатний тільки для опуклих полігонів. У процесі роботи цього алгоритму обходиться границя полігона і проводиться покроковий аналіз взаємного розміщення точки Q і кожного ребра. Оскільки полігон P опуклий, то точка Q лежить усередині полігона тільки тоді, коли вона лежить зліва від кожного ребра при обході полігона проти годинникової стрілки, і справа при обході за годинниковою стрілкою.

9.3.3. Променевий тест

Для розв'язування задачі орієнтації точки $Q(x, y)$ відносно полігона P випускаємо з точки Q у довільному напрямку V промінь $p = Q + Vt$, $t \geq 0$ і обчислюємо кількість точок перетину цього променя з границею полігона. Якщо відкинути випадок, коли промінь проходить через вершини полігона, то розв'язування задачі тривіальне – точка лежить усередині, якщо загальна кількість точок перетину променя з ребрами полігона непарна, і зовні, якщо кількість точок перетину парна.

Зрозуміло, що для довільного полігона завжди можна побудувати промінь, що не проходить через жодну з вершин. Однак побудова такого променя пов'язана з деякими труднощами і, крім цього, перевірку перетину границі полігона з довільним променем провести складніше, ніж, наприклад, із горизонтальним.

Розглянемо різні випадки перетину горизонтального променя з вершинами полігона (рис. 9.3). У випадку *a*, коли ребра, що виходять із вершини, лежать по один бік від променя, парність кількості перетинів не змінюється.

У випадку *б*, коли ребра, що виходять із вершини, лежать по різні сторони від променя, парність кількості перетинів змінюється.

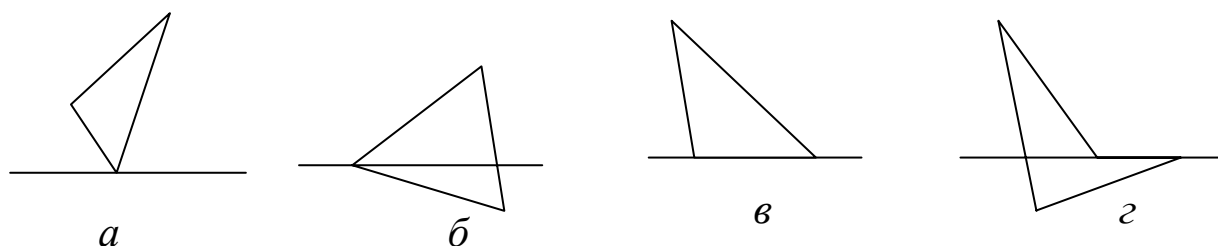


Рис. 9.3. Різні випадки перетину променя з полігоном

До випадків *в* і *г*, коли промінь зустрічається з горизонтальним ребром, такий підхід застосувати не можна.

Отже, у випадку, коли полігон не має горизонтальних ребер, має-

мо такий алгоритм. Випускаємо з точки Q горизонтальний промінь і всі ребра полігона перевіряємо на перетин із цим променем. Якщо промінь проходить через вершину (формально перетинає 2 ребра, що виходять із цієї вершини), зараховуємо перетин тільки для тих ребер, для яких ця вершина верхня.

Зауваження 2. У випадках б, г, коли промінь зустрічається з горизонтальним ребром $P_i P_{i+1}$, необхідно перевірити, чи вершини P_{i-1} та P_{i+2} лежать по різні боки від променя. Якщо ці вершин лежать по різні боки від променя, то перетин зараховуємо; якщо по один бік, то перетин не зараховуємо.

9.3.4. Кутовий тест

Кутовий тест займає важливе місце в задачах комп'ютерної графіки. Кутовий тест базується на обчисленні й аналізі алгебраїчної суми кутів $\delta_i = \angle(V_i, V_{i+1})$ між суміжними векторами $V_i = P_i - Q$, що з'єднують точку Q з вершинами P_i при обході полігона P по замкнутому контуру.

Тест ґрунтується на такому факті: спостерігач, що проглядає вершини полігона з внутрішньої точки Q_6 , здійснює навколо себе повний оберт (рис. 9.4, а), а з зовнішньої точки Q_3 – жодного оберту (рис. 9.4, б).

Тоді при всіх ненульових векторах V_i радіанний варіант кутового тесту матиме вигляд

$$\left| \sum_{i=1}^n \text{ang}(V_i, V_{i+1}) \right| = \begin{cases} 0 < \pi, & Q_3 \notin P, \\ 2\pi > \pi, & Q_3 \in P, \end{cases} \quad (9.1)$$

де $\text{ang}(V, W)$ – це функція обчислення кута між векторами V і W , як кута найкоротшого повороту від V до W , тобто

$$\text{ang}(V, W) = \text{if} \left(d = \frac{|V|}{|W|} \neq 0 : \text{sign}(d); 1 \right) \arccos \left(\frac{V \cdot W}{|V| \cdot |W|} \right).$$

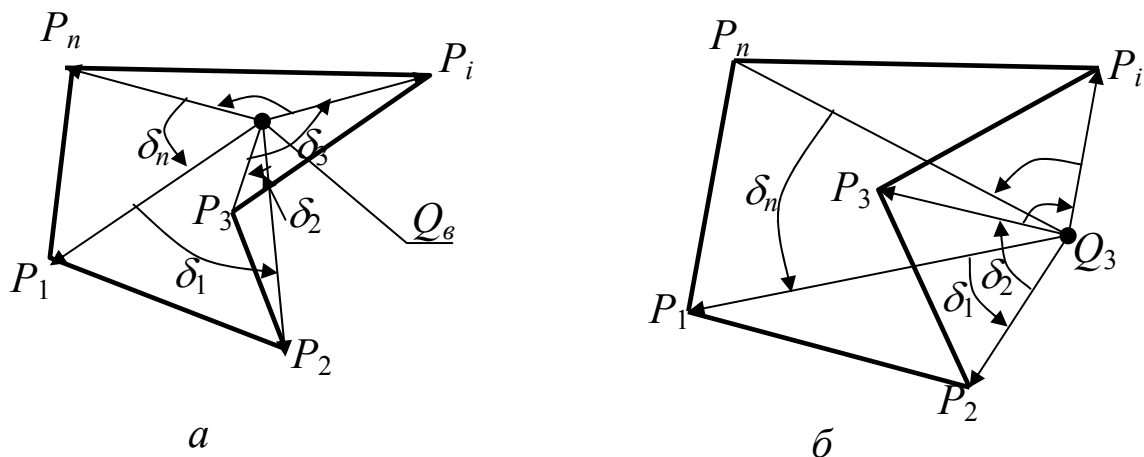


Рис. 9.4. Кутовий тест

У зв'язку з неминучими обчислювальними похибками, що виникають при знаходженні суми кутів, результат у (9.1) не можна порівнювати ні з нулем, ні з 2π (він завжди буде відрізнятися від цих чисел). Тому для суми кутів пропонують взяти порогове значення π і тоді при будь-яких похибках одержимо правильний результат тестування.

Гранична точка полігона розрізняється так:

- якщо при розрахунку векторів матимемо нульовий вектор, то точка збігається з вершиною полігона;
- якщо при розрахунку кутів δ_i буде одержаний розгорнутий кут з модулем $|\delta_i| = \pi$, тоді точка лежить на ребрі. Однак оскільки обчислення здійснюються з плаваючою крапкою, то належність точки Q ребру $P_i P_{i+1}$ за допомогою умови $|\delta_i| = \pi$ на практиці замінюється умовою $\| |\delta_i| - \pi \| < \varepsilon$, де ε – експериментально підібрана мінімальна константа.

Зауважимо, що для того, щоб не проводити розрахунок кутів із високою точністю, існують відповідні модифікації кутового тесту.

Значно спрощується тест орієнтації точки Q відносно трикутника $P_1 P_2 P_3$ [14]. Для цього достатньо записати розклад вектора QP_1 за векторами сторін трикутника, тобто

$$q - p_1 = Vt + W\tau, \quad (9.2)$$

де $V = p_2 - p_1$, $W = p_3 - p_1$.

Далі систему (9.2) необхідно розв'язати відносно τ , t або скористатись методом Грамма і скласти нову систему двох лінійних рівнянь зі скалярних добутків

$$\begin{aligned} (q - p_1)V &= t(V \cdot V) + \tau(W \cdot V), \\ (q - p_1)W &= t(V \cdot W) + \tau(W \cdot W) \end{aligned}$$

та знайти розв'язок

$$(t, \tau) = ((q - p_1)V \ (q - p_1)W) \begin{bmatrix} (V \cdot V) & (V \cdot W) \\ (V \cdot W) & (W \cdot W) \end{bmatrix}^{-1}. \quad (9.3)$$

Тоді при $\{t > 0\} \cap \{\tau > 0\} \cap \{t + \tau < 1\}$ точка лежить усередині трикутника, при $\{t < 0\} \cup \{\tau < 0\} \cup \{t + \tau > 1\}$ – зовні трикутника, а в решті випадків – на його границі.

Матриця, що фігурує в (9.3), невироджена, оскільки визначник

$$|V|^2 \cdot |W|^2 - (V \cdot W)^2 = |V|^2 \cdot |W|^2 \sin^2(\angle(V, W)) > 0$$

і дорівнює нулю лише при колінеарності вершин трикутника $P_1 P_2 P_3$.

9.4. Тести перетину

9.4.1. Тест перетину прямої з полігоном

Пряма перетинає полігон, якщо існує хоча б одна пара вершин полігона, що лежать по різні боки від цієї прямої (рис. 9.5).

Зазначимо, що необхідно порівнювати розміщення відносно прямої лінії $f(p) = 0$ не тільки суміжні вершини P_i та P_{i+1} полігона, а й решту пар вершин $\{P_i, P_j\}$, оскільки пряма може перетинати полігон тільки у вершинах (рис. 9.5, б). Різні знаки чисел $f(p_i)$ і $f(p_j)$ дають умову перетину прямої $f(p) = 0$ з полігоном P .

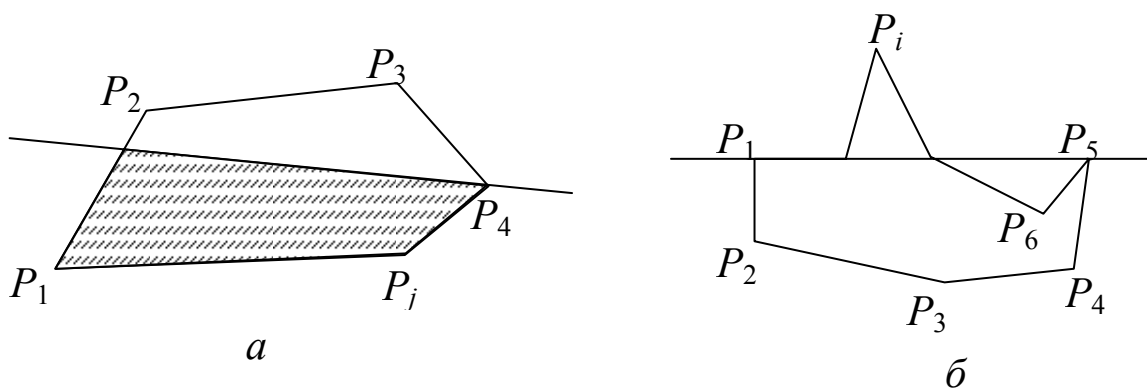


Рис. 9.5. Перетин прямої з полігоном

Але кількість пар вершин полігона дорівнює $\frac{n(n-1)}{2}$. Якщо розв'язувати цю задачу прямим шляхом, то при великих n (наприклад, n – число порядку тисяч) кількість операцій порівнянь недопустимо велика (пропорційна n^2). За допомогою спеціальних прийомів цю кількість операцій можна зменшити. Розглянемо деякі з цих алгоритмів.

- У зовнішньому циклі $i = 1, 2, \dots, n$ обчислюємо значення $s_i = f(p_i)$. У внутрішньому циклі $j = 1, 2, \dots, i - 1, i > 1$ аналізуємо знак добутку $s_i \cdot s_j$ (зауважимо, що $s_j = f(p_j)$ для $j < i$ вже обчислено). При $s_i \cdot s_j < 0$ тест негайно завершується, повертаючи ознаку перетину прямої з полігоном, інакше після нормального завершення зовнішнього циклу одержуємо або відсутність перетину, або дотик прямої з полігоном.
- У циклі по $i = 1, 2, \dots, n$ обчислимо значення $s_i = f(p_i)$ і виконаємо сортування вектора s . Результат тестування задається числом $res = \text{sign}(s_1 \cdot s_n)$. При $res = -1$ крайні елементи мають різні знаки, що означає перетин із полігоном, при $res = 0$ пряма дотикається до полігона, при $res = 1$ відсутній перетин прямої з полігоном. Недолік цього методу – наявність операції сортування, в якій кількість операцій порівняння оцінюється величиною $n \lg n$, що менше n^2 .

- Ще один варіант алгоритму тесту перетину прямої з багатокутником використовує три прапорці, що сигналізують про розміщення вершини зліва (прапорець l), справа (прапорець r) і точно на прямій (прапорець e). На початку всі прапорці виставляємо в нуль. У циклі по $i = 1, 2, \dots, n$ обчислюємо значення $s = f(p_i)$ і виставляємо в 1 один із прапорців: $l = 1$, якщо $s < 0$; $e = 1$, якщо $s = 0$; $r = 1$, якщо $s > 0$. З допомогою умови $l \cdot r \neq 0$ перевіряємо наявність пари вершин полігона, що знаходяться по різні боки від прямої. Якщо ця умова виконується, то тест негайно завершується, повертаючи 1 – ознаку перетину прямої з полігоном. Після нормального закінчення циклу обходу вершин алгоритм повертає значення $e-1$, яке дорівнює 0, якщо пряма дотикається до полігона, та -1 , якщо відсутній перетин прямої з цим полігоном. Блок-схему цього алгоритму [21] подано на рис. 9.6.

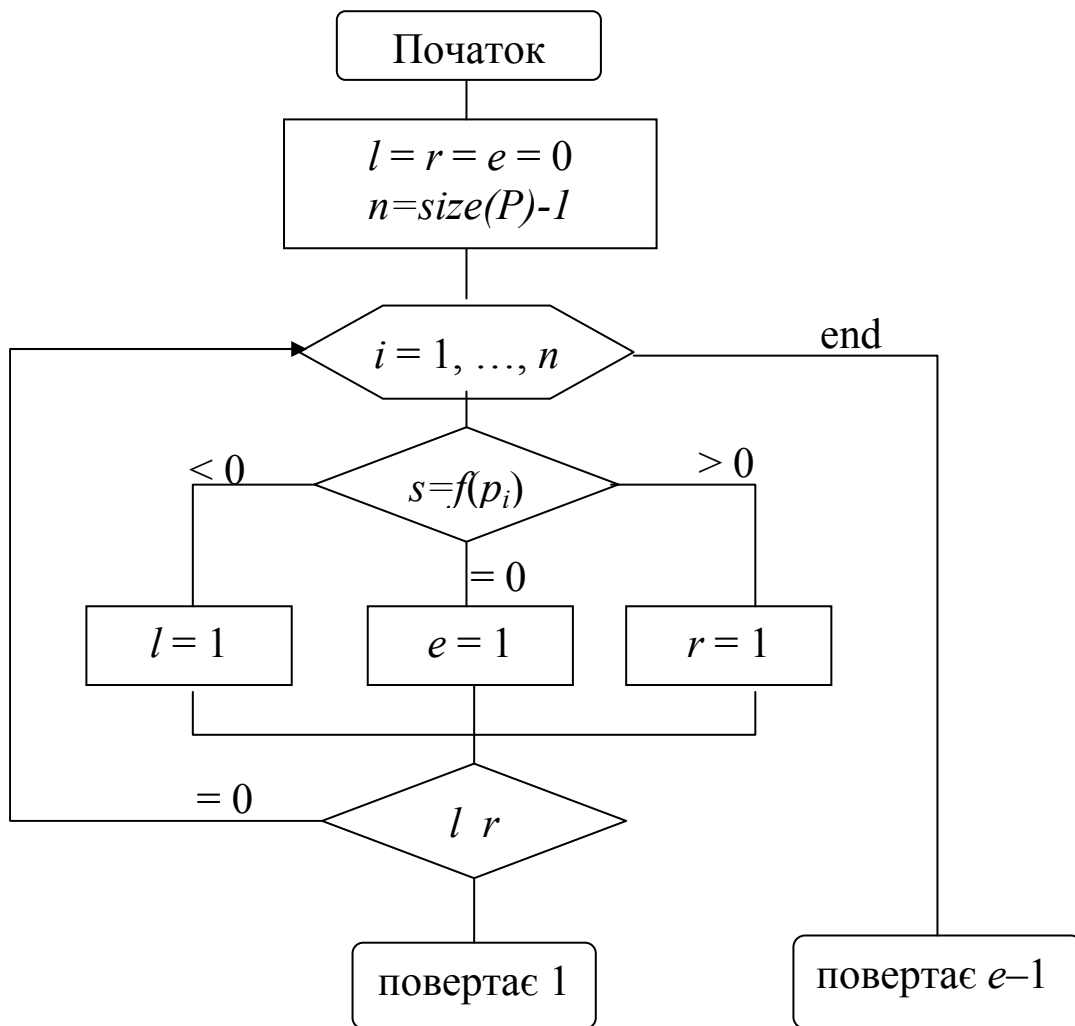


Рис. 9.6. Блок-схема тесту перетину прямої з полігоном

9.4.2. Тест перетину відрізків

Нехай задано відрізки $[a, b]$ та $[c, d]$ параметричними рівняннями

$$\begin{aligned}x &= x_a + (x_b - x_a)t, & x &= x_c + (x_d - x_c)\tau, \\y &= y_a + (y_b - y_a)t, & y &= y_c + (y_d - y_c)\tau, \quad t \in [0, 1]; \quad \tau \in [0, 1].\end{aligned}$$

З системи

$$\begin{cases}x_a + (x_b - x_a)t = x_c + (x_d - x_c)\tau \\y_a + (y_b - y_a)t = y_c + (y_d - y_c)\tau\end{cases}$$

знаходимо

$$(t \ \tau) = (x_c - x_a \ y_c - y_a)M^{-1}, \quad \text{де } M = \begin{pmatrix} x_b - x_a & x_c - x_d \\ y_b - y_a & y_c - y_d \end{pmatrix}$$

Якщо $|M| = 0$, то відрізки паралельні. У випадку непаралельності відрізків умови $0 \leq t \leq 1$, $0 \leq \tau \leq 1$ визначають існування точки перетину двох відрізків.

Алгоритм розрахунку точки перетину відрізків ab та cd , повертає ознаку перетину:

- -1 , якщо прямі відрізків паралельні;
- 0 , якщо непаралельні відрізки не перетинаються;
- 1 , якщо відрізки перетинаються

і має такий вигляд [21]:

`cross_segm(ab, cd, t, tau, q)`

$V = b - a$, $M = \begin{pmatrix} V \\ c - d \end{pmatrix}$ // матриця направляючих векторів

якщо $|M| = 0$, то повертаємо -1 // тест паралельності прямих

$(t \ \tau) = (c - a)M^{-1}$ // параметри перетину прямих

$q = a + Vt$ // точка перетину прямих

повертаємо $(0 \leq t \leq 1) \cap (0 \leq \tau \leq 1)$ // ознака перетину відрізків

Цей алгоритм вимагає максимально 11 операцій додавання і 10 операцій множення/ділення.

9.5. Алгоритми відсікання

Процес виділення тієї частини геометричного об'єкта, яка лежить поза видимою областю (вікном), називається *відсіканням* (*clipping*).

Задачі відсікання видимого зображення по границях деякої області в комп'ютерній графіці зустрічаються досить часто. Вони є головними геометричними задачами при візуалізації геометричних об'єктів. У простіших випадках областю відсікання виступає прямокутник (рис. 9.7).

Часто графічні зображення складаються із сукупності відрізків. Для задачі відсікання за допомогою алгоритму `cross_seg` можна було б розрахувати ті частини відрізків, які знаходяться зовні вікна. Однак це потребує виконання великої кількості відповідних обчислень. Графічні об'єкти складаються з відрізків трьох типів: цілком видимі, цілком невидимі і частково видимі (рис. 9.7).

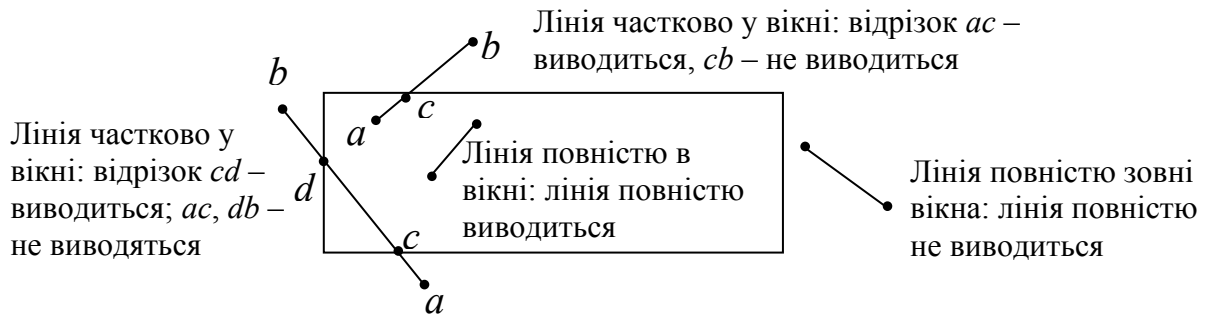


Рис. 9.7. Двовимірне відсікання прямокутним вікном

9.5.1. Двовимірний алгоритм Сазерленда-Коена

Алгоритм Сазерленда-Коена достатньо просто й ефективно дозволяє прийняти або відсікти відрізки цілком відносно довільного прямокутника. У цьому алгоритмі для двох кінців відрізка будуються 4-бітні коди (зліва направо):

- біт 0 визначає чи точка лежить лівіше вікна;
- біт 1 визначає чи точка лежить вище вікна;
- біт 2 визначає чи точка лежить правіше вікна;
- біт 3 визначає чи точка лежить нижче вікна.

Біт набуває одиничного значення, якщо умова істинна, і нуль, якщо умова хибна. Тобто площина з вікном чотирма прямими розбивається на 9 областей, у кожній з яких є свій код (рис. 9.8). Ці коди використовуються для визначення чи знаходиться відрізок повністю у вікні чи зовні вікна.

Отже, маємо такий алгоритм відсікання відрізків.

- Формуємо коди (*код1* та *код2*) для кінців відрізка.
- Якщо два коди для кінців відрізка дорівнюють 0000, то відрізок повністю знаходиться у вікні.
- Відрізок цілком відсікається, якщо *код1* and *код2* \neq 0000.

0011	0010	0110
0001	0000	0100
1001	1000	1100

Рис. 9.8. Чотирибітні коди

- Якщо *код1* and *код2* = 0000, то відрізок не можна ні відсікти, ні прийняти. У цьому випадку необхідно шукати точки перетину відрізка з

ребрами вікна, причому оскільки відомо, в які області попадають кінці відрізка, то, аналізуючи коди кінців відрізка, легко зрозуміти, з якими саме ребрами вікна може перетинатися відрізок.

9.5.2. Відсікання відрізка опуклим полігоном

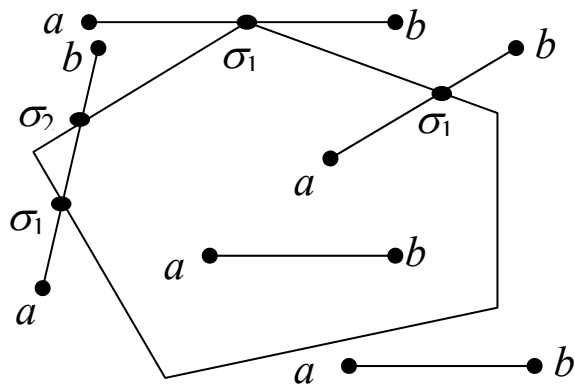


Рис. 9.9. Відсікання полігоном

Розглянемо задачу зовнішнього відсікання відрізка опуклим полігоном $P = \{P_1 P_2 \dots, P_n P_1\}$.

Необхідно відсікти ті частини відрізка, які знаходяться зовні полігона (рис. 9.9).

Заропонований алгоритм базується на розрахунках перетину прямої ab із відрізками $P_i P_{i+1}$ — сторонами полігона.

Алгоритм повертає через аргумент ab фрагмент відрізка, що потрапляє всередину полігона й ознаку видимості відрізка ab у вікні полігона P , що приймає значення

- 0, якщо відрізок повністю знаходиться зовні полігона і відсікається цим полігоном;
- 1, якщо відрізок повністю або частково розміщений усередині полігона.

Робота алгоритму починається з ініціалізації лічильника перетинів $k = 0$. Далі виконується обхід ребер полігона $P_i P_{i+1}$ для $i = 1, 2, \dots, n$ ($P_{n+1} = P_1$) і розрахунок перетину з ними не самого відрізка ab , а прямої, що містить цей відрізок (це можна зробити за допомогою функції `cross_segm`).

Якщо пряма перетинає ребро (це розпізнається за умовою $0 \leq \tau \leq 1$), то лічильник перетинів k інкрементується, а параметр t зберігається у змінній σ_k , $k = 1, 2$. Якщо знайдено два перетини ($k = 2$), то подальший обхід полігона припиняється (саме тому розраховується перетин ребер полігона з прямою, що містить відрізок ab , а не з самим відрізком).

Кінцеві точки видимої частини відрізка $[a, b]$ визначаються за допомогою значень $\sigma_1 = t_i$, $\sigma_2 = t_j$ параметра t , при яких пряма перетнула ребра $P_i P_{i+1}$ та $P_j P_{j+1}$:

- значення $\{\sigma_1 < 0, \sigma_2 > 1\}$ і $\{\sigma_1 > 1, \sigma_2 < 0\}$ відповідають розміщенню всього відрізка $[a, b]$ у вікні полігона і відсікання не виконується;
- значення $\{\sigma_1 < 0, \sigma_2 < 0\}$ або $\{\sigma_1 > 1, \sigma_2 > 1\}$ повністю відсікають відрізок;

- значення $\{0 \leq \sigma_1 \leq 1, \sigma_2 < 0\}$ повертають відрізок $[a, a + V\sigma_1]$;
 - значення $\{0 \leq \sigma_1 \leq 1, \sigma_2 > 1\}$ повертають відрізок $[a + V\sigma_1, b]$;
 - значення $\{\sigma_1 < 0, 0 \leq \sigma_2 \leq 1\}$ повертають відрізок $[a, a + V\sigma_2]$;
 - значення $\{\sigma_1 > 1, 0 \leq \sigma_2 \leq 1\}$ повертають відрізок $[a + V\sigma_2, b]$;
 - значення $\{0 \leq \sigma_1 \leq 1, 0 \leq \sigma_2 \leq 1\}$ стягують кінцеві точки відрізка $[a, b]$ на границю полігона – в точки $a + V\sigma_1$ і $a + V\sigma_2$, $V=b-a$.
- Блок-схему цього алгоритму подано на рис. 9.10 [21].

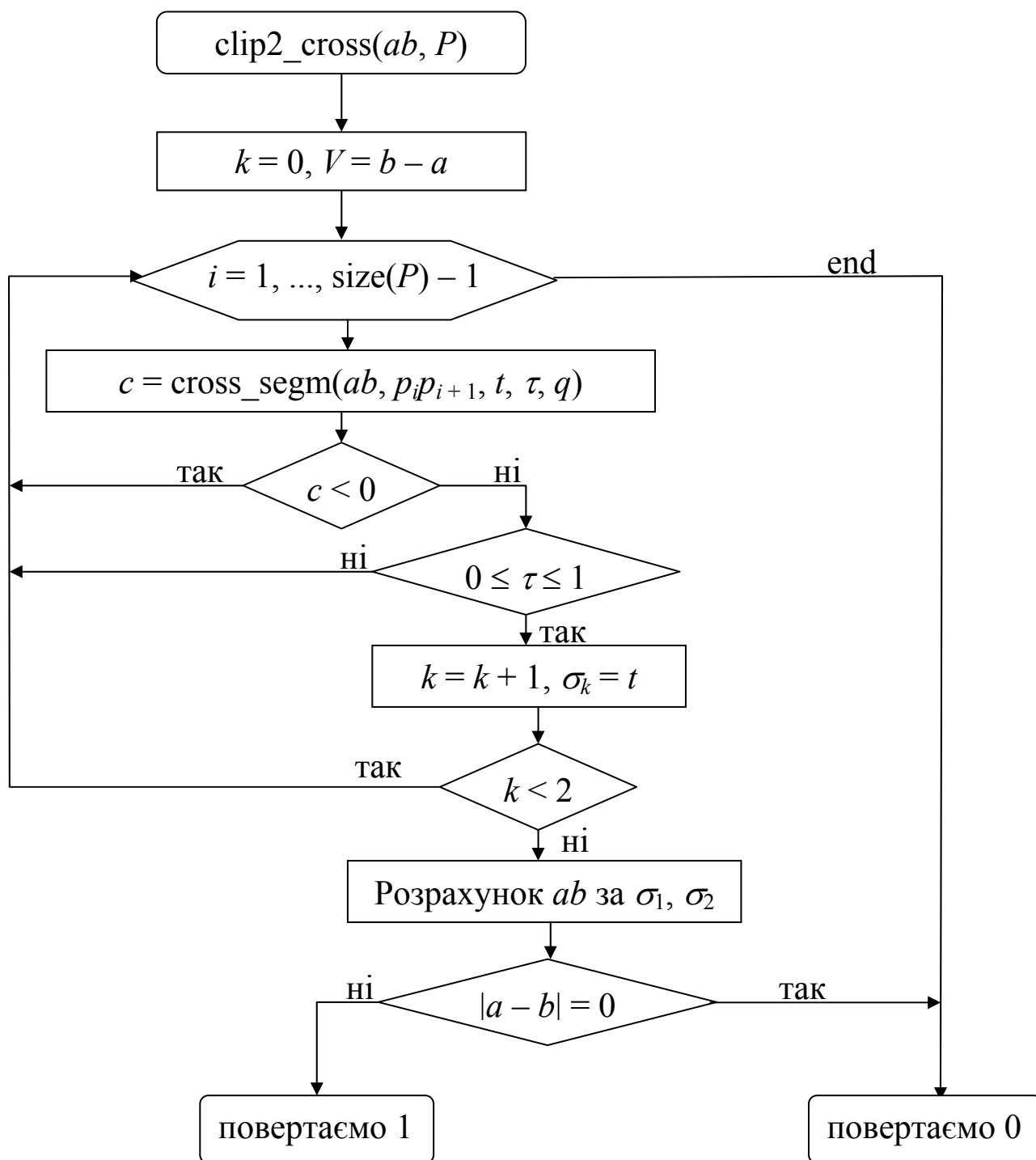


Рис. 9.10. Блок-схема алгоритму відсікання

9.5.3. Перетин та об'єднання опуклих полігонів

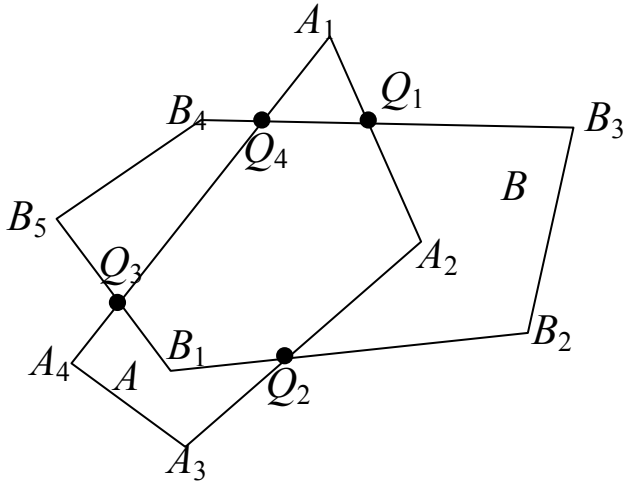


Рис. 9.11. Перетин полігонів

Розглянемо задачу побудови перетину опуклих полігонів $A = \{A_1 A_2 \dots A_n A_1\}$ та $B = \{B_1 B_2 \dots B_m B_1\}$. Необхідно знайти спільну частину полігонів, що належить полігонам A та B , тобто їхній логічний добуток $A \cap B$.

Алгоритм знаходження перетину полігонів пояснимо на прикладі опуклих полігонів, які зображені на рис. 9.11.

- Спочатку функцією $\text{clip2_cross}(A_i A_{i+1}, B)$, $i = 1, 2, \dots, n$ виконуємо зовнішнє відсікання ребер полігона A полігоном B . У результаті одержуємо список відрізків $Ls = \{Q_1 A_2, A_2 Q_2, Q_3 Q_4\}$.
- Далі функцією $\text{clip2_cross}(B_j B_{j+1}, A)$, $j = 1, 2, \dots, m$ виконуємо зовнішнє відсікання ребер полігона B полігоном A і додаємо в список Ls нові відрізки $B_1 Q_2, Q_1 Q_4, Q_3 B_1$.
- З елементів одержаного списку

$$Ls = \{Q_1 A_2, A_2 Q_2, Q_3 Q_4, B_1 Q_2, Q_1 Q_4, Q_3 B_1\}$$

збираємо замкнутий контур шляхом з'єднання відрізків збіжними кінцями. Для цього беремо перший відрізок $Q_1 A_2$ і серед інших елементів списку Ls шукаємо відрізок, кінець якого збігається з A_2 . Це відрізок $Q_2 A_2$, який разом із відрізком $Q_1 A_2$ утворює ламану $C = \{Q_1 A_2 Q_2\}$. Далі знаходимо продовження ламаної $B_1 Q_2$ і одержуємо ламану $C = \{Q_1 A_2 Q_2 B_1\}$ і т.д., поки не одержимо замкнений контур $C = \{Q_1 A_2 Q_2 B_1 Q_3 Q_4 Q_1\}$. Якщо полігони не перетинаються, то після виконання зовнішнього відсікання список Ls буде порожнім.

9.6. Інші алгоритми відсікання відрізків

В багатьох випадках більшість відрізків лежить цілком всередині або зовні відсікаючого вікна, тому необхідно мати можливість якомога раніше без складних обчислень визначити, які з відрізків цілком видимі, а які цілком невидимі і їх можна зразу відкинути. Задачу відсікання можна розв'язувати різними методами.

За способом розв'язування цієї задачі всі алгоритми відсікання поділяються на 2 класи [7]:

- алгоритми, що використовують коди кінців відрізка або самого відрізка;
- алгоритми, що використовують параметричне задання самих відрізків і сторін вікна.

До першого класу алгоритмів належить алгоритми Сазерленда-Коена, *FC*-алгоритм, до другого класу – алгоритм Кіруса-Бека, Ейлера-Азертонна і більш пізній алгоритм Ліанга-Барскі та ін.

Алгоритми з кодуванням застосовуються до прямокутних вікон, сторони яких паралельні осям координат, а алгоритми з параметричним заданням – до будь-яких вікон.

Вище ми вже розглянули алгоритм Сазерленда-Коена, який є стандартом для алгоритмів відсікання. Розглянемо тепер *FC*-алгоритм.

9.6.1. Двовимірний *FC*-алгоритм

В цьому алгоритмі кодуються не кінці відрізка, а весь відрізок цілком. Схема кодування близька до тієї, що використовується в алгоритмі Сазерленда-Коена.

Площина з вікном розбивається на 9 областей і вони кодуються цифрами від 1 до 9 (рис. 9.12). Область з вікном має код 5.

1	2	3
4	5	6
7	8	9

Рис. 9.12. Коди областей для *FC*-алгоритму

Відрізок кодується наступним чином:

$$LineCode(ab) = Code(a) * 10 + Code(b),$$

$Code(a)$ – код області з початковою точкою відрізка,

$Code(b)$ – код області з кінцевою точкою відрізка,

$LineCode(ab)$ – код відрізка.

Оскільки кожний код може набувати дев'ять значень, то існує 81 можливий варіант розміщення відрізка і для кожного варіанту, тобто для кожного коду відрізка необхідно мати свій набір обчислень для відсікання відрізка.

Якщо $Code(a) = Code(b)$, то $LineCode(ab) = LineCode(ba)$. Існує 9 таких випадків: (1 – 1), (2 – 2), ..., (9 – 9).

У випадку (5 – 5) (код відрізка 55) відрізок повністю видимий, у решті випадків – частково або повністю невидимий.

Отже, кількість різних варіантів тепер зменшується до 72, але і серед них існує всього 6 основних випадків відсікання, решта – симетричні їм. Для того, щоб виділити ці 6 випадків, потрібно розглянути різні варіанти розміщення відрізка для некутових та кутових областей. Ілюстрації можливих варіантів розміщення відрізка для перших п'яти випадків наведені на рис. 9.13, для шостого випадку – на рис. 9.14.

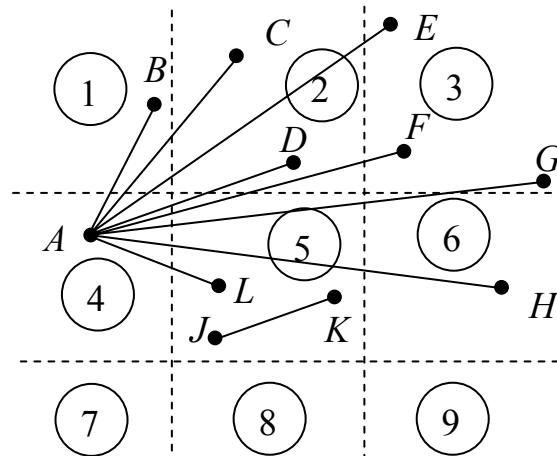


Рис. 9.13. Варіанти розміщення відрізка для некутових областей

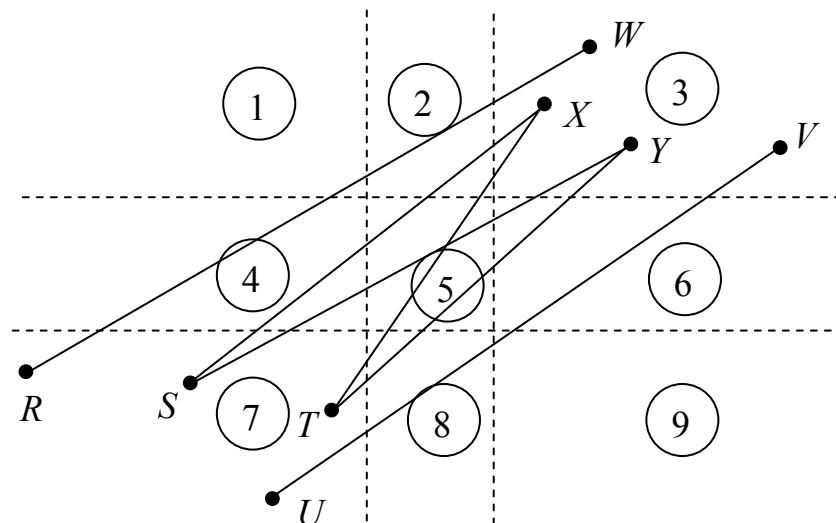


Рис. 9.14. Варіанти розміщення відрізка для кутових областей

Випадок 1 – початкова точка відрізка знаходиться в області 4, а кінцева – в області 1 (відрізок AB , $LineCode = 41$). Відрізок не перетинає видиму область і відкидається.

Випадок 2 – початкова точка відрізка знаходиться в області 4, кінцева – в області 2. Це відрізки AC і AD , $LineCode = 42$. Вони перетинають ліву межу вікна. Знайдемо точки перетину відрізків AC і

AD з лівою межею вікна – $(x_{лів}, y_n)$. Якщо $y_n > y_{верхн}$, то це відрізок AC , він не перетинає відрізка з лівим ребром видимої області й відкидається. Якщо $y_n < y_{верхн}$, то це відрізок AD , що перетинається також з верхньою межею вікна. Тому необхідно обчислити точку перетину відрізка AD з верхньою межею вікна й зобразити видиму частину відрізка.

Випадок 3 – початкова точка відрізка знаходиться в області 4, кінцева – в області 3 (відрізки AE , AF і AG , $LineCode = 43$). Обчислюємо точки перетину відрізків з лівим ребром вікна $(x_{лів}, y_n)$. Якщо $y_n > y_{верхн}$, то це відрізок AE . Він не перетинає видиму область і відкидається, інакше це відрізки AF і AG . Для них обчислюємо точку перетину з правим ребром $(x_{пр}, y_n)$. Якщо $y_n > y_{верхн}$, то це відрізок AF , тому необхідно обчислити точку перетину з верхнім ребром вікна і зобразити видиму частину відрізка. Інакше це відрізок AG , для нього вже точки перетину з ребрами вікна знайдені, залишається зобразити лише видиму частину відрізка.

Випадок 4 – початкова точка відрізка знаходиться в області 4, кінцева – в області 6 (відрізок AH , $LineCode = 46$). Обчислюємо точки перетину відрізка з лівим і правим ребрами вікна і зображаємо видиму частину відрізка.

Випадок 5 – початкова точка – в області 4, кінцева – в області 5 (відрізок AL , $LineCode = 45$). Обчислюємо точку перетину відрізка з лівим ребром і зображаємо видиму частину відрізка.

Випадок 6 – початкова точка знаходиться в області 7, кінцева – в області 3 (відрізки RW , SX , SY , TX , TU , UV , $LineCode = 73$). Обчислюємо точку перетину відрізка з лівим ребром $(x_{лів}, y_{n1})$. Якщо $y_{n1} > y_{верхн}$, то це відрізок RW і він відкидається. Обчислюємо точку перетину з правим ребром $(x_{пр}, y_{n2})$. Якщо $y_{n2} < y_{нижн}$, то це відрізок UV , він теж відкидається. При $y_{n1} > y_{нижн}$ це або відрізок SX , або SU . Далі, якщо $y_{n2} < y_{верхн}$, то це відрізок SU і можемо зобразити його видиму частину. Інакше це відрізок SX , тому необхідно обчислити точку перетину з верхнім ребром вікна і зобразити його видиму частину. Залишилося розглянути відрізки TX , TU . Якщо $y_{n2} < y_{верхн}$, то це відрізок TU і ми можемо зобразити його видиму частину. Інакше це відрізок TX , тому обчислюємо для нього точку перетину з верхньою стороною вікна і теж відображаємо його видиму частину.

Зауваження. Для більш швидкої роботи алгоритму можна використовувати коди в шістнадцятковій системі числення, що визначаються зі співвідношення

$$LineCode(ab) = Code(a) shl 4 + Code(b).$$

Вказівка. Для побудови програми можна скористатися оператором *switch* і задати дії для кожного з 81 випадків.

9.6.2. Алгоритм Кіруса-Бека

Попередні алгоритми виконували відсікання прямокутним вікном, сторони якого паралельні осям координат. Однак в багатьох випадках потрібне відсікання довільним многокутником, наприклад в алгоритмах усунення невидимих частин сцени.

Алгоритм Кіруса-Бека реалізує відсікання довільним опуклим вікном і базується на методі визначення орієнтації лінії, яка містить відрізок, що відсікається, по відношенню до сторони многокутника, а також на методі визначення місцезнаходження точки відрізка відносно вікна – всередині, на границі або зовні вікна. Для цього в алгоритмі Кіруса-Бека використовується вектор внутрішньої нормалі до ребра вікна.

Розглянемо опуклу двовимірну область P , тобто опуклий многокутник. Внутрішньою нормаллю \mathbf{n} до сторони вікна називається нормаль, яка направлена в бік області, що складає вікно відсікання. Як видно з рис. 9.15, внутрішня нормаль \mathbf{n} у точці a , що лежить на границі, задовольняє умову

$$(\mathbf{n} \cdot (\mathbf{b} - \mathbf{a})) \geq 0, \quad (9.4)$$

де \mathbf{b} – будь-яка інша точка границі опуклої області P .

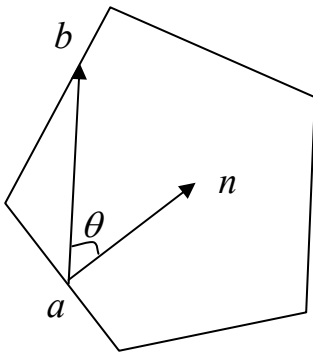


Рис. 9.15. Визначення місцезнаходження точки відносно вікна відсікання в алгоритмі Кіруса-Бека

Скалярний добуток в (9.4) для внутрішньої нормалі \mathbf{n} невід’ємний тому, що $\theta \in [0, \frac{\pi}{2}]$, тобто $\cos \theta \geq 0$.

Зазначимо, що внутрішні нормалі можна обчислювати й іншим способом. Для цього необхідно полігон обходити за годинниковою стрілкою і вибрати нормалі, які направлені вправо від кожного ребра (див. п.9.1.1).

Нехай \mathbf{n}_i – внутрішня нормаль до i -ї граничної лінії многокутного вікна, а $\mathbf{V} = \mathbf{b} - \mathbf{a}$ – вектор, що визначає орієнтацію відрізка ab , який відсікається вікном P . Тоді орієнтація відрізка ab відносно i -ї сторони вікна визначається знаком скалярного добутку $r_i = (\mathbf{n}_i \cdot \mathbf{V})$:

- при $r_i < 0$ відрізок ab направлений з внутрішньої сторони на зовнішню сторону i -ї граничної лінії вікна (рис. 9.16, *a*).
- при $r_i = 0$ точки a, b або збігаються (відрізок вироджується в точку) або відрізок ab паралельний i -й стороні вікна (рис. 9.16, *б*).
- при $r_i > 0$ відрізок ab направлений із зовнішньої сторони на внутрішню сторону i -ї граничної лінії вікна (рис. 9.16, *в*).

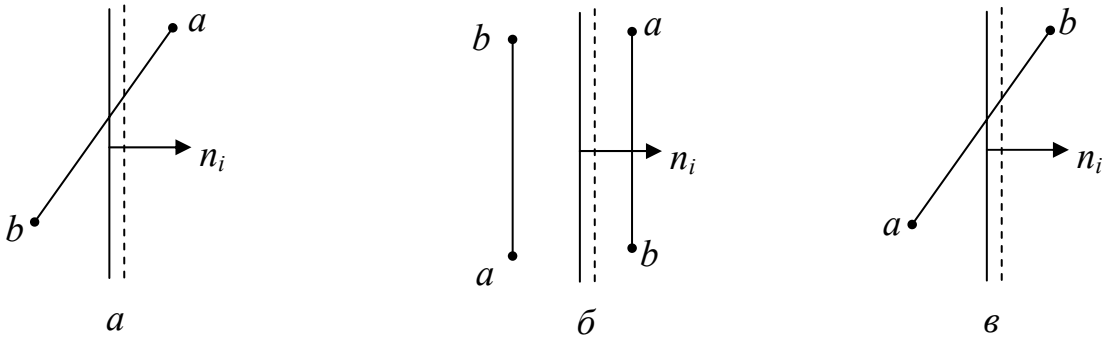


Рис. 9.16. Орієнтація відрізка ab відносно сторони вікна: *a* -- зсередини назовні; *б* – паралельно границі; *в* – ззовні всередину

Для визначення розміщення точки $V(t)$ відрізка ab відносно i -ї сторони вікна розглянемо скалярний добуток внутрішньої нормалі \mathbf{n}_i на вектор $\mathbf{Q}_i = V(t) - P_i$, що починається в початковій точці P_i ребра вікна і закінчується в деякій точці $V(t)$ відрізка ab , а саме

$$q_i = (\mathbf{n}_i \cdot \mathbf{Q}_i), \quad i = 1, 2, \dots \quad (9.5)$$

Аналогічно попередньому маємо (рис. 9.17):

- якщо $q_i < 0$, то точка $V(t)$ лежить із зовнішньої сторони границі;
- якщо $q_i = 0$, то точка $V(t)$ лежить на лінії границі вікна;
- якщо $q_i > 0$, то точка $V(t)$ лежить із внутрішньої сторони i -ї границі вікна.

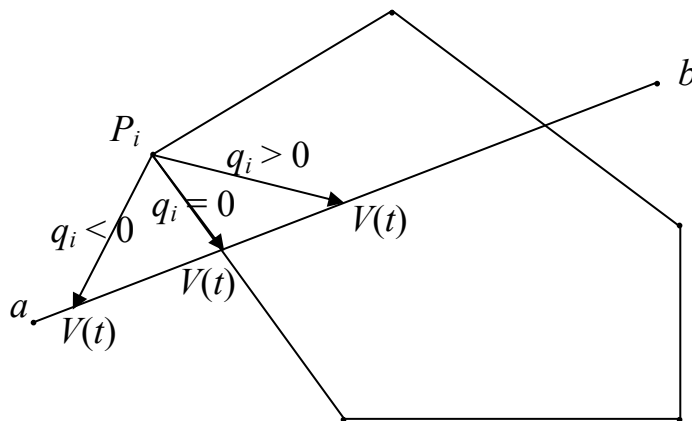


Рис. 9.17. Розміщення точки відносно границі вікна

Згадаємо векторно-параметричне рівняння відрізка ab

$$V(t) = a + (b - a)t, \quad t \in [0, 1]. \quad (9.6)$$

Оскільки багатокутник опуклий, то може бути дві точки перетину відрізка ab із вікном P і нам необхідно знайти два значення параметра t , що відповідають початковій і кінцевій точкам видимої частини відрізка.

З умови належності точки $V(t)$ границі вікна, тобто з умови $q_i = 0$, враховуючи (9.5) та (9.6), маємо

$$(\mathbf{n}_i \cdot (a + (b - a)t - P_i)) = 0,$$

або

$$(\mathbf{n}_i \cdot (a - P_i)) + (\mathbf{n}_i \cdot (b - a)t) = 0.$$

Звідки

$$t = -\frac{(\mathbf{n}_i \cdot (a - P_i))}{(\mathbf{n}_i \cdot (b - a))} = -\frac{(\mathbf{n}_i \cdot Q_i)}{(\mathbf{n}_i \cdot V)} = -\frac{q_i}{r_i}.$$

(9.7)

Алгоритм Кіруса-Бека має наступний вигляд.

Шукані значення параметрів t_0, t_1 точок перетину відрізка з вікном ініціалізуємо значеннями 0, 1, що відповідають початку і кінцю відрізка ab .

В циклі по i для кожної сторони вікна відсікання обчислюємо значення скалярних добутків

$$r_i = (\mathbf{n}_i \cdot (b - a)).$$

Якщо $r_i = 0$, то відрізок ab або вироджується в точку, або паралельний i -й стороні вікна. При цьому проаналізуємо значення $q_i = (\mathbf{n}_i \cdot (a - P_i))$. Якщо $q_i < 0$, то відрізок знаходиться зовні вікна і відсікання закінчено, інакше переходимо до наступної сторони вікна.

Якщо $r_i \neq 0$, то за формулою (9.7) обчислюємо значення t для точки перетину відрізка ab з i -ю границею вікна. Оскільки точки відрізка ab відповідають значенням $t \in [0, 1]$, то $t \notin [0, 1]$ відкидаємо.

При $r_i < 0$ лінія ab направлена з внутрішньої сторони i -граничної лінії на зовнішню і знайдені значення t визначатимуть кінцеву точку видимої частини відрізка ab . В цьому випадку визначається $\min t$ серед всіх одержаних t (рис. 9.18). Воно дає значення t_1 для кінцевої точки шуканого відрізка

$$t_1 = \min \left(\left\{ -\frac{q_i}{r_i} \mid r_i < 0, i = 1, 2, \dots \right\}, 1 \right).$$

Якщо поточне значення t_1 стане меншим за t_0 , то відрізок ab відкидається, оскільки не виконується умова $t_0 \leq t_1$.

При $r_i > 0$ лінія ab направлена із зовнішньої сторони граничної лінії на внутрішню, тому знайдені значення t визначатимуть початкову точку

видимої частини відрізка ab . В цьому випадку визначається $\max t$ серед всіх одержаних t . Воно дає значення t_0 для початкової точки видимої частини відрізка

$$t_0 = \max \left(\left\{ -\frac{q_i}{r_i} \mid r_i > 0, i = 1, 2, \dots \right\}, 0 \right).$$

Якщо поточне значення t_0 стане більшим за t_1 , то відрізок відсікається.

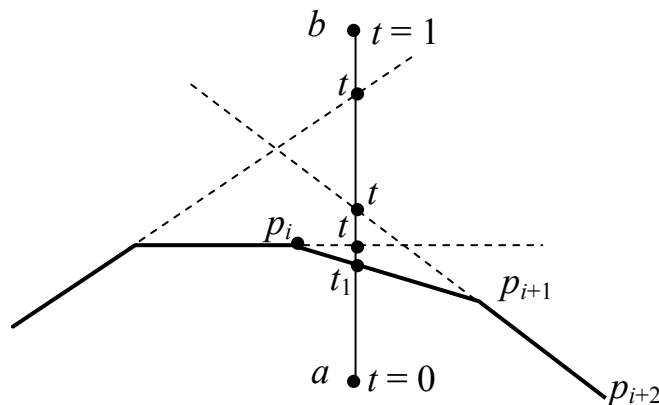


Рис. 9.18. Визначення кінцевої точки $V(t_1)$

На заключному етапі алгоритму значення t_0, t_1 використовуються для знаходження координат точок перетину відрізка ab з вікном P , тобто для побудови видимої частини відрізка.

При цьому, якщо $t_0 = 0$, то початкова точка залишається та сама й обчислення $V(t_0)$ не потрібні. Аналогічно при $t_1 = 1$ кінцева точка – b . Обчислення $V(t_1)$ теж не потрібні.

9.6.3. Відсікання полігонів. Алгоритм Вейлера-Азертонна

Многокутник можна розглядати як сукупність відрізків, але в цьому випадку при відсіканні вихідна фігура може перетворитися просто в набір відрізків (рис. 9.19). Тому необхідно мати ефективні алгоритми відсікання многокутників. До таких алгоритмів належать алгоритм Сазерленда-Ходжмена для відсікання опуклих многокутників та більш сильний алгоритм Вейлера-Азертонна для відсікання довільних областей.

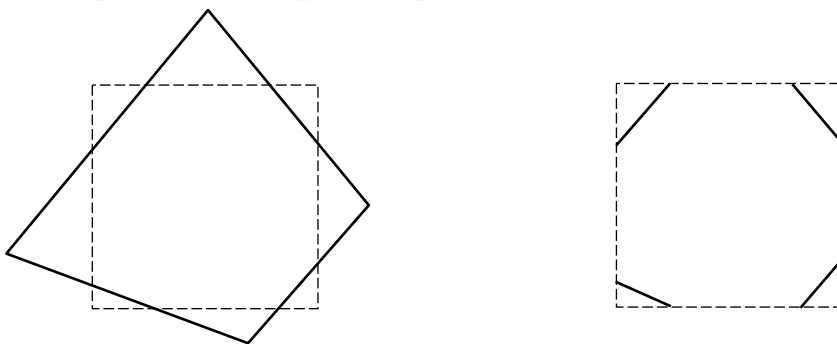


Рис. 9.19. Відсікання вікном ребер многокутника

На відміну від інших алгоритмів відсікання, алгоритм Вейлера-Азертонна дозволяє відсікати довільний неопуклий многокутник з дірками іншим неопуклим многокутником з дірками, причому можливе як зовнішнє, так і внутрішнє відсікання. Розглянемо його.

Кожний з многокутників у цьому алгоритмі задається циклічним списком вершин, причому вершини зовнішньої границі кожного многокутника обходяться за годинниковою стрілкою, а внутрішня границя (дірки) – проти годинникової стрілки так, щоб внутрішня область многокутників завжди залишалась справа від границі.

Границі многокутника, що відсікається, і вікна можуть перетинатися або не перетинатися. Якщо вони перетинаються, то виникають точки перетину ребер двох типів:

- вхідні точки, коли орієнтовані ребра многокутника, що відсікається, входять у вікно;
- вихідні точки, коли ребра многокутника, що відсікається, виходять із внутрішньої сторони вікна на зовнішню.

Точки перетину заносяться в циклічні списки вершин двох многокутників так, щоб не порушити правила обходу границі многокутників, тобто якщо точка перетину I_k знаходиться на ребрі, що з'єднує вершини P_i, P_{i+1} , то послідовність $P_i P_{i+1}$ перетворюється в послідовність $P_i I_k P_{i+1}$.

Алгоритм починається з точки перетину вхідного типу. Далі він простежує модифікований список вершин (границю многокутника) доти, поки не буде знайдено наступну точку перетину (вхідну чи вихідну). В цій точці відбувається поворот „направо”, тобто переключення на інший список в цю саму точку перетину. Далі в цьому списку знаходиться наступна точка перетину і відбувається переключення на перший список (поворот „наліво”) і т.д., поки не зустрінеться початкова вхідна точка.

Загальна схема алгоритма Вейлера-Азертонна має такий вигляд:

- побудувати циклічні списки вершин многокутника, що відсікається та вікна, згідно з наведеним вище правилом обходу;
- відшукати всі точки перетину вікна і многокутника (при цих обчисленнях дотикання не вважається перетином (рис. 9.20, 9.21));

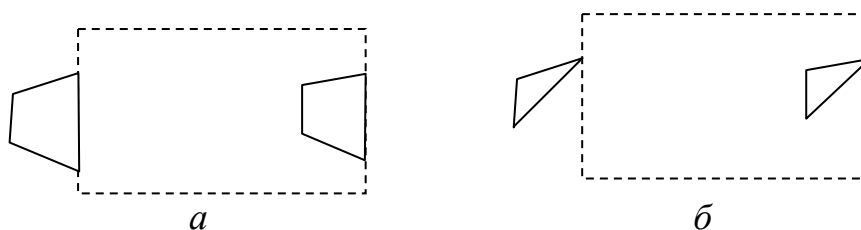


Рис. 9.20. Випадки дотикання (не вважається перетином): *a* – дотикання ребром; *б* – дотикання вершиною

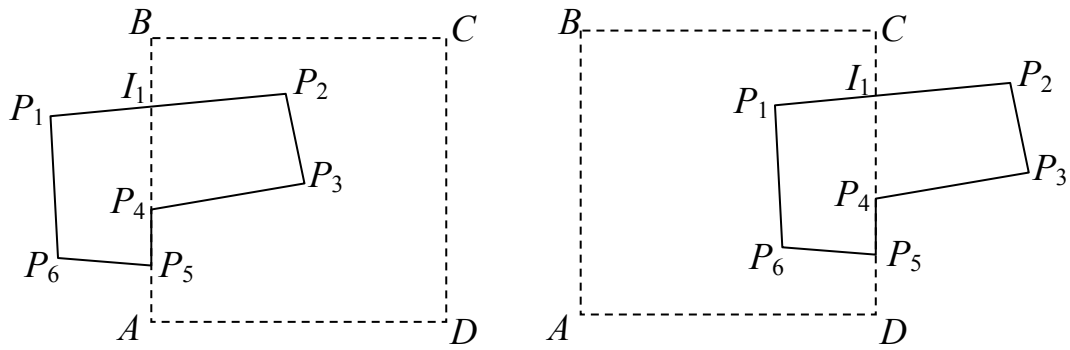


Рис. 9.21. Окремі випадки перетину

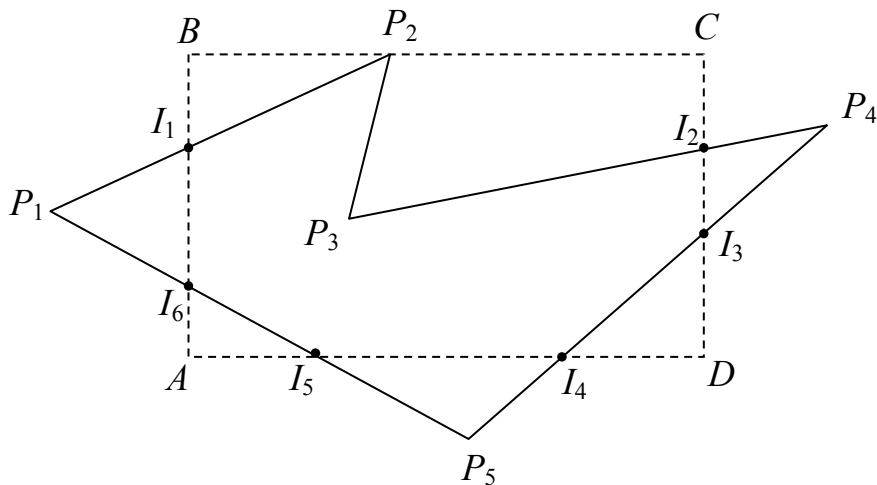


Рис. 9.22. Відсікання багатокутника вікном

- додати ці точки перетину до циклічних списків вершин багатокутників. При цьому встановлюються двосторонні зв'язки між однойменними точками перетину в двох списках;
- вибрати вхідну точку;
- проглянути список вершин поточного багатокутника, доки не знайдемо наступну точку перетину. При цьому формуємо результуючий список із пройдених вершин і точок перетину, не включаючи останньої;
- використовуючи двосторонні зв'язки перейти до перегляду списку вершин вікна, рухаючись по вершинах вікна до знайденої наступної точки перетину, і всі пройдені точки, за винятком останньої, занести в результат;
- ці дії повторюємо доти, поки не буде досягнута початкова вершина. Це означає, що знайдено чергову частину багатокутника, яка належить вікну;
- зобразити одержаний результат;
- якщо список вхідних точок перетину вікна з багатокутником, що відсікається, не вичерпується, то вибрати чергову вхідну точку і повторити алгоритм спочатку.

Для пошуку зовнішнього відсікання алгоритм необхідно починати з вихідної точки перетину, при цьому вершини в списку вершин вікна треба переглядати у зворотному порядку.

Для наочності наведемо приклад. Нехай вікном $ABCD$ відсікається багатокутник $P_1P_2P_3P_4P_5$ (рис. 9.22).

Для внутрішнього відсікання циклічні списки вершин і порядок роботи алгоритма зображені на рис. 9.23. Як вхідна вибрана точка I_1 . Пройдені точки формують результат $I_1P_2P_3I_2I_3I_4I_5I_6I_1$.

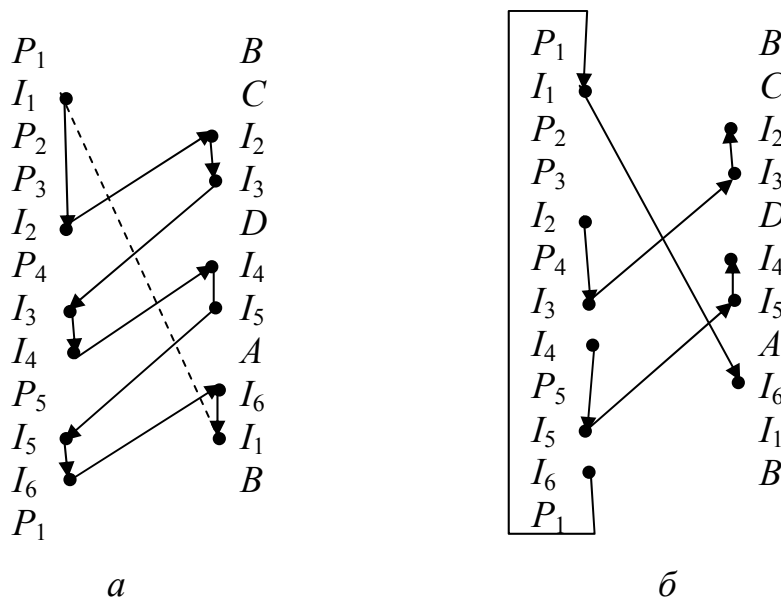


Рис. 9.23. Схема алгоритму Вейлера-Азерттона: a – внутрішнє відсікання; b – зовнішнє відсікання

Для виконання зовнішнього відсікання як початкову беремо вихідну точку перетину, наприклад I_2 . Далі діємо згідно з алгоритмом, проходячи вершини вікна знизу вгору.

В результаті одержуємо три частини багатокутника: $I_2P_4I_3I_2$, $I_4P_5I_5I_4$, $I_6P_1I_1I_6$, що відсікаються вікном.

9.7. Побудова опуклої оболонки масиву точок

Припустимо, що на площині задано масив точок $P = \{P_1, P_2, \dots, P_n\}$ з координатами $P_i(x_i, y_i)$, $i = 1, 2, \dots, n$, $n \geq 2$.

Опукла полігональна оболонка масиву точок є об'єднанням усіх полігонів, побудованих на точках цього масиву як на вершинах. Її можна зобразити як опуклий полігон мінімальної площі, що покриває всі точки масиву (зазначимо, що не всі точки з масиву P є вершинами опуклої оболонки). Існує багато різних алгоритмів для пошуку опуклої оболонки. Наведемо деякі з них.

