Маценко Василь Григорович

Народився 25.05.1953 р. у с. Архангельське Баштанського району Миколаївської області. Закінчив Чернівецький державний університет у 1975 р., аспірантуру Обчислювального центру АН СРСР у 1981 р.
Кандидат фізико-математичних наук, доцент кафедри прикладної математики та інформаційних технологій Чернівецького національного університету імені Юрія Федьковича. Сфера наукових інтересів: математичне моделювання екологічних систем, інформаційні технології.

Автор понад 130 науково-методичних праць, серед них навчальні посібники та монографія:

- Основи математичного моделювання : навч. пос. – Чернівці : Рута, 2004.
- Комп'ютерна графіка : навч. пос. – Чернівці : Чернівецький нац. ун-т, 2009.
- Інформатика та обчислювальна техніка : навч. пос. – Чернівці : Чернівецький нац. ун-т, 2012.
- Математичне моделювання : навч. пос. – Чернівці : Чернівецький нац. ун-т, 2014
- Інформаційні технології в прикладній лінгвістиці : навч. пос. – Чернівці : Чернівецький нац. ун-т, 2017.
- Математичне моделювання динаміки вікової структури біологічних популяцій : монографія – Чернівці : Чернівецький нац. ун-т, 2018.
- Математичне моделювання екологічних процесів : навч. пос. – Чернівці : Чернівецький нац. ун-т, 2019.

ЧНУ РУТА
ISBN 978-966-423-777-9
9 789664 237779

ОБЧИСЛЮВАЛЬНА ГЕОМЕТРІЯ ТА КОМП'ЮТЕРНА ГРАФІКА

В. Г. Маценко

В. Г. Маценко

ОБЧИСЛЮВАЛЬНА ГЕОМЕТРІЯ ТА КОМП'ЮТЕРНА ГРАФІКА

Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

В. Г. Маценко

Обчислювальна геометрія та комп'ютерна графіка

Навчальний посібник

Чернівці
Чернівецький національний університет
імені Юрія Федьковича
2023

УДК 004.92(075.8)
М 367

Друкується за ухвалою вченої ради
Чернівецького національного університету імені Юрія Федьковича
(протокол № 3 від 3.04.2023 р.)

Рецензенти:

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

Кузенков О. О. Кандидат фіз.-мат. наук, доцент кафедри обчислювальної математики та математичної кібернетики Дніпровського національного університету імені Олеся Гончара.

Маценко В. Г.

М 367 Обчислювальна геометрія та комп'ютерна графіка : навч. посіб. Чернівці : Чернівец. нац. ун-т ім. Ю. Федьковича, 2023. 440 с.

ISBN 978-966-423-777-9

У посібнику викладено основи обчислювальної геометрії та комп'ютерної графіки, наведено широкий спектр основних понять та алгоритмів. Висвітлено фізичні основи сприйняття кольору та особливості колірних моделей. Описано технічні та програмні засоби комп'ютерної графіки. Значну увагу приділено моделюванню афінних перетворень на площині та просторі, побудові проєкцій геометричних об'єктів. 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 комп'ютерних проєктів.

Для студентів вищих навчальних закладів, які навчаються за спеціальностями „Прикладна математика", „Комп'ютерні науки" та ін.

УДК 004.92(075.8)

ISBN 978-966-423-777-9

Навчальне видання
В. Г. **Маценко**
Обчислювальна геометрія
та комп'ютерна графіка
Навчальний посібник

Відповідальний за випуск ***Бігун Я. Й.***

Літературний редактор ***Колодій О. В.***
Комп'ютерна верстка ***Фонарюк Н. І.***
Технічний редактор ***Кудрінська О. М.***
Дизайн обкладинки ***Цвіра А. В.***

Підписано до друку 04.04.2023. Формат 60x84/16 Папір офсетний. Друк різогрфічний.
Ум.-друк. арк. 24,1.Обл.-вид. арк. 25,9. Зам. Н-027.
Видавництво та друкарня Чернівецького національного університету імені Юрія Федьковича
58002, м. Чернівці, вул. Коцюбинського, 2
e-mail: ruta@chnu.edu.ua
Свідоцтво суб'єкта видавничої справи ДК №891 від 08.04.2002 р.

ЗМІСТ

ПЕРЕДМОВА 9
Розділ 1. Основні відомості з аналітичної геометрії 13
1.1. Елементи векторного аналізу 13
1.2. Моделі прямої лінії на площині 18
1.3. Площина та пряма в просторі 21
1.4. Криві другого порядку 23
1.5. Системи координат у геометрії 26
Розділ 2. Вступ до комп'ютерної графіки 30
2.1. Основні поняття 30
2.2. Кілька фактів з історії розвитку комп'ютерної графіки 32
2.3. Простіші моделі графічних об'єктів 33
2.4. Математичні моделі об'єктів графічних сцен 40
2.5. Застосування комп'ютерної графіки 48
Розділ 3. Технічне та програмне забезпечення комп'ютерної графіки 59
3.1. Технічне забезпечення комп'ютерної графіки 59
3.1.1. Загальні відомості про обчислювальні засоби 59
3.1.2. Центральні апаратні засоби 62
3.1.3. Графічна підсистема комп'ютера 68
3.1.4. Пристрої введення графічної інформації 74
3.1.5. Пристрої виведення графічної інформації 78
3.1.6. Масштабування та дискредитація 88
3.2. Програмне забезпечення 91
3.2.1. Основні поняття 91
3.2.2. Найвідоміші графічні редактори 92
3.2.3. Системи комп'ютерної математики 95
3.2.4. Мови програмування графіки 95
3.2.5. Графічні засоби відеосистем 97
3.3. Види комп'ютерної графіки 101
3.3.1. Растрова графіка 101
3.3.2. Векторна графіка 102
3.3.3. Фрактальна графіка 104
3.3.4. Тривимірна графіка 106
3.4. Графічні файлові формати 106
Розділ 4. Колір. Моделі кольору 113
4.1. Природа кольору 113
4.2. Моделі кольорів 120
4.2.1. Адитивна модель кольору RGB 120
4.2.2. Субтрактивна модель кольорів CMY/CMYK 122
4.2.3. Суб'єктивна модель кольорів HSB (HSV) 125

4.3. Баланс кольорів 127
4.4. Кодування кольору. Палітра кольорів 128
4.5. Передача даних про колір 132
4.6. Оптимальне поєднання кольорів при побудові зображень 133
Розділ 5. Растрові алгоритми генерування ліній 137
5.1. Числові методи 137
5.2. Інкрементні алгоритми 139
5.2.1. Алгоритм Брезенхема для відрізка 140
5.2.2. Алгоритм Брезенхема для кола 143
5.2.3. Інкрементний алгоритм виведення еліпса 146
5.2.4. Інкрементний метод Жордана 148
Розділ 6. Растрові алгоритми зафарбовування і заповнення областей 152
6.1. Основні поняття 152
6.2. Рекурсивні алгоритми зафарбовування областей 154
6.3. Пострічковий алгоритм зафарбовування із затравкою 155
6.4. Алгоритм зафарбовування області за критерієм парності 158
6.5. Зафарбовування полігонів. YX-алгоритм 160
6.6. Заповнення фігур. Текстури 161
6.7. Обробка растрових зображень 163
Розділ 7. Побудова інтерполяційних та згладжуючих кривих 171
7.1. Основні поняття 171
7.2. Поліноміальна інтерполяція 172
7.2.1. Інтерполяційний багаточлен Лагранжа 172
7.2.2. Інтерполяційні сплайни 172
7.3. Згладжуючі кубічні сплайни 176
7.4. Сплайнові криві 177
7.5. Криві Безьє 180
7.5.1. Основні поняття 180
7.5.2. Властивості кривих Безьє 181
7.5.3. Складені криві Безьє 184
7.5.4. Геометричний алгоритм для кривої Безьє 185
7.6. B-сплайнові криві 185
7.7. Інтерполяційні кубічні криві Ерміта 193
7.8. ТСВ-сплайни 196
Розділ 8. Математичні моделі поверхонь 200
8.1. Білінійна та лінійчаста поверхні 201
8.2. Інтерполяційні бікубічні сплайни 203
8.3. Сплайнові поверхні 204
8.3.1. Поверхні Безьє 204
8.3.2. B-сплайнові поверхні 206

Розділ 9. Основні алгоритми комп'ютерної геометрії 209
9.1. Тести орієнтації 209
9.1.1. Орієнтація нормального вектора 209
9.1.2. Розміщення точки відносно прямої 209
9.1.3. Тест напряму обходу трьох точок 210
9.2. Тест опуклості полігона 211
9.3. Тести орієнтації точки відносно полігона 211
9.3.1. Габаритний тест 211
9.3.2. Тест, що визначає орієнтацію точки відносно кожного ребра . 212
9.3.3. Променевий тест 212
9.3.4. Кутовий тест 213
9.4. Тести перетину 215
9.4.1. Тест перетину прямої з полігоном 215
9.4.2. Тест перетину відрізків 217
9.5. Алгоритми відсікання 217
9.5.1. Двовимірний алгоритм Сазерленда–Коена 218
9.5.2. Відсікання відрізка опуклим полігоном 219
9.5.3. Перетин та об'єднання опуклих полігонів 221
9.6. Інші алгоритми відсікання відрізків 221
9.6.1. Двовимірний *FC*-алгоритм 222
9.6.2. Алгоритм Кіруса–Бека 224
9.6.3. Відсікання полігонів. Алгоритм Вейлера–Азертона 228
9.7. Побудова опуклої оболонки масиву точок 231
9.7.1. Метод загортання подарунка 231
9.7.2. Метод обходу Грехема 232
9.8. Тріангуляція полігонів і точкових множин 234
9.8.1. Тріангуляція опуклих полігонів 235
9.8.2. Тріангуляція неопуклих полігонів 236
9.8.3. Тріангуляція точкової множини 237
9.8.4.Тріангуляція Делоне 239
9.8.5. Діаграма Вороного 240
9.9. Тести властивостей поліедра 242
9.9.1. Тест перетину площини та поліедра 242
9.9.2. Тест опуклості поліедра 243
9.9.3. Тести орієнтації точки відносно поліедра 243
Розділ 10. Фрактали в комп'ютерній графіці 247
10.1. Поняття фрактала 247
10.2. Конструктивні фрактали 250
10.2.1. Крива Коха 250
10.2.2. Гілка папороті та дендрити 256
10.2.3. Зіркові фрактали 259

10.3. Аналіз конструктивних фракталів 261
10.4. Динамічні фрактали 268
10.4.1. Множини Жуліа і Мандельброта 269
10.4.2. Фрактали Жуліа 270
10.4.3. Фрактали Мандельброта 272
10.4.4. Фрактали Ньютона 274
10.5. Застосування фракталів 276
Розділ 11. Моделювання 2D/3D-перетворень 279
11.1. Афінні перетворення на площині 279
11.2. Афінні перетворення в просторі 288
11.3. Приклади складніших 3*D*-перетворень 291
11.4. Методи задання складних афінних перетворень 296
Розділ 12. Моделювання проєкцій 304
12.1. Класифікація проєкцій 304
12.2. Ортографічна проєкція 306
12.3. Аксонометрична проєкція 307
12.4. Косокутна проєкція 313
12.5. Одноточкова (однофокусна) перспективна проєкція 317
12.6. Двоточкове та триточкове перспективні перетворення 320
12.7. Методи створення перспективних видів 322
Розділ 13. Системи координат та їх перетворення 326
13.1. Системи координат 326
13.2. Видове перетворення 329
13.3. Перспективне проєктування 332
13.4. Відображення у вікно виведення 336
Розділ 14. Усунення невидимих ліній і граней 338
14.1. Основні поняття 338
14.2. Алгоритм поточного горизонту 339
14.3. Алгоритм Робертса 345
14.4. Метод *Z*-буфера 353
14.5. Огляд деяких інших методів 356
14.5.1. Метод відсікання нелицьових граней 356
14.5.2. Алгоритм розбиття картинної площини Варнока 357
14.5.3. Метод сортування за глибиною. Алгоритм художника 358
Розділ 15. Зафарбовування видимих поверхонь 363
15.1. Основні поняття 363
15.2. Моделі відбиття світла 366
15.2.1. Дзеркальне відбиття світла 366
15.2.2. Дифузне відбиття 368
15.3. Обчислення нормалей до поверхні відбиття світла 371

15.4. Зафарбовування поверхонь 374
15.4.1. Зафарбовування з постійною інтенсивністю 374
15.4.2. Метод Гуро 375
15.4.3. Метод Фонга 376
15.5. Методи трасування променів 377
Розділ 16. Програмування графіки на OpenGL 384
16.1. Основні поняття 384
16.2. Вершини і примітиви 388
16.3. Перетворення координат і проєкцій 393
16.4. Функції виведення тривимірних об’єктів 397
16.5. Функції визначення видимих поверхонь 398
16.6. Моделювання освітлення 398
16.7. Область виведення 402
Завдання до лабораторних робіт 405
Лабораторна робота № 1. Вступні завдання 405
Лабораторна робота № 2. Фрактали 408
Лабораторна робота № 3. Растрові алгоритми. Сплайнові криві. Алгоритми відсікання 413
Лабораторна робота № 4. Візуалізація тривимірних об’єктів 415
Комп’ютерні проєкти 418
СПИСОК ЛІТЕРАТУРИ 420
ДОДАТОК 424

Найголовніша справа в житті – це молитва,
а всі інші справи другорядні.

Св. Григорій Богослов

Моїм онукам Костянтину
та Олександрі присвячую

ПЕРЕДМОВА

Обчислювальна геометрія – це наука (розділ інформатики), в якій розглядаються алгоритми розв'язування геометричних задач для роботи з графічними зображеннями, наприклад задача тріангуляції полігонів, побудови згладжуючих кривих, геометричного пошуку побудови перетину та об'єднання об'єктів, здійснення афінних та проєктивних перетворень тощо.

Класична геометрія часто не дає можливості спроєктувати ефективні алгоритми для розв'язування задач на ЕОМ. Тому необхідно було запропонувати нові підходи в геометрії, які сприятимуть ефективності обчислень на комп'ютері. Обчислювальна геометрія надала класичній геометрії обчислювальної форми.

Обчислювальна геометрія має свою специфіку і відрізняється від класичної геометрії, на яку вона опирається. Знання з обчислювальної геометрії важливі для ефективного розв'язування геометричних задач у різних галузях людської діяльності. Основу обчислювальної геометрії заклали геометричні задачі, для яких необхідно було розробити оптимальні алгоритми їх розв'язування, хоча геометричні обчислення мають давню історію. Як самостійна дисципліна вона зародилася в кінці 1970-х років. Термін "обчислювальна геометрія" запропонований у 1975 р. в книзі Препати і Шеймоса [46].

Її успіх як окремої галузі наукових досліджень можна пояснити широтою застосування – комп'ютерна графіка, геоінформаційні системи (ГІС), робототехніка, конструкторська справа та інші галузі, в яких геометричні алгоритми відіграють значну роль. Знання обчислювальної геометрії сприяють ефективному розв'язуванню геометричних задач у різноманітних галузях.

Багато задач обчислювальної геометрії виникає при візуалізації геометричних об'єктів. Зокрема, статичні задачі перетину геометричних об'єктів, тріангуляції полігонів та набору точок, задачі геометричного пошуку, динамічні задачі в яких поступово змінюються вхідні дані.

Виходячи з назви книги, ми будемо розглядати основи обчислювальної геометрії, що використовуються в комп'ютерній графіці.

Комп'ютерна графіка – це розділ комп'ютерної математики, що займається формуванням зображень на екрані комп'ютера. Вона є складовою сучасних комп'ютерних технологій. Графічний інтерфейс став стандартом для програмного забезпечення різних класів.

Комп'ютерна графіка привертає увагу багатьох спеціалістів із різних галузей знань – програмістів, проектувальників засобів візуалізації, інженерів, фізиків, математиків тощо. При розробці сучасних програм робота над графікою займає до 90 % робочого часу програмістів. Комп'ютерна графіка використовується практично у всіх наукових та інженерних дисциплінах. Комп'ютерна графіка – важливий компонент освіти сучасного спеціаліста. Знання її основ необхідне кожному вченому й інженеру.

Комп'ютерна графіка – галузь знань, в якій, з одного боку, накопичено значний багаж знань, а з іншого, відбувається постійний розвиток методів, алгоритмів, практичних застосувань, це складна і різноманітна дисципліна. В багатьох випадках потреби в графіці можуть бути забезпечені різними існуючими графічними бібліотеками та системами. Однак постійно виникає потреба 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 підкреслити, що навіть оволодіння існуючими графічними засобами вимагає знань теоретичних основ комп'ютерної графіки.

Зауважимо, що для наукових досліджень у галузі обчислювальної геометрії та комп'ютерної графіки в світі проводяться міжнародні конференції та семінари, випускаються книги, друкується багато статей, виходять спеціалізовані журнали, зокрема "Journal of Computational Geometry", "Геометрія і графіка", в 1977 р. з'явився журнал "Computer Graphics Word".

Курс з комп'ютерної графіки для спеціальностей "Прикладна математика", "Комп'ютерні науки", "Системний аналіз" (для студентів, що вивчають вищу математику і програмування) передбачає оволодіння основними алгоритмами та методами, які можна було б використовувати при створенні нових реальних систем машинної графіки в поєднанні із застосуванням існуючих програмних засобів, отримання знань із геометричного моделювання та візуалізації об'єктів. Мета курсу – сформувати у студентів теоретичні знання і практичні навички з основ обчислювальної геометрії та комп'ютерної графіки, навчити студентів розуміти особливості роботи алгоритмів комп'ютерної графіки та створювати ефективні алгоритми для розв'язування задач комп'ютерної графіки.

Предметом курсу "Обчислювальна геометрія та комп'ютерна графіка" є математичні основи та базові алгоритми комп'ютерної графіки, її програмне забезпечення.

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.

Організація посібника. Книга задумана як навчальний посібник з курсу обчислювальної геометрії та комп'ютерної графіки.

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

У теоретичній частині курсу подано основні означення та поняття, зроблено опис технічних та програмних засобів комп'ютерної графіки, описано організацію графічних режимів та моделей кольору, наведено базові растрові алгоритми, основні алгоритми обчислювальної геометрії, методи моделювання координатних перетворень, методи й алгоритми тривимірної графіки, види проекцій, алгоритми усунення невидимих ліній та граней, методи рендерінга, технології програмування графіки, приклади розв'язування типових задач. Більшість 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 і навичок самостійної розробки алгоритмів та програмного забезпечення. Тут наведено завдання для лабораторних робіт та більш комплексні завдання для розробки комп'ютерних проектів. Оволодівши методикою побудови зображень відносно простих графічних елементів, студенти отримають навички для конструювання значно складніших геометричних об'єктів.

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

Зміст посібника значною мірою побудований на основі курсу "Обчислювальна геометрія та комп'ютерна графіка", який більше двадцяти років викладається автором на факультеті математики та інформатики Чернівецького національного університету імені Юрія Федьковича для студентів спеціальностей "Прикладна математика", "Комп'ютерні науки", "Системний аналіз". Видання може бути використане студентами як денної, так і заочної форм навчання, аспірантами, викладачами вищої школи та спеціалістами, які працюють у галузі програмування графіки.

Побажання і пропозиції щодо поліпшення структури та змісту посібника надсилайте за електронною адресою *v.matsenko@chnu.edu.ua*.

Розділ 1. Основні відомості з аналітичної геометрії

Вважатимемо, що на площині задана правостороння декартова система координат, тобто поворот від осі OX до суміщення з віссю OY здійснюється на кут 90° проти годинникової стрілки. Тепер геометричні об'єкти можна задавати аналітично. Так, щоб задати відрізок, досить вказати координати його кінців.

1.1. Елементи векторного аналізу

При розв'язуванні геометричних задач основним інструментом є вектори. Наведемо деякі відомості про них.

Вектором називається направлений відрізок у просторі або на площині, у якого точку A вважають початком, а точку B – кінцем і позначають $\overrightarrow{AB}$. Вектор позначатимемо і символом $\vec{a}$, і жирною буквою, наприклад $\mathbf{V}$.

Довжиною (*модулем*) $|\overrightarrow{AB}|$ вектора $\overrightarrow{AB}$ називається число, що дорівнює довжині відрізка AB, який зображає цей вектор.

Якщо точки A і B задані координатами $A(x_1, y_1)$, $B(x_2, y_2)$, то пара чисел $(x_2 - x_1, y_2 - y_1)$ називається *координатами вектора*.

Довжина вектора $\overrightarrow{AB}$ дорівнює

$$|\overrightarrow{AB}| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Вектори можна додавати, віднімати і множити на число. Для цього потрібно додати, відняти, або помножити на число координати вектора. Тобто, якщо $\vec{a}(a_x, a_y)$, $\vec{b}(b_x, b_y)$, то $\vec{a} + \vec{b} = (a_x + b_x, a_y + b_y)$, $\vec{a} - \vec{b} = (a_x - b_x, a_y - b_y)$, $\lambda\vec{a}(\lambda a_x, \lambda a_y)$.

Вектори $\vec{a}$, $\vec{b}$, які лежать на одній прямій або на паралельних прямих, називаються ***колінеарними***. Колінеарні вектори можуть бути співнаправлені або протилежно направлені. Необхідною і достатньою умовою колінеарності двох векторів $\vec{a}$ і $\vec{b}$ є умова

$$\vec{b} = \lambda\vec{a},$$

де $\lambda \neq 0$. Нульовий вектор колінеарний довільному вектору $\vec{a} \| \vec{0}$.

З умови колінеарності двох векторів одержуємо умову *колінеарності трьох точок* $p_1(x_1, y_1)$, $p_2(x_2, y_2)$, $p_3(x_3, y_3)$ (точки лежать на одній прямій):

$$\begin{vmatrix} p_2 - p_1 \\ p_3 - p_1 \end{vmatrix} = 0, \quad \text{або} \quad \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0.$$

Умова колінеарності n точок $p_1, p_2, \ldots, p_n$ має вигляд

$$rang\begin{vmatrix} p_2 - p_1 \\ p_3 - p_1 \\ \cdots\cdots \\ p_n - p_1 \end{vmatrix} = 1.$$

Скалярним добутком двох ненульових векторів $\vec{a}\left(a_x, a_y\right)$, $\vec{b}\left(b_x, b_y\right)$ називається число

$$\left(\vec{a}, \vec{b}\right) = \left|\vec{a}\right| \cdot \left|\vec{b}\right| \cdot \cos\varphi,$$

де φ – кут між векторами $\vec{a}$ і $\vec{b}$.

В координатах скалярний добуток обчислюється так:

$$\left(\vec{a}, \vec{b}\right) = a_x b_x + a_y b_y.$$

Як видно з наведених формул, скалярний добуток можна використовувати для знаходження кута між векторами

$$\cos\varphi = \frac{\left(\vec{a}, \vec{b}\right)}{\left|\vec{a}\right| \cdot \left|\vec{b}\right|}.$$

Якщо $\left(\vec{a}, \vec{b}\right) = 0$, то вектори $\vec{a}$ і $\vec{b}$ перпендикулярні, і навпаки, якщо вектори $\vec{a}$ і $\vec{b}$ перпендикулярні, то $\left(\vec{a}, \vec{b}\right) = 0$.

Кутом між векторами вважається найменший кут φ, $0 \le \varphi \le \pi$, на який потрібно повернути один з векторів до його суміщення з другим.

Для обчислення часто більш зручне поняття *орієнтованого кута*, тобто кута, який враховує взаємне розміщення векторів. Значення орієнтованого кута за абсолютною величиною дорівнює звичайному куту між векторами.

Орієнтований кут між векторами $\vec{a}$ і $\vec{b}$ додатний, якщо поворот від вектора $\vec{a}$ до вектора $\vec{b}$ здійснюється в додатному напрямку (проти годинникової стрілки), і від'ємний – у протилежному випадку.

На рис. 1.1 *а* орієнтований кут між векторами $\overrightarrow{OA}$, $\overrightarrow{OB}$ додатний, на рис. 1.1 *б* – від'ємний.

а *б*

Рис. 1.1. Орієнтований кут між векторами $\overrightarrow{OA}$, $\overrightarrow{OB}$

Псевдоскалярним, або *косим,* добутком векторів на площині називають число

$$\left[\vec{a},\vec{b}\right]=\left|\vec{a}\right|\cdot\left|\vec{b}\right|\cdot\sin\varphi,$$

де φ – кут орієнтованого повороту від вектора $\vec{a}$ до $\vec{b}$ проти годинникової стрілки (рис. 1.2).

а *б*

Рис. 1.2. Кут, що визначає косий добуток

Назва косого добутку векторів пов'язана з властивістю косої симетрії: $\left[\vec{a},\vec{b}\right]=-\left[\vec{b},\vec{a}\right]$.

Якщо вектори задані координатами $\vec{a}\left(a_x,a_y\right)$, $\vec{b}\left(b_x,b_y\right)$, то псевдоскалярний добуток знаходиться за формулою

$$\left[\vec{a},\vec{b}\right]=a_xb_y-a_yb_x,\quad\text{або}\quad\left[\vec{a},\vec{b}\right]=\begin{vmatrix}a_x & a_y\\ b_x & b_y\end{vmatrix}.$$

Величина $\left[\vec{a},\vec{b}\right]$ додатна, якщо пара векторів додатно орієнтована (рис. 1.2 *а*) і від'ємна – в протилежному випадку (рис. 1.2 *б*).

Для ненульових векторів косий добуток дорівнює нулю, якщо вектори колінеарні, тобто $\varphi=0$ або $\varphi=\pi$.

Геометрично косий добуток векторів $\left[\vec{a},\vec{b}\right]$ – це орієнтовна площа паралелограма, побудованого на цих векторах, тоді

$$S=\frac{1}{2}\left(a_xb_y-a_yb_x\right)$$

– орієнтовна площа трикутника (рис. 1.3). За орієнтовною площею можна знаходити звичайну площу трикутника

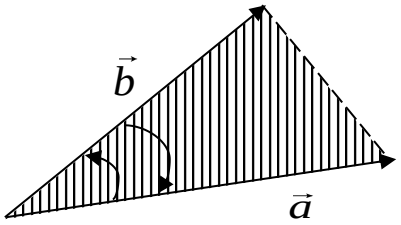


Рис. 1.3. Орієнтовна площа трикутника

Якщо задано три точки $p_1\left(x_1,y_1\right)$, $p_2\left(x_2,y_2\right)$, $p_3\left(x_3,y_3\right)$, то орієнтовна площа трикутника, тобто площа трикутника, взятого зі знаками $\pm$, за-

лежить від напряму повороту, що утворюється точками p_1, p_2, p_3, проти годинникової стрілки або за нею. При цьому орієнтована площа, як і косий добуток, обчислюється як величина визначника, що складений з координат точок p_1, p_2, p_3:

$$2S = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}.$$

Або, розкриваючи визначник, маємо

$$2S = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1).$$

Отже, якщо визначник буде додатний, то поворот $p_1p_2p_3$ здійснюється в додатному напрямку (проти годинникової стрілки), якщо від'ємний, то у від'ємному напрямку (за годинниковою стрілкою). Якщо визначник дорівнює нулю, то всі три точки лежать на єдиній прямій.

Функція, що реалізує обчислення подвійної орієнтовної площі трикутника $p_1p_2p_3$ може мати вигляд

```
int triangle (int x1, int y1, int x2, int y2, int x3, int y3)
return (x2–x1)*(y3–y1)–(y2–y1)*(x3–x1);
```

Косий добуток в обчислювальній геометрії відіграє значну роль. Багато геометричних задач мають більш просте розв'язування, якщо використати косий добуток, наприклад знаходження площі трикутника.

Співставляючи формули для скалярного і косого добутків векторів, маємо

$$\operatorname{tg}\varphi = \frac{\left[\vec{a},\vec{b}\right]}{\left(\vec{a},\vec{b}\right)} = \frac{a_x b_y - a_y b_x}{a_x b_x + a_y b_y} = q.$$

Знаючи $\operatorname{tg}\varphi$, можна знайти кут між векторами, обчисливши значення $\operatorname{arctg} q$ Для цього ще потрібно з'ясувати гострий кут чи тупий. Це можна встановити за знаком скалярного добутку.

Зазначимо, що в деяких мовах програмування з обернених тригонометричних функцій реалізована лише функція arctg, тому вигідно знаходити саме $\operatorname{tg}\varphi$.

Для лівосторонньої системи координат (такою є система екрана комп'ютера – вісь абсцис дивиться вправо, вісь ординат – вниз) додатним вважається поворот за годинниковою стрілкою. Викладені вище факти можна застосовувати і для лівосторонньої системи координат.

Для тривимірних векторів у просторі розглянемо ще поняття векторного добутку.