9.7.1. Метод загортання подарунка

Спочатку обираємо напрям обходу точок масиву (наприклад, додатний) та точку, яка точно є вершиною границі опуклої оболонки. За таку точку можна взяти точку P_m із мінімальною ординатою y_m . Якщо таких точок декілька, то оберемо найлівішу з них, тобто точку з мінімальною абсцисою x_m . Шляхом обміну $P_1 \leftrightarrow P_m$ поставимо знайдену точку на початок масиву, а її копію добавимо в кінець масиву як точку P_{n+1} для перевірки замикання оболонки.

Задаємо початковий напрям вектора охоплення V вздовж осі x та індекс вершини оболонки $k = 1$. Далі організуємо зовнішній цикл пошуку вершин оболонки і накопичення їх по порядку в масиві P . У внутрішньому циклі по i від $k+1$ до $n+1$ для кожної точки P_i обчислюється вектор пробного напрямку $W = P_i - P_k$. При ненульовій довжині цього вектора $r = |W| \neq 0$ (така перевірка дозволяє відкинути крайні точки) знаходимо

$$c = \cos \varphi = \frac{V \cdot W}{|V| \cdot |W|}.$$

Перебравши точки P_i для всіх $i > k$, знаходимо таку точку P^* , яку видно з точки P_k під мінімальним кутом φ , тобто при максимальному значенні c . Якщо таких точок декілька, то обираємо найдальшу від P_k . Інкрементуючи індекс $k = k + 1$, зробимо її новою вершиною оболонки шляхом обміну $P^* \leftrightarrow P_k$. Новий вектор напрямку обходу $V = P_k - P_{k-1}$.

Зовнішній цикл і побудова оболонки закінчується при співпадінні точки P_k із початковою точкою P_1 .

Алгоритм повертає субмасив вершин оболонки $\{P_1, P_2, \dots, P_k\}$.

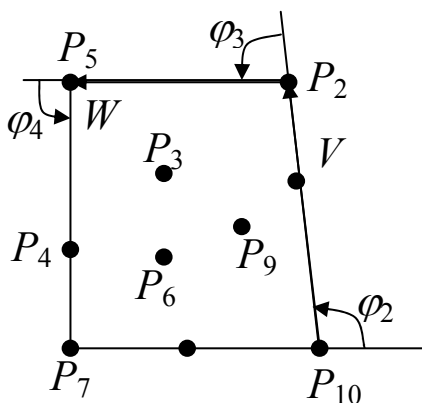


Рис. 9.24. Опукла оболонка

Робота алгоритму зображена на рис. 9.24. Цей алгоритм називають алгоритмом загортання подарунка або побудови загороди. Він вимагає $O(nh)$ операцій, де h – кількість вершин, що містить опукла оболонка,

n – кількість точок у вихідному масиві.

Алгоритм побудови опуклої оболонки використовується при побудові опуклих полігонів, тіні опуклого поліедра P . Для цього спочатку обчислюємо

точки – проєкції вершин поліедра на площину, будуємо опуклу оболонку цих точок, а далі виконуємо зовнішнє відсікання оболонки замкнутим контуром і зафарбовуємо одержаний полігон кольором тіні.

9.7.2. Метод обходу Грехема

Розглянемо ще один алгоритм побудови опуклої оболонки – метод обходу Грехема. У цьому методі опукла оболонка скінченного набору точок S знаходиться у два етапи.

На першому кроці вибираємо деяку екстремальну точку $P_e \in S$ і сортуємо решту точок відповідно до кута нахилу променя, який проведений у ці точки з точки P_e , до горизонталі. Якщо дві точки мають однаковий полярний кут, то точка, яка розміщена ближче до P_e , ставиться у відсортованому масиві раніше. Щоб здійснити це впорядкування, необхідно обчислити відстань або порівняти суму абсолютних значень координат. Як екстремальну вибираємо точку з мінімальною y -координатою і міняємо її місцем з точкою P_1 .

На другому кроці алгоритму виконуємо покрокову обробку відсортованих точок, формуючи нову послідовність тимчасових точок (ребер), які зрештою утворюють шукану опуклу оболонку. Попереднє сортування точок спрощує виконання побудови опуклої оболонки.

Для розв'язування питання про те, яка саме точка повинна бути включена в границю опуклої оболонки, використовується той факт, що при обході границі опуклої оболонки проти годинникової стрілки кожна вершина повинна відповідати повороту вліво.

Якщо для точки P_i обхід $P_{i-1}P_iP_{i+1}$ не відповідає повороту вліво, то точка P_i вилучається з опуклої оболонки, а далі потрібно перейти на перевірку повороту вліво обходу $P_{i-2}P_{i-1}P_{i+1}$. Якщо і цей поворот не є поворотом вліво, то вилучаємо точку P_{i-1} , хоча раніше вона була включена в тимчасову послідовність, тобто поворот $P_{i-2}P_{i-1}P_i$ був лівим. Цей процес продовжуємо доти, поки не знайдемо лівий поворот, тобто поточна опукла оболонка будується на основі уже оброблених точок. На рис. 9.25 зображено процес роботи алгоритму Грехема.

Розглянемо обробку точки P_5 (рис. 9.25, δ). Оскільки обхід $P_4P_5P_6$ не є лівим, то P_5 вилучається з поточної оболонки. Далі перевіряємо обхід $P_3P_4P_6$ – він теж не є лівим. Тому і P_4 вилучаємо з поточної оболонки.

Ситуація змінюється при розгляді точки P_3 . Тут обхід $P_1P_3P_6$ лівий, тому в поточній оболонці залишаємо точку P_3 та переходимо до обробки наступної точки P_6 , тобто розглядаємо поворот $P_3P_6P_0$.

Метод обходу Грехема вимагає $O(n \lg n)$ арифметичних операцій.

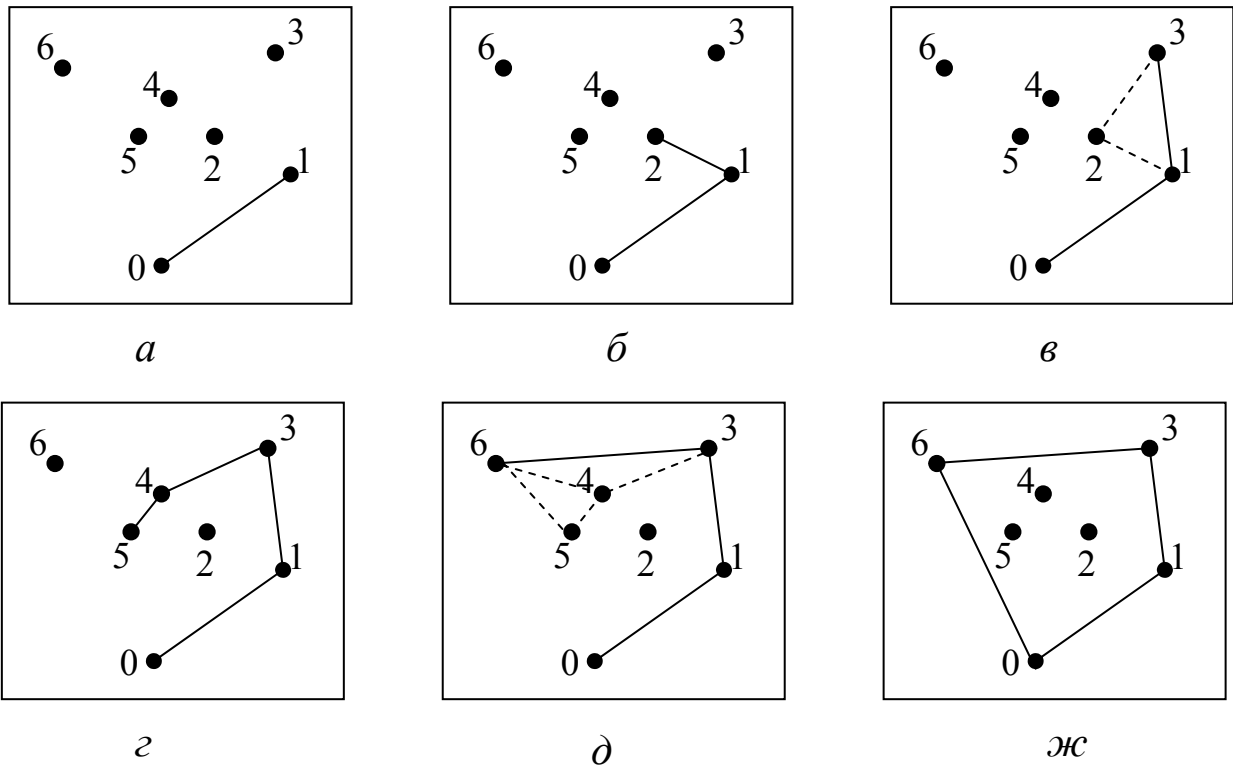


Рис. 9.25. Робота алгоритму обходу Грехема (точки P_i , $i = 0, 1, \dots, 6$ вже заздалегідь відсортовані за полярним кутом)

Зауваження 3. До недоліків алгоритму Грехема можна віднести той факт, що він не допускає розбиття вихідної задачі на підзадачі для паралельної обробки даних. Тому розроблені алгоритми побудови опуклої оболонки, що базуються на розбитті вихідної множини точок на дві підмножини і об'єднанні двох ламаних в випуклу оболонку.

Зауваження 4. В алгоритмі Грехема використовуються тригонометричні функції, а їх обчислення вимагають великих обчислювальних затрат в часі. Тому для швидкодійних алгоритмів необхідно уникати цих обчислень. Для цього вчиняють так: серед n точок знаходять найбільш ліву і найбільш праву точки, які вочевидь є вершинами опуклої оболонки. Пряма, що проходить через ці дві точки, ділить дану множину на дві частини. Обидві ці частини породжують дві частини опуклої оболонки, причому вони є монотонними ламаними відносно осі x . Тому, окремо відсортувавши точки за значенням абсциси, виконуємо обхід Грехема.

Зауваження 5. Алгоритм Грехема не відкритий, оскільки перед початком роботи алгоритму потрібно знати усі n точок. Однак у деяких випадках потрібно мати алгоритм, який буде змінювати опуклу оболонку в міру надходження нових точок. Такі алгоритми розроблені і називаються динамічними.

9.8. Тріангуляція полігонів

Широко розповсюдженою задачею обчислювальної геометрії є задача тріангуляції полігонів. *Тріангуляцією* називається операція розрізування простих полігонів на трикутники за допомогою неперетинних внутрішніх діагоналей. Вона часто використовується для зведення розв'язування різних задач (перетину, відсікання, усунення невидимих граней тощо) в області, що має складну конфігурацію, до сукупності задач, що розв'язуються в області трикутника, оскільки трикутник є простішою областю і завжди опуклим, а в цьому випадку задачі розв'язуються значно простіше. Наприклад, для визначення належності точки неопуклому полігону можна здійснити його тріангуляцію й подальшу перевірку належності точки хоча б одному трикутнику.

При аналізі складних криволінійних поверхонь поверхню можна апроксимувати сіткою трикутників, для яких значно простіше розв'язуються задачі візуалізації, зокрема зафарбовування точок трикутної грані в однорідний колір завдяки її планарності. Розрізування полігона на трикутники може бути ручним і алгоритмічним. Ручне розрізування можна застосувати до апріорі відомого полігона. Якщо полігон обчислюється в процесі роботи програми та заздалегідь не відома його конфігурація, то необхідно використовувати алгоритми тріангуляції.

9.8.1. Тріангуляція опуклих полігонів

Задача тріангуляції опуклого n -полігона розв'язується просто. Для цього досить провести $n - 3$ діагоналей $P_1P_3, P_1P_4, \dots, P_1P_{n-1}$, у результаті чого одержуємо $n - 2$ трикутники (рис. 9.26, а). Доведено, що цей факт правильний для будь-якого n -полігона, оскільки в будь-якому полігоні можна провести $n - 3$ діагоналі.

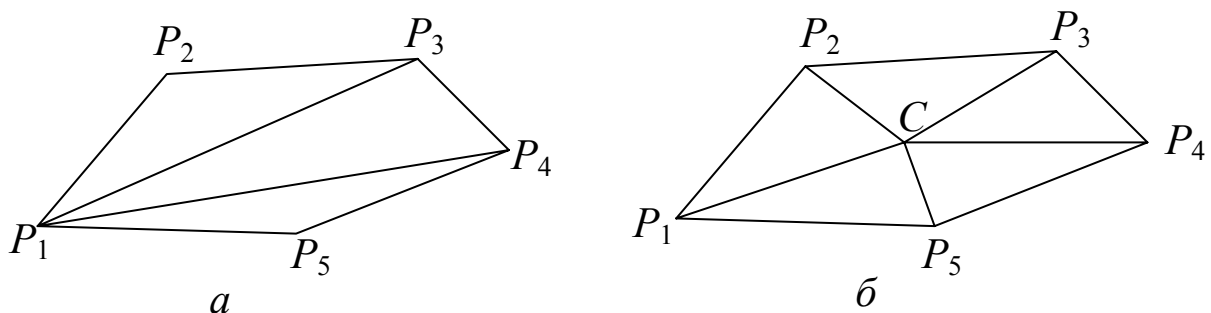


Рис.9.26. Тріангуляція опуклих полігонів

Опуклий полігон можна розрізати на n трикутників ще й в інший спосіб (рис. 9.26, б):

$$CP_1P_2 \cup CP_2P_3 \cup \dots \cup CP_nP_1,$$

ввівши додаткову вершиною C , яку можна обчислити як геометричний центр полігона

$$C = \frac{1}{n} \sum_{i=1}^n P_i.$$

В цьому випадку одержується більше трикутників і вони мають менш витягнуту форму, ніж при розрізуванні діагоналями.

9.8.2. Тріангуляція неопуклих полігонів

Для тріангуляції неопуклого полігона спочатку шукають вершину, з якої можна провести відсікаючу діагональ. Для цього підходить не кожна опукла вершина (наприклад, точка P_2 на рис. 9.28). Діагональ P_1P_3 не належить внутрішній частині полігона P . Діагональ називається *внутрішньою*, якщо вона цілком лежить усередині полігона (відомо, що у довільному простому багатокутнику з $n \geq 4$ вершинами можна провести внутрішню діагональ).

Діагональ опуклої вершини P_i може бути лише кандидатом для відсікаючої діагоналі. Питання вибору діагоналі можна розв'язати, якщо врахувати довжину діагоналей. Для цього здійснюють обхід полігона проти годинникової стрілки по всіх опуклих вершинах полігона, починаючи з точки P_1 . Нагадаємо, що при цьому вершина P_i опукла, якщо

$$D_i = \begin{vmatrix} x_{i-1} & y_{i-1} & 1 \\ x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \end{vmatrix} > 0.$$

Далі вибирають найкоротшу діагональ $P_{i-1}P_{i+1}$ серед діагоналей опуклих вершин P_i . Ця діагональ буде відсікати від полігона трикутник

$P_{i-1}P_iP_{i+1}$. Потім цю процедуру застосовують до полігона, що залишився після першого відсікання і т.д.

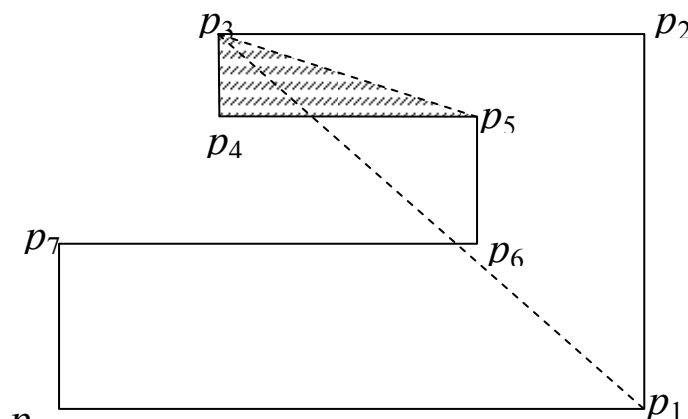


Рис. 9.27. Тріангуляція неопуклого полігона

Наприклад, для полігона, зображеного на рис. 9.27, потрібно спочатку розглянути діагоналі опуклих вершин $P_1, P_2, P_3, P_4, P_7, P_8$ (вершини P_5, P_6 неопуклі, бо для них $D < 0$), тобто діагоналі $P_8P_2, P_1P_3, P_2P_4, P_3P_5, P_6P_8, P_7P_1$.

Серед цих діагоналей найменшу довжину має діагональ P_3P_5 , тому відтинаємо трикутник $P_3P_4P_5$. Залишиться полігон

$P_1P_2P_3P_5P_6P_7P_8P_1$, до якого необхідно застосувати ті ж самі операції, що і для вихідного полігона і т.д., поки полігон, який залишається, не стане трикутником.

Зауваження 6. Задачу триангуляції можна розв'язувати й іншим способом, наприклад, методом розрізування полігона хордами – продовженнями його ребер, що складають неопуклу вершину, на кілька опуклих підполігонів. Далі опуклі підполігони розрізуємо діагоналями на трикутники.

9.8.3. Триангуляція Делоне

Розглянемо задачу триангуляції набору точок S на площині. Усі точки набору S повинні лежати у вершинах трикутників. Ребра трикутників не повинні перетинатися. Усі точки набору S можна розбити на граничні точки, тобто на такі, які лежать на границі опуклої оболонки, і внутрішні, що лежать усередині опуклої оболонки. Так само можна класифікувати й ребра, що одержуються в результаті триангуляції, – ребра оболонки і внутрішні ребра.

Будь-який набір точок S , за винятком деяких тривіальних випадків, допускає більше одного способу триангуляції і при цьому одержується одна і та ж сама кількість трикутників (рис. 9.28). За теоремою Ейлера кількість трикутників не перевищує $2n$, де n – кількість точок масиву S .

Серед усіх можливих триангуляцій виділяється спеціальний вид – *триангуляція Делоне* [26]. Ця триангуляція характеризується тим, що утворені трикутники наближаються до максимально правильних.

Означення. Триангуляція набору точок S називається триангуляцією Делоне, якщо всередині кола, описаного навколо кожного трикутника, немає точок набору S , окрім їх вершин.

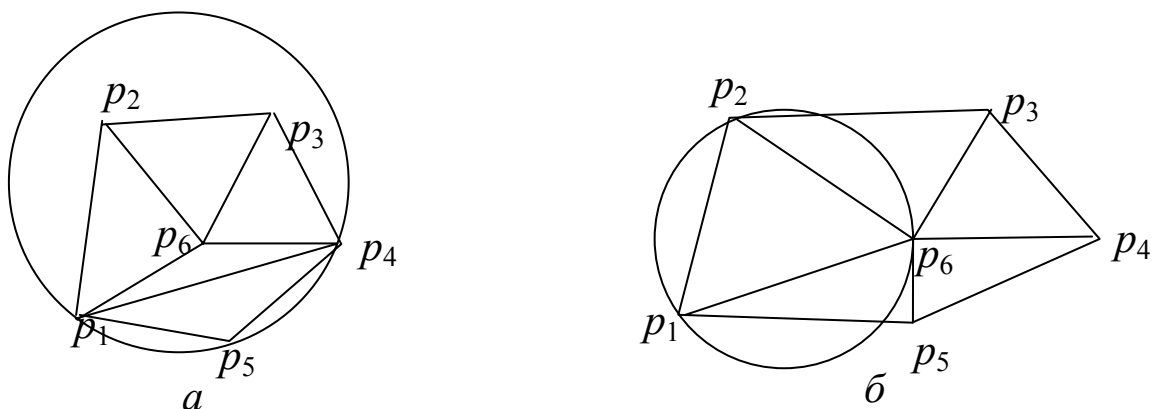


Рис. 9.28. Триангуляція набору точок

Триангуляцію, що зображена на рис. 9.28, а, не можна віднести до триангуляції Делоне, а триангуляція на рис. 9.28, б є триангуляцією Делоне.

Використовуючи означення триангуляції Делоне, можна розробити алгоритм її побудови, час роботи якого прямо пропорційний кількості точок масиву S , і виконана побудова є однозначною [38].

Задачі триангуляції виникають, наприклад, при побудові ізоліній функцій двох змінних, які задані множиною дискретних значень на нерегулярній сітці. Зокрема, це задачі створення топографічних карт, які є картами ізоліній функції висоти рельєфу місцевості: на практиці значення цих функцій найчастіше доступні лише в точках скінченної нерегулярної множини.

Контрольні питання та завдання

1. Як перевірити напрям обходу трьох точок?
2. Як визначити чи опуклий полігон?
3. Як визначити чи пряма лінія перетинається з полігоном?
4. В чому суть габаритного тесту?
5. Сформулюйте основну ідею променевого тесту.
6. На яких ідеях ґрунтується кутовий тест?
7. Як визначити перетин двох відрізків?
8. На площині задано два трикутники. Як визначити чи трикутники перетинаються, чи один трикутник лежить всередині іншого?
9. В чому полягає суть задачі відсікання?
10. Яка основна ідея алгоритму Сазерленда-Коена?
11. Як будуються двійкові коди в алгоритмі Сазерленда-Коена?
12. Як знаходяться кінцеві точки видимої частини відрізка при відсіканні його опуклим полігоном?
13. Як визначити внутрішню/зовнішню нормаль ребра опуклого многокутника?
14. Як знайти перетин опуклих полігонів?
15. Як класифікуються алгоритми відсікання?
16. Як утворюється коди відрізків в FC -алгоритмі?
17. Як здійснюється відсікання відрізків, якщо їх код дорівнює 73?
18. Яка основна ідея алгоритму Кіруса-Бека?
19. Як визначається орієнтація відрізка відносно сторони вікна в алгоритмі Кіруса-Бека?
20. Опишіть основні кроки алгоритму Кіруса-Бека.
21. Які можливості має алгоритм Вейлера-Азертонна?
22. Наведіть загальну схему алгоритму Вейлера-Азертонна.
23. Як визначаються вершини опуклої оболонки в методі загортання подарунка?
24. Опишіть роботу алгоритма Грехема на конкретному прикладі.
25. Що таке триангуляція полігонів?

26. Як виконується триангуляція неопуклих полігонів?
27. Розкрийте поняття триангуляції Делоне.

Вправи і задачі для самостійного виконання

1. Розробити алгоритм непараметричного пошуку перетину відрізка зі сторонами вікна шляхом поділу відрізка його навпіл.
2. Розробити алгоритм відсікання відрізків кількома вікнами, якщо вікна мають пріоритет.
3. Написати процедуру для розпізнавання опуклих багатокутників шляхом обчислення векторних добутків векторів сторін.
4. Для одержання сферичного трикутника на сфері беруть три точки і з'єднують дугами. Обчислити точку перетину променя зі сферичним трикутником.
5. Розробити алгоритм визначення перетину променя з багатокутником в тривимірному просторі.
6. Розробити алгоритм визначення належності точки многограннику.
7. Розробити тест опуклості просторового багатокутника.
8. Розробити алгоритм аналізу двійкового коду в алгоритмі Сазерленда-Коена для знаходження ребра, що перетинається з відрізком.
9. Узагальнити алгоритм Сазерленда-Коена для 3D-простору.
10. Узагальнити алгоритм Кіруса-Бека для тривимірного випадку.
11. Розробити алгоритм генерування довільного полігона.
12. Узагальнити променевий тест визначення приналежності точки полігону на випадок променя з довільним напрямом V .
13. Модифікувати УХ-алгоритм для перевірки належності точки полігону.
14. Розробити тест перетину двох плоских багатокутників.
15. Розробити алгоритм об'єднання опуклих полігонів.
16. Розробити алгоритм відсікання опуклого/неопуклого полігона півплощиною (рис. 9.5, *a*).
17. Навести приклад відсікання неопуклого багатокутника з отворами іншим неопуклим багатокутником з отвором. На цьому прикладі проілюструвати роботу алгоритма Вейлера-Азертонна.
18. Розробити алгоритм розрізання неопуклого багатокутника на опуклі багатокутники, використовуючи операції повороту.
19. Розробити алгоритм відсікання багатокутної зафарбованої області.
20. Розробити програму для виконання різноманітних бінарних операцій над багатокутниками, що задаються в інтерактивному режимі.

Тема 10. Фрактали в комп'ютерній графіці

10.1. Поняття фрактала

Поняття фрактала, фрактальної геометрії з'явилося в кінці 70-х років минулого століття і стало широко застосовуватися математиками і комп'ютерними художниками. Фрактали знайшли широке застосування в комп'ютерній графіці завдяки компактності математичного апарату, необхідного для їх відтворення на екрані комп'ютера.

За останній час з'явилося багато праць (особливо на Заході), присвячених фракталам (наприклад, [18, 20]). В цих працях вивчаються математичні основи фракталів, а також наводяться оригінальні графічні зображення фракталів, одержаних за допомогою комп'ютерів.

Слово *фрактал* походить від латинського *fractus* і в перекладі означає “складений із фрагментів”. Цей термін запропонований математиком Б. Мандельбротом у 1975 р. для позначення самоподібних структур. Поняття фрактала як математичного об'єкта досі залишається невизначеним – зроблено лише окремі спроби визначити фрактал. Перша спроба базується на топологічній розмірності множин (точка має розмірність 0, лінія – 1, плоска фігура – розмірність 2). Фігури з дробовою розмірністю описують властивості фракталів. Але це визначення не точне, оскільки трапляються фрактали з розмірністю 2. У комп'ютерній графіці використовують таке означення фрактала.

Фракталом, за Мандельбротом, називається геометрична фігура, яка складається з частин, які в певному розумінні подібні за формою на всю фігуру, тобто це геометрична фігура, в якій один і той самий фрагмент повторюється при кожному зменшенні масштабу (частина об'єкта схожа на сам об'єкт).

Характерною рисою фрактала є інваріантність (самоподібність) відносно масштабу, наприклад, гірський камінь має такі ж обриси, як і гірський хребет.

Існують *конструктивні (геометричні) та динамічні (алгебраїчні) фрактали*. Найбільш наочними є *геометричні фрактали*, оскільки їхня форма може бути описана як послідовність простих геометричних операцій. На площині їх отримують за допомогою деякої ламаної, яка називається *генератором*. За один крок алгоритму кожен із відрізків ламаної замінюється на ламану-генератор у відповідному масштабі. У результаті нескінченного повторення кроків цього алгоритму одержуємо геометричний фрактал. Тому алгоритми побудови фракталів в більшості випадків носять рекурсивний характер. Рекурсивність обумовлює властивість самоподібності фрактала. Прикладом геометричного фрактала є крива Коха. Послідовні наближення до фрактала Коха зображено на рис. 10.1.

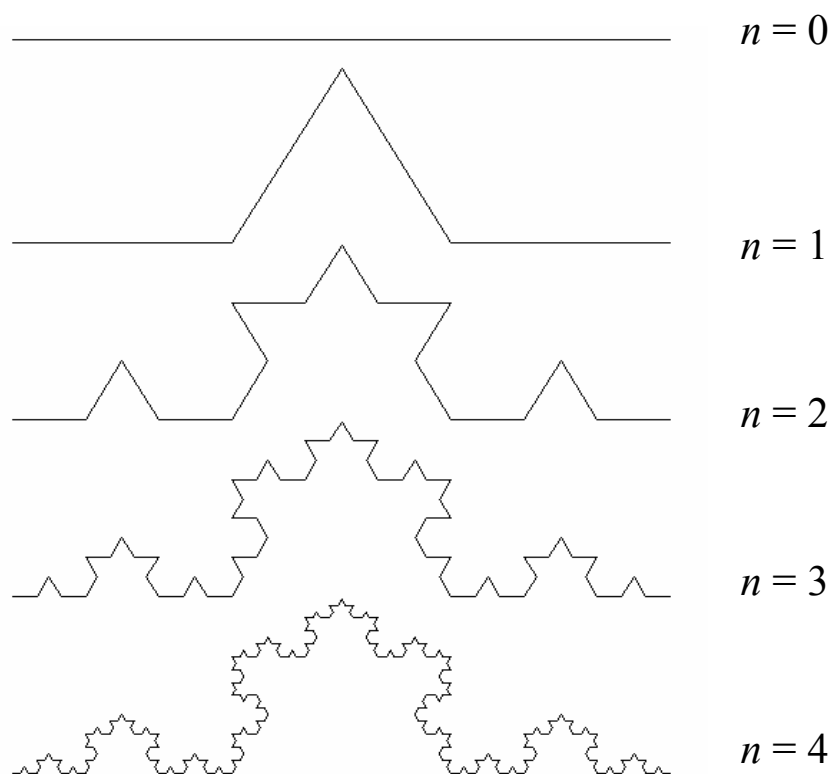


Рис. 10.1. Фрактал Коха

Прикладом конструктивного фрактала може бути дерево, стовбур якого розділений на менші гілки. У свою чергу кожна з цих гілок ділиться на дві менші і т.д. Практично ми зупинимося на певному кроці (уявно цей процес можна продовжити до нескінченності). Фігура, що з'явиться в результаті цих операцій і є фракталом, в якому кожна його частина у свою чергу подібна на весь фрактал (рис. 10.2).

Поряд із конструктивними фракталами були відкриті множини, схожі на фрактали (вони можуть володіти масштабною інваріантністю лише наближено). Такі множини називаються *динамічними* (алгебраїчними) фракталами. Як правило, вони виникають у нелінійних дискретних динамічних системах.

Якщо нелінійна система досить складна, то вона у своєму розвитку проходить етапи стійкого і хаотичного розвитку. Нелінійні системи володіють кількома стійкими станами (атракторами). Існують області початкових даних, з яких система потрапляє в область притягання атрактора. Замальовуючи області притягання різними кольорами, можна отримати кольоровий фазовий портрет цієї системи.

Динамічні фрактали одним із перших описав у 1918 р. французький математик Г. Жуліа. На той час були ще відсутні зображення фракталів. Комп'ютери зробили видимим те, чого не уявляли в часи Жуліа. Комп'ютерні візуальні результати перевершили всі очікування.

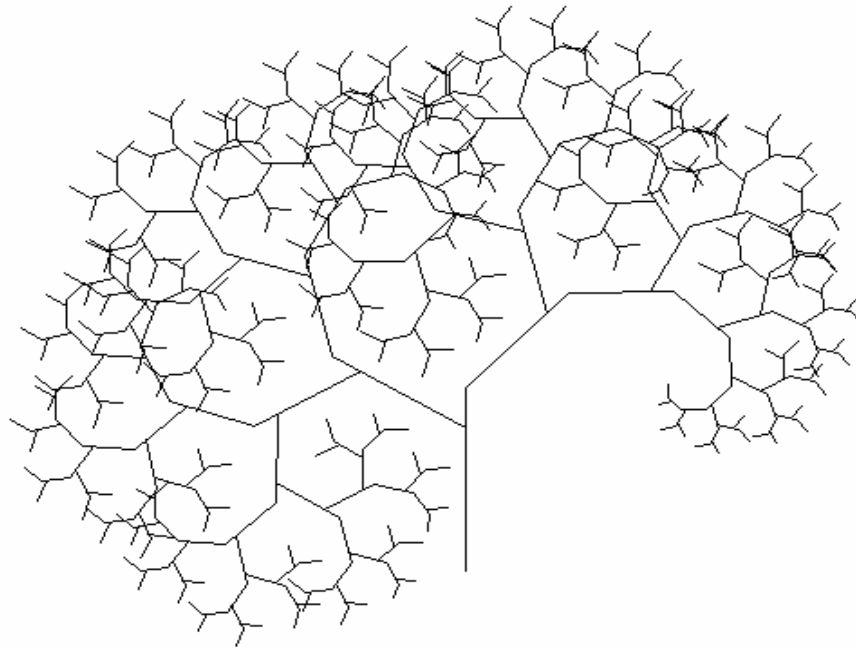


Рис. 10.2. Фрактал „дерево”

10.2. Конструктивні фрактали

Фрактали, в яких фрагменти повторюються при зменшенні масштабу, називаються *конструктивними фракталами*. Розглянемо математичні основи деяких конструктивних фракталів.

10.2.1. Крива Коха

У 1904 р. Кох побудував криву, яка в кожній точці не має похідної, а сама крива і кожна її частина має нескінченну довжину. На рис. 10.1 зображено наближення кривої Коха. Побудову кривої Коха починаємо з відрізка-основи. Для цього усуваємо середню третю частину відрізка і замінюємо її на сторони рівностороннього трикутника. Далі чинимо аналогічно з кожним відрізком ламаної (рис. 10.1). Криву Коха можемо уявити як границю таких операцій.

Якщо основа має одиничну довжину, то ламана при $n = 1$ складається з 4 відрізків (кожен довжиною $1/3$). При $n = 2$ одержуємо ламану з 16 відрізків загальною довжиною $16/9 = (4/3)^2$.

Якщо застосувати $n = p$ перетворень до основи, то результатом буде ламана, що складається з 4^p відрізків довжиною $1/3^p$. Загальна довжина цієї ламаної дорівнює $(4/3)^p$.

Якщо p прямує до нескінченності, то довжина кожного відрізка прямує до нуля, а загальна довжина кривої Коха – до нескінченності. Крива Коха є образом одиничного відрізка $[0,1]$, а будь-яка частина кривої Коха – мініатюрною копією всієї кривої.

Якщо за основу кривої Коха взяти сторони правильного многокутника, то можна одержати гарні зображення (варіації на тему кривої

Коха). Наприклад, якщо за основу взяти квадрат, а як генератор – фрагмент Коха, орієнтований назовні квадрата, то одержимо фігуру (острів Коха), що зображена на рис. 10.3 ($p = 4$).

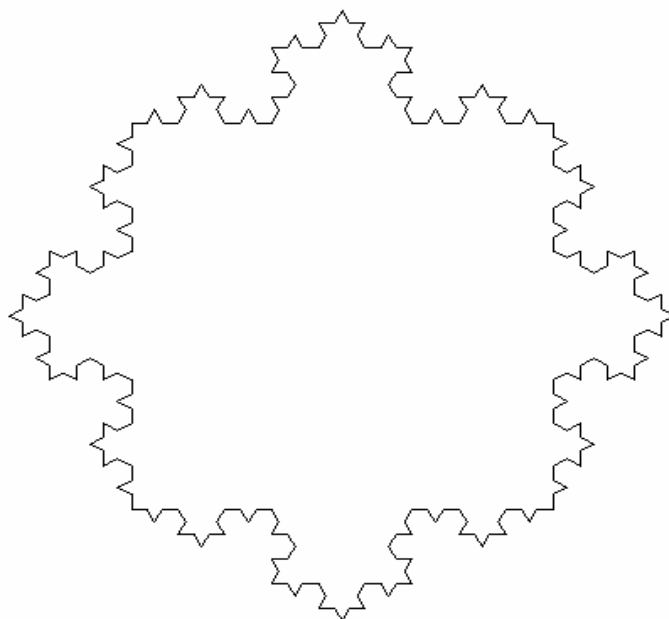


Рис. 10.3. Острів Коха при $p = 4$

Розглянемо загальну схему побудови подібних фракталів. Припустимо, що основа складається з u відрізків, а фрагмент – із v відрізків. Вершини для основи і фрагмента повинні бути задані явно, тоді при заданому порядку апроксимації p із кожного відрізка основи одержується ламана з $v^p - 1$ вершинами.

Розглянемо деякі випадки.

1. Основою є квадрат з одиничною довжиною. Відрізок $[0, 1]$ замінюється фрагментом з проміжними точками $(0,4; 0,2)$, $(0,6; 0,2)$ (рис. 10.4, *a*). У результаті послідовних наближень одержимо фрактал, який називають островом Мінковського (рис. 10.4, *б*).

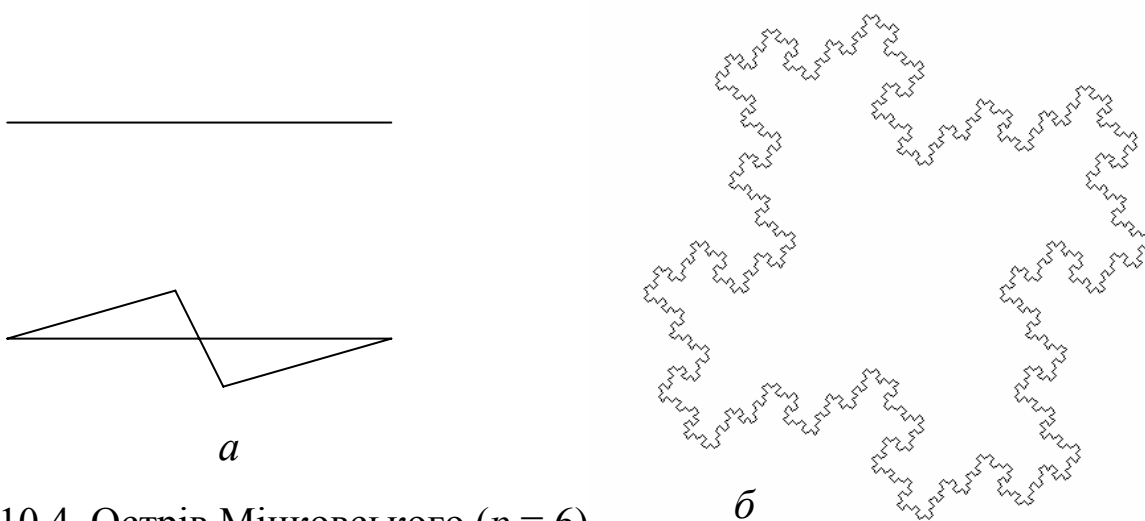


Рис. 10.4. Острів Мінковського ($p = 6$)

2. Основою є квадрат з одиничними сторонами, фрагментом – ламана з проміжними точками $(0,47; 0)$, $(0,5; 0,47)$, $(0,53; 0)$ (рис. 10.5, *a*). Фрактал (льодовий квадрат) зображений на рис. 10.5, *б*.

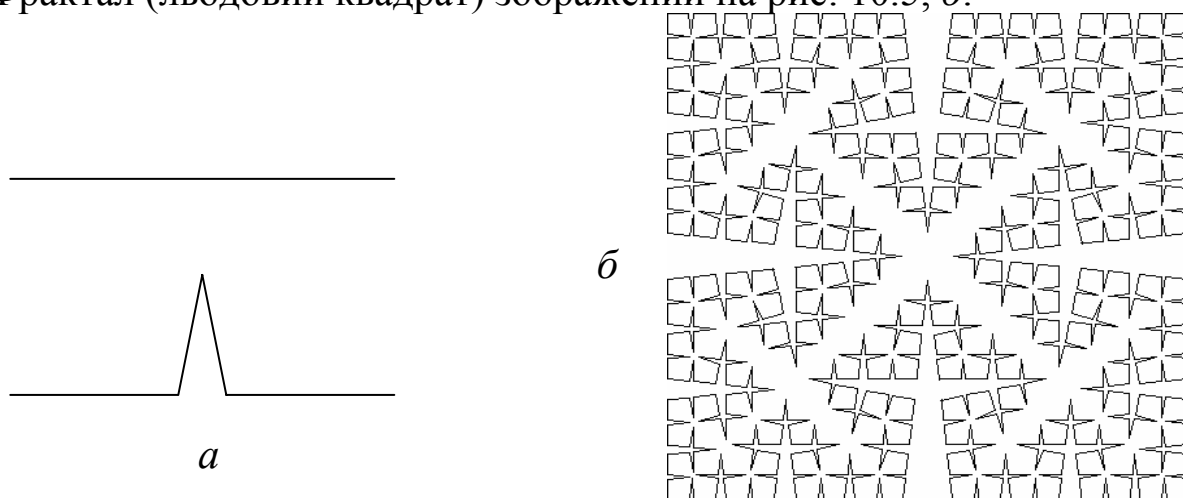


Рис. 10.5. Льодовий квадрат ($p = 4$)

10.2.2. Зіркові фрактали

Зірковий фрактал складається з п'ятикутної зірки та гірлянди з п'яти менших зірок, а кожна з п'яти зірок має чотири вільних кінці, на яких у свою чергу, знаходяться ще менші зірки. Теоретично цей процес можна продовжувати до нескінченності – в результаті одержимо зірковий фрактал. Задавши деяку кількість кроків p , одержуємо наближення зіркового фрактала. На рис. 10.6 зображено наближення зіркового фрактала при $p = 2$. Виконаємо нумерацію відрізків фрактала від 0 до 19.

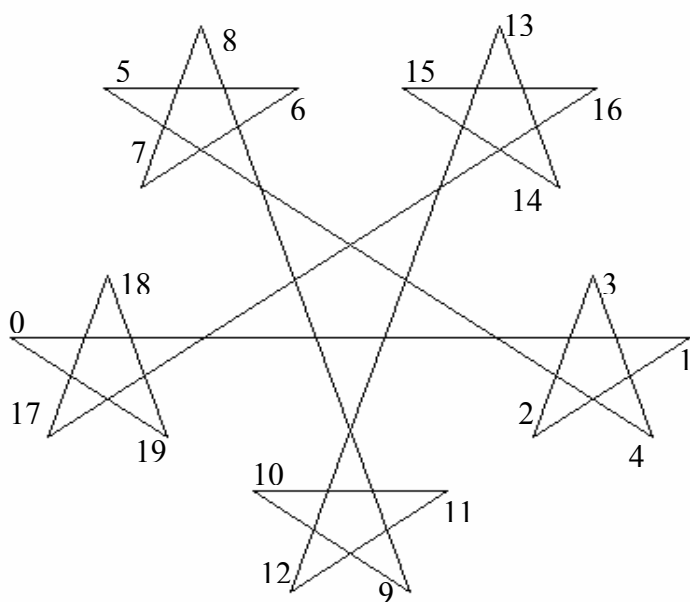


Рис 10.6. Зірковий фрактал при $p = 2$

Зірковий фрактал можна побудувати як замкнену ламану лінію, послідовні відрізки якої завжди перетинаються під одним і тим же кутом $\alpha = \pi - (\pi/5) = 4\pi/5$, тому що зірка є правильною і її внутрішній кут дорівнює $\pi/5$. Якщо початковий відрізок із номером $n = 0$ має напрям $\theta = 0$, то відрізок із номером n має напрям $\theta = n\alpha$. Фрагмент фрактала зображений на рис. 10.7.

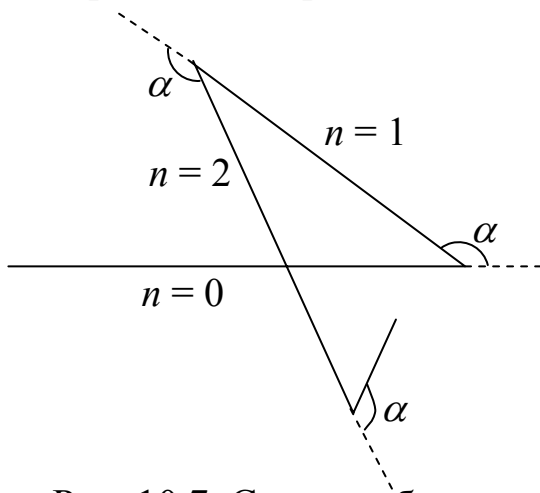


Рис. 10.7. Схема побудови ламаної

При побудові зіркового фрактала ми повинні враховувати правило, за яким визначається довжина n -го відрізка ламаної, якщо відома довжина першого відрізка.

Зображений фрактал на рис. 10.6 складається з відрізків двох різних довжин. Якщо прийняти довжину більшого відрізка за одиницю, то довжина меншого дорівнюватиме r , де r – показник зменшення ($0 < r < 1$). Наприклад, на рис. 10.6 $r = 0,35$. Як видно з рис. 10.6, відрізки з номерами $n = 0, 4, 8, 12, 16$ мають одиничну довжину, а відрізки з номерами $n =$

$1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15, 17, 18, 19$ – довжину r .

Для зіркового фрактала при $p = 3$ (рис. 10.8) правило довжин таке:

$n = 0, 16, 32, \dots$ – довжина 1;

$n = 4, 8, 12, 20, \dots$ – довжина r ;

$n = 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15, \dots$ – довжина $r^2 = r^{p-1}$.

Отже, довжина відрізка з індексом n залежить від кількості множників 4 у числі n . Загальне правило довжин відрізків ламаної для довільного числа кроків p таке:

якщо номер відрізка n не має множника 4, то його довжина r^{p-1} ,

якщо номер відрізка n має 1 множник 4, то його довжина r^{p-2} ,

.....

якщо номер відрізка n має хоча би $p - 1$ множників 4, то довжина відрізка дорівнює 1.

У загальному випадку число відрізків зіркового фрактала обчислюється за формулою $(N + 1)N^{p-1}$, де $N = 4$. Кількість відрізків з одиничною довжиною буде $N + 1$.

Наприклад, при $p = 5$ маємо $1280 = 5 \times 4^4$ – відрізків ламаної, з них 5 відрізків мають одиничну довжину.

Зауважимо, що значення $N = 4$ можна замінити на деяке інше число і тоді, визначивши відповідний кут, на який повинні повертатися відрізки ламаної, можна будувати інші зіркові фрактали.

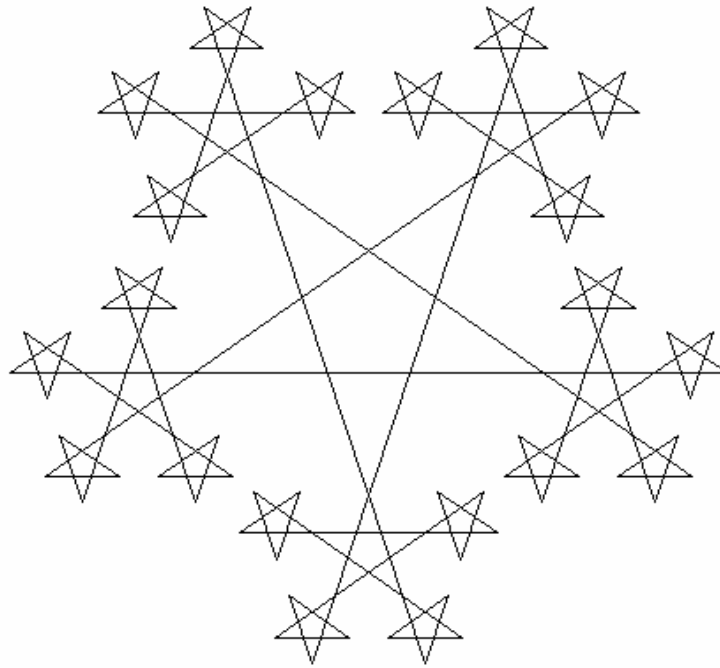


Рис. 10.8. Зірковий фрактал при $p = 3$

Відображення, що описує поворот точки (x_2, y_2) на кут α (проти годинникової стрілки) відносно довільної точки (x_1, y_1) , має вигляд

$$x_3 = (x_2 - x_1)\cos\alpha - (y_2 - y_1)\sin\alpha + x_1, \quad (10.1)$$

$$y_3 = (x_2 - x_1)\sin\alpha + (y_2 - y_1)\cos\alpha + y_1.$$

10.3. Аналіз конструктивних фракталів

Фрактали можна визначити як множини точок інваріантних відносно перетворення поворот-стиск. У простішому випадку стиск – це масштабне зменшення з поворотом (або без нього) і задається лінійним відображенням площини (x, y) в (x', y') , тобто

$$x' = ax + by, \quad y' = cx + dy. \quad (10.2)$$

Якщо $\det(A) = ad - cb \neq 0$, то перетворення (10.2) має єдину нерухому точку $O(0,0)$.

Фрактали можна одержати в результаті перетворень подібності – повороту зі стиском/розтягом. Поворот зі стиском – це добуток перетворень повороту та стиску (порядок не має значення). Розтягування/стиск відносно початку координат (центральне перетворення) задається формулами $x' = cx$, $y' = cy$, а відносно точки (x_0, y_0) – формулами

$$x' = c(x - x_0) + x_0, \quad y' = c(y - y_0) + y_0.$$

При $c > 1$ маємо перетворення розтягування, при $|c| < 1$ – перетворення стиску (для фракталів, як правило, $|c| < 1$). При $c = -1$ – дзеркальне відображення (поворот на 180°).

Перетворення поворот-розтяг відносно початку координат має вигляд $x' = ax - by$, $y' = by + ax$, а відносно довільної точки:

$$x' = ax - by + c, \quad y' = ay + bx + d. \quad (10.3)$$

Величина $Q = a^2 + b^2$ характеризує величину розтягу. При $Q > 1$ маємо розтяг, при $Q < 1$ – стиск.

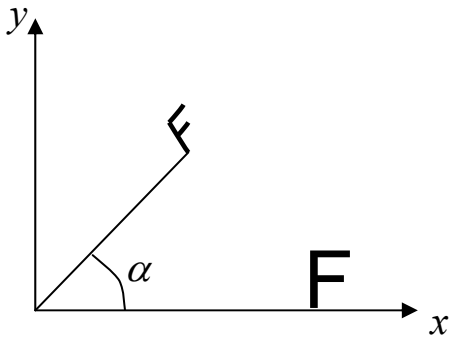


Рис. 10.9. Перетворення поворот-стиск

Перетворення поворот-стиск зображено на рис. 10.9. Як приклад застосування повороту-стиску розглянемо криву Леві, основою якої вибирають відрізок $[0, 1]$, як фрагмент – ламану з двох відрізків P_0P_2, P_2P_1 , де $P_2(0,5; 0,5), P_0(0, 0), P_1(1, 0)$. Перші кроки побудови кривої Леві показано на рис. 10.10, *a*. Наближення фрактала визначається своїми вершинами, тому потрібно на кожному кроці знайти вершини ламаної і

з'єднати їх прямими лініями (рис. 10.10, *б*).

З рис. 10.10, *a* видно, що при побудові фрактала Леві фігурують два перетворення повороту-стиску. Перше перетворення L має центр повороту в точці $(0, 0)$, кут повороту $\pi/4$, коефіцієнт стиску $1/2^{1/2}$. Друге перетворення R має центр у точці $(1, 0)$, кут повороту $-\pi/4$, коефіцієнт стискування $1/2^{1/2}$. Перетворення L та R задаються відповідно формулами (впливає з (10.1))

$$L: \begin{aligned} x' &= (x - y)/2, \\ y' &= (y + x)/2, \end{aligned}$$

$$R: \begin{aligned} x' &= (x + y + 1)/2, \\ y' &= (y - x + 1)/2. \end{aligned}$$

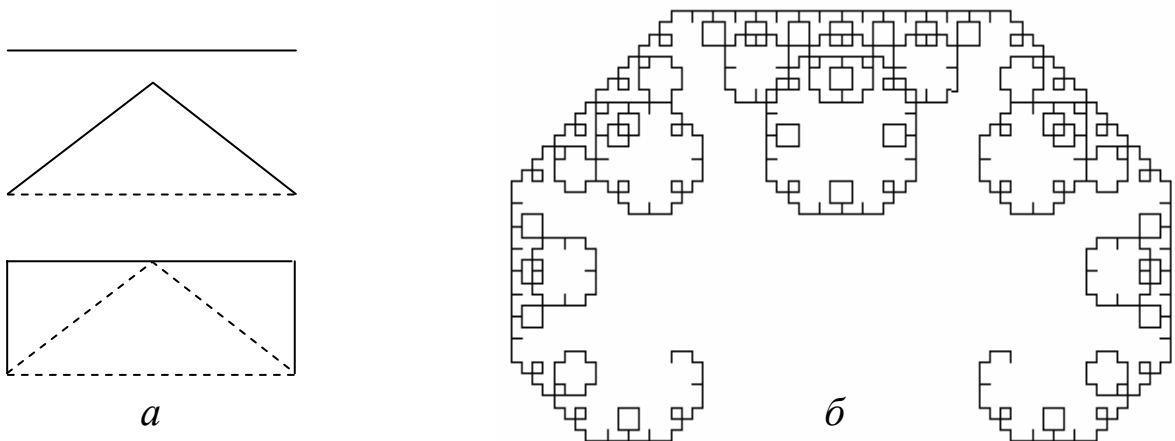


Рис. 10.10. Фрактал Леві ($n = 10$)

Точки фрактала Леві на кожному кроці p можна одержати, наприклад, піддаючи праву точку $(1,0)$ послідовності перетворень L та R . Якщо обираємо довільну точку із фрактала як початкову і здійснюємо над нею перетворення L і R кілька разів, то кількість точок на кожному кроці подвоюється і всі ітерації, отримані з цієї точки, теж будуть точками фрактала.

Розглянемо туку перетворення.

$$\begin{aligned} L: x' &= ax - by, & R: x' &= cx - dy + 1 - c, \\ y' &= ay + bx, & y' &= cy + dx - 1. \end{aligned} \quad (10.4)$$

Перетворення (10.4) є частинним випадком більш загальних перетворень повороту-стиску (10.3). Перетворення L – це поворот-стиск відносно точки $(0, 0)$ з коефіцієнтом $(a^2 + b^2)^{1/2}$, а R – поворот-стиск відносно точки $(1, 0)$ з показником $(c^2 + d^2)^{1/2}$.

Наведемо конкретні приклади таких фракталів [20]:

- a)* $a = d = 0, b = c = 0.7, p = 17$ (рис. 10.11, *a*);
- б)* $a = b = 0.6, c = 0.53, d = 0, p = 18$ (рис. 10.11, *б*).

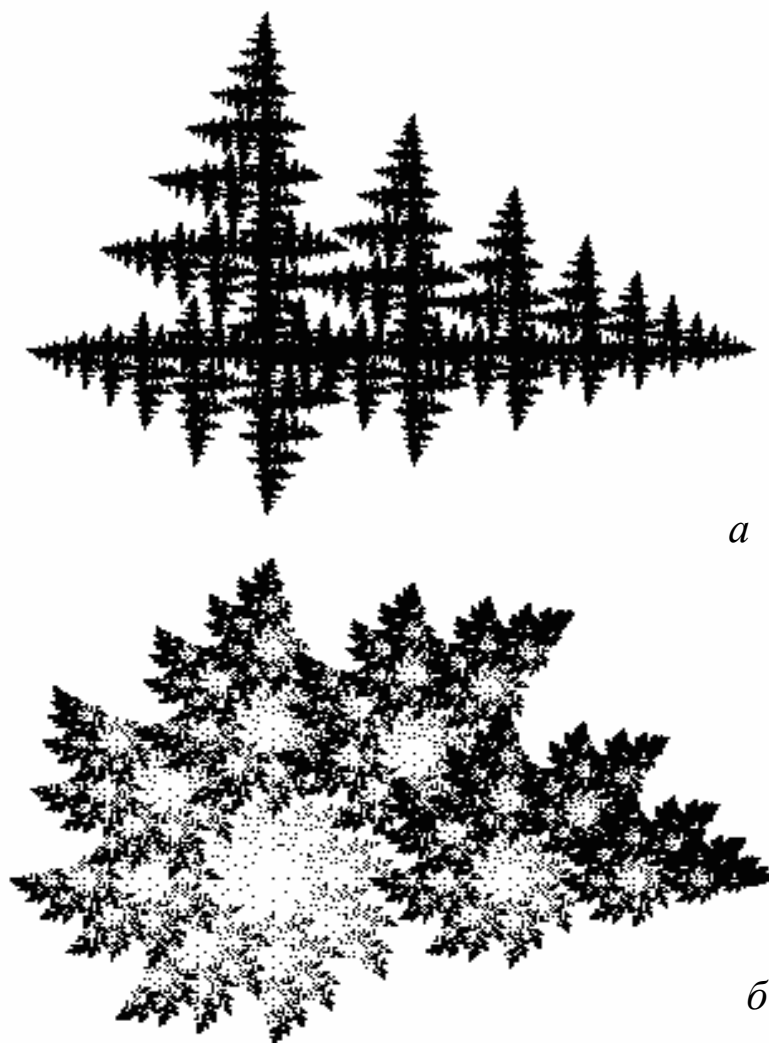


Рис. 10.11. Фрактал, одержаний за допомогою повороту-стиску

Розглянемо приклад фрактала, який має вигляд рослини (стовбур із гілками). Всі елементи рослини зобразимо відрізками прямої.

Для початку ітерацій задаємо стартові координати відрізка. Нехай це будуть точки P_1, P_2 . На кожному кроці ітерації будемо розраховувати координати інших точок. Спочатку знайдемо точку P_3 – це точка P_2 , по-

вернута на кут α відносно точки P_1 (рис. 10.12). Це перетворення задається формулою (10.1). У випадку $\alpha = 0$ стовбур і гілки є прямими. Потім знаходимо точку P_4 , що одержується з P_3 за допомогою перетворення стиску відносно точки P_1 з коефіцієнтом стиску $0 < k < 1$.

Координати точки P_4 знаходяться за формулами

$$x_4 = x_1(1 - k) + x_3k, \quad y_4 = y_1(1 - k) + y_3k.$$

Із точки P_4 будуть виходити нові гілки. Побудуємо ці гілки. Для цього спочатку знаходимо точку P_5 із відрізка P_3P_4 за формулами стиску відносно довільної точки

$$x_5 = x_4(1 - k_1) + x_3k_1, \quad y_5 = y_4(1 - k_1) + y_3k_1, \quad 0 < k_1 < 1.$$

Кінці гілок (точки P_6, P_7) – це точка P_5 , повернута відносно точки P_4 на кути β і $-\beta$. Їхні координати знаходимо за формулами

$$\begin{aligned} x_6 &= (x_5 - x_4)\cos\beta - (y_5 - y_4)\sin\beta + x_4; \\ y_6 &= (x_5 - x_4)\sin\beta + (y_5 - y_4)\cos\beta + y_4; \\ x_7 &= (x_5 - x_4)\cos\beta + (y_5 - y_4)\sin\beta + x_4; \\ y_7 &= -(x_5 - x_4)\sin\beta + (y_5 - y_4)\cos\beta + y_4. \end{aligned}$$

Крім розрахунку опорних точок на кожному кроці ітерацій будемо рисувати один відрізок P_1P_4 . Зазначимо, що в залежності від номера ітерації можна змінювати колір відрізка і його товщину, наприклад, пропорційно його довжині. Величини α, β, k, k_1 є константами, які визначають вигляд фрактала в цілому (рис. 10.13).

Алгоритм побудови фрактала можна записати у вигляді такої рекурсивної процедури `branch` [5].

branch (x1, y1, x2, y2, num)

початок

якщо $\{(\text{num} \neq 0) \cup (x1 - x2)^2 + (y1 - y2)^2 > l_{\min}\}$ *то*

{обчислюємо координати точок $P_3 - P_7$ }:

$x3 := (x2 - x1) * \cos\alpha - (y2 - y1) * \sin\alpha + x1;$

$y3 := (x2 - x1) * \sin\alpha + (y2 - y1) * \cos\alpha + y1;$

$x4 := (1 - k) * x1 + k * x3;$ // точки розгалуження

$y4 := (1 - k) * y1 + k * y3;$

$x5 := (1 - k1) * x3 + k1 * x4;$

$y5 := (1 - k1) * y3 + k1 * y4;$

$x6 := (x5 - x4) * \cos\beta - (y5 - y4) * \sin\beta + x4;$ // вершини гілок

$y6 := (x5 - x4) * \sin\beta + (y5 - y4) * \cos\beta + y4;$

$x7 := (x5 - x4) * \cos\beta + (y5 - y4) * \sin\beta + x4;$

$y7 := -(x5 - x4) * \sin\beta + (y5 - y4) * \cos\beta + y4;$

line (x1, y1, x4, y4); // рисування лінії-стовбура

branch (x4, y4, x3, y3, num); // продовження стовбура

branch (x4, y4, x6, y6, num – 1); // права гілка
branch (x4, y4, x7, y7, num – 1); // ліва гілка
 кінець.

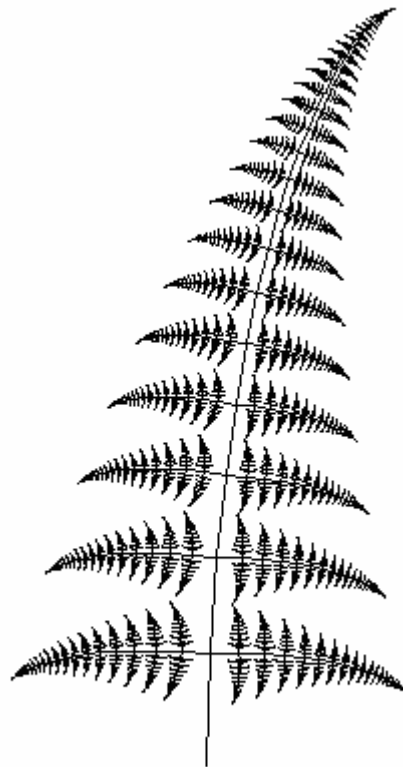
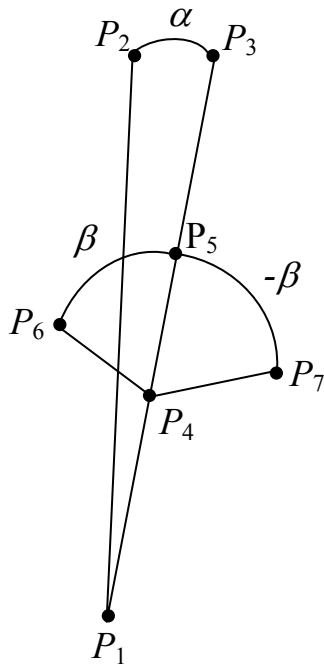


Рис. 10.12. Схема фрактала

Рис. 10.13. Фрактал „гілка папороті”

Тут значення *num* показує ступінь деталізації фрактала. Один цикл ітерації містить багато кроків, що відповідає одному значенню *num*. Числове значення *num* використовується для зупинки ітераційного процесу і для визначення поточного кольору елементів рослин та їх товщини. В даному алгоритмі завершення циклів ітерації відбувається ще і тоді, коли довжина гілки стає меншою від деякої величини l_{\min} .

Наведений метод генерування фракталів називається *методом систем ітеративних функцій* (IFS). Метод IFS з'явився в середині 80-х років як простий засіб одержання фрактальних структур. Тут координати нових точок фрактала знаходяться за формулами

$$x_{k+1} = F_1(x_k, y_k), \quad y_{k+1} = F_2(x_k, y_k),$$

де F_1, F_2 – функції перетворення координат, наприклад афінні перетворення.

Цей метод використовується не тільки для створення зображень, але й для ефективного стиску графічної інформації при записі у файли, оскільки складні зображення (фрактали) одержуються за допомогою простих ітерацій, а опис ітерацій вимагає значно меншого обсягу пам'яті.

Отже, для кодування зображень необхідно розв'язувати обернену задачу. Для графічного зображення необхідно підібрати коефіцієнти афін-

ного перетворення. Цей метод використовується для запису кольорових фотографій у файли зі стискуванням в сотні разів без помітної втрати якості зображення.

10.4. Динамічні фрактали

Вище ми розглядали конструктивні фрактали, для побудови яких використовувалися лінійні двовимірні відображення.

Для утворення динамічних фракталів використовують нелінійні відображення, які належать до дискретних динамічних систем, тому надалі будемо користуватися термінологією динамічних систем: фазовий портрет, атрактор тощо. Відомо, що нелінійні динамічні системи володіють кількома стійкими станами, або, як кажуть, атракторами, причому кожний стійкий стан має деяку область початкових станів, з яких система потрапляє в ці стійкі стани, тобто фазовий простір розбивається на області притягування атракторів (басейни атракторів). Якщо фазовий простір двовимірний, то, зафарбовуючи області притягування різними кольорами, можна одержати кольоровий фазовий портрет цієї системи. Змінюючи алгоритм вибору кольору, отримуємо складні фрактальні структури з надзвичайними візерунками. Але аналіз нелінійних відображень супроводжується значними труднощами (це помітно навіть на прикладі простіших нелінійних відображень).

Одновимірні комплексні відображення породжують найбільш популярні фрактали Жуліа, Мандельброта, Ньютона та ін., які можна побудувати за допомогою примітивних алгоритмів.

У 1918 р. Гастон Жуліа опублікував мемуари про комплексні відображення, але оскільки в цій роботі були відсутні зображення, то робота Жуліа майже півстоліття не використовувалася.

За останні роки інтерес до таких відображень зріс, завдяки, зокрема, красивим графічним зображенням множин Жуліа, побудованих за допомогою примітивних алгоритмів.

10.4.1. Множини Жуліа і Мандельброта

Множини Жуліа одержуються за формулою

$$z_{k+1} = z_k^2 + c, \quad k = 0, 1, 2, \dots, \quad (z = x + iy, \quad c = a + ib, \quad i = \sqrt{-1}),$$

де $z_0 \neq 0$ – різні точки комплексної площини C , c – деяка фіксована комплексна точка.

Фрактал Мандельброта одержується за цією ж ітераційною формулою $z_{k+1} = z_k^2 + c$, $k = 0, 1, 2, \dots$, але з фіксованим початковим значенням $z_0 = 0$ і при різних значеннях $c \in C$. Тут ітерації виконуються для кожної стартової точки c прямокутної або квадратної області.

Причому стартові значення координат точки c – це координати точки зображення.

Для зручності запишемо ітераційні формули у вигляді

$$z_{k+1} = f(z_k), \quad k = 0, 1, 2, \dots,$$

де $f: C \rightarrow C$ – поліном степеня $n \geq 2$.

Позначимо через f^k k -ту ітерацію функції f , тобто $f^k = f(f(\dots(f(z))))$. Якщо $f(\omega) = \omega$, то точка ω називається нерухомою точкою відображення f . Якщо $f^p(\omega) = \omega$, $p > 1$, то точка ω називається періодичною точкою відображення f . Найменше p , таке, що $f^p(\omega) = \omega$, називається періодом точки ω . Множину точок $\{\omega, f(\omega), f^2(\omega), \dots, f^{p-1}(\omega)\}$ називають орбітою періоду p .

Нехай ω – періодична точка періоду p з $(f^p(\omega))' = \lambda$ (штрих позначає комплексне диференціювання), тоді точка ω (або орбіта періоду p) називається притягувальною, якщо $0 < |\lambda| < 1$; індиферентною, якщо $|\lambda| = 1$; відштовхувальною, якщо $|\lambda| > 1$.

Множина Жуліа $J(f)$ визначається як замикання множини відштовхувальних періодичних точок відображення f . Доведено, що множина Жуліа $J(f)$ є інваріантною множиною для відображення f та оберненого відображення f^{-1} , тобто $J = f(J) = f^{-1}(J)$ і множина J непорожня та компактна. Більше того, ітерації f поведуться хаотично на J і J , як правило, є фракталом.

Розглянемо простіший випадок, коли $f(z) = z^2$, тоді $f^p(z) = z^{2^p}$. Періодичні точки періоду p відображення f визначаються з рівняння

$$z^{2^p} = z, \text{ або } z(z^{2^p-1} - 1) = 0.$$

Точки ω , що є коренями рівняння $z^{2^p-1} = 1$, визначаються за формулою Муавра і лежать на одиничному колі, причому в цих точках $|(f^p(\omega))'| = 2^p > 1$. Очевидно, що ітерації $f^k(z)$ залишаються на $J(f)$, якщо точки z брати на колі $|z| = 1$, тобто $J = f(J) = f^{-1}(J)$. Якщо $|z| < 1$, то $f^k(z) \rightarrow 0$ при $k \rightarrow \infty$ і $f^k(z) \rightarrow \infty$ при $k \rightarrow \infty$, якщо $|z| > 1$.

Таким чином, множина Жуліа $J(f)$ – це коло одиничного радіуса $|z| = 1$ (в цьому особливому випадку $J(f)$ є виродженим фракталом (рис. 10.14, а). Множина Жуліа є границею між множинами точок z , ітерації яких прямують до нуля і до нескінченності.

Якщо розглядати функцію $f(z) = z^2 + c$, де c – невелике комплексне число, то все ж таки ще $f^k(z) \rightarrow \omega$ при $k \rightarrow \infty$, якщо $|z|$ достатньо малі, а ω – нерухома точка, близька до нуля, і $f^k(z) \rightarrow \infty$, якщо $|c|$ достатньо велике. Отже, множина Жуліа – це границя між двома типами поведінки, тепер J вже є фракталом, схематично зображеним на рис. 10.14, б.

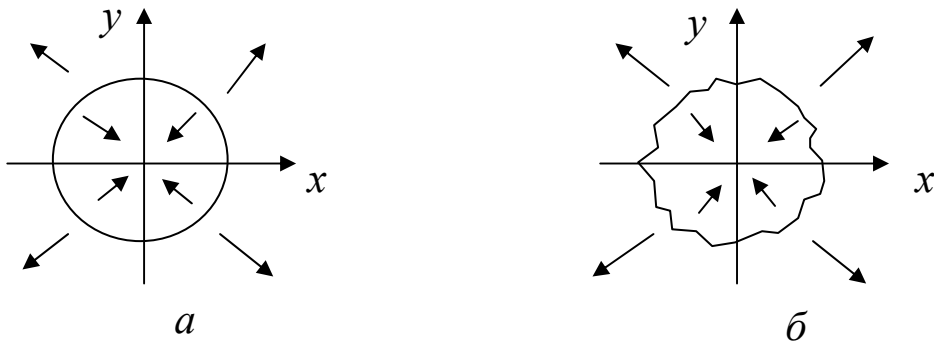


Рис. 10.14. Множини Жуліа: $a - c = 0$; $b - c \neq 0$

10.4.2. Фрактали Жуліа

Як відомо, фрактали Жуліа на множині комплексних чисел описуються рекурентним співвідношенням $z_{k+1} = z_k^2 + c$, яке після виділення дійсної та уявної частин перепишеться у вигляді

$$\begin{cases} x_{k+1} = x_k^2 - y_k^2 + a, \\ y_{k+1} = 2x_k y_k + b, \end{cases} \quad k = 0, 1, 2, \dots \quad (10.5)$$

Це відображення при різних a і b дає множину фракталів, які відповідають множині Жуліа $J(a, b)$. Множина Жуліа $J(a, b)$ – це границя області притягування до нескінченності. Очевидно, при $a = 0$, $b = 0$ множина Жуліа – це коло одиничного радіуса. При малих a , b множина $J(a, b)$ вже не має форми кола і, як правило, є фракталом. Множина $J(a, b)$ при відображенні (10.5) переходить у саму множину $J(a, b)$.

З формули (10.5) видно, що фрактали Жуліа симетричні відносно початку координат, а якщо $b = 0$, то вони симетричні відносно двох осей (цей факт потрібно використати при написанні програм).

Фрактали Жуліа дають багато чудових графічних зображень, особливо коли вести побудову кольорових зображень. При побудові кольорових фракталів використовується умова наближення орбіти до атрактора (для фракталів Жуліа – до нескінченності).

Колір кожної точки на екрані визначається кількістю ітерацій, потрібних орбіті, щоб наблизитися до атрактора. Якщо використовувати 16 кольорів, то номер кольору можна визначати, наприклад за формулою $k \bmod 16$, де k – кількість проведених ітерацій.

Оскільки для наближення до атрактора орбіті може бути потрібна велика кількість ітерацій, то в програмі необхідно задати деяке граничне значення кількості ітерацій K_{\max} , при якому вважаємо, що орбіта піді-

йшла до атрактора. Умовою того, що орбіта прямує до нескінченності, є виконання нерівності $x_k^2 + y_k^2 > 4$ при $k < K_{\max}$.

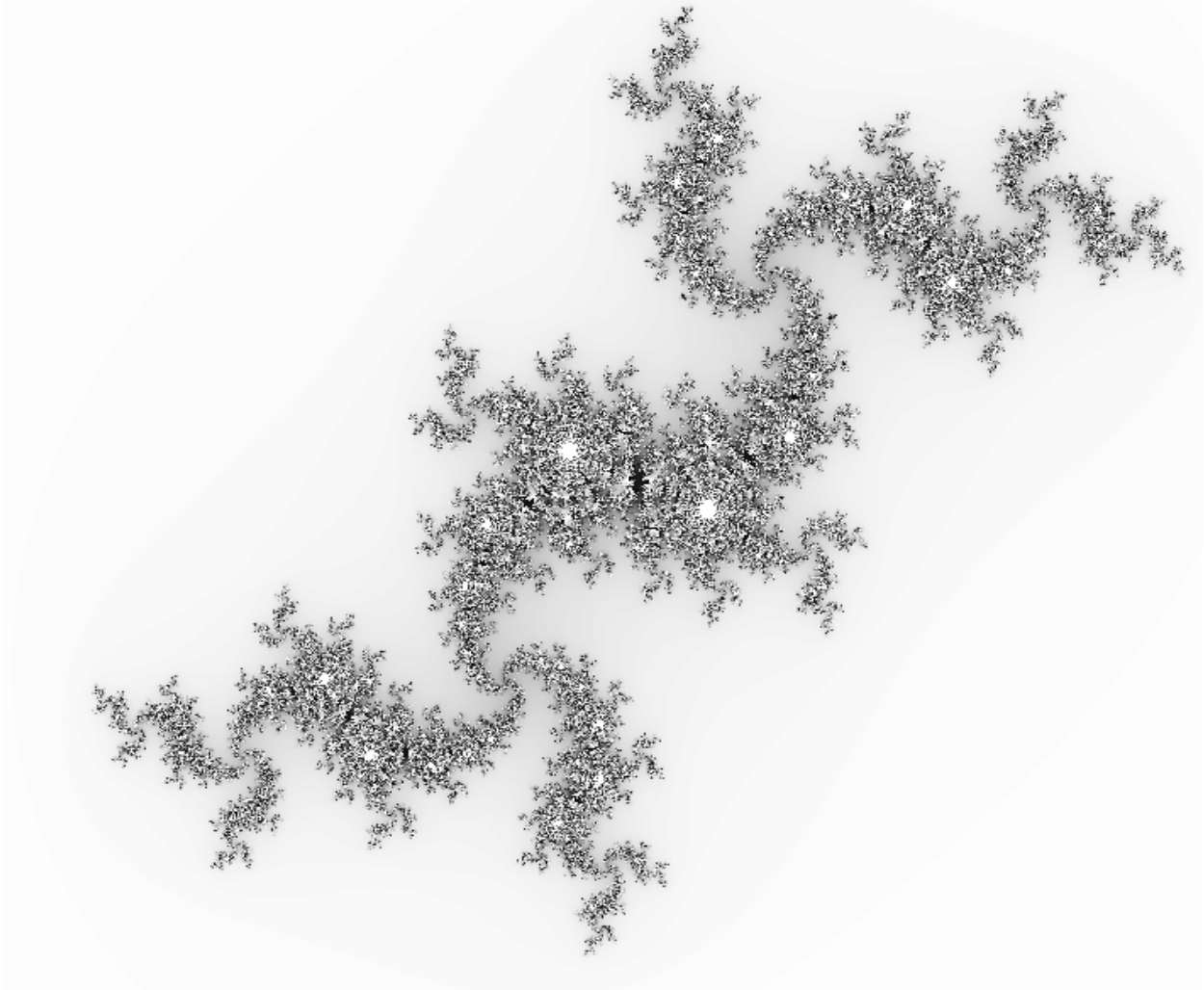


Рис. 10.15. Фрактал Жуліа

Якщо при $k = K_{\max}$ $x_k^2 + y_k^2 \leq 4$ (орбіта не покидає круг за максимальну кількість ітерацій), то точка $z_0 = x_0 + iy_0$ належить множині Жуліа (зафарбовуємо в чорний колір). Цикл ітерацій для фрактала можна виконувати для невеликих x_0 та y_0 , наприклад, $x_0 \in [-2,2; 1]$, $y_0 \in [-1,2; 1,2]$; як параметри a , b можна взяти 1) $a = -0,22$; $b = -0,74$ (рис. 10.15); 2) $a = 0,11$; $b = 0,66$. Для того, щоб отримати зображення в растрі, необхідно перераховувати координати x_0 , y_0 в піксельні.

10.4.3. Фрактали Мандельброта

Фрактали Жуліа бувають зв'язні та незв'язні. В іншому випадку фрактал Жуліа являє собою незчисленну множину дискретних точок (фрактали такого типу називають “порох Фату” – за ім'ям математика Фату).

Якщо фрактал зв'язний, то він складається з набору ліній, інколи – з однієї замкненої кривої, інколи – з петель усередині петель тощо. Зв'язність для фракталів Жуліа залежить від значень параметрів a , b . Мандельброт указав, як знайти множину параметрів на площині (a, b) , для яких фрактал Жуліа зв'язний. Ключ для такої перевірки такий: потрібно перевірити, чи орбіта, що виходить із точки (a, b) , прямує до нескінченності. Якщо ця орбіта прямує до нескінченності, то $J(a, b)$ – незв'язний.

Множина всіх точок (a, b) , для яких орбіта не прямує до нескінченності, складає множину Мандельброта, тобто точка $c = a + ib$ належить множині Мандельброта тоді і тільки тоді, коли усі точки рекурентної послідовності $z_{k+1} = z_k^2 + c$ скінченні й утворюють збіжну послідовність. Однак не існує простої формули, яка б дозволила визначити належність точки c множині Мандельброта. Єдиний спосіб перевірки – це виконання ітерацій. Якщо зафарбовувати точки c у залежності від кількості ітерацій, то одержимо зображення фракталів Мандельброта.

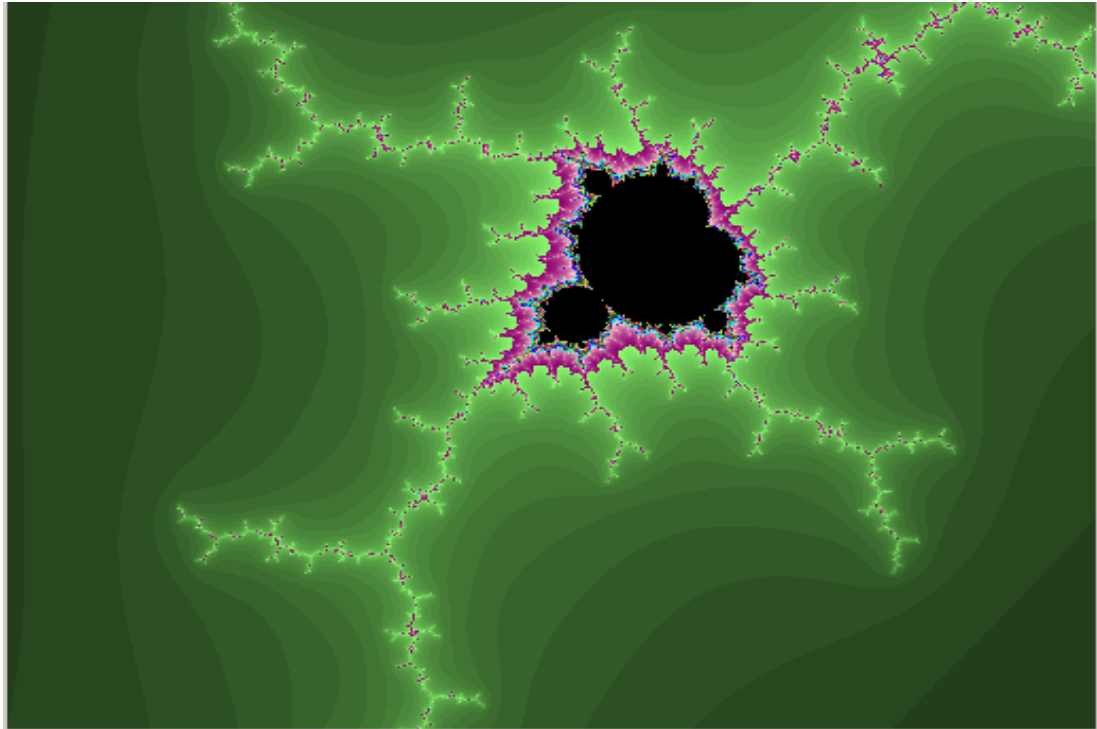
Графічна інтерпретація множини Мандельброта дивовижно красива і багатоманітна. Границею цієї множини є фрактал Мандельброта, бо в границі присутні фрагменти, подібні до усього фрактала (рис. 10.16).

Для побудови фракталів Мандельброта на екрані дисплея задамо прямокутник $(2n + 1) \times (2m + 1)$ пікселів. Кожний піксель відповідає парі значень (a, b) . Щоб одержати множину Мандельброта, вибираємо значення (a, b) так, щоб $a \in [-2, 5; 1, 5]$, $b \in [-2, 2]$. Стартові значення a, b – це координати зображення. Щоб одержати растрове зображення на екрані, необхідно перерахувати координати (a, b) у піксельні координати. Далі підібрані значення (a, b) для кожного пікселя перевіряємо на умову, чи ітераційний процес (10.5) із вибраними початковими значеннями $x_0 = a, y_0 = b$ прямує до нескінченності.

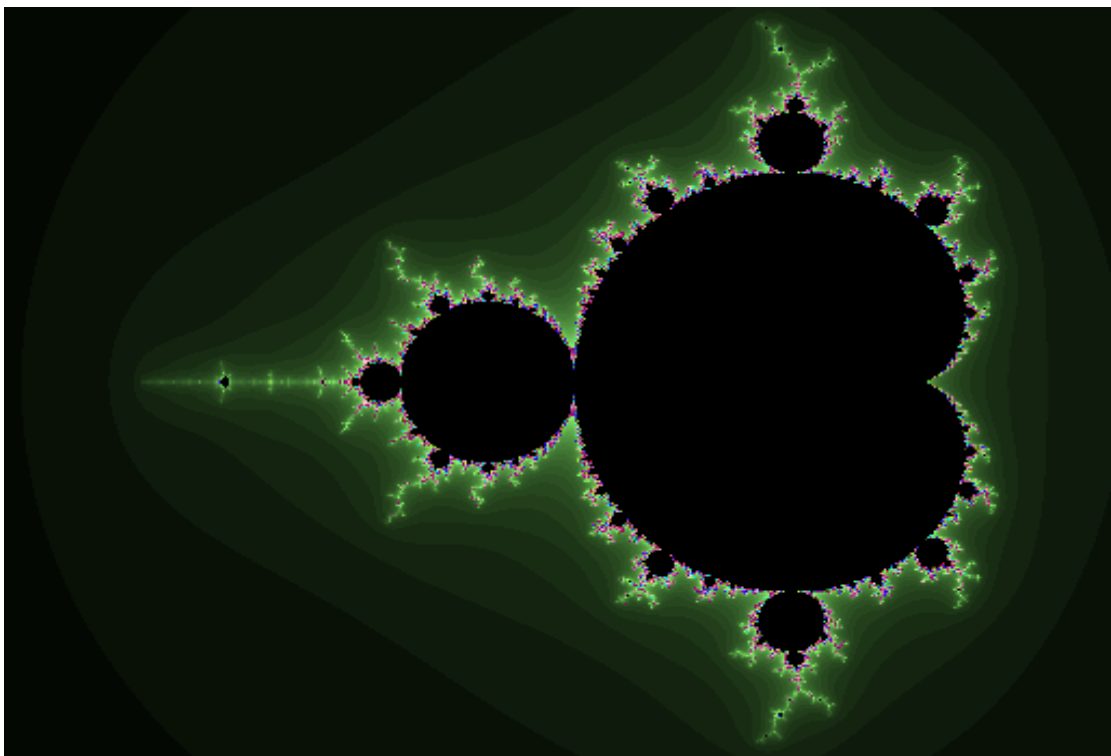
Якщо за максимальну кількість ітерацій K_{\max} (K_{\max} вибираємо заздалегідь) орбіта не залишає круга $x^2 + y^2 \leq 4$, то точка (a, b) належить множині Мандельброта. На екрані ці точки зафарбовуємо чорним кольором.

Якщо за k ітерацій, де $k < K_{\max}$, орбіта, що виходить з точки (a, b) , покидає указаний круг, то вважаємо, що вона прямує до нескінченності. Іншими словами, якщо при деякому k виконується нерівність $|z_k| > r_{\min}$, де $r_{\min} = 2$ – мінімальний радіус розбіжності множини Мандельброта, то послідовність розбігається ($\lim_{k \rightarrow \infty} |z_k| = \infty$). Цей факт математично строго доведений.

Колір точки (a, b) вибираємо, наприклад, за формулою $k \bmod 256$, якщо використовуємо 256 кольорів, або шляхом задання палітри.



$$a \in [-0,19920; -0,12954], \quad b \in [1,01480; 1,06707]$$



$$a \in [-2,4; 0,8], \quad b \in [-1,2; 1,2]$$

Рис. 10.16. Фрактал Мандельброта

Зауваження. Якщо кожній точці (a, b) поставити у відповідність кількість ітерацій k , то одержимо функцію $k = \Phi(a, b)$ і тоді можна побудувати об'ємне зображення множини Мандельброта, яке виглядатиме як гірська скала, водоспад чи гірська печера тощо.

10.4.4. Фрактали Ньютона

Для знаходження комплексних коренів нелінійного рівняння $f(z) = 0$ можна використовувати алгоритм Ньютона, який має вигляд

$$z_{k+1} = z_k - f(z_k) / f'(z_k), \quad k = 0, 1, 2, \dots \quad (10.6)$$

Якщо вдало підібрати z_0 , то за допомогою попередньої формули одержуємо послідовність z_0, z_1, z_2, \dots , яка збігається до кореня рівняння.

Якщо за $f(z)$ взяти поліном $z^n - a$, то (10.6) матиме вигляд

$$z_{k+1} = \frac{(n-1)z_k^n + a}{nz_k^{n-1}}, \quad k = 0, 1, 2, \dots \quad (10.7)$$

Підбираючи початкове значення z_0 , можна знаходити $\sqrt[n]{a}$. Крім цього, відомо, що рівняння $z^n = a$ має n коренів, розміщених на колі радіусом $\sqrt[n]{a}$ із кутовим інтервалом $\frac{2\pi}{n}$, які обчислюються за формулою

$$z_{k+1} = \sqrt[n]{a} \left(\cos \frac{2k\pi}{n} + i \sin \frac{2k\pi}{n} \right), \quad k = 0, 1, \dots, n-1.$$

Корені $\sqrt[n]{a}$ є стійкими нерухомими точками відображення (10.7).

Основна проблема в застосуванні методу Ньютона пов'язана з вибором початкового наближення. Для цього потрібно вказати області притягання нерухомих точок відображення (10.7). Границя областей притягання різних коренів має фрактальну структуру. Побудова фракталів Ньютона на комп'ютері аналогічна побудові фракталів Мандельброта. Наприклад, при $a = 1$ вибираємо множину початкових точок $z_0 \in [-2, 2] \times [-2, 2]$. Далі для кожного пікселя екрана підбираємо значення z_0 і перевіряємо, чи орбіта, що виходить з точки z_0 , наближається до нерухомих точок (коренів рівняння). Умовою припинення циклу ітерацій є наближення значень $|z^n - 1|$ до нуля (наприклад, $|z^n - 1| < 0,01$) або при досягненні $k = K_{\max}$.

При $a = 1, n = 3$ формули (10.7) набувають вигляду

$$z_{k+1} = \frac{2z_k^3 + 1}{3z_k^2}, \quad k = 0, 1, 2, \dots$$

Відокремивши дійсну та уявну частини, одержуємо

$$x_{k+1} = \frac{2}{3}x_k + \frac{x_k^2 - y_k^2}{3(x_k^2 + y_k^2)^2}, \quad y_{k+1} = \frac{2}{3}y_k \left(1 - \frac{x_k}{(x_k^2 + y_k^2)^2} \right).$$

Піксель, що відповідає значенню (x_k, y_k) , зафарбовуємо в колір $k \bmod 256$, де k – номер ітерації, починаючи з якого, виконується нерівність $|z_k^n - 1| < 0,01$.

Отже, моделювання ітераційних процесів обчислювальної математики практично довело фрактальність областей збіжності ітерацій до коренів нелінійних рівнянь.

Можна придумати багато інших ітераційних процедур вигляду

$$z_{\kappa+1} = F(z_{\kappa}, c), \quad \kappa = 0, 1, 2, \dots,$$

що породжують різноманітні фрактальні зображення.

10.5. Застосування фракталів

За останні 20 років фрактали стали досить популярними. Вони знаходять широке застосування в багатьох областях КГ, наприклад у комп'ютерному дизайні. Фрактальна геометрія використовується при моделюванні природних явищ, при вивченні нелінійних динамічних систем, при обробці та класифікації сигналів складної форми, при аналізі коливання курсу валют. Фрактали застосовуються у фізиці твердого тіла, електроніці, фізиці суцільних середовищ (так з'явилася теорія фрактальних тріщин).

Структури, що схожі на фрактали, можна знайти в навколишньому середовищі: контури хмар, контури морських берегів, турбулентні потоки в рідинах, тріщини в породах, зображення структур деяких речовин, отримані під мікроскопом тощо. Фрактали описують природні форми більш витончено і точніше, ніж об'єкти евклідової геометрії.

Останнім часом методи ітеративних функцій застосовуються не тільки для створення зображень, але й для ефективного стиску графічних зображень при записі у файли. Ідея фрактального стиску полягає в тому, що в графічному зображенні знаходять подібні області і зберігають у файлі тільки коефіцієнти перетворення подібності. При цьому необхідно розв'язувати обернену задачу: для зображення підібрати коефіцієнти перетворення, але проблема полягає в тому, що досить важко знайти ці перетворення.

Графічні зображення кодуються кількома простими перетвореннями, тобто коефіцієнтами цих перетворень. Наприклад, якщо деяке зображення закодуємо двома афінними перетвореннями вигляду

$$x' = ax + by + c, \quad y' = dx + ey + f,$$

то ми його можемо однозначно визначити за допомогою 12 коефіцієнтів.

Такі методи стиску графічної інформації називають методами фрактального стиску. Ці алгоритми дозволяють стискати інформацію в сотні разів без помітного погіршення зображення.

Контрольні питання та завдання

1. Дайте означення фрактала за Мандельбротом.
2. Які фрактали називаються конструктивними? Наведіть приклади.
3. Які фрактали називаються динамічними? Наведіть приклади.
4. Опишіть принципи побудови геометричних фракталів, зокрема фракталів Коха, Леві, Мінковського.
5. Опишіть структуру зіркових фракталів.
6. Які афінні перетворення використовуються при побудові зіркових фракталів?
7. Запишіть алгоритм побудови фрактала „гілка папороті”.
8. Як описуються фрактали Жуліа та Мандельброта? Яка між ними різниця? Як їх побудувати?
9. Опишіть процес побудови фракталів Ньютона на комп’ютері.
10. За якою ознакою зупиняють обчислювальний процес при побудові фракталів Ньютона?
11. Назвіть області застосування фракталів.
12. Як на вашу думку можна будувати стохастичні фрактали?

Вправи і задачі для самостійного виконання

1. Модифікувати алгоритм побудови фрактала „гілка папороті” на випадок урахування товщини стовбура.
2. Розробити алгоритм побудови різнокольорового фрактала „гілка папороті”.
3. Написати програму побудови фрактала, який одержується за ітераційною формулою $z_{k+1} = i(z_k^2 + 1)$, $k = 0, 1, 2, \dots$.
4. Розробити алгоритм побудови зіркового фрактала, який складається з квадрата та гірлянди з чотирьох менших квадратів у трьох вільних кінцях яких містяться ще менші квадрати і т.д. Сформулюйте правило довжин відрізків ламаної для такого фрактала.
5. Придумайте одновимірні комплексні відображення, які на вашу думку можуть породити цікаві динамічні фрактали.
6. Побудувати систему ітеративних функцій (IFS) для трикутника Серпінського.
7. Написати фрагмент програми, яка для систем ітеративних функцій буде випадковим чином вибирати функцію із списку n функцій.
8. Побудуйте систему ітеративних функцій (IFS) для трикутника Серпінського.
9. Побудувати фрактали Жуліа та Мандельброта у випадку коли z – гіперкомплексне число.

Тема 11. Моделювання 2D/3D-перетворень

Конструювання та виконання різноманітних дій з геометричними об'єктами є центральною задачею в графічних системах. Тому вибір математичних методів і алгоритмів для її реалізації суттєво впливає на ефективність графічної системи. У сучасній комп'ютерній графіці досить широко використовується метод координат, оскільки графічне зображення складається з пікселів, які задаються координатами. Крім цього, координати використовуються для опису розміщення об'єктів та для створення зображень шляхом перетворень з однієї системи координат в іншу. При цьому використовуються двовимірні та тривимірні системи координат.

У комп'ютерній графіці все, що стосується двовимірного випадку, позначають символом 2D (2-dimension), а тривимірного – 3D (3-dimension).

На координатному методі базується й аналітична геометрія, яку можна вважати фундаментом КГ. Підходи та ідеї геометрії паралельно з постійно розширюючими можливостями комп'ютерних технічних засобів є істотним джерелом розвитку КГ та її ефективного використання на практиці. Інколи навіть простіші геометричні знання допомагають у розв'язанні складних графічних задач. Усі маніпуляції над графічними зображеннями (перенесення, поворот, масштабування тощо) можна виконати з допомогою відповідного математичного апарату геометрії. Цей апарат повинен бути адаптований до задач комп'ютерної графіки. Тому для засвоєння методів КГ важливе вивчення методів перетворення координат. У цьому розділі описано геометричні перетворення координат на площині та в просторі.

11.1. Афінні перетворення на площині

У прямокутній системі координат на площині точка M визначається своїми координатами – впорядкованою парою чисел (x, y) або матрицею розміром 1×2 .

Якщо на площині ввести ще одну систему координат, то точці $M(x, y)$ ставиться у відповідність нова пара чисел (x', y') . Перехід від однієї системи координат на площині до іншої задається формулами

$$x' = ax + by + m, \quad y' = cx + dy + n, \quad (11.1)$$

де a, b, c, d, m, n – довільні числа, $\Delta = ad - bc \neq 0$.

Перетворення, що задаються формулами (11.1), називаються *афінними*. Афінні перетворення мають такі властивості:

- n -вимірний об'єкт відображається в n -вимірний, точка – в точку, лінія – в лінію, поверхня – в поверхню;

- зберігається паралельність ліній і площин;
- зберігаються пропорції паралельних об'єктів (довжин відрізків на паралельних прямих і площ на паралельних площинах).

Ці властивості дозволяють будувати прообрази полігонів на площині й поліедрів у просторі за скінченним набором точок – їх вершин. Зазначимо, що для однозначного визначення коефіцієнтів перетворення (11.1) необхідно задати три пари неколінеарних точок, які повинні відображатися одна в одну перетворенням (11.1).

Формули (11.1) можна розглядати двояко: або зберігається точка, а змінюється система координат (рис. 11.1, а), або змінюється точка, а система координат зберігається (рис. 11.1, б). В останньому випадку формули (11.1) задають відображення точки $M(x, y)$ у точку $M'(x', y')$.

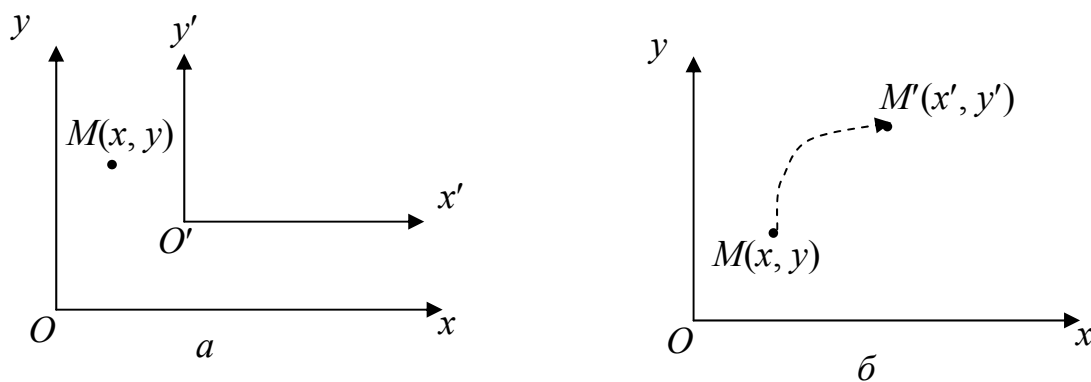


Рис. 11.1. Перетворення координат

В афінних перетвореннях площини особливу роль відіграють декілька важливих окремих випадків, а саме

- паралельний зсув координат точки на вектор (m, n) в даній системі координат (рис. 11.2, а) визначається за формулами:

$$x' = x + m, \quad y' = y + n; \quad (11.2)$$

- зсув системи координат на вектор (m, n) (рис. 11.2, б) задається формулами

$$x' = x - m, \quad y' = y - n.$$

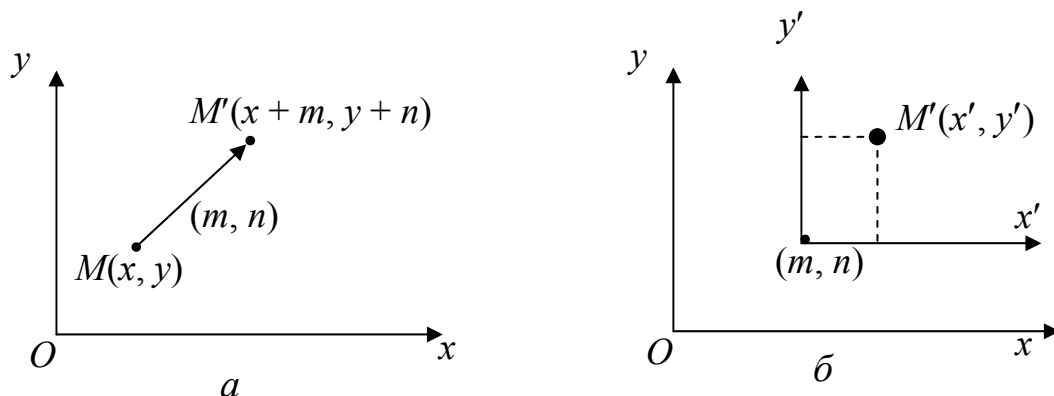


Рис. 11.2. Перетворення зсуву

- поворот точки відносно початку координат на кут φ проти годинникової стрілки (рис. 11.3, а) описується формулами

$$x' = x \cos \varphi - y \sin \varphi, \quad y' = x \sin \varphi + y \cos \varphi; \quad (11.3)$$

- поворот системи координат на кут φ проти годинникової стрілки (рис. 11.3, б) задається формулами

$$x' = x \cos \varphi + y \sin \varphi, \quad y' = -x \sin \varphi + y \cos \varphi;$$

- розтяг/стиск вздовж координатних осей можна записати формулами

$$x' = ax, \quad y' = dy. \quad (11.4)$$

Якщо в (11.4) $a = d$, то маємо пропорційне масштабування, якщо $a \neq d$, то масштабування – непропорційне. При $a = d > 1$ відбувається збільшення зображення, при $a = d < 1$ – рівномірний стиск. При $a = 1, d = -1$ одержуємо дзеркальне відображення відносно осі x , при $a = -1, d = 1$ – дзеркальне відображення відносно осі y .

В аналітичній геометрії доведено, що довільне афінне відображення (11.1) можна задати за допомогою композиції елементарних відображень (11.2) – (11.4).

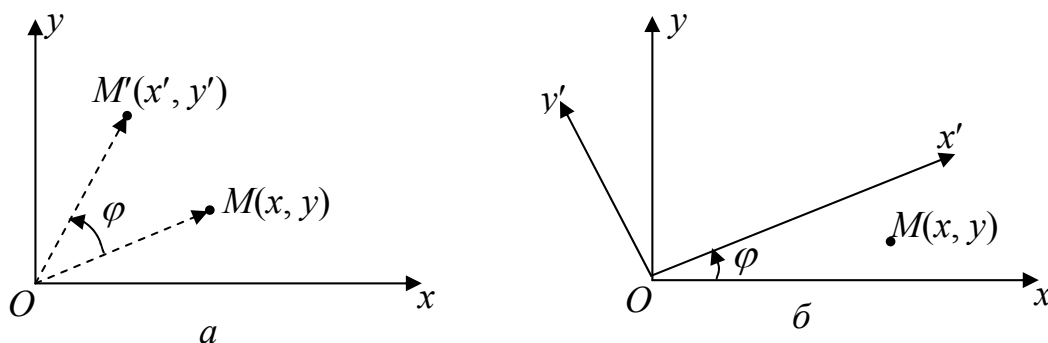


Рис. 11.3. Перетворення повороту

Для ефективного використання формул (11.2) – (11.4) у задачах КГ переходять до матричного запису цих формул. Наприклад, для перетворення (11.3) маємо:

$$X' = X \cdot R = (x \ y) \begin{pmatrix} \cos \varphi & \sin \varphi \\ -\sin \varphi & \cos \varphi \end{pmatrix}. \quad (11.5)$$

Тобто перетворення повороту відносно початку координат у додатному напрямку на довільний кут φ задається матрицею

$$R = \begin{pmatrix} \cos \varphi & \sin \varphi \\ -\sin \varphi & \cos \varphi \end{pmatrix}, \quad \det R = 1 \neq 0.$$

Зауважимо, що повороти вважаються додатними, якщо вони здійснюються проти годинникової стрілки.

У матричному вигляді можна записати і перетворення масштабування

$$X' = X \cdot S, \text{ де } S = \begin{pmatrix} a & 0 \\ 0 & d \end{pmatrix}, \quad a, d \neq 0.$$

Матричний запис геометричних перетворень дає можливість здійснювати обернені перетворення, тобто точку (об'єкт) X' можна легко повернути в початковий стан X . Наприклад, для повороту обернене перетворення $X = X' R^{-1}$ задається матрицею

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

Зауважимо, що обернена матриця R^{-1} водночас є транспонованою до R (такі матриці називаються ортогональними). Цей факт є досить важливим для швидкодії алгоритмів.

Але, на жаль, паралельне перенесення (11.2) не вдається задати перетворенням вигляду

$$X' = X \cdot T, \tag{11.6}$$

тим більше, не можна подати довільні сумарні афінні перетворення в матричній формі. Щоб задати перетворення (11.1) у матричному вигляді (11.6), необхідно перейти до однорідних координат.

Однорідні координати вводяться штучно: довільній точці $M(x, y)$ площини ставиться у відповідність точка $M'(x, y, 1)$ або $M'(hx, hy, h)$, $h \neq 0$ в просторі. Фіктивна z -координата має значення скалярної константи, найчастіше $h = 1$.

Вектор із координатами (hx, hy, h) є напрямним вектором прямої, що проходить через точки $O(0, 0, 0)$ і $M'(x, y, 1)$. Ця пряма перетинає площину $z = 1$ в точці $(x, y, 1)$, яка однозначно визначає точку (x, y) координатної площини Oxy (рис. 11.4).

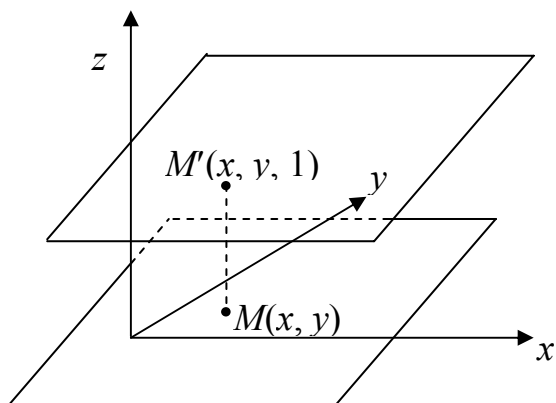


Рис. 11.4. Однорідні координати

Отже, між однорідними координатами $(x, y, 1)$ точки та її декартовими координатами (x, y) існує взаємно однозначна відповідність.

Еквівалентність векторів

$$(hx, hy, h) \sim (x, y, 1)$$

означає, що вони мають однакові декартові координати.

Для зведення однорідного вектора до декартової форми необхідно помножити всі координати на $\frac{1}{h}$, тобто поділити на h .

Над однорідними координатами за спеціальними формулами можна проводити операції, і тільки на останньому етапі перед виведенням зображення на екран необхідно перейти до декартових координат.

Основні операції з однорідними координатами:

- додавання/віднімання

$$\begin{aligned} (x_1, y_1, h_1) + (x_2, y_2, h_2) &\sim \left(\frac{x_1}{h_1}, \frac{y_1}{h_1}, 1 \right) + \left(\frac{x_2}{h_2}, \frac{y_2}{h_2}, 1 \right) = \\ &= \left(\frac{x_1}{h_1} + \frac{x_2}{h_2}, \frac{y_1}{h_2} + \frac{y_2}{h_2}, 1 \right) \sim (x_1 h_2 + x_2 h_1, y_1 h_2 + y_2 h_1, h_1 h_2); \end{aligned}$$

- множення вектора на скаляр

$$\alpha(x, y, h) \sim \alpha \left(\frac{x}{h}, \frac{y}{h}, 1 \right) = \left(\frac{\alpha x}{h}, \frac{\alpha y}{h}, 1 \right) \sim (\alpha x, \alpha y, h).$$

Застосування однорідних координат дає багато зручностей при розв'язуванні графічних задач, наприклад для пристроїв, що працюють з цілими координатами для точки $M(0,5; 0,1)$ можна вибрати $h = 10$ і мати справу з однорідними цілими координатами $M'(5; 1; 10)$.

Крім цього, однорідні координати дозволяють записувати безмежно віддалені точки простору, уникаючи переповнення розрядної сітки ЕОМ, завдяки нормалізації чисел. За допомогою однорідних координат можна оперувати точкою, що знаходиться в нескінченності.

Але основною метою введення в розгляд однорідних координат у КГ є зручність при заданні геометричних перетворень у матричній формі.

За допомогою однорідних координат і матриць третього порядку можна описати довільне афінне перетворення. Формули (11.1) можна подати в матричній формі (11.6), тобто

$$(x' \ y' \ 1) = (x \ y \ 1) \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{pmatrix}, \quad (11.7)$$

де елементи a, b, c, d визначають масштабування і поворот, а m, n – сумарний зсув. Отже, всі перетворення зсуву, масштабування, повороту відносно початку координат в однорідних координатах мають однакову форму – це добуток вектора вихідних координат на матрицю перетворення.

Випишемо матриці основних елементарних перетворень:

- матриця повороту (rotation) точки відносно початку координат у додатному напрямку:

$$R = R(\varphi) = \begin{pmatrix} \cos \varphi & \sin \varphi & 0 \\ -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}; \quad (11.8)$$

- матриця розтягу (стиску) (dilation) відносно початку координат:

$$D = D(a, d) = \begin{pmatrix} a & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{pmatrix}; \quad (11.9)$$

- матриця перенесення (translation) точки на вектор (m, n) :

$$T = T(m, n) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{pmatrix}; \quad (11.10)$$

- матриця дзеркального відображення (reflection) відносно осі x :

$$Ref = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (11.11)$$

Перетворення системи координат задається матрицями:

- $R(-\varphi)$ – поворот системи координат на кут φ у додатному напрямі;
- $T(-m, -n)$ – зсув системи координат на вектор (m, n) .

Зауважимо, що для простоти запису аргументи наведених матриць іноді опускаються.

Елементи довільної матриці афінного перетворення (11.7) не несуть в собі явно вираженого геометричного змісту. Для того, щоб реалізувати відображення (11.7), потрібно знайти елементи матриці, виходячи з геометричного опису перетворення. Як правило, побудову такої матриці в залежності від складності задачі розбивають на кілька етапів, що

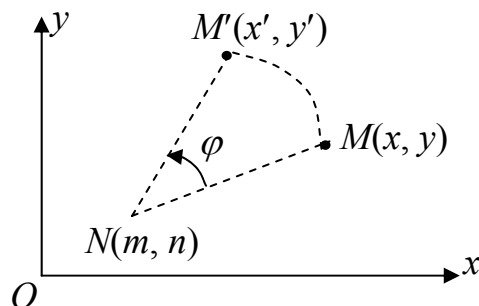


Рис. 11.5. Поворот точки M у додатному напрямку відносно точки N

відповідають основним елементарним перетворенням, які володіють повністю визначеними геометричними властивостями.

Розглянемо приклади складніших афінних перетворень на площині.

11.2. Приклади афінних перетворень на площині

Приклад 1. Побудувати матрицю повороту точки $M(x, y)$ відносно довільної точки $N(m, n)$ на кут φ у додатному напрямку (рис. 11.5).

Матриця R задає поворот точки відносно початку координат. Однорідні координати дають можливість знайти матрицю повороту відносно довільної точки. У загальному випадку поворот відносно довільної точки може бути реалізований шляхом таких перетворень:

- 1) переміщення точки $N(m, n)$ на вектор $(-m, -n)$ так, щоб центр повороту сумістився з початком координат. Матриця цього перетворення

$$T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{pmatrix};$$

- 2) повороту точки на кут φ у додатному напрямку відносно початку координат. Матриця цього перетворення R визначається формулою (11.8);
- 3) переміщення одержаного результату назад так, щоб центр повороту сумістився з точкою N . Матриця цього перетворення T вписана в (11.10).

Отже, для знаходження результуючого повороту точки $M(x, y)$ відносно точки $N(m, n)$ потрібно перемножити матриці T^{-1} , R , T за вказаним порядком.

У результаті одержимо шукане перетворення у вигляді

$$\begin{aligned} (x' \ y' \ 1) &= (x \ y \ 1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{pmatrix} \begin{pmatrix} \cos \varphi & \sin \varphi & 0 \\ -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{pmatrix} = \\ &= (x \ y \ 1) \begin{pmatrix} \cos \varphi & \sin \varphi & 0 \\ -\sin \varphi & \cos \varphi & 0 \\ -m \cos \varphi + n \sin \varphi + m & -m \sin \varphi - n \cos \varphi + n & 1 \end{pmatrix}. \end{aligned}$$

Зокрема, поворот точки на 90° у додатному напрямку відносно точки $N(m, n)$ виконується перетворенням

$$(x' \ y' \ 1) = (x \ y \ 1) \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ n + m & n - m & 1 \end{pmatrix}.$$

Приклад 2. Побудувати матрицю неоднорідного масштабування з коефіцієнтом a вздовж осі x та коефіцієнтом b уздовж осі y з центром у точці $N(m, n)$.

Це перетворення реалізуємо за допомогою трьох перетворень:

- 1) переміщення точки на вектор $N(-m, -n)$. Матрицею цього перетворення є матриця T ;
- 2) розтягу вздовж координатних осей із коефіцієнтами a та b відповідно. Матриця D цього перетворення визначається формулою (11.9);
- 3) перенесення одержаного результату на вектор $N(m, n)$. Матриця T цього перетворення має вигляд (11.10).

Перемножуючи ці матриці, одержуємо шукане перетворення

$$(x' \ y' \ 1) = (x \ y \ 1) \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ (1-a)m & (1-b)n & 1 \end{pmatrix}.$$

Аналогічно, виділяючи послідовність операцій перенесення, повороту та масштабування (ці перетворення описуються матрицями T , R , D), можна побудувати матрицю для будь-якого складного афінного перетворення. Тоді афінне перетворення об'єктів на площині можна здійснити так: якщо об'єкт задається сукупністю точок, то складаємо матрицю X однорідних координат точок цього об'єкта і цю матрицю множимо на матрицю складного афінного перетворення.

Розглянемо приклад симетричного відображення трикутника відносно прямої L .

Приклад 3. Нехай пряма L задається рівнянням $y = x + 1$, а однорідні координати $A(1; 1; 1)$, $B(6; 1; 1)$, $C(6; 4; 1)$ задають координати вершин трикутника ABC . Побудувати трикутник, симетричний даному відносно прямої L .

Представимо перетворення симетрії точки відносно прямої у вигляді послідовності простіших афінних перетворень (рис. 11.6):

- переміщення лінії та трикутника так, щоб лінія L пройшла через початок координат. Це перетворення задає матриця T із параметрами $(0, -1)$;
- поворот лінії та об'єкта відносно початку координат до збігу лінії L з віссю x . Це перетворення задає матриця R із кутом $\varphi = -\frac{\pi}{4}$ (у від'ємному напрямку);
- відображення відносно координатної осі x ;
- обернений поворот відносно початку координат;

- паралельне перенесення у вихідне положення.

Останні три перетворення задаються матрицями Ref , $R(-\varphi)$ та $T(0, 1)$, відповідно. У матричному вигляді перетворення симетрії має вигляд

$$(x' \ y' \ 1) = (x \ y \ 1) \cdot T(0, -1) \cdot R\left(-\frac{\pi}{4}\right) \cdot Ref \cdot R\left(\frac{\pi}{4}\right) \cdot T(0, 1).$$

Для симетрії трикутника ABC відносно прямої L маємо

$$\begin{pmatrix} x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \\ x'_3 & y'_3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 6 & 1 & 1 \\ 6 & 4 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \\ \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 1 \\ 0 & 7 & 1 \\ 3 & 7 & 1 \end{pmatrix}.$$

Отже, декартові координати вершин симетричного трикутника $A'B'C'$ мають значення $A'(0; 2)$, $B'(0; 7)$, $C'(3; 7)$.

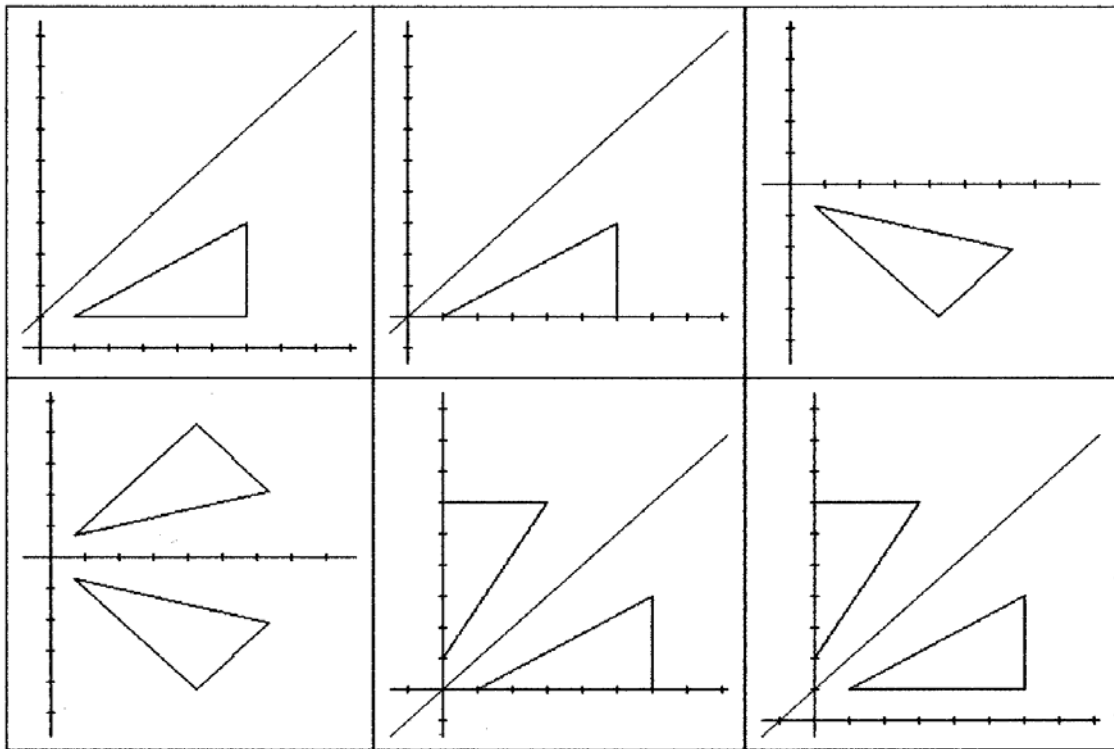


Рис. 11.6. Побудова симетричного трикутника

11.3. Афінні перетворення в просторі

Розглянемо тепер тривимірний випадок – 3D-перетворення простору. Всі перетворення будемо здійснювати в *правосторонній* системі координат $Oxyz$, тобто в системі координат, в якій із кінця осі z поворот від осі x до осі y бачимо проти годинникової стрілки. Якщо з вершини z поворот від x до y видно за годинниковою стрілкою, то така система координат називається *лівосторонньою*. У тривимірному випадку можна ввести однорідні координати так, що декартовим координатам (x, y, z) відповідатиме вектор (hx, hy, hz, h) , $h \neq 0$, координати якого визначаються однозначно з точністю до множника h . Запропонований підхід (введення однорідних координат) дає можливість використовувати

матричний запис всіх афінних перетворень у просторі, тобто у вигляді

$$\begin{pmatrix} x' & y' & z' & 1 \end{pmatrix} = \begin{pmatrix} x & y & z & 1 \end{pmatrix} A,$$

де A – матриця афінного перетворення, для якої $\det A \neq 0$.

Зауважимо, що у випадку $\det A = 0$ перетворення A називається *проективним*.

Розглянемо елементарні перетворення в тривимірному просторі і побудуємо відповідні їм матриці перетворень.

а) На відміну від 2D-випадку, в 3D-просторі визначаються три основні повороти: відносно осі x , відносно осі y , відносно осі z . При повороті точки в просторі відносно осей правосторонньої системи координат додатні напрями поворотів показані на рис. 11.7. Тобто додатними вважаються повороти від x до y , від y до z , від z до x .

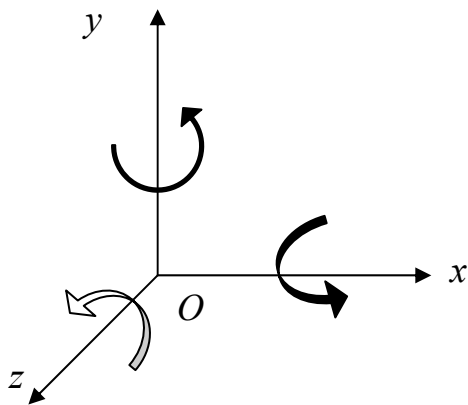


Рис. 11.7. Додатні напрями поворотів

Поворот простору відносно осі z на кут χ проти годинникової стрілки відповідає додатному повороту в площині xu (якщо дивитися з кінця вектора z), z -координата залишається при цьому незмінною. Фактично поворот проходить у площинах, перпендикулярних осі z , тому матриця такого повороту одержується з матриці двовимірного повороту і має вигляд

$$R_z(\chi) = \begin{pmatrix} \cos \chi & \sin \chi & 0 & 0 \\ -\sin \chi & \cos \chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (11.12)$$

Поворот проти годинникової стрілки відносно осі x на кут φ відповідає додатному повороту в площині yz , тобто це обертання аналогічне попередньому з точністю до перейменування осей ($x \rightarrow y, y \rightarrow z, z \rightarrow x$), тому переставляючи рядки і стовпці матриці (11.12) одержуємо матрицю повороту відносно осі x

$$R_x(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi & 0 \\ 0 & -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (11.13)$$

Аналогічно матриця повороту точки відносно осі y на кут ψ проти годинникової стрілки має вигляд

$$R_y(\psi) = \begin{pmatrix} \cos \psi & 0 & -\sin \psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \psi & 0 & \cos \psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (11.14)$$

Для побудови матриці повороту системи координат у додатному напрямку потрібно у формулах (11.12) – (11.14) значення кутів змінити на їх протилежні значення, тобто поворот системи координат здійснюється інвертованими матрицями, а інверсія матриць повороту виконується шляхом транспонування.

Наприклад, поворот системи координат на кут χ відносно осі z у додатному напрямку задається матрицею

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

б) Розтяг (стиск) вздовж осей x, y, z із коефіцієнтами α, β, γ відповідно задається матрицею

$$D = \begin{pmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (11.15)$$

Якщо $\alpha = \beta = \gamma = s$, то маємо однорідне масштабування: при $s > 1$ – однорідне розтягування, при $s < 1$ – однорідний стиск.

в) Матриці дзеркального відображення відносно координатних площин xy , yz , zx мають вигляд

$$M_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_{yz} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_{xz} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (11.16)$$

г) Перенесення (зсув) простору на вектор (l, m, n) описується матрицею

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{pmatrix}, \quad (11.17)$$

тобто однорідні координати зміщеної точки одержуються за допомогою перетворення

$$(x', y', z', 1) = (x, y, z, 1) \cdot T.$$

Зауважимо, що, коли необхідно зробити кілька тривимірних поворотів, то треба мати на увазі некомутативність цих перетворень. Щоб продемонструвати цей факт, розглянемо дві послідовності поворотів на один і той же кут – спочатку відносно осі x , а потім відносно осі y і навпаки. Використовуючи матриці (11.13) та (11.14) при $\psi = \varphi$, одержимо

$$\begin{aligned} R_{xy} = R_x(\varphi) \cdot R_y(\varphi) &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi & 0 \\ 0 & -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \varphi & 0 & -\sin \varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \varphi & 0 & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \\ &= \begin{pmatrix} \cos \varphi & 0 & -\sin \varphi & 0 \\ \sin^2 \varphi & \cos \varphi & \cos \varphi \sin \varphi & 0 \\ \cos \varphi \sin \varphi & -\sin \varphi & \cos^2 \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \end{aligned}$$

З іншого боку, поворот спочатку відносно осі y , а потім відносно осі x з кутом $\psi = \varphi$ дає матрицю

$$R_{yx} = R_y(\varphi) \cdot R_x(\varphi) = \begin{pmatrix} \cos \varphi & 0 & -\sin \varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \varphi & 0 & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi & 0 \\ 0 & -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos \varphi & \sin^2 \varphi & -\cos \varphi \sin \varphi & 0 \\ 0 & \cos \varphi & \sin \varphi & 0 \\ \sin \varphi & -\cos \varphi \sin \varphi & \cos^2 \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Порівнюючи матриці R_{xy} та R_{yx} , бачимо, що вони різні, тобто перетворення повороту в просторі некомутативне. Комутативними є лише перетворення перенесення-перенесення і масштабування-масштабування.

Послідовні перетворення можуть бути об'єднані в одне перетворення, що дає той самий результат, але, оскільки множення матриць є некомутативною операцією, то важливий порядок їх виконання. Аналогічно до $2D$ -випадку, для $3D$ -простору правильною є наступна теорема.

Теорема. Довільне афінне перетворення в $3D$ -просторі можна подати у вигляді послідовності елементарних перетворень $a) - z)$.

Нагадаємо, що загальна форма афінних перетворень у просторі задається системою рівнянь

$$\begin{aligned} x' &= ax + by + ez + m, \\ y' &= cx + dy + fz + n, \\ z' &= gx + hy + pz + l, \end{aligned}$$

де (x', y', z') – точки (вектори), що одержуються в результаті цих перетворень.

Розглянемо приклади складніших перетворень, які подамо у вигляді композиції елементарних перетворень.

11.4. Приклади складніших $3D$ -перетворень

Приклад 4. Побудувати матрицю повороту відносно прямої, паралельної координатній осі.

Матриці R_x , R_y , R_z описують повороти відносно координатних осей, однак часто необхідно обертати об'єкт відносно осей, що не збігаються з координатними.

Розглянемо окремий випадок повороту відносно локальної осі x' , яка паралельна до осі x . Локальна вісь x' проходить через точку $O'(0, m, n)$. Обертання тіла відносно локальних осей виконується за допомогою такої послідовності перетворень:

- зсув об'єкта так, щоб локальна вісь збігалася з координатною;
- поворот об'єкта відносно координатної осі x ;
- переміщення перетвореного об'єкта у вихідне положення.

Ці перетворення можна записати у вигляді добутку матриць

$$X' = X \cdot T \cdot R_x \cdot T^{-1},$$

де X' – матриця координат перетвореного об'єкта, X – матриця координат вихідного об'єкта, T – матриця переміщення на вектор $(0, -m, -n)$, R_x – матриця повороту відносно осі x , T^{-1} – обернена матриця до матриці переміщення.

Використовуючи формули (11.13), (11.17), одержуємо

$$T \cdot R_x \cdot T^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -m & -n & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi & 0 \\ 0 & -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times$$

$$\times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & m & n & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi & 0 \\ 0 & -\sin \varphi & \cos \varphi & 0 \\ 0 & m(1 - \cos \varphi) + n \sin \varphi & n(1 - \cos \varphi) - m \sin \varphi & 1 \end{pmatrix}.$$

Приклад 5. Побудувати матрицю обертання на кут θ відносно прямої L , що проходить через точку $A(x_0, y_0, z_0)$ з напрямним вектором (a, b, c) (рис. 11.8).

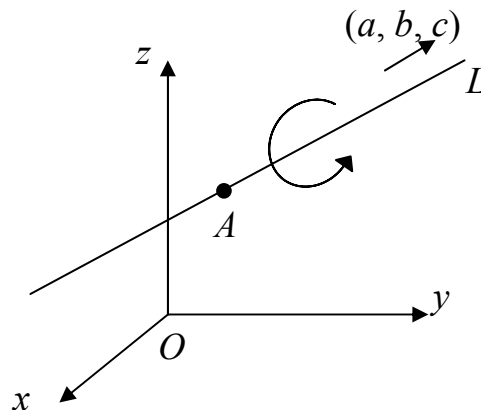


Рис. 11.8. Обертання відносно довільної прямої

Нехай (a, b, c) – одиничний вектор. Узагальнений випадок обертання відносно довільної осі в просторі зустрічається часто, наприклад у робототехніці. Основна ідея розв'язування цієї задачі полягає в суміщенні прямої L з однією з координатних осей. Для цього потрібно виконати наступні кроки.

- 1) Перенесення простору на вектор $(-x_0, -y_0, -z_0)$ за допомогою матриці

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{pmatrix},$$

в результаті якого пряма L пройде через початок координат.

- 2) Виконання відповідних поворотів, щоб вісь обертання збіглася, наприклад, із віссю аплікат.

У загальному випадку, щоб довільна пряма, що проходить через початок координат, збіглася з однією з координатних осей, необхідно зробити два послідовних повороти відносно двох інших координатних осей. Для суміщення довільної осі обертання з віссю z спочатку виконуємо поворот відносно осі x , а потім – відносно y .

Щоб визначити кут повороту φ відносно осі x , спершу спроектуємо на площину Oyz напрямний вектор (a, b, c) (рис. 11.9, а). Спроектований вектор має координати $(0, b, c)$. Довжина його проекції $d = \sqrt{b^2 + c^2}$.

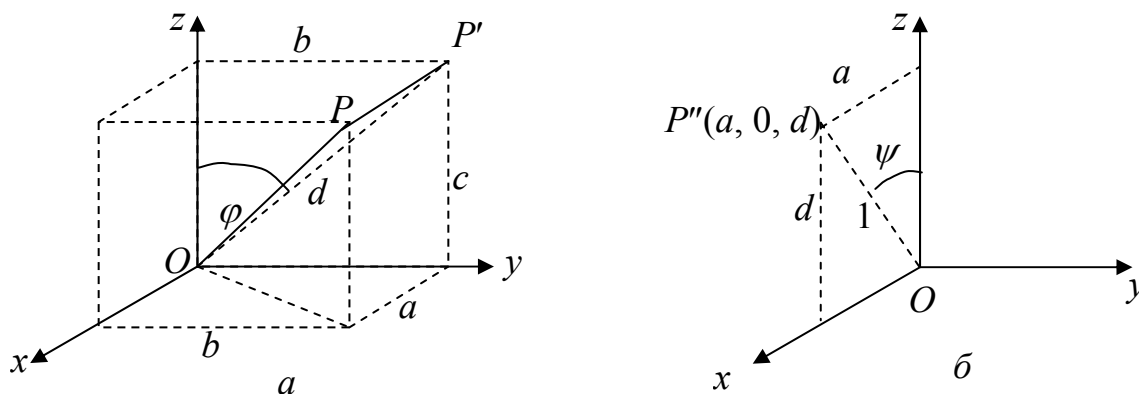


Рис. 11.9. Суміщення вектора OP з віссю z

Із рис. 11.9, а видно, що $\cos \varphi = \frac{c}{d}$, $\sin \varphi = \frac{b}{d}$. Тоді матриця повороту відносно осі x має вигляд

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{b}{d} & 0 \\ 0 & -\frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Якщо здійснити такий поворот відносно осі x на кут φ , то пряма L суміститься з площиною Oxz і займе положення OP'' (рис. 11.9, б). Під

дією цього перетворення координати вектора $(a, b, c, 1)$ зміняться на $(a', b', c', 1)$, де

$$(a' \ b' \ c' \ 1) = (a \ b \ c \ 1) R_x = (a \ 0 \ d \ 1).$$

Далі, щоб сумістити напрям OP'' із віссю z , необхідно виконати поворот відносно осі y в напрямку від x до z , тобто за годинниковою стрілкою (у від'ємному напрямку) на кут ψ . З рис. 11.9, б маємо

$$\sin \psi = a, \quad \cos \psi = d.$$

Відповідно матриця такого повороту має вигляд

$$R_y^- = \begin{pmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

- 3) Виконання повороту відносно прямої L на кут θ . Оскільки пряма L лежить на осі z , то відповідна матриця повороту в додатному напрямку має вигляд

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

- 4) Виконання перетворення, оберненого до повороту відносно осі y на кут ψ . Це перетворення описується матрицею

$$R_y = \begin{pmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

- 5) Виконання перетворення, оберненого до обертання відносно осі x на кут φ . Це перетворення задається матрицею

$$R_x^- = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & -\frac{b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

- 6) Виконання оберненого перенесення, тобто зсуву на вектор (x_0, y_0, z_0) . Це перетворення задає матриця

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{pmatrix}.$$

Перемноживши знайдені матриці в порядку їх побудови, одержимо матрицю повороту відносно довільної прямої L у вигляді

$$M = T^- \cdot R_x^- \cdot R_y^- \cdot R_z \cdot R_y \cdot R_x^- \cdot T.$$

Загальний вигляд матриці M не виписуємо через її громіздкий запис. На практиці при написанні програм для знаходження матриці M користуються саме останньою формулою.