Векторним добутком вектора $\vec{a}$ на вектор $\vec{b}$ називається вектор $\vec{c}$ (рис 1.4), який

1) перпендикулярний векторам $\vec{a}$ і $\vec{b}$, тобто $\vec{c} \perp \vec{a}$, $\vec{c} \perp \vec{b}$;

2) має довжину $|\vec{c}| = |\vec{a}| \cdot |\vec{b}| \cdot \sin\varphi$, де φ – кут між векторами $\vec{a}$ і $\vec{b}$;

3) з кінця вектора $\vec{c}$ коротший поворот від вектора $\vec{a}$ до вектора $\vec{b}$ видно таким, що виконується проти годинникової стрілки.

Векторний добуток векторів $\vec{a}$ і $\vec{b}$ позначається $\vec{a} \times \vec{b}$ і має властивість $\vec{b} \times \vec{a} = -(\vec{a} \times \vec{b})$

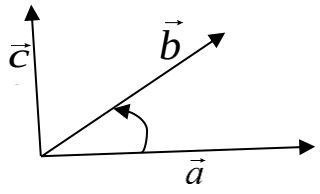


Рис. 1.4. Векторний добуток векторів $\vec{a}$ і $\vec{b}$

Якщо вектори $\vec{a}$ і $\vec{b}$ задаються в координатній формі $\vec{a}(a_x, a_y, a_z)$, $\vec{b}(b_x, b_y, b_z)$, то

$$\vec{a} \times \vec{b} = \begin{vmatrix} a_y & a_z \\ b_y & b_z \end{vmatrix} \vec{i} - \begin{vmatrix} a_x & a_z \\ b_x & b_z \end{vmatrix} \vec{j} + \begin{vmatrix} a_x & a_y \\ b_x & b_y \end{vmatrix} \vec{k} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix},$$

де $\vec{i}$, $\vec{j}$, $\vec{k}$ – одиничні орти, направлені вздовж координатних осей.

Геометричний зміст векторного добутку полягає в тому, що його модуль чисельно дорівнює площі паралелограма, побудованого на векторах $\vec{a}$ і $\vec{b}$ ($S = |\vec{a} \times \vec{b}|$).

Три точки $p_1(x_1, y_1, z_1)$, $p_2(x_2, y_2, z_2)$, $p_3(x_3, y_3, z_3)$ в просторі колінеарні, якщо $(p_2 - p_1) \times (p_3 - p_1) = 0$. Якщо $\vec{a} \times \vec{b} \neq 0$, то вектори $\vec{a}$ і $\vec{b}$ – лінійно незалежні.

Мішаний добуток $(\vec{a}, \vec{b}, \vec{c})$ векторів $\vec{a}$, $\vec{b}$ і $\vec{c}$ – це скалярний добуток вектора $\vec{a}$ на векторний добуток векторів $\vec{b}$ і $\vec{c}$

$$(\vec{a}, \vec{b}, \vec{c}) = (\vec{a}, \vec{b} \times \vec{c}) = (\vec{a} \times \vec{b}, \vec{c}).$$

Мішаний добуток $(\vec{a}, \vec{b}, \vec{c})$ в правосторонній декартовій системі координат дорівнює визначнику матриці, складеної з координат векторів $\vec{a}$, $\vec{b}$, $\vec{c}$

$$(\vec{a}, \vec{b}, \vec{c}) = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix}.$$

У лівосторонній системі координат цей визначник потрібно взяти зі знаком «мінус».

Мішаний добуток $\left(\vec{a},\vec{b},\vec{c}\right)$ за абсолютною величиною дорівнює об'єму паралелепіпеда, утвореного векторами $\vec{a}$, $\vec{b}$, $\vec{c}$ (рис. 1.5).

Знак мішаного добутку залежить від того, чи є трійка векторів правою чи лівою.

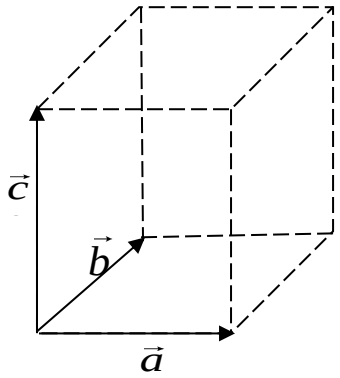


Рис. 1.5. Три вектори, що формують паралелепіпед

Чотири точки p_1, p_2, p_3, p_4 ***компланарні***, тобто лежать в одній площині, якщо

$$\left(\left(p_2-p_1\right)\times\left(p_3-p_1\right),p_4-p_1\right)=0.$$

1.2. Моделі прямої лінії на площині

Неявне (загальне) рівняння прямої задається співвідношенням вигляду

$$Ax+By+D=0,$$

де A,B,D – числові коефіцієнти.

Вектор $\vec{N}=\left(A,B\right)$ є нормальним (перпендикулярним) до прямої, тоді напрямний вектор прямої має вигляд $\vec{V}=\left(-B,A\right)$ або $\vec{V}=\left(B,-A\right)$.

Неявне рівняння прямої можна звести до *явного*, якщо визначити y,

$$y=-\frac{A}{B}x-\frac{D}{B} \quad \text{або} \quad y=kx+b.$$

Останнє рівняння називають рівнянням прямої з *кутовим коефіцієнтом* $k=\mathrm{tg}\varphi$, де φ – кут між додатним напрямком осі *OX* і прямою.

Якщо для двох прямих $y=k_1x+b_1$ і $y=k_2x+b_2$ їхні кутові коефіцієнти рівні ($k_1=k_2$), то прямі паралельні. Для того, щоб ці прямі були перпендикулярними, необхідно і достатньо, щоб їхні кутові коефіцієнти задовольняли співвідношення $k_1=-\frac{1}{k_2}$.

Явне рівняння можна використовувати для побудови лінії по точках в інтервалі $\left[x_{\min},x_{\max}\right]$. Але цей спосіб не можна використовувати для вертикальних ліній із коефіцієнтом $B=0$ і єдиною x – координатою для всіх точок таких прямих.

Нехай положення прямої визначається двома величинами – довжиною p і напрямом перпендикуляра *OP*, опущеного з початку координат на пряму. Напрям задається величиною кута α, який утворює цей перпендикуляр з віссю *OX* (рис. 1.6). Тоді рівняння прямої має вигляд

$$x\cos\alpha + y\sin\alpha - p = 0,$$

яке називається ***нормальним*** *рівнянням* прямої.

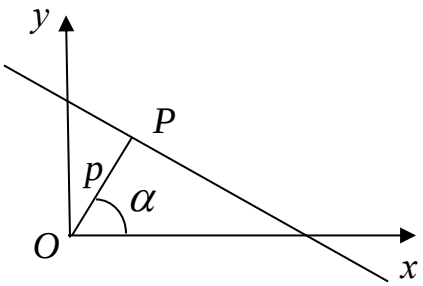


Рис. 1.6. Параметри нормального рівняння прямої

Для зведення загального рівняння прямої $Ax + By + D = 0$ до нормального необхідно його помножити на нормувальний множник $\mu = \dfrac{1}{\pm\sqrt{A^2 + B^2}}$. Знак радикала вибирається з умови $\mu D < 0$. Далі, якщо ввести позначення $\dfrac{A}{\pm\sqrt{A^2 + B^2}} = \cos\alpha$, $\dfrac{B}{\sqrt{A^2 + B^2}} = \sin\alpha$, $\dfrac{D}{\pm\sqrt{A^2 + B^2}} = p > 0$, то одержимо нормальне рівняння прямої.

Параметричне рівняння прямої, що проходить через точку $p_0(x_0, y_0)$ в напрямку вектора $\vec{v} = (v_x, v_y)$, має вигляд $x = x_0 + v_x t$, $y = y_0 + v_y t$, або $p(t) = p_0 + \vec{v}t$, де t – числовий параметр, $p(t) = (x, y)^T$. Координати точки на лінії представлені у вигляді функцій деякого параметра. Параметрична форма задання ліній і поверхонь найбільш універсальна в комп'ютерній графіці.

Параметрична форма зручна для задання частин прямих – відрізків і променів. Для цього необхідно в параметричному рівнянні вказати межі зміни параметра t: $t \ge 0$ – для променя, $t \in [t_0, t_1]$ – для відрізка.

Якщо відомо, що лінія проходить через дві точки $a(a_x, a_y)$ і $b(b_x, b_y)$ $(a \ne b)$, то параметричне рівняння лінії має вигляд $x = a_x + (b_x - a_x)t$, $y = a_y + (b_y - a_y)t$, або $p(t) = a + (b - a)t$. При зміні параметра t від $t_0 = 0$ до $t_1 = 1$ одержимо відрізок $[a, b]$. При t >1одержуємо точки, які лежать на лінії правіше *b*, а при t<0 – лівіше *a*.

Для знаходження параметра t, що відповідає положенню точки $p(t) = a + (b - a)t$, на прямій відносно відрізка ab знайдемо скалярний добуток

$$(p - a, b - a) = ((b - a)t, b - a) = |b - a|^2 t.$$

Звідки

$$t = \frac{(p-a, b-a)}{(b-a, b-a)}.$$

Рівняння бісектриси кута. Нехай вектори $\overrightarrow{p_0 p_1} = (x_1, y_1)$, $\overrightarrow{p_0 p_2} = (x_2, y_2)$ мають спільний початок – точку $p_0(x_0, y_0)$. Знайдемо рівняння бісектриси кута $p_1 p_0 p_2$. Поділимо кожен із векторів на його довжину, при цьому одержимо одиничні вектори. Вектор суми цих одиничних векторів буде лежати на бісектрисі цього кута (рис. 1.7).

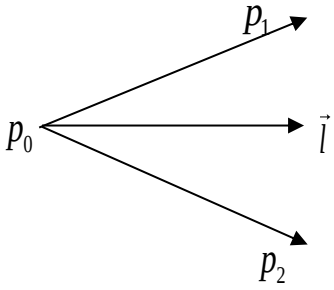


Рис. 1.7. Бісектриса кута

Координати вектора бісектриси дорівнюють

$$\vec{l} = \left(\frac{x_1}{\sqrt{x_1^2 + y_1^2}} + \frac{x_2}{\sqrt{x_2^2 + y_2^2}}, \frac{y_1}{\sqrt{x_1^2 + y_1^2}} + \frac{y_2}{\sqrt{x_2^2 + y_2^2}} \right).$$

Нехай $p(x, y)$ – довільна точка, що лежить на бісектрисі кута, тоді вектори $\overrightarrow{p_0 p}$ і $\vec{l}$ колінеарні.

Виходячи з умови колінеарності, запишемо рівняння бісектриси кута

$$\left(\frac{y_1}{\sqrt{x_1^2 + y_1^2}} + \frac{y_2}{\sqrt{x_2^2 + y_2^2}} \right)(x - x_0) - \left(\frac{x_1}{\sqrt{x_1^2 + y_1^2}} + \frac{x_2}{\sqrt{x_2^2 + y_2^2}} \right)(y - y_0) = 0.$$

Запишемо ще рівняння прямої, що перпендикулярна до цієї прямої і проходить через задану точку $p_0(x_0, y_0)$.

Нехай $p(x, y)$ – довільна точка шуканої прямої, тоді вектор $\overrightarrow{p_0 p}$ перпендикулярний до заданої прямої. Припустимо, що задана пряма визначається двома точками $p_1(x_1, y_1)$, $p_2(x_2, y_2)$, тоді вектори $\overrightarrow{p_0 p}$ і $\overrightarrow{p_1 p_2}$ перпендикулярні. Це означає, що скалярний добуток векторів $\left(\overrightarrow{p_0 p}, \overrightarrow{p_1 p_2}\right)$ дорівнює нулю, тобто

$$(x - x_0, x_2 - x_1) + (y - y_0, y_2 - y_1) = 0.$$

Це і є рівняння шуканої прямої, яке можна переписати і в іншій формі.

Важливою задачею геометрії на площині є знаходження точки q перетину двох прямих.

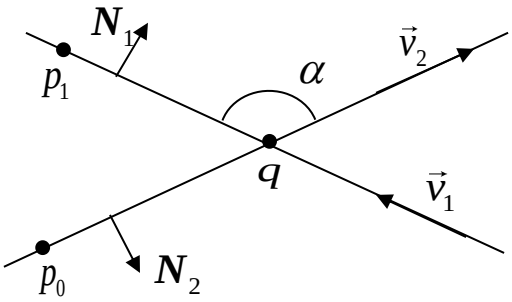


Рис. 1.7. Перетин прямих, кут між прямими

У випадку параметричної форми задання прямих маємо:

$$p_1 + \vec{v}_1 t = p_0 + \vec{v}_2 \tau \quad \Rightarrow \quad (t, \tau) = (p_0 - p_1)\begin{pmatrix} \vec{v}_1 \\ -\vec{v}_2 \end{pmatrix}^{-1} \Rightarrow \quad q = p_1 + \vec{v}_1 t = p_0 + \vec{v}_2 \tau.$$

Точка перетину існує, якщо визначник $\begin{vmatrix} \vec{v}_1 \\ -\vec{v}_2 \end{vmatrix} \neq 0$, тобто прямі не паралельні.

Для визначення кута перетину двох прямих введемо в розгляд функцію обчислення кута між векторами $\boldsymbol{V}$ і $\boldsymbol{W}$

$$angl(\boldsymbol{V}, \boldsymbol{W}) = if\left(d = \begin{vmatrix} \boldsymbol{V} \\ \boldsymbol{W} \end{vmatrix} \neq 0; sign(d), 1 \right) \arccos \frac{(\boldsymbol{V}, \boldsymbol{W})}{|\boldsymbol{V}| \cdot |\boldsymbol{W}|},$$

як кута коротшого повороту від $\boldsymbol{V}$ до $\boldsymbol{W}$.

У випадку, коли визначник d матриці, побудованої на цих векторах, не дорівнює нулю ($d \neq 0$), модуль кута дорівнює $\arccos \frac{(\boldsymbol{V}, \boldsymbol{W})}{|\boldsymbol{V}| \cdot |\boldsymbol{W}|} \in (0, \pi)$, а знак кута визначається знаком визначника $d \neq 0$.

При $d = 0$ вектори $\boldsymbol{V}$ і $\boldsymbol{W}$ колінеарні, знак $sign(d) = 0$ для функції $angl(\boldsymbol{V}, \boldsymbol{W})$ ігнорується. Це відбувається у випадках однонаправлених векторів $\boldsymbol{V}$, $\boldsymbol{W}$ $(\arccos 1 = 0)$ або різнонаправлених векторів $(\arccos(-1) = \pi)$.

Отже, кут перетину двох прямих (рис. 1.7) знаходиться за допомогою функції $angl$ як $\alpha = angl(\boldsymbol{V}_1, \boldsymbol{V}_2)$ або $\alpha = angl(\boldsymbol{N}_1, \boldsymbol{N}_2)$.

1.3. Площина та пряма в просторі

Положення площин у просторі R^3 повністю визначається заданням вектора $\boldsymbol{N} = (A, B, C)$ перпендикулярного до цієї площини і деякої точки $p_0(x_0, y_0, z_0)$, що лежить у цій площині.

Рівняння площини у цьому випадку має вигляд

$$A(x - x_0) + B(y - y_0) + C(z - z_0) = 0,$$

$$f(p) = ((p - p_0), \boldsymbol{N}) = 0.$$

$Ax + By + Cz \pm D = 0$ – це загальне рівняння площини.

Вектор $\boldsymbol{N}=(A,B,C)$ називається нормальним вектором площини. Знак $q=Ax+By+Cz-D$ визначає, з якої сторони по відношенню до площини знаходиться точка (x,y,z). Якщо $q>0$, то точка (x,y,z) лежить з тієї сторони площини, куди вказує нормальний вектор (A,B,C). Якщо $q<0$, то на протилежній стороні. $q=0$ означає, що точка лежить на площині.

Рівняння площини, що проходить через три задані точки $M_1(x_1,y_1,z_1)$, $M_2(x_2,y_2,z_2)$, $M_3(x_3,y_3,z_3)$ записується у вигляді

$$\begin{vmatrix} x-x_1 & y-y_1 & z-z_1 \\ x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \end{vmatrix}=0,$$

або у вигляді $Ax+By+Cz=D$, де

$$A=\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix},\ B=\begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix},\ C=\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix},\ D=\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}.$$

Рівняння

$$\frac{x}{a}+\frac{y}{b}+\frac{z}{c}=1$$

називається *рівнянням площини у відрізках* на осях координат.

Параметричне рівняння площини, що задається точкою і двома лінійно незалежними векторами $\boldsymbol{V}$ і $\boldsymbol{W}$ ($\boldsymbol{V}\times\boldsymbol{W}\neq\vec{0}$), має вигляд

$$p(\tau,t)=p_0+\boldsymbol{V}t+\boldsymbol{W}\tau.$$

Параметрична форма зручна як для всієї площини, так і її частин. Наприклад,

- значення параметра $t\in(-\infty,\infty)$ і $\tau\geq 0$ визначає півплощину від прямої $p_0+\boldsymbol{V}t$ в сторону вектора $\boldsymbol{W}$;
- значення $t\in(-\infty,\infty)$, $\tau\in[0,1]$ визначають смугу вздовж вектора $\boldsymbol{V}$;
- значення $t\in[0,1]$, $\tau\in[0,1]$ визначають паралелограм між чотирма точками p_0, $p_0+\boldsymbol{V}$, $p_0+\boldsymbol{W}$, $p_0+\boldsymbol{V}+\boldsymbol{W}$;
- значення $t\in[0,1]$, $\tau\in[0,t]$ дають трикутник з вершинами у точках p_0, $p_0+\boldsymbol{V}$, $p_0+\boldsymbol{V}+\boldsymbol{W}$.

Зауважимо, що, маючи точку p_0 і два вектори $\boldsymbol{V}$ і $\boldsymbol{W}$, можна записати загальне рівняння площини у вигляді

$$f(p)=((p-p_0),\boldsymbol{V}\times\boldsymbol{W})=0.$$

Під кутом між двома площинами розуміють кут φ між нормальними векторами цих площин $\boldsymbol{N}_1=(A_1,B_1,C_1)$ і $\boldsymbol{N}_2=(A_2,B_2,C_2)$, а отже

$$\cos\varphi = \frac{(\boldsymbol{N}_1, \boldsymbol{N}_1)}{|\boldsymbol{N}_1| \cdot |\boldsymbol{N}_2|}, \quad \text{або} \quad \cos\varphi = \frac{(A_1A_2 + B_1B_2 + C_1C_2)}{\sqrt{A_1^2 + B_1^2 + C_1^2}\sqrt{A_2^2 + B_2^2 + C_2^2}}.$$

Відстань від точки $M_0(x_0, y_0, z_0)$ до площини $Ax + By + Cz = D$ визначається за формулою

$$d = \frac{|Ax_0 + By_0 + Cz_0 + D|}{\sqrt{A^2 + B^2 + C^2}}.$$

Положення прямої в просторі повністю визначається заданням довільної фіксованої точки $M_0(x_0, y_0, z_0)$ і вектора $\boldsymbol{V} = (l, m, n)$, який паралельний цій прямій або лежить на ній.

Рівняння

$$\frac{x - x_0}{l} = \frac{y - y_0}{m} = \frac{z - z_0}{n}$$

називається ***канонічним*** рівнянням прямої, яке можна переписати в *параметричному* вигляді

$$x = x_0 + lt,\ y = y_0 + mt,\ z = z_0 + nt,\ t \in R.$$

Зауважимо, що формулу, яка містить відношення при програмуванні не можна використовувати для перевірки належності точки прямій, оскільки навіть у випадку цілих координат будуть виникати помилки дійсної арифметики при операціях ділення.

Кут між двома прямими, що задаються напрямними векторами $\boldsymbol{V}_1 = (l_1, m_1, n_1)$ і $\boldsymbol{V}_2 = (l_2, m_2, n_2)$, визначається за формулою

$$\cos\varphi = \frac{(\boldsymbol{V}_1, \boldsymbol{V}_2)}{|\boldsymbol{V}_1| \cdot |\boldsymbol{V}_2|},$$

або

$$\cos\varphi = \frac{(l_1l_2 + m_1m_2 + n_1n_2)}{\sqrt{l_1^2 + m_1^2 + n_1^2}\sqrt{l_2^2 + m_2^2 + n_2^2}}.$$

Умови паралельності і перпендикулярності двох прямих рівносильні, відповідно, умовам колінеарності і перпендикулярності їхніх напрямних векторів $\boldsymbol{V}_1$ і $\boldsymbol{V}_2$.

1.4. Криві другого порядку

Рівняння

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0, \tag{1.1}$$

де A, B, C, D, E, F – задані дійсні числа і принаймні одне з чисел A, B, C відмінне від нуля, визначає в прямокутній системі координат XOY *криву другого порядку*.

До кривих другого порядку належать коло, еліпс, гіпербола, парабола, пара пересічних або паралельних прямих. Якщо немає точок із дійсними координатами, які задовольняють рівняння (1.1), то кажуть, що рівняння (1.1) визначає уявну криву другого порядку.

Рівняння ***кола*** з центром у точці $P_0(x_0, y_0)$ і радіусом R має вигляд

$$(x-x_0)^2+(y-y_0)^2=R^2.$$

Для практики більш корисне параметричне рівняння кола, яке задається у вигляді

$$x=x_0+R\cos t,\quad y=y_0+R\sin t,\quad t\in[0,2\pi),$$

де x_0, y_0 – координати центра кола; R – радіус; t – кут між радіусом вектором і віссю OX, який відраховується проти годинникової стрілки.

Еліпс – це множина всіх точок площини, сума відстаней кожної з яких від двох даних точок цієї площини, які називаються фокусами, є сталою величиною, більшою за відстань між фокусами (рис. 1.8).

Канонічне рівняння еліпса має вигляд

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1.$$

Точки перетину еліпса з осями є вершинами еліпса. Вони мають координати $(a,0)$, $(0,b)$, $(-a,0)$, $(0,-b)$. Фокуси еліпса мають координати $F_1(-c,0)$ і $F_2(c,0)$, де $c^2=a^2-b^2$, $a>b$.

Параметричні рівняння еліпса можна задати так:

$$x=a\cos t,\quad y=b\sin t,\quad t\in[0,2\pi).$$

Дотична до еліпса в точці (x_0, y_0), яка належить еліпсу, задається рівнянням

$$\frac{x_0}{a^2}x+\frac{y_0}{b^2}y=1.$$

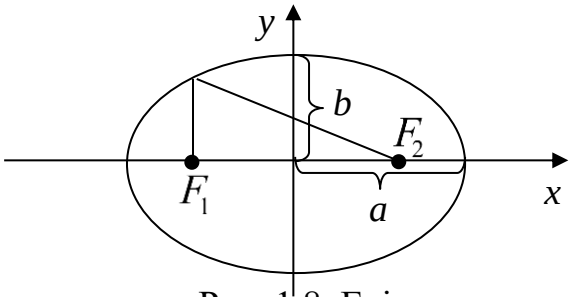


Рис. 1.8. Еліпс

Гіперболою називається множина точок площини, абсолютна величина різниці відстаней кожної з яких від двох даних точок цієї площини, які називаються фокусами, є сталою величиною, що не дорівнює нулю і менша від відстані між фокусами (рис. 1.9).

Канонічне рівняння гіперболи має вигляд

$$\frac{x^2}{a^2}-\frac{y^2}{b^2}=1.$$

Гіпербола має дві осі симетрії, які збігаються з координатними осями. Осі симетрії гіперболи називаються осями гіперболи. Вісь гіперболи, на якій розташовані фокуси, називається фокальною віссю. Точки перетину

гіперболи з фокальною віссю називаються її вершинами. Абсциси її вершин $x=\pm a$. Відрізок довжиною $2a$, який з'єднує вершини гіперболи, називається дійсною віссю, а числа a і b – дійсною та уявною півосями гіперболи. Асимптоти гіперболи задаються рівняннями $y=\pm\frac{b}{a}x$. Фокуси гіперболи мають координати $F_1(-c,0)$, $F_2(c,0)$, де $c^2=a^2+b^2$.

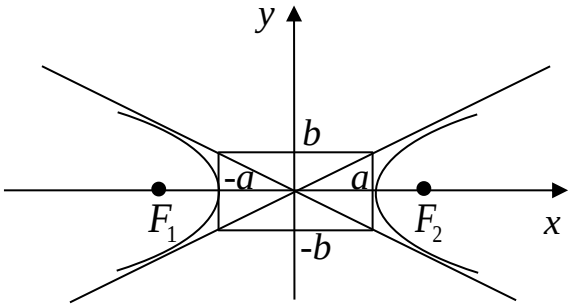


Рис. 1.9. Гіпербола

Гіпербола, представлена рівнянням $\frac{x^2}{a^2}-\frac{y^2}{b^2}=1$, може бути задана за допомогою параметричних рівнянь

1) $\begin{cases} x=\pm a\cos ht, \\ y=b\sin ht, \end{cases}$ $t\in R$, $h=const$;

2) $\begin{cases} x=\pm a\dfrac{t^2+1}{2t}, \\ y=b\dfrac{t^2-1}{2t}, \end{cases}$ $t>0$.

Полярні координати для гіперболи використовуються для декартової системи координат так, що їх початок знаходиться у фокусі, а полярна вісь направлена в початок канонічної системи координат, як це показано на графіку (рис. 1.10).

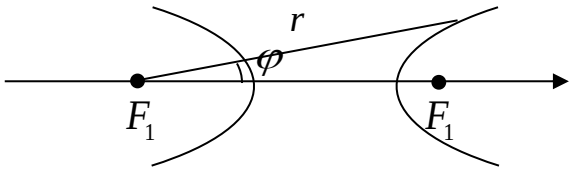


Рис. 1.10. Гіпербола. Полярні координати, в яких полюс збігається з фокусом

Відповідно до цієї системи координат маємо $r=\frac{p}{1\pm e\cos\varphi}$, $p=\frac{b^2}{a}$, $e=\frac{c}{a}$, $-\arccos\left(-\frac{1}{e}\right)<\varphi<\arccos\left(-\frac{1}{e}\right)$.

Якщо полюс полярних координат збігається із центром гіперболи,

тобто точкою M, то полярне рівняння гіперболи має вигляд

$$r = \frac{b}{\sqrt{e^2 \cos^2 \varphi - 1}}.$$

Для правої гілки гіперболи діапазон для φ становить

$$-\arccos\left(\frac{1}{e}\right) < \varphi < \arccos\left(\frac{1}{e}\right).$$

Парабола – геометричне місце точок, що рівновіддалені від точки і прямої (рис. 1.11). Точка називається фокусом, а пряма – директрисою.

Канонічне рівняння параболи має вигляд

$$y^2 = 2px.$$

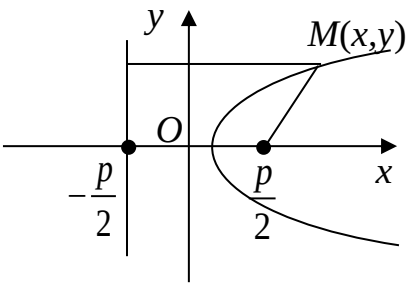


Рис. 1.11. Парабола та її фокус і директриса

Вісь симетрії параболи називається фокальною віссю, а точка O перетину параболи з віссю симетрії – її вершиною. Для параболи $x^2 = 2py$, $p > 0$ її вершина знаходиться в початку координат, а віссю симетрії є вісь Oy.

Якщо $p > 0$, то парабола $y^2 = 2px$ матиме полярні координати такого вигляду:

$$r = 2p\frac{\cos\varphi}{\sin^2\varphi}, \quad \varphi \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \setminus \{0\}.$$

Параболічні форми широко використовуються у природі, техніці. Зокрема, траєкторії космічних тіл, що проходять поблизу масивного об’єкта на досить великій швидкості, мають форму параболи (або гіперболи). При відсутності опору повітря траєкторія польоту тіла в однорідному гравітаційному полі є параболою. Парабола має властивість фокусувати пучок променів, що паралельні до осі параболи.

1.5. Системи координат у геометрії

Координати точки – це набір чисел, який визначає її розташування на площині або в просторі.

Система координат – це площина або простір, в якому визначені початок координат та осі, що дозволяє обчислювати координати точки.

У декартовій системі координат положення точки визначається відстанню від точки до осей координат (два параметри *x*, *y* на площині та три параметри *x*, *y*, *z* у просторі).

Для тривимірної декартової системи координат введене поняття орієнтації системи координат. Розрізняють ліву та праву системи координат. Це визначається напрямом повороту від осей *x* до *y*, *y* до *z*, *z* до *x*. Вони зображені на рис. 1.12.

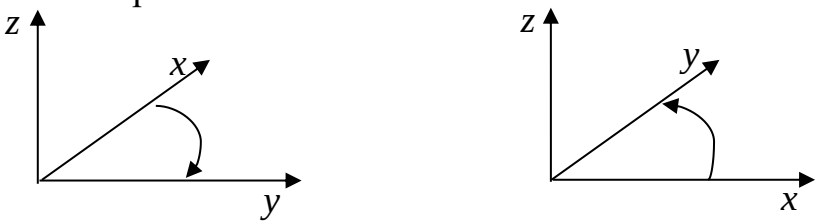


Рис. 1.12. Лівостороння (зліва) та правостороння (справа) системи координат

Полярна система координат (рис. 1.13) на площині – це така система координат, в якій положення точки визначається парою чисел, одне з яких є відстанню ρ від заданої точки до початку координат (полюса), а друге – кутом φ, що утворений радіусом-вектором цієї точки і полярною віссю. Декартові та полярні координати точки зв'язані між собою формулами

$$x = \rho\cos\varphi, \quad y = \rho\sin\varphi.$$

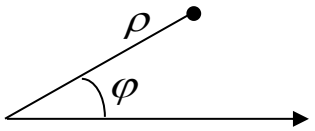


Рис. 1.13. Полярна система координат

Циліндрічні координати – це система координат, у якій розташування точки *P* задається трійкою чисел (r,φ,z), де

$0 \le r$ (радіус) – відстань від осі *Z* до точки *P*;

$0 \le \varphi < 360^\circ$ – кут між додатним напрямом осі *OX* і проєкцією *OP′* відрізка *OP*;

z – висота, що відповідає декартовій координаті *z* точки *P* (рис. 1.14).

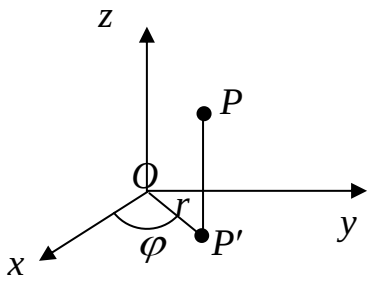


Рис.1.14. Циліндрична система координат

У ***сферичній системі*** координат розташування точки *P* визначається трьома компонентами (ρ,φ,θ), де (у термінах декартової системи координат)

$0 \le \rho$ (радіус) – відстань від точки *P* до початку координат;

$0 \le \varphi < 360^\circ$ – кут між додатною піввіссю *OX* і проєкцією відрізка *OP* на площину *XY*;

$0 \le \theta \le 180^\circ$ – кут між додатною піввіссю *Z* і відрізком OP (рис. 1.15).

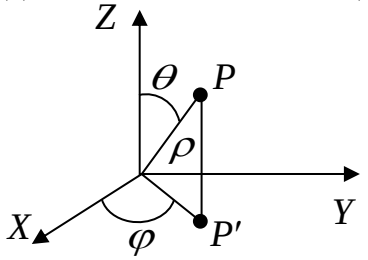


Рис. 1.15. Сферичні координати

Декартові та сферичні координати зв'язані між собою формулами

$$x = \rho \sin\varphi \cos\theta,$$
$$y = \rho \sin\varphi \sin\theta,$$
$$z = \rho \cos\varphi.$$

Зауваження. У програмах, які розв'язують геометричні задачі часто використовуються дійсні числа. При проведенні з ними операцій необхідно враховувати похибку обчислень, яка обумовлена машинною математикою (накопичення похибок у останніх розрядах). Це приводить до того що дійсні числа не можна порівнювати безпосередньо – їх потрібно, як правило, порівнювати з заданою точністю eps, де eps підбирається експериментально. Наприклад, при виясненні рівності двох чисел a та b потрібно записувати abs(a-b)<eps.

Також варто мати на увазі, що похибки можуть виникати у зв'язку з переповненням цілих типів. Наприклад, для обчислення віддалі між двома цілочисловими точками використовується вираз d=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)). Одержаний результат буде дійсним. Але перетворення в дійсний тип з цілого відбувається тільки при добуванні кореня, що робить можливим переповнення цілого типу при попередніх операціях віднімання, множення і додавання.

Контрольні запитання та завдання

1. Як визначити, чи два вектори колінеарні?
2. Що таке скалярний добуток векторів (як він обчислюється)?
3. Як визначити величину орієнтованого кута між векторами $\vec{a}$ і $\vec{b}$?
4. Дайте означення псевдоскалярного добутку.
5. Як обчислити орієнтовну площу трикутника?
6. Дайте означення векторного добутку. Сформулюйте його властивості.
7. Що таке мішаний добуток векторів? Який його геометричний зміст?
8. Запишіть різні типи рівнянь прямої лінії на площині.
9. Як записати параметричне рівняння відрізка?
10. Як побудувати рівняння бісектриси кута?
11. Як знаходиться точка перетину двох прямих?

12. Назвіть типи рівнянь площини у просторі.
13. Як, використовуючи параметричне рівняння площини, задати фрагмент площини, наприклад паралелограм?
14. Які рівняння визначають криві другого порядку?
15. Запишіть канонічні рівняння еліпса, гіперболи, параболи.
16. Запишіть параметричні рівняння кола, еліпса, гіперболи.
17. Опишіть системи координат, які використовуються в обчислювальній геометрії.

Вправи і задачі для самостійного виконання

1. Трикутник заданий своїми вершинами. Визначити тип трикутника (тупокутний, гострокутний чи прямокутний).
2. Побудувати рівняння прямої, що паралельна даній прямій і знаходиться від неї на віддалі d.
3. Задано коло із центром у точці $P_0(x_0, y_0)$ і радіусом R. Знайти рівняння дотичної до кола, яка проходить через точку $P_1(x_1, y_1)$. Розглянути випадки, коли точка P_1 лежить на колі і поза колом. Вказівка: задача має кілька способів розв'язання.
4. Виписати умови, коли два кола перетинаються, дотикаються або не мають спільних точок. Розглянути випадки, коли одне коло лежить усередині іншого або знаходиться зовні.
5. Побудувати коло описане навколо трикутника / вписане в трикутник.
6. Відрізок заданий двома цілочисловими точками. Визначити, скільки точок з цілочисловими координатами лежить на відрізку.
7. Побудуйте рівняння дотичної до еліпса в точці (x_0, y_0), що належить еліпсу.
8. Виведіть формули віддалі між двома точками в циліндричній і сферичній системах координат.

Розділ 2. Вступ до комп'ютерної графіки

2.1. Основні поняття

Формування комп'ютерної графіки (надалі – КГ) як самостійного напрямку інформаційних технологій припадає на 60-ті роки XX ст., коли Сазерленд створив спеціалізований пакет машинної графіки. Ця програма була призначена для малювання найпростіших фігур на екрані (крапок, ліній, прямокутників тощо).

Уперше 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.

Справжнього широкого розвитку КГ зазнала з появою персональних комп'ютерів. Розвиток комп'ютерної графіки проходить жваво: дещо швидко відживає, водночас з'являється багато нового. Нині важко уявити розвиток будь-якої галузі людської діяльності, пов'язаної з наукою чи технікою, без використання КГ. Вона є багатофункціональною складовою всіх інформаційних технологій і важливим компонентом 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. Великі обсяги інформації людина часто не може сприйняти в інших формах. З цього приводу можна згадати прислів'я „краще раз побачити, ніж сто разів почути" або 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, найбільш концентрована, 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, поки воно не з'явиться на екрані. Пасивний контроль не дає можливості втрутитися в процес проєктування 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. Наведемо деякі визначення.

Графіка *– вид мистецтва, що включає малюнок і художнє зображення (походить від грецьк. слова графо, що означає „пишу, креслю, малюю”).* ***Комп'ютерна графіка*** *– це галузь інформатики, що вивчає та розробляє методи і засоби синтезу, візуалізації, збереження, перетворення й застосування цифрових зображень за допомогою ЕОМ та інших технічних пристроїв.*

Цифрове зображення *– це модель реального або синтезованого (створеного штучно) зображення, що зберігається у файлах певного формату 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 проєктування (САПР), які з'явилися в 60-х роках XX ст.

КГ – це галузь інформатики, у сферу інтересів якої входять усі аспекти формування зображення за допомогою комп'ютера. Цифрові зображення немислимі без комп'ютерної обробки графічної інформації, їх простіше зберігати, тиражувати, компонувати 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 даних у різних сферах людської діяльності, 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.

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, тобто на вхід системи подають зображення, а на виході маємо символ.

Тому обробка та розпізнавання зображень віднесені в самостійні напрями у сфері обробки графічної інформації. Питаннями роботи з зображеннями на ЕОМ займаються три напрями інформатики: ***комп'ютерна графіка (computer graphics), обробка зображень (image processing), розпізнавання образів (computer vision).*** Ми розглядатимемо в основному лише задачі першого напряму.

2.2. Кілька фактів з історії розвитку комп'ютерної графіки

Точка відліку початку розвитку комп'ютерної графіки – 1930 р., коли була винайдена електронна трубка.

Початком ери комп'ютерної графіки вважається 1951 р., коли в Массачусетському технологічному інституті для військово-морського флоту відбулася перша спроба використати дисплей для виведення графічного зображення. Наступним кроком у розвитку комп'ютерної графіки стала розробка графічної системи Sketch Pad, яку розробив Сазерленд у 1961 р. Ця програма для зображення на екрані простіших фігур використовувала світлове перо. Отримані картинки можна було зберігати й відновлювати. У цій програмі, крім ліній, точок, уже можна було малювати й прямокутник, що задавався своїми розмірами й розташуванням.

У 1960 р. компанія General Motors представила систему автоматизованого проєктування автомобіля з використанням комп'ютерних технологій. У 1961 р. створена перша комп'ютерна ігра Spacewar (зоряна війна).

У 1970-х роках відбувся різкий стрибок у розвитку обчислювальної техніки, винайшли мікропроцесор, в результаті чого почалася мінітюаризація комп'ютерів і ріст їх продуктивності. У цей час виникає індустрія комп'ютерних ігор і комп'ютеризується кіноіндустрія.

У середині 1970-х років графіка вже може створювати реалістичні зображення. У цей час з'являється алгоритм Фонга зафарбовування

поверхонь, алгоритми текстурування поверхонь, растрові алгоритми Брезенхема для побудови відрізків, кола, еліпса, метод Z-буфера, реалізовано побудову зображення напівпрозорої поверхні. У 1980 р. розроблені загальні принципи методу трасування променів для підвищення реалістичності зображень.

У зв'язку з успіхами в комп'ютерній графіці великі корпорації стали проявляти інтерес для неї, що в свою чергу стимулювало прогрес її розвитку. У 1980-тих роках з'являється цілий ряд компаній, що займаються прикладними розробками в галузі комп'ютерної графіки. У 1982 р. створюється Silicon Graphics, тоді ж виникає Ray Tracing Corporation, Adobe System та ін.

Отже у процесі розвитку комп'ютерної графіки виділяються кілька етапів. У 1970-ті вона формується як наукова дисципліна, розробляються основні методи та алгоритми комп'ютерної графіки, растрові алгоритми побудови примітивів, алгоритми відсікання, алгоритми усунення невидимих ліній та граней, моделювання освітлення тощо. У 1980-ті комп'ютерна графіка розвивається як прикладна дисципліна. Розробляються методи її застосування у різних галузях людської діяльності. У 1990-ті роки методи комп'ютерної графіки через графічні зображення стають основним засобом організації діалогу "людина–комп'ютер" і залишаються такими по теперішній час.

2.3. 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 готові 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:

- точку,
- відрізок,
- ламану лінію (замкнену і незамкнену),
- трикутник,
- трикутний стріп (рис. 2.1, *а*),

- трикутний фен (рис. 2.1, *б*),
- чотирикутний стріп (рис. 2.1, *в*),
- багатокутник (полігон).

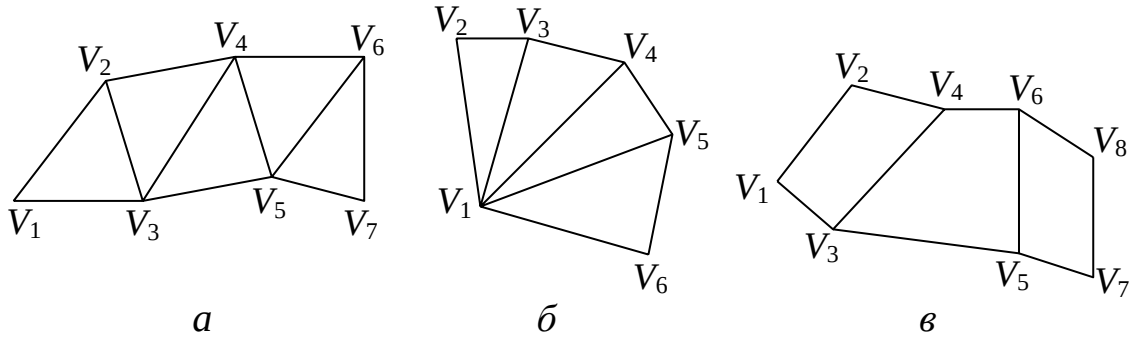


Рис. 2.1. Елементи множини примітивів: *а* – трикутний стріп; *б* – фен; *в* – чотирикутний стріп

Трикутний стріп – це множина трикутників $T = \{V_iV_{i+1}V_{i+2}\}$, що утворюється послідовністю вершин $V_1,V_2,V_3,...,V_{n+2}$. Перший трикутник стріпа $V_1V_2V_3$. Другий трикутник $V_2V_3V_4$ містить уже відоме ребро V_2V_3 і тому для його задання необхідно вказати тільки одну вершину і т. д. Тому стріп із 5 трикутників (рис. 2.1, *а*) задається послідовністю $V_1,V_2,...,V_7$ 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 $T = \{V_1V_{i+1}V_{i+2}\}$, утворених послідовністю вершин $V_1,V_2,...,V_{n+2}$ (рис. 2.1, *б*), тобто трикутників, які задаються першою і кожною наступною парою вершин. При заданні фену спочатку вказується загальна вершина, а потім решта вершин упорядку обходу спільних ребер.

Чотирикутний стріп – це множина чотирикутників $Q = \{V_{2i-1}\ V_{2i}\ V_{2i+2}\ V_{2i+1}\}$, утворених послідовністю вершин $V_1,V_2,...,V_{2n+2}$. Як видно з рис. 2.1, *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 самому 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.

Ланцюг вершин називається монотонним, якщо будь-яка вертикальна лінія перетинає його не більше одного разу. Полігон, складений з двох таких ланцюгів, називається ***монотонним***. Тобто простий полігон є *монотонним* відносно деякої прямої *L*, якщо будь-яка пряма, перпендикулярна до *L*, перетинає його лише в двох точках.

Для програмних 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 усередині цієї області. Будь-яка вершина в опуклому полігоні опукла (в неопуклому полігоні принаймні одна вершина неопукла). З цього випливає той факт, що при обході границі опуклого полігону 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.

У сучасних системах комп'ютерної графіки інформація про графічні об'єкти поділяється на дві категорії: параметри та атрибути. Примітиви теж характеризуються параметрами й 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 багатокутників – режим контуру чи заповнення. Цей параметр вказує спосіб задання багатокутників. Можна малювати тільки вершини або ребра, а можна зображати зафарбовані багатокутники, 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. Кілька граней у просторі складають просторовий об'єкт у вигляді замкненої полігональної поверхні (багатогранника), або незамкненої поверхні (полігональної сітки). На практиці часто

криволінійні граничні поверхні апроксимують системою полігональних граней, що приводить до утворення полігональних сіток. Полігональна сітка – найпростіша форма задання поверхонь.

Полігональна сітка – це сукупність плоских полігонів, з'єднаних між собою в такий спосіб, що кожне ребро належить не більше ніж двом багатокутникам. Кожне ребро є межею суміжних полігонів, а кожна вершина належить принаймні двом ребрам.

Існує кілька способів описання поліедрів та полігональних сіток. ***Прямий*** (явний) метод передбачає їх опис як сукупність окремих багатокутників їхніми вершинами, тобто ***модель поліедра*** (*полігональної сітки*) $H = \{P, G\}$ складається з двох масивів:

- координатного списку вершин $P = \{P_1, \ldots, P_n\}$, пронумерованих у довільному порядку. Завдяки однорідності елементів списку вершини поліедра можна задавати у вигляді матриці вершин;
- масиву списків граней (багатокутників) $G = \{G_1, \ldots, G_m\}$. Елемент $G_i = \{g_{i1},..., g_{in}, g_{i1}\}$ масиву G є списком номерів вершин полігона i-тої грані, перерахованих у порядку обходу їх по замкненому контуру. Масив G містить відомості про порядок з'єднання ребрами вершин полігона, що задані у масиві P (усі послідовні вершини, а також перша й остання з'єднуються).

Для прикладу побудуємо модель поліедра зображеного на рис. 2.2

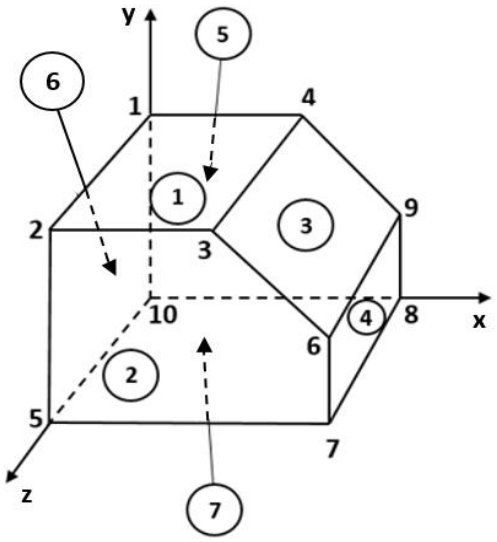


Рис. 2.2. Модель поліедра

У цього поліедра маємо 10 вершин та 7 граней, тому модель поліедра має вигляд

$$P = \begin{pmatrix} 0 & 0 & x_3 & x_4 & 0 & x_6 & x_7 & x_8 & x_9 & 0 \\ y_1 & y_2 & y_3 & y_4 & 0 & x_6 & 0 & 0 & y_9 & 0 \\ 0 & z_2 & z_3 & 0 & z_5 & z_6 & z_7 & 0 & 0 & 0 \end{pmatrix},$$

$G=\{G_1, G_2, G_3, G_4, G_5, G_6, G_7\}$.

$G_1=\{1, 2, 3, 4, 1\}$, $G_2=\{3, 2, 5, 7, 6, 3\}$, $G_3=\{6, 9, 4, 3, 6\}$, $G_4=\{6, 7, 8, 9, 6\}$, $G_5=\{1, 4, 9, 8, 10, 1\}$, $G_6=\{1, 10, 5, 2, 1\}$, $G_7=\{5, 10, 8, 7, 5\}$.

Для кожного окремого багатокутника цей спосіб ефективний, проте для полігональної сітки дає втрати пам'яті внаслідок дублювання спільних вершин (це не ефективно, оскільки дані, які потрібно зберігати, будуть повторюватися). При цьому пошук усіх багатокутників, що мають спільні вершини, вимагає порівняння трійок координат вершин одного багатокутника з трійками координат вершин усіх інших багатокутників, що не ефективно.

Більше можливостей дає спосіб опису полігональних сіток, який використовує ***списки вершин***. У цьому випадку інформація про поліедри та полігональні сітки зберігається у вигляді переліку координат вершин (кожна вершина записується один раз у масив вершин $V = ((x_1, y_1, z_1), (x_2, y_2, z_2), \ldots (x_m, y_m, z_m))$. А потім кожний багатокутник задається порядковими номерами вершин у списку вершин. При цьому економиться пам'ять і можна легко змінювати вершини сітки, проте ще непросто відшуковувати багатокутники зі спільними ребрами.

Ще для задання полігональних сіток більш ефективне ***явне задання ребер***. У цьому представленні є список вершин і список ребер, в якому ребра зустрічаються лише один раз.

Кожне ребро в списку ребер указує на дві вершини в списку вершин, що визначають це ребро, а також на один або два багатокутники, яким це ребро належить. Отже, багатокутники розглядаються як сукупність вказівників на елементи списку ребер, тобто ми описуємо багатокутники як $P = (E_1, E_2, \ldots E_l)$, де ребро $E = (V_1, V_2, P_1, P_2)$. Якщо ребро належить одному багатокутнику, то P_1 або P_2 порожні.

При цьому полігональна сітка зображається шляхом викреслювання не всіх багатокутників, а усіх ребер. Окремі багатокутники теж досить просто зображаються.

Для роботи з поліедрами необхідно вміти знаходити зовнішню нормаль до граней поліедра. Вона знаходиться за формулою векторного добутку $\boldsymbol{N} = (p_2 - p_1) \times (p_3 - p_2)$, де p_1, p_2, p_3 – три перші неколінеарні точки грані. Але, щоб так знаходити зовнішню нормаль, потрібно вершини для кожної грані G_i задавати в додатному напрямку, тобто проти годинникової стрілки з точки зору зовнішнього спостерігача. Якщо полігон грані неопуклий, то перші три вершини повинні складати опуклий кут.

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 поверхнонь 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. Для поліпшення апроксимації поверхні полігональною сіткою потрібно збільшувати кількість полігонів. Наприклад, у сучасних іграх у графічній сцені використовується до 1 млн ÷ 3 млн полігонів. Якість зображення об'єктів залежить від якості апроксимації і зумовлює подальше зростання кількості полігонів у моделях, а це призводить до додаткових витрат пам'яті і часу роботи алгоритмів, що працюють з такими зображеннями.

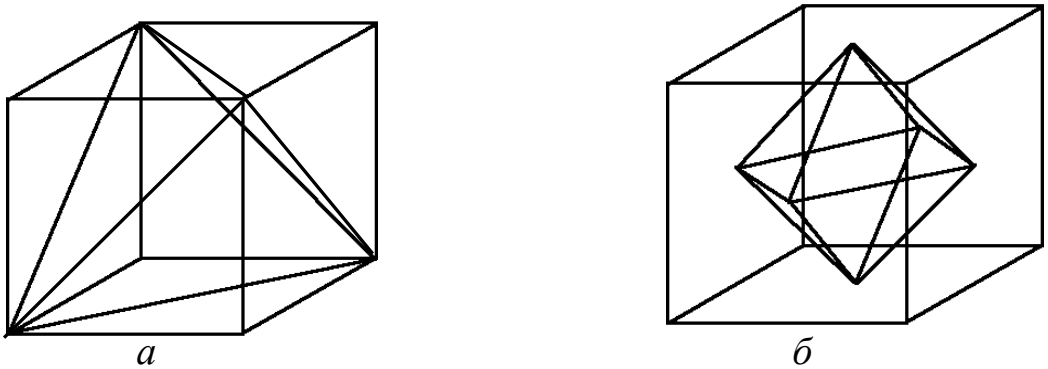
У природі існує цілий ряд поліедральних об'єктів. Серед них правильні багатогранники, призми, піраміди тощо.

Правильні багатогранники *(платонові тіла)* – це опуклі багатогранники, всі грані яких є правильними багатокутниками і всі багатогранні кути при вершинах рівні між собою.

Існує 5 правильних багатогранників (це довів ще Евклід): тетраедр, гексаедр (куб), октаедр, додекаедр, ікосаедр. Префікс перед грецьким коренем 'едр' означає кількість граней поліедра.

Для повного опису правильних багатогранників 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. 2.3, *а*). Вершини октаедра знаходяться в центрах граней куба (рис. 2.3, *б*).



а *б*

Рис. 2.3. Приклади простіших поліедрів: *а* – тетраедр; *б* – октаедр

Опишемо спосіб конструювання ікосаедра. Побудуємо циліндр одиничного радіуса, вісь якого збігається з віссю аплікат *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. 2.4.

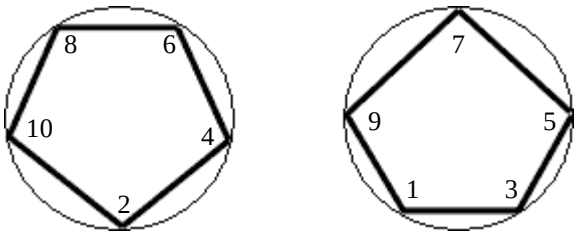


Рис. 2.4. Послідовність з’єднання вершин

Далі нумеруємо ці точки проти годинникової стрілки парними та непарними числами і потім послідовно, відповідно до нумерації, з’єднуємо ці точки відповідними відрізками (рис. 2.5). Одержуємо десять середніх трикутних граней ікосаедра.

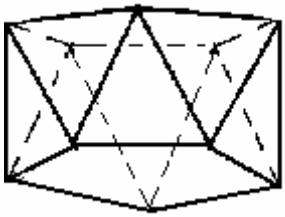
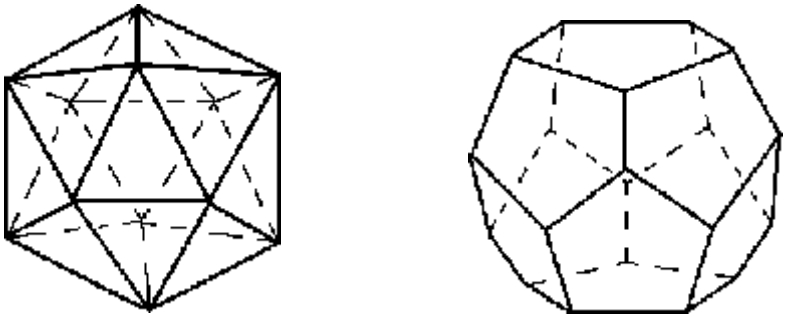


Рис. 2.5. Середні грані ікосаедра

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 збігаються 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 багатогранник із вершинами в цих точках буде мати 20 граней, кожна з яких є правильним трикутником (рис. 2.6, *а*). Декартові координати вершин побудованого ікосаедра легко обчислюються.



а *б*

Рис. 2.6. Складніші поліедри: *а* – ікосаедр; *б* – додекаедр

Залишається побудувати додекаедр (кількість граней – 12). Вершини додекаедра є центрами трикутних граней ікосаедра. Отже, координати кожної вершини додекаедра можна знайти шляхом обчис-

лення середнього арифметичного координат вершин ікосаедра. Далі необхідно з'єднати центри суміжних граней ребрами (рис. 2.6, *б*).

Множина видимих на екрані об'єктів, упорядкованих згідно з просторовими відношеннями (наприклад, за відношенням дальше – 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 відомості про форму, структуру кожного елемента незалежно від його розміщення та розмірів і місцезнаходження спостерігача. Методи моделювання змін форми об'єкта, тобто поступового перетворення одного образа 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 елементів сцени, параметри взаєбагато 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.

2.4. 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, багатокутників

тощо) і тривимірних (кубів, пірамід, куль, циліндрів, конусів 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 у 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 – Pr1UPr2 (графічний об'єкт, кожна точка якого належить хоча б одному з примітивів), перетин (добуток) примітивів Pr1∩Pr2 (тіло, кожна точка якого належить одночасно двом примітивам), різниця двох примітивів Pr1\Pr2 (тіло, яке складається з точок Pr1, що не належать до Pr2).

Отже, для підвищення ефективності роботи програміст повинен мати створену бібліотеку графічних функцій, процедур і алгоритмів, що зробить програми модульними, більш зрозумілими, зручнішими у налагодженні та внесенні коректив у структуру моделі.

Більшість графічних редакторів надають можливість використання примітивів для конструювання різних форм.

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 $y = \kappa x + b$. Відрізок, що визначається двома точками p_0, p_1, задається векторно-параметричним рівнянням вигляду $p = p_0(1 - t) + p_1 t$, де $p = (x, y)$, $t \in [0, 1]$.

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.*

Прикладом параметричних кривих є ***циклічні*** *(циклоїдальні) криві.* Простіші циклічні криві на площині утворюються додаванням рівномірного обертального руху точки по колу, можливо, змінного радіуса, з переносним поступальним рухом центра кола по кривій $p_ц(t)$ і описуються рівнянням $p(t) = p_ц(t) + r(t)c(wt)$, де $r(t)$ – закон зміни радіуса кола, $p = (x, y)$, $c(wt) = (sinwt, coswt)$ – параметричне рівняння одиничного кола із центром у початку координат, w – частота відносного обертання точки й дорівнює кількості обертів точки при зміні t на величину 2π (рис. 2.7).

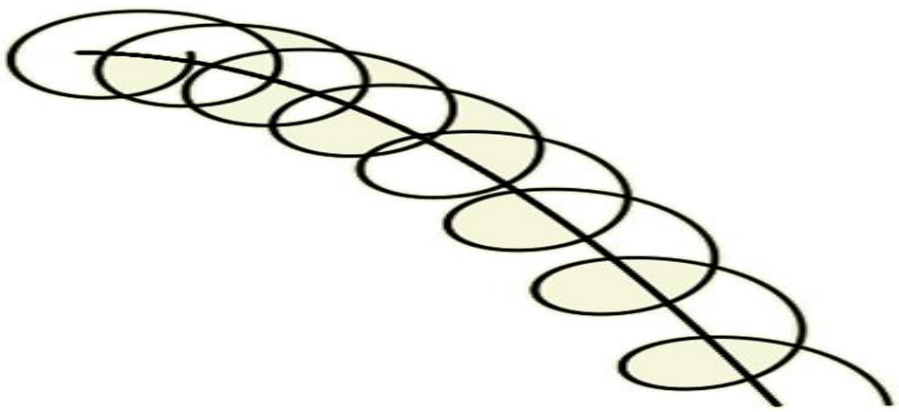


Рис. 2.7. Циклічна крива

Випишемо рівняння деяких циклічних кривих на площині.