У простішому випадку, коли $x_0 = y_0 = z_0 = 0$, маємо

$$M = \begin{pmatrix} a^2 + (1 - a^2)\cos\theta & a(1 - \cos\theta)b + c\sin\theta & a(1 - \cos\theta)c - b\sin\theta & 0 \\ a(1 - \cos\theta)b - c\sin\theta & b^2 + (1 - b^2)\cos\theta & b(1 - \cos\theta)c + a\sin\theta & 0 \\ a(1 - \cos\theta)c + b\sin\theta & b(1 - \cos\theta)c - a\sin\theta & c^2 + \cos\theta(1 - c^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Розглядаючи довільні перетворення в просторі, в результаті їх комбінування одержуємо матриці вигляду

$$A = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 & 0 \\ \beta_1 & \beta_2 & \beta_3 & 0 \\ \gamma_1 & \gamma_2 & \gamma_3 & 0 \\ l & m & n & 1 \end{pmatrix}.$$

Матрицю A розміром 4×4 можна розбити на 4 окремих частини (блоки). Верхній лівий блок розміром 3×3 задає сумарні лінійні перетворення: масштабування, поворот, відображення. Лівий нижній задає переміщення, правий верхній – перспективні перетворення (див. п. 12.5). Елемент $a_{44} = 1$ – загальний коефіцієнт масштабування.

За допомогою матриці A можна перетворювати довільні просторові фігури, наприклад здійснювати перетворення опуклого многогранника. Опуклий многогранник однозначно визначається набором своїх вершин $X_i(x_i, y_i, z_i)$, що утворюють матрицю X . Далі на матрицю X діємо невідродженою матрицею A і одержуємо набір вершин $X' = X \cdot A$ нового многогранника, який є образом вихідного многогранника.

11.5. Методи задання складних афінних перетворень

Складним (комбінованим) називається перетворення, яке містить ланцюжок базових перетворень (не менше двох). Зауважимо, що майже

всі афінні перетворення залежать від порядку їх виконання. Наприклад, перетворення обертання некомутативні.

Розглянемо два основні методи знаходження матриць складних перетворень:

- *метод розкладу (декомпозиції)* складного руху об'єкта на n елементарних рухів, що описуються матрицями $C_i, i = 1, 2, \dots, n$. Одержаний ланцюжок із n перетворень еквівалентний одному перетворенню з матрицею $C = \prod_{i=1}^n C_i$. В однорідних координатах матриця перетворення C знаходиться за допомогою тільки операцій множення матриць. Зазначимо, що розклад матриці C на добуток матриць C_i можна здійснити не єдиним способом, тому задача програміста полягає в оптимізації множини елементарних перетворень, що задаються матрицями C_i ;
- *метод парних точок*, що визначає матрицю афінних перетворень за трьома (на площині) та чотирма (в просторі) парами лінійно незалежних точок образу X та прообразу X' . За умови, що трійки точок неколінеарні, а четвірки некомпланарні, матриці, які задають ці об'єкти, невироджені, тому завжди можна знайти матриці афінних перетворень.

Найкраща обумовленість матриць досягається при ортогональності систем векторів, що побудовані на вибраних точках образу і прообразу.

Отже, із системи рівнянь $X' = X \cdot C$ знаходимо $C = X^{-1} \cdot X'$.

Приклад 6. Знайти матрицю перетворення трикутника $P_1P_2P_3$ у трикутник $P'_1P'_2P'_3$ (рис. 11.10).

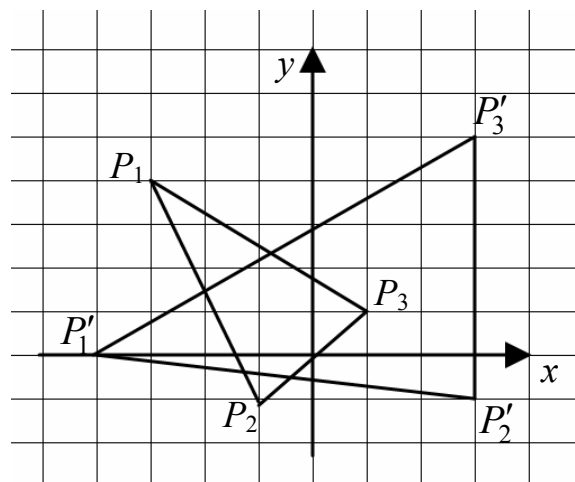


Рис. 11.10. Перетворення трикутників

Однорідні координати вершин трикутника $P_1P_2P_3$ складають матрицю

$$X = \begin{pmatrix} -3 & 4 & 1 \\ -1 & -1 & 1 \\ 1 & 1 & 1 \end{pmatrix},$$

а трикутника $P'_1P'_2P'_3$ – матрицю

$$X' = \begin{pmatrix} -4 & 0 & 1 \\ 3 & -1 & 1 \\ 3 & 5 & 1 \end{pmatrix}.$$

Зі співвідношення $C = X^{-1} \cdot X'$ одержуємо

$$C = \begin{pmatrix} 1 & 2 & 0 \\ -1 & 1 & 0 \\ 3 & 2 & 1 \end{pmatrix}.$$

Тепер можна спробувати підібрати послідовність і параметри елементарних перетворень трикутника $P_1P_2P_3$ у трикутник $P'_1P'_2P'_3$ і переконатися, що цей спосіб побудови матриці C у даному випадку складніший.

Приклад 7. Розглянути задачу про зміну просторової орієнтації системи координат, як це зображено на рис. 11.11 і на цьому прикладі порівняти алгоритм розкладу та метод парних точок.

Застосуємо спочатку метод розкладу для обчислення матриці складного афінного перетворення на прикладі зміни орієнтації системи координат так, щоб початок координат буде суміщений із точкою $O'(m, n, l, 1)$, а напрям деякої осі, наприклад z , – з напрямком заданого вектора $V(a, b, c)$ (рис. 11.11) (розміщення двох інших осей не регламентується).

Розв'язок задачі досягається за допомогою двох перетворень:

1. Перенесення початку системи координат xuz у точку O' . Це перетворення описується матрицею $T(-m, -n, -l) = T^-$.
2. Поворот системи координат так, щоб вісь z мала напрям вектора V , що виходить уже з початку перенесеної системи координат. Позначимо матрицю цього перетворення $A(V)$, тоді матриця повного перетворення $C = T^- \cdot A(V)$.

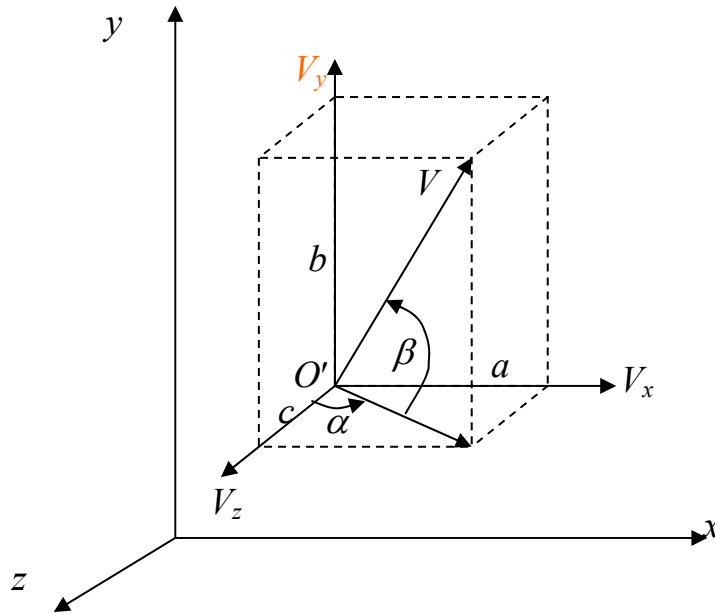


Рис. 11.11. Зміна просторової орієнтації системи координат

Виразимо матрицю $A(V)$ через матриці елементарних поворотів.

- Перший поворот системи координат виконуємо відносно осі y проти годинникової стрілки на кут α , для якого

$$\cos \alpha = \frac{c}{d}, \sin \alpha = \frac{a}{d}, d = \sqrt{a^2 + c^2} \neq 0, \alpha \in [-180^0, 180^0].$$

Це перетворення виконує матриця повороту $R_y(-\alpha)$.

- Друге перетворення – це поворот відносно нової осі x' на кут β у від'ємному напрямку, для якого

$$\cos \beta = \frac{d}{|V|}, \sin \beta = \frac{b}{|V|}, \beta \in [-90^0, 90^0].$$

Це перетворення задається матрицею $R_x(\beta)$.

Отже, матриця повороту системи координат для суміщення осі z із вектором V обчислюється за формулою

$$A(V) = R_y(-\alpha) \cdot R_x(\beta).$$

Якщо $d = 0$ (тобто $V \parallel Oy$), то кут α невизначений і суміщення осі z із вектором V досягається не за два, а за одне обертання системи координат відносно осі x матрицею

$$A(V) = R_x(\text{sign}(V_y)90^0).$$

Зауваження. Пряма підстановка

$$\cos \alpha = \frac{c}{d}, \sin \alpha = \frac{a}{d}, \cos \beta = \frac{d}{|V|}, \sin \beta = \frac{b}{|V|}$$

у матриці поворотів $R_y(-\alpha)$, $R_x(\beta)$ та їх множення приводить до громіздкого запису матриці $A(V)$, тому краще множення матриць виконувати програмно.

Застосуємо тепер метод парних точок. Для знаходження матриці $A(V)$ методом парних точок знайдемо в просторі три неколінеарні точки, в які перейдуть кінці ортів x^0, y^0, z^0 після повороту системи координат. Одна з цих точок, кінець вектора V , є прообразом z^0 . У випадку $V \neq z^0$ виберемо прообрази y^0 і x^0 на кінцях векторів нормалі $N = z^0 \times V$ (\times – означає векторний добуток) і бінормалі $W = N \times V$. Довжини векторів однакові, тому відсутні спотворення кутів при відображенні, а отже,

$$\begin{pmatrix} x^0 \\ y^0 \\ z^0 \end{pmatrix} C = \begin{pmatrix} W \\ N \\ V \end{pmatrix},$$

де C – матриця перетворення об'єкта. З останнього співвідношення маємо

$$C = \begin{pmatrix} x^0 \\ y^0 \\ z^0 \end{pmatrix}^{-1} \begin{pmatrix} W \\ N \\ V \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} W \\ N \\ V \end{pmatrix} = \begin{pmatrix} W \\ N \\ V \end{pmatrix}.$$

Якщо об'єкт залишити нерухомим, а перетворювати тільки систему координат, то таке пасивне перетворення задається оберненою матрицею, тобто

$$A(V) = C^{-1} = \begin{pmatrix} N \times V \\ N \\ V \end{pmatrix}^{-1}.$$

Отже, метод парних точок дозволяє знайти матрицю повного перетворення системи координат без розкладу на проміжні перетворення.

Контрольні питання та завдання

1. Які перетворення називаються афінними? Назвіть їх властивості.
2. Що таке однорідні координати? Чому вони застосовуються в КГ?
3. Як задаються афінні перетворення в однорідних координатах?
4. Як побудувати матрицю повороту відносно довільної точки на площині?
5. Випишіть матриці елементарних перетворень в просторі.
6. Які перетворення в просторі є комутативними/некомутативними?
7. Як побудувати матрицю повороту відносно довільної осі?

8. Які повороти називаються додатними?
9. Назвіть та порівняйте методи знаходження матриць складних афінних перетворень.
10. Сформулюйте ідею методу парних точок.

Вправи і задачі для самостійного виконання

1. Довести, що перетворення зсуву адитивне, а саме послідовне виконання двох зсувів точки еквівалентне одному зсуву, тобто $T(m_1, n_1) T(m_2, n_2) = T(m_1 + m_2, n_1 + n_2)$.
2. Довести, що два послідовних повороти адитивні, тобто $R(\varphi_1 + \varphi_2) = R(\varphi_1) \cdot R(\varphi_2)$.
3. Доведіть, що однорідне масштабування і поворот утворюють комутативну групу.
4. Довести, що сумарне масштабування буде мультиплікативним, тобто якщо виконати масштабування $D(a_1, b_1)$, а потім $D(a_2, b_2)$, то результуюче масштабування матиме вигляд $D = D(a_1 a_2, b_1 b_2)$.
5. Побудувати матрицю такої послідовності рухів точки:
 - а) зсув по осі x на Δx ;
 - б) поворот відносно попереднього положення точки на кут α ;
 - с) зсув по осі y на Δy .
6. Довести, що відображення відносно прямої в площині xy , що проходить через початок координат еквівалентно повороту відносно початку координат. Розглянути випадок прямої $y = x$.
7. Побудувати матрицю загального повороту в 3D-просторі відносно трьох координатних осей, тобто матрицю $R_{xyz} = R_z(\alpha) \cdot R_y(\beta) \cdot R_x(\gamma)$. Знайти обернену матрицю цього перетворення.
8. Задано три точки P_1, P_2, P_3 в просторі. Побудувати матрицю перетворення, в результаті якого вектор P_1P_2 буде направлений вздовж осі z , а вектор P_1P_3 буде лежати в площині $x = 0$.
9. Задана трикутна призма $ABCDEF$ з вершинами $A(5; 5; 0)$, $B(5; 15; 0)$, $C(15; 5; 0)$, $D(5; 5; 10)$, $E(5; 15; 10)$, $F(15; 5; 10)$. Обчислити координати повернутої призми відносно ребра CB на кут 45° .

Тема 12. Моделювання проєкцій

12. 1. Класифікація проєкцій

Для побудови графічних зображень тривимірних об'єктів на площині, екрані дисплея чи папері використовують проєкції (від лат. projection – викидати вперед). Проєкції відображають тривимірні об'єкти на двовимірну картинну площину (надалі КП), тобто перетворюють точки 3D-простору у 2D-простір. Проєкції будуються за допомогою прямих (проєкторів), що виходять з центра проєкування. Ці прямі проходять через кожен точку об'єкта, перетинаючи КП, і на ній утворюють слід – проєкцію об'єкта. Оскільки, проєкцією відрізка є відрізок, то для побудови проєкції відрізка досить побудувати лише проєкції його кінців. Ми розглядатимемо лише проєкції на площину (плоскі геометричні проєкції), хоча існують проєкції на скривлені поверхні – так звані картографічні проєкції.

У комп'ютерній графіці найбільш широко застосовують два класи проєкцій: паралельна та центральна (перспективна) проєкції.

У випадку *центрального проєкування* всі прямі виходять з однієї точки – центру проєкування, що знаходиться в скінченній точці тривимірного простору (рис. 12.1, б). Центральну проєкцію називають ще і перспективною проєкцією, або перспективою (від лат. perspicio – ясно бачу). Якщо центр проєкування знаходиться на нескінченності, то всі проєктори паралельні між собою і в результаті проєкування одержується *паралельна проєкція* (рис. 12.1, а).

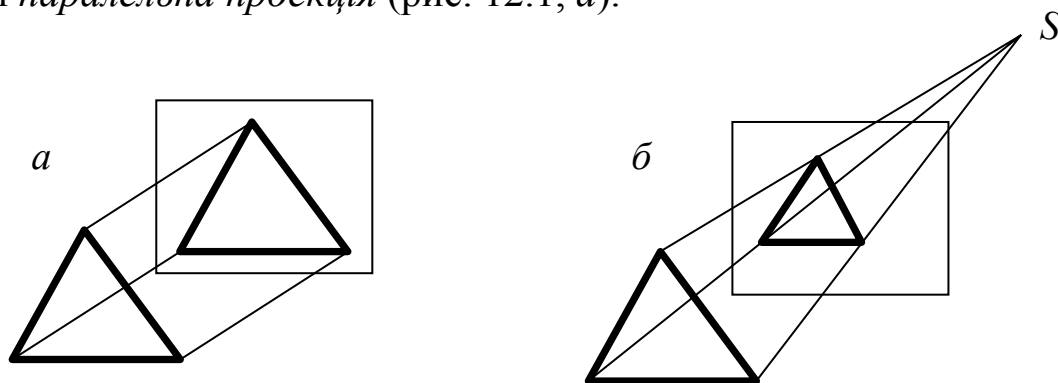


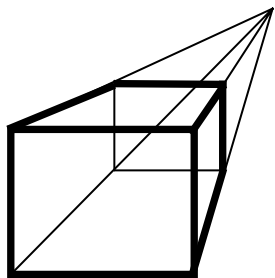
Рис. 12.1. Паралельна і центральна проєкції

До важливих властивостей методу проєкування належить ступінь достовірності сприйняття об'єкта спостерігачем за його проєкціями, тобто розпізнавання об'єкта за його плоским зображенням. Призначення того чи іншого методу проєкування полягає в зменшенні невизначеності найбільш важливих властивостей об'єкта (будь-яка проєкція завжди містить у собі похибку, оскільки проєкування не є афінним перетворенням, тому воно взагалі не є оберненим). За час розвитку цивілізації знайшли

багато різних способів зображення просторових об'єктів на площині.

Зі сказаного вище випливає, що найкращої проєкції не існує. В технічних кресленнях для точної передачі розмірів деталей в основному використовуються паралельні проєкції. Художниками і архітекторами часто використовуються перспективні види для створення найбільш реалістичного зображення тривимірних об'єктів.

Центральна проєкція породжує візуальний ефект, аналогічний до того, який виникає при спостереженні людиною чи фотографуванні, тобто досягається деякий ступінь реалістичності завдяки ефекту перспективного вкорочування.



При центральному проєктуванні розмір об'єкта зменшується зі збільшенням відстані до центра проєкції і відбувається пропорційне спотворення ліній об'єкта, яке залежить від орієнтації лінії та її віддалі від центру проєктування. Все це сприяє кращому сприйняттю глибини сцени, однак не зберігає форму об'єкта.

Рис. 12.2. Точка збігу

форми об'єкта.

Центральні проєкції довільної сукупності паралельних прямих, не паралельних до проєкційної площини, перетинаються в точці збігу (рис. 12.2). Завдяки цій властивості плоскі зображення (проєкції) виглядають об'ємно і реалістично.

Кожен із цих двох класів проєктування розбивається на підкласи в залежності від взаємного розміщення КП і координатних осей. Центральні проєкції умовно можна класифікувати за кількістю точок збігу зображення куба чи паралелепіпеда. Наприклад, проєкції каркасних кубів мають одну, дві або три точки збігу і, відповідно, називаються *одноточковими*, *двоточковими* або *треточковими* перспективами.

Види проєктування наведені на рис. 12.3. Побудуємо математичні моделі базових проєктивних перетворень. Спочатку розглянемо паралельні проєкції.

12.2. Ортографічна проєкція

Основним типом координатних систем у проєктивній геометрії є однорідні координати. Використання однорідних координат для задання операцій проєктування дозволяє значно полегшити розв'язання задач геометричного моделювання проєкцій. А саме операції проєктування в однорідних координатах можна подати в матричному вигляді, тобто точка-проєкція X' одержується з точки X за допомогою перетворення $X' = X \cdot P$, де P – матриця проєктування розміром 4×4 .

При ортографічному проектуванні КП збігається з однією із координатних площин або паралельна їй. Якщо проектори паралельні осі z , то точка (x, y, z) проектується в точку $(x, y, 0)$. Матриця ортогонального проектування вздовж осі z має вигляд

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



Рис. 12.3. Класифікація проекцій

Аналогічно виписуються матриці ортогонального проектування вздовж двох інших координатних осей

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

Зауваження 1. Для адекватного відтворення форми об'єкта недостатньо однієї ортогональної проекції, оскільки матриці ортогонального проектування вироджені. Необхідно мати кілька ортогональних проекцій, а саме проекції на площини $z = 0$, $x = 0$, $y = 0$. Вони є видами об'єкта спереду (фронтальний), справа (поздовжній) і зверху (горизонтальний). В технічних кресленнях за основні площини проекцій приймають шість граней куба. Важливо зазначити, що всі шість проекцій можуть бути

одержані шляхом комбінування перетворень відображення, повороту, перенесення з наступним ортогональним проектуванням на площину $z = 0$.

12.3. Аксонометрична проекція

Одна ортографічна проекція не може дати уяви про загальну форму тривимірного об'єкта. Це обмеження можна усунути за допомогою аксонометричних проекцій. Аксонометрична проекція (від грецького *axo* – вісь, *metreo* – вимірювати) утворюється маніпулюванням об'єкта за допомогою поворотів так, щоб були видимими принаймні три сусідні грані.

При аксонометричній проекції проектори теж перпендикулярні КП, але вже не паралельні координатних осей. Відповідно до взаємного розміщення КП і координатних осей розрізняють три види аксонометричних проекцій:

- триметрію – нормальний вектор КП утворює з осями координат попарно різні кути (рис. 12.4, а);
- диметрію – два кути між нормаллю КП і осями координат однакові (рис. 12.4, б);
- ізометрію – всі три кути між нормаллю КП і координатними осями однакові (рис. 12.4, в).

У цих проекціях можна розглядати також кути між осями координат та самою КП (рис. 12.4).

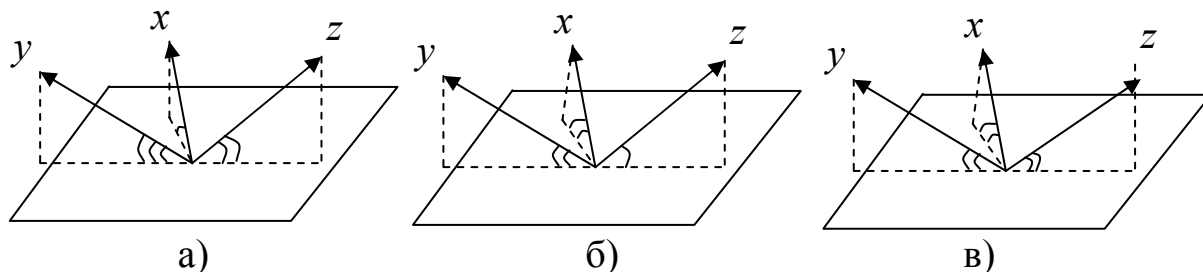


Рис. 12.4. Три види аксонометричної проекції: а – триметрія; б – диметрія; в – ізометрія

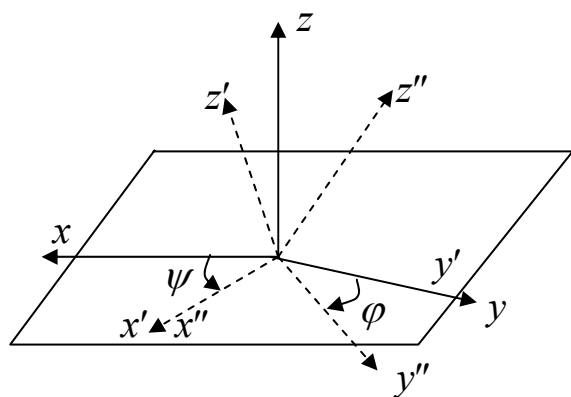


Рис. 12.5. Повороти системи координат

Здійснивши комбінацію відповідних поворотів, можна аксонометричне проектування звести до ортографічного. Для цього повернемо систему координат xyz на кут ψ відносно осі y так, щоб вісь x попала в КП. Цей поворот здійснюється в напрямку від z до x (рис. 12.5), тобто в додатному напрямку, і описується матрицею $R_y^-(\psi)$. Далі одержану систему координат

$x'y'z'$ повернемо навколо осі x' на кут φ у напрямку від z' до y' (від'ємний напрям) так, щоб вісь y' попала в КП (рис. 12.5). Це перетворення здійснює матриця $R_x(\varphi)$. Після цього виконуємо ортогональне проектування вздовж осі z'' . В результаті одержуємо *матрицю аксонометричного проектування*

$$A = R_y^-(\psi) \cdot R_x(\varphi) \cdot P_z = \begin{pmatrix} \cos \psi & 0 & \sin \psi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \psi & 0 & \cos \psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times$$

$$\times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi & 0 \\ 0 & -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos \psi & -\sin \psi \sin \varphi & 0 & 0 \\ 0 & \cos \varphi & 0 & 0 \\ -\sin \psi & -\cos \psi \sin \varphi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Триметрична проекція здійснюється довільними поворотами, що виконуються в будь-якому порядку навколо координатних осей із наступним ортографічним проектуванням на КП, тобто φ та ψ у матриці A для триметрії є довільними.

На рис. 12.6 зображено триметричні проекції зрізаного куба для різних значень кутів φ та ψ . Куб зі зрізаною вершиною задається матрицею X , де

$$X = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0,5 & 1 & 1 \\ 0,5 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0,5 & 1 \end{pmatrix}.$$

Координати триметричної проекції зрізаного куба обчислюємо за формулою $X' = X \cdot A$, де A – матриця аксонометричного проектування.

Для диметричної проекції значення кутів ψ та φ потрібно відповідним чином підібрати. Для цього розглянемо матрицю X з одиничних ортів координатних осей

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

Подіємо на X матрицею аксонометричного проектування A . Маємо

$$X^* = X \cdot A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \cos \psi & -\sin \psi \sin \varphi & 0 & 0 \\ 0 & \cos \varphi & 0 & 0 \\ -\sin \psi & -\cos \psi \sin \varphi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos \psi & -\sin \psi \sin \varphi & 0 & 1 \\ 0 & \cos \varphi & 0 & 1 \\ -\sin \psi & -\cos \psi \sin \varphi & 0 & 1 \end{pmatrix}.$$

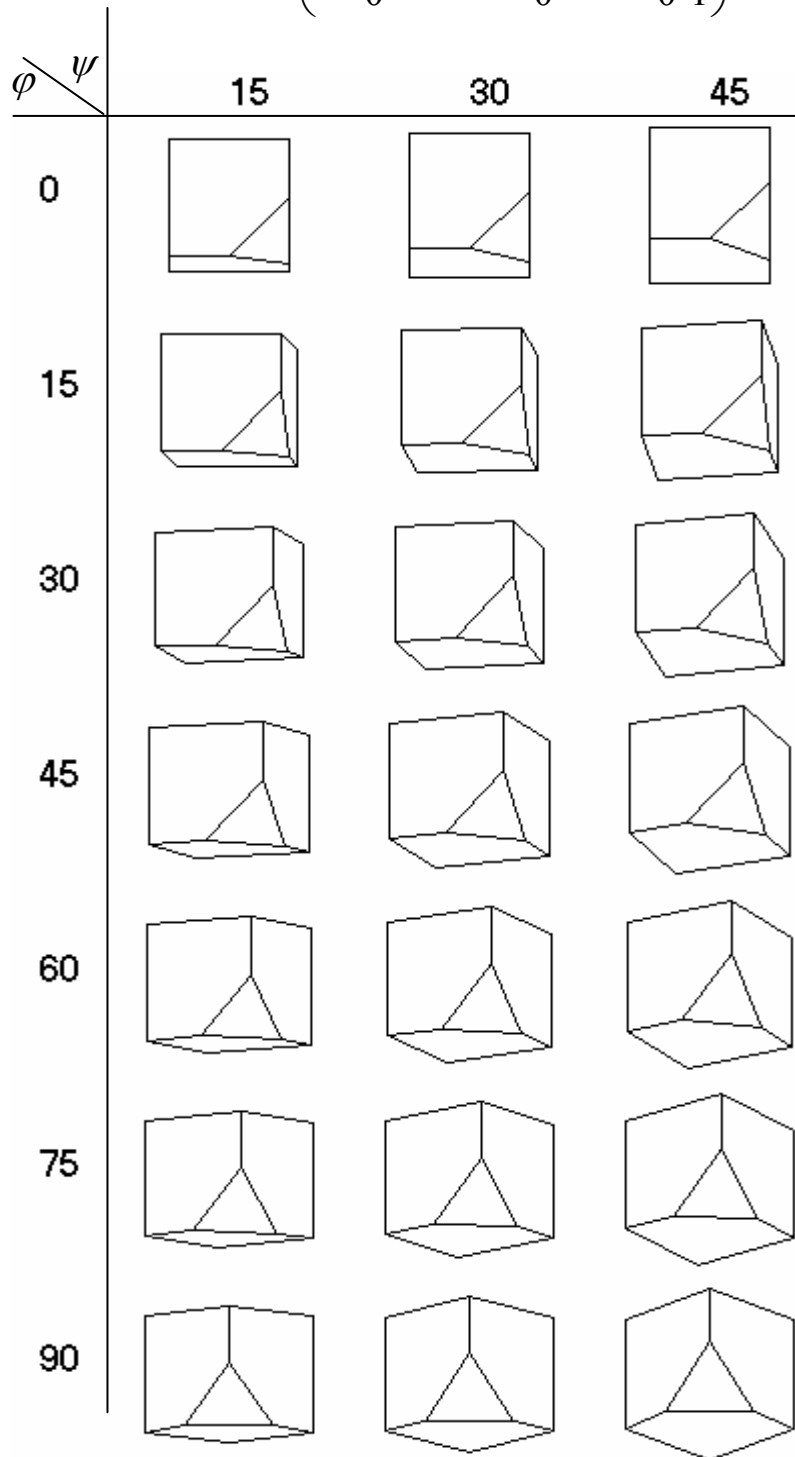


Рис. 12.6. Триметричні проєкції зрізаного куба

Квадрат довжини проекції одиничного орта вздовж осі x (коефіцієнт спотворення) дорівнює

$$\begin{aligned} k_x^2 &= \cos^2 \psi + \sin^2 \psi \sin^2 \varphi = 1 - \sin^2 \psi + \sin^2 \psi \sin^2 \varphi = \\ &= 1 - \sin^2 \psi \cos^2 \varphi \leq 1. \end{aligned} \quad (12.1)$$

Аналогічно квадрати довжин проекцій одиничних ортів вздовж осей y та z дорівнюють, відповідно,

$$k_y^2 = \cos^2 \varphi \leq 1, \quad (12.2)$$

$$k_z^2 = \sin^2 \varphi + \cos^2 \varphi \sin^2 \psi = 1 - \cos^2 \varphi \cos^2 \psi \leq 1. \quad (12.3)$$

Масштабні коефіцієнти k_x, k_y, k_z є довжинами проекцій одиничних ортів на КП. Із формул (12.1), (12.2), (12.3) випливає умова взаємозв'язку

$$k_x^2 + k_y^2 + k_z^2 = 1 - \sin^2 \psi \cos^2 \varphi + \cos^2 \varphi + 1 - \cos^2 \varphi \cos^2 \psi = 2 \quad (12.4)$$

між коефіцієнтами k_x, k_y, k_z .

Часто для аксонометричної проекції задають не кути, а масштаби k_x, k_y, k_z , тоді з формул (12.2), (12.3) можна знайти параметри φ та ψ , тобто

$$\cos^2 \varphi = k_y^2, \quad \cos^2 \psi = \frac{1 - k_z^2}{k_y^2}.$$

Інколи необхідно забезпечити бажані співвідношення масштабів

$$k_x : k_y : k_z = m_x : m_y : m_z.$$

Тоді коефіцієнти k_x, k_y, k_z повинні бути пропорційні значенням m_x, m_y, m_z , тобто $k_x = cm_x, k_y = cm_y, k_z = cm_z$ і $c^2(m_x^2 + m_y^2 + m_z^2) = k_x^2 + k_y^2 + k_z^2 = 2$.

Звідси випливає, що

$$c = \sqrt{\frac{2}{m_x^2 + m_y^2 + m_z^2}}, \quad k_i = cm_i, \quad i = 1, 2, 3. \quad (12.5)$$

Диметрія характеризується тим, що довжини двох проекцій одиничних ортів збігаються. Прирівнюючи коефіцієнти масштабування k_x і k_y , одержуємо

$$\cos^2 \psi + \sin^2 \psi \sin^2 \varphi = \cos^2 \varphi,$$

а, враховуючи формули $\cos^2 \psi = 1 - \sin^2 \psi$, $\cos^2 \varphi = 1 - \sin^2 \varphi$, маємо

$$\sin^2 \psi (1 - \sin^2 \varphi) = \sin^2 \varphi, \quad (12.6)$$

або

$$\sin^2 \psi = \operatorname{tg}^2 \varphi. \quad (12.7)$$

Фіксуючи один із кутів, наприклад ψ , з (12.7) знаходимо

$$\varphi = \operatorname{arctg}(\pm \sin \psi).$$

Тоді, наприклад, якщо вибрати $\psi = 60^\circ$, з (12.7) одержуємо $\operatorname{tg}^2 \varphi = \frac{3}{4}$ і для

кута φ можна вибрати значення $\varphi = \operatorname{arctg}\left(\frac{\sqrt{3}}{2}\right) \approx 40,9^0$.

Окремим випадком диметрії є стандартна диметрична проекція, при якій співвідношення масштабів складає $k_x : k_y : k_z = 2 : 2 : 1$. За формулами (12.5) знаходимо $k_x = \frac{\sqrt{2}}{3} \cdot 2$, $k_y = \frac{\sqrt{2}}{3} \cdot 2$, $k_z = \frac{\sqrt{2}}{3}$. Тоді

$$\cos^2 \varphi = k_y^2 = \frac{8}{9}, \quad \cos^2 \psi = \frac{1 - k_z^2}{k_y^2} = \frac{7}{8},$$

а отже, $\varphi = \arccos \sqrt{\frac{8}{9}} \approx 19,47^0$, $\psi = \arccos \sqrt{\frac{7}{8}} \approx 20,7^0$.

Матриця стандартного диметричного проектування має вигляд

$$A_{\text{дмм}}^{cm} = \begin{pmatrix} \sqrt{\frac{7}{8}} & -\sqrt{\frac{1}{72}} & 0 & 0 \\ 0 & \sqrt{\frac{8}{9}} & 0 & 0 \\ -\sqrt{\frac{1}{8}} & -\sqrt{\frac{7}{72}} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \approx \begin{pmatrix} 0,935 & -0,118 & 0 & 0 \\ 0 & 0,943 & 0 & 0 \\ -0,354 & -0,312 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Умова $k_x = k_y \neq k_z$ для диметричної проекції дозволяє знаходити розміри спроектованого об'єкта з однаковим масштабом по осях x та y , а виміри вздовж осі z вимагають іншого масштабного коефіцієнта. Це може призвести до помилок у випадку, коли потрібне точне масштабування розмірів спроектованого об'єкта.

Цей недолік усуває ізометрична проекція. У випадку ізометрії маємо $k_x^2 = k_y^2 = k_z^2$. Раніше з умови $k_x^2 = k_y^2$ випливало рівняння (12.6) або

$$\sin^2 \psi = \frac{\sin^2 \varphi}{1 - \sin^2 \varphi}, \quad (12.8)$$

аналогічно з умови $k_y^2 = k_z^2$ одержуємо

$$\sin^2 \psi = \frac{1 - 2 \sin^2 \varphi}{1 - \sin^2 \varphi}. \quad (12.9)$$

Прирівнюючи праві частини (12.8) і (12.9), знаходимо $\sin^2 \varphi = \frac{1}{3}$,

або $\varphi = \arcsin\left(\pm \frac{1}{\sqrt{3}}\right) \approx \pm 35,26^0$. Тоді з (12.8) $\sin^2 \psi = \frac{1}{3} : \left(1 - \frac{1}{3}\right) = \frac{1}{2}$, а

отже, $\psi = \pm 45^0$.

Коефіцієнт спотворення для ізометричної проекції $k_y = \sqrt{\cos^2 \varphi} = \sqrt{2/3} \approx 0,8165$ і є однаковим уздовж усіх осей ($k_x = k_y = k_z$). Тому можна проводити виміри вздовж осей координат з одним і тим же масштабом.

Отже, перетворення ізометричного проектування, що одержується при $\psi = 45^\circ$ та $\varphi = 35,26^\circ$, має вигляд

$$A_{izo} = \begin{pmatrix} \sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{6}} & 0 & 0 \\ 0 & \sqrt{\frac{2}{3}} & 0 & 0 \\ -\sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{6}} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0,707 & -0,408 & 0 & 0 \\ 0 & 0,816 & 0 & 0 \\ -0,707 & -0,408 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Якщо взяти куб із відрізаною вершиною, який задається матрицею X , яка була наведена вище, і подіяти на неї матрицею ізометричного проектування A_{izo} , то одержимо координати ізометричної проекції куба, тобто

$$X' = X \cdot A_{izo} = \begin{pmatrix} -0,707 & -0,408 & 0 & 1 \\ 0 & -0,816 & 0 & 1 \\ 0 & -0,408 & 0 & 1 \\ -0,354 & 0,204 & 0 & 1 \\ -0,707 & 0,408 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0,707 & -0,408 & 0 & 1 \\ 0,707 & 0,408 & 0 & 1 \\ 0 & 0,816 & 0 & 1 \\ 0,354 & 0,204 & 0 & 1 \end{pmatrix}.$$

На рис. 12.8 зображено чотири можливі ізометричні проекції зрізаного куба з кутами $\psi = \pm 45^\circ$, $\varphi = \pm 35,26^\circ$.

12.4. Косокутна проекція

На відміну від ортографічної і аксонометричної проекцій, для яких проектори перпендикулярні до КП, косокутна проекція формується паралельними проекторами, що проходять не під прямим кутом до КП.

Косокутні проекції показують загальну форму просторового об'єкта, але істинні розміри та форма зберігаються лише для граней об'єкта, які паралельні КП, тобто кути і довжини при косокутному проектуванні

зберігаються тільки для цих граней. Грані, які не паралельні площині проєкції, спотворюються.

Косокутні проєкції – основний інструмент для побудови тіні об’єкта на площину і для відображення об’єкта в плоскому дзеркалі.

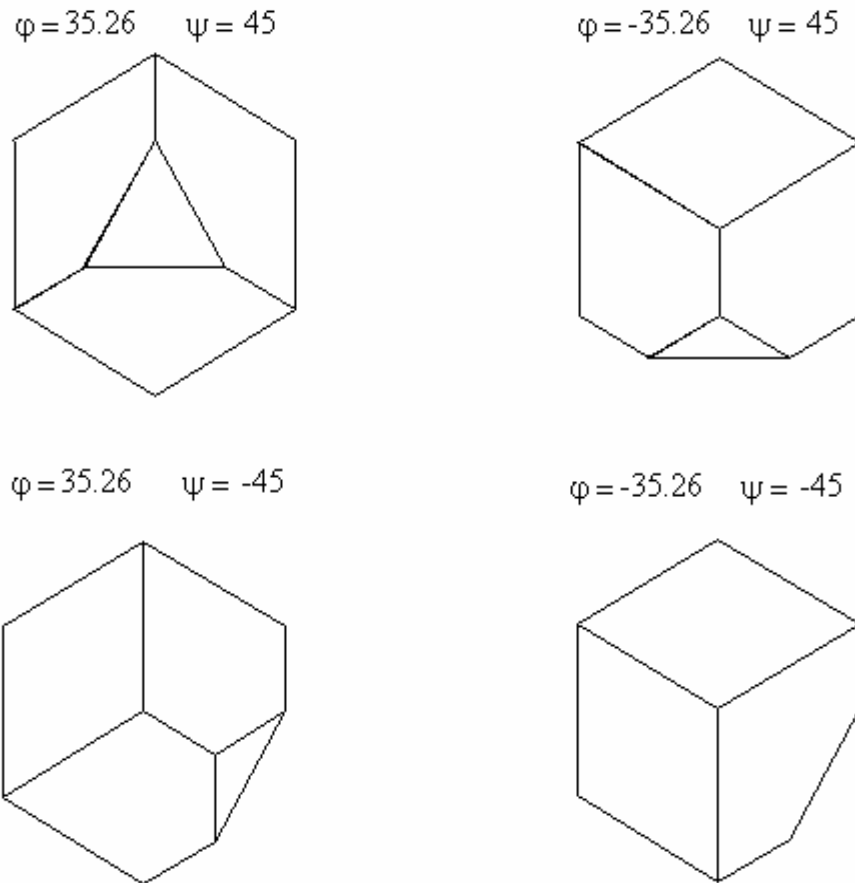


Рис. 12.8. Ізометричні проєкції зрізаного куба

Щоб побудувати матрицю косокутного проєктування, необхідно задати напрям проєктування. Для цього на осі z вибираємо одиничну точку $P(0, 0, 1)$, а в площині проєкцій ($z = 0$) – точку P_1 (рис. 12.9) і вважаємо, що напрям $\overline{PP_1}$ задає напрям проєктування. Позначимо довжину OP_1 через l і нехай OP_1 утворює з віссю x кут α , тоді $x_{p_1} = l \cos \alpha$, $y_{p_1} = l \sin \alpha$. Отже, маємо координати точки $P_1(l \cos \alpha, l \sin \alpha, 0)$ і напрям $\overline{PP_1} = (l \cos \alpha, l \sin \alpha, -1)$.

Крім цього, при косокутному проєктуванні відрізок OP проєктується у відрізок OP_1 , тобто одиничний відрізок проєктується у відрізок довжиною l , тому l можна назвати коефіцієнтом спотворення при косокутному проєктуванні. Як видно з рис. 12.9, $l = \text{ctg} \beta$ ($\text{arcctg} l = \beta$), де β – кут нахилу косих проєкторів до площини проєкції $z = 0$.

Нехай тепер задано довільну точку $Q(x, y, z)$. Знайдемо матричне перетворення косокутного проектування, тобто перетворення, яке точку Q перетворює в точку Q' .

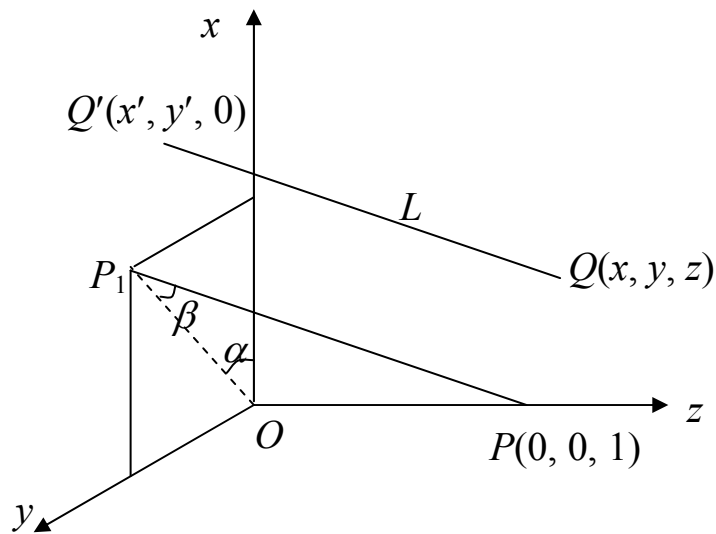


Рис. 12.9. Напрямок косокутного проектування

Рівняння прямої, що проходить через точку Q паралельно вектору $\overline{PP_1}$, має вигляд

$$\frac{X-x}{l \cos \alpha} = \frac{Y-y}{l \sin \alpha} = \frac{Z-z}{-1},$$

де X, Y, Z – біжучі точки прямої L , тому для точки Q' знаходимо

$$\begin{aligned} x' &= x + lz \cos \alpha, \\ y' &= y + lz \sin \alpha, \end{aligned}$$

або в матричній формі

$$(x' \ y' \ 0 \ 1) = (x \ y \ z \ 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ l \cos \alpha & l \sin \alpha & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Це і є матриця косокутного проектування. Змінюючи значення α, β , одержуємо різні варіанти косокутної проекції. Надалі для спрощення запису для матриці косокутного проектування будемо виписувати тільки верхню ліву частину розміром 3×3 .

Зауваження 2. Якщо напрям проектування $\overline{PP_1}$ задати вектором (p_x, p_y, p_z) , то

$$l = \operatorname{ctg} \beta = \frac{\sqrt{p_x^2 + p_y^2}}{p_z}, \quad \sin \alpha = \frac{p_y}{\sqrt{p_x^2 + p_y^2}}, \quad \cos \alpha = \frac{p_x}{\sqrt{p_x^2 + p_y^2}},$$

і матриця косокутного проектування набуває вигляду

$$K_z = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ q_x & q_y & 0 \end{pmatrix}, \text{ де } q_x = \frac{p_x}{p_z}, q_y = \frac{p_y}{p_z}.$$

Аналогічно можна виписати матриці косокутного проектування K_x, K_y

$$K_y = \begin{pmatrix} 1 & 0 & 0 \\ q_x & 0 & q_z \\ 0 & 0 & 1 \end{pmatrix}, \text{ де } q_x = \frac{p_x}{p_y}, q_z = \frac{p_z}{p_y},$$

$$K_x = \begin{pmatrix} 0 & q_y & q_z \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \text{ де } q_y = \frac{p_y}{p_x}, q_z = \frac{p_z}{p_x}.$$

Деякі напрямки вектора $\overline{PP_1}$ дають стаціонарні косокутні проекції. Особливий інтерес являють собою дві косокутні проекції – проекція Кавальє (вільна проекція) і кабінетна проекція. *Проекція Кавальє* одержується тоді, коли кут між проєкторами і площиною проєкції дорівнює 45° ($l = 1$), однак результат цієї проекції виглядає неприродно потовщеним (рис. 12.10).

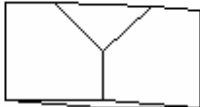
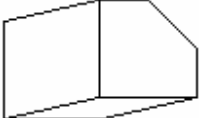
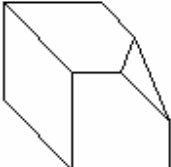
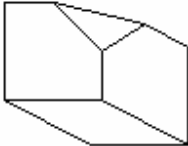
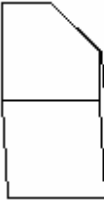
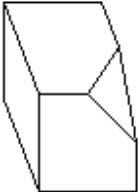
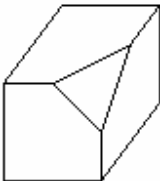
	-45		15
	-30		30
	-15		45
	0		

Рис. 12.10. Проекція Кавальє

Цього недоліку немає в *кабінетній проекції*, в якій коефіцієнт спотворення $l = \frac{1}{2}$, тобто кут між проєкторами і КП $\beta = \text{arcctg}(1/2) = 63,43^\circ$.

Проекція Кавальє визначається вектором $\overline{PP_1} = (1, 1, \sqrt{2})$ і задається матрицею

$$K_{\text{кав}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0,707 & 0,707 & 0 \end{pmatrix}.$$

Кабінетна проекція визначається вектором $(1, 1, 2\sqrt{2})$ і задається матрицею

$$K_{\text{каб}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0,354 & 0,354 & 0 \end{pmatrix}$$

Проекція Кавальє зрізаного куба наведена на рис. 12.10, а кабінетна проекція – на рис. 12.11.

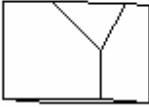

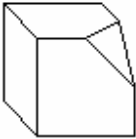
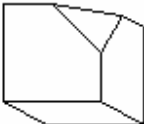
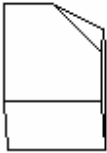
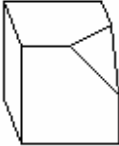
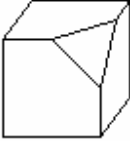
	-45		15
	-30		30
	-15		45
	0		

Рис. 12.11. Кабінетна проекція

12.5. Одноточкова (однофокусна) перспективна проєкція

Спочатку розглянемо простіший випадок центрального проєкування. Припустимо, що проєктивна площина перпендикулярна осі координат, наприклад z , і проходить через точку $C(0,0,c)$ на осі z , а сам спостерігач (камера) знаходиться в початку координат. Нехай задана точка $P(x, y, z)$ у просторі. Знайдемо координати проєкції точки P на КП. Позначимо шукану точку через $P'(x', y', z')$ (рис. 12.12).

З подібності трикутників (рис. 12.12) запишемо пропорції

$$\frac{c}{z} = \frac{y'}{y}, \quad \frac{c}{z} = \frac{x'}{x},$$

звідки $x' = \frac{cx}{z}$, $y' = \frac{cy}{z}$. Якщо розглядати однорідні координати точок

$P(x, y, z, 1)$, $P'(\frac{x}{z/c}, \frac{y}{z/c}, c, 1) = (x, y, z, \frac{z}{c})$, то перетворення точки P у P'

можна задати матричною формулою $P' = P \cdot M_1$, де

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/c \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

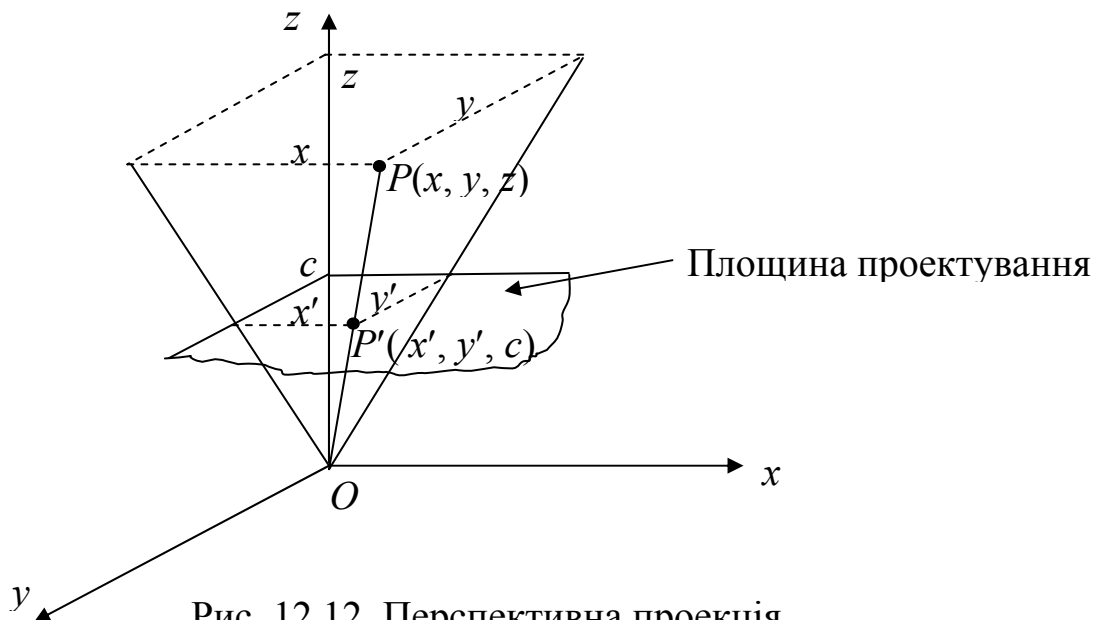


Рис. 12.12. Перспективна проєкція

Розглянемо інший варіант центрального проєкування. Нехай КП знаходиться у площині $z = 0$, а камера – в точці $C(0, 0, c)$ (рис. 12.13).

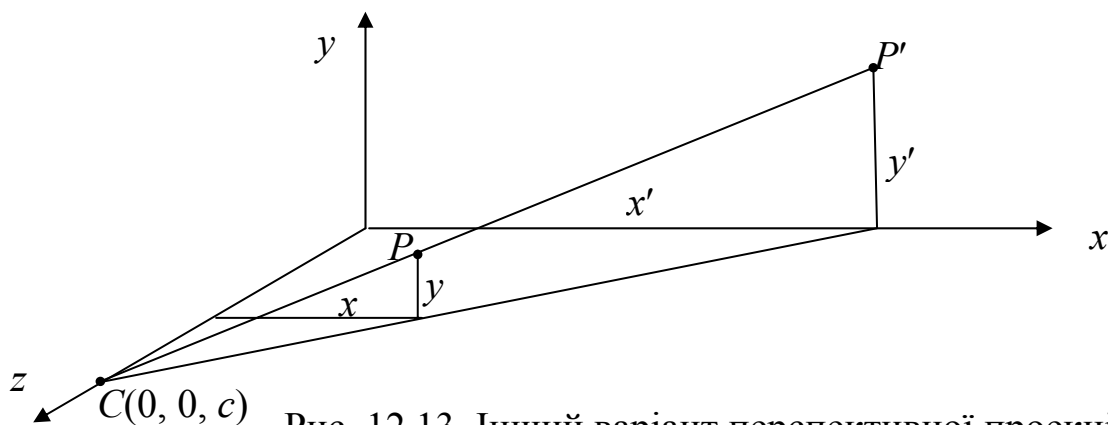


Рис. 12.13. Інший варіант перспективної проєкції

Візьмемо в просторі довільну точку $P(x, y, z)$ і побудуємо пряму, що проходить через точки C та P . Параметричне рівняння цієї прямої має вигляд

$$X = xt, \quad Y = yt, \quad Z = c + (z - c)t.$$

Знайдемо координати точки перетину побудованої прямої з площиною $z = 0$. З умови $Z = 0$ маємо $t = \frac{c}{c - z}$, а отже, $x' = \frac{xc}{c - z} = \frac{x}{1 - \frac{z}{c}}$, $y' =$

$$\frac{yc}{c - z} = \frac{y}{1 - \frac{z}{c}}, \text{ тобто одержуємо точку } P' \left(\frac{x}{1 - \frac{z}{c}}, \frac{y}{1 - \frac{z}{c}}, 0, 1 \right) = P'(x, y, 0, 1 - \frac{z}{c}).$$

Ці ж перетворення можна записати і в матричній формі

$$(x' \ y' \ 0 \ 1) = (x \ y \ 0 \ 1 - \frac{z}{c}) = (x \ y \ z \ 1) \cdot M_2,$$

де M_2 – матриця перспективного проєктування на площину $z = 0$:

$$M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{c} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Зауважимо, що наявність нульового третього стовпця в матриці M_2 зумовлена тим, що z -координата проєкції нульова. Очевидно, що при такому перетворенні повністю губиться інформація про глибину сцени. Формально зберегти z -координату при виконанні перспективного перетворення можна шляхом заміни в матриці M_2 нуля на перетині третього рядка та третього стовпчика на 1. При цьому одноточкове перспективне перетворення (без проєктування на площину) з центром проєктування на осі z задається формулою $X' = X \cdot P_c$, де

$$P_c = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/c \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

або

$$(x' \ y' \ z' \ 1) = (x \ y \ z \ 1) \cdot P_c = (x \ y \ z \ 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{c} \\ 0 & 0 & 0 & 1 \end{pmatrix} = (x \ y \ z \ 1 - \frac{z}{c}).$$

Отже, координати перспективного перетворення точки (x, y, z) в цьому випадку проектування визначаються за формулою

$$(x', y', z', 1) = \left(\frac{x}{1 - z/c}, \frac{y}{1 - z/c}, \frac{z}{1 - z/c}, 1 \right).$$

Тепер перспективне проектування на деяку двовимірну площину може бути одержане як послідовне виконання операцій перспективного перетворення та ортогонального проектування. Наприклад, перспективне проектування на площину $z = 0$ задається матрицею $M_2 = P_c \cdot P_z$, де P_c – матриця перспективного перетворення, а P_z – матриця ортогонального проектування на площину $z = 0$. Аналогічно можна побудувати матриці перспективних перетворень P_a, P_b для випадку, коли центр проєкції знаходиться на осях x та y .

Формули перетворення перспективи містять у знаменнику координату z , а це означає, що такі перетворення нелінійні (точніше дробово-лінійні), тому центральне проектування не зберігає відношення площ і довжин, а паралельні між собою прямі, що не паралельні проєктивній площині, переходять у прямі, які збігаються в одній точці.

Розглянемо пучок прямих паралельних осі z і знайдемо результат дії на них матрицею перспективного перетворення. Кожна пряма такого пучка однозначно визначається точкою $M(a, b, 0)$ своєї перетину з площиною $z = 0$. Рівняння цієї прямої $x = a, y = b, z = t$. Перейдемо до однорідних координат і до них застосуємо матрицю перспективного перетворення P_c . В результаті одержимо

$$(a \ b \ t \ 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{c} \\ 0 & 0 & 0 & 1 \end{pmatrix} = (a \ b \ t \ 1 - \frac{t}{c}) = \left(\frac{a}{1 - \frac{t}{c}} \quad \frac{b}{1 - \frac{t}{c}} \quad \frac{t}{1 - \frac{t}{c}} \quad 1 \right).$$

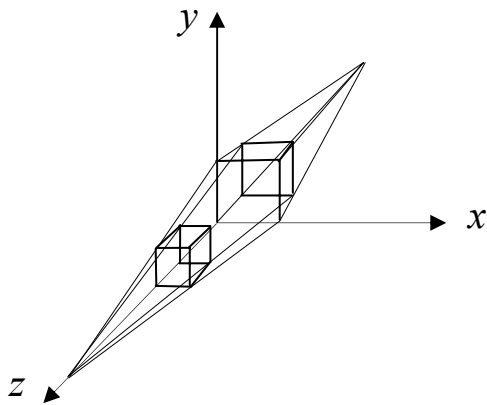


Рис. 12.14. Проекція з однією точкою збігу

При цьому точки образу $(\frac{a}{1-\frac{t}{c}}, \frac{b}{1-\frac{t}{c}}, \frac{t}{1-\frac{t}{c}}, 1)$ при $t \rightarrow \infty$

наближаються до єдиної точки $(0, 0, -c, 1)$, що не залежить від значень a та b . Це означає, що перспективними проєкціями відрізків, паралельних до осі z , є відрізки, які збігаються в єдиній точці $(0, 0, -c, 1)$, що симетричні центру проєкції відносно площини $z = 0$ (рис. 12.14). Ця точка називається *точкою збігу*, вона може роз-

глядатися як проєкція нескінченно віддаленої точки. З практичної точки зору точка збігу – це така точка на передньому або задньому плані об'єкта, в напрямку до якої відбувається поступове зменшення розмірів об'єкта (в границі – об'єкт перетворюється в точку на горизонті).

Зауваження 3. Можна показати, що довільний пучок паралельних прямих, який не паралельний КП, під дією матриці перспективного перетворення P_c переходить у пучок прямих, що збігаються в одній точці.

12.6. Двоточкове та триточкове перспективні перетворення

Одноточкові перспективні перетворення ще не дають адекватного сприйняття тривимірної форми об'єкта. Для цього застосовують складніші перспективні перетворення.

Якщо в четвертому стовпці матриці перспективного перетворення два елементи з перших трьох не дорівнюють нулю, то таке перетворення є двоточковим перспективним перетворенням.

Двоточкові перспективні перетворення задаються формулою

$$\begin{aligned}
 (x' \ y' \ z' \ 1) &= (x \ y \ z \ 1) \begin{pmatrix} 1 & 0 & 0 & -\frac{1}{a} \\ 0 & 1 & 0 & -\frac{1}{b} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = (x \ y \ z \ 1 - \frac{x}{a} - \frac{y}{b}) = \\
 &= (\frac{x}{1 - \frac{x}{a} - \frac{y}{b}} \ \frac{y}{1 - \frac{x}{a} - \frac{y}{b}} \ \frac{z}{1 - \frac{x}{a} - \frac{y}{b}} \ 1).
 \end{aligned}$$

Зауважимо, що двоточкове перспективне перетворення можна одержати шляхом об'єднання двох одноточкових перспективних перетворень, тобто

$$P_{ab} = \begin{pmatrix} 1 & 0 & 0 & -\frac{1}{a} \\ 0 & 1 & 0 & -\frac{1}{b} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -\frac{1}{a} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\frac{1}{b} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = P_a \cdot P_b.$$

Тому двоточкове перспективне перетворення має два центри проєкції: перший на осі x у точці $(a, 0, 0, 1)$, другий на осі y в точці $(0, b, 0, 1)$ і дві точки збігу: на осі x у точці $(-a, 0, 0, 1)$ і на осі y в точці $(0, -b, 0, 1)$ (рис. 12.15). Далі, щоб одержати двоточкову проєкцію на площину $z = 0$, потрібно ще подіяти матрицею ортогонального проєктування на площину $z = 0$.

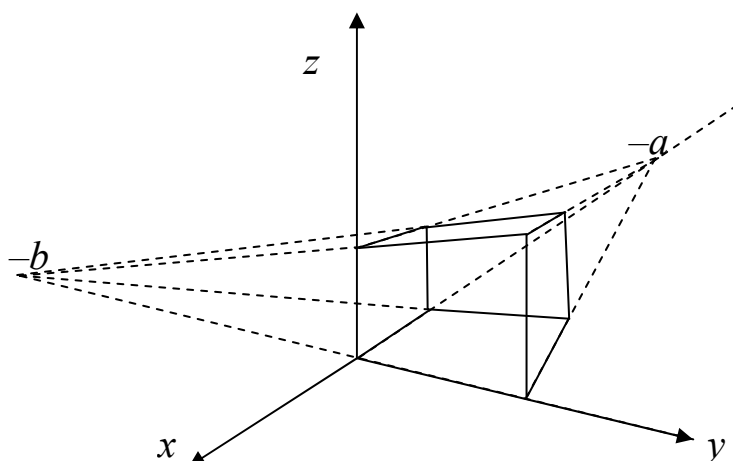


Рис. 12.15. Двоточкове центральне перетворення

Двоточкова центральна проєкція широко застосовується в архітектурному, інженерному та промисловому проєктуванні, в комп'ютерній графіці.

Триточкова перспектива задається матрицею, в якій три перших елементи четвертого стовпчика матриці перетворення не дорівнюють нулю. Триточкове перспективне перетворення

$$(x' \ y' \ z' \ 1) = (x \ y \ z \ 1) \begin{pmatrix} 1 & 0 & 0 & -\frac{1}{a} \\ 0 & 1 & 0 & -\frac{1}{b} \\ 0 & 0 & 1 & -\frac{1}{c} \\ 0 & 0 & 0 & 1 \end{pmatrix} = (x \ y \ z \ 1 - \frac{x}{a} - \frac{y}{b} - \frac{z}{c})$$

має три центри проєкції та три точки збігу. Зазначимо, що триточкове перспективне перетворення може бути одержане за допомогою послідовного застосування трьох одноточкових перспективних перетворень, по одному на кожену координатну вісь.

Триточкові центральні проєкції рідко застосовуються на практиці, бо, по-перше, їх важко конструювати, а по-друге, вони додають мало нового з погляду реалістичності в порівнянні з двоточковими проєкціями.

12.7. Методи створення перспективних видів

Як видно з рис. 12.14, всі одноточкові перспективні проєкції неінформативні, оскільки з кожного центру проєкції видно тільки одну грань куба. Щоб сприйняти тривимірну форму об'єкта на основі тільки однієї проєкції потрібно, щоб було видно кілька граней цього об'єкта. Наприклад, для куба потрібно бачити три грані. Для одноточкової проєкції з фіксованим центром на осях координат із площиною проєктування, яка паралельна до координатних площин, вид із декількома гранями можна одержати, попередньо здійснивши перенесення і/або поворот об'єкта.

Спочатку розглянемо перенесення об'єкта з подальшим одноточковим проєктуванням на площину $z = 0$ із центром проєкції в точці $z = c$. Це перетворення записується у вигляді

$$M = T \cdot M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/c \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/c \\ l & m & 0 & 1-n/c \end{pmatrix}.$$

Одержане перетворення показує, що перенесення вздовж напрямків x та y відкриває три грані куба (рис. 12.16). Перенесення об'єкта вздовж напрямку z приводить до зміни масштабу (через елемент $1 - \frac{n}{c}$) – це відповідає фізичній реальності, оскільки об'єкти, що знаходяться далі

від спостерігача, виглядають меншими. Декілька граней можна відкрити, якщо виконати обертання об'єкта. Поворот відносно однієї осі відкриє принаймні дві грані об'єкта, два повороти відносно різних осей відкриють як мінімум три грані.

Матриця перетворення для повороту навколо осі y на кут ψ і подальшим перспективним проектуванням на площину $z = 0$ із центром проєкції в точці $z = c$ має вигляд

$$M = R_y \cdot M_2 =$$

$$= \begin{pmatrix} \cos \psi & 0 & -\sin \psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \psi & 0 & \cos \psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/c \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos \psi & 0 & 0 & \frac{\sin \psi}{c} \\ 0 & 1 & 0 & 0 \\ \sin \psi & 0 & 0 & -\frac{\cos \psi}{c} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

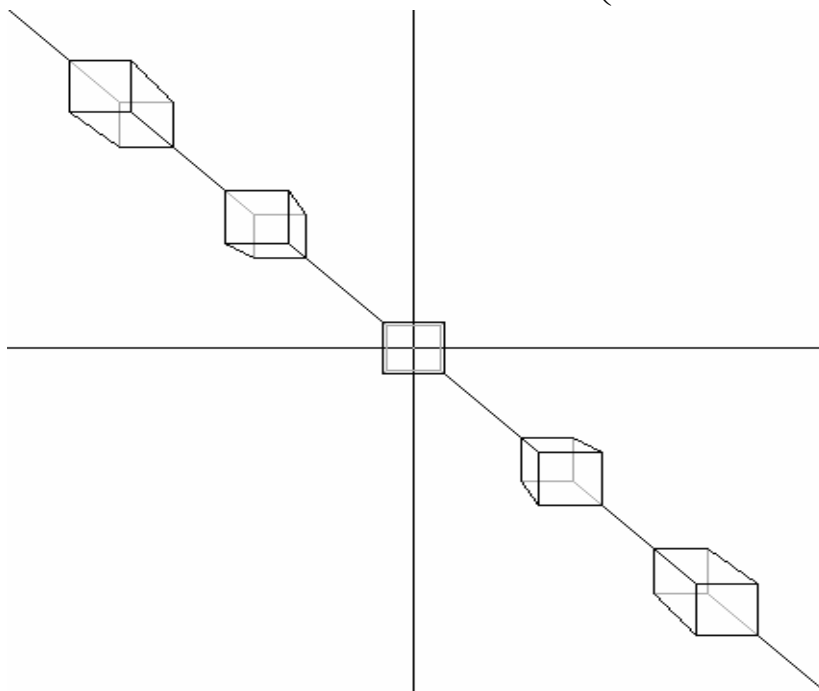


Рис. 12.16. Одноточкова перспективна проєкція перенесеного об'єкта в x, y -напрявках

Звідси видно, що обертання навколо осі x (аналогічно y) з подальшим проектуванням еквівалентне двоточковому перспективному перетворенню (не дорівнюють нулю два елементи матриці 3×1 , що відповідають перспективному перетворенню). Тобто поворот навколо головної осі не відкриває всі три грані об'єкта. Для цього поворот можна скомбінувати з переміщенням або здійснити обертання відносно двох осей x та y , а потім перспективне проектування на площину $z = 0$. Таке перетворення задається матрицею $M = R_y \cdot R_x \cdot M_2$.