Циклоїда – це траєкторія руху точки кола радіусом r, яке котиться по прямій $y = 0$ (рис. 2.8). Координати (рівняння) циклоїди мають вигляд

$$x(t) = r(t - sint),\ y(t) = r(1 - cost),\ t > 0.$$

Епіциклоїда – це траєкторія руху точки кола радіусом r, якщо воно котиться зовні кола радіусом r_0 (рис. 2.9, *а*). Її параметричне рівняння:

$$p(t) = (r_0 + r)(sint, cost) - r(sinwt, coswt),\ w = (r_0/r + 1),\ t > 0.$$

Гіпоциклоїда – це траєкторія руху точки кола радіусом r, яке котиться всередині кола радіусом r_0 (рис. 2.9, *б*). Рівняння гіпоциклоїди має вигляд:

$$p(t) = (r_0 - r)(sint, cost) + r(-sinwt, coswt),\ w = (r_0/r - 1),\ t > 0.$$

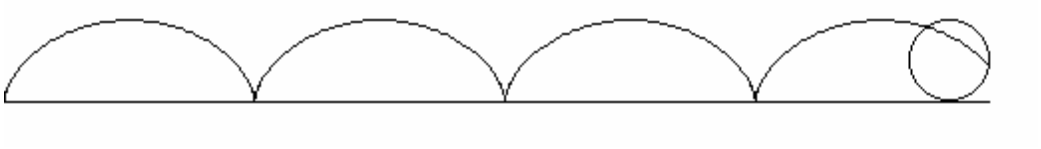
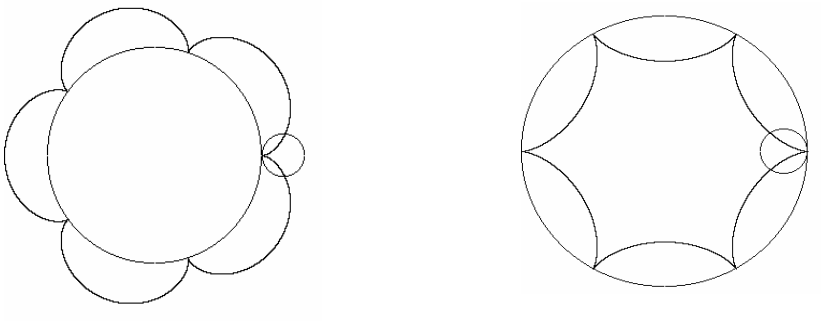


Рис. 2.8. Циклоїда



а *б*

Рис. 2.9. Циклічні криві: *а* – епіциклоїда; *б* – гіпоциклоїда

Часто трансцендентні криві задаються в полярній системі координат, тобто функцією $r = r(\varphi)$, де φ – полярний кут. Покладаючи $t = \varphi$, на основі формул зв'язку між координатами в декартовій і полярній системах координат переходимо до параметричних рівнянь кривої вигляду

$x(t) = r(t)\cos(t)$, $y(t) = r(t)\sin(t)$, або $p(t) = r(t)c(t)$, де $c(t) = (\cos(t), \sin(t))$.

Наприклад, равлик Паскаля описується такими рівняннями в полярній і параметричній формах:

$r(\varphi) = a\cos(\varphi) + b$, $p(t) = (a\cos(t) + b)c(t)$, де $c(t) = (\cos(t), \sin(t))$.

До речі неявне рівняння цієї кривої має вигляд $(x^2 + y^2 - ax)^2 - b^2(x^2 + y^2) = 0$, яке зовсім незручне для побудови графіка функції.

У комп'ютерній графіці для моделювання графічних об'єктів (кривих і поверхонь) використовується кінематичний метод. Його суть полягає у неперервному формуванні за допомогою афінних перетворень проміжних станів об'єкта на основі початкових станів як функції деякого параметра. Кінцевим результатом такого підходу є зображення на екрані. Побудову зображення легше виконувати не за складними аналітичними рівняннями об'єкта, а безпосередньо за векторною моделлю, задавши початкове положення об'єкта, необхідні матриці перетворення та інтервали зміни параметрів.

Наприклад, для побудови зображення одиничного кола кінематичним методом (рис. 2.10, *а*) досить задати:

- початкове положення точки $p(0)$ =(1, 0);
- векторне рівняння, яке описує обертання точки $p(0)$ навколо початку координат $p(t)= p(0) \cdot R(t)$, де $R(t)$ матриця повороту;
- значення параметра t, яке змінюватиметься в інтервалі [0, 2π] з кроком дискретизації $\frac{2\pi}{n}$.

Крива, яка будується на екрані кінематичним методом зображується послідовністю точок $p(t_i)$, які з'єднуються відрізками прямих, утворюючи полілінію.

Для того щоб ламана лінія була гладкою кривою, потрібно щільно розміщувати точки, віддаль між сусідніми точками повинна бути порядку одного пікселя. Оптимальна щільність точок пропорційна кривизні кривої. Оскільки крива кола має сталу кривизну, то точки t_i, i=1, 2, …, n рівномірно розподілені на [0, 2π].

Існують ще і некіматичні параметричні рівняння кола, наприклад

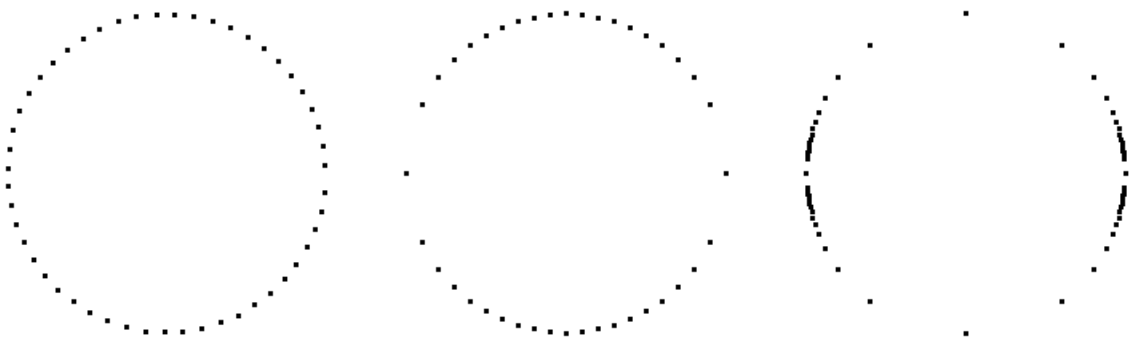
$$x(t)=\frac{1-t^2}{1+t^2}, \quad y(t)=\frac{2t}{1+t^2}.$$

Для цього рівняння сталий крок по t приводить до нерівномірного розподілу точок на колі (рис. 2.10, *в*). Крім цього точка (–1, 0) відповідає значенню $t = \pm\infty$.

Невдала для кола також некінематична модель, в якій теж ми маємо нерівномірний розподіл точок на колі при зміні t зі сталим кроком (рис. 2.10, *б*):

$$x(t)=t, \quad y(t)=\sqrt{1-t^2}, \quad t \in [-1, 1].$$

Тому для якісного зображення об'єктів на екрані досить важливо вміти правильно підбирати рівняння моделі та їх параметри.



а *б* *в*

Рис 2.10 Графіки кіл, побудовані за різними рівняннями

Параметричні рівняння довільного еліпса на площині з центром в точці $p_0(x_0, y_0)$ півосями a і b кутом нахилу великої півосі α можна одержати за рахунок трьох перетворень (масштабування, поворот і зсув) одиничного кола $c(t)$ = (cos$t$ sin$t$). А саме

$$p(t) = (\cos t \ \ \sin t)\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}\begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} + p_0,\ t \in [0, 2\pi],$$

або в координатній формі

$x(t) = a\cos\alpha\ \cos t - b\sin\alpha\ \alpha\sin t + x_0,$

$y(t) = b\sin\alpha\ \cos t + b\cos\alpha\ \alpha\sin t + y_0.$

Оскільки модуль вектора руху точки по еліпсу залежить від параметра t, то при апроксимації еліпса полігоном щільність розподілу його вершин $p(t_i)$ повинна бути змінною: максимальна поблизу апогея і мінімальна біля перигея.

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

Параметричну модель будь-якої поверхні $p(\tau, t)$ можна зобразити параметричними лініями $p(\tau_i, t)$ $p(\tau, t_j)$, $i = 0,1,\ldots n$, $j = 0,1,\ldots m$. Побудувавши поздовжні $p(\tau_i, t)$ та поперечні $p(\tau, t_j)$ лінії, ми одержимо каркас поверхні.

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 площини має вигляд $p(\tau, t) = p_0 + \boldsymbol{V}t + \boldsymbol{W}\tau$, де p_0 – точка на площині, $\boldsymbol{V}$, $\boldsymbol{W}$ – два напрямні лінійно незалежні вектори, що визначають площину.

У скалярній формі параметричне рівняння площини задається співвідношеннями

$$x(\tau, t) = x_0 + V_x t + W_x\tau,\ \ y(\tau, t) = y_0 + V_y t + W_y\tau,\ \ z(\tau, t) = z_0 + V_z t + W_z\tau.$$

Параметричне рівняння сфери має вигляд

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

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. 2.11;

– нові вершини проєктуються на поверхню сфери. Для цього із центра сфери через вершини проводяться промені й 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.

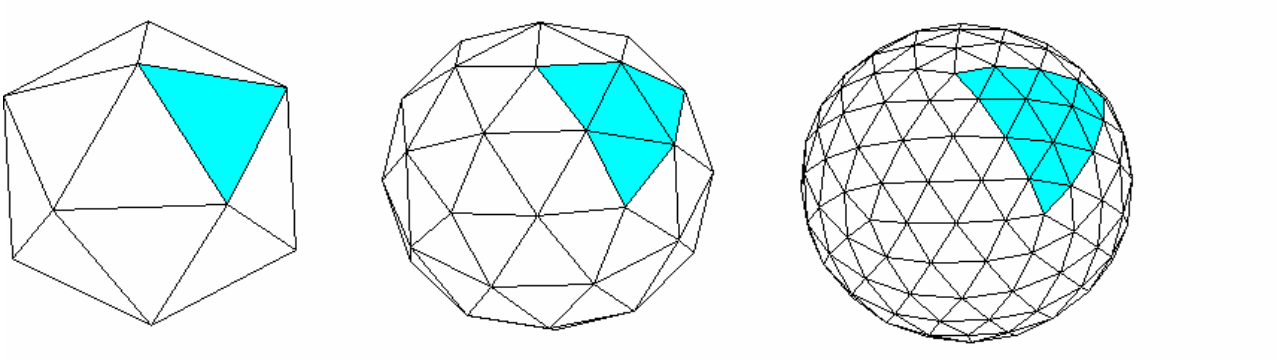


Рис. 2.11. Побудова зображення сфери

Бічну поверхню кругового циліндра з радіусом основи r і висотою H можна побудувати шляхом обертання прямої навколо однієї з осей координат на 360^0. У випадку коли обертання здійснюється навколо осі Oy, кінематична модель кругового циліндра має вигляд

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

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

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

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 $r = r_1(v)(1-u) + r_2(v)u$.

Найбільш поширеним способом представлення тривимірних поверхонь є полігональні сітки і параметричні бікубічні сплайни, які розглянемо в розділі 8. Для зображення поверхонь із заданою точністю потрібно значно менше бікубічних сплайнів, ніж при апроксимації полігональною сіткою. Проте алгоритми для роботи з бікубічними сплайнами складніші за алгоритми для роботи з полігональною сіткою.

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 $x = x(t),\ y = y(t),\ z = z(t)$.

Лінія має один степінь вільності. Такі методи, при яких лінії та поверхні отримуються шляхом руху відповідно точки й лінії, називаються *кінематичними*.

Неявна форма вимагає розв'язування системи нелінійних рівнянь відносно координат x, y, z (точки p), що для практичного застосування незручно.

Параметричне рівняння прямої лінії в просторі має вигляд:
$p(t) = p_0 + Vt$, або $x(t) + x_0 + V_x t$, $y(t) = y_0 + V_y t$, $z(t) = z_0 + V_z t$, $t \in (-\infty,\infty)$, де $V = (V_x, V_y, V_z)$ – напрямний вектор прямої лінії.

Прикладом параметричної просторової кривої є циліндрична гвинтова лінія (рис. 2.12), 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 бічній 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.

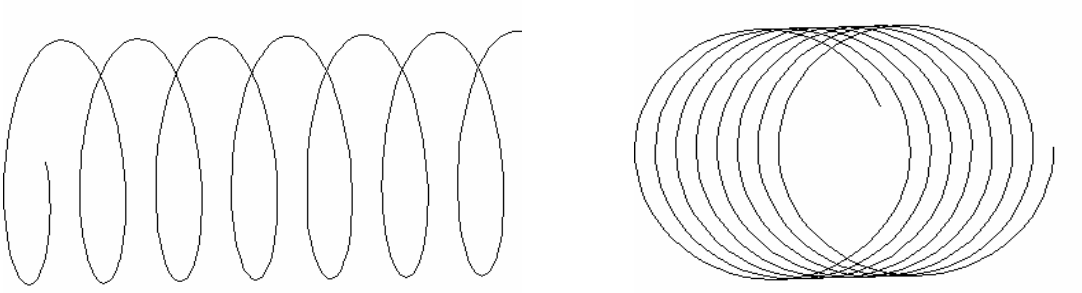


Рис. 2.12. Циліндрична гвинтова лінія

У більш загальному випадку гвинтова лінія утворюється додаванням обертального руху точки навколо нерухомої осі і перенесенням її вздовж цієї осі за певним законом.

Параметричне рівняння гвинтової кривої

$$p(t) = p_ц(t) + r(t)c(t),\ 0 \le t \le 2\pi n,$$

де $r(t)$ – закон зміни радіуса кола, $p = (x, y)$, $c(t) = (\sin t, \cos t)$ – параметричне рівняння одиничного кола із центром у початку координат, $p_ц(t)$– закон переносного руху точки, n – кількість витків.

Розглянемо ще приклад обертання точки по спіралі Архімеда, тобто $r(t) = r_0 + r_1 t$. Рівняння гвинтової лінії при цьому має вигляд $x(t)=(r_0 + r_1 t)\cos(t)$, $y(t) = (r_0 + r_1 t)\sin(t)$, $z=ht$.

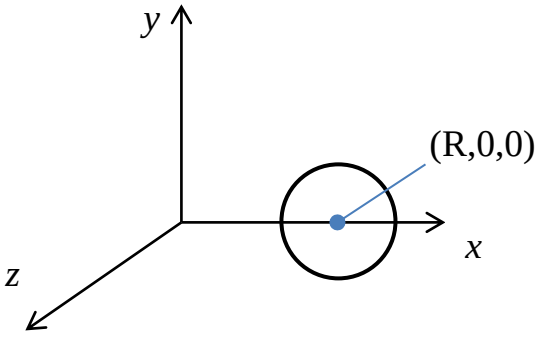


Рис. 2.13. Формування тора

Торова спіральна лінія одержується додаванням руху точки по колу з частотою w з обертанням кола навколо однієї з осей координат.

У площині *xy* побудуємо коло радіуса *r* з центром у точці (*R*,0,0) і виконаємо його обертання навколо осі *Oy* (рис. 2.13).

Закон руху точки по колу має вигляд

$x(t) = r\cos(wt) + R, \quad y(t) = r\sin(wt), \quad z = 0.$

Тепер точку з координатами *x, y, z* будемо обертати навколо осі *Oy*, одержимо параметричне рівняння торової спіральної кривої

$x(t) = (r\cos(wt) + R)\cos(t), \quad y(t) = r\sin(wt), \quad z = (r\cos(wt) + R)\sin(t).$

Графік торової лінії наведений на рис. 2.14.

Зауважимо, що маючи математичні моделі ліній та поверхонь, для побудови їх графіків можна використовувати універсальні математичні пакети, зокрема MathCAD.

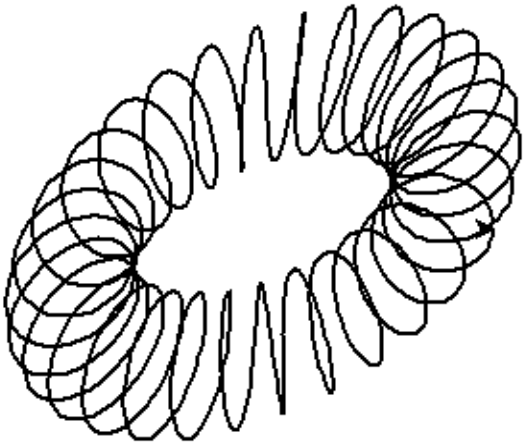


Рис. 2.14. Торова лінія

2.5. Застосування комп'ютерної графіки

У наш час комп'ютерна графіка застосовується майже в усіх галузях людської діяльності і частка графічних даних неупинно зростає. Нині графічний інтерфейс користувача став основним засобом спілкування людини з ЕОМ, усі сучасні операційні системи використовують графічні елементи управління. Без комп'ютерної графіки не обходиться жодна мультимедійна програма. Комп'ютерна графіка має найрізноманітніші застосування в:

1. системах автоматизації проектування, конструювання (інженерна графіка) та розробки дизайну;
2. автоматизованих системах наукових досліджень;
3. інформаційних системах;
4. системах ілюстративної та ділової графіки;
5. системах машинної геометрії;
6. анімаційних та мультимедійних задачах (поєднання графіки зі звуком);
7. комп'ютерних іграх;
8. відеотренажерах (тренування льотчиків, диспетчерів, військових);

9. мистецтві, видавничій та рекламній діяльності, засобах масової інформації, криміналістиці, медицині та в інших галузях діяльності.

Зупинимося на вищезазначених сферах детальніше.

1. В галузі проектування і дизайну широко використовується математичне і геометричне моделювання. *Інженерна комп'ютерна графіка* (іншими словами, конструкторська) призначена для автоматизації креслярсько-графічних та конструкторських робіт, для створення і корегування графічної документації (креслень) в електронній формі. При цьому комп'ютер виконує більшу частину рутинної роботи з проєктування, вивільняючи час інженера-конструктора для творчої діяльності. Тим самим підвищується якість проєктування.

Сучасна інженерна графіка – це складний програмний комплекс, що дозволяє вести наскрізне проєктування, від постановки задачі та математичного моделювання до випуску всієї необхідної виробничо-технологічної документації, й використовується в усіх галузях зв'язаних із проєктуванням. 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. При цьому ставляться високі вимоги до точності параметрів створюваних об'єктів, якості та швидкості відображення об'єктів на екрані, чіткості, контрастності, правильної передачі кольорів, відсутності геометричних спотворень.

Головне практичне втілення інженерної комп'ютерної графіки – це системи автоматизованого проектування (САПР). У САПР комп'ютерна графіка використовується для проєктування пристроїв та їх складових. Кінцевою метою САПР є випуск креслень деталей, вузлів, пристроїв. Сьогодні існуючі САПР дозволяють швидко створювати креслення і виконувати точні розрахунки. Системи типу САПР, активно використовуються в багатьох галузях (наприклад, у машинобудуванні, електроніці, будівництві, архітектурі). Одними з перших були створені САПР для проєктування 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, має можливість вибору варіантів, 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 проєкції з усуненням невидимих ліній, проводити повний аналіз цих деталей. Провідні інженерні компанії прагнуть до повного цифрового представлення різноманітних конструкцій. САПР також оснащується засобами графічного моделювання, що дозволяє не тільки виконати проектування, а й проаналізувати його функціональні можливості.

Використання КГ у проєктуванні 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'ютери 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 1987 р. використовує комп'ютерну графіку для архітектурного дизайну, розміщення посадкових місць, планування приміщень тощо. Відомо ряд ефективних застосувань у проєктуванні 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 CAD-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 формування наукової документації, 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 з наочним поданням результатів.

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 символьні зображення. 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 сприяння пізнанню, тобто 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 символьного 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 наочність, 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). Типовими для кожної ГІС 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. Ілюстративна комп'ютерна графіка дозволяє використовувати персональний комп'ютер як інструмент для художника. Спеціальні програми перетворюють комп'ютер у віртуальну майстерню художника. Ілюстративна графіка має практичні застосування в усіх галузях людської діяльності: промисловості, сільському господарстві, бізнесі, менеджменті, телебаченні, пресі, освіті, поліграфії, шоубізнесі тощо.

Системи ілюстративної графіки призначені для створення та художньої обробки різних комп'ютерних зображень (малюнків, фотографій, карт тощо). Користувач систем ілюстративної графіки має можливість
– вводити зображення;
– будувати власні зображення шляхом компонування 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 і сприймання їх у зручній формі у вигляді графіків, схем, діаграм. Прикладами таких систем є MS Excel, Quattro Pro, SuperCalc, PowerPoint, Boeing Graph тощо.

5. Поява комп'ютерів сприяла автоматизації геометричного моделювання з використанням засобів машинної графіки. Геометричне моделювання – це розв'язування різних геометричних задач у 2D-вимірному, 3D-вимірному та *n*-вимірному просторах. Геометричні моделі тривимірних сцен складні, тому вимагають розробки та застосування розвинутого програбагато забезпечення. Користувач повинен мати можливість оперувати геометричними об'єктами та здійснювати операції над ними, тобто користувач оперує категоріями, а результати роботи програми одержуються як у символьній, 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 (рухомих) 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 (елемента зображення) на екран;
– стирання спрайта;
– виведення з деяким зсувом іншого варіанта спрайта.

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 замовчуваням;
– 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; редагування фільмів. Вони застосовуються для автоматизації підготовки мультфільмів, відеокліпів, створення художніх фільмів із 3D-спецефектами.

Відеопродукція ставить високі вимоги до реалістичності створюваних сцен. Дійсно, в деяких випадках не можна розрізнити об'єкти, що існують у реальному світі, та розроблені на комп'ютері. Фотореалістична графіка високої якості вимагає великих затрат ресурсів, тому візуалізацією кадрів займаються мережі комп'ютерів або суперкомп'ютери. Обробка одного кадра може зайняти кілька годин, а окремий епізод з фільма може зайняти декілька днів на потужному комп'ютері. Тому в кіновиробництві використовуються сучасні графічні станції та спеціалізовані програми.

Один із перших суперкомп'ютерних центрів був створений саме на студії Діснея і використовувався для створення мультфільмів. Це пояснюється тим, що при створенні мульфільмів 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", 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 розмірів анімаційного зображення, володіє широкими можливостями роботи зтекстом.

Macromedia Flash – прийнятий стандарт для використання в анімації векторних зображень, що 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. Ігрова поведінка – одна з потреб людини. Комп'ютерні ігри мають широке застосування (розваги, освіта, спорт, медицина та ін.) завдяки позитивним якостям, як-от сприяння розвитку мислення, вироблення навичок швидкої реакції, розвиток соціальних навичок, оптимізація діяльності, навчання в різних предметних галузях тощо. У США комп'ютерні ігри визнані окремим видом мистецтва. З розвитком мережі Internet помітно поширилися комп'ютерні ігри в режимі реального часу. Розповсюдження мобільних телефонів та смартфонів привело до розвитку індустрії ігор для цих пристроїв. Нині набирає обертів кіберспорт, тобто змагання на основі відеоігор. Індустрія комп'ютерних ігор за прибутковістю перевищує кіноіндустрію. В Україні теж існують комп'ютерні фірми, які займаються розробкою відеоігор.

Індустрія розваг активно використовує методи комп'ютерного синтезу для створення численних комп'ютерних ігор, в яких віртуальні персонажі діють у віртуальних комп'ютерних світах. Комп'ютерна гра забезпечується комп'ютерною програмою, яка призначена для ведення ігрового процесу, вона передбачає взаємодію людини (групи людей) із комп'ютером або декількох людей між собою за допомогою комп'ютера. Ігри занурюють гравців у віртуальну реальність, під час гри створюється імітація взаємодії користувача з віртуальними персонажами. Аналізуючи ігрову ситуацію на екрані, користувач може впливати на її хід за допомогою спеціальних програм та апаратних засобів.

Системи віртуальної реальності 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 швидкодією та можливостями побудови високореалістичних зображень. Засоби відображення можуть бути різними – від звичайних моніторів із високою роздільною здатністю до екранів на всю стіну або стереоскопічних систем відображення.

8. Розвиток засобів віртуальної реальності почався з досліджень по створенню авіаційних тренажерів. Тепер 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 тренувань і підготовки до роботи в реальному світі льотчиків, шоферів, диспетчерів, операторів складних хімічних, енергетичних установок, військових тощо. Наприклад, у кінці 70-х років для космічних кораблів "Шатл" з'явилися льотні тренажери, що базуються на КГ. Перші кроки в цьому напрямку 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 важелями управління, за допомогою яких можна змінити умови спостереження. В пам'яті ЕОМ закладені необхідні дані. Під час бою, змінюючи положення важелів, інформація надходить в ЕОМ і в режимі реального часу переформатовується видима картина. Тут симулятор пілотування літака використовується для навчання управлінням справжньою машиною. У процесі симуляції відточується високий рівень майстерності.

9. Комп'ютерна графіка використовується для зображення географічних та природних явищ (географічних та рельєфних карт, карт погоди, карт для розвідки нафти й газу, карт забруднень). Графічні системи використовуються для створення творів мистецтва, музейної та реставраційної діяльності, оздоблення інтер'єрів, підготовки рекламних роликів і відеокліпів, різних заставок і телеефектів, оформлення Web-сторінок та спецефектів у кіно, спрощення та скорочення часу підготовки друкованих матеріалів різного призначення, підвищення якості друкованої продукції, оформлення обкладинок книг і журналів.

У медицині КГ використовується в комп'ютерній томографії, для автоматизованого проєктування імплантантів (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.

Контрольні запитання 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. Опишіть їх 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/приклади застосування КГ (зокрема, 3D-графіки).
15. Коротко охарактеризуйте можливості найбільш відомих графічних пакетів.
16. Дайте визначення інженерної, ілюстраційної та ділової комп'ютерної графіки. 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. Розробити алгоритм аналізу простоти/опуклості багатокутника.
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. Знайти 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. Знайти на поверхні сфери чотири рівновіддалених між собою точки. *Вказівка: ці точки – вершини вписаного тетраедра.*
7. Розробити алгоритм визначення видимих та невидимих граней платонових тіл.
8. Вивести рівняння тора (тор утворюється обертанням кола відносно осей координат).
9. Вивести рівняння траєкторії руху точки кола радіусом r_0, якщо воно котиться зовні/всередині еліпса з півосями a та b.
10. Розробити алгоритм дедалі точнішої апроксимації сфери багатогранною трикутною поверхнею. Для цього візьміть на початку тетраедр і кожну його грань середніми лініями розбийте на чотири трикутники. Вершини нових трикутників із центра сфери спроєктуйте на сферу і на цих точках побудуйте нові трикутні грані, які краще апроксимують сферу і т. д.
11. Побудувати сферу мінімального радіуса, що обмежує n точок у просторі.
12. За матеріалами комп’ютерних періодичних видань проаналізуйте останні досягнення в галузі практичних застосувань комп’ютерної графіки. Наведіть кілька прикладів.

Розділ 3. Технічне та програмне забезпечення комп'ютерної графіки

Технічні засоби і системне програмне забезпечення (операційні системи) є інструментальним середовищем графічної системи, в якому виконуються програми комп'ютерної графіки (КГ). Вони реалізують різні функції, що створюють, перетворюють і зберігають графічну інформацію. Швидке зростання функціональних можливостей комп'ютерної техніки та розробка відповідного програмного забезпечення створила базу для подальшого інтенсивного розвитку КГ.

3.1. Технічне забезпечення комп'ютерної графіки

Технічні засоби КГ розв'язують такі задачі: введення графічної інформації, забезпечення спілкування користувача з графічною системою, обробка, відображення, збереження та документування графічної інформації.

До апаратного забезпечення комп'ютерної графіки належать обчислювальні системи, що включають центральні пристрої – процесори, оперативну пам'ять, графічну підсистему, запам'ятовуючі пристрої графічної інформації, засоби зберігання графічної інформації тощо та периферійні пристрої – засоби введення і виведення.

3.1.1. Загальні відомості про обчислювальні засоби

Для обробки графічної інформації використовуються різні ЕОМ, від простіших до багатопроцесорних супер ЕОМ.

До основних технічних параметрів комп'ютера відносять продуктивність, розрядність машинного слова, ємність оперативної пам'яті, ємність зовнішньої пам'яті та ін.

Продуктивність вимірюється кількістю операцій, що виконуються за одиницю часу. Продуктивність для різних типів ЕОМ складає від сотень тисяч до сотень трильйонів операцій за секунду.

Ємність оперативної пам'яті визначає можливості виконувати складні програми з обробкою великої кількості даних. Ємність оперативно запам'ятовуючих пристроїв змінюється в діапазоні від десятків мегабайт до декількох гігабайт. Ємність зовнішньої пам'яті складає від десятків гігабайт до одиниць терабайт.

Для обробки графічної інформації використовуються комп'ютери двох класів – універсальні та спеціалізовані. В більшості випадків це універсальні машини, спеціалізовані системи використовуються для розв'язування складних задач комп'ютерної графіки. Наведемо класи машин в порядку зменшення їх функціональних можливостей.

Супер ЕОМ – це багатопроцесорні ЕОМ (містять сотні і тисячі процесорів) максимальної продуктивності з швидкодією сотні млн, млрд операцій за секунду, мають велику оперативну пам'ять, що

становить десятки Гбайт, зовнішня пам'ять – десятки Тбайт, розрядність супер ЕОМ – 128 біт. Вони створені для розв'язування найскладніших задач, що вимагають великої кількості обчислень, наприклад моделювання атмосферних явищ, розв'язування астрономічних задач та ін. У сучасному світі без суперкомп'ютерів немислиме розв'язання ні економічних, ні фінансових, ні науково-технічних, ні оборонних загальнодержавних задач.

Сучасні суперкомп'ютери створюються по кластерній технології, за цією технологією комп'ютер складається з декількох десятків обчислювальних машин, що зв'язані між собою і функціонують як єдине ціле.

Існує проєкт Top500 (https://www.top500.org/) для складання рейтингу і описань 500 найпотужніших суспільно відомих комп'ютерних систем світу. З червня 1993 р. він оприлюднюється два рази на рік (перший в червні, другий в листопаді).

Лідером рейтинга серед 500 найпотужніших суперкомп'ютерів світу на листопад 2021 р. став суперкомп'ютер Fugaky фірми Fujitsu. Він знаходиться в Японії в центрі обчислювальних наук і використовується, зокрема, для аналізу великих даних, досліджень вірусів, створення нових ліків, кліматичних досліджень, розробки нових типів акумуляторів тощо. Перші результати він уже продемонстрував при підборі ліків, що полегшують перетікання хвороби COVID-19.

Fugaky складається з спеціальних процесорів, так що загальна кількість ядер складає 7 630 848. Згідно з методологією визначення швидкодії Linpack, Fugaky працює з реальною швидкодією 442,01 Пфлоп за сек ($442{,}01*10^{15}$ операцій з плаваючою крапкою за секунду), що більше 400 квадрильйонів операцій за секунду.

А в 2022 р. лідером став суперкомп'ютер Frontier фірми Cray з кількістю ядер 8730112 та швидкодією 1102 PFlops.

Великі ЕОМ (менфрейми) – це високопродуктивні надійні машини з великими обчислювальними ресурсами зі значним обсягом оперативної та зовнішньої пам'яті, з розвинутими засобами введення/виведення. Вони забезпечують багатокористувацький режим роботи (обслуговують одночасно до тисячі користувачів). Основні напрямки використання мейнфреймів – це розв'язування науково-технічних задач, робота з великими базами даних, управління роботою мереж, створення великих веб-вузлів. На мейнфреймах зараз знаходиться до 70% комп'ютерної інформації.

Робочі станції – це відносно недорогі, але достатньо потужні машини на базі сучасних процесорів, які використовуються для організації робочих місць в різних галузях. Робочі станції конфігуруються і для роботи зі складною графічною інформацією, вони

містять потужну графічну підсистему, яка багато графічних функцій реалізує апаратно.

Персональний комп'ютер (ПК) – це комп'ютер особистого використання. Можливості ПК щодо обробки інформації дуже широкі.

Наведемо основні напрямки використання ПК:

- математичні розрахунки;
- бази і банки даних;
- текстові редактори; видавництво і поліграфія;
- комп'ютерна та інженерна графіка, живопис;
- комунікація, обмін інформацією;
- Web-технології; моделювання об'єктів і явищ;
- навчання; розваги і дозвілля тощо.

Фірма IBM, яка випускала великі ЕОМ, в 1981 р. на базі 16-розрядного мікропроцесора Intel 8086 створила персональний комп'ютер, який згодом витіснив з ринку інші персональні комп'ютери. IBM PC став стандартом для персональних комп'ютерів. Сьогодні комп'ютери сумісні з IBM PC складають 90% усього парку ПК. Згодом й інші фірми почали збирати ПК на базі мікропроцесорів Intel та AMD.

Персональні комп'ютери бувають настільними і переносними. Настільні комп'ютери складаються із системного блока (центральна частина), монітора, клавіатури і миші. Сучасні переносні комп'ютери – це ноутбуки. ***Ноутбуки*** – це мобільні комп'ютери, вони функціонально схожі з настільними ПК, в них використовується теж саме програмне забезпечення й операційні системи. Конструктивно ноутбук містить дисплей на базі рідких кристалів, клавіатуру, яка суміщена з системним блоком, жорсткий диск і оптичний дисковод. Поряд із клавіатурою знаходиться маніпулятор для управління мишею.

Як і настільні комп'ютери, ноутбуки розрізняються типом процесора, чіпсета, системної плати, пам'яті, відеокарти, вінчестера тощо. Важливими характеристиками ноутбука є: розмір, вага, час автономної роботи від батареї, продуктивність.

В переважній більшості ноутбуків використовуються процесори фірми Intel, хоча зустрічаються моделі з чіпсетами AMD. Нову епоху в сфері мобільних комп'ютерів відкрила поява архітектури Core. Домінуюче місце займають процесори Intel Core з різними параметрами. Ноутбуки на базі цих процесорів мають високу продуктивність, якою не поступаються багатьом настільним системам. В ноутбуки інтегрується ряд пристроїв: мережевий адаптер, модуль Bluetooth, картовід (Card Reader), модуль безпровідного зв'язку Wi-Fi та ін. Мережевий адаптер призначений для під'єднання до локальної мережі і є практично в кожному сучасному ноутбуку.

Bluetooth забезпечує стійкий зв’язок. Використовуючи ноутбук з мобільним телефоном, можна входити в Інтернет (якщо такого модуля немає, то його можна придбати окремо і під’єднати до USB-порту).

Картовід, вбудований у ноутбук, дозволяє читати карти пам’яті, які використовуються в цифрових фотоапаратах і мобільних телефонах.

Модуль безпровідного зв’язку Wi-Fi забезпечує доступ в Інтернет у зонах Wi-Fi. Цей модуль дає змогу переміщуватись з ноутбуком в приміщенні без зайвих проводів і залишатися в мережі Інтернет.

Для підключення до ноутбука різних периферійних пристроїв використовуються порти USB. Через USB-порт до ноутбука можна під’єднати мишу, Flash-пам’ять, зовнішній оптичний DVD-RW дисковод, принтер, сканер, цифровий фотоапарат, безпровідну точку доступу та інші пристрої. Сучасні ноутбуки оснащені більш швидкісним портом USB 2.0. Нестачу USB-портів можна компенсувати окремим USB-концентратором.

На ноутбуці може ще бути рознімання Fire Wire, яке є менш розповсюдженим, ніж USB і використовується для підключення відеокамер та інших пристроїв. Практично на всіх ноутбуках наявне рознімання для під’єднання мультимедійного проєктора, що здійснює виведення зображення на великий екран.

Окремим класом ноутбуків є *нетбуки*, вони призначені в основному для роботи в мережі Інтернет і запуску додатків, що не вимагають значних ресурсів. Їхня особливість – це мала вага (менше 1 кг), невеликий екран (не більше 10,2 дюйма), наявність інтерфейсів Wi-Fi та Bluetooth.

3.1.2. Центральні апаратні засоби

До центральних пристроїв належить процесор, різні електронні схеми, оперативна пам’ять і підсистема введення.

Материнська плата

Найважливішим вузлом ПК є системна (материнська) плата. Основна функція материнської плати – забезпечити зв’язки (мости) між пристроями ПК. За всіма пристроями комп’ютера потрібний контроль, їх роботу треба координувати.

Материнська плата – це основна електронна схема ПК, на якій містяться основні компоненти комп’ютера.

Ось декілька пристроїв, з яких складається материнська плата:

- системна шина – магістраль, яка зв’язує пристрої ПК в єдине ціле. Саме по шині передаються сигнали керування та дані;

- базовий набір мікросхем логіки – *чіпсет*, за допомогою якого материнська плата здійснює контроль над усім, що відбувається всередині системного блока. У кожному чіпсеті є два мости (чіпи): північний, що з'єднує між собою процесор, оперативну пам'ять і відеошину AGP, та південний, що відповідає за роботу з під'єднаними до цієї шини периферійними пристроями. Чіпсет є основою будь-якої материнської плати, від нього залежить тип процесора, тип пам'яті та продуктивність материнської плати;
- схема BIOS. Основна функція BIOS – це забезпечити первинну роботу комп'ютера, управління стандартними периферійними пристроями, а саме дисководами, клавіатурою, принтером, таймером тощо. BIOS відшуковує і завантажує в ОП програму-вантажник операційної системи із системного диска в ОП.

Решта елементів розміщуються на окремих платах і вставляються в рознімні з'єднання на материнській платі – так звані *слоти*, що мають вигляд довгих гнізд. Кількість слотів розширення визначає, скільки можна вставити в комп'ютер додаткових плат. Відеокарта від'єднується через спеціальний слот, що має назву AGP або PCI Express, решта слотів називаються PCI. На материнській платі є слоти для установки ОП. Цих слотів може бути від 1 до 4. Слоти чітко прив'язані до типу ОП.

Мікропроцесори встановлюються на материнській платі в квадратні гнізда, що називаються *сокетами*. Ці гнізда схожі між собою, але вони відрізняються кількістю ніжок. Для різних груп мікропроцесорів існують різні материнські плати з відповідними гніздами для мікропроцесорів, наприклад, для процесорів Intel Core i9/i7 (мікропроцесори 11, 10 покоління) Socket 1200. Отже, материнську плату потрібно вибирати відповідно до мікропроцесора.

Крім цього, на материнській платі знаходяться розняття (слоти) для установлення модулів оперативної пам'яті, розняття для під'єднання нагромаджувачів жорстких дисків, дисководів, розняття для підключення електроживлення. На задню стінку ПК з материнської плати виведені рознімні з'єднання, що називаються *портами* для під'єднання зовнішніх пристроїв.

Нові конструкції системних плат для під'єднання клавіатури та миші підтримують вбудований порт USB (Universal Serial Bus).

Мікропроцесор (МП) *– це центральний пристрій, який виконує арифметичні та логічні операції, здійснює обробку інформації відповідно до виконуваної програми та управління обчислювальними процесами, координує роботу пристроїв системи.*

Сучасні мікропроцесори – це одна мікросхема, яка виготовлена з напівпровідникового кристалу кремнію з щільним пакуванням фізич-

них елементів, завдяки чому на кристалі площею близько 1 см^2 можна розмістити велику кількість елементів: транзисторів, кондесаторів тощо.

Мікропроцесор – це пристрій, що виконує дві основні функції:

1. Обчислення згідно з програмою, яка зберігається в ОП.
2. Забезпечує загальне керування апаратурою комп'ютера та обчислювальними процесами. При цьому МП виконує:
   - читання та дешифрацію команд з ОП;
   - читання даних з ОП та даних з регістрів зовнішніх пристроїв;
   - обробку даних та запис їх в ОП.

Для того, щоб МП знав, що робити, він неперервно повинен отримувати потік команд. Ці команди складають програми. Завдяки програмі обчислення в ЕОМ відбуваються автоматично. В програмі складний обчислювальний процес розбивається на множину елементарних команд, які може виконувати МП. Кількість команд сучасного МП – 220. У кожної команди є свій код.

Найбільш важливі вузли процесора – це кеш-пам'ять, шини процесора, регістри, пристрої управління, арифметичний пристрій.

Кеш-пам'ять – це високопродуктивна пам'ять процесора, вона використовується для прискореного обміну даними між процесором і оперативною пам'яттю. В кеші зберігаються найчастіше використовувані команди і дані.

Шина – це канал, який забезпечує обмін інформацією між пристроями. В процесорі існує система внутрішніх шин для обміну даними між блоками самого процесора.

Регістри – це внутрішня пам'ять процесора, тобто пристрої для тимчасового зберігання даних, вони створюються для кращої організації виконання арифметичних та логічних операцій.

Пристрій управління – це основний блок процесора, який управляє обчисленнями, координує роботу всіх пристроїв та забезпечує їх взаємодію.

Арифметичний пристрій виконує арифметичні та логічні операції над цілими і дійсними числами.

МП відрізняються трьома характеристиками: тактовою частотою, розрядністю і типом (моделлю).

Тактова частота – важливий технічний параметр, який визначає швидкодію процесора. Кількість команд, які процесор може виконати за 1 секунду, залежить від тактової частоти. Кожна команда, що виконується в ЕОМ, займає декілька тактів, тому час виконання команди вимірюється в тактах. Тривалість одного такту залежить від тактової частоти. Вимірюється тактова частота у мегагерцах (1 МГц відповідає 1 мільйону тактів за секунду). Чим більша тактова частота, тим менша тривалість такту і тим швидше працює ПК. Наприклад,

МП Intel 8086 працював на тактовій частоті 4,7 МГц, сучасні МП давно перейшли рубіж 3 Ггц (3000 МГц) і досягли 5 Ггц. Тактова частота генерується тактовим генератором.

Ще однією важливою характеристикою процесорів є їхня розрядність. Процесор оперує з двійковими числами, що подані як послідовність нулів та одиниць. *Розрядність МП* – це кількість розрядів двійкових чисел, які обробляються процесором за один такт в паралель. Мікропроцесори перших ПК були 8-розрядними, а всі сучасні моделі МП вже 32- та 64-розрядні.

Модель МП визначається особливістю його архітектури, моделлю фірми виробника та типом процесора.

В IBM-сумісних ПК найчастіше застосовуються МП фірми Intel, а також сумісні з ними моделі МП інших фірм – AMD, Cyrix, IBM тощо. Наведемо приклади моделей МП фірми Intel за порядком зростання їх продуктивності:

Intel 8086 (1976 р.), Intel 8088 (1979 р.) – перші 16-розрядні процесори і мали тактову частоту 5…10МГц;

Новим етапом у виробництві МП став процесор Pentium (1993 р.). Згодом з'явилися досить успішний процесор Pentium Pro (1995 р.) та Pentium II, в кінці лютого 1999 р. анонсовані перші МП Pentium III. Наприкінці 1999 р. з'явилося 9 моделей МП цього типу: Pentium 500E, 550, 533, 600, 700, 733 тощо. Потім з'явилися Intel Pentium 750, 800, 900, 1140. Цифри після назви означають тактову частоту в МГц.

В листопаді 2000 р. фірма Intel представила процесор PentiumIV. Архітектура його стала відрізнятися від архітектури попередників, завдяки чому змогли істотно наростити частоту процесора. Перші МП Pentium IV мали частоту 1,4 – 1,5 ГГц і містили 42 млн транзисторів на площі 217 мм^2 (в два рази більше, ніж Pentium III). В 2002 р. Intel анонсувала МП Pentium IV 3,06 ГГц. Такої високої тактової частоти вдалося добитися завдяки організації обчислень у кілька потоків. Тактові частоти останніх Pentium знаходяться в межах 4 ГГц. У процесорах Pentium використовується 64-розрядна шина даних та 32-розрядна шина адреси (2^{32} =4 294 967 296 комірок, приблизно 4 Гб ОП).

Відгалуженням від процесорів сім'ї Pentium стали процесори сім'ї Celeron – більш спрощений та здешевлений варіант процесорів Pentium та процесори Xeon, призначені для багатопроцесорних серверів (лінійка Xeon продовжується й нині).

Революційною подією на ринку МП став момент появи (середина 2006 р.) продуктів Intel Core 2 (восьме покоління мікропроцесорів). Core 2 – це ефективна система взаємодії кількох процесорних ядер, але для їх ефективної роботи необхідно, щоб програмні продукти були адаптовані для багатопроцесорних систем.

Кількість ядер суттєво впливає на продуктивність комп'ютера: два ядра – необхідний мінімум для роботи комп'ютера (одноядерні процесори уже не випускаються), 4,6,8 ядер – середній рівень, 10 ядер і більше для роботи з високопрофесійними програмами.

Нині виробник Intel пропонує широкий вибір моделей від бюджетних двоядерних до професійних 18-ядерних процесорів. Серед актуальних на сьогоднішній день можна виділити

- Pentium, Celeron, Core i3 (2 ядра) – бюджетні варіанти для офісних і домашніх ПК. Вони вже відходять у минуле, але мають низьку вартість;
- Core i5 – процесор середнього рівня, підходять для комфортної роботи з неспеціалізованим програмним забезпеченням і невимогливих сучасних ігор (2,4,6 ядер, об'єм кеша 6,8,12 Мб);
- Core i7 – потужні процесори (4,6,8 ядер, об'єм кеша 8,9,12 Мб). Останні випуски належать до 10-го покоління мікропроцесорів;
- Core i9 – новітні, найпродуктивніші процесори одинадцятого покоління для вимогливих програм, наприклад обробки відео (16,18 ядер, об'єм кеша 30 Мб, максимальна тактова частота перевищує 5 Ггц).

Паралельно з фірмою Intel фірма AMD в 1999 р. випустила МП Athlon (K7). Крім K7, на ринку з'явилася МП Athlon MP та Athlon XP (32-розрядні МП), що склали конкуренцію Pentium IV. В жовтні 2002 р. AMD випустила новий МП: Athlon XP, а в 2003 р. на ринок надійшли МП фірми AMD 8-го покоління під назвою Athlon 64 FX (Hammer).

Компанія AMD в 2005 р. анонсувала випуск двоядерних процесорів Athlon 64 X2 для настільних систем та лінійку серверних двоядерних процесорів Opteron. Нині фірма AMD випускає мікропроцесори Opteron різних моделей з 2, 4, 6, 8, 12, 16 ядрами.

Оперативна пам'ять

Оперативна пам'ять призначена для збереження, прийому і видачі даних та кодів програм. Внутрішня пам'ять ПК складається з оперативно запам'ятовуючого пристрою (ОЗП, RAM-пам'ять, оперативна пам'ять) та постійно запам'ятовуючого пристрою (ROM BIOS).

Постійна пам'ять – це енергонезалежна пам'ять, в яку інформація заноситься при її виготовленні. До постійної пам'яті „прошиті" деякі програми та дані, які комп'ютер не може змінити. Ця пам'ять призначена тільки для зчитування інформації. Як правило, в постійній пам'яті зберігаються програми обслуговування пристроїв комп'ютера та ініціалізації завантаження операційної системи.

Оперативна пам'ять *(ОП) – це спеціальні мікросхеми, що складаються з комірок пам'яті, до яких можна отримати швидкий прямий доступ, які призначені для тимчасового зберігання даних (код програми, проміжні обчислення, поточні параметри ОС, настройки драйверів тощо) і поточної зміни інформації при роботі ПК.*

ОП використовується для збереження даних і програмного коду, що виконується мікропроцесором. Будь-яка інформація записується до електронних комірок пам'яті у вигляді двійкових чисел. Розташування інформації в пам'яті називається записом, а отримання інформації з пам'яті – зчитуванням. Під час запису попередні дані, які зберігалися в комірках пам'яті, стираються. У фізичну комірку пам'яті записується 1 байт інформації. Ця ємність комірки достатня, щоб до неї записати один символ. Кожна комірка має свою адресу. Коли комп'ютер відправляє дані в ОП, він запам'ятовує адреси, потім за відомою адресою вибирає дані з пам'яті.

Найважливішими характеристиками ОП є тип пам'яті, обсяг модуля ОП, її розрядність, ємність і швидкодія. ПК з операційною системою Windows 95 працювали з 8 Мб ОП. Додатки з аудіографікою, комп'ютерні ігри вимагають великих обсягів ОЗУ.

Нині 8 ГБ ОЗУ – це оптимальне рішення для більшості сучасних ноутбуків для комфортної роботи з простими редакторами та іграми, 16 Гб для роботи з професійними програмами і 32 Гб для сучасних ігор.

Крім обсягу ОП, актуальний вибір *типу пам'яті*. За принципом роботи (принципом зберігання інформації) RAM можна розділити на *динамічну і статичну*. Різниця між динамічною і статичною пам'яттю полягає в конструктивних особливостях елементарних комірок для збереження окремих бітів. Нині для ОП використовується динамічна пам'ять DRAM. Вона побудована на мікросхемах, що потребують для збереження інформації її періодичного відновлення (регенерації), тобто на конденсаторах. Однобітова комірка пам'яті – це конденсатор, який може бути заряджений до високої або низької напруги (логічна 1 або 0).

За своєю логічною організацією DRAM може бути *асинхронною* й *синхронною*. Щоб забезпечити високу швидкість роботи пам'яті, нині використовується синхронна динамічна пам'ять DDR SDRAM. SDRAM означає, що пам'ять є синхронною динамічною, тобто при роботі з пам'яттю SDRAM забезпечується синхронізація всіх вхідних і вихідних сигналів із тактами системного генератора.

Абревіатура DDR (Double Data Rate) означає подвійну швидкість передачі даних (до 4 Гб/с і більше). В чотири рази більшу швидкість передачі даних має стандарт DDR2. В 2009 р. основну частку ринку

ОП завоював стандарт DDR3 (продовження стандарта DDR2). Перші модулі пам'яті DDR3 мали ємність 1 Гб, наступні – 2 і навіть 4 Гб. Нині модулі DDR3 уже застаріли, а використовуються модулі DDR4, DDR5. Об'єм одного модуля пам'яті DDR4 становить від 4 Гб до 128 Гб.

Залежно від форм-фактора розрізняють SIMM-модулі та DIMM-модулі пам'яті (з'явився в 1998 р.). У сучасних ПК використовується 184-контактні DIMM DDR-модулі пам'яті у вигляді окремих маленьких плат з напаяними на них мікросхемами. На відміну від модулів SIMM, дворядні модулі пам'яті – модулі DIMM – мають електрично незалежні контакти по обидва боки розняттів.

На материнську плату у відповідні гнізда можна вставити 1, 2, 3 і навіть 4 мікросхеми пам'яті типу DIMM. Різні структури ОП відрізняються швидкістю доступу до пам'яті та їх пропускною здатністю.

Статична пам'ять використовується як допоміжна пам'ять – кеш-пам'ять, яка призначена для оптимізації роботи процесора. Оперативна пам'ять працює більш повільніше, ніж процесор, тому він оснащується запам'ятовуючим пристроєм невеликого об'єму (кеш-пам'яттю) для проміжного зберігання даних.

3.1.3. Графічна підсистема комп'ютера

Графічна підсистема – це невід'ємна частина сучасного комп'ютера, оскільки вона займається візуалізацією всієї інформації, що виводиться комп'ютером. Використовується всіма прикладними програмами – від простіших текстових редакторів до програм моделювання 3D-графіки.

Графічна підсистема комп'ютера складається з апаратної і програмної частин. Апаратна частина включає в себе відеоадаптер (зовнішній або вбудований), інтерфейси між відеоадаптером і пристроями відображення. Графічний адаптер складається з різних блоків: відеопам'яті, графічного процесора, відеоBIOS, цифровоаналогового перетворювача, інтерфейсу зв'язку із системною платою та інших інтерфейсів, що часто входять у графічний процесор.

Програмна частина забезпечує підтримку інтерфейсів відеокарт, монітора і додатків на рівні BIOS, драйверів. У процесі роботи графічна система розв'язує різні задачі 2D-графіки, 3D-графіки та виведення й обробки відеографіки. Сучасні відеокарти здатні самостійно створювати і здійснювати обробку геометричних моделей як двовимірних, так і тривимірних сцен, тобто більша частина графічних функцій в них реалізуються апаратно. Графічна підсистема містить апаратні засоби, що пришвидшують обробку, компресію та декомпресію відео, підвищують якість зображення на екрані монітора.

Різні напрямки 3D-графіки стимулюють швидкий розвиток графічної підсистеми комп'ютера та спеціалізованих програмних засобів. Серед них – створення відеопродуктів, промислове проектування і дизайн, комп'ютерні ігри, навчальні системи та комп'ютерні симулятори. Поява ігрових програм стала одним з основних стимулів розвитку графічних адаптерів. Більша частина відеоадаптерів орієнтована на комп'ютерні програми, які найбільш повно використовують можливості графічних карт.

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 попіксельне прораховування 3D-сцени, а лише посилає відеоадаптеру просторове положення об'єктів, які необхідно відобразити. Ці об'єкти складаються з величезної кількості многокутників. Каркас багатокутників необхідно покрити поверхнею, щоб одержати реалістичне зображення на екрані комп'ютера. Цю роботу і бере на себе відеокарта.

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

Екран сучасного дисплея – це прямокутна матриця окремих дискретних точок (пікселів), які формують зображення. *Піксель* є мінімальним елементом зображення, який може бути згенерований комп'ютером. 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 фіксована кількість бітів – *атрибут пікселя* (код кольору). Ці біти й 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.

У дисплейних адаптерах із 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 кольорів, які при змішуванні визначають результуючі кольори пікселів.

Відеопам'ять є складовою частиною відеоадаптера. У відеопам'яті зберігається кадровий буфер та інші дані. Ємність графічної пам'яті та її тип визначає складність зображення, якість та швидкість його виведення. 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. Вони розміщені на відеокарті, тобто чіпи відеопам'яті припаюються до текстоліту плати.

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. Пікова швидкість передачі даних для сучасних відеокарт (наприклад, для типу пам'яті GDDR5x) досягає 480 Гбайт за секунду.

Зображення на моніторі повністю відповідає поточному вмісту відеопам'яті. Ємність відеопам'яті більшості сучасних відеокарт знаходиться в діапазоні від 256 Мб до 48 Гб. Така ємність потрібна для зберігання й оброблення великої кількості даних (буфер кадра, полігональні сітки, буфер глибини, параметри вершин, текстури тощо), що використовуються графічним процесором для побудови тривимірних рухомих зображень на екрані монітора, для швидкого виведення зображень з високою роздільною здатністю. Ємність графічної пам'яті визначає складність зображень, параметри якості відображення.

Програма, що виконується на комп'ютері в графічному режимі, має доступ (читання/запис) до всіх комірок відеопам'яті. Для збереження кількох кадрів зображення в певних режимах у відеопам'яті передбачені окремі ділянки 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 мож-

ливість 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 регістра 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, дані вершин, текстури ще й постійно запам'ятовуючий пристрій, в якому записані набори шрифтів для текстових та графічних режимів функції BIOS для роботи з відеоадаптером. У мікросхемі BIOS теж зберігаються програми, що здійснюють ініціалізацію відеокарти та інші необхідні компоненти.

Окрім цього, відеоадаптер містить складні керуючі пристрої, що забезпечують обмін даними та формування зображення.

Зростання вимог до якості зображень зумовило створення графічної багатопроцесорної системи, яка займається виключно розрахунками та формуванням зображень і називається *графічним процесором*. Графічні процесори включають комплекс пристроїв, які раніше виконувались у вигляді окремих мікросхем. 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 кількох, кожен з яких виконує свій набір графічних функцій. Наприклад, графічний процесор ATI Radeon X1800 XT містить 320 млн транзисторів і кілька десятків спеціалізованих процесорів.

Сучасні графічні процесори багатопроцесорні, оскільки містять набір спеціалізованих процесорів, зокрема групу *вершинних* та *піксельних* процесорів, які можуть опрацьовувати кілька вершин і пікселів паралельно. Під вершиною розуміють вершину полігона, точку в тривимірному просторі, яка задається однорідними координатами x, y, z, h. В графічних процесорах для ПК при описі тривимірних об'єктів використовують полігональні моделі, що складаються з множини полігонів, трикутників.

Вершинні процесори використовуються для виконання спеціальних програм обробки координат вершин і точок (вершинних шейдерів) при переміщенні тривимірних об'єктів, сортуванні вершин, усуненні невидимих граней тощо. Кожен вершинний процесор вико-

нує свої команди, так що різні процесори виконують різні частини програм над різними вершинами. Швидкість обробки полігонів залежить від кількості вершинних конвеєрів (наприклад, їх може бути вісім), тактової частоти графічного процесора, особливостей використання програмного забезпечення та інших факторів. Для сучасних адаптерів швидкість обробки полігонів досягає значення 400 млн трикутників за секунду і більше.

Піксельні процесори змішують значення розрахованого кольору з кольором буферу кадра (якщо установлений такий режим), або просто обчисляють і записують значення кольору та глибини в буфер кадру а також виконують додаткові операції. Швидкість зафарбовування пікселів залежить від кількості піксельних конвеєрів та тактової частоти графічного процесора.