Контрольні питання та завдання

1. Як будуються проєкції? Дайте їх класифікацію.
2. Випишіть матриці ортогональних перетворень.
3. В чому полягають переваги та недоліки перспективних проєкцій?
4. Як утворюється аксонометрична проєкція? Назвіть її різновиди.
5. Випишіть матрицю аксонометричного проєктування.
6. Які властивості має диметрична/ізометрична проєкції?
7. Під яким кутом відображаються на площині екрану осі ізометричної проєкції?
8. Як вибирається напрям косокутного проєктування?
9. Який вигляд має матриця косокутного проєктування?
10. Назвіть дві особливі косокутні проєкції. Порівняйте їх.
11. Як побудувати матрицю одноточкової перспективної проєкції?
12. Які властивості має матриця перспективного перетворення?
13. Що таке точка збігу? Як довести її існування?
14. Як задаються двоточкові перспективні перетворення?
15. Які властивості має двоточкова перспективна проєкція?
16. Випишіть матрицю триточкової перспективної проєкції. Вкажіть центри проєктування та точки збігу.
17. Обчислити матрицю $M = R_y \cdot R_x \cdot M_2$.
18. Як створити таку перспективу, щоб можна було сприйняти тривимірну форму об'єкта?

Вправи і задачі для самостійного виконання

1. Побудувати вид зліва/ззаду зрізаного куба за допомогою ортогонального проєктування на площину $z = 0$. *Вказівка. Попередньо необхідно здійснити відповідні повороти.*
2. Обчислити координати ортогональної проєкції на площину $z = 0$ одиничного куба, який повернуто на кут 45° відносно осі, що проходить через початок координат і протилежну вершину куба.
3. Обчислити координати ортогональної проєкції на площину $z = 0$ одиничного куба, який спочатку необхідно симетрично відобразити відносно площини, що проходить через три задані точки.
4. Побудувати матрицю триметричного проєктування при $\psi = 30^\circ$, $\varphi = 45^\circ$. Обчислити координати проєкції зрізаного куба.
5. Знайти координати проєкції Кавальє зрізаного куба при $\alpha = 45^\circ$.
6. Виконати одноточкове перспективне перетворення одиничного куба на площину $z = 0$ з центром проєкції в точці $(0, 0, 5)$.
7. Побудувати двоточкову перспективну проєкцію одиничного куба на площину $z = 0$ для центрів проєкції $(5, 0, 0)$ та $(0, 5, 0)$.
8. Побудувати матриці перетворення диметрії в ізометрію і навпаки.

Тема 13. Системи координат та їх перетворення

13.1. Системи координат

У комп'ютерній графіці використовуються декілька різних систем координат. З одного боку, координати точок об'єкта зручно задавати в локальній системі координат об'єкта. З іншого боку, якщо в сцені багато об'єктів, то для їх розміщення необхідно мати одну загальну (світову) систему координат. Далі спостерігач дивиться на об'єкт сцени через камеру, причому камера може переміщуватися і змінювати напрям спостереження. Тому необхідно мати систему координат, що зв'язана з камерою. Від характеристик камери залежить об'єм простору, який спостерігач бачить через камеру. Різні об'єми видимості камери перед подальшою обробкою (візуалізацією на екрані комп'ютера) зручно спочатку перевести в деякий стандартний об'єм (просторовий куб), що має свою систему координат. Результатом візуалізації є виведення видимої частини об'єкта через камеру на екран або на частину екрану (вікно або область виведення), які мають свої системи координат.

Розглянемо більш детально ці системи координат та зв'язок між ними.

- *Система координат об'єкта (СКО).* Ця система координат жорстко зв'язана власне з об'єктом і в ній задаються координати точок об'єкта, наприклад координати вершин полігональної моделі. Це локальна система координат і координати в ній зберігаються в незмінній формі. Розглянемо деяку точку P_0 об'єкта (на рис. 13.1 це куб) із локальними координатами (x_0, y_0, z_0) та відповідними однорідними координатами $P_0(x_0, y_0, z_0, 1)$.

- *Світова система координат (ССК).* У цій системі координат можна представити координати всіх об'єктів сцени одночасно. Вона теж є правосторонньою системою координат. У цій системі координат точка P_0 вже буде мати інші координати $(x, y, z, 1)$. Позначимо через M_0 матрицю перетворення локальної системи координат у світову. Таке перетворення називається *модельним*, а матриця M_0 – *матрицею модельного перетворення*, тоді зв'язок між координатами в цих системах задається формулою

$$P = P_0 \cdot M_0. \quad (13.1)$$

- *Видова система координат (ВСК).* Це система координат спостерігача. Вважається, що спостерігач знаходиться в початку координат, а напрям його погляду на об'єкт збігається з віссю z , яка направлена на "центр" об'єкта. Вісь x направлена вправо від спостерігача, а вісь y – вгору, тобто видова система координат є лівосторонньою, а це відпо-

відає вибору напрямку осей екранної системи координат (вісь z направлена вглиб екрану). З видовою системою координат пов'язане *видове перетворення*, яке здійснює відображення світових координат у видові координати.

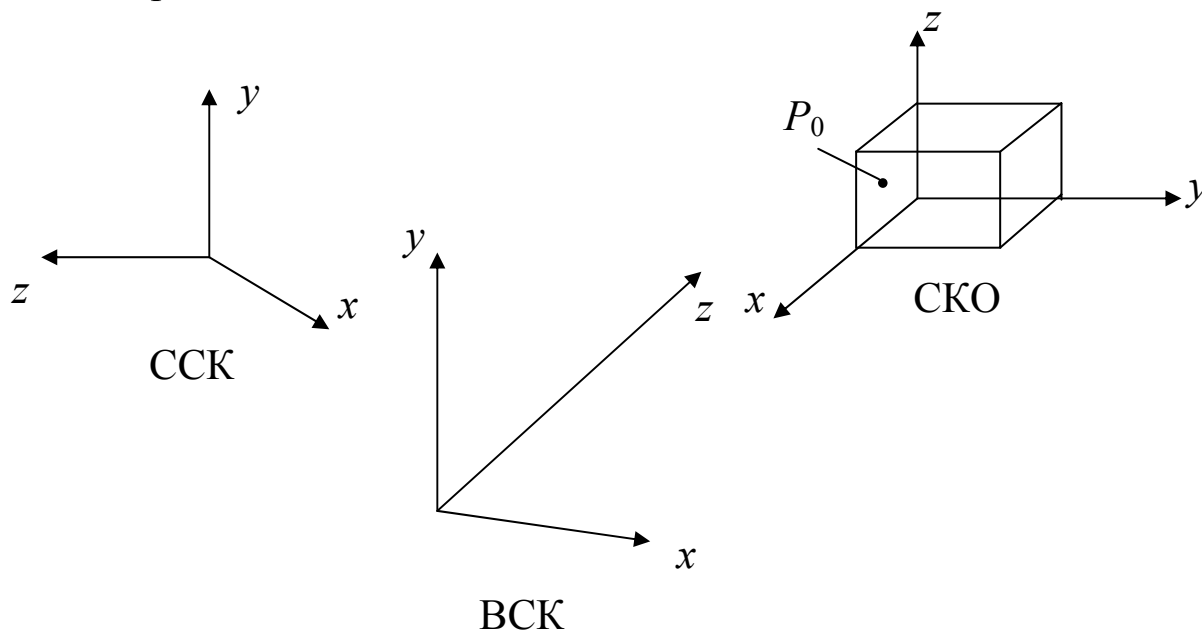


Рис. 13.1. Локальна (СКО), світова (ССК) та видова (ВСК) системи координат

Позначимо матрицю видового перетворення через M_v , а координати точки P у видовій системі координат – через $P_v(x_e, y_e, z_e, 1)$. Тоді одержуємо

$$P_v = P \cdot M_v. \quad (13.2)$$

Із рівностей (13.1), (13.2) маємо ще один вираз для координат точки P_v у видовій системі координат

$$P_v = P_0 \cdot M_0 \cdot M_v, \quad (13.3)$$

Матриця $M_{0v} = M_0 \cdot M_v$ називається *матрицею модельно-видового перетворення*.

Спостерігач не може бачити через камеру весь простір одразу. Він бачить деякий обмежений об'єм, що задається у вигляді зрізаної піраміди при центральному проектуванні або у вигляді паралелепіпеда при ортографічному проектуванні. Кінцевою метою графічної системи є відображення тривимірної піраміди видимості на двовимірне вікно екрану.

- *Система координат виведення зображення (СКВЗ)*. Ця система координат чотиривимірною простору, точки якого являють собою однорідні координати тривимірного простору в лівосторонній системі координат. Перетворення проєкції відображають простір видової системи координат у дану систему так, що множина відображених точок утворює куб із центром у початку координат і гранями, паралельними

осям координат. Тобто точка $P_v(x_e, y_e, z_e, 1)$ при проектуванні відобразиться в точку $P_c(x_c, y_c, z_c, h_c)$ за допомогою перетворення

$$P_c = P_v \cdot M_{pr}, \quad (13.4)$$

де M_{pr} – матриця перетворення проєкції.

Далі необхідно нормалізувати координати, тобто точці $P_c(x_c, y_c, z_c, h_c)$ поставити у відповідність точку $P_d(x_d, y_d, z_d) = (\frac{x_c}{h_c}, \frac{y_c}{h_c}, \frac{z_c}{h_c})$ так, щоб $x_d, y_d, z_d \in [-1, 1]$. Цю систему координат можна назвати системою нормалізованого об'єму видимості.

- *Система координат області виведення Viewport (СКОВ).* Область виведення – це прямокутна область вікна (екрану), в яку виводиться зображення, тобто вміст нормалізованого об'єму видимості (точки P_d). За замовчуванням область виведення збігається з усім вікном (екраном), але інколи для області виведення задають її розміри та координати розміщення. Незважаючи на те, що область виведення двовимірна, система координат області виведення містить координату z для правильного врахування глибини сцени при подальшій обробці графічного зображення.

Початок тривимірної системи координат області виведення розміщений у центрі області виведення, вісь x_p направлена горизонтально вправо, вісь y_p – вертикально вгору, а вісь z_p – углиб екрана (рис. 13.2).

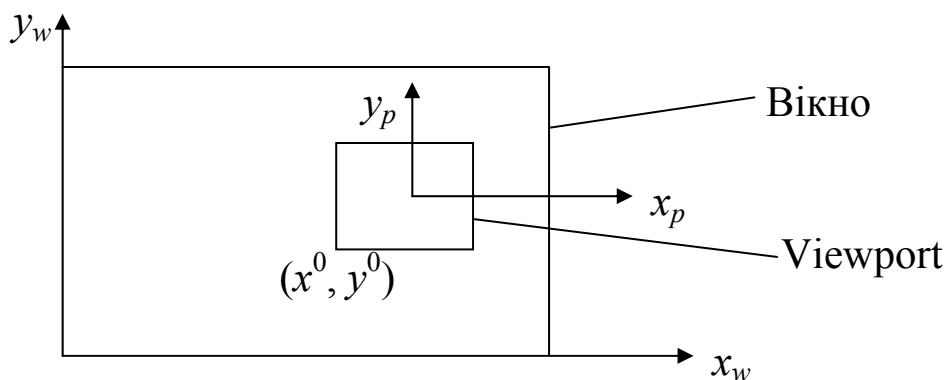


Рис. 13.2. Віконна система координат x_w, y_w та система координат області виведення (Viewport) x_p, y_p

У цій системі координат точки $P_p(x_p, y_p, z_p)$ одержуються з координат $P_d(x_d, y_d, z_d)$ за допомогою матриці перетворення M_{vp} , тобто

$$P_p = P_d \cdot M_{vp}. \quad (13.5)$$

- *Віконна система координат (ВСК).* Початок цієї системи координат розміщується в лівому нижньому куті вікна, вісь x_w направлена горизонтально вправо, а вісь y_w – вгору. Таким чином, ця система координат одержується із системи координат області виведення шляхом пере-

несення початку координат. Перенесення системи координат теж можна записати в матричній формі

$$P_w = P_p \cdot M_w, \quad (13.6)$$

де M_w – матриця перенесення.

На рис. 13.3 схематично показана вся описана вище послідовність перетворень координат.

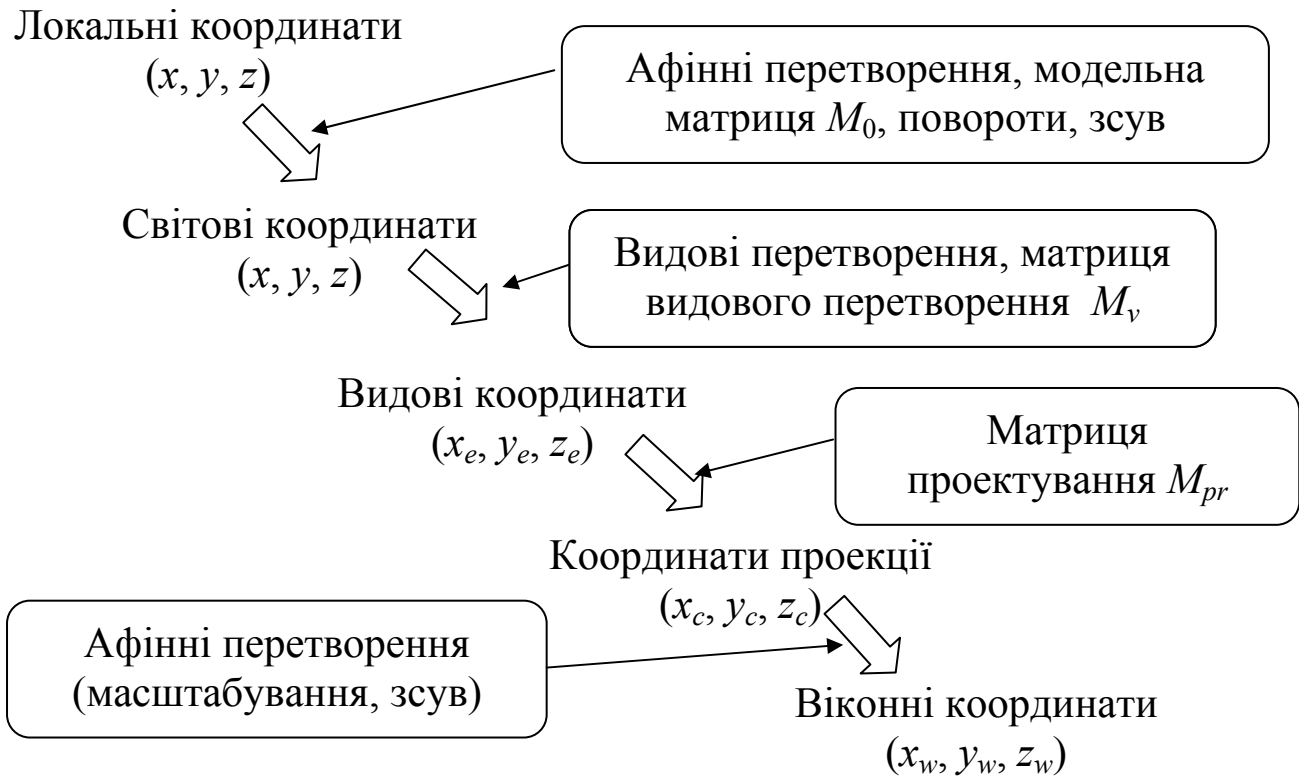


Рис. 13.3. Послідовність перетворень координат

У кожний момент у графічній системі знаходяться модельно-видова M_{mv} та проєкційна M_{pr} матриці, а також M_{vp} – матриця області виведення. При виконанні візуалізації об’єктів, координати кожної вершини об’єкта послідовно множаться на ці матриці (див. співвідношення (13.3) – (13.6)). Розглянемо, як реально формуються ці матриці.

13.2. Видове перетворення

Для виконання видових перетворень повинні бути задані точка спостереження E і об’єкт у світовій системі координат, яка завжди є правосторонньою. Вважається, що початок світової системи координат знаходиться поблизу “центру” об’єкта. Якщо ця умова не виконується, то необхідно здійснити перенесення системи координат. Нехай точка спостереження E задана у сферичних координатах ρ, θ, φ (рис. 13.4). Тоді світові координати можуть бути обчислені за формулами

$$x_E = \rho \sin \varphi \cos \theta, \quad y_E = \rho \sin \varphi \sin \theta, \quad z_E = \rho \cos \varphi.$$

І навпаки, якщо задані декартові координати x_E, y_E, z_E точки E , то можна обчислити відповідні їй сферичні координати.

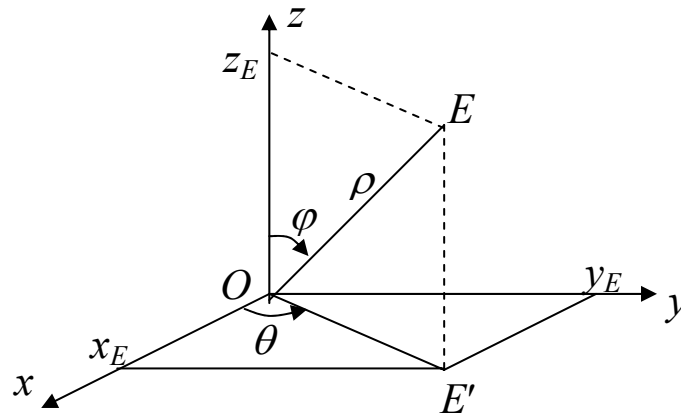


Рис. 13.4. Сферичні координати точки спостерігача E

Вектор EO задає напрям спостереження. З точки спостереження E можна бачити об'єкти сцени, що знаходяться всередині деякого конуса, вісь якого збігається з EO , а вершина конуса знаходиться в точці E .

Для видової системи координат напрям осі z_e збігається з напрямом EO , додатна піввісь x_e направлена вправо, а додатна піввісь y_e – вверх (рис. 13.5).

Отже, система видових координат лівостороння. Таке розміщення осей зручне для комп'ютерної графіки (вісь z_e направляється вглиб екрану). Видове перетворення може бути записане в формі

$$(x_e, y_e, z_e, 1) = (x, y, z, 1) \cdot M_v,$$

де M_v – матриця видового перетворення розміром 4×4 .

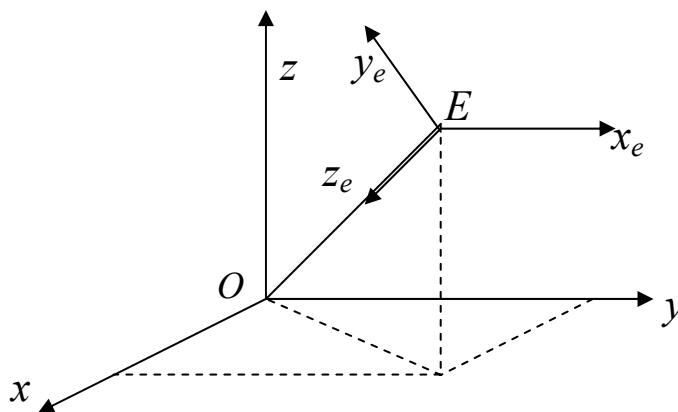


Рис. 13.5. Система видових координат

Для знаходження матриці M_v необхідно здійснити ряд перетворень.

1) Перенесення системи координат так, щоб точка E стала новим початком координат. Матриця цього перетворення має вигляд

$$T^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_E & -y_E & -z_E & 1 \end{pmatrix}$$

2) Поворот системи навколо осі z .

Повернемо систему координат навколо осі z на кут $(\frac{\pi}{2} - \theta)$ в напрямку від y до x (від'ємний напрямок). В результаті вісь y збігатиметься з горизонтальною складовою відрізка EO , а вісь x стане перпендикулярною до площини EOE' (рис. 13.6).

Матриця повороту системи координат у від'ємному напрямку еквівалентна матриці повороту точки в додатному напрямку, тобто

$$R_z = \begin{pmatrix} \cos\left(\frac{\pi}{2} - \theta\right) & \sin\left(\frac{\pi}{2} - \theta\right) & 0 & 0 \\ -\sin\left(\frac{\pi}{2} - \theta\right) & \cos\left(\frac{\pi}{2} - \theta\right) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \sin\theta & \cos\theta & 0 & 0 \\ -\cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

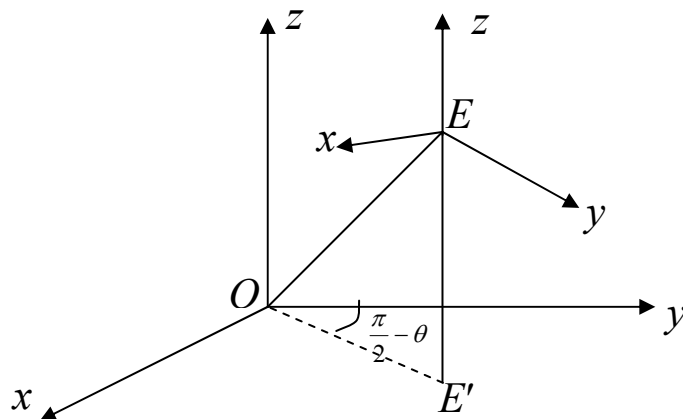


Рис.13.6. Поворот відносно осі Z на кут $(\frac{\pi}{2} - \theta)$

3) Поворот системи координат навколо осі x . Оскільки нова вісь z повинна збігатися за напрямом із відрізком EO , то повернемо систему координат навколо осі x на кут $(\pi - \varphi)$ у додатному напрямку (рис. 13.7). Це перетворення можна здійснити за допомогою матриці

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\pi - \varphi) & -\sin(\pi - \varphi) & 0 \\ 0 & \sin(\pi - \varphi) & \cos(\pi - \varphi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -\cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & -\cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

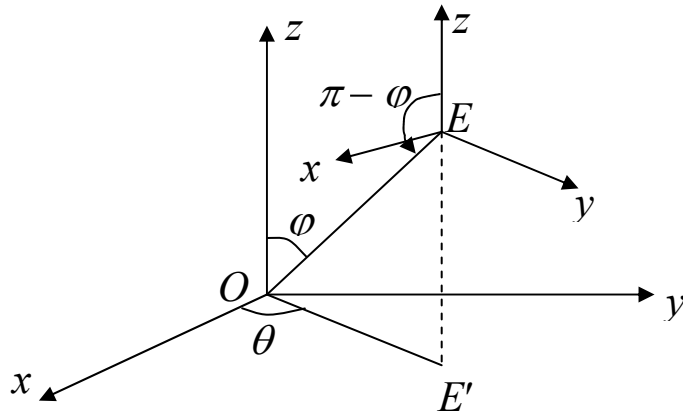


Рис. 13.7. Поворот відносно осі E_x

У результаті цього повороту нова вісь z буде направлена вздовж EO .

4) Зміна напрямку осі x . Після трьох перетворень вісі z та y матимуть правильну орієнтацію, а вісь x повинна бути направлена в протилежний бік, тому необхідно здійснити перетворення $x' = -x$. Матриця цього перетворення має вигляд

$$M_{yz} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Тоді матриця видового перетворення M_v обчислюється як добуток

$$M_v = T^{-1} \cdot R_z \cdot R_x^{-1} \cdot M_{yz}.$$

Перемножуючи ці матриці, знаходимо

$$M_v = \begin{pmatrix} -\sin \theta & -\cos \varphi \cos \theta & -\sin \varphi \cos \theta & 0 \\ \cos \theta & -\cos \varphi \sin \theta & -\sin \varphi \sin \theta & 0 \\ 0 & \sin \varphi & -\cos \varphi & 0 \\ 0 & 0 & \rho & 1 \end{pmatrix}.$$

Отже, застосовуючи видове перетворення M_v , одержуємо координати точок об'єкта у видовій системі координат, тобто (x_e, y_e, z_e) . Використовуючи ці координати, можемо побудувати ортогональну чи центральну проекцію об'єкта. Значення z_e будуть використовуватися для усунення невидимих ліній та граней, наприклад у методі Z -буфера.

13.3. Перспективне проектування

Нам уже відомі координати точок об'єкта (x_e, y_e, z_e) у видовій системі координат. Надалі світові координати вже не будуть використовуватися, тому видові координати (x_e, y_e, z_e) будемо просто позначати через (x, y, z) . Якщо картинну площину розмістити перпендикулярно осі z на віддалі s від спостерігача, то координати центральної проекції знахо-

дяться за формулами

$$x_c = \frac{cx}{z}, \quad y_c = \frac{cy}{z}$$

або за допомогою матричного перетворення $P_c = P \cdot M_1$ із матрицею M_1 ,

де

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/c \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Осі x_c, y_c, z_c мають такий же напрям, що й x_e, y_e, z_e (рис. 13.8).

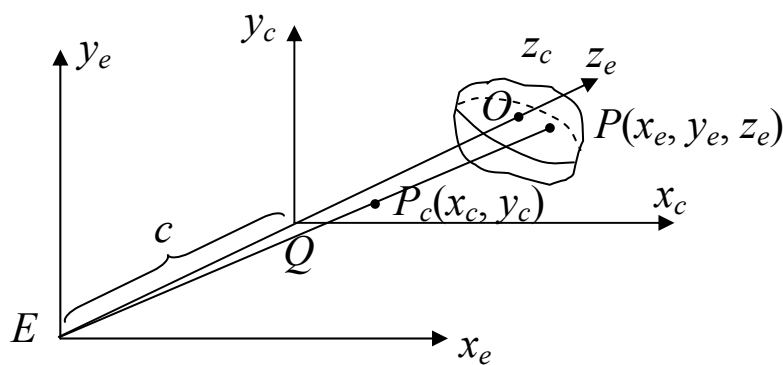


Рис. 13.8. Картинна площина і видові координати

Точка $Q(0, 0, c)$ буде знаходитися в центрі проєкції, тобто для точок проєкції діапазон координат x_c (аналогічно y_c) буде містити точку нуль десь посередині цього інтервалу і їх можна нормувати, наприклад так, щоб $x_c, y_c \in [-1, 1]$.

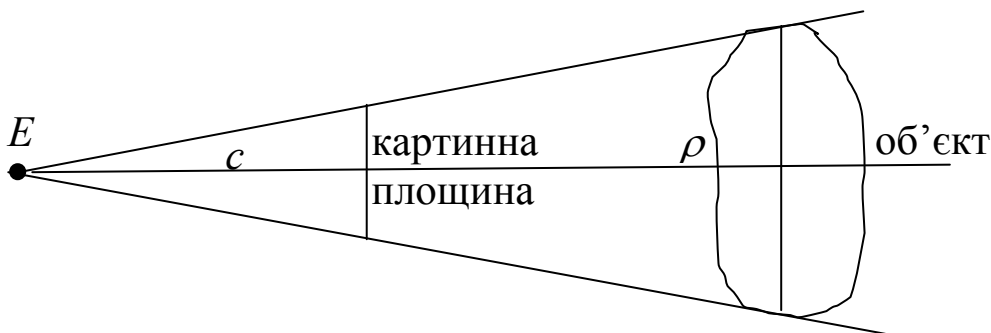


Рис. 13.9. Розмір картини і об'єкта

Окрім цього, розмір картини проєкції можна підбирати за рахунок відповідного вибору числа c – віддалі між точкою спостереження E і картинною площиною $x_c Q y_c$.

Грубо кажучи, маємо співвідношення

$$\frac{\text{розмір картини}}{c} = \frac{\text{розмір об'єкта}}{\rho},$$

яке випливає з подібності трикутників (рис. 13.9), звідки

$$c = \rho \frac{\text{розмір картини}}{\text{розмір об'єкта}},$$

де ρ – віддаль від точки E до “центру” об'єкта.

Це співвідношення можна застосовувати для оцінки горизонтальних і вертикальних розмірів, а не для точного визначення розмірів, оскільки тривимірний об'єкт може мати складну форму і тоді незрозуміло, що є розміром об'єкта.

Співвідношення довжин горизонтальної та вертикальної сторін ($l : h$) прямокутної рамки, через яку людина спостерігає за навколишнім світом, визначають кути огляду. Для нормального людського ока горизонтальний кут огляду $\alpha \approx 150^\circ$, вертикальний кут $\beta \approx 110^\circ$ і оптимальна пропорція параметрів рамки $l : h = 7,5 : 3$.

Для аналізу глибини сцени нам потрібно мати ще й третю координату z_c , оскільки різним z відповідають різні z_c . Для цього потрібно модифікувати матрицю M_1 , оскільки перетворення M_1 для всіх точок (x, y, z) дає однакове $z_c = c$.

Відобразимо точку z у z_c за формулою $z_c = \frac{z - \rho}{z}$, тоді точка $P(x, y, z)$ відобразиться в точку $P_c\left(\frac{x}{z/c}, \frac{y}{z/c}, \frac{z - \rho}{z}, 1\right) = \left(x, y, \frac{z - \rho}{c}, \frac{z}{c}\right)$. Це перетворення можна задати за допомогою формули

$$\left(x, y, \frac{z - \rho}{c}, \frac{z}{c}\right) = (x, y, z, 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{c} & \frac{1}{c} \\ 0 & 0 & -\rho/c & 0 \end{pmatrix} = (x, y, z, 1) M'_1$$

(матриця M'_1 постійна для всіх z).

Побудову модифікованої матриці M'_1 можна виконати й іншим способом: задамо у видовій системі координат дві обмежуючі площини для об'єкта по осі z , тобто площини $z = n$ та $z = f$ ($0 < n < f$) (рис. 13.10).

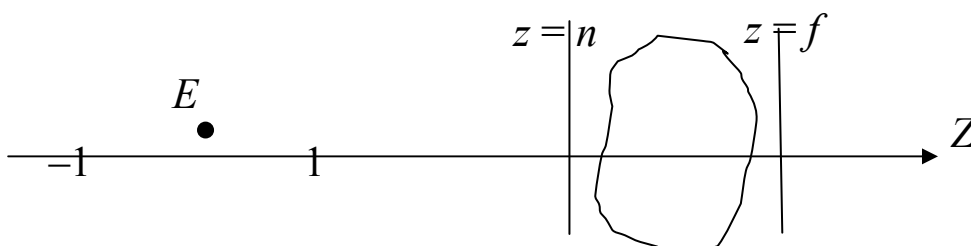


Рис. 13.10. Обмежуючі площини об'єкта

Далі відобразимо площину $z = n$ у $z = -1$, а $z = f$ в $z = 1$. Будемо шукати матрицю цього перетворення у вигляді

$$M'_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & p & q \\ 0 & 0 & r & s \end{pmatrix}.$$

Помножимо на неї вектор $(x, y, z, 1)$, тоді одержимо

$$\begin{aligned} (x_c^*, y_c^*, z_c^*, 1) &= (x, y, z, 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & p & q \\ 0 & 0 & r & s \end{pmatrix} = (x, y, pz + r, qz + s) = \\ &= \left(\frac{x}{qz + s}, \frac{y}{qz + s}, \frac{pz + r}{qz + s}, 1 \right). \end{aligned}$$

Точка $\frac{x}{qz + s}$ повинна відобразитися в $\frac{x}{z/c}$, тобто $x_c = \frac{x}{qz + s} = \frac{x}{z/c}$.

Звідки знаходимо, що $s = 0$, $q = \frac{1}{c}$. При $s = 0$, $q = \frac{1}{c}$, $y_c = \frac{y}{qz + s} = \frac{y}{z/c}$.

Далі площину $z = n$ відображаємо в площину $z = -1$ і одержуємо співвідношення

$$\frac{pn + r}{qn + s} = -1, \quad pn + r = -\frac{n}{c}.$$

Аналогічно площина $z = f$ відображається в площину $z = 1$. З умов

$$\frac{pf + r}{qf + s} = 1, \quad pf + r = \frac{f}{c}.$$

знаходимо

$$p = \frac{1}{c} \frac{f + n}{f - n}, \quad r = -\frac{1}{c} \frac{2fn}{f - n}.$$

Отже, матриця M'_1 має вигляд

$$M'_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{c} \frac{f + n}{f - n} & \frac{1}{c} \\ 0 & 0 & -\frac{1}{c} \frac{2fn}{f - n} & 0 \end{pmatrix}.$$

Зауваження 1. У графічних системах (наприклад, OpenGL) для кожного виду проектування задається піраміда видимості, тобто той об'єм простору, який необхідно відобразити у вікні. На основі цієї ін-

формації графічна система створює матрицю проектування. При візуалізації об'єктів координати точок об'єктів спочатку автоматично перемножують на модельно-видову матрицю, а потім – на поточну проєкційну матрицю.

13.4. Відображення у вікно виведення

Ми вже здійснили перехід до нових координат – координат проєкції (x_c, y_c). Ці координати можуть бути використані для формування зображення на графічних пристроях виведення, але при цьому необхідно здійснити додаткові перетворення, оскільки система координат у проєкційній площині може не збігатися із системою координат пристрою виведення зображення. Наприклад, необхідно зображення, яке вимірюється в метрах (см), зобразити на дисплеї, а на дисплеї одиниці вимірювання – пікселі, тому потрібно перерахувати лінійні одиниці вимірювання в пікселі.

Позначимо

$$X_{\max} = \max(|x_c|), Y_{\max} = \max(|y_c|),$$

тоді $-X_{\max}, X_{\max}, -Y_{\max}, Y_{\max}$ задають границі проєкції сцени (рис. 13.11).



Рис. 13.11. Границі сцени та області виведення

Координати точки (x_p, y_p) у системі координат області виведення будуть одержуватися за формулами масштабування

$$\begin{aligned} x_p &= k_x x_c, \\ y_p &= k_y y_c. \end{aligned}$$

Знайдемо k_x, k_y враховуючи, що координати початку системи координат x_p, y_p знаходяться в центрі області виведення.

Нехай розміри області виведення $2c_1$ по горизонталі та $2c_2$ по вертикалі (в пікселях), тоді

$$k_x = \frac{c_1}{X_{\max}}, k_y = \frac{c_2}{Y_{\max}}.$$

Для перетворення, що зберігає пропорції об'єктів, необхідно вибрати однаковий коефіцієнт розтягу/стиску k для всіх координат, тобто

$$k = \min(k_x, k_y).$$

Зауваження 2. Якщо відомі нормалізовані координати проєкції x_d, y_d , то для знаходження x_p, y_p користуються формулами $x_p = c_1 x_d, y_p = c_2 y_d$.

Знайдемо віконні координати точки, виходячи з координат точки в області виведення (x_p, y_p) . Нагадаємо, що область виведення – це прямокутник, в якому (x^0, y^0) – координати лівого нижнього кута вікна прямокутника (рис. 13.2), а $2c_1, 2c_2$ визначають його розмір (рис. 13.11). Отже, оскільки початок системи координат області виведення знаходиться в точці $(x_0 + c_1, y_0 + c_2)$, то віконні координати точки одержуються з координат області виведення за формулами (див. рис. 13.2):

$$\begin{aligned}x_w &= x_p + x^0 + c_1, \\y_w &= y_p + y^0 + c_2.\end{aligned}$$

Зауваження 3. Координати z у віконній системі координат не перетворюються, але будуть у подальшому використовуватися для аналізу глибини сцени.

Контрольні питання та завдання

1. Які системи координат використовують у комп'ютерній графіці?
2. Яке перетворення задається модельною матрицею?
3. Наведіть визначення видової системи координат.
4. Як здійснити видове перетворення?
5. Який вигляд має матриця видового перетворення?
6. Як одержати віконні координати точки?
7. Поясніть як одержати координати об'єкта в системі координат області виведення.

Вправи і задачі для самостійного виконання

1. Задана піраміда $ABCD$ з вершинами $A(10; 20; 10)$, $B(20; 20; 10)$, $C(20; 10; 20)$, $D(20; 20; 40)$. Побудувати перспективу з точками збігу $(50; 0; 0)$, $(0; 0; 50)$.
2. Обчислити координати точки $A(10; 20; 30)$ у видовій системі координат з початком координат у точці $(30; 40; 50)$.
3. Нехай відсікаюче вікно в системі координат x_w, y_w має координати $(x_{wmin}, y_{wmin}) - (x_{wmax}, y_{wmax})$. Відобразити його вмістиме в одиничний нормований квадрат. Відображення вмістимого записати у матричному вигляді.
4. Відобразити вмістиме нормованого квадрата в прямокутну область виведення на екрані.

Тема 14. Усунення невидимих ліній і граней

14.1. Основні поняття

Задача усунення невидимих ділянок ліній і поверхонь (задача загороджування) – одна з найважливіших та найважчих задач при створенні на комп'ютері якісного реалістичного зображення 3D-об'єктів.

Нехай задано тривимірний об'єкт і видові параметри, що визначають тип проекції та картинну площину. Потрібно визначити, які ребра та грані поверхні об'єкта видимі, якщо на об'єкт дивитися ззовні з деякого центру проектування (для центральних проекцій) або вздовж напрямку проектування (для паралельних проекцій). Виводити на екран необхідно тільки видимі ребра та грані.

Хоча ця задача формулюється просто, однак вона вимагає багато обчислень і машинного часу. Складність цієї задачі привела до появи великої кількості різних методів її розв'язання, в тому числі й апаратних. Але, незважаючи на існування численних методів, не існує універсального алгоритму розв'язання цієї задачі, тобто не існує алгоритму, однаково придатного для різних типів об'єктів та сцен. Натомість розроблені ефективні методи розв'язання окремих класів цієї задачі.

При побудові каркасних зображень (об'єкти в цій моделі задаються сукупністю ребер) використовуються методи усунення невидимих ліній, для візуалізації суцільних тіл (тут об'єкти задаються сукупністю видимих граней) використовуються методи усунення невидимих поверхонь.

Для моделювання процесів у реальному часі потрібні швидкі алгоритми, а для задач мультиплікації потрібні ще й алгоритми, які генерують зображення з тінями, ефектами освітлення тощо. В міру створення швидких алгоритмів можна будувати дедалі більш реалістичні зображення.

Всі алгоритми усунення невидимих ліній та поверхонь містять сортування, тому їх ефективність істотно залежить від ефективності процесу сортування. Застосування до розв'язування задачі загороджування тих чи інших алгоритмів залежить від форми, властивостей та взаємного розташування компонент 3D-сцени. Слід окремо виділити опуклі многогранники (кожна грань опуклого многогранника або повністю видима, або повністю невидима). Якщо об'єкти сцени мають складну форму, то задачу усунення невидимих граней деталізують і спрощують.

Алгоритми усунення невидимих ліній та поверхонь у залежності від вибору системи координат або простору, в якому вони працюють, можна поділити на два класи:

- алгоритми, що працюють у просторі об'єкта на основі аналізу взаєм-

ного розташування геометричних фігур. Вони особливо корисні в тих додатках, в яких необхідна висока точність;

- алгоритми, що працюють у просторі зображень на основі аналізу взаємного розташування проєкцій геометричних фігур на картинній площині (тут маємо справу із системою координат екрана).

Існують алгоритми, що по чергово працюють в об'єктному просторі та у просторі зображень. Найрозповсюдженішими на практиці методами розв'язування задач усунення невидимих ліній та поверхонь є метод прямого перебирання, метод Z-буфера, метод поточних горизонтів, метод усунення нелицьових граней, метод розбиття картинної площини, алгоритм Робертса, метод аналізу кількісної невидимості точок поверхні, метод сортування за глибиною (алгоритм художника), метод трасування променів та ін. Розуміння суті цих методів лежить в основі роботи з 3D-графікою навіть при використанні сучасних графічних бібліотек. Кожний алгоритм має свої особливості й обмеження, які слід враховувати при створенні графічних програм.

14.2. Алгоритм поточного горизонту

Алгоритм поточного горизонту найчастіше використовують для усунення невидимих ліній поверхні $F(x, y, z) = 0$, тобто для побудови графіка функції двох змінних $z = f(x, y)$ у вигляді сітки координатних ліній. Такі задачі виникають в багатьох застосуваннях математики.

Щоб побудувати поверхню, потрібно кожній точці $P(x, y, z)$ поверхні поставити у відповідність точку картинної площини. Для цього можна використати матрицю аксонометричного проєктування.

Для задання напрямку вектора проєктування, тобто напрямку нормалі до картинної площини, потрібно задати два кути φ , ψ . Нехай вектор нормалі до картинної площини утворює кут ψ із площиною xu , а проєкція нормалі на площину xu із віссю y – кут φ (рис. 14.1). Тоді напрям проєктування задається вектором

$$\vec{n} = (-\sin\varphi \cos\psi, -\cos\varphi \cos\psi, -\sin\psi), \quad \varphi \in [0, 2\pi], \quad \psi \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right].$$

Для того, щоб виконати ортогональне проєктування поверхні на картинну площину, вісь z' нової системи координат $x'y'z'$ повинна збігатися з напрямом проєктування (рис. 14.1).

Розглянемо перетворення системи координат xuz у систему $x'y'z'$. Це перетворення можна здійснити за два кроки.

1) Поворот системи координат відносно осі z на кут φ (рис. 14.1) у напрямку від y до x (від'ємний напрям). Оскільки система координат xuz лівостороння, то цей поворот задається матрицею

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

2) Поворот одержаної системи координат відносно осі абсцис на кут $(\frac{\pi}{2} - \psi)$ у напрямку від z до y . Матриця повороту має вигляд

$$R_x^-(\frac{\pi}{2} - \psi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\frac{\pi}{2} - \psi) & -\sin(\frac{\pi}{2} - \psi) & 0 \\ 0 & \sin(\frac{\pi}{2} - \psi) & \cos(\frac{\pi}{2} - \psi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \sin \psi & -\cos \psi & 0 \\ 0 & \cos \psi & \sin \psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

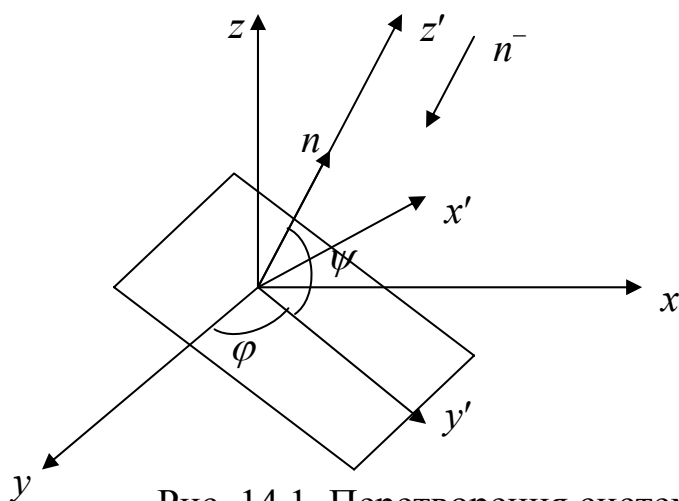


Рис. 14.1. Перетворення системи координат

Перетворення системи координат xuz у $x'y'z'$ задається добутком матриць $R_z^-(\varphi) R_x^-(\frac{\pi}{2} - \psi)$, тобто матрицею

$$A = \begin{pmatrix} \cos \varphi & -\sin \varphi \sin \psi & \sin \varphi \cos \psi & 0 \\ \sin \varphi & \cos \varphi \sin \psi & -\cos \varphi \cos \psi & 0 \\ 0 & \cos \psi & \sin \psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (14.1)$$

Це означає, що точка (x, y, z) з поверхні переходить у точку (x', y') на картинній площині, де

$$\begin{aligned} x' &= x \cos \varphi + y \sin \varphi, \\ y' &= -x \sin \varphi \sin \psi + y \cos \varphi \sin \psi + z \cos \psi. \end{aligned} \quad (14.2)$$

Обернене перетворення задається матрицею $A^{-1} = A^{\text{TP}}$, тобто співвідношеннями

$$\begin{aligned}
x &= x' \cos \varphi - y' \sin \varphi \sin \psi + z' \cos \psi \sin \varphi, \\
y &= x' \sin \varphi + y' \sin \psi \cos \varphi - z' \cos \psi \cos \varphi, \\
z &= y' \cos \psi + z' \sin \psi.
\end{aligned}
\tag{14.3}$$

Здійснивши заміну змінних (14.3) в рівнянні $F(x, y, z) = 0$, одержуємо рівняння поверхні $F(x', y', z') = 0$ у новій системі координат $x'y'z'$, в якій напрям ортогонального проектування вже паралельний до осі z' і проектування ведеться на площину $x'y'$. Надалі координати x' , y' , z' позначатимемо через x , y , z .

Алгоритм поточного горизонту працює в просторі зображення. Головна ідея методу полягає в зведенні тривимірної задачі до двовимірної шляхом перетину вихідної поверхні послідовністю паралельних площин, що мають постійне значення однієї з координат. Іншими словами, для створення зображення поверхні застосовується полігональний спосіб побудови графіка: функція наближається прямокутною матрицею значень функції у вузлах сітки, а сам графік поверхні задається набором ламаних ліній, що відповідають постійним значенням z та x .

Розглянемо спочатку побудову графіка функції у вигляді набору ліній, що відповідають постійним значенням z . Поверхня $F(x, y, z) = 0$ в цьому випадку апроксимується послідовністю кривих $y = f(x, z)$, де $z = \text{const}$.

Для кожного z , що змінюється в циклі від z_1 до z_n , $z_1 < z_n$, перебираємо значення x з деякого діапазону і так знаходимо точки (x, y, z) , що лежать на поверхні. Це означає, що написати програму побудови графіка функції двох змінних без усунення невидимих ліній нескладно, однак таке зображення поверхні буде досить незрозумілим. Тому постає задача побудови такої поверхні, на якій би усувались невидимі лінії.

Шляхом перетину поверхні послідовністю паралельних площин $z = z_i$ ($i = 1, 2, \dots, n$) тривимірну задачу зводимо до двовимірної: поверхня представляється послідовністю ліній $y = f(x, z_i)$, які лежать у паралельних площинах $z = z_i$ і, як наслідок, не перетинаються (рис. 14.2).

Отже, маємо такий алгоритм побудови графіка функції $y = f(x, z)$. Для кожної площини $z = z_i$, у порядку зростання z , починаючи з найближчої до точки спостереження, малюються лінії. При зображенні поточної лінії виводиться лише та її частина, яка не закривається раніше намальованими лініями.

Якщо в деякій площині $z = \text{const}$, при деякому x одержуємо точку y , більшу за всі попередні y , то поточна крива видима в цій точці, інакше – невидима. Для визначення тих частин лінії $y = f(x, z_i)$, які не закриваються раніше намальованими лініями, вводяться в розгляд дві лінії горизонту – верхня та нижня (два масиви, довжина яких дорівнює кількості

точок x у просторі зображення). У цих масивах зберігатимуться поточні значення горизонтів.

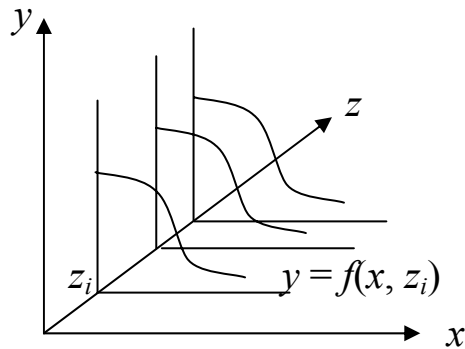


Рис. 14.2. Апроксимація поверхні лініями

На початку лінії горизонту не ініціалізуються. Перша лінія поверхні виводиться повністю, позаяк вона ближче, ніж інші, розміщена до спостерігача, тому закривати її ніщо не може. Після цього дві лінії горизонту ініціалізуються так, що для x , при яких виводиться графік, вони збігаються з першою лінією.

Друга лінія теж повністю виводиться, а лінії горизонту корегуються так: нижня лінія горизонту в усіх своїх точках x прирівнюється до мінімуму серед значень y для двох уже виведених ліній, а верхня лінія горизонту – до максимуму. Область екрана між верхньою та нижньою лінією горизонту є проекцією частини графіка функції $y = f(x, z)$ у смугі $z_1 < z < z_2$.

Для інших ліній $y = f(x, z_i)$ при $i > 2$ ті частини ліній, які при проектуванні попадають в область між двома лініями горизонту, невидимі (при цьому способі проектування). Ця область заборонена для подальшого втручання. Наступна лінія буде малюватися лише для тих x , для яких її проекція лежить зовні області, що задається двома лініями горизонту. Тобто алгоритм тепер такий: якщо при деякому x значення y на поточній кривій більше за відповідне значення верхнього горизонту, то точка y видима, якщо менше за значення нижнього горизонту, то точка y теж видима, інакше – невидима.

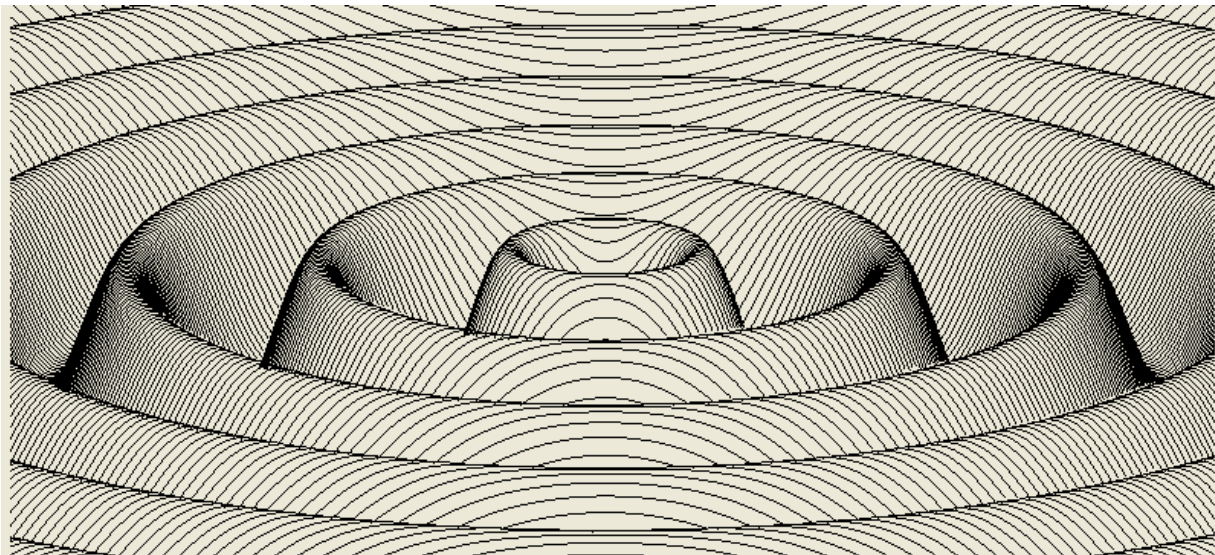
Позначимо проекцію лінії $y = f(x, z_i)$ на картинну площину через $y = y_i(x)$, де (x, y) – координати точок лінії на картинній площині, тоді контурні лінії горизонтів $y_k^{\max}(x)$ і $y_k^{\min}(x)$ на k -му кроці визначаються співвідношеннями:

$$y_k^{\max}(x) = \max_{1 \leq i \leq k-1} y_i(x), \quad y_k^{\min}(x) = \min_{1 \leq i \leq k-1} y_i(x).$$

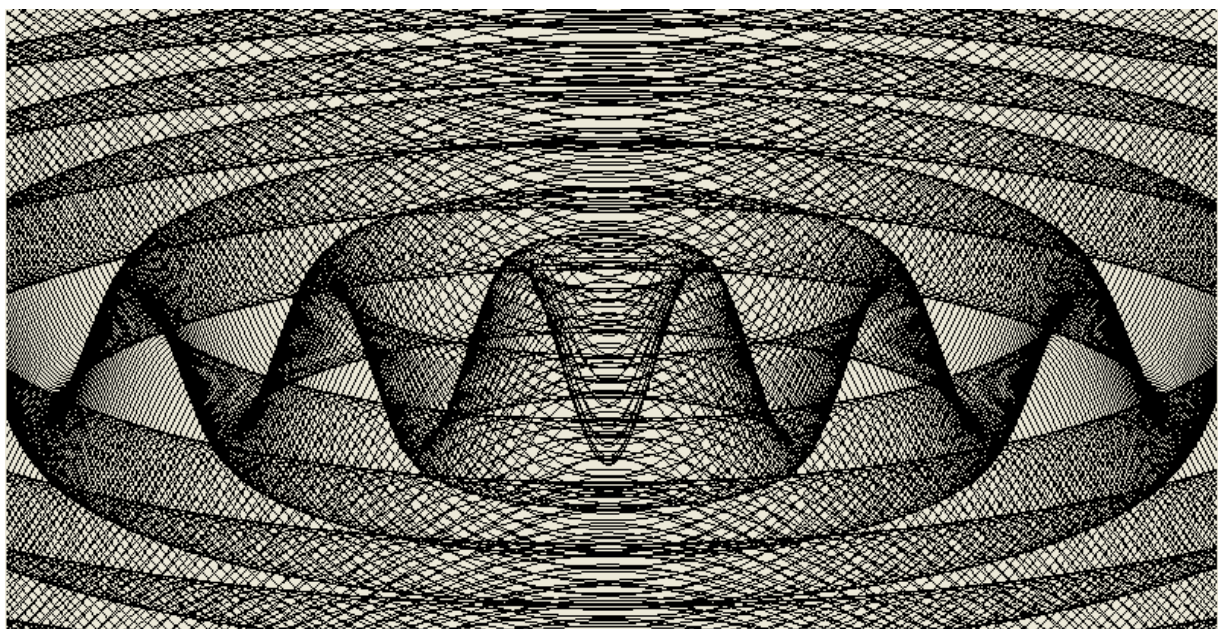
На k -му кроці малюються тільки ті частини лінії $y = y_k(x)$, які знаходяться вище лінії $y_k^{\max}(x)$, або нижче лінії $y_k^{\min}(x)$. Якщо $y_k^{\min}(x) \leq y_k(x) \leq y_k^{\max}(x)$, то точка на поверхні невидима і вона не відображається. Такий

алгоритм і називається *методом поточного горизонту*. На рис. 14.3, *а* наведено приклад поверхні побудованої цим методом. Для порівняння наведено зображення без усунення невидимих ліній (рис. 14.3, *б*)

Можливі різні способи побудови ліній горизонту, але однією з найпростіших і найефективніших реалізацій цього методу є растрова реалізація, при якій кожна лінія горизонту задається набором значень із кроком в один піксель.



а



б

Рис. 14.3. Поверхня $z = f(x, y) = \frac{\cos(b\sqrt{x^2 + y^2})}{1 + c\sqrt{x^2 + y^2}}$.

Зауваження 1. Якщо при деякому x значення $y(x)$ невідоме, то для визначення $y(x)$ потрібно використовувати інтерполяцію (можна лінійну). У випадках, коли при x_n точка y_n видима, а при x_{n+k} точка y_{n+k} невидима, необхідно визначити частину видимої лінії, що проходить через точки (x_n, y_n) і (x_{n+k}, y_{n+k}) , тобто необхідно знати точки перетину сегмента поточної лінії і ліній горизонту (рис. 14.4).

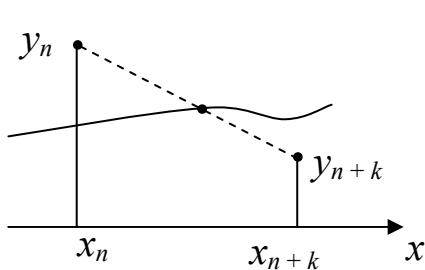


Рис. 14.4. Визначення видимої частини лінії

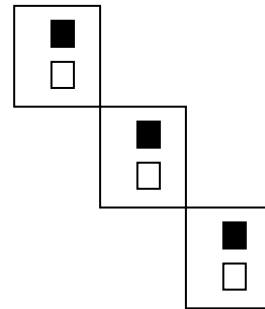


Рис. 14.5. Випадні піксели

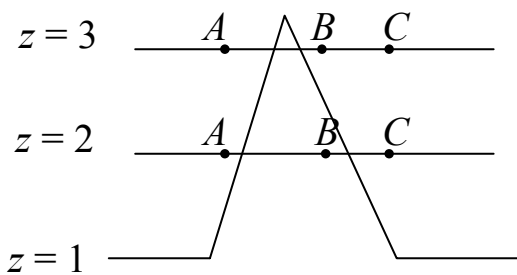


Рис. 14.6. Дефект алгоритму при наявності піків

Зауваження 2. Випадок відрізків ламаної з кутовим коефіцієнтом, за модулем, більшим одиниці, вимагає спеціальної обробки відрізків, щоб не з'явилися випадні піксели (рис. 14.5).

Зауваження 3. Якщо функція задана малою кількістю точок і має досить гострі піки, то наведений алгоритм може дати некоректні результати.

Наприклад, розглянемо випадок ліній, що зображені на рис. 14.6. Оскільки лінія $z = 3$ у точках A та B видима, то можна зробити висновок, що відрізок AB ($z = 3$) видимий, хоча це не так, бо цей відрізок частково перекривається піком лінії $z = 1$.

Для лінії $z = 2$ будуть знаходитися видимі частини відрізків AB та BC , позаяк один із кінців відрізка видимий, а другий – невидимий. Для лінії $z=3$ алгоритм працює неправильно, для лінії $z = 2$ – правильно.

Наведений алгоритм дає дефект, коли поточна крива може з'явитися зліва або справа від попередніх кривих. Наприклад, на рис. 14.7 криві $z = 1$, $z = 2$ видимі, але при $x \in [2, 5)$ верхнім горизонтом є крива $z = 2$ і для кривої $z = 3$ координати y менші, ніж значення верхнього горизонту, отже, видимі. Тобто при $x = 4$ алгоритм визначає криву $z = 3$ видимою, тому виникають зазубрини. Цього дефекту можна уникнути шляхом введення в масиви верхнього та нижнього горизонтів ординат, що відпові-

дають штриховим лініям, які утворюються хибними бічними ребрами.

Введення лівого хибного ребра P_1P_3 для випадку рис. 14.7 зробить лінію $z = 3$ при $x \in [3, 5)$ невидимою. Хибне ребро не вводиться, якщо крайні точки на кривих $z = 1, z = 2$ мають однакові значення координати x . Аналогічно вводяться у розгляд праві хибні бічні ребра.

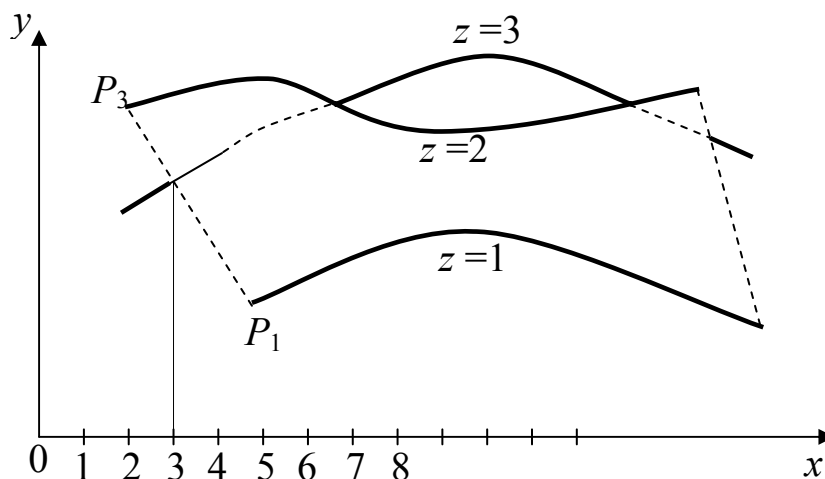


Рис. 14.7. Введення хибних бічних ребер

14.3. Алгоритм Робертса

Одним із перших алгоритмів усунення невидимих ліній був алгоритм Робертса (1963 р.). Цей алгоритм працює безпосередньо в просторі самих об'єктів і застосовується у випадку сцен, що складаються з многогранних об'єктів, кожна грань яких є опуклим многокутником, тобто для опуклих об'єктів. Неопуклі тіла потрібно розбивати на опуклі частини. Крім цього, грані об'єктів не повинні між собою перетинатися.

Основна ідея цього алгоритму полягає в тому, що на початку з кожного об'єкта сцени усуваються ті ребра та грані, які закриваються самим об'єктом (нелицьові грані). Потім кожне з ребер, що залишилося, перевіряється на видимість по відношенню до лицьових граней. На початку програма повинна прочитати необхідні дані із файла.

Вхідні дані. Об'єкт задається списком вершин, тобто номерами вершин та їхніми координатами, і списком ребер об'єкта (масив RIB) у світовій системі координат. Окрім цього, для побудови перспективної проекції, потрібно задати координати точки E спостерігача (рис. 14.8).

Вважаємо, що центр об'єкта знаходиться в початку світової системи координат, інакше потрібно виконати переміщення об'єкта. Щоб одержати координати вершин у видовій системі координат, необхідно здійснити видове перетворення. Видові координати вершин зберігатимемо в масиві VER . z -координати цих вершин ($VER[i].z$) повинні бути більшими

за нуль або дорівнювати нулю. Якщо ці значення менші за нуль, то необхідно зупинити програму. Надалі оперуватимемо тільки координатами у видовій системі координат.

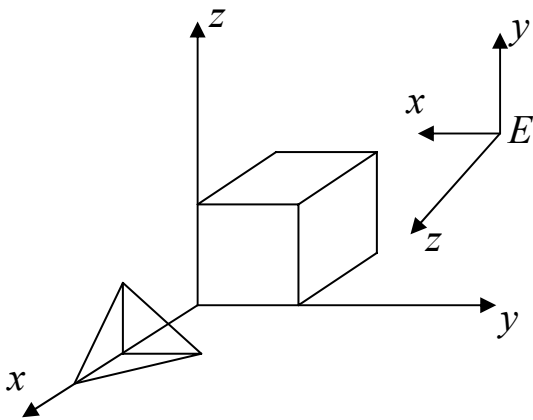


Рис. 14.8. Видові координати

зовні.

Список трикутників вигідно запам'ятати в деякому масиві – назовемо його *TRIANGLE*, кожний елемент якого матиме структуру

$$A \ B \ C \quad a \ b \ c \ h,$$

де A, B, C – номери точок, що утворюють трикутник, а числа a, b, c, h – коефіцієнти рівняння площини $ax + by + cz = h$, в якій розміщений цей трикутник.

Коефіцієнти a, b, c, h обчислюються з рівняння площини

$$\begin{vmatrix} x & y & z & 1 \\ x_A & y_A & z_A & 1 \\ x_B & y_B & z_B & 1 \\ x_C & y_C & z_C & 1 \end{vmatrix} = 0.$$

Тобто

$$a = \begin{vmatrix} y_A & z_A & 1 \\ y_B & z_B & 1 \\ y_C & z_C & 1 \end{vmatrix}, \quad b = - \begin{vmatrix} x_A & z_A & 1 \\ x_B & z_B & 1 \\ x_C & z_C & 1 \end{vmatrix}, \quad c = \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix}, \quad h = \begin{vmatrix} x_A & y_A & z_A \\ x_B & y_B & z_B \\ x_C & y_C & z_C \end{vmatrix}.$$

Розглянемо проєкції точок A, B, C на картинну площину. Відомо, що координати перспективної проєкції точки A на площину $z = d$ знаходяться за формулами

$$x'_A = \frac{dx_A}{z_A}, \quad y'_A = \frac{dy_A}{z_A}.$$

Напрямок обходу точок-проєкцій може бути встановлений на основі аналізу визначника

$$D = \begin{vmatrix} x'_A & y'_A & 1 \\ x'_B & y'_B & 1 \\ x'_C & y'_C & 1 \end{vmatrix}.$$

Оскільки

$$D = \begin{vmatrix} \frac{dx_A}{z_A} & \frac{dy_A}{z_A} & 1 \\ \frac{dx_B}{z_B} & \frac{dy_B}{z_B} & 1 \\ \frac{dx_C}{z_C} & \frac{dy_C}{z_C} & 1 \end{vmatrix} = d^2 \begin{vmatrix} x_A & y_A & z_A \\ x_B & y_B & z_B \\ x_C & y_C & z_C \end{vmatrix} / (z_A z_B z_C) = \frac{hd^2}{z_A z_B z_C}, z_A, z_B, z_C > 0,$$

то D і h однакового знаку. Тоді можливі такі випадки:

1) якщо $h = 0$, то площина ABC проходить через точку E і ΔABC нічого не закриває;

2) якщо $h < 0$ (рівносильно $D < 0$), то ΔABC – задній, тому що при $D < 0$ маємо обхід проєкцій точок A, B, C за годинниковою стрілкою (напрям обходу змінився). Задні трикутники теж ігноруються, оскільки вони закриваються видимими гранями. Зауважимо, що у випадку одного опуклого многогранника виділення задніх трикутників повністю розв'язує задачу усунення невидимих граней. Для складних графічних сцен цей факт часто використовується для усунення нелицьових граней перед

застосуванням інших алгоритмів усунення невидимих ліній та граней;

3) якщо $h > 0$, то ΔABC записуємо в масив $TRIANGLE$, тому що він належить до передніх граней об'єкта.