Ще до складу графічних процесорів входять спеціалізовані блоки (***текстурні*** процесори), які виконують операції відображення текстур для підвищення реалістичності зображення на екрані комп'ютера. Якщо б не було текстур, об'єкти би мали каркасну форму. ***Текстура*** – це плоске або тривимірне зображення елементарної частинки реальної поверхні, яке накладають на тривимірні об'єкти з урахуванням їх форми, розміщення та рівня деталізації.

Накладання текстур – це базовий функціонал будь-якого графічного прискорювача. Показник продуктивності накладання текстур є одним із найважливіших, від нього залежить, наскільки швидко об'єкти виводяться на екран.

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*. Хоча існують програми, які не використовують прискорювачі.

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 і створення віртуальної реальності.

Професійні карти призначені для швидкого і якісного відображення тривимірних сцен з великою кількістю об'єктів складної форми, різних джерел освітлення та великою кількістю різноманітних структур. Сучасні графічні адаптери використовують останні досягнення тривимірної графіки.

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. Тепер 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. Усі відеоадаптери, що випускаються для ПК, можна розділити на три групи: *бюджетні*, *ігрові* та *професійні* відеокарти. В останній час у практику впроваджена технологія SLI паралельного використання двох відеоадаптерів, яка підтримує два режими роботи. В одному режимі кадр поділяється на дві частини і кожну частину обробляє окремий відеоадаптер, в іншому – кожний адаптер опрацьовує свій кадр. Компанією nVidia розроблений ще варіант технології SLI на базі чотирьох чіпів, тобто в один слот вставляється двочіпова карта. Очевидно, що продуктивність чотирьохпроцесорної конфігурації досить висока.

Крім варіанта, коли графічний адаптер виконаний у вигляді окремої карти, що вставляється в окремий слот, існують відеоадаптери, що вбудовані в чіпсет системної плати. Їх називають *приско-*

рювачами графіки. Традиційно вважається, що вони мають гірші показники, ніж окрема відеокарта. Однак, можливості сучасних вбудованих графічних прискорювачів неперервно зростають, тому якість та продуктивність у 2D-додатків для вбудованих прискорювачів і відеокарт практично однакові. Вбудоване графічне ядро добре справляється також із простішими 3D-об'єктами, але для сучасних тривимірних ігор та складних 3D-додатків вони поступаються можливостям відеокарт.

Зупинимся ще на питанні *програмного інтерфейсу*. Основою сучасних відеоадаптерів є графічні процесори, які мають різну архітектуру та різні системи команд. Для ефективного використання можливостей процесора необхідна наявність програмного інтерфейсу між його програмним забезпеченням і прикладними програмами. Однак, при широкій номенклатурі графічних процесорів неможливо написати програму, яка б ефективно працювала з будь-якою системою команд графічного процесора. Тому необхідно мати спеціальні програмні засоби, які будуть перетворювати запити прикладних програм у послідовність команд графічного процесора. Такими засобами є спеціалізовані прикладні програмні бібліотеки. Їх використання дозволяє програмістам розробляти програми, які незалежні від команд графічного процесора. Нині більшість прикладних програм використовують одну з двох типових бібліотек – Open GL або DirectX.

Підводячи підсумок, зазначимо, що якісний прискорювач – це прискорювач, який підтримує бібліотеки Open GL, Direct3D 5X, 6.0, забезпечує можливість опрацьовувати 1000000–3000000 трикутників за секунду (100 млн точок за секунду), здійснює апаратну трилінійну фільтрацію, має глибину кольору 32 біти (режим True Color), Z-буфер 32 біти і не менше 8 Мб пам'яті для карти із шиною PCI.

3.1.4. Пристрої введення графічної інформації

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

Дігітайзери. Назва *дігітайзер* походить від англ. digit (цифра) і означає пристрій, який здійснює аналогово-цифрове перетворення.

Дігітайзери (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 дігітайзерів робоча поверхня планшета має тактильну чутливість на базі

п’єзоелектричного ефекту. Завдяки цьому введення інформації відбувається без спеціальних пір’їн. П’єзоелектричні дігітайзери дозволяють креслити так само, як на креслярській дошці.

Для перенесення зображення з плівки в комп’ютер використовують спеціальні дігітайзери. За допомогою таких дігітайзерів у кіностудіях можна вводити в комп’ютер фотографії 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, pcx, 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 документа та перетворенні відбитого світла в цифрову форму. Існують планшетні, барабанні, ручні сканери, слайд-сканери (для сканування плівкових слайдів) та ін. Найуживаніші нині *планшетні сканери,* призначені для оцифрування окремих аркушів. У планшетних сканерах папір кладуть на спеціальну прозору поверхню. Лінійка фотоприймачів переміщується відносно паперу. Світловий потік, відбиваючись від поверхні непрозорого об’єкта, через систему дзеркал сприймається матрицею фоторецепторів, потім переводиться у форму електричних сигналів і перетворюється з аналогової форми у цифрову і передається в комп’ютер. Так здійснюється сканування документа та введення його в комп’ютер. Розрізняють чорно-білі (для введення тексту і рисунків, виконаних контуром), напівтонові, в яких кольори заміняються різним тоном сірого кольору, і кольорові сканери.

Сімейство *барабанних сканерів* надзвичайно різноманітне. Вони не схожі за своїми конструктивними характеристиками і використовують різні принципи дії. У спрощеній схемі об’єкт сканування

закріпляється на барабані (циліндрі), який обертається і відповідно переміщує оригінал. В середині барабана розміщене джерело світла. Направлений пучок світла проходить через оригінал і попадає на систему фотоелектронних помножувачів. Далі аналогово-цифровий перетворювач конвертує сигнал з аналогової форми у цифрову.

Ручні сканери з'явились чи не найпершими (наприкінці 80-х рр. минулого століття). В їх основі покладено процес реєстрації відбитих променів від поверхні документа. До їх переваг належать низька вартість, портативність, але вони володіють вузькою смугою сканування.

Якість сканера, об'єм графічної інформації, яку здатний опрацювати сканер, суттєво залежать від його ***роздільної здатності*** (максимальна кількість точок, яку розрізнює сканер). Роздільна здатність 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), або ppi (pixel per inch). Першу величину ще інколи називають оптичною роздільною здатністю, а другу – механічною. Як правило, механічна роздільна здатність більша за оптичну.

Планшетні сканери випускаються з різною роздільною здатністю. Простіші моделі сканерів мають дозвіл 300×600 dpi, сканери середнього класу – 600×1800 dpi. У сучасних звичайних моделях сканерів ці характеристики досягають 1200 × 2400 dpi, а в професійних – 2400 × 4800 dpi і більше (це основна характеристика).

Часто використовується інша характеристика – лінійна роздільна здатність. Під *лінійною роздільною здатністю* розуміють максимальну кількість точок, які можна розмістити на горизонтальному або 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* – це важлива технічна характеристика будь-якого оцифровуючого пристрою. 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 при трьох кольорових каналах RGB відповідають 16,7 млн кольорів. Зрозуміло, що в побуті в такій кількості кольорів немає потреби, хоча кращі марки напівпрофесійних сканерів мають 48-бітну глибину внутрішнього кольору, тобто по 16 бітів на один канал. А це дозволяє утворити 2^{48}=281474976710656 тонових градацій. Проте людське око не спроможне відрізняти відтінки навіть простору True Color і не існує друкуючих принтерів, які можуть передати таку велику кількість градацій. Причин збільшення глибини кольору дві. Перша – технологічна, яка полягає в тому, що матриця фоторецепторів з більш високою розрядністю володіє підвищеною чутливістю. Друга причина – програмна. Оскільки при деяких операціях обробки зображень зменшується кількість градацій, то 16-бітні канали мають деякий запас і втрата інформації не призведе до зменшення якості зображень. Зовсім іншу ситуацію маємо при роботі з 8-бітними каналами.

Для зв'язку сканера з комп'ютером використовують спеціальні 8- або 16-розрядні плати, що під'єднані до шини ISA. 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 – це пристрої, в яких світлочутливим елементом є матриця, що складається з світлочутливих елементів, сигнал з яких оцифровується і зберігається у самому апараті. 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). Нині 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 проєктується не на фотоплівку, а на світлочутливу матрицю. Далі значення електричних зарядів оцифровується і формується матриця пікселів. Якість зображення залежить від роздільної здатності. Роздільна здатність вимірюється двома цілими числами, де перша цифра – це кількість пікселів у рядку (ширина), друга – це кількість пікселів у стовпці (висота). Інколи кількість пікселів у зображенні визначають

одним числом, що виражає загальну кількість пікселів у зображенні. Вона виражається в мегапікселях (1 Мріх = 1 мільйону пікселів). Наприклад, зображення 3020 × 2016 = 6088320 пікселів, або 6,1 Мріх.

У професійних камерах роздільна здатність може досягати 3264 × 2448 пікселів і значно більше, тобто це десятки Мріх. Потім цифрове зображення записується в FLASH-пам'ять і зберігається в форматі JPEG, TIFF. Цифрові фотокамери обладнані LCD-екранами, які дають можливість одразу переглянути відзняті кадри і можуть під'єднуватись до ПК або до фотопринтера. Файл із цифровим зображенням на комп'ютері можна редагувати, ретушувати, споряджувати спецефектами, кадрувати, передавати каналами зв'язку тощо.

3.1.5. Пристрої виведення графічної інформації

Існує багато різноманітних пристроїв для виведення графічних зображень. Основним пристроєм оперативного виведення графічної інформаціїі в сучасних ЕОМ є різні монітори. Найбільш типові – це монітори на електронно-променевих трубках, дисплеї на рідких кристалах та плазмові монітори. Для формування твердих копій використовують графобудівники, матричні, струминні та лазерні принтери тощо.

Монітори. Растровий кольоровий *монітор* нині є основним і найбільш розповсюдженим пристроєм виведення інформації. Растрові монітори, незалежно від способу їх виконання, для відтворення кольорів використовують модель RGB, тобто три значення. Ці дані зберігаються у відеопам'яті і передаються за допомогою цифро-аналогового перетворювача на монітор.

Екран дисплейного монітора являє собою набір дискретних точок, що утворюють регулярну прямокутну решітку – растр (від лат. rastrum – граблі). Точки растру формують зображення. Їх називають ***пікселами***. В сучасних графічних системах для кодування кольору на кожний піксель відводиться 24 (або більше) біти (вісім для червоної складової, вісім – для зеленої і вісім для синьої). При цьому три незалежних кольорових канали можуть відтворити 2^{24} = 16 777 215 кольорових градацій.

Якість зображення та кількість кольорових відтінків на екрані монітора визначається як можливостями самого монітора, так і можливостями відеокарти (відеоадаптера). Монохромні монітори дозволяли відображати тільки два кольори. Сучасні графічні системи здатні відобразити 16,7 і більше мільйонів кольорів.

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

Існує кілька 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 екрана не меншої 17 дюймів. Інакше робоче середовище більшості графічних редакторів не поміщатиметься на екрані. Для професійної роботи з графікою використовують монітори з діагоналями 21 або 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 люмінофору (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. У професійних моніторах віддаль між двома точками люмінофора становить 0,22 …. 0,24 мм. Для сучасних CRT-моніторів величина зерна часто вимірюється віддаллю між сусідніми точками люмінофора. Для професійних моніторів віддаль між двома точками люмінофора становить 0,22 …. 0,24 мм.

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. Для знаходження загальної кількості пікселів на екрані ці два числа потрібно перемножити.

Наведемо приклади деяких стандартних режимів:

800 × 600 для 15-дюймових моніторів;

1024 × 768 для 17-дюймових моніторів;

1280 × 1024 для 19-дюймових моніторів.

На практиці будь-який із цих моніторів може підтримувати й вищі роздільні здатності.

Іншою характеристикою моніторів є *частота вертикальної розгортки*, тобто частота оновлення кадрів, яку ще часто називають *частотою 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. 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 – CRT-монітори) і світлопропускні (монітори на рідких кристалах – LCD-монітори). Монітори на базі електронно-променевих трубок (ЕПТ) досі залишаються неперевершеними в основних вимогах, що пояснюється високою якістю зображення, якісною передачею кольору, відсутністю інерційності зображення, більш високою швидкістю зміни зображення, більшим кутом огляду, дешевою технологією виготовлення тощо. Водночас монітори на базі ЕПТ випромінюють широкий спектр електромагнітних променів, що може зашкодити здоров'ю людини.

CRT-монітори отримують зображення від пучка електронів, що випускається електронною гарматою й управляється електромагнітним полем. В результаті попадання електронного пучка на люмінофор на ньому утворюється світла точка – піксель. Яскравість 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 електрон-

ний промінь сфокусувати до заданих розмірів у площині екрана, 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). Впровадження нових технологій дозволило випускати монітори з плоским екраном та укороченими трубками. До переваг CRT-моніторів слід віднести і той факт, що програмно можна установити різну роздільну здатність (навіть більшу за стандартну), чого не можна зробити на інших моніторах.

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. Так, для монітора з діагоналлю 15 дюймів роздільна здатність дорівнює 1024×768, а для 17 дюймів – 1280×1024.

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

До недоліків LCD-моніторів відносять одну оптимальну роздільну здатність, яка забезпечує добру якість, обмеженість діапазонів кутів комфортного зору (вертикального та горизонтального), тобто варто трохи повернути дисплей, як помітно зміниться яскравість і кольори. Випускають такі монітори компанії LG, BenQ. В останній час LCD-монітори витісняють CRT-монітори для настільних комп’ютерів.

Зовсім недавно з’явилися *плазмові дисплеї* (PDP). Плазмові монітори складаються з порожньої скляної панелі, що заповнена газом. На поверхню внутрішньої сторони стінок виведені мікроскопічні електроди, що утворюють дві симетричні матриці, а зовні стінки покриті шаром люмінофора. Коли на контакти подається струм, між ними виникає розряд, який змушує світитися розміщені поряд молекули 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. Панелі, виготовлені за цією технологією, характеризуються високою яскравістю, контрастністю, широким кутом зору, малою товщиною (менше 10 см). Плазмові панелі мало годяться для моніторів, але є ідеальним засобом відображення для колективного користування в демонстраційних залах. Це пов'язано з технічними особливостями плазмових панелей – мінімальна довжина діагоналі 32 дюйми, тобто вони не мають конкурентів в сегменті великих діагоналей.

Останнім часом з'явилися й нові технології розробки засобів відображення графічної інформації. Це органічний монітор OLED, у якому електролюмінесценція відбувається в тонкому пласті органічного напівпровідника, розташованого між двома тонкими плівковими провідниками. OLED – це тонкоплівковий пристрій зі світло випромінюючою поверхнею, яка утворена множиною комірок, випромінюючих світло.

Розроблено також 3D-монітор, який передає тривимірне зображення. Цей монітор в двох площинах створює два зображення, що мало відрізняються між собою, і розміщує їх таким чином, що дозволяє добитись ефекту тривимірності.

Принтери, перові графобудівники, плотери.

Принтер – це пристрій для одержання твердих копій цифрових зображень, для друку на папері тексту, графіків, зображень, креслень. Нині найрозповсюджені *лазерні, струминні, матричні* принтери.

В ***матричних*** *принтерах* зображення формується друкуючою головкою, оснащеною набором голок. Головка пересувається порядково, при цьому голки вдаряють по паперу через фарбну стрічку фарба з якої відбивається на папері. В різний час випускалися принтери з різною кількістю голок, але найбільш широкого розповсюдження набули принтери з 9 та 24 ударними голками.

Майже всі матричні принтери монохромні. Перевагою матричних принтерів є міцність і надійність принтера, можливість друку на папері через копірку (до 6 копій). Є дешеві фарба і стрічка. Кольорове зображення на матричних принтерах одержується за допомогою багатоколірних стрічок. Використовується чотири колірна стрічка, на яку нанесено три основних кольори: блакитний, пурпурний, жовтий та чорний. Роздільна здатність становить 180-300 dpi. Найбільш сучасні 48-голчаті матричні принтери мають роздільну здатність більше 300 dpi.

Матричний принтер – найстаріший варіант друкувального пристрою (створений в 1964 р.), але не дивлячись на те що з'явилися нові типи принтерів, він понині використовується. В деяких випадках він просто незамінимий, зокрема, там, де потрібний друк на таких матеріалах, на яких принтери інших типів друкувати не можуть.

Лазерні *принтери*. Процес лазерного друкування розроблений фірмою Xerox. На спеціальному фоточутливому барабані (головна частина лазерного принтера) променем світла створюються ділянки заряджені електронами (картинка малюється променем по барабану, створюється фотоелектричне зображення). Поверхня барабана, оброблена лазером, проходить повз картридж та зарядженими областями притягує порошок-тонер, який складається з частинок фарбувального пігменту, покритих пластмасою. Потім барабан обертається над аркушем паперу, який заряджений сильніше за барабан, при цьому частинки тонера переносяться з барабана на папір і розігріваються, утворюючи водотривке зображення. В лазерних принтерах використовується папір у вигляді аркушів.

Однією з основних характеристик принтера є його роздільна здатність. Вона вимірюється кількостю точок на один дюйм (dpi). Зрозуміло, чим вища роздільна здатність принтера, тим більше реальних точок може бути в одній віртуальній точці, а це означає більш високу якість друкованого зображення. Роздільна здатність сучасних монохромних моделей коливається від 600 до 1200 dpi. Різниця між надрукованим текстом 600 dpi і 1200 dpi несуттєва, але стає помітною на графічних зображеннях. Для кольорових моделей роздільна здатність становить 1200 dpi.

Струминні *принтери* виводять текст і кольорову графіку за іншою технологією та коштують значно дешевше, ніж лазерні. Крім цього, вони компактніші, використовують менше енергії, але мають меншу швидкість друкування та вищу вартість витратних матеріалів (чорнила, картридж). Подібно до лазерного друку, струминний друк безударний. Друкувальна головка має групу мікросопел, кожне з яких в діаметрі менше за діаметр людської волосини.

Принцип струминної технології базується на вистрілюванні на носій зображення мікрокапель чорнила зі спеціального сопла (їх називають дюзами). Різні кольори одержуються за рахунок нанесення на папір різної кількості чорнил різного кольору. Цей розрахунок виконується драйвером. Кольорові пристрої струминного друкування мають, як правило, чотири форсунки: три – для основних кольорів (блакитного, пурпурного, жовтого) і одну – для чорного. Ця модель кольору називається CMYK. Вона має менший кольоровий охват, тому надруковане зображення може виглядати не так, як на екрані.

Основна перевага струминної технології полягає в можливості змішувати кольори, оскільки різні частини фарби наносяться за один прохід і встигають перемішуватися до висихання фарби. Це дозволяє отримувати такі глибину і різкість кольору, яких не можна досягти при іншій технології. Важливу роль для якості друкування відіграє якість паперу та чорнил.

Надрукувати якісні кольорові фотографії на звичайному принтері не можна, тому в кінці 1990 р. в сегменті струминних принтерів сформувався підклас фотопринтерів. ***Фотопринтери*** відрізняються від звичайних підвищеною роздільною здатністю та іншими параметрами. На відміну від струменевих принтерів, у фотопринтерах стали використовувати шестиколірну модель кольору CcMmYK, де c і m означають додаткові світло-блакитну та світло-пурпурну фарби, що робить фотографії яскравішими і насиченішими. Упродовж багатьох років шестифарбна схема залишалася стандартом для струминних принтерів непрофесійного класу.

В кінці 2003 р. компанія Hewlett-Packard представила нову еволюцію технології фотореалістичного друкування – HP PhotoRet Pro, в якій уже використовувалися не шість, а вісім кольорів: до класичного набору були додані *два відтінки сірого* (сірий і світло-сірий). Це дало змогу збільшити кількість відтворюваних кольорів до 79,2 млн відтінків. В 2005 р. компанія HP випустила фотопринтер HP Photosmart 8753, який уже використовував дев'яти колірну схему (додали ще синій колір).

В лінійці професійних фотопринтерів використовують десять і дванадцять кольорів. Вони є широкоформатними, мають друкуючі головки високої чіткості з технологією друку краплями різного розміру (містять 600 сопел для чорнила), вбудовані спектрофотометри для забезпечення точності передачі кольорів і використовуються в дизайнстудіях. Паралельно компанії Epson і Canon випускають свої моделі фотопринтерів, в яких реалізовані інші технології фотодруку.

Оскільки основна функція фотопринтера – друк чітких зображень на спеціальному папері, то при виборі принтера експерти радять враховувати такі характеристики:

- формат паперу (професійні можуть друкувати на папері шириною до 60 дюймів у ширину);
- кількість кольорів. Мінімальна допустима кількість кольорів – шість, але чим більше тим краще;
- роздільна здатність кольорового зображення (приклади $2400\times 1200, 4800 \times 1200, 5760 \times 1440$ dpi);
- швидкість друку, що важливо для професійних пристроїв (кращі принтери друкують до 15 сторінок формату A4 за хвилину);

• мінімальний обсяг краплі (перевагу надавати моделям у яких мінімальний обсяг краплі дорівнює 1,5).

Зазначимо, що нині існують моделі фотопринтерів, які можуть друкувати зображення зі змінних носіїв і без комп'ютера. Вони володіють багатьма функціональними можливостями, жорстким диском ємністю до 500Гб і навіть невеликим кольоровим дисплеєм.

Графобудівники *(плотери)* – це пристрої для автоматичного виведення зображень креслень, графіків, схем, карт та іншої графічної інформації. Плотер – один із головних пристроїв виведення інформації в системах автоматизованого проєктування, невід'ємна частина автоматизованого робочого місця конструктора.

Плотери – це пристрої траєкторного типу, в яких зображення на папір наноситься рухомим вузлом за допомогою пера з високою якістю зображень.

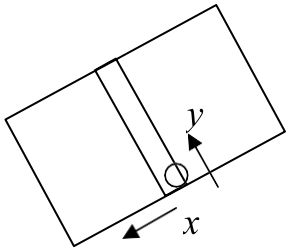


Рис. 3.1. Схема графобудівника

За способом переміщення носія розрізняють *планшетні*, *барабанні* (рулонні) та *фрикційні* графобудівники. У планшетних носій зображення нерухомо закріплюється на площині графобудівника. На планшетних графобудівниках отримують досить високоякісні високоточні зображення та гарну передачу кольорів.

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 часток 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 різаки тощо.

Таке різноманіття типів носіїв та писальних вузлів забезпечує багато сфер застосування планшетних графобудівників. Наприклад, різні модифікації графобудівників використовують для розкрою одягу, матеріалу. Якщо замість писального вузла використовується ніж, то такі ріжучі плотери називаються *катерами* (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 4*g*).

Графобудівники з рухомим носієм можна класифікувати як *барабанні*, *фрикційні* та *рулонні*. 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 плоттерами нижча порівняно з планшетними, але відповідає вимогам більшості застосувань. Ці плотери компактніші та можуть відрізати автоматично від рулону паперу аркуш потрібного розміру.

У фрикційних графобудівниках папір переміщується за допомогою фрикційних роликів, вони мають менші розміри, ніж барабанні. Рулонні графобудівники подібні фрикційним, але в них використовується носій з крайовою перфорацією.

Крім перових плоттерів, виділяють плотери, які використовують растровий спосіб створення зображення. Наприклад, струминні 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 CMYK.

Проєктори. Для демонстрації комп'ютерних зображень на великих екранах, наприклад під час проведення лекцій чи презентацій, використовують *відеопроєктори*. Проектор під'єднується до комп'ютера або ноутбука паралельно з монітором або замість нього і сигнал на нього подається безпосередньо з відеокарти комп'ютера. На якість зображення суттєво впливають технології формування зображень, якість оптики. Важливою частиною проєктора є лампа, що створює світловий потік. При виборі проєктора потрібно враховувати термін служби лампи через її високу вартість (її ресурс становить менше 3 тис. годин). Основними характеристиками проєкторів є світловий потік, контрастність, роздільна здатність, діапазон розмірів екрана, вхідні та вихідні інтерфейси тощо.

Яскравість і графічна роздільна здатність – найважливіші властивості проєкторів для організації презентацій. Яскравість визначається кількістю світла, що випромінюється проєктором. Світловий потік не залежить від віддалі об'єктива проектора до екрана. Роздільна здатність проєктора може бути різною. Для відтворення відео можна використовувати проєктори з роздільною здатністю 800×600 точок. Для якісного відтворення комп'ютерних зображень з дрібними деталями використовують проєктор з роздільною здатністю 1024×768 точок, при підвищених вимогах до контрастності зображень – 1400×1050 точок.

При нормальному розміщенні проєктора нижній край розміщення зображення на екрані знаходиться на рівні об'єктива проєктора, в інших випадках у проєктора передбачена корекція вертикальних трапецеїдальних спотворень.

Серед технологій формування зображень на проєкційному екрані можна виділити чотири основні, які відрізняються елементами, що формують зображення: CRT, LCD, D-ILA, DLP.

Проєктори на базі електронно-променевих трубок (CRT) мають три електронно-променевих трубки, кожна з яких генерує свій колір (червоний, синій, зелений), разом вони формують кольорове зображення відповідно до моделі RGB. Ці проєктори володіють високою якістю відтворення зображень (роздільна здатність, чіткість, точність передачі кольорів), забезпечують глибокий рівень чорного і широкий діапазон передачі кольорів, але мають значні габарити, масу і вимагають постійних точних налаштувань.

У LCD-мультимедійних проєкторах зображення формує LCD-матриця, тобто при проходженні світлового потоку від лампи через рідинно-кристалічну панель, що складається з матриці пікселів. LCD-технологія дозволила суттєво здешевити проєктори, зменшити їх габарити й одночасно збільшити світловий потік. Ці проєктори прості в експлуатації, і мають широке застосування для проведення презентацій.

Технологія D-ILA з'явилась порівняно недавно і вважається одним із найперспективніших напрямків розробки проєкційного устаткування. Подібно LCD-технології, вона базується на технології рідинних кристалів, однак працює не на просвічуванні, а на відображенні. Головна відмінність технології D-ILA від LCD полягає в тому, що всі управляючі пікселями електроди розміщені за шаром рідинних кристалів, а не між комірками. Це сприяє формуванню зображень на більшій площі матриці. В результаті межа між пікселами практично непомітна, світловіддача матриці зростає, що сприяє створенню повноколірного зображення. Ця технологія дозволяє добиватися високої роздільної здатності, більшість D-ILA-проекторів базується на матрицях із роздільною здатністю 1365×1024 і матрицях з підвищеною роздільною здатністю 2048×1536.

В основі DLP-проєктора лежить нова технологія формування зображення, що базується на використанні кремнієвої поверхні (DMD-кристала) на якій розміщені сотні тисяч (до мільйона) керованих дзеркал. DMD-кристал виконує також функції формувача зображень в DLP-проєкторах. Вони характеризуються роздільною здатністю, що визначається кількістю дзеркал DMD-матриці. DLP-технологія дозволяє створювати як потужні проєкційні апарати з великим світловим потоком, так і мініатюрні проєкти.

3.1.6. Масштабування та дискредитація

Якісна картинка вимагає щільної упаковки точок зображення (пікселів). Для кожної з елементарних частинок зображення потрібно зберігати інформацію про колір і яскравість. Розміри картинок визначають обсяг оперативної пам'яті, її можна контролювати вже на стадії первинної оцифровки зображень. Збільшення роздільної здатності в два рази збільшує об'єм пам'яті в 4 рази. Навіть невелика різниця між 150 і 200 dpi може вимагати оперативної пам'яті, що відрізняється на мегабайти. Будь-яка програма управління сканером видає дані про обсяг оцифрованої картинки. Результат залежить від розмірів оригінала, роздільної здатності сканування, вибраної моделі кольорів.

Відскановане зображення виводиться на екран комп'ютера і опрацьовується в растровому редакторі з метою отримання друкованої версії потрібної якості, при цьому виконується ланцюжок перетворень над оригіналом.

Екранна версія зображення – це матриця точок, яка буде формувати зображення, займаючи частину екранного простору на будь якому моніторі. Наприклад, зображення 600×400 точок займе майже весь екран, якщо монітор має роздільну здатність 640×480 і ¼ екрана на моніторі з дозволом 1024×768. При цьому розміри зображення в дюймах, сантиметрах залежатимуть від діагоналі монітора. Цікаво, а які будуть розміри зображення, що виводиться на принтер? В більшості растрових редакторів за замовчуванням фізичні розміри друкованої версії збігаються з розмірами ділянки сканування.

Припустимо, що екранна версія зображення уже сформувалась на етапі сканування. Виникає питання, як буде впливати зміна області друку на якість друкованої версії. Наприклад, якщо потрібно роздрукувати екранне зображення розміром 800×800 пікселів і розміри квадратної картинки мають мати розміри 10 дюймів, то роздільна здатність дорівнюватиме 800 dot/10 inch=80 dpi.

Тобто треба мати на увазі, що зміна області друку (її розмірів) не змінює якість екранної версії зображення (якість уже сформувалася на етапі сканування), а на якість друкованої версії впливає, причому

суттєво. Для кожного друкуючого пристрою існує деяке оптимальне значення роздільної здатності цифрового зображення, при якому друкуючий пристрій зможе передати максимальну кількість деталей оригінала. Зауважимо, що якість друку залежить і від паперу, особливо для кольорових струминних принтерів. Так, якщо для вибраного принтера і паперу оптимальне значення 200 dpi, то при виведенні на друк оригінала з роздільною здатністю 120 dpi відбудеться втрата якості зображення, оскільки деякі деталі щезнуть при друці. А якщо на принтер передати картинку з роздільною здатністю 300 dpi, то принтеру буде передана надлишкова інформація, якою він скористатися не зможе. Тобто якщо сканована версія зображення на моніторі демонструє невисоку якість, то підвищення роздільної здатності для друку не зможе поправити ситуацію, оскільки така дія не додає графічної інформації. Принтер використовує тільки ті дані, які закладені в зображення на етапі його оцифрування.

Отже модифікація області друку при фіксованих точкових розмірах зображення приводить до зміни роздільної здатності, а тому і якості друку. Наприклад, якщо розмір картинки збільшити вдвоє, то роздільна здатність зменшиться вдвоє. В растровій графіці це перетворення називається ***масштабуванням***. Зокрема, масштабуванням займаються у випадку, коли зображення з цифрової камери, яке має невеликі розміри, хочуть роздрукувати на принтері з більшими розмірами картинки. Масштабування не змінює параметри екранного зображення (кількість точок, глибину кольору), а результати масштабування можна побачити лише при друкуванні.

Розглянемо тепер поняття дискретизації. Зміна кількості точок растрового зображення називається ***дискретизацією***. Дискретизація, вочевидь, впливає на розміри екранної версії зображення, оскільки змінює габарити растрового поля. Збільшення роздільної здатності в два рази викликає зростання кількості точок і збільшення лінійних розмірів екранного зображення і навпаки. Наприклад, якщо для зображення 600×600 зменшити його екранні розміри до 400×400 (на перший погляд незначно змінили), то кількість точок зображення зменшиться з 360000 до 160000, тобто майже в два рази. Зрозуміло, що ця операція викличе зміну якості картинки.

При зменшенні розмірів програма просто відкидатиме зайві пікселі, а при збільшенні кількості точок додаткові точки потрібно згенерувати. Додавання нових точок відбувається за спеціальними алгоритмами інтерполяції.

У растровій графіці найбільш розповсюджені три основних методи дискретизації: метод ближнього сусіда, методи білінійної та бікубічної інтерполяції.

В *методі ближнього сусіда* для нового пікселя беруть характеристики його ближнього сусіда. Цей метод хоча забезпечує високу швидкість роботи, але результати роботи мають не надто високу якість.

Метод білінійної інтерполяції дає кращі результати, нові точки розраховуються шляхом усереднення кольорів сусідніх точок.

Метод бікубічної інтерполяції – це кращий метод інтерполяції, він використовується в професійних растрових редакторах (наприклад, у Photoshop). Новий колір теж розраховується на основі сусідніх точок, але за більш складними алгоритмами.

При будь-якій дискретизації повинні змінюватись або розміри зображення, або роздільна здатність, оскільки ці параметри пов'язані таким співвідношенням:

кількість точок = розмір × роздільну здатність.

Операцію дискретизації можуть виконувати не тільки графічні програми, але й пристрої оцифрування. Для прикладу, якщо потрібно оцифрувати оригінал шириною в 3 дюйми з максимальною оптичною роздільною здатністю сканера в 600 dpi, то зрозуміло, що в процедурі сканування буде задіяно 600×3=1800 світлочутливих елементів із лінійки фотоприймачів.

Але якщо установити роздільну здатність 300 dpi (половина від вихідного), то в процесі оцифровки буде задіяно 900 датчиків, тобто кожний другий. У цьому випадку алгоритм управління сканером не потребує глибоких змін.

Інша справа, якщо вибрати таку щільність оцифровки, яка не є цілою частиною максимальної оптичної роздільної здатності. Тоді ми уже не будемо мати регулярність вибору активних датчиків, тому результати сканування можуть бути сформовані лише з допомогою спеціальних алгоритмів, що працюють за принципом білінійної інтерполяції. Отже, варто вибирати щільність оцифровки, яка задовільнить користувача і одночасно є цілою частиною максимальної оптичної роздільної здатності.

Так якщо сканер має максимальну оптичну роздільну здатність 300 dpi, то значення 75, 100, 150 dpi мають перевагу перед іншими значеннями (наприклад, перед 175 dpi). Якщо необхідно отримати скануюче зображення з роздільною здатністю 140 dpi краще встановити 150 dpi.

3.2. Програмне забезпечення

3.2.1. 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 ставляться 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:

- стандартні графічні пакети (графічні редактори, системи автоматизованого проєктування);
- програмні системи, що не спеціалізуються на графіці, але підтримують певний набір графічних функцій (наприклад, офісні пакети, системи комп'ютерної математики);
- 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 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.

3.2.2. Найвідоміші графічні редактори

Графічні редактори – це прикладні програми, призначені для створення й обробки графічних зображень на комп'ютері в діалоговому режимі і зберігання їх у файлах графічних форматів. 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.

Більш довершений графічний редактор ***Paint.NET***, який дозволяє виконувати досить складні графічні операції. ***Paint.NET*** – це користувацький безкоштовний і ефективний растровий графічний редактор, що підтримує роботу з шарами, прозорістю, змішуванням та розширенням його можливостей за допомогою плагінів. Водночас час простий і доступний в користуванні, має зрозумілий і зручний інтерфейс користувача. Використання шарів дозволяє створити одне зображення шляхом поєднання кількох різних зображень або їх частин. Підтримує роботу з графічними файлами багатьох популярних форматів (BMP, JPEG, PNG, GIF, TGA та ін.), власний формат файлів PDN.

Інший клас растрових редакторів спрямований на професійну обробку готових растрових рисунків із метою поліпшення їхньої якості. До таких програм належить, зокрема, найдовершеніший і популярний комерційний растровий редактор ***Adobe Photoshop***. Можливості програм практично невичерпні, пакет програм має різноманітні професійні інструменти для обробки зображень, можливості створення багатошарових зображень, засоби для збільшення та зменшення різкості зображень. Програма забезпечує найточніше регулювання кольорів, роботу з текстурами, вибір різних кольорових моделей і файлів різних форматів. Цьому сприяє величезний набір різноманітних фільтрів та спецефектів. Пакет володіє засобами відновлення пошкоджених зображень, ретушування фотографій тощо. Компанія Adobe поновлює графічний редактор Photoshop приблизно

раз у два роки. Наприклад, версія Photoshop CS3 Extended має усі 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.

В новітній версії Photoshop 23.0 (2021 р.)

– здійснено оновлення інструментів, зокрема інструмента для виділення об’єктів, що дозволяє більш швидко створювати композиції;
– розроблені нові довершені фільтри;
– поліпшено управління кольором;
– відбулась модифікація 3D-функцій;
– поліпшена взаємодія з додатком Illustrator;
– розширена підтримка нових моделей камер 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***. Крім того, ця програма містить багато інструментів для обробки зображень.

XnView – це теж популярна програма для переглядання файлів зображення. Вона підтримує приблизно 400 різних графічних і мультимедійних форматів файлів, а також дозволяє здійснювати конвертування графічних файлів. Ця програма може виконувати і базові операції редагування графічних файлів (вирізати частину зображення, здійснювати поворот та змінювати розмір зображення, змінювати кольори та обчислювати кількість кольорів у зображенні, вставляти текстові надписи, одержувати зображення зі сканера та ін.)

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 (комерційні) 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 комерційних векторних графічних редакторів. Цей багатофункціональний редактор призначений для обробки і створення, редагування векторної та растрової графіки, для підготовки файлів для графобудівників. Він є головною програмою для дизайнерів, зайнятих у сфері поліграфії та реклами. Corel Draw для конструювання зображень надає користувачу широкий набір інструментів та операцій. Програма підтримує величезну кількість заливок, текстур, рекламні тексти, які можна масштабувати, повертати, робити об'ємними, накривати тінню і текстурами. Вона дозволяє працювати з величезною кількістю об'єктів (кілька сотень і навіть тисячі) за рахунок об'єднання об'єктів у групи і далі з групою об'єктів працювати як з одним.

До складу пакета Corel Draw входить редактор растрової графіки (Corel Photo Paint), програма для перетворення растрової графіки у векторну (Corel Trace), програма для створення векторної анімації та ін.

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, спецефектів 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 інтернету і комп'ютерні ігри.

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

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

Для автоматизації проєктно-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 проєктуванні 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 проєктування і є світовим лідером тривимірної векторної графіки.

Досить розповсюдженим пакетом САПР є комплекс програм ***КОМПАС***. КОМПАС являє собою САПР загального призначення. Ця програма зорієнтована насамперед на підприємства машинобудівної галузі і їй надають перевагу більшість промислових підприємств.

Не дивлячись на можливості графічних пакетів, математичні моделі об'єктів, їх алгоритми побудови заховані від користувача.

3.2.3. Системи комп'ютерної математики

Широкі можливості для роботи з графікою, візуалізації даних мають системи комп'ютерної математики (СКМ) MathCad, MathLab, Mathematika, Maple та ін. СКМ можуть виконувати різноманітні математичні операції практично з усіх розділів сучасної математики, підтримують інтерактивну візуалізацію, засоби мультимедіа і дозволяють комбінувати алгоритми, формули, результати обчислень, текст, графіку, діаграми та анімацію зі звуком в одному файлі. Програми мають розвинутий графічний інтерфейс і забезпечують розв'язування багатьох задач в графічній формі. Наприклад, для кусково-поліноміальної апроксимації табличних функцій $y(x)$ у MathLab призначена функція interp1.

3.2.4. 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++, Python 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, 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 проєктно-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.

3.2.5. 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 трудоємкістю програмування та переносимістю програм.

В MS DOS програміст має вільний доступ до апаратних засобів, зокрема до відеопам'яті за допомогою низькорівневих мов Pascal, Assembler.