Розробимо алгоритм визначення невидимих ліній, тобто алгоритм, який викреслюватиме тільки видимі частини відрізків PQ зі списку ребер RIB .

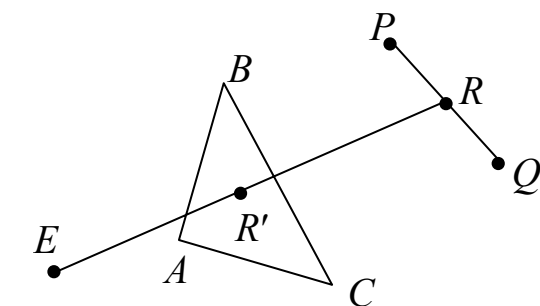


Рис. 14.9. Видимість точки R

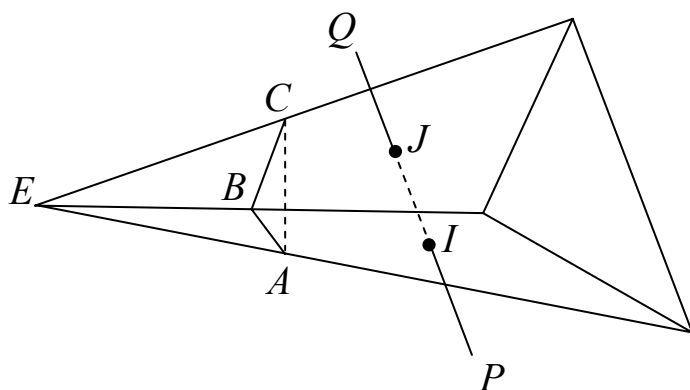
Для цього потрібно виконати перевірку для кожного ΔABC зі списку в масиві $TRIANGLE$, щоб з'ясувати, чи буде ΔABC закривати все ребро PQ , чи його частину.

Нехай E – точка зору. Кажуть, що ΔABC загороджує точку $R \in PQ$, якщо відрізок ER перетинає ΔABC у внутрішній точці R' трикутника і у внутрішній точці R відрізка PQ (рис. 14.9).

Точки границі ΔABC і граничні точки P, Q не вважаємо внутрішніми. Отже, точка R не загороджується ΔABC , якщо

- 1) точка $R' \notin \Delta ABC$,
- 2) точка $R' \in$ границі ΔABC .

Якщо точка R не загороджується ΔABC , то точка R називається *видимою* відносно ΔABC . Якщо із відрізка PQ відносно ΔABC видно скінченну кількість точок, то будемо говорити, що ΔABC загороджує ребро PQ . ΔABC частково закриває PQ , якщо він закриває нескінченно



багато точок відрізка PQ і водночас нескінченно багато точок є видимими відносно ΔABC .

Якщо ΔABC закриває відрізок PQ , то PQ закритий і не повинен зображатися. При цьому немає необхідності виконувати перевірку для решти трикутників.

Рис. 14.10. Піраміда видимості

Розглянемо алгоритм визначення видимості відрізка PQ відносно ΔABC . Нагадаємо, що ми вже маємо видові координати для п'яти точок P, Q, A, B, C .

Побудуємо нескінченну піраміду, вершина якої знаходиться в точці E , а бічні грані проходять через сторони трикутника ABC (рис. 14.10).

Все, що знаходиться всередині піраміди позаду ΔABC , є невидимим, а всі точки, що лежать зовні піраміди або в піраміді перед ΔABC , видимі відносно ΔABC .

Розглянемо питання перетину відрізка PQ із гранями піраміди. Рівняння прямої PQ запишемо у вигляді

$$\begin{aligned} x &= x_P + \lambda(x_Q - x_P), \\ y &= y_P + \lambda(y_Q - y_P), \\ z &= z_P + \lambda(z_Q - z_P). \end{aligned} \tag{14.4}$$

Точка I належить площині AEB , рівняння якої має вигляд

$$\begin{vmatrix} x & y & z \\ x_A & y_A & z_A \\ x_B & y_B & z_B \end{vmatrix} = 0,$$

або

$$C_1x + C_2y + C_3z = 0, \tag{14.5}$$

де $C_1 = y_Az_B - z_Ay_B$, $C_2 = x_Bz_A - x_Az_B$, $C_3 = x_Ay_B - y_Ax_B$.

Підставляючи (14.5) в (14.4), знаходимо

$$\lambda = - \frac{C_1 x_P + C_2 y_P + C_3 z_P}{C_1(x_Q - x_P) + C_2(y_Q - y_P) + C_3(z_Q - z_P)}. \quad (14.6)$$

Знаючи λ , з (14.4) можна знайти координати точки I (точки перетину відрізка PQ з гранню AEB). Якщо $\lambda \in [0, 1]$, то точка I лежить на відрізку PQ , для інших значень λ точка I не належить відрізку PQ .

Але не можна стверджувати, що якщо $\lambda \in [0, 1]$, то відрізок PQ перетинає піраміду (справедливе лише обернене твердження), оскільки перетин відрізка PQ може відбутися з площиною, в якій лежить грань AEB .

Щоб переконатися, що точка I належить грані AEB , проведемо площину PEQ і дізнаємось, чи ця площина перетинає відрізок AB .

Аналогічно сказаному вище, запишемо рівняння прямої AB

$$\begin{aligned} x &= x_A + \mu(x_B - x_A), \\ y &= y_A + \mu(y_B - y_A), \\ z &= z_A + \mu(z_B - z_A) \end{aligned} \quad (14.7)$$

та рівняння площини PEQ

$$K_1 x + K_2 y + K_3 z = 0, \quad (14.8)$$

де

$$K_1 = y_P z_Q - y_Q z_P, \quad K_2 = x_Q z_P - x_P z_Q, \quad K_3 = x_P y_Q - x_Q y_P.$$

Із рівнянь (14.7), (14.8) знаходимо

$$\mu = - \frac{K_1 x_A + K_2 y_A + K_3 z_A}{K_1(x_B - x_A) + K_2(y_B - y_A) + K_3(z_B - z_A)}. \quad (14.9)$$

Отже, відрізок PQ і грань AEB піраміди мають спільну точку I тоді і тільки тоді, коли

$$0 \leq \lambda \leq 1, \quad 0 \leq \mu \leq 1.$$

Аналогічно досліджується перетин відрізка PQ з іншими гранями BEC , AEC .

Відрізок PQ може не мати або мати один чи два перетини з пірамідою. Відповідно точки P , Q лежатимуть всередині, зовні піраміди або безпосередньо на піраміді.

Такі перевірки потрібно виконувати для кожної пари відрізків-трикутник, а це вимагає великих обчислювальних затрат. З метою зменшення кількості перевірок бажано починати алгоритм з аналізу таких випадків, які не вимагають значних затрат машинного часу.

Опишемо цей алгоритм у вигляді тестів. При успішному виконанні одного з тестів наступні тести не перевіряються.

На початку зі списку ребер відкидаємо всі ребра, до яких прилягають тільки задні грані (жодне з таких ребер апріорі невидиме). Далі перевіряємо на видимість кожне ребро, що залишилося в списку, відносно усіх передніх граней об'єктів.

Тест 1. Якщо точки P та Q лежать перед площиною ABC або на ній (але не позаду), то відрізок PQ – видимий. Цей тест виконується, якщо справедливі нерівності

$$ax_P + by_P + cz_P \leq h, \quad ax_Q + by_Q + cz_Q \leq h.$$

Тест 2. Якщо відрізок PQ лежить зовні піраміди, то він видимий. Для перевірки цієї умови підставляємо значення координат точок A, B, C у ліву частину рівняння площини PEQ (14.8). Якщо всі три обчислені значення (для точок A, B, C) мають однаковий знак (всі додатні або всі від’ємні), то всі точки A, B, C лежать по один бік від площини PEQ (рис. 14.11, *a*), отже, ребро PQ – видиме.

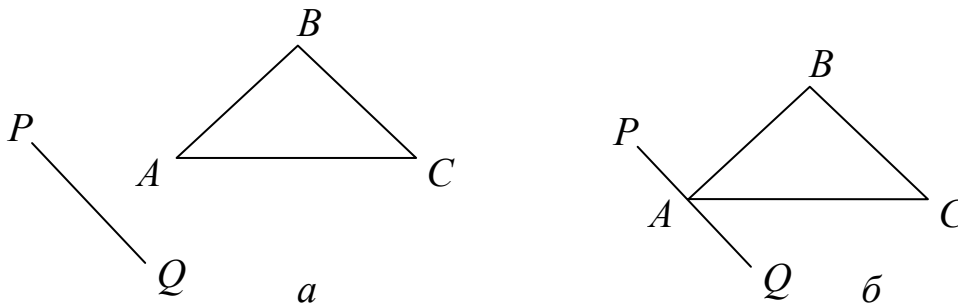


Рис. 14.11. Тест 2: *a* –модуль суми знаків = 3; *б* –модуль суми знаків = 2

Зазначимо, що умова на знаки може бути послаблена: одне зі значень може дорівнювати нулю (рис. 14.11, *б*). Тобто, якщо припустити, що кожен зі знаків позначається числом $-1, 0, 1$, то потрібно перевіряти, чи сума знаків дорівнюватиме одному з чисел $-2, -3, 2, 3$.

Тест 3. Рівняння (14.5) визначає площину AEB . Якщо ліва частина цього рівняння при підстановці точок P і C набуває різних знаків, то точки P і C лежать по різні боки від цієї площини. Отже, точка P лежить зовні піраміди за площиною AEB (рис. 14.12). Запам’ятаємо цю інформацію в логічній змінній P_{out} (точка P зовні).

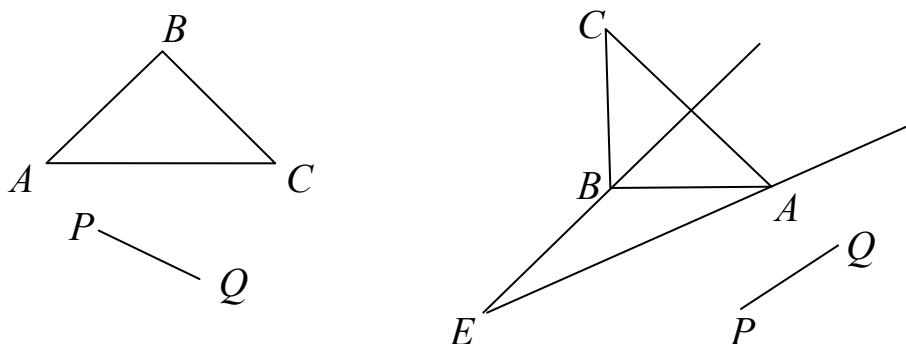


Рис. 14.12. Відрізок PQ видимий

Аналогічно для точки Q вводимо змінну Q_{out} . Якщо дві точки P, Q лежать за площиною AEB , то відрізок PQ видимий (рис. 14.13, *a*).

Відрізок PQ буде видимим, якщо одна з точок лежить у площині AEB , а інша точка видима (рис. 14.13, б).

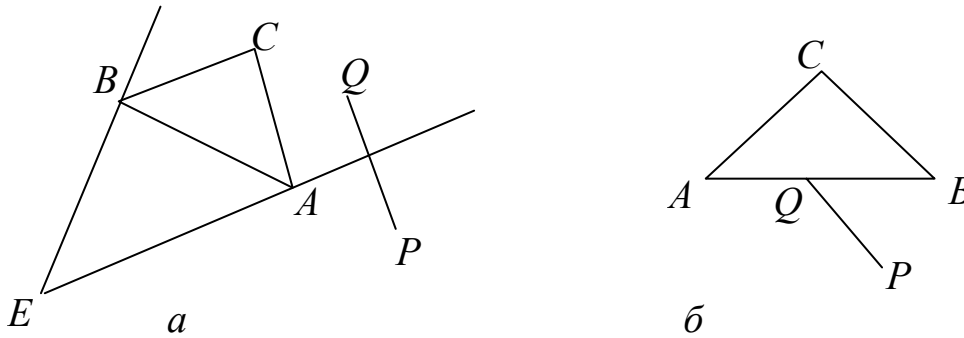


Рис. 14.13. Тест3: a – точки P, Q видимі; $б$ – точка P видима, Q належить площині AEB

Якщо не встановлена видимість точок P, Q по відношенню до площини AEB , то далі, аналогічно викладеному вище, вивчаємо видимість відрізка PQ по відношенню до площин AEC і BEC .

Тест 4. Якщо точки P і Q знаходяться всередині піраміди і попередні тести не виконалися, то відрізок PQ лежить позаду $\triangle ABC$ (рис. 14.14, a), а отже, є невидимим.

При виконанні цього тесту може зустрітися особлива ситуація, коли відрізок PQ лежить на піраміді позаду відрізка AB (рис. 14.14, б). Якщо відрізок AB є діагоналлю грані, то грань закриває відрізок PQ і він не виводиться. Якщо AB – ребро, то відрізок PQ теж не потрібно зображати, оскільки зображення відрізка PQ збігається із зображенням AB , а зображати збіжні відрізки двічі немає змісту.

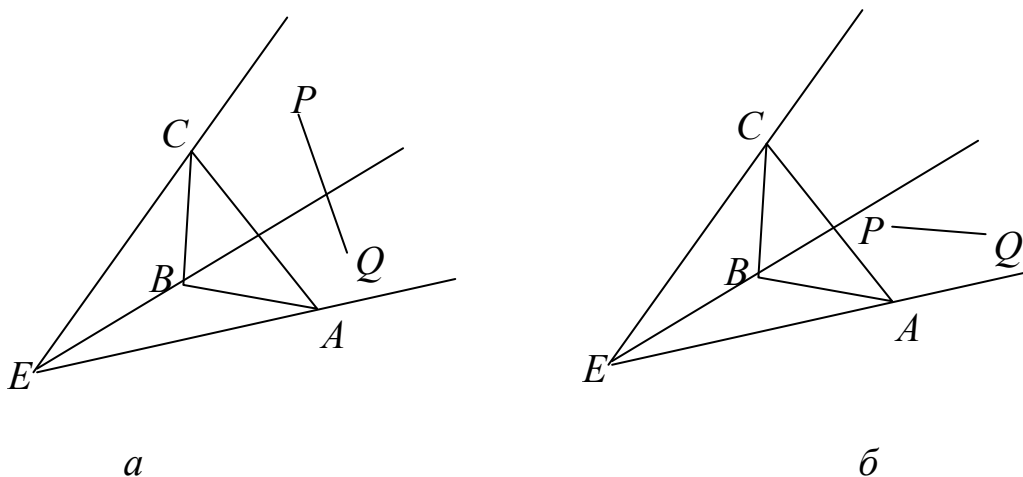


Рис. 14.14. Тест 4: a – PQ позаду $\triangle ABC$; $б$ – PQ повністю закривається відрізком AB

Тест 5. Знайдемо точку перетину прямої лінії PQ із гранями піраміди AEB, BEC, AEC . Обчислимо значення параметрів λ для точки

перетину прямої лінії PQ із площиною AEB за формулою (14.6) і μ для точки перетину прямої AB з площиною PEQ за формулою (14.9).

Аналогічно знаходимо параметри λ , μ для двох інших граней. Отже матимемо три пари значень (λ_i, μ_i) , $i = 1, 2, 3$. Після цього знайдемо мінімальне і максимальне значення серед λ_i , що задовольняють умови

$$0 \leq \lambda_i \leq 1, \quad 0 \leq \mu_i \leq 1, \quad i = 1, 2, 3.$$

Ці значення позначимо λ_{\min} , λ_{\max} .

Якщо при деякому $0 \leq \lambda_i \leq 1$ знайдена з формули (14.4) точка I (точка перетину PQ з гранню піраміди) лежить перед $\triangle ABC$, то PQ – видимий, оскільки PQ не може проходити через внутрішні точки $\triangle ABC$.

Тест 6. Цей тест виконується тоді, коли всі попередні тести не виконалися, тобто коли відрізок PQ перетинає піраміду позаду $\triangle ABC$.

Якщо $\lambda_{\min} \neq \lambda_{\max}$, то ці значення дають можливість знайти, відповідно, дві точки I та J і визначити невидимий відрізок IJ (рис. 14.15).

Якщо точка P знаходиться зовні піраміди, то відрізок PI видимий, якщо точка Q знаходиться зовні піраміди, то відрізок JQ видимий.

Якщо $\lambda_{\min} = \lambda_{\max}$ (точки I, J збігаються), то P і Q одночасно не знаходяться зовні піраміди, тому видимим буде або відрізок PI , або IQ , у залежності від значень P_{out} , Q_{out} .

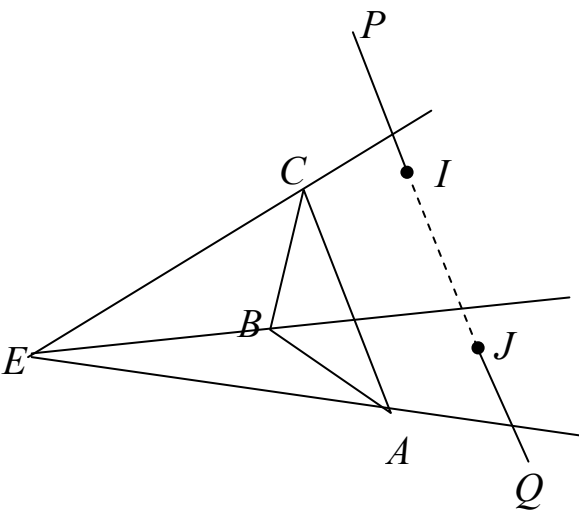


Рис. 14.15. IJ – невидима частина відрізка PQ

У цьому тесті можуть з'явитися два нових відрізки PI і JQ , які є видимими відносно $\triangle ABC$, тому надалі необхідно двічі звертатися до цього ж алгоритму, щоб визначити видимість цих нових відрізків відносно решти трикутників зі списку *TRIANGLE*.

Зазначимо, що складність цього алгоритму становить порядок n^2 , де n – кількість відрізків, тому, з метою зменшення часу роботи цього алгоритму розроблено багато його модифікацій.

Зауваження 4. Через складність обчислень інколи доцільно використовувати тип *double* замість *float*, тобто числа з плаваючою крапкою подвійної точності. Крім цього, всі числа в ЕОМ подаються зі скінченною точністю, тому інколи одну перевірку потрібно замінити іншою. Наприклад, замість умовного оператора $x = a$ (для чисел із плаваючою крапкою) записують оператор $\text{abs}(x - a) \leq \text{epsilon}$, де *epsilon* – мала додатна величина.

Нині в зв'язку з широким розповсюдженням растрових пристроїв найбільш розповсюджені алгоритми, які працюють в просторі зображень, оскільки вони вимагають менше обчислювальних операцій (складність цих алгоритмів пропорційна добутку кількості об'єктів на роздільну здатність екрана візуалізації). Це знизило інтерес до алгоритму Робертса, але його математичні методи застосовуються в багатьох задачах комп'ютерної графіки.

14.4. Метод Z-буфера

Одним із найпростіших методів усунення невидимих граней і поверхонь є метод Z-буфера (буфера глибини). Метод Z-буфера використовується для зображення складних сцен із застосуванням ортогонального проектування на картинну площину. Крім цього, цей метод тривіально розв'язує задачу перетину складних поверхонь. Тут для кожного пікселя картинної площини визначається грань об'єкта, яка лежить далі від картинної площини вздовж напрямку проектування. По суті, алгоритм зводиться до знаходження по x та y максимального значення функції $z(x,y)$.

Кожному пікселю (x, y) картинної площини поставимо у відповідність колір $c(x, y)$, який зберігатимемо у буфері кадру, та координату $z(x, y)$ – глибину кожного видимого пікселя картинної площини (відстань точок просторових об'єктів до картинної площини вздовж напрямку проектування), яку зберігатимемо в Z-буфері. Тобто, буфер кадру служить для запам'ятовування інтенсивності кожного пікселя в просторі зображень, а Z-буфер – для запам'ятовування координати z кожного видимого пікселя в просторі зображень.

В процесі роботи значення z кожного нового пікселя, який потрібно занести в буфер кадру, порівнюється з глибиною пікселя, який уже занесений в Z-буфер. Якщо це порівняння показує, що новий піксель розміщений попереду пікселя, що знаходиться в буфері кадру, то новий піксель заноситься в буфер кадру і Z-буфер корегується новим значенням z .

Формальний запис алгоритму має наступний вигляд.

1. Масив глибин (Z-буфер) ініціалізується максимальним значенням z (значенням глибини фону), а буфер кадру заповнюється значеннями атрибутів (кольором) фону.
2. Для виведення на картинну площину довільної грані об'єкта будують растрову розгорнуту проекцію цієї грані, тобто перетворюють трикутні грані у набори пікселів. Цей процес називається *растеризацією*.
3. Для кожного пікселя (x_i, y_j) цієї розгортки обчислюють глибину z_{ij} .
4. Порівнюють значення z_{ij} з відповідним $Z_{ij}^{буф}$, що зберігається в

Z-буфері. Якщо $z_{ij} < Z_{ij}^{буф}$ (піксель (x_i, y_j) – видимий, тобто знаходиться попереду відповідного пікселя з буфера кадру), то $Z_{ij}^{буф}$ приймають значення z_{ij} , заносючи значення z_{ij} нового пікселя до відповідної позиції Z-буфера, обчислюють колір поверхні в цій точці і атрибути пікселя (x_i, y_j) записують до буфера кадру. Якщо порівняння дає протилежний результат, то ніяких дій не виконується (*Z-буфер зберігає координати z_{ij} кожного пікселя, видимого на поточному кроці аналізу зображення*).

- Беруть наступну необроблену грань, причому порядок немає ніякого значення та переходять до кроку 2.

Досить ефективним у цьому методі є суміщення побудови растрової розгортки з виведенням у Z-буфер, при цьому для обчислення глибини пікселів можуть застосовуватися інкрементні методи.

Грань об'єкта зображається стрічка за стрічкою; для знаходження необхідних значень використовується лінійна інтерполяція (рис. 14.16).

$$x_a = x_1 + (x_2 - x_1) \frac{y - y_1}{y_2 - y_1},$$

$$x_b = x_1 + (x_3 - x_1) \frac{y - y_1}{y_3 - y_1},$$

$$z_a = z_1 + (z_2 - z_1) \frac{y - y_1}{y_2 - y_1},$$

$$z_b = z_1 + (z_3 - z_1) \frac{y - y_1}{y_3 - y_1},$$

$$z = z_a + (z_b - z_a) \frac{x - x_a}{x_b - x_a}.$$

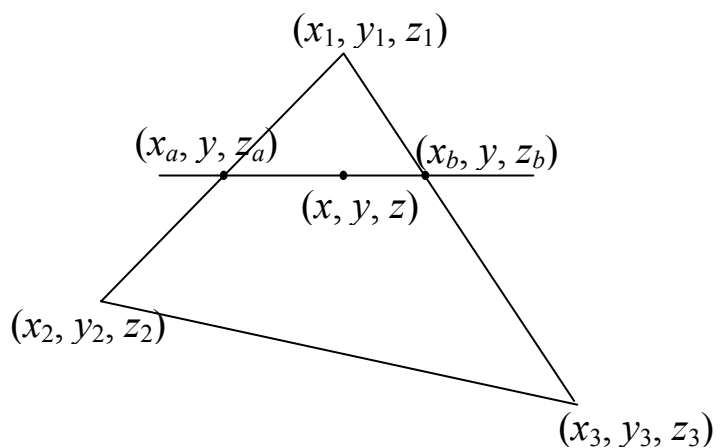


Рис. 14.16. Лінійна інтерполяція грані об'єкта

Метод Z-буфера працює виключно в просторі картинної площини, тобто в просторі зображень. Порядок виведення граней на екран не відіграє жодної ролі, тому їх не потрібно сортувати за глибиною, отже, економиться обчислювальний час. Час роботи цього алгоритму лінійно залежить від кількості точок растра та кількості граней.

Метод Z-буфера вимагає великого обсягу пам'яті, оскільки z -координату необхідно обробляти з більшою точністю, ніж координатну інформацію на площині $xу$. Наприклад, буфер кадру розмірності $512 \times 512 \times 24$ бітів у комбінації з Z-буфером розмірності $512 \times 512 \times 20$ бітів вимагає майже 1,5 Мб

пам'яті. Для економії пам'яті можна виводити не все зображення відразу, а малювати його частинами. Для цього картинна площина розбивається на частини (частіше – на горизонтальні смуги) і кожна така частина обробляється незалежно.

Метод Z-буфера зручний для апаратної реалізації через простоту алгоритму і тому більшість сучасних комп'ютерів містить у собі графічні плати з апаратною реалізацією методу Z-буфера, включаючи апаратну побудову растрової розгортки граней разом із зафарбуванням методом Гуро. Такі відеокарти забезпечують високу швидкість рендерінгу (близько декількох мільйонів трикутників за секунду), оскільки вони часто мають власну пам'ять для Z-буфера, доступ до якої здійснюється швидше, ніж до оперативної пам'яті. Головною характеристикою Z-буфера є його роздільна здатність. Чим вища роздільна здатність, тим вища дискретність z-координати і тим точніше виконується рендерінг віддалених об'єктів. Сучасні відеокарти мають 32-розрядний буфер.

Недоліком методу Z-буфера, попри великий обсяг пам'яті, є виведення всіх граней і розрахунок кольору для кожного пікселя різних граней, що накривають даний піксель. При використанні складних моделей освітлення і текстур ці обчислення можуть вимагати багато часових затрат. Тому існують декілька модифікацій алгоритму Z-буфера (наприклад, метод ієрархічного Z-буфера), які дозволяють суттєво скоротити кількість задіяних граней.

Зазначимо, що, якщо в алгоритмі Z-буфера перевірку нерівності $z_{ij} < Z_{ij}^{\text{буф}}$ замінити порівнянням $z^{\text{неп}} \leq z_{ij} < Z_{ij}^{\text{буф}}$, де $z^{\text{неп}}$ – глибина шуканого перетину, то автоматично можна одержати зображення перетину поверхні площиною $z = z^{\text{неп}}$.

14.5. Огляд деяких інших методів

14.5.1. Метод відсікання нелицьових граней

Розглянемо многогранник, для кожної грані якого заданий одиничний вектор зовнішньої нормалі \mathbf{n} (рис. 14.17).

Якщо вектор зовнішньої нормалі \mathbf{n} грані утворює з вектором \mathbf{l} , що задає напрям проектування, тупий кут, тобто вектор нормалі направлений в бік спостерігача, то ця грань видима (рис. 14.18). Така грань називається *лицьовою*. Якщо відповідний кут гострий, то грань невидима. При паралельному проектуванні умову видимості грані можна записати з допомогою скалярного добутку у вигляді нерівності $(\mathbf{n} \cdot \mathbf{l}) < 0$ (рис. 14.18, а).

При центральному проектуванні з центром у точці C вектор проектування для точки P визначається зі співвідношення $\mathbf{l} = P - C$. Тоді, щоб визначити, чи грань є лицьовою, достатньо взяти будь-яку точку P

цієї грані і перевірити виконання умови $(\mathbf{n} \cdot \mathbf{l}) < 0$ (рис. 14.18, б).

Для опуклих многогранників цей метод дає повний розв'язок задачі усунення невидимих граней (достатньо видалити всі неліцеві грані). Для неопуклих многогранників додатково аналізують видимість лицевих граней.

Зауваження 5. Зовнішні нормалі граней можна знаходити як векторний добуток відповідних векторів, що лежать у цій грані.

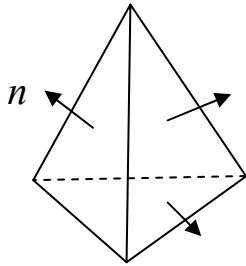


Рис. 14.17. Зовнішні нормалі многогранника

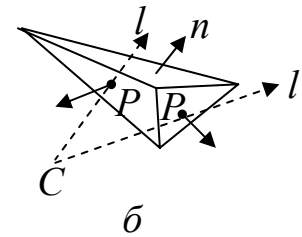
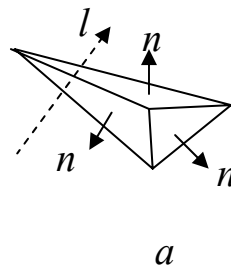


Рис. 14.18. Визначення видимості грані при проектуванні: а – паралельному; б – центральному

14.5.2. Алгоритм розбиття картинної площини Варнока

Алгоритм Варнока ґрунтується на тому факті, що для обробки областей із малою кількістю інформації докладається мало зусиль і, навпаки, для областей з великим інформаційним змістом витрачаються значні ресурси.

Цей алгоритм працює в просторі зображення. Тут використовується поняття активного вікна – геометричної області, яку можна пересувати вздовж картинної площини та розбивати її на вікна меншого розміру.

Вміст активного (прямокутника) вікна аналізують на взаємне розташування з проекціями елементів 3D-сцени (найчастіше з многокутниками, що є проекціями граней многогранників). Процедура прийняття рішення застосовується в кожній малій області. Якщо вікно порожнє або його вміст достатньо простий, то відбувається візуалізація вікна. Якщо ця умова не виконується, то вікно розбивається на частини доти, поки вміст вікна не стане достатньо простим. По мірі того, як розміри вікна зменшуються, кожену таку область перетинає все менша і менша кількість многокутників. При такому підході робота ведеться в просторі зображення. При цьому ефективним є використання списку тих об'єктів, проекції яких перетинають дане вікно.

Існує багато реалізацій алгоритму Варнока. Конкретна реалізація залежить від складності вікна і критерію визначення простоти вмісту вікна. В оригінальній версії алгоритму вікно зі складним зображенням розбивається на чотири вікна.

Алгоритм Варнока працює за такими тестами :

- ✓ якщо вікно повністю покрите проекцією найближчого до спостерігача многокутника (многокутник охоплює вікно), то вікно зафарбовується кольором проекції цього многокутника (рис. 14.19, а);
- ✓ якщо до вікна не потрапила жодна з проекцій елементів сцени (всі многокутники сцени зовнішні відносно вікна), то його зафарбовують кольором фону (рис. 14.19, б);
- ✓ якщо існує єдиний внутрішній (многокутник повністю знаходиться у вікні) або єдиний перетинаючий многокутник, то все вікно зафарбовується кольором фону, а потім частина вікна, яка відповідає многокутнику, зафарбовується кольором многокутника (рис. 14.19, в);
- ✓ якщо не виконалась жодна з умов, то вікно розбивається на чотири частини та для кожної з них повторюються попередні тести (рис. 14.19, г). Подрібнення вікна можна виконувати до розміру пікселя.

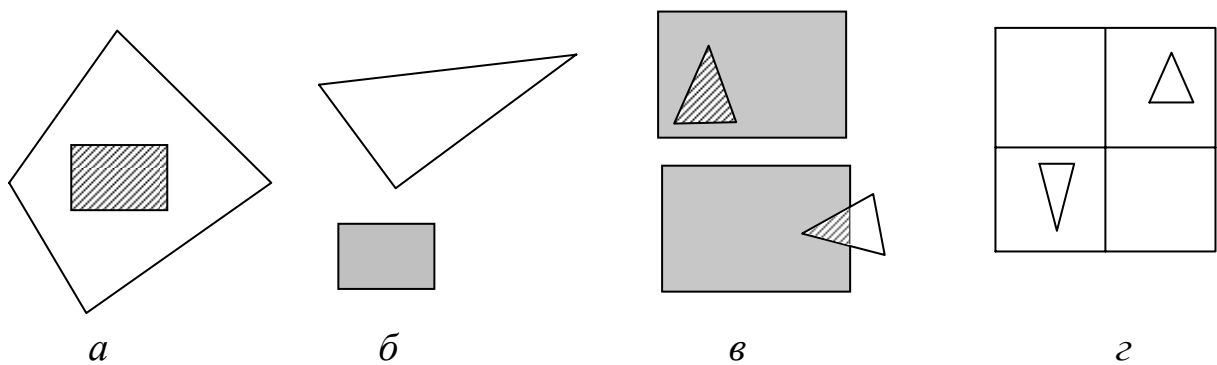


Рис. 14.19. Розміщення вікна та проекції грані

14.5.3. Метод сортування за глибиною. Алгоритм художника

Як художник спочатку малює більш далекі об'єкти, а потім поверх них більш близькі, так і метод сортування за глибиною спочатку впорядковує лицьові грані з наближенням до спостерігача, а потім виводить їх у цьому ж порядку. Спочатку виводяться більш далекі грані, а потім більш близькі, тобто та грань, яка при проектуванні може закривати

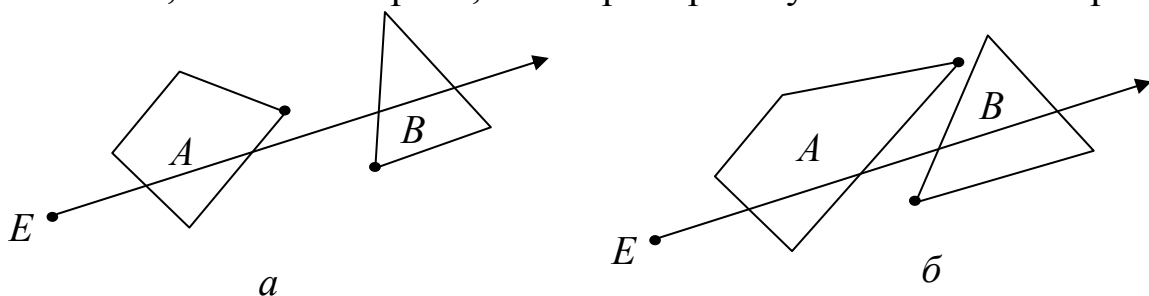


Рис. 14.20. Упорядкування граней за глибиною (E – точка спостерігача)

іншу, буде виводитися пізніше. Зауважимо, що дві опуклі грані, які не мають спільних точок, завжди можна впорядкувати (для неопуклих граней це в загальному випадку неправильно).

Цей алгоритм базується на такому факті: якщо для двох граней A і B найдальша точка грані A знаходиться ближче до спостерігача, ніж найближча точка грані B (назвемо це умовою AB), то грань B не може закрити грань A (рис. 14. 20, a). У випадку, коли проектування ведеться вздовж осі z , це означає, що їхні z -оболонки не перетинаються.

Тому, якщо для двох лицьових граней виконується умова AB , то для впорядкування таких граней достатньо просто відсортувати їх за відстанню до спостерігача (картинної площини). Отже, маємо таку реалізацію алгоритму сортування за глибиною:

- множину всіх лицьових граней сортуємо за глибиною (відстанню до спостерігача або до картинної площини) порядку її зменшення;
- ці грані виводимо в буфер кадру в порядку наближення до спостерігача.

Наведений алгоритм добре працює для цілого ряду типових сцен. Однак таке розміщення граней зустрічається не завжди: можуть зустрітися пари граней в яких найдальша точка однієї грані знаходиться до спостерігача не ближче за найближчу точку іншої грані (умова AB не виконується), тобто перекриваються їхні z -оболонки (рис. 14.20, b). В цьому випадку грань B може виводитися швидше за грань A , тому після попереднього сортування потрібно уточнити порядок виведення граней.

Існують різні методи уточнення відсортованої послідовності граней. Розглянемо *алгоритм Ньюела-Санча*. В цьому алгоритмі не накладається ніяких обмежень на складність сцени і тип многокутників.

Для простоти розглянемо паралельне проектування вздовж осі z . На початку необхідно відсортувати лицьові грані за значенням z_{min} для кожної грані в порядку наближення до спостерігача. Позначимо попередній у списку многокутник через P , а наступний – через Q . Для кожного многокутника P необхідно перевірити його розміщення по відношенню до многокутника Q .

При $z_{max}(Q) \leq z_{min}(P)$ (рис. 14.21, a) ніяка частина P не може закрити Q . У цьому випадку многокутники відсортовані правильно, тому спочатку виводимо многокутник P , а потім Q .

Для сцени, що зображена на рис. 14.21, b ($z_{max}(P) \geq z_{min}(Q)$), не можна одержати остаточний список пріоритетів за глибиною простим сортуванням за z . Якщо P і Q упорядковані за z_{min} , то P в списку передуватиме Q і одержиться, що Q частково закриває P , хоча це не так. В цьому випадку P і Q в списку потрібно поміняти місцями.

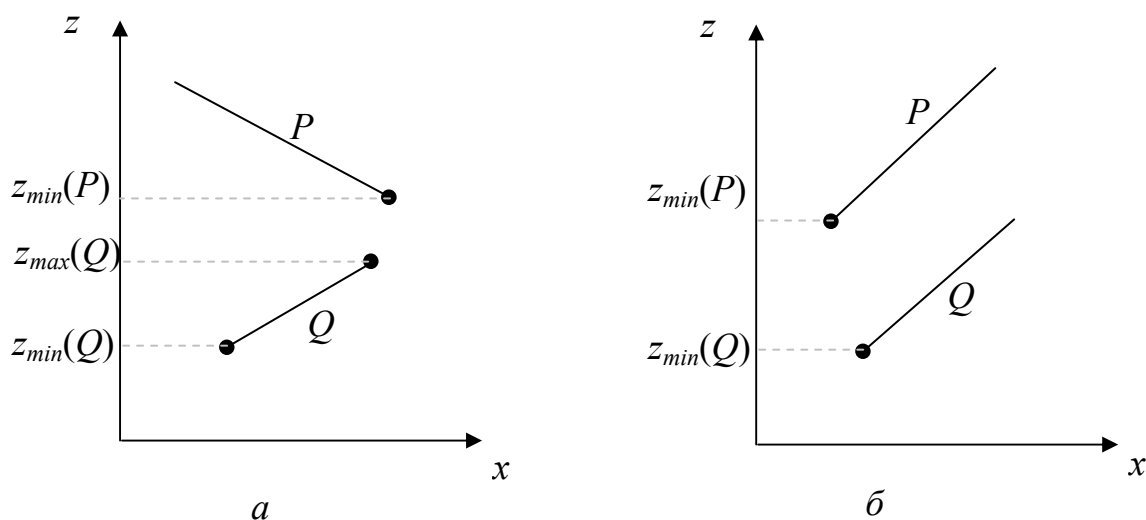


Рис. 14.21. Сортування многокутників за z_{min} ($z_{min}(P) \geq z_{min}(Q)$)

Отже, перед виведенням чергової грані P необхідно переконатися, що ніяка інша грань Q , яка знаходиться в списку після P і z -оболонка якої перетинається з z -оболонкою грані P (якщо немає перетину їхніх z -оболонок, то порядок виведення P , Q однозначно відомий), не може закриватися гранню P . В цьому випадку грань P буде виведена раніше за грань Q . Для цієї перевірки необхідно виконати чотири тести (в порядку зростання їх складності).

Перші два тести використовуються для перевірки перетину проєкцій граней на осях x та y (зауважимо, що впорядковувати необхідно лише ті грані, проєкції яких перетинаються).

- 1) Якщо x -оболонки многокутників не перетинаються, то й самі многокутники та їх проєкції на вісь Ox не перетинаються.
- 2) Якщо y -оболонки многокутників не перетинаються, то й самі многокутники та їх проєкції на вісь Oy не перетинаються.

Якщо виконується хоча б один із цих двох тестів, то не має значення порядок виведення граней. Тому вважатимемо, що грані P , Q упорядковані правильно.

Якщо обидва тести не виконалися, то переходимо до перевірки наступних двох тестів.

- 3) Чи знаходиться грань P і спостерігач по різні боки від площини, що проходить через грань Q (рис. 14.22, а);
- 4) Чи знаходиться грань Q і спостерігач по один бік від площини, що проходить через грань P (рис. 14.22, б).

Якщо хоча б один із тестів 3) або 4) справджується, то вважатимемо, що грані P , Q упорядковані правильно (P передує Q), і далі порівнюємо P із наступною після Q гранню.

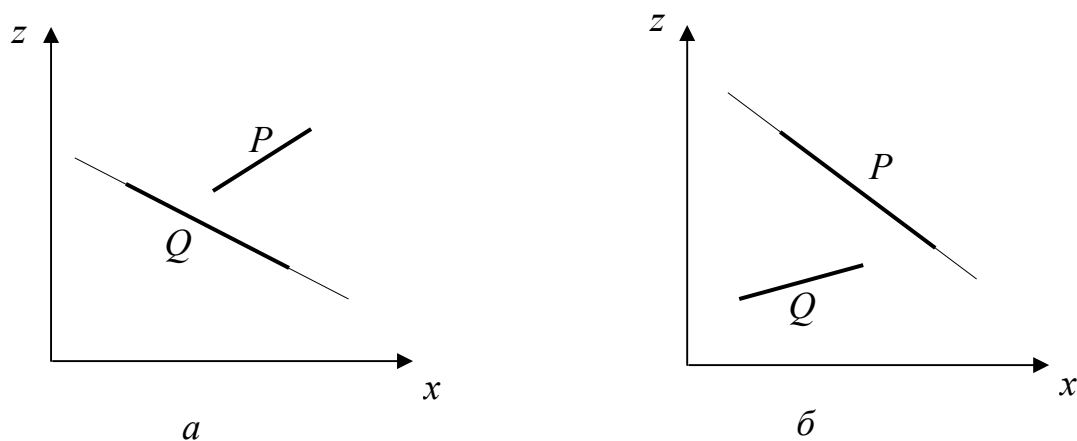


Рис. 14.22. Тести перекриття багатокутників

У випадку, коли жоден із тестів не виконався, міняємо місцями грані P , Q і знову провіряємо тести, аналогічні 3), 4), тобто

3') чи знаходиться грань Q і спостерігач по різні боки від площини, що проходить через грань P ;

4') чи знаходиться грань P і спостерігач по один бік від площини, що проходить через грань Q .

Якщо на одне з цих питань отримана позитивна відповідь, то Q передує P і грань Q буде виводитись раніше.

Водночас зустрічаються випадки, коли задані грані упорядкувати не вдається (рис. 14.23), тобто жоден із тестів не дає позитивної відповіді. Тоді одна з граней розбивається на частини площиною, що проходить через іншу грань, і питання про впорядкування цілої грані і частин іншої грані розв'язується за допомогою тестів 3), 4) або 3'), 4').

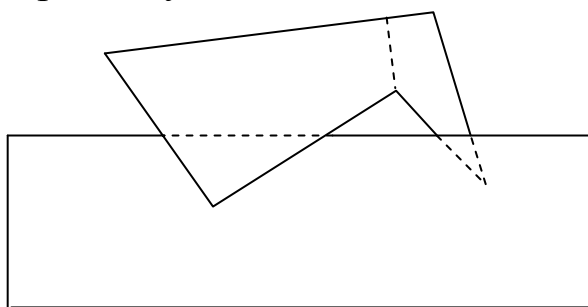


Рис. 14.23. Циклічне перекриття багатокутників

Зауваження 6. Метод сортування за глибиною властивий той самий недолік, що й методу Z -буфера, а саме вимагається виводити всі лицьові грані. Щоб уникнути цього, розроблені модифікації методу, де грані виводяться в зворотному порядку – від найближчих до найдальших, але при цьому для чергової грані

необхідно зображати тільки ті пікселі, які ще не були виведені.

Зауваження 7. Останнім часом з'явилися і більш прийнятні методи розв'язування задач загороджування: метод BSP -дерев, метод октодерев, метод порталів та ін.

Контрольні питання та завдання

1. Які ви знаєте алгоритми усунення невидимих ліній та граней? Як їх можна класифікувати?
2. Як здійснюється побудова поверхні в алгоритмі поточного горизонту?
3. Для чого вводяться хибні бокові ребра в алгоритмі поточного горизонту?
4. Яку структуру даних вимагає алгоритм Робертса?
5. Як в алгоритмі Робертса визначаються нелицьові грані?
6. Як ще можна відсікти нелицьові грані?
7. В чому суть методу Z -буфера?
8. Назвіть недоліки та переваги алгоритму Z -буфера.
9. Як визначити зовнішні нормалі до граней многогранника?
10. Назвіть способи розміщення многокутника відносно вікна.
11. Які тести перевіряє алгоритм Варнока?
12. Як можна легко впорядкувати многокутники в 3D-просторі?
13. Які тести необхідно виконати в алгоритмі Ньюела-Санча?
14. Як перевірити, що грань P і спостерігач знаходяться по різні боки від площини, що проходить через грань Q ?

Вправи і задачі для самостійного виконання

1. Графік поверхні $z = f(x, y)$ можна будувати у вигляді сітки з прямокутних комірок, що формуються множиною значень $\{f(x_i, y_j), i = 1, 2, \dots, n; j = 1, 2, \dots, m\}$, елементи якої розраховуються з постійним приростом по x та y . В результаті одержується зображення, в якому необхідно усунути невидимі грані. Розробити алгоритм відображення такої сітки з усуненням невидимих граней.
2. Узагальнити алгоритм Z -буфера для знаходження перетину сцени площиною.
3. Розробити алгоритм Варнока для усунення невидимих ліній.
4. Для реалізації алгоритму Варнока необхідно мати методи визначення місцерозміщення многокутника відносно прямокутного вікна. Розробити такі алгоритми. *Вказівка. Узагальнити габаритний, променевий та кутовий тести на випадок прямокутного вікна.*
5. Розробити тест визначення нормалі грані многогранника.
6. Розробити алгоритм перевірки тестів 3) та 4) в алгоритмі сортування за глибиною.
7. Написати програму виведення зображення тора на основі методу сортування за глибиною. Модифікувати алгоритм художника для усунення невидимих ліній.

Тема 15. Зафарбовування видимих поверхонь

Послідовність операцій для побудови високоякісних реалістичних 3D-зображень називається 3D-конвеєром.

3D-конвеєр реалізується в декілька етапів. На початку цієї послідовності за допомогою математичних моделей визначають 3D-об'єкти, які будуть відображатися на подальших етапах роботи конвеєра.

На наступному етапі, виходячи з математичної моделі об'єкту, створюється зовнішній вигляд об'єктів у виді геометричних примітивів, тобто генерується його каркасна модель, що складається найчастіше з трикутників (більшість сучасних програм та прискорювачів працюють саме з ними).

Далі визначаються нові координати всіх вершин примітивів у видовій системі координат, виходячи з положення спостерігача та напряму його погляду і виконується проектування сцени зі збереженням ін формації про відстань до спостерігача.

Кінцевий зовнішній вигляд об'єкту формується на етапі, який називається рендерінгом. *Рендерінг* (від англ. "rendering" – візуалізація) – це процес створення високореалістичних зображень 3D-об'єктів. Головною задачею цього етапу є оформлення каркасної моделі об'єктів у вигляді реалістичних поверхонь. Рендерінг умовно можна поділити на такі підетапи: растеризація, усунення невидимих граней та поверхонь, зафарбовування примітивів і фінальна обробка зображення як єдиного цілого.

Видалення прихованих поверхонь – це відсікання пікселів, які є невидимими (його здійснюють за допомогою Z-буферу, що визначає на основі просторових даних, який із трикутників знаходиться найближче до спостерігача та перекриває інші трикутники).

Після виділення невидимих граней для створення реалістичних зображень наступною проблемою є зафарбовування видимих поверхонь, що обмежують побудовані об'єкти. Існує три простіших методи зафарбовування полігональних моделей: метод постійного зафарбовування, метод Гуро та метод Фонга.

Далі для підвищення реалістичності зображень, тобто для максимального наближення графічного зображення до оригіналу, виконують тонування поверхонь, яким присвоюють певний тип матеріалу та враховують умови освітлення, оскільки зовнішній вигляд поверхні суттєво залежить від джерела світла, що освітлює об'єкт, від розміщення та орієнтації поверхні відносно джерел світла, від моделі світла і властивостей поверхні. Тому сучасна фотореалістична графіка – це сплав обчислювальної фізики, математичних методів та витончених алгоритмів.

Спочатку розглянемо деякі фізичні властивості та моделі освітлення.

15.1. Моделі відбиття світла

Процеси моделювання освітлення базуються на законах геометричної оптики, таких як закони заломлення, відбиття, прямолінійності тощо.

Енергія світла, що падає на поверхню від джерела світла, може бути частково поглинута, відбита або пропущена через поверхню. Кількість енергії, що поглинається, відбивається і пропускається залежить від довжини λ світлової хвилі. Якщо об'єкт поглинає повністю все падаюче світло, то він буде виглядати невидимим. У цьому випадку його називають абсолютно чорним тілом. Якщо поглинається майже все світло, то об'єкт виглядає чорним, а якщо поглинається тільки невелика частина світла, то сірим або білим. Об'єкт можна побачити тільки тоді, коли він відбиває або пропускає світло. Властивості відбитого світла, які визначають колір поверхні, залежать від форми і напрямку джерела світла, від орієнтації поверхні, на яку падає світло та від властивостей самої поверхні. Цей факт враховується в різних моделях освітлення. Світло, відбите від поверхні об'єкта, може бути *дифузним* (diffuse) та *дзеркальним* (specular). Дифузно відбите світло розсіюється рівномірно у всіх напрямках, дзеркальне відбиття відбувається вздовж одного променя.

Дійсна теорія світла є досить складною з теоретичної та обчислювальної точок зору, тому будуються спрощені моделі освітлення.

Розглянемо простіші моделі освітлення і вкажемо, як можна визначити колір пікселів для зображення поверхні, враховуючи інтенсивність відбитого світла, взаємне розміщення поверхні, джерел світла та спостерігача.

15.1.1. Дзеркальне відбиття світла

До дзеркальних відносяться скляні поверхні, оброблені металеві поверхні, деякі кам'яні поверхні, чистий лід тощо. Світлова енергія променя світла, що падає на дзеркальну поверхню, відбивається по лінії відбитого променя (будь-яке розсіювання вбік від цього променя відсутнє). Кут відбиття від ідеально дзеркальної поверхні дорівнює куту падіння. В будь-якому іншому положенні спостерігач не бачить дзеркально відбите світло, тобто вектор спостереження S збігається з вектором відбиття R і кут $\alpha = 0$ (рис 15.1).

Поверхня вважається *ідеально дзеркальною*, якщо на ній немає шорсткості. В природі немає ідеально дзеркальних поверхонь, тому вважають, що глибина шорсткості істотно менша за довжину хвилі випромінювання. Для видимого спектра можна вважати, що глибина шорсткості для ідеального дзеркала має не перевищувати 0,5 мкм.

Якщо поверхня не ідеальна, то кількість світла, що попадає до спостерігача, залежить від просторового розподілу дзеркально відбитого світла. Для більш гладких поверхонь розподіл відбитого світла більш вузький і сфокусований уздовж вектора відбиття, для шорстких – більш широкий.

Інтенсивність дзеркально відбитого світла залежить від кута падіння світла, від довжини хвилі і від властивостей матеріалу.

До простіших моделей дзеркального освітлення відносять емпіричну модель Буї-Туонга Фонга, згідно з якою інтенсивність дзеркально відбитого світла I_s знаходиться за формулою

$$I_s = I \omega(\theta, \lambda) \cos^p \alpha,$$

де I – інтенсивність джерела випромінювання світла, $\omega(\theta, \lambda)$ – крива відбиття світла, α – кут між відбитим від точки поверхні променем і вектором спостерігача, тобто вектором, направленим із точки поверхні в точку, де знаходиться спостерігач (рис. 15.1), p – показник, що знаходиться в діапазоні від 1 до 200, залежить від ідеальності поверхні і характеризує просторовий розподіл дзеркально відбитого світла, тобто p – це степінь дзеркального відображення матеріалу. Степінь p впливає на розмір відблиску на поверхні (при дзеркальному відбитті на блискучих предметах виникають світлові бліки). Великі значення p задають сфокусовані розподіли для металів, малі p – для неметалевих поверхонь. Варто зазначити, що відбитий дзеркальний промінь від металу зберігає властивості падаючого променя. Наприклад, при освітленні блискучої синьої поверхні білим світлом виникають білі, а не сині бліки.

Функція $\omega(\theta, \lambda)$ досить складна, тому її часто заміняють константою k_s , яка називається *коефіцієнтом дзеркального відбиття* і визначається експериментально. Тобто на практиці модель Фонга використовується у формі

$$I_s = I k_s \cos^p \alpha, \quad 0 < k_s < 1. \quad (15.1)$$

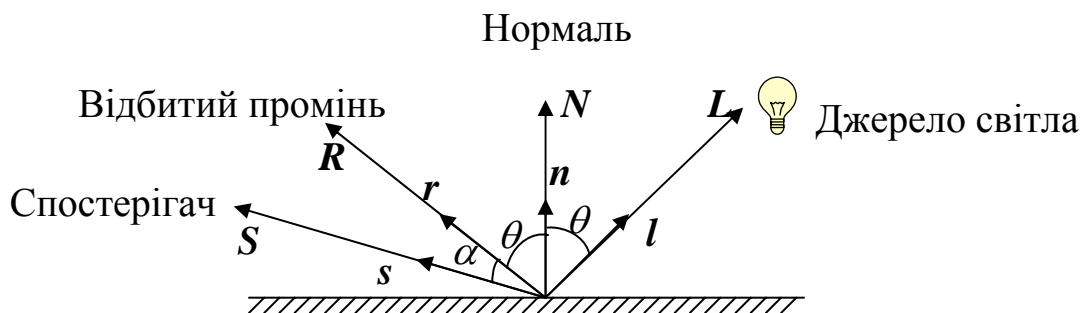


Рис. 15. 1. Дзеркальне відбиття світла: r, s, n, l – одиничні вектори

15.1.2. Дифузне відбиття

Цей тип відбиття характерний для шорстких поверхонь (матових), де розмір шорсткості настільки великий, що падаюче світло розсіюється в усі боки. Такий тип відбиття світла характерний для паперу, гіпсу, піску.

Якщо поверхня є слабо шорсткуватою, то відбите світло розсіюється в зоні лінії ідеально відбитого променя.

При наявності шорсткуватості інтенсивність відбитого світла залежить від кута падіння. Розміщення спостерігача не має значення, оскільки дифузно відбите світло розсіюється в усіх напрямках рівномірно.

Дифузне відбиття світла описується законом Ламберта (законом косинусів), згідно з яким інтенсивність відбитого світла I_d пропорційна косинусу кута θ між напрямком L від точки поверхні на джерело світла та нормаллю N до поверхні в цій точці (рис.15.2), тобто

$$I_d = I k_d \cos \theta, \quad 0 < \theta < \frac{\pi}{2}, \quad (15.2)$$

де I – інтенсивність точкового джерела світла, k_d – коефіцієнт дифузного відбиття, що залежить від властивостей матеріалу поверхні ($0 \leq k_d \leq 1$) [21,25], θ – кут між напрямком від точки, в якій визначається колір на джерело світла, і зовнішньою нормаллю до площини в цій точці. З формули Ламберта видно, що інтенсивність відбитого світла максимальна для кутів θ , близьких до 0° . Поверхня предметів, що зображається за допомогою моделі освітлення Ламберта (з дифузним відбиттям), виглядає бляклою і матовою.

Але в природі не існує ідеально дзеркальних або повністю матових поверхонь, тому при побудові реальних графічних зображень необхідно поєднувати дві моделі відбиття світла в деякій пропорції, що характерна для конкретного матеріалу поверхні.

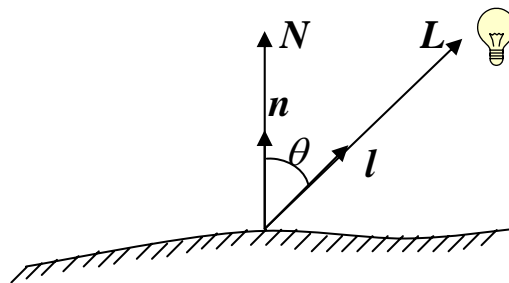


Рис. 15.2. Дифузне відбиття

У цьому випадку модель відбиття світла записують у вигляді суми дифузної і дзеркальної компоненти

$$I^* = I(k_d \cos \theta + k_s \cos^p \alpha). \quad (15.3)$$

Згідно з формулою (15.3), інтенсивність відбитого світла дорівнює нулю для деяких кутів θ і α , однак у реальних сценах немає абсолютно темних місць. Це означає, що необхідно враховувати розсіяне (ambient) світло, багатократно відбите від інших об'єктів. Розсіяне світло йде з усіх напрямків і складає фонове освітлення сцени. При такому освітленні об'єкти однаково освітлені з усіх боків і ці об'єкти не утворюють тіні. Наприклад, освітлення вдень у похмуру погоду (коли небо в хмарах і не видно сонця). В комп'ютерній графіці вважається, що джерело фоновому підсвічування завжди одне і освітлення в будь-якій точці сцени в довільному напрямку має однакову інтенсивність. У цьому випадку інтенсивність відбитого світла знаходиться з такої моделі освітлення

$$I^* = I_a k_a + I(k_d \cos \theta + k_s \cos^p \alpha), \quad (15.4)$$

де I_a – інтенсивність фоновому розсіяного світла (постійна для всіх об'єктів), $0 \leq k_a \leq 1$ – коефіцієнт відбиття поверхнею розсіяного світла, k_d, k_s , як і раніше, коефіцієнти дифузного і дзеркального відбиття, відповідно.

Приклади параметрів моделі освітлення (15.4) для деяких матеріалів наведені в табл.15.1.

Таблиця 15.1.

Матеріал	Фонові коеф. k_a			Дифузійні коеф. k_d			Дзеркальні коеф. k_s			Степінь p
	R	G	B	R	G	B	R	G	B	
латунь	0.3294	0.2235	0.0275	0.7804	0.5687	0.1137	0.9922	0.9412	0.8078	28
бронза	0.2125	0.1275	0.0540	0.7140	0.4284	0.1814	0.3935	0.2719	0.1667	26
хром	0.2500	0.2500	0.2500	0.4000	0.4000	0.4000	0.7746	0.7746	0.7746	77
мідь	0.1913	0.0735	0.0225	0.7038	0.2705	0.0828	0.2568	0.1376	0.0860	13
поліроване срібло	0.2313	0.2313	0.2313	0.2775	0.2775	0.2775	0.7739	0.7739	0.7739	90
чорна пласмаса	0.0	0.0	0.0	0.0100	0.0100	0.0100	0.5000	0.5000	0.5000	32

Інтенсивність світла, природно, залежить від віддалі d точки поверхні до джерела світла (об'єкт, що знаходиться даліше від джерела світла має бути темнішим). Тому в комп'ютерній графіці для більшої реалістичності використовують модель освітлення з лінійним затуханням

$$I^* = I_a k_a + \frac{I}{d+k} (k_d \cos \theta + k_s \cos^p \alpha), \quad (15.5)$$

де d – віддаль від об'єкта до джерела світла, k – деяка константа.

Зауваження 1. Інколи у формулі (15.5) замість коефіцієнта зменшення інтенсивності $\frac{1}{k+d}$ використовують $\frac{1}{k+c_1d+c_2d^2}$, де пара-

метри k , c_1 , c_2 називають коефіцієнтами постійного, лінійного та квадратичного ослаблення світла в залежності від віддалі d .

Функція (15.5) і є функцією зафарбовування. Вона використовується для розрахунку тону зафарбовування точок поверхні, тобто пікселів графічного зображення.

Найпростіше виконувати розрахунки в градаціях сірого кольору (наприклад, для білого джерела світла та сірих об'єктів). У цьому випадку інтенсивність відбитого світла відповідає яскравості. Складніше з кольоровими джерелами світла, які освітлюють кольорові поверхні. В цьому випадку розрахунок інтенсивності за формулою (15.5) ведеться для кожної компоненти кольору окремо (наприклад, для R , G , B) і колір кожного пікселя зображення визначають значення складових інтенсивностей світла для відповідних компонент кольору.

Якщо точкових джерел світла декілька (наприклад m), то їх ефекти сумуються. В цьому випадку модель інтенсивності освітлення має вигляд

$$I^* = I_a k_a + \sum_{j=1}^m \frac{I_j}{d_j + k} (k_d \cos \theta_j + k_s \cos^{p_j} \alpha_j). \quad (15.6)$$

Використовуючи одиничні вектори зовнішньої нормалі \mathbf{n} , а також одиничні вектори, що визначають напрями на джерело світла (вектор \mathbf{l}), відбитого променя (вектор \mathbf{r}), на спостерігача (вектор \mathbf{s}) (рис. 15.1), функцію зафарбовування (15.5) можна записати у вигляді

$$I^* = I_a k_a + \frac{I}{d + k} (k_d (\mathbf{n} \cdot \mathbf{l}) + k_s (\mathbf{r} \cdot \mathbf{s})^p), \quad (15.7)$$

тобто косинуси кутів можна замінити скалярними добутками одиничних векторів.

Ще однією характеристикою точкового джерела світла є його *сфокусованість*, тобто розподіл інтенсивності світла по поверхні об'єкта. Найбільша інтенсивність освітлення сфокусованим джерелом знаходиться в точці перетину напрями джерела світла з поверхнею. В околі цієї точки інтенсивність буде зменшуватися, при цьому інтенсивність світла обчислюється за законом $I_p = I \cos^t \varphi$, де φ – кут між вектором, що задає напрям джерела світла та вектором \mathbf{m} , напрямленим від джерела світла до точки поверхні (рис. 15.3), t – величина, що описує сфокусованість джерела світла. Параметр t може набувати цілих значень від 0 до 128. Чим більше t , тим більше сфокусоване світло; при $t = 0$ маємо розсіяне світло. В цьому випадку замість формули (15.7) використовуємо наступну формулу

$$I^* = (I_a k_a + \frac{I}{d+k} (k_d (\mathbf{n} \cdot \mathbf{l}) + k_s (\mathbf{r} \cdot \mathbf{s})^p)) \cos^t \varphi.$$

Зауважимо, що існують більш складні моделі освітлення, наприклад моделі Варна та Торренса-Сперроу.

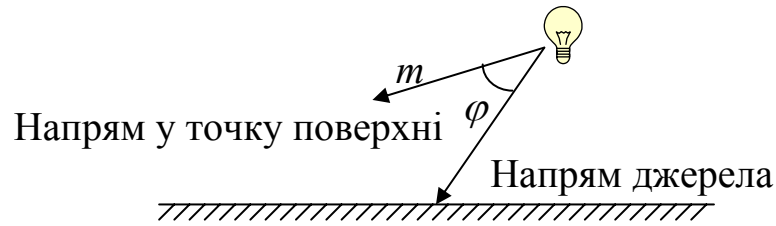


Рис. 15.3. Сфокусоване джерело світла

15.2. Обчислення нормалей до поверхні відбиття світла

Вектори \mathbf{l} та \mathbf{s} задаються, а вектори \mathbf{n} та \mathbf{r} потрібно знаходити. Якщо освітлювана поверхня гладка, то вектор зовнішньої нормалі обчислюється безпосередньо, наприклад, якщо рівняння поверхні має вигляд $F(x, y, z) = 0$, то вектор зовнішньої нормалі задають частинні похідні

$$\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}.$$

У випадку многогранної поверхні вектори зовнішніх нормалей можна знайти тільки для граней, тобто якщо грані поверхні лежать у площинах, що описуються рівняннями

$$a_i x + b_i y + c_i z + d_i = 0, \quad i = 1, 2, \dots, m,$$

то нормальні вектори цих площин (a_i, b_i, c_i) є векторами зовнішніх нормалей для многогранної поверхні. Для знаходження коефіцієнтів a_i, b_i, c_i досить мати три вершини грані, якщо полігональна поверхня має не трикутні грані, а, наприклад, чотирикутні, то розрахунок нормалі можна здійснювати за довільними трьома вершинами грані. Що ж до напрямків векторів зовнішніх нормалей на ребрах або в вершинах многогранника, то їхні значення можна знайти лише наближено.

Нехай грані зі спільною вершиною мають нормальні вектори з координатами (a_i, b_i, c_i) , $i = 1, 2, \dots, m$, тоді за нормальний вектор у вершині многогранника можна взяти суму векторів, тобто $(a, b, c) = \sum_{i=1}^m (a_i, b_i, c_i)$

(рис. 15.4). Інколи для знаходження вектора нормалі в вершині використовують нормовану зважену суму векторів нормалі граней, яким належить ця вершина.

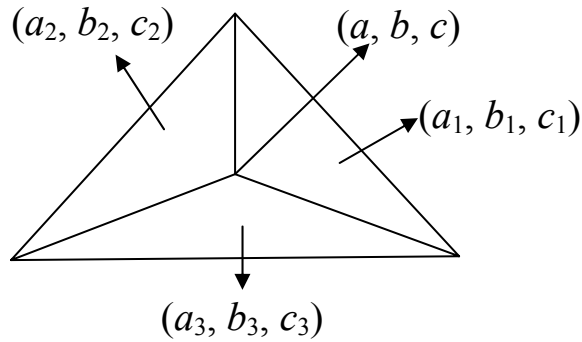


Рис. 15.4. Нормаль у вершині при $m = 3$

Для визначення напрямку нормалі в точці, що належить ребру многогранної поверхні, достатньо додати вектори нормалей тих граней, які прилягають до даного ребра. А можна зробити по-іншому, тобто апроксимувати змінний вектор нормалі вздовж ребра многогранної поверхні можна за допомогою вже знайдених векторів зовнішніх нормалей у вершинах, що утворюють дане ребро. Якщо многогранна поверхня задана своїми вершинами V_1, V_2, V_3, V_4 , то вектор зовнішньої нормалі в її вершинах можна знайти, використовуючи векторні добутки, які побудовані на векторах, що йдуть вздовж ребер і виходять із цих вершин. Наприклад, для того, щоб знайти зовнішню нормаль у вершині V_1 , необхідно додати векторні добутки $V_1V_2 \times V_1V_3$, $V_1V_3 \times V_1V_4$, $V_1V_4 \times V_1V_2$ (рис. 15.5).

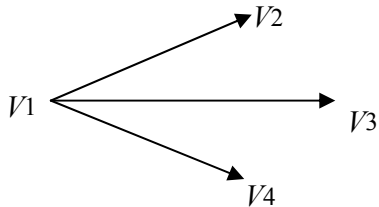


Рис. 15.5. Визначення вектора зовнішньої нормалі

Зауваження 2. Якщо перед додаванням знайдені векторні добутки пронормувати, то знайдена сума буде відрізнятися від попередньої і за довжиною, і за напрямком.

Якщо розміщення джерела світла описується вектором l , направленим на джерело світла, то косинус кута θ з вектором нормалі n знаходять із формули скалярного добутку

$$\cos \theta = \frac{(n \cdot l)}{|n| \cdot |l|}. \quad (15.8)$$

У випадку, коли джерело світла розміщується в скінченній точці простору (x_c, y_c, z_c) , для визначення $\cos \theta$ потрібно спочатку знайти вектор, який направлений на джерело світла.

Вивчимо питання про те, як знайти напрямок відбитого променя r та $\cos\alpha$. Будемо вважати, що заданий вектор l напрямлений на джерело світла, а також відомий вектор нормалі n .

Як відомо, при дзеркальному відображенні вектори l, r, n лежать в одній площині, причому кути між векторами l, n та n, r однакові (рис. 15.1).

Розглянемо модель освітлення з одним точковим джерелом. Припустимо, що світло падає вздовж осі z у від'ємному напрямку, тобто якщо розрахунки проводяться у видовій системі координат, то це означає, що джерело світла розміщується на одній осі з камерою. В цьому випадку координати одиничного вектора відображення визначаються за формулами

$$r_x = 2n_x n_z, \quad r_y = 2n_y n_z, \quad r_z = 2n_z^2 - 1. \quad (15.9)$$

Знайдемо вектор r у загальному випадку (рис. 15.6).

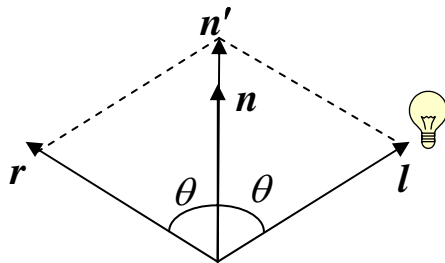


Рис. 15.6. Вектори r, l, n одиничної довжини

Вектор $r + l = n'$ є діагоналлю ромба. Напрямок n' збігається з n , причому $|n'| = 2\cos\theta$, тому $n' = 2n \cos\theta$, тобто $r + l = 2n \cos\theta$. Звідси знаходимо одиничний вектор відбитого променя

$$r = 2n \cos\theta - l.$$

Враховуючи, що $\cos\theta = (l \cdot n)$, одержуємо

$$r = 2n(l \cdot n) - l,$$

а у скалярній формі

$$\begin{aligned} r_x &= 2n_x(l_x n_x + l_y n_y + l_z n_z) - l_x, \\ r_y &= 2n_y(l_x n_x + l_y n_y + l_z n_z) - l_y, \\ r_z &= 2n_z(l_x n_x + l_y n_y + l_z n_z) - l_z. \end{aligned}$$

Тепер уже можна обчислити $\cos\alpha = (s \cdot r)$, де s – одиничний вектор, направлений на спостерігача.

15.3. Зафарбовування поверхонь

Для зафарбовування поверхонь використовують метод зафарбовування з постійною інтенсивністю. У цьому випадку кожна грань об'єкта зафарбовується постійним кольором, що визначається моделлю освітлення. Існують спеціальні методи зафарбовування, які дозволяють створити ілюзію гладкості. Ці методи забезпечують неперервність освітле-

ності вздовж границь об'єктів. Найбільш відомі методи побудови згладжених зображень – це методи Гуро та Фонга. Більш простий з них метод Гуро. Розглянемо ці методи.

15.3.1. Зафарбовування з постійною інтенсивністю

Це найпростіша модель зафарбовування граней. Її зміст полягає у тому, що на грані береться точка, наприклад остання точка в списку вершин грані, і визначається її освітленість. Потім вся грань зафарбовується однаковим кольором, що визначається знайденою освітленістю.

Запропонований алгоритм має великий недолік, оскільки побудоване при цьому зображення має яскраво виражений многогранний вигляд. Відразу видно, що графічний об'єкт складається з окремих плоских граней. Це пояснюється тим, що освітленість сцени, яка так визначається, не є неперервною величиною, а має кусково-постійний характер, тобто освітленість об'єкта сцени зазнає розривів на границях граней. А якщо кожна грань має один постійний колір, визначений моделлю освітлення, то різні кольори сусідніх граней помітні, і об'єкт виглядає як многогранник, а не як гладка поверхня. Для створення ілюзії гладкості можна було б збільшувати кількість граней при апроксимації поверхні об'єкта, але оскільки зір людини має властивість помічати перепади яскравості кольору на границях суміжних граней, то для створення ілюзії гладкості потрібно істотно збільшувати кількість граней, що веде до великої обчислювальної роботи і, як наслідок, зменшується швидкість візуалізації зображення.

Зауваження 3. При використанні однорідного зафарбовування границі областей здаються більш яскравими (ефект Маха), в результаті чого об'єкти з постійною інтенсивністю сприймаються як об'єкти з дещо змінною інтенсивністю.

15.3.2. Метод Гуро

Метод Гуро використовується для створення ілюзії гладкої криволинійної поверхні, яка задається у вигляді многогранників або полігональної сітки з плоскими гранями за умови, що точкове джерело світла випромінює розсіяне світло.

Цей метод базується на ідеї зафарбовування кожної грані об'єкта не одним кольором, а різними відтінками, які плавно змінюються, тобто в цьому методі визначається освітленість грані у вершинах, а потім вона білінійним чином інтерполюється на всю грань.

Зафарбовування методом Гуро здійснюється за чотири кроки:

- обчислення нормалей до кожної грані;

- обчислення нормалей у вершинах, наприклад методом усереднення нормалей сусідніх граней (рис. 15.4);
- на основі нормалей у вершинах обчислюються значення інтенсивностей у вершинах згідно з вибраною моделлю, наприклад (15.4);
- зафарбовуються грані кольором, що відповідає лінійній інтерполяції значень інтенсивності світла у вершинах.

Розглянемо, як можна визначити інтенсивність відбитого світла в кожній точці грані (а отже, і колір кожного пікселя). Ця операція виконується в циклі під час заповнення полігону.

Нехай задана грань ABC (рис. 15.7) в екранних координатах. Використовуючи моделі освітлення, знайдемо інтенсивності відбитого світла у вершинах – нехай це будуть значення I_A, I_B, I_C . Знайдемо освітленість у точці P . Зображаючи грань рядок за рядком, будемо знаходити значення інтенсивності в кінцях кожного горизонтального відрізка MN шляхом

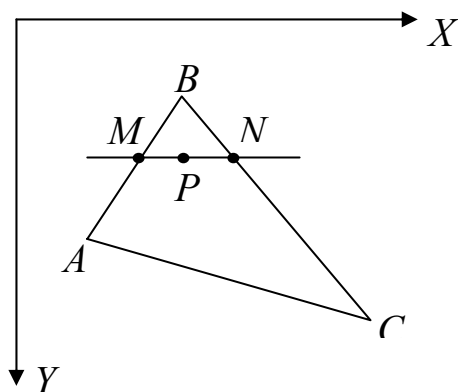


Рис. 15.7. Заповнення грані

лінійної інтерполяції значень вздовж ребер.

Так, освітленість у точці M обчислимо за формулою

$$I(M) = I(A)t + I(B)(1 - t), \quad t = \frac{|MA|}{|AB|}.$$

Аналогічно обчислюємо $I(N)$. Маючи значення $I(M), I(N)$, за допомогою лінійної інтерполяції знаходимо $I(P)$:

$$I(P) = I(M)(1 - s) + I(N)s, \quad s = \frac{|PM|}{|MN|}, \quad \text{або } s = \frac{|X_P - X_M|}{|X_N - X_M|}.$$

Легко перевірити, що колір точки P є лінійною комбінацією кольорів у точках A, B, C із коефіцієнтами, сума яких дорівнює одиниці.

Метод Гуро забезпечує неперервність інтенсивності освітлення многогранної поверхні, але не забезпечує гладкість (диференційованість), оскільки застосовувалася лінійна інтерполяція освітлення. Цей метод дає багато інших дефектів на зображуваній поверхні, однак найоптимальніший в обчислювальному плані, тому його використовують тільки в попередніх розрахунках. Наступний метод забезпечує гладкість зміни освітленості.

15.3.3. Метод Фонга

Цей метод аналогічний до методу Гуро (він теж базується на інтерполюванні), але, на відміну від методу Гуро, тут інтерполюються не значення інтенсивності відбитого світла, а значення вектора зовнішньої нормалі. Тобто, метод Фонга полягає в побудові для кожного пікселя вектора, що відіграє роль нормалі і цей вектор використовується для обчислення інтенсивностей компонент відбитого світла в кожній точці згідно з вибраною моделлю освітлення.

Тому зафарбовування методом Фонга вимагає значно більше обчислень, оскільки інтерполюються три векторних компоненти, проте одержуються більш реальні графічні зображення.

Розглянемо, як можна одержати вектор нормалі, а отже, колір зафарбовування в довільній точці P грані ABC (рис. 15.8).

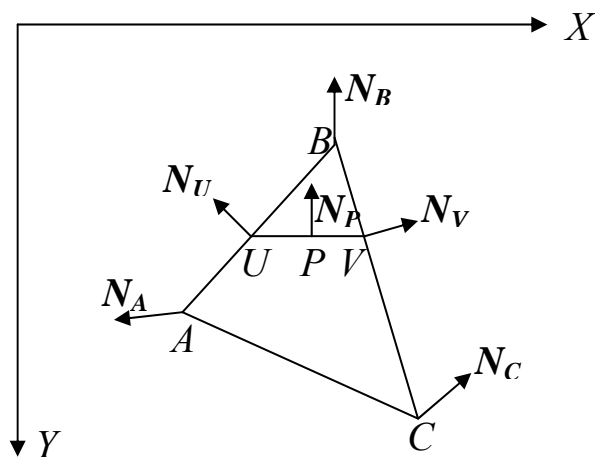


Рис. 15.8. Інтерполяція нормалей

Маючи нормалі примикаючих граней, знаходимо нормалі в опорних точках (вершинах A , B , C). За допомогою лінійного інтерполювання векторів N_U , N_V знайдемо N_P (рис. 15.8), а для знаходження N_U , N_V інтерполюємо вектор нормалі вздовж ребер граней, тобто знаходимо

$$N_U = (1 - t)N_A + tN_B, t = \frac{|UA|}{|AB|},$$

$$N_V = (1 - t)N_B + tN_C, t = \frac{|BV|}{|BC|},$$

$$N_P = (1 - s)N_U + sN_V, s = \frac{|UP|}{|UV|}.$$

Але оскільки в моделях освітленості використовуються одиничний вектор нормалі, то вектори N_U , N_V , N_P потрібно нормувати.

Зауваження 4. Змінюючи довільно вектор нормалі можна моделювати поверхні з різними властивостями.

Як бачимо, перевагою моделей зафарбовування Гуро і Фонга є їх відносна простота, але одержаний результат може бути не завжди задовільним. Тому метод зафарбовування Фонга часто не підтримується апаратно, а в OpenGL навіть не включений у графічні бібліотеки.

Поліпшувати якість зображення дозволяють досконаліші методи, наприклад методи трасування променів.

15.4. Методи трасування променів

Природно, що простіші локальні моделі освітлення мають певні обмеження, оскільки в цьому випадку кожна поверхня аналізується окремо, а отже, неможливо сформуванати тінь, що падає від одного об'єкта на інший, як і відблиски на поверхнях, викликані дзеркальним відбиттям світла від інших поверхонь об'єктів.

Для створення високореалістичних зображень необхідно враховувати не тільки прямі промені від джерела світла, але й другорядні (промені, що відбиваються від поверхні, і ті, що проходять через поверхню та заломлюються). Тому на практиці використовують більш складні моделі глобального розподілу світла, з яких найбільш потужними і універсальними є методи трасування променів та аналізу випромінюваності. Відомо багато різних прикладів реалізації цих алгоритмів для побудови якісного зображення досить складних тривимірних сцен.

Модель трасування променів адекватно передає оптичні ефекти в сценах із дзеркальними поверхнями, наприклад, коли значна частина об'єктів зроблена зі скла, полірованого металу та інших аналогічних матеріалів. Модель аналізу випромінюваності краще передає оптичні ефекти в сценах із дифузними поверхнями.

Розглянемо, як в методі трасування формується зображення сцени, що складається з кількох просторових об'єктів. Основна ідея цього методу полягає в математичному моделюванні шляху кожного світлового променя, що приймає участь в формуванні зображення.

Від джерел випромінювання в різні напрямки виходить безліч променів світла. Деякі з них йдуть у вільний простір, а деякі потрапляють на об'єкти сцени. При взаємодії променя світла (фотонів) із поверхнею об'єкта він може бути поглинутий, дзеркально відбитий, дифузно відбитий або заломлений.

Якщо промінь падає на прозорі тіла, він переломлюючись, іде далі, але при цьому частина енергії світла поглинається. Якщо на шляху променя зустрічається дзеркальна поверхня, то промінь також змінює свій напрямок (дзеркально відбивається від поверхні) і частина світлової енергії теж поглинається. Якщо об'єкт дзеркальний і водночас прозорий, то зразу одержуємо два промені.

Отже, в результаті дії на об'єкт первинних променів одержуються вторинні промені, частина яких далі потрапляє на об'єкти, тобто кожна точка сцени може освітлюватись або безпосередньо джерелом світла, або відбитим чи заломленим світлом (рис.15.9).

В результаті численних проходів таких променів деякі з них попадуть в точку спостереження – око людини чи оптичну систему камери – і сформуєть у ній зображення сцени. Очевидно, що в точку спостереження можуть попадати і первинні промені безпосередньо від джерела світла. Таким чином, зображення сцени формується деякою підмножиною світлових променів.

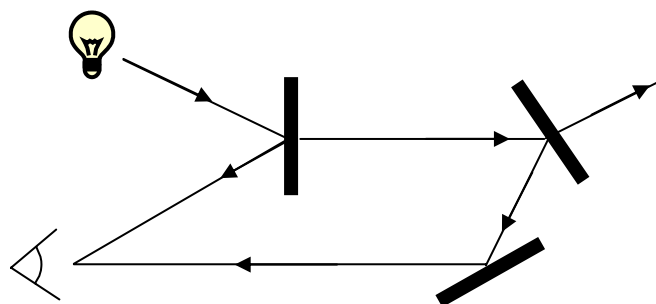


Рис. 15.9. Пряме трасування променів

Колір кожної точки зображення визначається спектром та інтенсивністю первинних променів джерел випромінювання, а також поглинанням світлової енергії в об'єктах, що зустрілися на шляху цих променів.

Можна спробувати створити алгоритм побудови зображень у такий спосіб. Цей метод називається *прямим трасуванням* променів. Але головний недолік цього методу – велика обчислювальна робота, тому що необхідно перебрати всі первинні промені для всіх джерел світла і всіх граней, потім – вторинні промені і т.д., тобто простежити всі траєкторії частинок світла від моменту генерації їх джерелом світла до моменту поглинання, потрапляння в об'єктив камери чи виходу зі сцени. При цьому у формуванні зображення візьме участь лише невелика частина променів, а саме ті, які потрапляють у точку спостереження (наприклад, на сітківку ока), тому багато зайвих обчислень для тих променів, які не потрапили в точку спостереження, проведені даремно. Такий алгоритм обчислювально неефективний, тому на практиці застосовують метод зворотного трасування.

Метод *зворотного трасування* дозволяє значно скоротити перебір світлових променів, а отже, уникнути зайвих обчислень. У цьому методі обчислюються інтенсивності тільки тих променів, які попадають в точку спостереження, тобто відстеження променів здійснюється не від джерел світла, а в зворотному напрямку – від точки спостереження до джерела

Розглянемо, як можна одержати растрове зображення деякої тривимірної сцени методом зворотного трасування. В простішій реалізації методу зворотного трасування відстежуються промені, що виходять з ока спостерігача через кожний піксель площини проєкцій у 3D-сцену. Це будуть первинні промені зворотного трасування.

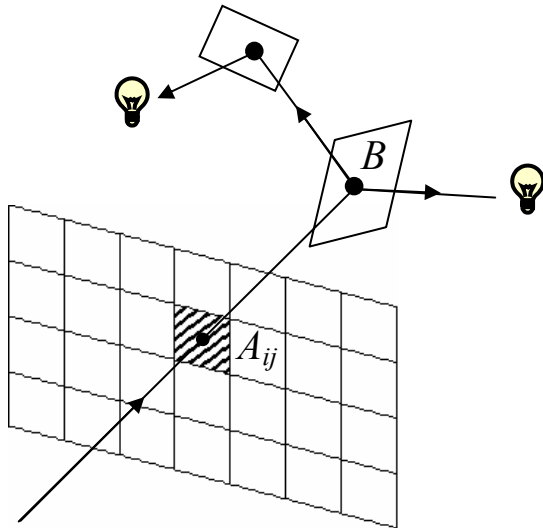


Рис. 15.10. Зворотне трасування променів

Виберемо центральну проєкцію з центром в точці O на деякій відстані від площини проєктування. Проведемо пряму лінію з центра проєктування O в середину пікселя A_{ij} площини проєктування. Якщо пряма лінія цього променя попадає в один чи кілька об'єктів сцени, то обираємо найближчу точку перетину B . Інтенсивність пікселя A_{ij} площини проєкцій однозначно визначається освітленістю точки B . В свою чергу, освітленість точки B визначають відбита та заломлювана енергія, отримана від джерел світла та від інших об'єктів сцени. Для визначення освітленості

точки B з точки B надсилають та аналізують промені до джерел світла, а також вторинні відбиті та заломлені промені.

Якщо поверхня дзеркальна, то вторинний промінь будуюмо як промінь падіння, вважаючи променем відбиття попередній (первинний) промінь зворотного трасування. Вище були наведені формули для обчислення вектора дзеркально відбитого променя. Ці ж формули можна використати для розрахунку падаючого променя (відбиття навпаки).

Якщо об'єкт прозорий, то необхідно побудувати ще один вторинний промінь, такий, який би при заломленні давав первинний трасований промінь (рис. 15.10). Вектор цього променя можна розрахувати на основі фізичних властивостей заломлення світла.

Далі ці два промені аналізуються окремо – вони можуть перетинатися з поверхнями, закінчуватися на деякому джерелі світла або назавжди покидати сцену.

Якщо поточний промінь зворотного трасування не перетинає будь-який об'єкт, а попадає у вільний простір, то на цьому трасування для цього променя закінчується. При стиканні вторинного променя з поверхнею об'єкта в загальному випадку виникають два нових вторинних промені – один із них задає напрям падіння, а другий напрям заломлення

і т.д. Ці промені відстежуються далі, щоб визначити перетин з поверхнями.

В результаті для кожного пікселя будується деревовидна структура із сотень променів. Вітки такого дерева задають розповсюдження променів в сцені, а вузли – перетин з поверхнями в сцені.

Якщо джерело світла видно з точки перетину, то на початку необхідно обчислити „вклад” від цього джерела в освітленість цієї точки, використовуючи стандартну модель відображення.

Для ідеального дзеркала (один промінь падає – один відбивається) достатньо знаходити лише чергову точку перетину вторинного променя з деяким об’єктом. Якщо дзеркало неідеальне, то метод зворотного трасування значно ускладнюється, оскільки потрібно уже відстежувати не один промінь, а множину (сотні) променів, які складають конус з віссю вздовж лінії падаючого променя для ідеального дзеркала.

Якщо об’єкт володіє властивостями дифузного відбиття, то знову, як і у випадку неідеального дзеркала, необхідно трасувати всі промені, що приходять від усіх об’єктів сцени, тобто джерелом дифузного світла може бути будь-який видимий з даної точки об’єкт, здатний передавати світлову енергію.

Якщо спробувати відстежити траєкторію цих променів, то задача знову значно ускладниться, тому при дифузному відбитті враховують тільки промені, що йдуть від джерел світла. Відбиті промені від поверхні об’єктів ігноруються.

Тому метод зворотного трасування найкраще підходить для 3D-сцен без складних джерел світла та дифузних поверхонь (наприклад, для сцен із точковими джерелами світла та ідеально дзеркальними або прозорими тілами)

Для визначення кольору кожного пікселя зображення потрібно враховувати спектральні оптичні властивості всіх поверхонь об’єкта, а також природу світла, яке падає у відповідну точку об’єкта і знаходити інтенсивності його складових частин. На основі складових частин (червона, зелена, синя) інтенсивності пікселя формують його колір.

При моделюванні хроматичного освітлення, обчислення для всіх трьох променів проводять зовсім незалежно. Якщо при цьому не враховують явище заломлення світла, то траси червоного, зеленого і синього кольорів повністю співпадають, тобто при дифузному і дзеркальному відбитті змінюється лише їх інтенсивності.

При дзеркальному відбитті колір пікселя (точки поверхні) визначається кольором відбитого променя. При дифузному відбитті колір освітленої точки поверхні визначається власним кольором поверхні й кольо-

леної точки поверхні визначається власним кольором поверхні й кольором джерел світла.

Алгоритм трасування є рекурсивним, тому для його завершення вводять граничне значення інтенсивності світла, яке впливає на формування кольору пікселя, або обмежують кількість ітерацій (кількість відбиттів і заломлень).

Зауважимо, що значна частина обчислень при реалізації методу трасування променів припадає на визначення перетинів променів із поверхнями об'єктів сцени. Такий метод складно реалізувати для сцен із значною сукупністю криволінійних об'єктів, обмежених складними параметричними поверхнями. Тому в більшості графічних сцен цей метод працює з об'єктами, що апроксимуються гранями (полігонами), або квадратичними поверхнями, які описуються функціями першого та другого порядків, відповідно. Вибір таких функцій обумовлений необхідністю аналітичного (не числового) розв'язування рівнянь перетину світлового променя з поверхнею.

Для більшої ефективності методу в складних сценах із багатьма об'єктами застосовують *метод оболонок*. Суть цього методу полягає ось у чому. Якщо промінь не перетинає оболонку об'єкта, то він не перетне об'єкт, а якщо промінь перетинає оболонку, то знаходиться точка перетину променя з об'єктом. Як оболонки використовують куб, прямокутний паралелепіпед, циліндр, сферу та інші прості форми.

Факт перетинання променя зі сферою визначається просто, зокрема, якщо віддаль від центра сферичної оболонки до променя більша за радіус цієї сфери, то промінь не перетинає оболонки.

Для складних параметричних поверхонь об'єктів сцени аналіз перетину проводиться для елементів поверхні. Якщо промінь перетинає сферичну оболонку елемента поверхні, то цей елемент розбивається на піделементи. Далі перевіряється перетин променя зі сферичними оболонками піделементів.

Якщо промінь перетинається зі сферичною оболонкою деякого піделемента, то останній розбивається на нові піделементи. Процес завершується, якщо жодна зі сферичних оболонок не перетинається з променем або досягнуто мінімального розміру оболонок. Ці сферичні оболонки мінімального розміру і є шуканими точками перетину променя з поверхнею об'єкта.

Слід зазначити, що метод оболонок можна застосовувати не тільки для трасування променів. Цей метод сприяє прискоренню обчислювальних процесів і часто використовується в різних алгоритмах комп'ютерної графіки.

Контрольні питання та завдання

1. Як обчислюється інтенсивність відбитого світла для дзеркальних та матових поверхонь?
2. Які моделі освітлення використовують у КГ?
3. Як обчислити координати вектора нормалі до параметрично заданої поверхні?
4. Як розрахувати тон зафарбовування точок поверхні?
5. Опишіть метод постійного зафарбовування. Назвіть його недолік.
6. Які ефекти фотореалістичної візуалізації тривимірних моделей не можна одержати методом Фонга?
7. Порівняйте кількість обчислень у методах Фонга та Гуро.
8. Запишіть алгоритм методу зворотного трасування променів.
9. Що є критерієм зупинки методу зворотного трасування променів?
10. Чому метод зворотного трасування більш ефективний, ніж метод прямого трасування?
11. Для чого використовується метод оболонок?
12. Як знайти перетин променя з многогранним об'єктом?
13. Як знайти перетин променя з об'єктом, що заданий аналітичною неявною функцією?

Вправи і задачі для самостійного виконання

1. Обчислити координати заломленого променя.
2. Вивести рівняння нормалі для параметрично заданої сфери.
3. Записати координати вектора нормалі для вершини багатогранної поверхні у скалярній формі.
4. Перевірити чи оболонка-куб перетинається з променем.
5. Розробити алгоритм перевірки перетину променя з гранями тетраедра/прямокутним паралелепіпедом.
6. Запишіть базову операцію зворотного трасування променів у вигляді рекурсивної функції.
7. Розробити алгоритм знаходження точки перетину променя зі сферою.
8. Написати програму для моделювання реалістичної освітленості методом трасування променів у сцені, що містить тільки сфери.
9. Написати процедуру зафарбовування трикутної грані методом Фонга.
10. Довести, що в методі Гуро колір будь-якої точки трикутної грані об'єкта є лінійною комбінацією кольорів у вершинах трикутника з коефіцієнтами, сума яких дорівнює одиниці.
11. Розробити алгоритм накладання текстури $T(x, y)$ на поверхню $P(u, v)$.
12. Розробити алгоритм моделювання туману.

Тема 16. Програмування графіки на OpenGL

16.1. Основні поняття

При створенні графічних додатків зручно користуватися спеціальними графічними бібліотеками. Нині найбільш відомими на ринку вважаються два стандарти для роботи з двовимірною та тривимірною графікою: OpenGL (Open Graphics Library) – стандарт для всіх графічних станцій і DirectX з підсистемою Direct3D – стандарт, запропонований фірмою Microsoft і призначений тільки для Windows. Але стандарт DirectX вважається незручним для практичного застосування, а стандарт OpenGL досить широко розповсюджений і є одним з найпопулярніших серед прикладних програмних інтерфейсів (API) для розробки графічних додатків.

OpenGL являє собою відкритий програмний графічний інтерфейс, він підтримується багатьма операційними системами і придатний для реалізації на різних апаратних платформах – від персональних комп'ютерів до потужних суперкомп'ютерів. Інтерфейс OpenGL реалізований у вигляді бібліотеки функцій, які програміст використовує при написанні інтерактивних програм для створення різних графічних об'єктів (у тому числі й досить складних), операцій над ними і відображення їх на екрані. Доступ до функцій бібліотеки можливий із різних мов програмування, в тому числі C, Delphi, C++, Visual C++, Java та ін.

Цей інтерфейс розроблений у 1992 р. дев'ятьма провідними фірмами з розробки програмного забезпечення, серед яких були Digital, Hewlett Packard, IBM, Intel, Silicon, Microsoft. В основу стандарту покладена бібліотека IRIS GL, розроблена фірмою Silicon Graphics.

Процедури OpenGL працюють як із растровою, так і з векторною графікою і з легкістю дозволяють програмісту:

- 1) створювати 2D- та 3D-вимірні об'єкти довільної форми на основі геометричних та растрових примітивів;
- 2) задавати сплайнові лінії та поверхні, що визначаються опорними точками;
- 3) розміщувати об'єкти в просторі, вибирати спосіб і параметри проєктування, задавати розміщення камери;
- 4) виконувати роботу з кольором об'єктів. Колір може бути заданий як явно (в режимі RGBA та індексному режимі) так і обчислюватися з урахуванням джерел світла, параметрів освітлення. Для об'єктів може задаватися матеріал, накладатися текстура тощо;
- 5) проводити математичні перетворення над об'єктами зображення, наприклад видові та модельні перетворення, виконувати усунення невидимих ліній і поверхонь;

- 6) пересувати джерела світла і камеру за заданими траєкторіями;
- 7) додавати спеціальні ефекти туману, диму, прозорості тощо.

Команди OpenGL реалізовані як модель "клієнт-сервер". Додаток виступає в ролі клієнта – він виробляє команди, а сервер OpenGL інтерпретує і виконує їх. Сам сервер може знаходитися як на тому ж комп'ютері, що й клієнт, так і на іншому.

До характерних особливостей OpenGL, що забезпечили його поширення і розвиток, можна віднести

- 1) *стабільність* (доповнення і зміни в стандарті реалізуються в такий спосіб, щоб зберегти сумісність із розробленим раніше програмним забезпеченням, тобто нова версія OpenGL відбувається тільки за рахунок розширення);
- 2) *надійність і переносимість* (додатки гарантують однаковий візуальний результат незалежно від конкретної апаратної і програмної платформи, типу використовуваної операційної системи. Для перенесення додатків достатньо просто перекомпілювати вихідний текст на новій платформі);
- 3) *легкість і простота* застосування (OpenGL має продуману структуру, інтуїтивно зрозумілий інтерфейс, що дозволяє з меншими затратами створювати ефективні додатки, які містять менше рядків коду в порівнянні з використанням інших графічних бібліотек).

Важливим фактором є також підтримка інтерфейса OpenGL виробниками графічних прискорювачів. Оскільки OpenGL відкритий стандарт, то виробники графічних прискорювачів самостійно можуть добавляти до OpenGL нові можливості, що підтримуються їхніми графічними процесорами (нові можливості зразу стають доступними програмісту).

Апаратна реалізація всіх базових функцій OpenGL забезпечує швидкодію графічних програм, які використовують OpenGL. Ця швидкодія істотно залежить від відеоадаптера. У наш час більшість відеоадаптерів містять спеціальний графічний процесор для підтримки графічних функцій. Відеоадаптер апаратно виконує всі базові функції OpenGL: перетворення координат, розрахування освітленості, накладання текстур, відсікання, виведення полігонів тощо. Тому OpenGL дозволяє достатньо просто створювати швидкодіючі програми і широко використовується на практиці (наприклад, при розробці комп'ютерних ігор).

OpenGL надає користувачу достатньо потужний низькорівневий набір команд, а всі операції високого рівня виконуються в термінах цих команд. Але хоча бібліотека OpenGL має практично необмежені можливості для моделювання і відтворення тривимірних сцен, деякі графічні функції в OpenGL безпосередньо відсутні, наприклад для камери необ-

хідно розраховувати проекційну матрицю, що вимагає додаткових обчислень. Тому для полегшення роботи разом з OpenGL постачаються бібліотеки додаткових команд. На даний момент реалізація Microsoft OpenGL містить кілька бібліотек:

- OpenGL – набір базових функцій (бібліотека `opengl32.dll`) для створення об'єктів та управління їх відображенням на екрані. Тобто OpenGL надає користувачу потужний набір функцій низького рівня, а всі інші операції високого рівня повинні задаватися в термінах цих команд. Імена базових команд починаються з префікса `gl`, а всі константи – з префікса `GL`. Кожне слово, що входить в ім'я функції, починається з великої літери, наприклад `glPolygonMode`;
- Glu-бібліотеку (бібліотека утиліт). Glu-бібліотека є невід'ємною частиною стандарту і поставляється разом з головною бібліотекою OpenGL (вона входить у поставку Windows – це бібліотека `glu32.dll`). Команди цієї бібліотеки доповнюють базові функції OpenGL і полегшують роботу програміста. До складу бібліотеки Glu увійшла множина більш складних функцій, таких як реалізація куба, кулі, циліндра, диска, сплайнових кривих і поверхонь, реалізація додаткових операцій над матрицями тощо. Усі вони реалізовані через базові функції OpenGL. Імена команд бібліотеки утиліт починаються з префікса `glu`;
- Glaux-бібліотеку. OpenGL безпосередньо не підтримує роботу з пристроями введення (миша, клавіатура), оскільки бібліотека платформи-незалежна. Але можна задіяти функції конкретної операційної системи, під яку пишеться програма, або скористатися надбудовами над OpenGL, таким як бібліотека `Glut` (для Unix) і `Glaux` (для Windows). Бібліотека `Glaux` має широкий набір засобів взаємодії з користувачем, вона містить функції для управління вікнами (кількома командами можна визначити вікно, в якому буде працювати Windows), меню, кольорами, палітрою тощо. Крім цього, вона надає додаткові функції для побудови складних графічних об'єктів (конуса, тетраедра, тора тощо). Команди бібліотеки `Glaux` починаються з префікса `aux`.

Щоб програми компілювалися і виконувалися, необхідно підключити ці бібліотеки. Бібліотеки діють через драйвери пристроїв. Це дозволяє ефективно працювати з графічними прискорювачами, не прив'язуючись до їх конкретних особливостей. Одна і та ж сама програма буде успішно працювати на різних прискорювачах без модифікації вихідного коду.

Повне ім'я команди в OpenGL має вигляд:

rtype glCommand_name {1 2 3 4}{b s i f d ub us ui}[v](type1 arg1,type N argN);

де *rtype* – тип, який повертає функція, *gl* – ім'я бібліотеки OpenGL, в якій описана ця функція (для бібліотек Glu, Glut – це імена *glu* та *glut* відповідно), *Command_name* – ім'я команди, *{1 2 3 4}* – кількість аргументів команди, *{b s i f d ub us ui}* – тип аргументу (суфікс): символ *b* означає тип *GLbyte* (аналог *char* у C/C++), символ *f* – тип *GLfloat* (аналог *float*), символ *i* – тип *GLint* (аналог *long*) і так далі (табл. 16.1).

Наявність символу *[v]* вказує, що як параметр функції використовується вказівник на масив значень.

Символи у квадратних дужках у деяких назвах не використовуються. Наприклад, команда *glVertex2i()* описана як базова в бібліотеці OpenGL і використовує як параметри два цілих числа, а команда *glColor3fv(a)* використовує як параметр вказівник на масив *a* із трьох дійсних чисел. Для цього потрібно попередньо визначити масив *a* за допомогою функції *GLfloat a[3]={1.0, 0.0, 0.0}*

Таблиця 16.1

Суфікс	Опис	Тип у C	Тип в OpenGL
b	8-бітне ціле	signed char	GLbyte
s	16-бітне ціле	short	GLshort
i	32-бітне ціле	long	GLint, GLsizei
f	32-бітне число з плаваючою крапкою	float	GLfloat, GLclampf
d	64-бітне число з плаваючою крапкою	double	GLdouble, GLclampd
ub	8-бітне беззнакове ціле	unsigned char	GLubyte, GLboolean
us	8-бітне беззнакове ціле	unsigned short	GLushort
ui	8-бітне беззнакове ціле	unsigned long	GLuint, GLenum
		void	GLvoid

Наведемо приклади деяких функцій OpenGL.

16.2. Вершини і примітиви

Усі геометричні об'єкти задаються в термінах вершин. Під вершиною розуміється точка в просторі. Кожна вершина задається набором чисел – максимум чотири значення (*x, y, z, w*), при цьому можна вказувати два (*x, y*) чи три значення (*x, y, z*), для решти змінних у цих випадках

використовуються значення за замовчуванням: $z = 0$, $w = 1$ (OpenGL працює з однорідними координатами).

Координатні осі розташовані так, що точка $(0, 0)$ знаходиться в лівому нижньому куті екрана, вісь x напрямлена вліво, вісь y – вгору, а вісь z – з екрана на нас.

Однак, щоб задати яку-небудь фігуру, одних координат вершин недостатньо, ці вершини треба об'єднати в одне ціле, тобто необхідно вказати, як їх з'єднати, й визначити необхідні властивості. Для цього в OpenGL використовується поняття примітивів, до яких належать точки, лінії, трикутники, грані тощо. Самі вершини задаються процедурою

```
void glVertex{2 3 4}{s i f d}[v](type x, ...);
```

наприклад,

```
glVertex2s(1, 2);
```

```
glVertex3f(2.3, 1.5, 0.2);
```

```
GLdouble vect[ ]=(1.0, 2.0, 3.0, 4.0);
```

```
glVertex4dv(vect) (тут суфікс  $v$  означає, що єдиним аргументом виступає масив, що містить необхідну кількість координат).
```

Для задання геометричних примітивів необхідно певним чином виділити набір вершин, що визначають цей об'єкт. Для цього використовуються процедури *glBegin()* і *glEnd()*.

Процедура *glBegin(GLenum mode)*;

визначає початок списку вершин, які описують геометричний примітив. Тип примітива (рис. 16.1) задається параметром *GLenum mode*, який може набувати таких значень:

GL_POINTS – набір окремих точок;

GL_LINES – пари вершин, які задають окремі відрізки;

GL_LINE_STRIP – незамкнена ламана;

GL_LINE_LOOP – замкнена ламана;

GL_POLYGON – простий опуклий багатокутник, внутрішня область якого заповнена поточними кольором;

GL_TRIANGLES – трійки вершин, що визначають вершини окремих трикутників;

GL_TRIANGLE_STRIP – трикутний стріп, де кожна наступна вершина задає трикутник разом із двома попередніми;

GL_TRIANGLE_FAN – віяло трикутників (фен) ;

GL_QUAD_STRIP – чотирикутний стріп, в якому чотирикутник з номером n визначається вершинами з номерами $2n - 1$, $2n$, $2n + 2$, $2n + 1$.

Процедура *glEnd()* визначає кінець списку вершин. Між командами *glBegin()* і *glEnd()* можуть знаходитися команди задання різних атрибутів вершин *glVertex()*, *glColor()*, *glNormal()* та ін.

```

Приклад 16.1. Процедура побудови кола
glBegin (GL_LINE_LOOP);
for (int i= 0; i<N; i++)
{
float angle=2*M_PI*i /N;
glVertex2f (R*cos(angle), R*sin(angle));
}
glEnd();

```

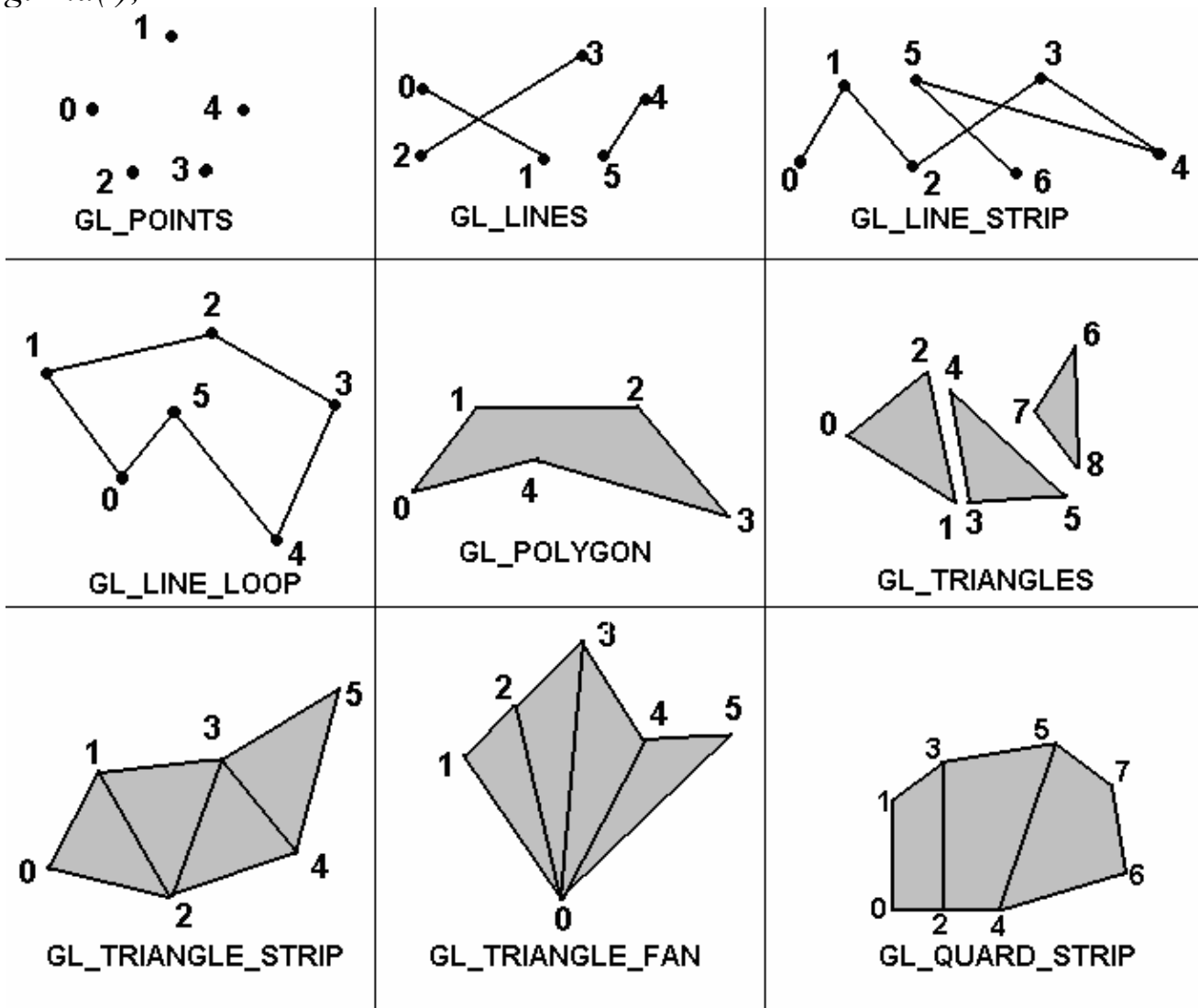


Рис. 16.1. Типи примітивів

Точки кола генеруються при виклику процедури *glVertex()*. Ці точки з'єднуються в замкнену ламану лінію, яка апроксимує коло, – на це вказує параметр *GL_LINE_LOOP*. У момент виклику *glVertex()* OpenGL присвоює створюваній вершині поточний колір, вектор нормалі і т.д. За замовчуванням вектор нормалі вважається таким, що дорівнює (0, 0, 1), колір – (1, 1, 1, 1), координати текстури дорівнюють нулю.

Для задання поточного кольору вершини використовується команда *void glColor {3 4} {b s i f u b u s u i} (Gltypе components);*.

Перші три параметри задають R, G, B-компоненти кольору, а останній параметр визначає alpha-компоненту, що задає рівень прозорості об'єкта. Alpha-коефіцієнт прозорості набуває значення від 0 до 1 для кожного кольорового пікселя. Ефект напівпрозорості створюється шляхом об'єднання кольору вихідного пікселя з пікселем, що вже знаходиться в буфері. В результаті колір точки є комбінацією кольорів переднього й заднього планів.

Якщо в назві команди зазначений тип *f (float)*, то значення всіх параметрів повинні належати відрізьку [0, 1], при цьому за замовчуванням значення alpha-компоненти дорівнює 1.0, що відповідає повній непрозорості. Якщо зазначений тип *ub (unsigned char)*, то значення аргументів повинні лежати на відрізьку [0, 255].

Якщо встановити поточний колір, то всі наступні об'єкти будуть зображатися цим же кольором доти, поки колір не буде змінений.

Різним вершинам можна призначити різні кольори і тоді буде проводитися лінійна інтерполяція кольорів на поверхні примітива.

Наприклад, щоб намалювати трикутник з різними кольорами у вершинах, можна використати такий код:

```
GLfloatBlueCol[3]={0.0, 0.0, 1.0};
glBegin (GL_TRIANGLE);
glColor3f (1.0, 0.0, 0.0); //червоний
glVertex3f (0.0, 0.0, 0.0);
glColor3ub (0, 255, 0); //зелений
glVertex3f (1.0, 0.0, 0.0);
glColor3fv (BlueCol); //синій
glVertex3f (0.0, 1.0, 0.0);
glEnd();
```

Многокутники можна зображати як заповнені області пікселів всередині границі, або рисувати тільки граничну лінію чи зображати набір граничних вершин. Для задання параметрів граничних вершин служить процедура

```
void glPointSize (GLfloat size);
```

що встановлює розмір точки в пікселях (за замовчуванням він дорівнює одиниці). Для задання ширини лінії в пікселях використовують процедуру

```
void glLineWidth (GLfloat width);
```

Многокутник має 2 сторони (грані) – передню (лицеву) і задню (нелицеву) і може бути зафарбований по-різному, в залежності від того, яка зі сторін многокутника повернута до спостерігача. За замовчуванням передньою вважається та сторона, вершини якої обходяться проти годинникової стрілки. Напрямо обходу вершин лицьових сторін можна змінити

викликом команди

```
void glFrontFace (GLenum mode);
```

зі значенням *GL_CW* параметра *mode*, а скасувати – з *GL_CCW*.

Щоб змінити відображення багатокутників, використовується команда

```
void glPolygonMode (GLenum face, GLenum mode);
```

Параметр *mode* визначає, як будуть відображатися багатокутники, при цьому параметр *mode* може приймати одне з таких значень:

GL_POINT – задає режим, при якому відображаються тільки вершини багатокутників;

GL_LINE – при такому режимі багатокутник буде зображатися набором відрізків;

GL_FILL – при такому режимі багатокутники будуть зафарбовуватися поточним кольором з урахуванням освітлення і цей режим установлений за замовчуванням.

Параметр *face* встановлює тип багатокутників, до яких буде застосовуватися ця команда, і може набувати таких значень:

GL_FRONT – для сторін багатокутників, які обходяться проти годинникової стрілки (передніх лицьових граней);

GL_BACK – для задніх нелицьових граней;

GL_FRONT_AND_BACK – для всіх граней.

Наприклад, команда *glPolygonMode (GL_FRONT, GL_FILL)*; зафарбує лицьові грані поточним кольором.

Окрім цього, можна вказати режим, який задаватиме тип граней (лицеві чи нелицеві), що відображатимуться на екрані. Для цього спочатку, викликавши команду *glEnable (GL_CULL_FACE)*, необхідно дозволити відсікання, а потім вибрати тип відображуваних граней за допомогою команди

```
void glCullFace (GLenum mode);
```

Виклик цієї команди з параметром *GL_FRONT* приводить до виведення зображення всіх передніх граней, а з параметром *GL_BACK* – зворотних граней.

Вимикається цей режим функцією *glDisable ()* з тим же параметром.

Лінія або заповнювана грань можуть бути можуть бути нарисовані або одним кольором (*GL_FLAT*), або шляхом інтерполяції кольорів вершин (*GL_SMOOTH*). Для задання режиму зафарбовування використовують процедуру

```
void glShadeModel (GLenum mode);
```

де параметр *mode* набуває двох значень: *GL_FLAT* або *GL_SMOOTH* (задається за замовчуванням).

Приклад 16.2. Інтерполяція кольорів відрізка.

```
glShadeModel(GL_SMOOTH);  
glBegin(GL_LINES);  
glColor3f(0.0, 0.0, 1.0);  
glVertex2i(100, 100);  
glColor3f(1.0, 0.0, 0.0);  
glVertex2i(200, 300);  
glEnd();
```

Якщо вершин багато, то, щоб не викликати для кожної з них команду *glVertex...()*, зручно об'єднувати вершини в масиви, використовуючи команду

```
void glVertexPointer (Glint size,GLenum type, Glsizei stride, void *ptr);,
```

яка визначає спосіб збереження і координати вершин. При цьому параметр *size* визначає кількість координат вершин (це число може дорівнювати 2, 3, 4), а *type* – типи даних (може набувати значень *GL_SHORT*, *GL_INT*, *GL_FLOAT*, *GL_DOUBLE*). Параметр *ptr* вказує на адресу масиву, де знаходяться дані. Параметр *stride* задає зсув координат від однієї вершини до наступної; якщо *stride* дорівнює нулю, то це означає, що координати розташовані послідовно.

Аналогічно визначаються масив нормалей, набір кольорів та масиви інших атрибутів вершин.

16.3. Перетворення координат і проєкцій

У процесі побудови зображень координати вершин постійно піддаються певним перетворенням. Для задання різних перетворень об'єктів сцени використовуються операції над матрицями. Усі вони мають розмір 4×4 і задають перетворення координат згідно із формулою $X' = A \cdot X$.

В OpenGL на різних етапах формування зображень об'єктів використовується три типи матриць. Один із них – матриці моделювання, що служать для задання розміщення об'єкта і його орієнтації в просторі, другий – матриці проєкцій, що задають спосіб проєктування три вимірних об'єктів на площину екрана (OpenGL підтримує 2 види проєктування – паралельне і перспективне). Третій тип – матриці текстури, які визначають накладання текстури на об'єкт.

Для того, щоб вибрати поточну матрицю моделювання, проєктування чи текстури, використовують процедуру

```
void glMatrixMode (GLenum mode);.
```

Параметр *mode* може набувати значення *GL_MODELVIEW*, *GL_PROJECTION* або *GL_TEXTURE*, що дозволяє як поточну матрицю вибрати видову матрицю, матрицю проєктування або матрицю перетворення текстури.

Для задання матриць в OpenGL передбачено декілька функцій. Наведемо деякі з них. Для безпосереднього задання числових значень 16 елементів матриць використовують функції:

```
glLoadMatrix {f d} (const GLtype *m);
```

які копіюють елементи масиву m у поточну матрицю. Цю функцію використовують тоді, коли матриця перетворень програмісту відома.

Команда

```
void glLoadIdentity (void);
```

заміняє поточну матрицю на одиничну.

Для множення зліва поточної матриці на деяку іншу матрицю використовується команда

```
void glMultMatrix{f d} (GLtype *m);
```

де m – задає матрицю розміром 4×4 у вигляді масиву.

Однак звичайно для заміни матриці того чи іншого типу зручно використовувати спеціальні команди, що за значеннями своїх параметрів створюють потрібну матрицю і перемножують її з поточною. Щоб зробити поточною створену матрицю, треба перед викликом певної команди викликати функцію *glLoadIdentity()*.

Для проведення операцій перенесення, повороту і зміни масштабу вздовж координатних осей досить помножити на відповідну матрицю кожену вершину об'єкта й одержати змінені координати цієї вершини, тобто

$$(x', y', z', 1)^T = M * (x, y, z, 1)^T,$$

де M – матриця перетворення.

Сама матриця M може бути створена за допомогою таких команд:

```
void glTranslate{f d} (GLtype x, GLtype y, GLtype z);
```

яка забезпечує зсув об'єкта на вектор (x, y, z) ;

```
void glRotate{f d} (GLtype angle, GLtype x, GLtype y, GLtype z);
```

що здійснює поворот об'єкта на кут $angle$ в градусах у напрямку проти годинникової стрілки навколо прямої з направляючим вектором (x, y, z) ;

```
void glScale{f d} (GLtype x, GLtype y, GLtype z);
```

яка виконує масштабування об'єкта (стиснення та розтяг), домножуючи відповідні координати вершин об'єкта на значення своїх параметрів.

Наведемо приклад застосування цих перетворень до деяких примітивів.

Приклад 16.3. Зсув прямокутника (рис 16.2).

```
glMatrixMode(GL_MODELVIEW);
```

```
//передньо модельна матриця є одиничною
```

```
glColor3f(0.0, 0.0, 1.0);
```

```
glRecti(100, 100, 200, 150);
```

```
//відображається правий синій багатокутник
glColor3f(1.0, 0.0, 0.0);
glTranslatef(-300.0, -50.0, 0.0);
//задаються параметри зсуву
glRecti(100, 100, 200, 150);
```

//відображається червоний зсунутий прямокутник

Якщо після цього в приладі 16.3 захочемо вихідний прямокутник повернути відносно осі z , то поточну матрицю потрібно зробити одиничною і створити матрицю повороту за допомогою команди

```
glRotatef(90.0, 0.0, 0.0, 1.0);.
```

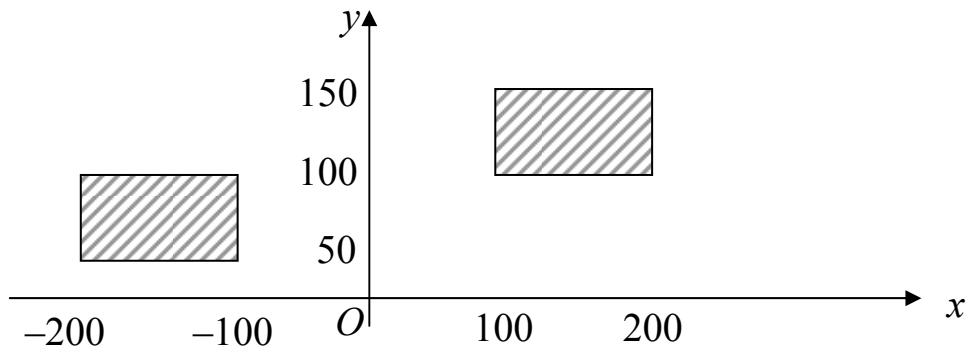


Рис. 16.2. Зсув прямокутника

Далі виконується згортка цієї матриці з поточною одиничною матрицею, і якщо тепер викликати процедуру побудови вихідного прямокутника, то одержимо вже повернутий на 90^0 прямокутник.

Якщо поточну матрицю не робити одиничною, а викликати процедуру `glRotatef(90.0, 0.0, 0.0, 1.0);`, то повернеться вже зсунутий прямокутник, тобто перетворення об'єднаються.

Отже, складні перетворення формуються множенням матриць зміщення, повороту, масштабування та інших перетворень. При цьому варто пам'ятати, що множення матриць некомутативне.

OpenGL містить стек матриць для кожного з трьох типів перетворень, при цьому поточну матрицю можна помістити в стек або зняти матрицю з вершини стека і зробити її поточною. Існують процедури для обробки стеків. Для розміщення поточної матриці в стек служить процедура

```
void glPushMatrix();,
```

для зняття матриці зі стека – процедура

```
void glPopMatrix();.
```

Перетворення проектування визначає, як саме будуть проектуватися об'єкти на екран і які частини об'єкта будуть відсікатись як такі, що не попадають у поле зору. OpenGL підтримує декілька різновидів проєкцій.

У випадку паралельного проектування видимим об'ємом є прямокутний паралелепіпед, який обмежується межами відсікання. Для задання паралельного проектування служить процедура

void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far).

Параметри *left* і *right* визначають координати лівої і правої (вертикальних), *bottom* і *top* – нижньої і верхньої (горизонтальних), а *near*, *far* – передньої і задньої площин відсікання.

Поле зору при перспективному проектуванні є зрізаною пірамідою (рис. 16.3).

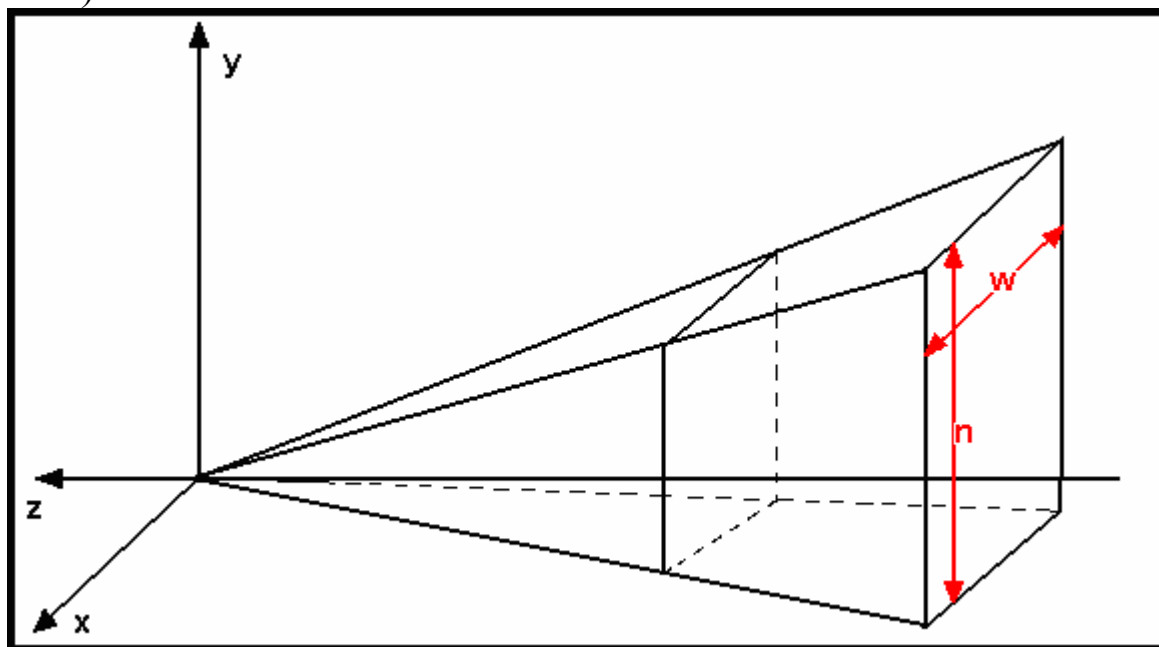


Рис. 16.3. Піраміда видимості при центральному проектуванні

Для задання перспективного проектування в OpenGL створено ряд процедур, наприклад

void gluPerspective (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);

Ця процедура створює матрицю для задання симетричного поля зору і множить поточну матрицю на неї. Значення параметрів процедури: *fovy* – кут зору камери в площині Oxz (лежить у діапазоні від 0^0 до 180^0), *aspect* – відношення ширини області до її висоти ($aspect = w/n$), параметри *zNear* і *zFar* – відстань уздовж від'ємного напрямку осі Oz , що визначає ближню і дальню площини відсікання.

В OpenGL точка огляду спочатку автоматично розташовується в центрі світових координат. Напрямок зору – вздовж від'ємного напрямку

осі z . Щоб зробити можливим огляд об'єктів сцени з будь-якої точки, необхідно сформувати видову матрицю. Матрицю видового перстворення можна сформувати за допомогою функцій *glTranslate*, *glRotate*. Найпростіший спосіб змодельовати камеру, що розташована в точці (x, y, z) і направлена на точку (xp, yp, zp) – це використати функцію *gluLookAt*:

```
glMatrixMode (GL_MODELVIEW); //визначаємо видову матрицю  
glLoadIdentity(); //спочатку одинична матриця  
gluLookAt (x,y,z,xp,yp,zp,1,0,0); //формуємо видову матрицю
```

Тепер точка огляду знаходиться в центрі видової системи координат.

16.4. Функції виведення тривимірних об'єктів

Функції для створення правильних многогранників передбачені в процедурах бібліотеки GLUT. Ця бібліотека пропонує 10 функцій генерування платонових тіл: 5 функцій виводять каркасні об'єкти, а інших 5 – зображають грані як зафарбовані об'єкти. Характеристики зафарбовуваної області визначаються властивостями матеріалу й умовами освітлення.

Наприклад, гексаедр (куб) виводиться за допомогою функцій

```
glutWireCube(edgeLenght); // каркасний куб,  
glutSolidCube(edgeLenght); // куб із зафарбованими гранями.
```

Параметру *edgeLenght* може бути присвоєно будь-яке додатне значення подвійної точності з плаваючою точкою.

Щоб згенерувати поверхні другого порядку з використанням функцій *Glu*, необхідно присвоїти ім'я поверхні, активізувати процедуру візуалізації поверхні, задати значення параметрів поверхні.

Наприклад, наведемо оператори, які ілюструють стандартну послідовність функцій *Glu* для відображення каркасної сфери із центром у початку зовнішньої системи координат:

```
GLUquadricObj *sphere;  
sphere1=gluNewQuadric();  
gluQuadricDrawStyle (sphere1, GLU_LINE);  
gluSphere (sphere1, r, nLongitudes, nLatitudes);
```

Перший оператор визначає назву об'єкта (в нашому випадку це *sphere1*). Потім це ім'я можна використовувати в інших функціях *Glu* для оперування цим об'єктом. За допомогою функції *gluNewQuadric()* активізується процедура візуалізації поверхні другого порядку, після чого за допомогою команди *gluQuadricDrawStyle* для поверхні *sphere1* вибирається режим відображення *GLU_LINE*. В такий спосіб сфера виводиться на екран у каркасній формі з відрізками між кожною парою вершин поверхні. Функція *gluSphere* виконує побудову об'єкта *sphere1* з параметрами: r – радіус сфери (число з плаваючою точкою подвійної

точності), *nLongitudes* – кількість ліній довготи, *nLatitudes* – кількість ліній широти. Ці лінії будуть використовуватися для апроксимації сферичної поверхні чотирикутною сіткою.

Для відображення поверхонь Glu другого порядку доступні три режими відображення. Якщо замість *GLU_LINE* використовувати символічну константу *GLU_POINT* як аргумент функції *gluQuadricDrawStyle*, то поверхня другого порядку відобразиться у вигляді точкового графіка. Якщо задати символічну константу *GLU_FILL*, то сфера буде зафарбованою.

16.5. Функції визначення видимих поверхонь

Усунення задніх граней виконується за допомогою функцій *glEnable (GL_CULL_FACE);* та *glCullFace (mode);*.

При активізації з *glEnable ()* функція *glCullFace ()* вказує, які грані (задні чи передні) будуть задіяні в операціях відбору. Для усунення задніх граней параметру *mode* присвоюють значення *GL_BACK*. Цю ж функцію можна використовувати і для усунення передніх граней, якщо параметру *mode* присвоїти значення *GL_FRONT*.

Для того, щоб використати процедури OpenGL усунення невидимих поверхонь за допомогою методу Z-буфера, спершу необхідно модифікувати функцію ініціалізації GLUT для режиму відображення, щоб вона включала запит до буфера глибини і буфера кадру. Це можна зробити за допомогою оператора

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);.

Значення буфера глибини ініціалізуються за допомогою функції

glClear (GL_DEPTH_BUFFER_BIT);

зі значенням 1.0 або іншим значенням, що задається функцією

glClearDepth(maxDepth);.

Зауважимо, що в OpenGL значення глибини нормуються в діапазоні від 0 до 1.

Процедура визначення видимих поверхонь методом Z-буфера активується функцією

glEnable(GL_DEPTH_TEST);.

Окрім цього, для буфера глибини можна задавати передню та задню площини відсікання.

16.6. Моделювання освітлення

Щоб створити фотореалістичне зображення, необхідно задати властивості поверхні й описати ефекти освітлення. В число цих ефектів входить відбиття світла, прозорість, текстура поверхні, тінь тощо.

Ми вже знаємо, що для розрахунку освітленості точки на поверхні використовують моделі освітлення.

OpenGL використовує модель освітлення, в якій освітлення здійснюється кількома джерелами світла. Окрім цього існує ще й амбітне світло.

Програміст може визначити до восьми джерел світла (в деяких версіях OpenGL і більше восьми) та їх властивості. Для задання цих властивостей використовують процедуру

`void glLight{if}[v](GLenum lightName, GLenum lightProperty, Type param);`, яка задає параметри для джерела світла *lightName*, що може набувати значення *GL_LIGHT0*, *GL_LIGHT1*, ..., *GL_LIGHT7*, тобто кожне джерело світла ідентифікується й налаштовується окремо.

Параметру *lightProperty* присвоюється значення однієї з десяти символічних констант (табл. 16.2).

Таблиця 16.2

Значення	Значення за замовчуванням	Коментарі
<i>GL_AMBIENT</i>	(0, 0, 0, 1)	фонові RGBA-освітленість
<i>GL_DIFFUSE</i>	(1, 1, 1, 1)	дифузна RGBA-освітленість
<i>GL_SPECULAR</i>	(1, 1, 1, 1)	блікова (Фонга) RGBA-освітленість
<i>GL_POSITION</i>	(0, 0, 1, 0)	(<i>x</i> , <i>y</i> , <i>z</i> , <i>w</i>) – координати джерел світла
<i>GL_SPOT_DIRECTION</i>	(0, 0, -1)	(<i>x</i> , <i>y</i> , <i>z</i>) – напрям для конічних джерел світла
<i>GL_SPOT_EXPONENT</i>	(0)	показник степеня у формулі Фонга
...

Джерело світла можна розглядати як таке, що має визначені координати (локальне світло) та світить у всіх напрямках, і як направлене джерело, що знаходиться в нескінченній точці та світить у заданому напрямку. Для ідентифікації джерела світла, тобто для знаходження місцезоміщення джерела світла в OpenGL, використовується символічна константа *GL_POSITION*. При цьому в масив *param* потрібно записати однорідні світові координати джерел світла у вигляді (*x*, *y*, *z*, *w*).

Якщо в команді *GL_POSITION* параметр *w* = 0, то джерело світла направлене і світить у напрямку (*x*, *y*, *z*). Якщо *w* ≠ 0, то джерело світла локальне і знаходиться в точці з координатами (*x/w*, *y/w*, *z/w*).

Для того щоб активізувати процедури освітлення OpenGL використовується команда

`glEnable (GL_LIGHTING);`

Після того, як для джерела світла будуть задані всі властивості, i -те джерело світла вмикається командою `glEnable (GL_LIGHTi);`, а вимикається функцією `glDisable (GL_LIGHTi);`.

Поверхні об'єктів візуалізуються, використовуючи розрахунок освітлення з урахуванням усіх увімкнених джерел світла.

У наступному прикладі світло 1 іде від локального джерела, розміщеного в точці (2.0, 0.0, 3.0), а світло 2 – від направленої джерела, що випромінює світло вздовж від'ємного напрямку осі y :

```
GLfloat light1Type [ ] = {2.0, 0.0, 3.0, 1.0};
GLfloat light2Type [ ] = {0.0, 1.0, 0.0, 0.0};
glLightfv(GL_LIGHT1, GL_POSITION, light1Type);
glEnable(GL_LIGHT1);
glLightfv(GL_LIGHT2, GL_POSITION, light2Type);
glEnable(GL_LIGHT2);
```

Якщо тип і координати джерела світла не задаються, то встановлюється значення за замовчуванням (0.0, 0.0, 1.0, 0.0), яке вказує на віддалене джерело світла, промені якого рухаються вздовж від'ємного напрямку осі z .

Параметри `GL_SPOT_DIRECTION`, `GL_SPOT_CUTOFF` задають джерела світла з кінчною направленістю. Наприклад, команди

```
GLfloat dirVector [ ] = {1.0, 0.0, 0.0};
glLightfv(GL_LIGHT3, GL_SPOT_DIRECTION, dirVector);
glLightfv(GL_LIGHT3, GL_SPOT_CUTOFF, 30.0);
glLightf(GL_LIGHT3, GL_SPOT_EXPONENT, 2.5);
```

для джерела 3 задають направлене кінчне світло так, що вісь конуса проходить уздовж додатного напрямку осі x . Кут конуса дорівнює 30° і параметр затухання становить 2,5.

За замовчуванням задається точкове джерело світла, яке випускає промені в усіх напрямках із нульовим коефіцієнтом затухання.

Затухання інтенсивності можна застосувати до локальних джерел. Для цього використовуються три константи радіального затухання світла: `GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION`, `GL_QUADRATIC_ATTENUATION`.

Наприклад,
`glLightf (GL_LIGHT4, GL_LINEAR_ATTENUATION, 0.75);`

За замовчуванням радіального затухання немає.

Фонове освітлення можна задати за допомогою команди
`glLightModel{if}[v] (GL_LIGHT_MODEL_AMBIENT, ambientColor);`

Наприклад,
`GLfloat ambientColor [] = {0.0, 0.0, 0.3, 1.0};`

glLightModel{if}[v] (*GL_LIGHT_MODEL_AMBIENT*, *ambientColor*);

За замовчуванням у цій команді використовується слабо інтенсивний білий колір (0.2, 0.2, 0.2, 1.0).

Розрахунок дзеркального відображення вимагає знання кількох векторів, в тому числі й вектора V від точки поверхні до точки спостерігача.

У процедурах OpenGL використовується постійний напрям вектора V незалежно від положення точки поверхні відносно спостерігача. Це вектор (0.0, 0.0, 1.0), що йде в додатному напрямі осі z (це значення вектора V за замовчуванням). Тобто вважається, що спостерігач знаходиться в нескінченності й напрям на спостерігача постійний для кожної вершини. Водночас для використання реальної точки спостереження (початок системи координат) для обчислення вектора V використовується команда

glLightModeli (*GL_LIGHT_MODEL_LOCAL_VIEWER*, *GL_TRUE*);

Хоча розрахунок дзеркального відображення при використанні реальної точки спостереження вимагає більше часу, в результаті одержується більш реалістичне зображення.

При виведенні зображення кожного об'єкта OpenGL визначає взаємну орієнтацію вектора напрямку джерела світла й поточного вектора нормалі. Напрямок поточного вектора нормалі не змінюється доти, поки не буде викликана функція нормалі *glNormal3f* (nx, ny, nz);

Оптичні властивості матеріалу, з якого зроблений об'єкт задаються за допомогою процедури

void glMaterial{if}[v] (*GLenum face*, *GLenum property*, *Type param*);

Параметр *face* вказує, для якої зі сторін граней буде застосовуватися ця властивість. Йому присвоюється одна із символічних констант *GL_FRONT*, *GL_BACK*, або *GL_FRONT_AND_BACK*. Параметр *property* вказує на властивості оптичної поверхні, а параметру *param* присвоюються відповідні значення коефіцієнтів відображення (табл. 16.3).

Таблиця 16.3

Значення	Значення за замовчуванням	Коментарій
<i>GL_AMBIENT</i>	(0.2, 0.2, 0.2, 1.0)	фоновий колір матеріалу
<i>GL_DIFFUSE</i>	(0.8, 0.8, 0.8, 1.0)	дифузний колір матеріалу
<i>GL_AMBIENT_AND_DIFFUSE</i>		фоновий і дифузний кольори матеріалу
<i>GL_EMISSION</i>	(0, 0, 0, 1)	колір свічення матеріалу
...

Наприклад,

```
GLfloat EmissionColor [ ] = {0.8, 0.8, 0.8, 1.0};
```

```
void glMaterialfv (GL_FRONT, GL_EMISSION, EmissionColor);
```

задає світло-сірий колір випромінювання з передніх граней. Зауважимо, що це випромінювання поверхні не буде освітлювати інші об'єкти сцени.

Щоб зображення виглядало реалістичним, коефіцієнтам фонового і дифузного освітлення необхідно присвоїти однакові векторні значення.

Це можна зробити, використовуючи символічну константу

```
GL_AMBIENT_AND_DIFFUSE.
```

Метод візуалізації поверхні OpenGL задається за допомогою функції

```
glShadeModel (Rendering Method);
```

Якщо параметру *Rendering Method* присвоїти значення *GL_FLAT*, то матимемо метод постійного зафарбовування, а якщо *GL_SMOOTH* – зафарбовування за методом Гуро (встановлюється за замовчуванням). Зауважимо, що в OpenGL нема процедур зафарбовування за Фонгом.

OpenGL пропонує широкий набір текстурних формул, підтримує одно- і двовимірні текстури і різні способи накладання текстури. Для задання двовимірної текстури використовують функцію *glTexImage2D*.

Це лише короткий опис основних можливостей OpenGL. Більш ґрунтовні знання можна почерпнути з [16],[35],[36],[40].

16.7. Область виведення

Наступним кроком після вибору паралельного або перспективного перетворення є задання області у вікні, в якій буде розміщене побудоване зображення. Для цього створена процедура

```
void glViewport (GLint x, GLint y, GLsizei width, GLsizei height);
```

де (x, y) задає лівий нижній кут прямокутної області у вікні, а *width* і *height* є її шириною і висотою.

OpenGL – універсальна бібліотека, яка може бути реалізована в будь-якому віконному середовищі. Для роботи OpenGL у Windows використовується поняття контексту відтворення, який зв'язує OpenGL з віконною системою координат (контекст пристрою стосується графічних компонент GDI). Щоб почати роботу з командами OpenGL, додаток повинен створити як мінімум один контекст відтворення і зробити його поточним.

Перед створенням контексту відтворення необхідно установити формат пікселів. Для установки формату пікселів використовують функцію

```
int ChoosePixelFormat (HDC, const PIXELFORMATDESCRIPTOR);
```

яка знайде найбільш підходящий формат. Далі необхідно установити цей формат у контексті пристрою за допомогою функції

BOOL SetPixelFormat (HDC hDC, int pixelFormat, const PIXELFORMATDESCRIPTOR);

Для роботи з контекстом відтворення у Windows існують дві функції:

HGLRC wglCreateContext(HDC hdc);

i

BOOL wglMakeCurrent (HDC, HGLRC hGLRC);

Перша з них створює контекст відтворення OpenGL для відтворення на пристрої, що задається контекстом *hDC*, друга встановлює поточний контроль відтворення.

Після закінчення роботи з OpenGL створений контекст відтворення необхідно знищити. Для цього використовують функцію

BOOL wglDeleteContext(HGLRC hGLRC);

Приклад 16.4. Створити проект у середовищі Cbuilder 6.0 для реалізації сфери, що обертається.

Файл Unit 1.ccp

//Програма розроблена Стратійчук Н. О.

#include <vcl.h>

#pragma hdrstop

//Підключаємо потрібні бібліотеки.

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/glaux.h>

#include "Unit1.h"

//-----

#pragma package(smart_init)

#pragma resource ".dfm"*

*TForm1 *Form1;*

*/**

Після підключення всіх файлів необхідно встановити формат пікселів.

Це виконає така функція:

**/*

BOOL bSetupPixelFormat(HDC hdc)

{

*PIXELFORMATDESCRIPTOR pfd, *ppfd;*

//Змінювати в цій функції нічого не потрібно.

int pixelformat;

ppfd = &pfd;

ppfd->nSize = sizeof(PIXELFORMATDESCRIPTOR);

```

ppfd->nVersion = 1;
ppfd->dwFlags = PFD_DRAW_TO_WINDOW |
PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
ppfd->dwLayerMask = PFD_MAIN_PLANE;
ppfd->iPixelFormat = PFD_TYPE_RGBA;

```

/*

iPixelFormat – формат задання кольору. Він може набувати значення *PFD_TYPE_RGBA* (колір вказується чотирма параметрами: червоний, зелений, синій і альфа) і *PFD_TYPE_COLORINDEX* (колір вказується індексом у палітрі);

cColorBits – глибина кольору;

cDepthBits – розмір буфера глибини (Z-буфера);

cStencilBits – розмір буфера трафарета (в програмі не використовується)

*/

```
ppfd->cColorBits = 16;
```

```
ppfd->cDepthBits = 16;
```

```
ppfd->cAccumBits = 0;
```

```
ppfd->cStencilBits = 0;
```

/*

Функція *ChoosePixelFormat()* підбирає формат пікселів згідно із заданими вище умовами і повертає його дескриптор.

Якщо не вдалось підібрати формат відповідно до введених умов, видається повідомлення про помилку.

*/

```
if ((pixelformat = ChoosePixelFormat(hdc, ppfd)) == 0)
```

```
{
```

```
    MessageBox(NULL, "ChoosePixelFormat failed", "Error", MB_OK);
```

```
    return FALSE;
```

```
}
```

/*

Функція *SetPixelFormat()* встановлює формат пікселів.

Якщо не вдалось встановити вибраний формат – помилка.

*/

```
if (SetPixelFormat(hdc, pixelformat, ppfd) == FALSE)
```

```
{
```

```
    MessageBox(NULL, "SetPixelFormat failed", "Error", MB_OK);
```

```
    return FALSE;
```

```
}
```

```
return TRUE;
```

```

}
/*Якщо все нормально, то функція поверне True.
Переходимо у файл Unit 1.h і в області private: додаємо такий код:
private:
HGLRC ghRC;
HDC ghDC;
void Draw();
де ghRC – вказівник на контекст відтворення,
ghDC – дескриптор пристрою (в нашому випадку це просто вказівник
на вікно),
Draw() – наша функція, яка буде щось малювати.
Повертаємось назад у файл Unit 1.cpp. Описуємо функцію FormResize.
Вона буде використовуватись нижче.
*/
void __fastcall TForm1::FormResize(TObject *Sender)
{
glViewport( 0, 0, Width, Height );
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
glOrtho(-5,5, -5,5, 2,12);
gluLookAt(0,0,5, 0,0,0, 0,1,0);
glMatrixMode( GL_MODELVIEW );
/*
glViewport() – встановлює колір області виведення (в нашому випадку –
чорний), яким буде заповнюватися екран при очищенні. Якби в iPixelFormat
було встановлено PDF_TYPE_COLORINDEX, то замість цієї функції
необхідно використати glClearColor(0.0);.
glMatrixMode() – встановлює режим матриці видового перетворення.
Простіше кажучи, якщо потрібно змінити тип проектування, поло-
ження чи напрям камери, то перед цим потрібно викликати функцію
glMatrixMode() з параметром GL_PROJECTION. Після завершення змін
викликаємо її з параметром GL_MODELVIEW.
glLoadIdentity() – замінює поточну матрицю видового перетворення на
одичну.
glOrtho() – встановлює режим ортогонального проектування. 6 пара-
метрів типу Gldouble або просто double: left, right, bottom, top, near, far
визначають координати відповідно: лівої, правої, нижньої, верхньої,
ближньої і дальньої площин відтину, тобто все, що не попадатиме в ці
межі, малюватись не буде. Фактично, цим задаються масштаби
координатних осей.

```

gluLookAt() – встановлює параметри камери: перша трійка – її координати, друга – вектор напрямку, третя – напрям осі Y.

```
*/  
}  
//-----  
void __fastcall TForm1::FormCreate(TObject *Sender)  
{  
    ghDC = GetDC(Handle);  
    if (!bSetupPixelFormat(ghDC))  
        Close();  
    ghRC = wglCreateContext(ghDC);  
    wglMakeCurrent(ghDC, ghRC);  
    glClearColor(0.0, 0.0, 0.0, 0.0);  
    FormResize(Sender);  
    glEnable(GL_COLOR_MATERIAL);  
    glEnable(GL_DEPTH_TEST);  
    glEnable(GL_LIGHTING);  
    glEnable(GL_LIGHT0);  
    float p[4]={3,3,3,1},  
          d[3]={-1,-1,-3};  
    glLightfv(GL_LIGHT0,GL_POSITION,p);  
    glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,d);  
/*
```

Заповнюємо функцію FormCreate.

*Дізнаємося поточний контекст відтворення. Встановлюємо формат пікселя. Якщо цього зробити не вдалося (функція *bSetupPixelFormat()* повернула *false*), закриваємо форму.*

Створюємо за заданим форматом пікселя новий контекст і робимо його активним.

**glClearColor()* – встановлюємо колір фону чорним.*

*Викликаємо функцію *FormResize*, яка встановлює тип проектування і камеру.*

*В *OpenGL* все включається і виключається за допомогою процедур *glEnable()* і *glDisable()*.*

*Таким чином, ми дозволили надавати нашим об'єктам певний колір (*GL_COLOR_MATERIAL*) і дозволили тест глибини (*GL_DEPTH_TEST*), щоб зображення було об'ємним.*

*За допомогою *GL_LIGHTING* ми вибрали освітлення і встановили “лампочку №0” (*GL_LIGHT0*).*

glLightfv() встановлює властивості джерел світла: позицію і напрям світла.

```
*/
}
//-----
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction
&Action)
{
    if(ghRC)
    {
        wglMakeCurrent(ghDC,0);
        wglDeleteContext(ghRC);
    }
    if(ghDC)
        ReleaseDC(Handle, ghDC);
}
/*
```

Після завершення роботи з OpenGL потрібно звільнити зайняті ресурси: звільнити контекст, викликавши wglMakeCurrent із параметром 0 для ідентифікатора контексту OpenGL і знищити цей контекст функцією wglDeleteContext.

Окрім цього, потрібно знищити дескриптор ghDC. Оскільки роботу з OpenGL звичайно завершують по закінченні роботи з програмою, то цей код ми помістили у функцію FormClose().

```
*/
}
/*
```

Тепер ми підходимо до рисування сфери. Спочатку нарисуємо сферу, а потім змусимо її обертатися. Поміщаємо на форму таймер і встановлюємо інтервал таймера на 10 мілісекунд.

```
*/
//-----
void TForm1::Draw()
{
    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
    GLUquadricObj *quadObj;
    quadObj=gluNewQuadric();
    gluQuadricDrawStyle(quadObj, GLU_FILL);
    glColor3f(1,0,0);
    gluSphere(quadObj, 2,10,10);
    glRotatef(3, 0,1,0);
}
}
/*
```

```

    gluDeleteQuadric(quadObj);
    SwapBuffers(ghDC);
}
/*

```

glClear() очищає буфер глибини і буфер кольору.

Для створення сфери використаємо бібліотеку *glu*.

Створюємо об'єкт типу *GLUquadricObj* й ініціалізуємо його функцією *gluNewQuadric()*. Далі встановлюємо стиль фігури функцією *gluQuadricDrawStyle*. Стиль може бути *GLU_FILL* (однотонно залитий об'єкт), *GLU_LINE* (у вигляді каркаса) або *GLU_POINT* (окремі точки).

glColor() встановлює колір об'єкта (наша сфера буде червоною).

gluSphere робить із *quadObj* сферу. Параметри – це радіус сфери і кількість розбиттів уперек і вздовж осі *Z*. Ми взяли невелику кількість розбиттів, щоб було видно як обертається сфера.

glRotate() змушує нашу сферу обертатися.

Звільняємо пам'ять, яку займав об'єкт *quadObj*.

SwapBuffers() – все виводить на екран.

Щоб намалювати щось інше, просто змінюємо функцію *Draw()*.

```
*/
```

Двічі натиснувши мишкою на таймері, отримаємо заголовок функції *Timer1Timer*, з якої викликаємо функцію *Draw()*;

```
*/
```

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
```

```
{
    Draw();
}
```

```
//-----
```

```
/*
```

Остаточно файл *Unit 1.h* виглядатиме так:

```
*/
```

Файл *Unit 1.h*

```
class TForm1 : public TForm
```

```
{
```

```
    __published:
```

```
        TTimer *Timer1;
```

```
        void __fastcall FormCreate(TObject *Sender);
```

```
        void __fastcall FormResize(TObject *Sender);
```

```
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
```

```
        void __fastcall Timer1Timer(TObject *Sender);
```

```
private:
```



```

HGLRC ghRC;
HDC ghDC;
void Draw();
public:
    __fastcall TForm1(TComponent* Owner);
};

```

Контрольні питання та завдання

1. Поясніть різницю між кореневою бібліотекою OpenGL і бібліотеками Glu і Glut.
2. Як в OpenGL задати многокутник списком його вершин/списком його ребер?
3. Яка команда OpenGL використовується для того, щоб задати світло-сірий/чорний колір вікна зображення?
4. Як в OpenGL задати джерела світла та їх властивості?
5. Як в OpenGL задати зміну інтенсивності джерел світла в залежності від віддалі?
6. Як в OpenGL задати джерела світла з кінчною направленістю?
7. Як в OpenGL задати оптичні властивості матеріалу поверхні?
8. Опишіть призначення команди *glLightModel*.
9. Назвіть функції візуалізації поверхні.
10. Наведіть приклад команди для задання кінчних джерел світла.
11. Як задати червоний колір для дифузного і дзеркального освітлення?

Вправи і задачі для самостійного виконання

1. Написати код програми, яка буде зображення п'ятикутної зірки.
2. Написати фрагмент коду для побудови зафарбованого шестикутника. Як модифікувати цей фрагмент, щоб одержати два окремих зафарбованих трикутники?
3. Написати фрагмент програми побудови чотирикутного стріпа з трьох чотирикутників.
4. Написати фрагмент програми, який заповнить внутрішню область многокутника червоним кольором, а його сторони синім кольором.
5. Використовуючи функції для роботи зі стеком замість функції *glLoadIdentity* з прикладу 16.2, виконати зміщення прямокутника.
6. Написати фрагмент коду для присвоєння трьом вершинам трикутника різних кольорів. Сам трикутник зафарбувати за допомогою лінійної інтерполяції кольорів цих вершин.

7. Написати процедуру побудови прямокутного паралелепіпеда з ребрами паралельними координатним осям.
8. Використовуючи функції OpenGL, написати програму повороту багатокутника відносно деякої зовнішньої точки. Вхідними параметрами процедури є вершини багатокутника, координати центра повороту і кут повороту.
9. Реалізувати програму, в якій би тривимірні процедури геометричних перетворень OpenGL застосовувались до тетраедра і в результаті одержувались масштабовані, повернуті та зсунуті фігури.
10. Написати процедуру побудови матриці складного двовимірного перетворення, об'єднавши три матриці базових перетворень (масштабування, повороту і зміщення). Після цього застосувати складне перетворення до трикутника так, щоб трикутник спочатку масштабувався відносно центра ваги, потім був повернутий на кут α і зміщений вздовж вектора $(\Delta x, \Delta y)$.
11. Написати програму візуалізації сферичної поверхні з використанням моделі Фонга для освітлення.
12. Написати програму відображення даного текстурного узору на грань куба/тетраедра/на поверхню сфери.
13. Написати програму для відображення сцени, що містить сферу і тетраедр. Сцена освітлюється двома джерелами світла: одне – локальне червоне джерело, друге – віддалене біле джерело. Задайте параметри поверхні для дифузного і дзеркального відображення. Виконати візуалізацію методом Гуро, застосувавши квадратну функцію затухання інтенсивності.
14. Модифікувати програму з вправи 9, замінивши локальне червоне джерело двома прожекторами – червоним і синім.
15. Написати програму відображення многогранника з усуненням невидимих граней, використовуючи функції OpenGL. Кожна грань многогранника повинна зафарбовуватись різними кольорами. Точка спостереження та інші параметри задаються як вхідні параметри. Модифікувати програму так, щоб многогранник можна було б спостерігати з будь-якої точки.
16. Ознайомтесь з процедурами OpenGL для генерування кривих Безьє. Використовуючи ці процедури, побудуйте різні криві Безьє за чотирма контрольними точками, розрахувавши 100 точок на кривій і з'єднавши їх відрізками. Криву Безьє зобразити синьою ламаною, а контрольні точки вивести на екран, як червоні точки розміру 5.

Завдання до лабораторних робіт

Лабораторна робота №1. Вступні завдання

1. Побудувати траєкторію руху кульки по прямокутному полю. Кут падіння кульки до стінки поля дорівнює куту відбивання, попадання кульки у вершину припиняє рух кульки У полі передбачити наявність статичних та динамічних об'єктів, в нижній частині екрана відобразити результати підрахунку кількості зіткнень.
2. Побудувати багатокутник з заокругленими кутами радіуса R .
3. Побудувати сім'ю дотичних до кола за таким правилом: перша точка A вибирається довільно поза колом і з неї проводиться дотична AP з точкою дотику в точці L так, що $AL = LP$. Потім дотична проводиться з точки P і т.д.
4. Побудувати анімацію дотичних до еліпса, що проходять через задану точку. Розглянути два випадки: а) задана точка рухається вздовж інтерактивно заданої кривої; б) еліпс паралельно зсувається так, що один з його фокусів знаходиться на колі з центром в заданій точці.
5. Побудувати квадрат, який обертається навколо свого центру на кут α , при цьому вершини нового квадрата лежать на сторонах попереднього квадрата. Зобразити систему таких 4×4 квадратів, що складають новий квадрат. Повороти в сусідніх квадратах повинні мати різні напрями.
6. Намалювати візерунок, утворений з'єднанням між собою усіх вершин правильного n -кутника між ($n \leq 25$). Колір діагоналей при цьому змінюється випадковим чином.
7. Побудувати художній візерунок, якщо в куті він будується наступним чином: відрізки сторін кута діляться на N частин; точки поділу з'єднуються за принципом перша з останньою, друга з передостанньою і т.д. За множину кутів узяти кути п'ятикутної зірки.
8. Побудувати систему з трьох зчеплених кілець, кожені два з яких між собою не зчеплені.
9. Написати процедуру, яка за вхідними даними, наприклад за результатами екзаменаційної сесії, будує кругову діаграму/гістограму.
10. Задано квадрат зі стороною R і деяке число N . На сторонах цього квадрата утворюємо $4N$ точок. По одній точці розміщуємо у вершинах квадрата, решту рівномірно розподіляємо по сторонах квадрата так, щоб відстані між двома послідовними точками

дорівнювали R/N . Указані пари точок з'єднуються між собою за правилом: P_i з'єднується з P_j , для яких $j - i \in$ числом Фібоначчі меншим ніж $4N$ (віднімання здійснюється за модулем $4N$). Наприклад, при $N = 10$ точка P_{32} з'єднується з точками $P_{33}, P_{34}, P_{35}, P_{37}, P_{40}, P_5, P_{13}$ та P_{26} . Зобразити такий малюнок (плетене мереживо).


11. Зобразити фігуру, яку описує фіксована точка P , яка лежить на колі, яке котиться по прямій (циклоїда). Розглянути випадки, коли точка P лежить в колі й поза колом і при коченні кола жорстко з ним зв'язана.
12. Побудувати фігуру, яку описує точка кола радіусом r , що котиться по внутрішній стороні заданого кола радіусом R (гіпоциклоїда). Розглянути випадки, коли точка знаходиться всередині внутрішнього кола і поза колом. Точка з рухомим колом зв'язана жорстко.
13. Побудувати фігуру, яку описує точка кола радіусом r , що котиться по зовнішній стороні заданого кола радіусом R (епіциклоїда). Розглянути випадки, коли точка знаходиться в колі й поза колом, яке рухається. Точка з рухомим колом зв'язана жорстко.
14. Продемонструвати рух двох кіл різних радіусів, що котяться по горизонтальній прямій. Зупинка руху кіл відбувається в момент їх дотику. Передбачити випадки однонаправленого та різнонаправленого руху кіл.
15. Продемонструвати рух двох кіл різних радіусів, що котяться назустріч по зовнішній/внутрішній стороні заданого кола радіуса R . Зупинити рух кіл в момент їх дотику.
16. Зобразити паркет (покриття площини без пропусків), якщо у вершині стику знаходяться правильні 6-, 4-, 3-кутники.
17. Замостити паркет (вкрити площину без пропусків) правильними 4-, 6-, 12-кутниками.
18. На горизонтальній прямій задано дві точки, відстань між якими дорівнює c . Зобразити множину точок, для яких добуток відстаней до двох заданих точок дорівнює p . Розглянути такі випадки: 1) $p = \frac{c^2}{4}$; 2) $p < \frac{c^2}{4}$; 3) $\frac{c^2}{4} < p < \frac{c^2}{2}$; 4) $p > \frac{c^2}{2}$.
19. На площині задано m точок F_1, F_2, \dots, F_m . Зобразити множину точок, для яких $\prod_{i=1}^m |MF_i| = p$, де p – наперед задане додатне число.

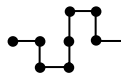
20. На прямій задано m точок. Написати програму, яка знайде на цій прямій точку, сума відстаней від якої до даних точок є мінімальною. Узагальнити задачу на випадок площини.
21. Побудувати діаграму Вороного на множині довільно розташованих на площині опорних точок. *Вказівка:* діаграма Вороного – це об'єднана межа многокутників Вороного, створених для кожної з опорних точок. Многокутником Вороного опорної точки A називається множина всіх точок площини, відстані від яких до точки A менші, ніж до іншої опорної точки.
22. На площині задано n -кутник (опуклий чи неопуклий). Зобразити розбиття полігона на трикутники.
23. Задано дві півплощини. Побудувати їхній перетин. За півплощину вибираємо ту, яка залишається справа від прямої, якщо рухатися вздовж цієї прямої.
24. На площині задано n точок. Зобразити опуклу оболонку цих точок та знайти її периметр. *Вказівка:* опукла оболонка точок це – опуклий полігон мінімальної площі, який містить ці точки.
25. Побудувати пряму, яка найменше відхиляється від заданих точок P_1, P_2, \dots, P_n .
26. Написати процедуру визначення розміщення точки відносно напрямленого відрізка (спереду, позаду, зліва, справа, на відрізку).
27. Поділити екран на 8 прямокутників (2×4). У верхніх чотирьох прямокутниках побудувати графік функції $\rho = a \cos(k\varphi)$, а в нижніх – $\rho = a \sin(k\varphi)$, якщо: 1) $k = 2, a = 1$; 2) $k = 4, a = 1$; 3) $k = 3, a = 2$; 4) $k = 8, a = 4$.
28. Написати програму виведення графіка функції $y = f(x)$ на весь екран, якщо $x \in [x_{\min}, x_{\max}]$. Зліва і знизу екрану виводити осі координат. Екран поділити m вертикальними та n горизонтальними лініями.
29. Скласти програму, що відображає параметрично задану криву
$$\begin{cases} x = (2 + 7 \cos(\sin \theta + \sin 121\theta)) \cos \theta, \\ y = (2 + 7 \cos(\sin \theta + \sin 121\theta)) \sin \theta, \frac{\pi}{2800} \leq \theta \leq 2\pi, \end{cases}$$
вибравши однаковий масштаб по Ox і Oy . Передбачити виведення координати точки і параметра θ при натисканні правої клавіші миші.
30. Написати процедуру знаходження перетину двох довільних трикутників, заданих на площині.
31. Дано два трикутники, які не перетинаються. В кожному з цих трикутників вибирається по одній точці. Перший трикутник почи-

- нається рухатись до другого паралельно вектору, що з'єднує ці точки. Зупинити рух трикутника в момент його дотику до другого трикутника. Узагальнити задачу для довільного полігона.
32. Задано квадрат і пряму, які не мають спільних точок. Через центр квадрата проведено перпендикуляр до прямої. Квадрат, обертаючись навколо свого центру проти годинникової стрілки, рухається до прямої так, що центр квадрата залишається на перпендикулярі. Зупинити рух квадрата в момент його дотику з прямою.
 33. Побудувати зображення платонових тіл: 1) гексаедра (куба), 2) тетраедра, 3) октаедра, 4) додекаедра, 5) ікосаедра.
 34. Розробити тест перетину прямої лінії $Ax + By + C = 0$ з полігоном $P = \{P_1, P_2, \dots, P_n, P_1\}$.
 35. Задано трикутник, одна з вершин якого рухається по інтерактивно введеної кривій (в простішому випадку по прямій). Побудувати анімацію кіл: а) вписаних в рухомий трикутник; б) описаних навколо рухомого трикутника.
 36. Розробити анімаційну програму, яка малює динамічні кольорові візерунки за допомогою радіальних багатокольорових ліній.
 37. Розробити анімаційну програму, яка виконує кольорову анімацію набору різнокольорових концентричних кіл так, що за рахунок зміни кольорів та плавної зміни розмірів кіл створити ілюзію руху.
 38. Розробити анімаційну програму, яка реалізовує виведення символів у задану позицію екрану та анімацію цього тексту (динамічна зміна його положення, орієнтації в просторі, розмірів).
 39. Написати програму, яка дозволяє у вікні прокручувати текст подібно до титрів у кінці фільму.

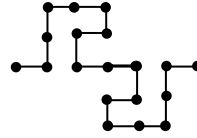
Лабораторна робота № 2. Фрактали

Побудувати наступні конструктивні та динамічні фрактали. *Вказівка:* конструктивні фрактали на площині створюються за допомогою деякої ламаної, яка називається генератором. За один крок алгоритму кожен із відрізків ламаної замінюється на ламану-генератор у відповідному масштабі. У результаті нескінченного (на практиці скінченного) повторення цього алгоритму одержується геометричний фрактал. Для конструктивного фрактала характерно задання основи та генератора, який повторюється при зменшенні масштабу. Динамічні фрактали отримуються за допомогою рекурентних формул.

1. Фрактал Коха. Основа – відрізок. Генератор . Усі ланки генератора мають однакову довжину.

2. Фрактал Мінковського. Основа – відрізок. Генератор . Усі ланки ламаної мають однакову довжину.

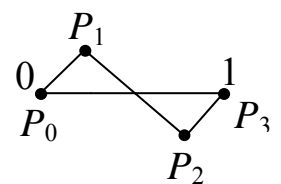
3. Фрактал Леві. Основа – одиничний відрізок. Генератор (ламана з проміжною точкою (0,5; 0,5)), тобто відрізок заміняємо половиною квадрата.



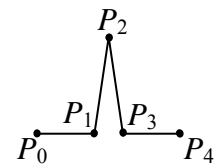
4. Основа – відрізок. Генератор . Усі ланки ламаної мають однакову довжину.

5. Острів Коха. Основа – правильний трикутник. Генератор Коха (задача 1) орієнтований всередину/назовні трикутника.

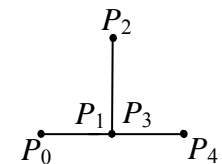
6. Острів Мінковського. Основа – одиничний квадрат. Генератор – ламана з проміжними точками:
а) $P_1(0,25; 0,25)$, $P_2(0,75; -0,25)$; б) $P_1(0,3; 0,3)$, $P_2(0,7; -0,3)$.



7. Різаний квадрат. Основа – одиничний квадрат. Генератор – ламана з проміжними точками $P_1(0,47; 0)$, $P_2(0,5; 0,47)$, $P_3(0,53; 0)$.



8. Льодовий квадрат. Основа – одиничний квадрат. Генератор – ламана з проміжними точками $P_1(0,5; 0)$, $P_2(0,5; 0,33)$, $P_3(0,5; 0)$.



9. Льодовий трикутник 1. Основа – правильний трикутник. Генератор – ламана із задачі 8.

10. Льодовий трикутник 2. Основа – рівносторонній трикутник із вершинами (0, 0), (0,5; 0,85), (0, 1). Генератор – ламана з проміжними точками $P_1(0,5; 0)$, $P_2(0,375; 0,2165)$, $P_3(0,5; 0)$.

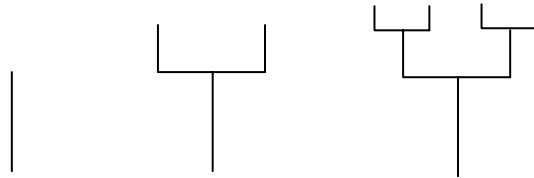
11. H-фрактал. Основа – горизонтальний відрізок. Через кінці горизонтального відрізка перпендикулярно до нього проводяться два коротших відрізки. На кінцях цих вертикальних відрізків будуються горизонтальні відрізки у зменшеному масштабі і т.д.

Коефіцієнт зменшення $\frac{1}{3}$.

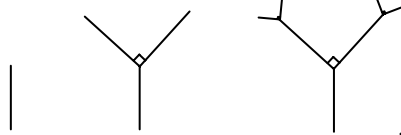
12. У колі радіусом R проведено 10 радіусів, кут між ними дорівнює $\frac{\pi}{5}$. На кінцях цих радіусів побудовано нові кола радіусом $\frac{R}{2}$, в яких так само проведено 10 радіусів і т.д.

13. Дерево Мандельброта. Основа – відрізок. Генератор \sqsupset .
Кожний з відрізків замінюється генератором у зменшеному масштабі. Коефіцієнт зменшення $\mu \in (0, 1)$.

14. Двійкове дерево. Основа – відрізок (стовбур). Генератор \sqsubset .
Перші наближення мають вигляд:



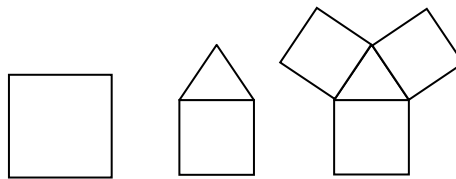
15. Оголене дерево Піфагора. Основа – відрізок (стовбур). Генератор \sphericalangle . Перші наближення мають вигляд:



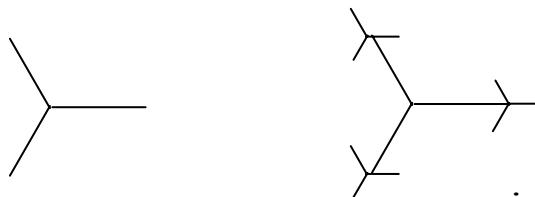
16. Реалістичне дерево. Основа – відрізок. Генератор \sphericalangle .

17. Кущова структура. Основа – відрізок. Генератор \sphericalangle .

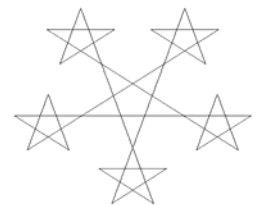
18. Дерево Піфагора. Основа – квадрат. Генератор – правильний трикутник. Перші наближення мають вигляд:



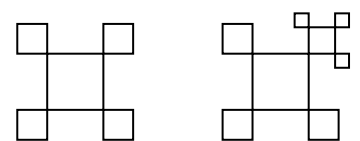
19. Трійкове дерево (дендрит). З однієї точки під кутом 120° виходять три відрізки. Кожен із кінців цих відрізків є точкою, звідки також виходять три менших відрізки і т.д. Перші наближення:



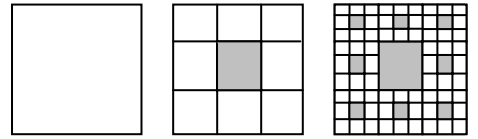
20. Зірковий фрактал. У вершинах правильної п'ятикутної зірки будується 5 менших зірок, далі на вільних кінцях цих зірок будуються ще менші зірки і т.д. Взяти коефіцієнт зменшення $r = 0,35$, а кількість кроків рекурсії $n = 5$.



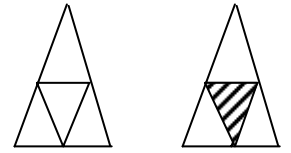
21. Зірковий квадратичний фрактал. У вершинах квадрата будуються 4 менших квадрати, на трьох вільних вершинах цих квадратів будуються ще менші квадрати і т.д.



22. Квадрат Серпінського. Заданий квадрат ділиться двома горизонтальними і двома вертикальними лініями на 9 квадратів. Одержаний центральний квадрат зафарбовуємо, а решту вісім квадратів перетворюємо як перший і т.д.



23. Трикутник Серпінського. У рівнобедреному трикутнику проводимо середні лінії. Трикутник, що утворений середніми лініями, зафарбовуємо, а інші трикутники перетворюємо аналогічно до першого і т.д.



24. Гілка папороті з параметрами: $\alpha = 2^0$, $\beta = 86^0$, $k = 0,14$, $k_1 = 0,3$.
 25. Фрактал Жулія. Множина Жулія задається рекурентною формулою

$$z_{n+1} = z_n^2 + c, \quad n = 0, 1, 2, \dots, \quad z = x + iy, \quad c = a + ib, \quad i = \sqrt{-1}, \text{ або}$$

$$x_{n+1} = x_n^2 - y_n^2 + a, \quad y_{n+1} = 2x_n y_n + b, \quad n = 0, 1, 2, \dots,$$

де z_0 – різні точки комплексної площини, $c \neq 0$ – фіксована комплексна точка. Для побудови ефектних зображень вибрати $z_0 \in [-2, 2; 1] \times [-1, 2; 1, 2]$ і $a = -0,22$; $b = -0,74$ та $a = 0,11$; $b = 0,66$.

26. Фрактал Мандельброта. Множина Мандельброта задається рекурентною формулою

$$z_{n+1} = z_n^2 + c, \quad n = 0, 1, 2, \dots, \quad z = x + iy, \quad c = a + ib, \quad i = \sqrt{-1}, \text{ де } z_0 -$$

фіксована точка, а $c \in \mathbb{C}$. За вихідні параметри взяти $z_0 = 0$, $a \in [-2,5; 2,5]$, $b \in [-2, 2]$. Передбачити вивід на екран будь-якої прямокутної області цього фрактального зображення.

27. Фрактал, що визначається рекурентною формулою $z_{n+1} = \lambda z_n(1 - z_n)$, $n = 0, 1, 2, \dots$, $z = x + iy$. Для z_0 вибрати область $[-2, 2; 1, 2] \times [-1, 2; 1, 2]$. Критерій завершення ітераційного процесу: $x_k^2 + y_k^2 > 5$.

28. Фрактал Ньютона. Фрактал Ньютона задається формулою

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}. \text{ Якщо } f(z) = z^k - a, \text{ тоді}$$

$$z_{n+1} = \frac{(k-1)z_n^k + a}{kz_n^k}, \quad n = 0, 1, 2, \dots$$

Розглянути варіанти: 1) $k = 3$, $a = 1$; 2) $k = 5$, $a = 1$.

29. Фрактал, який визначається рекурентними співвідношеннями

$$x_{k+1} = a_1 x_k^2 + a_2 y_k^2 + a_3 x_k y_k + a_4 x_k + a_5 y_k + a_6,$$

$$y_{k+1} = b_1 x_k^2 + b_2 y_k^2 + b_3 x_k y_k + b_4 x_k + b_5 y_k + b_6.$$

Підібрати значення параметрів a та b , так щоб одержати ефектні кольорові фрактальні зображення. Колір визначається кількістю ітерацій. Критерій завершення ітераційного процесу: $x_k^2 + y_k^2 > 10$.

Лабораторна робота №3. Растрові алгоритми. Сплайнові криві. Алгоритми відсікання

1. Скласти програму, яка реалізує алгоритм Брезенхема для відрізка прямої лінії з відповідною структурою.
2. Написати програму, яка демонструє роботу алгоритму Брезенхема для відрізка в сповільненій формі та в збільшеному виді.
3. Адаптувати алгоритм Брезенхема для відрізка прямої лінії з заданою товщиною лінії. *Вказівка:* для побудови широкої лінії можна перерисовувати зміщений квадрат відповідної орієнтації.
4. Реалізувати програму для побудови прямокутника з заокругленими кутами радіусом r , використавши в програмі алгоритм Брезенхема для побудови кола.
5. Реалізувати растровий алгоритм Жордана для побудови кривих другого порядку.
6. Реалізувати порядковий алгоритм заповнення з затравкою для багатокутних областей з діркою. Проілюструвати роботу алгоритму поетапно (в залежності від кількості кроків циклу).
7. Реалізувати алгоритм заповнення багатокутної області за критерієм парності.
8. Реалізувати УХ-алгоритм зафарбовування полігонів
9. Заповнити багатокутник похилими лініями.
10. Заповнити багатокутник текстурою. Зразок візерунка для тексела підібрати самостійно.
11. Реалізувати алгоритм для побудови інтерполяційного кубічного сплайна на сітці з трьох вузлів. Розглянути граничні умови першого типу.
12. Створити програму, яка дозволяє будувати згладжуючу криву Безьє на чотирьох точках (розглянути різний порядок точок).
13. Написати програму для побудови раціональної кривої Безьє на чотирьох точках. Експериментуючи з програмою, вивчити властивості раціональних кривих Безьє.
14. Написати програму побудови складеної кривої Безьє, передбачивши інтерактивне введення точок.
15. Написати програму, побудови B-сплайнової кривої на масиві чотирьох точок, та складеної кривої на масиві m точок.

16. За заданими вершинами P_0, P_1 та ненульовими векторами дотичних Q_0, Q_1 в цих вершинах побудувати інтерполяційну кубічну криву Ерміта. Передбачити інтерактивне введення даних.
17. Реалізувати геометричний алгоритм побудови кривих Безьє.
18. Розробити програму, яка реалізує побудову поверхні Безьє.
19. Реалізувати алгоритм побудови кубічних B-сплайнів.
20. Реалізувати програму побудови TCB-сплайнів.
21. Написати програму, яка виводить на екран просторову криву Безьє для набору з чотирьох контрольних точок. Для зображення кривої використати ортогональну проекцію.
22. На множині 6 точок побудувати криву Безьє та B-сплайн. *Вказівка*: опорні точки вибирати так, щоб контурна ламана мала профіль пили з різними зубами.
23. Реалізувати алгоритм Сазерленда-Коена для відсікання відрізків прямокутним вікном.
24. Реалізувати алгоритм відсікання багатокутників прямокутним вікном. *Вказівка*: послідовно виконати відсікання відносно всіх чотирьох границь прямокутного вікна.
25. Реалізувати двовимірний FC-алгоритм для відсікання відрізків.
26. Реалізувати алгоритм Кіруса-Бека для відсікання прямокутним вікном.
27. Реалізувати алгоритм Вейлера-Азертонна для відсікання багатокутним вікном.
28. Написати процедуру відсікання еліпса прямокутним вікном.
29. Написати програму для реалізації методу загортання подарунка.
30. Написати програму для реалізації методу обходу Грехема.
31. Написати програму тріангуляції опуклих та неопуклих полігонів.
32. Написати програму реалізації кутового тесту для визначення орієнтації точки відносно полігону.
33. Розробити програму, яка для напівтонового растрового зображення дозволить покращити його якість, добиваючись при цьому найбільшої збалансованості яскравості та контрастності зображення.
34. Написати програму, яка з растрового зображення дозволить усунути випадковий шум за допомогою медіанних фільтрів.
35. Написати програму, яка реалізовує двопрохідний/трипрохідний алгоритм геометричних перетворень для повороту растрових зображень на кут $\alpha \neq 90^\circ$.
36. Розробити програму для реалізації просторових алгоритмів обробки растрових зображень (розмиття, збільшення різкості, тиснення). Підібрати відповідні фільтри.

Лабораторна робота № 4. Візуалізація тривимірних об'єктів

1. Розробити алгоритм побудови симетрії трикутника відносно довільної прямої L . На екран виводити зображення результатів послідовного виконання кроків алгоритму.
2. Написати програму для відображення двовимірних об'єктів, кожний з яких вибирається з меню. Передбачити можливість їх масштабування та обертання.
3. Реалізувати алгоритм тривимірного повороту зрізаного куба зі стороною a навколо однієї з осей координат.
4. Розробити програму для реалізації процесу обертання паралелепіпеда $ABCDEFGH$ відносно локальної осі, що проходить через:
 - а) центр паралелепіпеда;
 - б) одну з вершин паралелепіпеда.

Координати вершин паралелепіпеда задаються матрицею

$$X = \begin{pmatrix} 1 & 1 & 2 & 1 \\ 2 & 1 & 2 & 1 \\ 2 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 \\ 2 & 2 & 1 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{matrix}$$

5. Розробити програму для реалізації обертання тетраедра/куба/октаедра відносно рухомої осі паралельної осі Ox , яка пересувається фіксованою траєкторією.
6. Побудувати триметричну проекцію зрізаного куба для $\psi = 15^\circ, 30^\circ, 45^\circ, \varphi = 0^\circ, 15^\circ, 30^\circ, 45^\circ, 60^\circ, 75^\circ, 90^\circ$.
7. Побудувати диметричну проекцію зрізаного куба, взявши для кутів ψ, φ такі значення: $\psi = \pm 60^\circ, \varphi = \pm \arctg\left(\frac{\sqrt{3}}{2}\right) = \pm 40.9^\circ$.
8. Побудувати одноточкову та двоточкову центральну проекцію двох різноорієнтованих кубів в одній сцені.
9. Побудувати аксонометричну проекцію куба з вписаними в його грані колами при співвідношенні масштабів $k_x : k_y : k_z = 6 : 5 : 4$.
10. Побудувати чотири можливі ізометричні проекції зрізаного куба.

Вказівка: ізометрична проекція одержується при кутах

$$\psi = \pm 45^\circ, \varphi = \pm \arccos\left(\frac{2}{3}\right) = \pm 35,26^\circ.$$

11. Побудувати стандартну диметричну проекцію зображення глобуса радіуса r , що містить паралелі з кроком 30° , меридіани з кроком 45° і контур глобуса. Реалізувати нахил зображення вправо відносно осі z на кут $\varphi = 23,5^\circ$. *Вказівка:* перетворення нахилу глобуса здійснюється матрицею повороту

$$R_z(-\varphi) = \begin{pmatrix} 0,917 & -0,4 & 0 \\ 0,4 & 0,917 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

12. Побудувати проекцію Кавальє зрізаного куба з кутом $\beta = 45^\circ$, а кут α змінюється від -30° до 30° з кроком 15° .
13. Побудувати кабінетну косокутну проекцію для зрізаного куба з кутом α , який змінюється від -30° до 30° з кроком 15° .
14. Підібравши кути α та β , побудувати косокутну проекцію сцени, яка складається з модельного куба з вписаними в його грані колами, якщо кола, що лежать у площині, яка паралельна до картинної площини, проектуються в кола, а кола з непаралельних граней до картинної площини проектуються в еліпси з відповідною орієнтацією півосей.
15. Написати програму обертання двох платонових тіл навколо деякої осі з усуненням невидимих ребер. В роботі використати одноточкову перспективну проекцію та алгоритм художника для усунення невидимих граней.
16. Побудувати одноточкову проекцію куба на площину $z = 0$, попередньо змістивши його вздовж прямої $y = x$.
17. Побудувати в одній сцені двоточкову проекцію куба, тетраедра та октаедра.
18. Розробити програму усунення невидимих граней для центральної проекції октаедра. Видимі грані зафарбовувати зеленим кольором.
19. Написати програму зображення перетину двох трикутників в просторі. Передбачити обертання такого об'єкта навколо осей координат.
20. Розробити програму обертання куба відносно заданої лінії з усуненням невидимих граней. Видимі грані зафарбовувати у світло-синій колір. *Вказівка:* для усунення невидимих граней можна використати орієнтацію зовнішньої нормалі.
21. Розробити програму відображення даного текстурного зразка на грані куба.

22. Методом поточного горизонту побудувати такі поверхні:
- 1) $y = f(x, z) = e^{-a(x^2+z^2)} \cos(\omega_x x) \cos(\omega_z z)$, де $a = 0,02$, $\omega_x = 1$, $\omega_z = 0,5$;
 - 2) $z = f(x, y) = \frac{\cos(b\sqrt{x^2 + y^2})}{1 + c\sqrt{x^2 + y^2}}$ (c, b підібрати самостійно).
23. Написати програму динамічного повороту 3D-графіка функції.
24. Реалізувати алгоритм постійного зафарбовування видимої поверхні для октаедра. Невидимі грані усунути, використовуючи алгоритм художника.
25. Реалізувати алгоритм Гуро для зафарбовування граней тетраедра та шестикутної призми при наявності одного джерела світла. Необхідні константи для моделі освітлення підібрати самостійно.
24. Реалізувати алгоритм Z-буфера для заданого об'єкта. Написати процедуру виведення грані в Z-буфер. Розмір масиву, що використовується для буфера глибини вибрати згідно з характеристиками системи.
25. Розробити програму, яка усуває невидимі грані октаедра на основі методу видалення нелицьових граней многогранників. Передбачити обертання цього октаедра навколо однієї з координатних осей. *Вказівка:* для побудови проекції об'єкта необхідно однорідні координати об'єкта помножити на відповідну матрицю проектування. Для моделювання процесів обертання, перенесення та масштабування використати відповідні матриці цих перетворень.
26. Написати програму анімації опуклого многогранника відносно осі, що проходить через об'єкт паралельно до картинної площини. Припускається, що об'єкт цілком лежить перед картинною площиною. Для послідовного відображення проекцій використати ортографічну проекцію.
27. Реалізувати алгоритм зафарбовування трикутної грані методом Фонга, вважаючи, що джерело світла спрямоване в центр трикутника.
28. Написати процедуру зафарбовування трикутної грані методом Гуро, вважаючи, що джерело світла спрямоване в центр трикутника. *Вказівка:* метод Гуро для полігональних моделей передбачає визначення освітленості в кожній вершині полігона й подальшу лінійну інтерполяцію освітленості для кожної точки грані.
29. Написати процедуру для зображення граней тетраедра з дифузним відображенням і одним джерелом світла.

Комп'ютерні проекти

1. Написати програму апроксимації сфери методом рекурсивного розбиття тетраедра в паралельній проекції. Зафарбувати сферу методом Фонга. Передбачити в програмі інтерактивне розміщення в просторі сцени джерела світла та камери спостереження.
2. Написати програму апроксимації сфери меридіанами та паралелями, використавши її параметричне рівняння. Розробити алгоритм накладання BMP-малюнків на поверхню сфери, наприклад карти земної кулі. Передбачити інтерактивне обертання сфери навколо своєї осі та зміну нахилу самої осі. Для відображення сцени обрати паралельну проекцію.
3. У будь-якій системі тривимірного геометричного моделювання, наприклад 3DStudio MAX чи Maya, створити сцену з площини, на якій розміщені примітивні тіла (паралелепіеди, сфери, конуси, циліндри, еліпсоїди тощо), задати три різні джерела світла і розмістити їх так, щоб в сцені були присутні тіні і взаємні відображення світла від створених об'єктів. Виконати накладання різних матеріалів на поверхні тіл та реалізувати ефект прозорості.
4. Реалізувати програму побудови сплайнових кривих. Вхідні дані: набір контрольних точок, що задаються інтерактивно. При додаванні або усуненні контрольної точки сплайн повинен автоматично перемальовуватися. Система координат має бути підписана та виведена координатна сітка. Передбачити як сумісне так і окреме виведення таких сплайнових кривих: криві Безьє, складені кубічні B-сплайнові криві, NURBS-криві, кусочно-кубічні Ермітові сплайни (ТСВ-сплайни).
5. Написати програму моделювання криволінійних поверхонь за допомогою елементарних фрагментів поверхонь Безьє, бікубічних B-сплайнових поверхонь та NURBS-поверхонь. Функція двох змінних задається своїми значеннями в вузлах прямокутної сітки. Значення функції в вузлах вводити за допомогою клавіатури.
6. Розробити проект для моделювання платонових тіл. Передбачити інтерактивний вибір їх типу, розміру і місцерозміщення, способу зображення, можливості задавати вектор обертання та кутову швидкість тощо. В проекті використати одноточкове центральне проектування та передбачити вибір видової системи координат.
7. Створити проект малювання поверхонь другого порядку. Передбачити задання обмежуючого паралелепіеда для перегляду нескінченних поверхонь, його переміщення, зміну розмірів та обертання. Поверхні вибирати з меню. Для зображення поверхонь використати ортогональні проекції та апроксимацію лініями.

8. Розробити проект для малювання конструктивних, динамічних і статистичних фракталів. Для динамічних фракталів передбачити вибір та масштабування прямокутної області фрактала.
9. Розробити програму візуалізації апроксимації 3D-поверхонь у наступній постановці. Задано дискретну множину значень невідомої функції від двох змінних у N довільно розташованих на 2D-площині опорних точках. Зобразити кусково-лінійну апроксимацію 3D-поверхні методом тріангуляції. Для цього спочатку на 2D-площині створити сітку неперетинних трикутників з вершинами в опорних точках, тобто виконати тріангуляцію Делоне. Побудувати апроксимацію кусково-лінійної 3D-поверхні, що складається з трикутних граней з вершинами у відповідних точках. При цьому використати паралельну проекцію та алгоритм сортування граней за глибиною для усунення невидимих граней. Координати спостерігача задавати інтерактивно.
10. Розробити діалогову програму, яка зображатиме лінійчаті, секторні та циліндричні 3D-поверхні. Як приклад лінійчатих поверхонь розглянути гвинтові поверхні. Напрямні лінії для секторних поверхонь вибрати серед просторових кривих Безьє. Каркас одержаної поверхні зобразити в паралельній проекції з усуненням невидимих ліній. Координати спостерігача задавати інтерактивно.
11. Розробити програму візуалізації 3D-полігональної моделі з нанесенням текстури. За текстуру вибрати растровий образ, відображений на грані об'єкта. Проекція – паралельна. Світло – паралельний пучок. Модель освітлення – з дифузними і дзеркальними складовими.
12. Розробити проект відсікання багатокутників прямокутним та багатокутним вікном. Передбачити інтерактивний вибір методу відсікання та інтерактивне задання багатокутника, який відсікається.
13. Розробити проект цифрової обробки зображень. Програма повинна завантажити і відобразити BMP-файл в форматі RGB-24. До даного зображення застосувати 3×3 фільтри для виконання розмиття та збільшення різкості. Коефіцієнти фільтра вводити в діалоговому вікні. Створити рівномірну палітру з 256 кольорів і виконати палітризацію зображення даною палітрою. Реалізувати зведення зображення до чорно-білого та застосувати фільтр зміни яскравості.
14. Розробити проект „Типи проекцій”. Передбачити вибір об'єктів для побудови проекцій, способу проектування та параметрів проекцій.
15. Розробити анімаційний проект падіння м'яча з висоти h_0 на деяку площину. В момент дотику м'яч сплющується і перетворюється в еліпсоїд. Потім м'яч відбивається від площини і т. д. Рух м'яча описується фізичними законами.

Список літератури

1. Абраш М. Таинства программирования графики. – К.: ЕвроСИБ, 2002. – 512 с.
2. Аммерал Л. Машинная графика: В 4 кн. – М.: СолСистем, 1992.
3. Анісімов В.А., Терещенко В.М., Кравченко І.В. Основні алгоритми обчислювальної геометрії: Навч. посібн. – К.: Київський університет, 2002. – 82 с.
4. Баяковский Ю.М. Начальный курс OpenGL. – М.: Планета знаний, 2007. – 219 с.
5. Блінова Т.О., Порєв В.М. Комп'ютерна графіка. – К.: Юніор, 2004. – 456 с.
6. Богуславский А.А. С++ и компьютерная графика. – М.: Компьютер Пресс, 2003. – 352 с.
7. Божко А.Н., Жук Д.М., Маничев В.Б. Компьютерная графика: Учеб. пособие. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2007. – 392 с.
8. Вержбицкий В.М. Основы численных методов. – М.: Высш. шк., 2002. – 840с.
9. Веселовська Г.В., Ходаков В.Є., Веселовський В.М. Комп'ютерна графіка. – Херсон: ОЛДІ-плюс, 2004. – 584 с.
10. Гайдуков С.А. OpenGL. Профессиональное программирование трехмерной графики на С++. – СПб.: Питер, 2004. – 716 с.
11. Гилой В. Интерактивная машинная графика. – М.: Мир, 1981. – 380 с.
12. Голованов Н. Н. и др. Компьютерная геометрия: Учеб. пособие для вузов. – М.: Академия, 2006. – 512 с.
13. Горобець С.М. Основы комп'ютерної графіки: Навч. посібн. – К.: Центр навчальної літератури, 2006. – 232 с.
14. Евченко А.И. OpenGL и DirectX. Программирование графики. – СПб.: Питер, 2006. – 349 с.
15. Иванов В. П., Батраков А.С. Трехмерная компьютерная графика. – М.: Радио и связь, 1995. – 223 с.
16. Краснов М.В. OpenGL. Графика в проектах Delphi. – СПб.: БХВ-Петербург, 2000. – 354 с.
17. Ласло М. Вычислительная геометрия и компьютерная графика на С++. – М.: Бином, 1997. – 304 с.
18. Мандельброт Д. Фрактальна геометрия природы. – Ижевск: РХД, 2002. – 480 с.
19. Михайлюк М.В. Основы компьютерной графики: Учеб. пособие. – М.: Изд-во Моск. техн. ун-та, 2002. – 80 с.
20. Морозов А. Д. Введение в теорию фракталов. – М.: Изд-во Инст. комп. исслед., 2002. – 160 с.

21. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики. – СПб.: БХВ-Петербург, 2003. – 560 с.
22. Ньюмен У., Спрул Р. Основы интерактивной машинной графики. – М.: Мир, 1976. – 574 с.
23. Павлидис Т. Алгоритмы машинной графики и обработка изображений. – М.: Радио и связь, 1986. – 400 с.
24. Осипов Д. Графика в проектах Delphi. – СПб.: Символ-Плюс, 2008. – 648 с.
25. Петровичев Е.И. Компьютерная графика: Учебное пособие. – М.: Изд-во Моск. гос. горного ун-та, 2003. – 207 с.
26. Поляков А.Ю., Бруснецов В.А. Методы и алгоритмы компьютерной графики в примерах на Visual C++. – СПб.: БХВ-Петербург, 2003. – 547 с.
27. Порев В.М. Компьютерная графика. – СПб.: БХВ-Петербург, 2002. – 432 с.
28. Роджерс Д. Алгоритмические основы машинной графики. – М.: Мир, 1989. – 512 с.
29. Роджерс Д., Адамс Дж. Математические основы машинной графики. – М.: Мир, 2001. – 604 с.
30. Романюк О.Н. Комп'ютерна графіка: Навч. посібн. – Вінниця: ВДТУ, 2001. – 130 с.
31. Сиденко Л.А. Компьютерная графика и геометрическое моделирование: Учеб. пособие для вузов. – СПб.: Питер, 2008. – 220 с.
32. Тихомиров Ю.В. OpenGL. Программирование трехмерной графики. – СПб.: БХВ-Петербург, 2002. – 304 с.
33. Тюкачев Н.А., Илларионов И.В., Хлебостроев В.Г. Программирование графики в Delphi. – СПб.: БХВ-Петербург, 2008. – 784 с.
34. Фоли Дж., ван Дэм А. Основы интерактивной машинной графики: В 2 кн. – М.: Мир, 1987.
35. Херн Д., Бейкер М. Компьютерная графика и стандарт OpenGL. – М.: Вильямс, 2005. – 1158 с.
36. Хилл Ф. OpenGL. Программирование компьютерной графики. – СПб.: Питер, 2002. – 1088 с.
37. Шикин Е.В., Плисс А.И. Кривые и поверхности на экране компьютера. – М.: Диалог МИФИ, 1996. – 240 с.
38. Шикин Е.В., Боресков А.В. Компьютерная графика. Полигональные модели. – М.: Диалог МИФИ, 2000. – 464 с.
39. Эгрон Д. Синтез изображений. Базовые алгоритмы. – М.: Радио и связь, 1993. – 216 с.
40. Эйнджел Э. Интерактивная компьютерная графика. Вводный курс на

- базе OpenGL. – М.: Издательский дом „Вильямс”, 2001. – 592 с.
41. Юань Фень. Программирование графики для Windows. – СПб.: Питер, 2002. – 1072 с.

Интернет-ресурсы

1. <http://graphics.cs.msu.su> – сайт з комп'ютерної графіки Московського державного університету ім. Ломоносова.
2. <http://cg.unicyb.kiev.ua> – сайт з комп'ютерної графіки Київського національного університету ім. Т. Г. Шевченка.
3. http://ermak.cs.nstu.ru/kg_rivs – сайт з комп'ютерної графіки Новосибірського технічного університету.
4. <http://graphics.cs.ucdavis.edu> – сайт з КГ інституту аналізу даних і візуалізації Каліфорнійського університету.
5. <http://www.cg.tuwien.ac.at/courses/cg2> – сайт Інституту комп'ютерної графіки і алгоритмів Віденського технічного університету.
6. <http://www.kgraph.narod.ru/lectures/lectures.htm> – лекції з КГ.
7. <http://www.codenet.ru/progr/cg/> – сайт з КГ.
8. <http://algotlist.manual.ru/graphics/> – графіка і обробка зображень.
9. <http://www.opengl.org.ru> – сайт з OpenGL.
10. <http://www.intuit.ru/department/graphics/rastrgraph/> – Інтернет-університет з інформаційних технологій.
11. <http://program.rin.ru/razdel/html/875.html> – сайт з програмування.

Додаткова література

1. Агостон Ж. Теория цвета и ее применение в искусстве и дизайне. – М.: Мир, 1982. – 184 с.
2. Гурский Ю.А., Гурская И.В. Photoshop CS3. – СПб.: Питер, 2007. – 512 с.
3. Завьялов Ю.С., Леус В.А., Скороспелов В.А. Слайны в инженерной геометрии. – М.: Машиностроение, 1985. – 204 с.
4. Луизов А. В. Цвет и свет. – Л.: Энергоатомиздат, 1989. – 256 с.
5. Льюис Г. Maya 5: Учеб. курс. – СПб.: Питер, 2003. – 527 с.
6. Маров М. 3DStudio MAX: Учеб. курс. – СПб.: Питер, 2003. – 608 с.
7. Математика и САПР: В 2 кн. – М.: Мир, 1988.
8. Панкратова Т.В. Photoshop CS: Учебный курс. – СПб.: Питер, 2004. – 586 с.
9. Пономаренко С.И. Adobe Illustrator CS3. – СПб.: БХВ-Петербург, 2008. – 721 с.
10. Романычева Э. Т. Трехмерное моделирование в AutoCAD. – М.: Радио и связь, 2003. – 384 с.
11. Тайц А. М., Тайц А. И. Corel DRAW. Краткий курс. – СПб.: Питер, 2002. – 328 с.