Наведемо приклад програми прямого звернення до комірок відеопам'яті в тринадцятому графічному режимі. Цей режим забезпечує 256 кольорів для пікселів та роздільну здатність 320 × 200. Атрибут кожного пікселя визначається одним 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 }
```

2. Наступним, більш високим рівнем програмування графічних задач є використання можливостей відео-BIOS для виконання простіших графічних операцій (наприклад, визначення/встановлення графічного режиму, виведення пікселя на екран тощо). При цьому трудоємкість програмування зменшується, ступінь переносимості програм збільшується і програмісту вже не потрібно оперувати регістрами, бітами, фізичними адресами. Всі функції відео-BIOS обслуговує програмне переривання 10H. Функції BIOS викликаються як програмне переривання за допомогою функцій:
int 10H //мовою Assembler,
intr($10, R) // мовою Pascal, genintrrupt(0×10) // мовою C++ .

3. Використання спеціалізованих графічних інтерфейсів та бібліотек, таких як OpenGL, DirectX, які підтримуються апаратними можливостями сучасних графічних процесорів.

OpenGL – найпопулярніший графічний стандарт (з'явився в 1992 р.), відкрита графічна бібліотека, яка широко використовується в різних розділах комп'ютерної графіки, наприклад для розробки CAD-систем та комп'ютерних ігор. Він підтримується багатьма операційними системами (в тому числі й Windows) і різними виробниками графічних акселераторів.

Програми, написані різною мовою з використанням функцій OpenGL, можна переносити практично на будь-які платформи і отримувати при цьому однаковий результат. Базовий набір OpenGL включає в себе біля 150 різних команд, які реалізують основні функції для роботи в двовимірному та тривимірному просторах. Але водночас деякі функції для роботи з тривимірними об'єктами відсутні в стандарті, тому створюються допоміжні бібліотеки (наприклад, GLAUX) і здійснюються модифікації базової версії OpenGL, які добавляють нові функції та модифікують існуючі. Розширення OpenGL (нові можливості) добавляють і виробники відеокарт.

Direct X – корпоративний стандарт, всі права на який належать компанії Microsoft. DirectX призначений тільки для платформи Intel під управлінням ОС Windows. Стандарт DirectX підтримує програмування двовимірної графіки, програмування тривимірної графіки, роботу зі звуками і музикою, розробку мережевих ігор тощо. Набір

DirectX дозволяє розробникам ігор та інших додатків отримувати доступ до функцій апаратного забезпечення без написання апаратно-залежного програмного коду. Популярність DirectX пояснюється ще й тим, що він забезпечує потреби розробників ігор та апаратних засобів для створення тривимірної графіки і підтримки мережних віртуальних світів.

4. Використання функцій операційної системи. Різні операційні системи в цьому плані мають різні можливості. Операційна система Windows забороняє прикладним програмам безпосередній доступ до відеопам'яті, оскільки вона тримає контроль над усім, що робиться на екрані. На C++ неможливо звернутись до відеокомірок напряму. Зате Windows 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 на базі бібліотеки 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, плоттер. На зміну GDI прийшов інтерфейс GDI+, який надає багато нових можливостей. Інтерфейс GDI апаратно-незалежний, тобто дозволяє програмісту однаково працювати з будь-яким графічним пристроєм, а прикладній програмі при виведенні графіки використовувати одні й ті ж 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. Контекст пристрою – це структура даних, що описує цей пристрій.

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. Контекст дисплея дозволяє інтерпретувати

кожне вікно прикладної програми як окремий пристрій відображення.

Функції GDI вимагають як один зі своїх аргументів посилання на контекст пристрою DC. Тип такої величини – hdc (Handle Device Context, посилання на контекст пристрою). Контекст пристрою потрібно отримати від ОС або самостійно створити. Наприклад, контекст пристрою відображення на весь екран DC=GetDC(NULL). Для принтера контекст пристрою потрібно створити, тобто

DC = GreateDC(параметри принтера).

Після одержання посилання на контекст пристрою, графічне виведення в GDI здійснюється шляхом виклику з Windows-додатків функцій, що малюють графічні примітиви. GDI підримує 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. Отже, 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. Найбільш відомими графічними інтерфейсами є бібліотеки OpenGL та DirectX із підсистемою Direct3D. Відомі також розробки інших інтерфейсів (наприклад, інтерфейс Glide для графічних відеоадаптерів Voodoo).

3.3. Види комп'ютерної графіки

Незважаючи на те, що для роботи з КГ існує багато різних програм, за способом формування та кодування графічної інформації розрізняють всього три види комп'ютерної графіки: растрова, векторна та фрактальна. Окремим видом комп'ютерної графіки вважають ще й тривимірну графіку. Вони визначаються принципами формування зображення при відображенні його на екрані монітора та при друкуванні на папері.

3.3.1. Растрова графіка

У растрових системах зображення складається із сукупності елементарних графічних елементів – точок, тобто формується з рядків, базовим елементом яких є точка. Растрова модель цифрового зображення – це прямокутна матриця елементів (регулярна сітка точок), кожен з яких у закодованому вигляді зберігає значення яскравості (кольору) відповідної точки зображення. Сукупність точок з їх атрибутами сприймається як одна картинка.

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, який не залежить від інших. 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 пікселя в рядку, y – номер рядка. Пікселі отримують фізичні розміри тільки при виведенні на конкретні пристрої.

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 екрана. Чим вища роздільна здатність, тим менші розміри елементів зображення і тим вища його якість. Точки більшого розміру знижують якість зображення, роблять видимою його дискретну структуру.

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. Файли растрової графіки займають велику кількість пам'яті, оскільки файл включає в себе дані про кожний піксель зображення. Наприклад, стандартна фотографія містить приблизно 1000 × 1500 пікселів (1,5 млн точок) і, якщо для 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 (з'являється ступінчатий ефект). Жодних додаткових деталей при збільшенні растрового зображення розглянути неможливо. При зменшенні зображення кілька точок перетворюються на одну, тому втрачаються дрібні подробиці.

До *переваг растрової графіки* належить висока реалістичність зображень, тонка передача кольорів, можливість впливу на кожний піксель зображення. Крім цього, пристрої виведення (такі як принтери) друкують зображення теж точками, тому растрові зображення легко друкувати.

Растрову графіку застосовують здебільшого не для створення графічних зображень, а при їх обробці. Наприклад, при корекції, ретушуванні фотографій або ілюстрацій, одержаних на сканері чи цифровому фотоапараті. Всі растрові програми умовно можна поділити на статистичні та динамічні. До статистичних редакторів належать Adobe Photoshop, Microsoft Photo Editor, Corel Photo Paint, Paint Shop Pro, Canvas, до динамічних – AnimagicGIF, Photo GifAnimator та ін.

3.3.2. Векторна графіка

Програмні засоби для роботи з векторною графікою призначені, насамперед, для створення ілюстрацій і меншою мірою для їх обробки. Оформлювальні роботи здійснюються засобами векторної графіки набагато простіше.

Растрові й векторні зображення суттєво відрізняються за способом подання графічної інформації. ***Векторні зображення*** – це зображення, які побудовані з графічних примітивів (лінія, крива, коло, прямокутник, полігон тощо), 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 вектором. Контур може бути прямою або кривою лінією, замкненим або від-

критим. 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, тому що 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), векторну графіку називають ще об'єкто орієнтованою графікою. У ній зображення створюється шляхом комбінації різних примітивів. *Файли векторної графіки для створення зображення містять тисячі різних команд типу "рисуй лінію від А до В", 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) існує 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, бо така будова екрана. 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 обчислювальною графікою.

Крім геометричних параметрів, векторні об'єкти мають параметри кольору контуру, 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: 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 вирішується 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. Виняток становить лише файловий формат AI програми Adobe Illustrator. Формат AI підтримується практично всіма 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 проєктно-конструкторських робіт. Векторні засоби широко застосовуються в рекламних агентствах, редакціях, видавництвах, студіях дизайну. За допомогою векторної графіки зручно створювати плакати великих розмірів, растрова графіка тут практично не придатна.

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 фільму, 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 Corel Draw, Adobe Illustrator, AutoCAD, Macromedia Freehand для роботи з двовимірною графікою та 3DStudio Max, 3DImpact, Bryce3D, Xara3D, Alias Maya для тривимірної графіки.

Окремо ще зазначимо проблему щодо перетворення зображень з одного виду графіки в інший. Перевести векторне зображення в растрове просто адже об'єкти на екрані зображуються у растровому вигляді (монітор є растровим пристроєм). Цей процес однозначний. Перетворення растрового зображення на векторне важче, оскільки тут доводиться розв'язувати задачу про те, як краще зобразити вектором ланцюжок пікселів. Цей процес неоднозначний.

3.3.3. Фрактальна графіка

Фрактальна графіка – це технологія створення зображень на основі фракталів, тобто графічне зображення фрактальних множин. Фрактальна графіка базується на фрактальній геометрії. Зображення у фрактальній графіці будується на основі математичних формул.

Під ***фракталом*** у КГ розуміють структуру, яка складається з частин, які в певному розумінні подібні до цілого. Поняття фрактала,

фрактальної геометрії з'явилося в кінці 70-х років і нині має широке застосування. Найвідоміші фракталами є дерева: від кожної гілки відходять менші, схожі на неї.

Фрактальна графіка, як і векторна обчислювальна, однак базовим елементом фрактальної графіки є *математична формула*, тобто жодні об'єкти в пам'яті комп'ютера не зберігаються. Створення фрактальних зображень полягає не в рисуванні, а в програмуванні. Саме зображення будується за допомогою математичних обчислень спеціальними програмами і 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 або 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 кількох 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 символьні перетворення.

У фрактальній математиці виникають чимраз новіші сфери застосування. Наприклад, перспективним напрямом є створення алгоритму фрактального стиску графічної інформації (алгоритм 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 640 × 480 пікселів.

3.3.4. Тривимірна графіка

Тривимірна графіка оперує об'єктами в тривимірному просторі через побудову проекцій на картинну площину. Усі об'єкти обмежені поверхнями. Поверхні задаються полігональними сітками. Як полігон зазвичай обирають трикутник. Усі перетворення над об'єктами задаються відповідними матрицями. Використовуються матриці повороту, перенесення та масштабування. Помноживши вектор координат об'єкта на відповідну матрицю, отримуємо нове положення об'єкта.

Далі необхідно здійснити візуалізацію видимих поверхонь, враховуючи освітлення. Властивості поверхонь описуються у створюваних масивах текстур, які містять інформацію про матеріал, рівень його прозорості, колір відблиску тощо. Прикладом програми для роботи з тривимірною графікою можна назвати 3Ds Max.

Тривимірна графіка, як правило, поєднує векторні і растрові способи формування зображень. Вона широко застосовується в науці, техніці, медицині, дизайні, інженерному проєктуванні, кінематографії, мультимедіа, комп'ютерних іграх тощо.

3.4. 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, тобто це структура даних відповідно до якої дані розташовуються у файлах.*

Графічна інформація у файлах кодується не так, як у пам'яті комп'ютера. Різні графічні файлові формати реалізовують 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:

– ті, що кодують тільки одне зображення;

– ті, що можуть кодувати кілька 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, BIC, 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), MP4 для зберігання мультимедіа, в якому є відео, аудіо і можуть бути субтитри. Перевага формату MP4 – це компактне упакування відео- і аудіопотоків. Максимальна роздільна здатність, яку забезпечує MP4 1440×1080 dpi. Формат MP3 містить тільки аудіо.

У форматі AVI сектори відеоданих чергуються із секторами звукових даних. У форматах мультимедіа 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, DWG);
– 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. Растрові формати призначені виключно для виведення на екран,

вони містять інформацію про екранні пікселі. Відрізняються вони один від одного здатністю нести додаткову інформацію.

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, застосовується для збереження растрових зображень. 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 – високий 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, який є значно коротшим. 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 браузери 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 (в них більше кольорових переходів). З меншими втратами стискуються зображення з високою роздільною здатністю (200–300 dpi) і навпаки. Не бажано зберігати у форматі JPG файли, для яких важливі всі особливості передачі кольорів. 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 у файлі кілька зображень і може використовувати різні моделі кольорів, має найбільш широкий діапазон передачі кольорів – від монохромного до 32-бітного, підтримує багато методів ущільнення, зокрема LZW-стискування.

TIFF на сьогодні найрозповсюдженіший формат (його підтримують практично всі графічні програми), найкращий формат для імпорту растрової графіки у векторні графічні програми.

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 Ilustrator. 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 являє собою повний опис креслення в текстовій формі коду ASCII, що допускає обробку в інших програмах.

Формат PDF – це формат подання 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. У цьому форматі до різного

типу інформації застосовуються найбільш підходящі для них методи стиску.

Формат PNG є відносно новим форматом, що прийшов на зміну GIF. Цей формат використовує стиснення без втрат. Цей формат використовує стискування без втрат методом схожим з LZW. Стиснуті PNG-файли, як правило, менші, ніж аналогічні GIF-файли. Глибина кольору може бути будь якою до 48 біт. В файл формата PNG записується інформація про гамма-корекцію, що забезпечує однаковий вигляд інформації на різних графічних апаратних засобах.

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.

Для передачі даних в мережі Інтернет варто виділити формат DWF для передачі векторних графічних даних, формати GIF, JPEG, PNG – для растрових і формат PDF для передачі документації.

Контрольні запитання та завдання

1. Які задачі розв’язують технічні засоби графічних систем?
2. Назвіть основні класи ЕОМ та їх параметри.
3. Опишіть призначення основних елементів та назвіть основні параметри графічних адаптерів.
4. Опишіть функції вершинного/піксельного процесорів.
5. Назвіть пристрої введення/виведення графічної інформації.
6. Вкажіть основні характеристики сканера, опишіть принципи їх роботи.
7. Порівняйте технології формування зображень на різних моніторах.

8. Назвіть основні характеристики сучасних моніторів.
9. Назвіть функції та характеристики графобудівників/принтерів.
10. Опишіть технології формування зображень різними проекторами.
11. Що таке роздільна здатність графічних пристроїв?
12. Коротко охарактеризуйте найпоширеніші графічні редактори.
13. Що таке вершинні/піксельні шейдери?
14. Які класи інструментальних засобів використовують для створення графічного програмного забезпечення?
15. Які класи графічних мов ви знаєте? Наведіть приклади.
16. Назвіть найвідоміші API для роботи з графікою.
17. Що являє собою інтерфейс GDI? Що таке контекст пристрою?
18. Назвіть види графіки. Дайте їх порівняльну характеристику.
19. Що таке піксель, растр, роздільна здатність растру?
20. Що собою являє растрове/векторне зображення в пам'яті комп'ютера?
21. Назвіть переваги та недоліки растрової/векторної графіки.
22. Що є основним елементом векторної/ фрактальної графіки?
23. Назвіть основні растрові та векторні графічні формати.
24. 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? Як зміниться роздільна здатність, якщо зображення буде містити ту ж саму кількість точок, а розмір картинки збільшиться/зменшиться в два рази?
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) Hight 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 600 dpi. Знайти кількість світлочутливих елементів лінійки фотоприймача, які будуть задіяні в цій процедурі. Скільки датчиків буде задіяно, якщо роздільну здатність сканера зменшити вдвоє?

Розділ 4. Колір. Моделі кольору

4.1. Природа кольору

Поняття кольору в КГ є основним. ***Колір*** – це один із факторів світлового випромінювання. 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). За допомогою хвильової теорії, висунутої Гюйгенсом у 1678 р., було пояснено багато властивостей світла.

Ми розглядатимемо світло як потік електромагнітних хвиль, який після взаємодії з навколишнім середовищем попадає в око, де в результаті фізичної і хімічної реакції виробляються електроімпульси, що сприймаються мозком людини як колір.

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 хвилі довжиною λ від 380 – 430 нм (фіолетовий) до 605 – 780 нм (червоний). Нагадаємо, що 1 нм = 10^{-9} м. Світло належить до досить вузького діапазону електромагнітних хвиль. Поза цим діапазоном знаходяться інші хвилі, наприклад ультракороткі, інфрачервоні, ультрафіолетові, рентгенівські хвилі (рис. 4.1).

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. Світло видимого діапазону з найбільшою довжиною хвилі буде червоним, із найменшою – синім (рис. 4.1). При зміні довжини хвилі кольори плавно переходять один в одний. Чисті кольори існують лише при певних довжинах хвилі (наприклад, чистий фіолетовий – при довжині 400 нм).

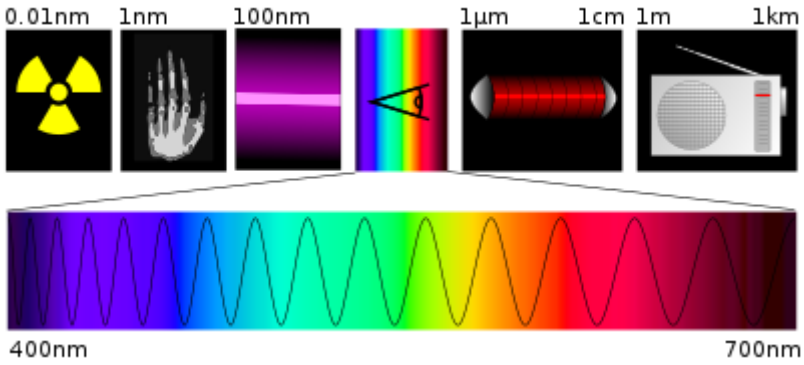


Рис. 4.1. Діапазон хвиль. Хвилі видимого світла

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 потоку світла за одиницю часу дорівнює інтегралу від спектральної функції по всьому видимому діапазону довжин хвиль

$$E = \int_{E} I(\lambda) d\lambda.$$

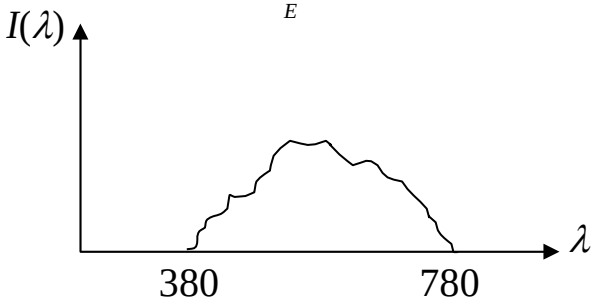


Рис. 4.2. Спектральна крива

Типова спектральна крива наведена на рис. 4.2. Отже, колір однозначно визначається спектральною функцією, але не навпаки. Якщо для спектральної кривої маємо різні значення для функції $I(\lambda)$, то світло називається ***хроматичним***, якщо одинакові значення, тобто всі видимі довжини хвиль наявні в рівних кількостях, то – ***ахроматичним***. Ахроматичне світло це різні відтінки сірого від чорного до білого.

Монохроматичним називається випромінювання, спектр якого складається з єдиної лінії, що відповідає певній довжині хвилі. Досить якісним джерелом монохроматичного випромінювання є лазер. Колір монохроматичного випромінювання й визначається довжиною цієї хвилі.

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 (такі 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. Ті хвилі, які предмет відбиває, попадають в око і визначають колір. Отже, колір визначається спектральним складом світлового потоку та індивідуальністю кожної людини. Світло з різним спектральним складом потрапляє в око людини і формує відчуття кольору.

Розглянемо, як відбувається сприйняття світла людським оком. Людське око дуже складна система (рис. 4.3). Коли очі дивляться на світ – світло попадає в око через рогівку, далі за допомогою кришталика проєктується на сітківку ока, де фоторецептори перетворюють світлову інформацію в імпульси у нервових волокнах. У людей з нормальним зором на сітківці виникає чітке зображення предметів, оскільки воно сфокусовано в центрі сітківки.

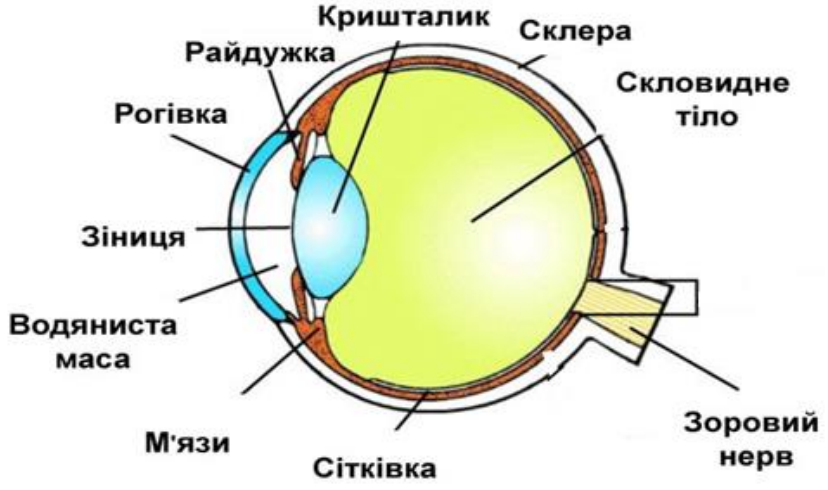


Рис. 4.3. Будова ока людини

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 спект-

ральними 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). У рецепторах енергія світла перетворюється в сигнал, який по зоровому нерву передається в мозок людини.

В основі трикомпонентної теорії світла лежить той факт, що в центральній 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.

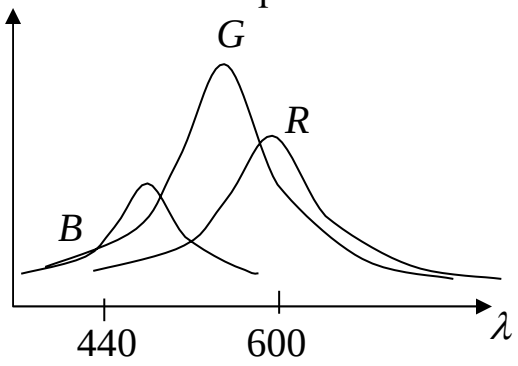


Рис. 4.4. Чутливість до кольорів

Ці три кольори називаються основними (базовими). На рис. 4.4 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. Низька чутливість зору людини до синіх кольорових тонів є причиною того, що синій колір фону добре підходить до ілюстрації кольорових зображень. Якщо, наприклад, білий шрифт надрукований на синьому фоні, то сприйняття фону ніби зменшується, а у відчутті зображення домінує шрифт або елементи зображення з іншим фарбуванням.

Графік кривої, що відповідає за загальну чутливість ока до світла, формується в результаті додавання всіх кривих із рис. 4.4. Сумарна крива спектральної чутливості ока показана на рис. 4.5. Найбільша чутливість людського кольорового зору спостерігається для хвиль із довжиною 555 нм, що відповідають зеленому кольору. Тому з екрана ПК найкраще сприймаються жовто-зелені об’єкти, червоні – дещо гірше, а сині об’єкти сприймаються погано. Крайні кольори важко сприймаються людським оком, яскраво-червоні та фіолетові кольори шкідливі для очей. Оскільки як видно з рис. 4.4 області чутливості колбочок перекриваються, то в процесі сприйняття кольору беруть участь усі три види колбочок. Якщо на всі три види колбочок діє однаковий рівень енергетичної яскравості, то світло буде білим.

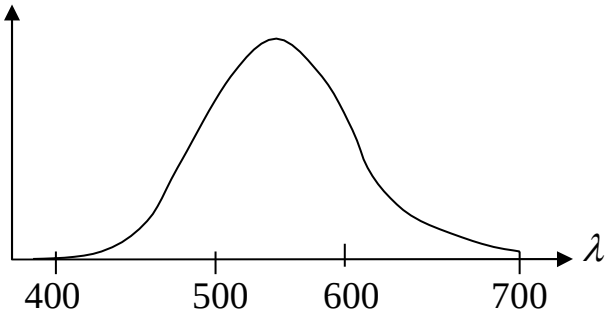


Рис. 4.5. Залежність чутливості людського зору від довжини хвилі світла

При низькому освітленні колбочки втрачають свою чутливість, зате зростає чутливість палочок, що забезпечує нашу здатність бачити при освітленні низького рівня, тому колбочки працюють вдень, сприймаючи колір, форму і деталі предметів, а палочки – вночі, не відчуваючи кольору.

Ньютон у 1676 р. провів експеримент і 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) спрямовували на скляну трикутну призму. Проходячи крізь призму, промінь заломлювався і на екрані давав кольорову смугу – спектр (рис. 4.6). Частинка видимого спектра має унікальне значення і називається *кольором*. У видимому спектрі наявні всі кольори, які плавно переходять один в одний. Видимий спектр складають мільйони кольорів. Ньютон розбив весь спектр на сім ділянок (рис. 4.7), 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.

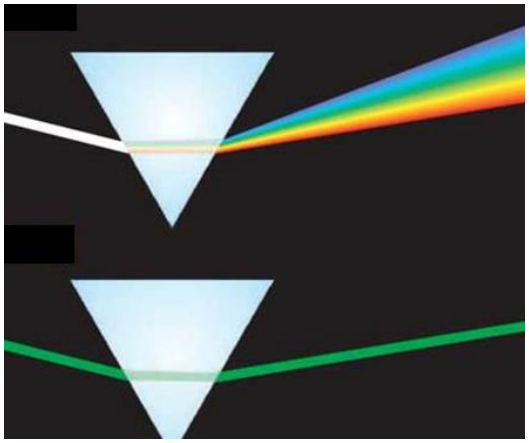


Рис. 4.6. Утворення кольорового спектра

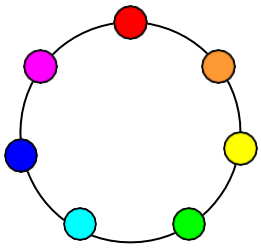


Рис. 4.7. Колірне коло Ньютона

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:

$$C = k_1C_1 + k_2C_2 + k_3C_3,$$

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 компонент 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 із забарвленням предмета певним типом фарби. Тон кольору (відтінок) визначає, яким саме є колір, і дозволяє людині відрізняти кольори (наприклад, зелений від червоного). Фізично тон кольору можна визначити переважаючою (усередненою) довжиною хвилі в спектрі випромінювання. Будь-яке хроматичне світло може бути співставлено з певною усередненою хвилею. Наприклад, світло, в якому переважає хвиля з довжиною 450 нм, буде сприйматися як відтінок синього кольору. Людське око спроможне розрізняти 350 тис. різних кольорів, хоча є й інші дані. Для характеристики відтінків кольорів цього тону вводять поняття яскравості і насиченості.
- ***Яскравість*** (*світлість*). Яскравість визначає, наскільки колір світлий чи темний. Кольори, які мають один і той же тон, можуть відрізнятися величиною світлості. Світлість визначається енергією, інтенсивністю випромінювання на одиницю площі і залежить від амплітуди електромагнітних коливань. Білий колір має максимальну яскравість (100%), чорний колір дає повну відсутність яскравості (0%), тобто чим менша яскравість, тим темніший відтінок. Наприклад, при зменшенні яскравості зеленого кольору колір наближатиметься до чорного, а при збільшенні – до білого. Людське око може розрізняти близько тисячі різних рівнів яскравості.

- ***Насиченість*** характеризує рівень чистоти кольору і визначає кількість білого у відтінку того чи іншого кольору, тобто визначає, наскільки колір є інтенсивним, віддаленим від сірого. Вона виражає співвідношення між основним домінуючим компонентом 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. При зменшенні насиченості кожний хроматичний колір наближається до сірого. 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.

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

Необхідно ще уточнити, що ми розуміємо під тоном кольору. Аналіз спектра, зображеного на рис. 4.8, *а*, дозволяє стверджувати, що випромінює світло-зелений колір, оскільки чітко виділяється одна спектральна лінія на фоні рівномірного спектру білого кольору.

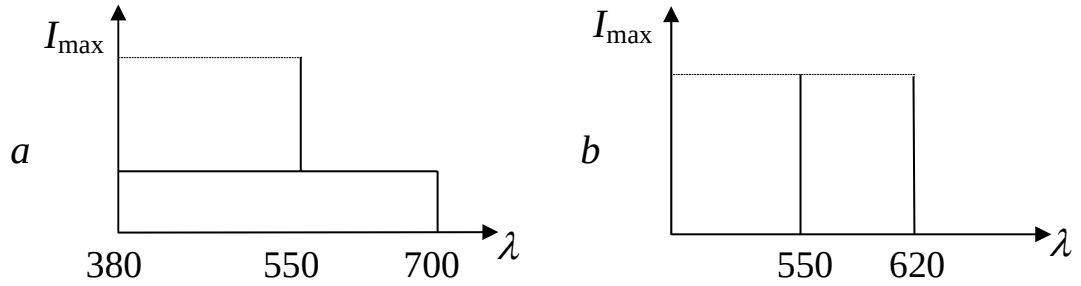


Рис. 4.8. Два спектри: *а* – явна перевага однієї складової;
б – дві складові однакової інтенсивності

А який тон кольору відповідає спектру на рис. 4.7, *б*? Тут неможливо виділити в спектрі переважаючу складову, оскільки наявні червона та зелена лінії спектра 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).

4.2. 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***, тобто *модель кольору* – це система кодування кольорів, яка використовується для зберігання, відображення на екрані та друку зображень, тобто його опис у термінах вибраного кольорового простору.

Існують десятки 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.

4.2.1. Адитивна модель кольору RGB

Адитивна модель кольору найпростіша для розуміння. Вона є досить штучним прийомом. Її використовують для опису кольорів, отриманих за допомогою технічних пристроїв, робота яких базується на принципах випромінювання світла, тобто для пристроїв, що світяться (монітори, побутові телевізори). Це адитивна модель кольору, оскільки для формування потрібного кольору три 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 аргоновий лазер.

Діаграма змішування кольорів зображена на рис. 4.9.

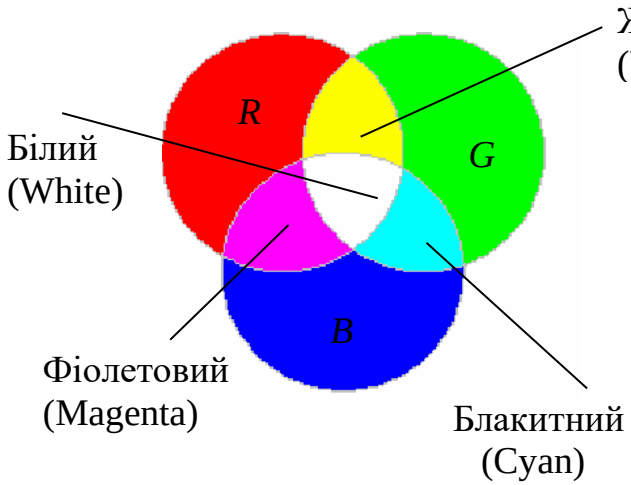


Рис. 4.9. Діаграма змішування кольорів:
жовтий = червоний + зелений
блакитний = зелений + синій
фіолетовий = синій + червоний
білий = червоний + зелений + + синій

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. В багатьох програмах растрової графіки координати R, G, B задаються в цілих числах від 0 до 255. Тоді білий колір – це вектор (255; 255; 255).

Отже, наші очі й мозок, сприймаючи біле світло на екрані, говорять нам неправду, бо це комбінація червоних, зелених і синіх точок.

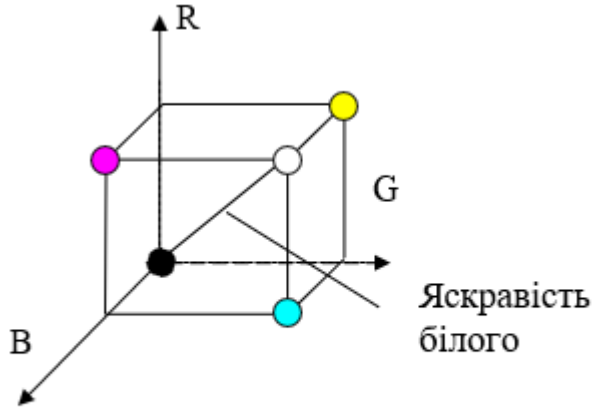


Рис. 4.10. Колірний куб у моделі RGB

Змішування кольорів R, G, B у різних пропорціях дає новий колір, тобто простором кольорів є колірний куб (рис. 4.10). Точки з рівними значеннями R, G, B (R = G = B означає однаковий внесок трьох базових кольорів) лежать на головній діагоналі колірного 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. При виведенні на друк ця модель автоматично перетворюється програмою в модель CMYK.

4.2.2. Субтрактивна модель кольорів CMY/CMYK

Субтрактивна модель використовується для підготовки не екранних, а друкованих зображень, тобто для несамосвітливих 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. Замальований папір деякі електромагнітні хвилі з оптичного діапазону поглинає, решту відбиває, а наше око сприймає лише відбиті хвилі. Колір тут формується як сума відбитих хвиль світлового потоку. Тому для підготовки друкованих зображень використовується не адитивна модель RGB, а субтрактивна модель CMY. На відміну від моделі RGB, біла точка в CMY – це відсутність фарб на папері. Основні правила додавання кольорів тепер такі:

жовтий + пурпурний = червоний,

жовтий + блакитний = зелений,

пурпурний + блакитний = синій,

жовтий + пурпурний + блакитний= чорний.

Назва цієї моделі складається з назв субтрактивних кольорів (протилежних до R, G, B) – блакитного (Cyan), пурпурного (Magenta) і жовтого (Yellow) (рис. 4.11), 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 й доповнюваного 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*.

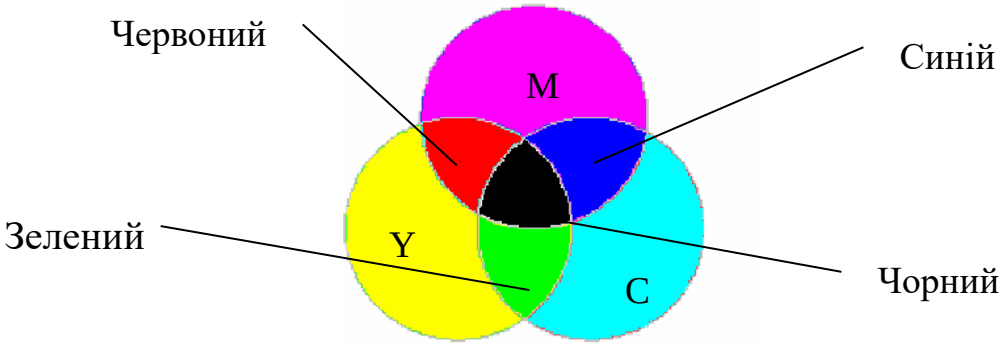


Рис. 4.11. Основні кольори системи CMY

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. 4.12).

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. 4.12).

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. 4.13) Чорний колір відповідає поглинанню всіх кольорів при відображенні (рис. 4.13). 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. 4.12).

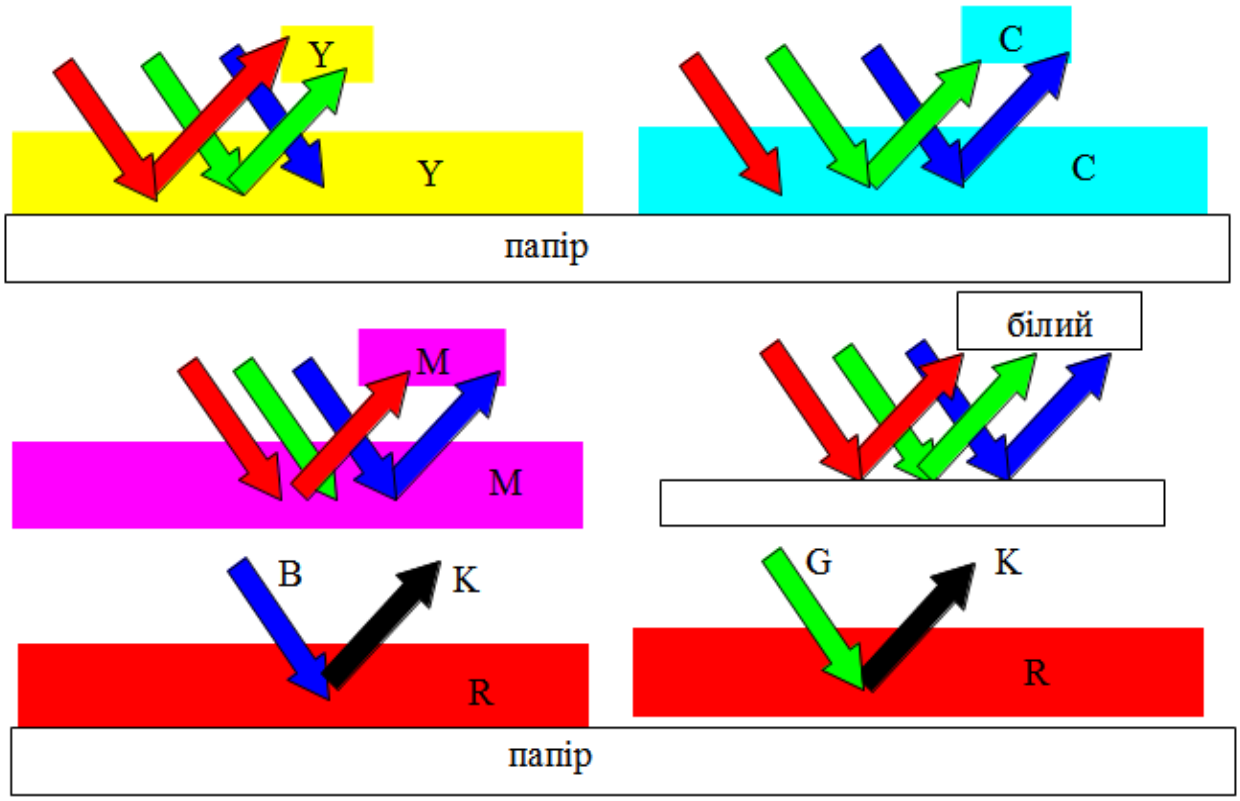


Рис. 4.12. Поглинання (віднімання) кольорів

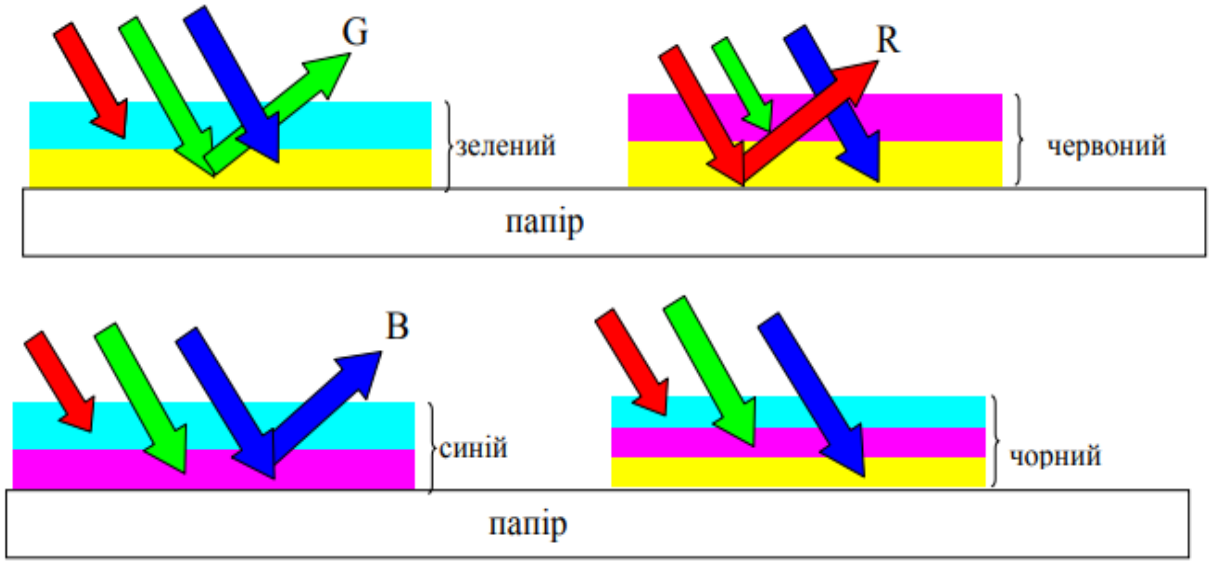


Рис. 4.13. Субтрактивність для двох і трьох кольорів

Істотною проблемою в поліграфії є чорний колір. Теоретично його можна отримати змішуванням трьох доповнюваних 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 струминних принтерах, через небажані візуальні ефекти, тому в принтерах до базових фарб CMY доводиться додавати ще й фарбу чорного кольору (black). Така модель кольору називається CMYK.

При друці малюнка на кольоровому принтері з чотирма кольорами драйвер принтера перетворює RGB-малюнок у модель CMYK. Однак багато відтінків, створених в кольоровій системі RGB, не вдається передати при друці на принтері. А це означає, що колірне охоплення системи CMYK менше, ніж колірне охоплення системи RGB (рис. 4.14). 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 повна за колірним охопленням. Під ***колірним охопленням*** розуміють діапазон кольорів, який може бути відтворений моделлю кольорів.

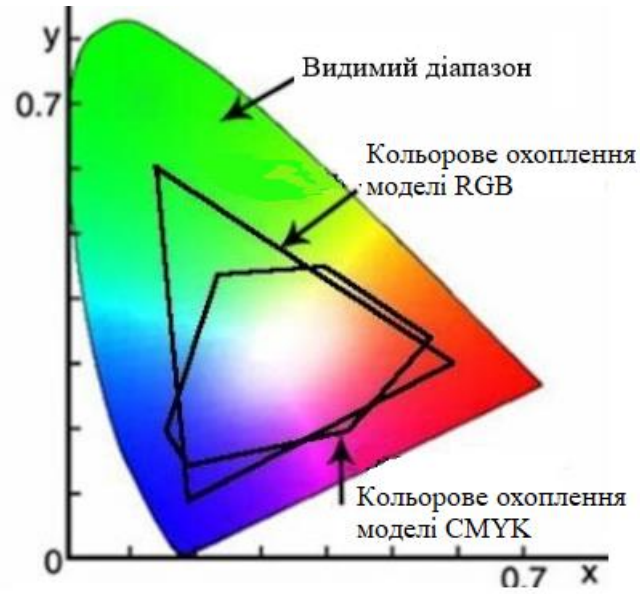


Рис. 4.14. Колірне охоплення моделей RGB та CMYK

У типографіях кольорові зображення друкують у кілька етапів, накладаючи почергово 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 однокольорових зображення. Цей процес називається діленням кольору (separations). Сучасні графічні редактори мають спеціальні засоби для здійснення кольороподілу. Ці програми самі визначають суміш CMYK.

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 зсуненими одна від одної, щоб не було накладання кольорів.

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

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

Зауважимо, що моделі RGB і CMYK дають такий опис кольору, який залежить від типу пристрою виведення, умов освітленості та ін.

4.2.3. 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

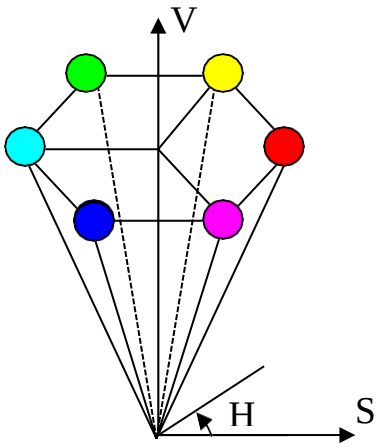


Рис. 4.15. Модель HSB

людського сприйняття кольору. В цій моделі всі кольори теж визначаються трьома координатами. Вона проста та інтуїтивно зрозуміла.

Модель дозволяє задавати кольори, опираючись на інтуїтивні поняття тону кольору H (Hue), насиченості кольору S (Saturation) та яскравості кольору B/V (Brightness/Value). Тому 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 на створення власних художніх творів.

У цій моделі використовується циліндрична система координат, а множина всіх допустимих кольорів є конусом, покладеним на вершину (рис. 4.15). Основа конуса – яскраві кольори, що відповідають значенню V=1 (конус має одиничну висоту).

Значення відтінку H змінюється в градусах від 0^0 до 360^0, оскільки кольори в спектрі видимості світла 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. На колі можна відобразити до 360 відтінків. Кольори, що доповнюють один одного до білого, на колі знаходяться навпроти.

Положення кольору на радіусі кольорового круга визначає його насиченість S (рис. 4.16). Чим далі розміщений колір від центра, тим більш насичений відтінок. Величина S змінюється від 0 до 1.

Значення V = 0 відповідає чорному кольору. При S = 0 (тобто на осі V) маємо сірі відтінки (рис. 4.16). Білий колір кодується як S = 0, V = 1. При S = 0 значення H не має змісту.

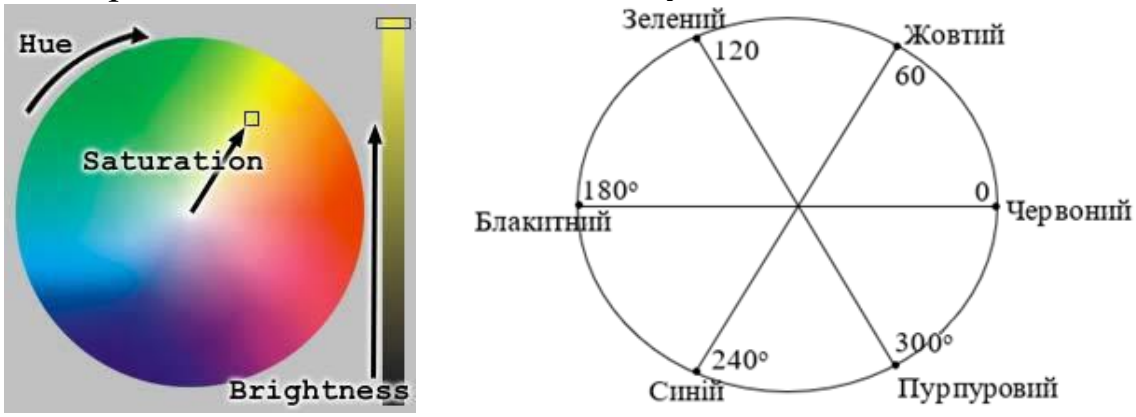


Рис 4.16. Простір кольорів HSB

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).

Людині важко синтезувати потрібний колір у системі RGB, оскільки не існує простого алгоритму для визначення координат R, G, B за тоном, яскравістю та насиченістю. Більш зручно користуватися моделлю HSB, яка дозволяє безпосередньо задати потрібний відтінок. Однак при записі даних у відеопам'ять відбувається переведення значень тону, яскравості, насиченості в систему RGB. У графічних редакторах підбір кольору можна здійснити шляхом інтерактивного вибору з палітри кольорів або підбираючи значення R, G, B до потрібного результату.

4.3. 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 колірного кола (рис. 4.17), 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 з максимальною інтенсивністю дає чистий жовтий колір. Зменшення інтенсивності червоного кольору зміщує результат змішування в сторону зелених відтінків, а зменшення інтенсивності зеленого робить колір оранжевим. І так можна сформувати правила відносно кожної пари кольорів із колірного круга.

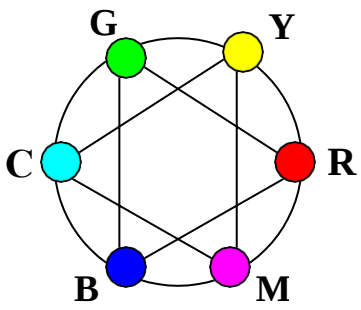


Рис. 4.17. Спрощена схема колірного кола

Щоб підсилити будь-який колір, можна ще здійснити послаблення доповнюваного кольору (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.

Варто ще зазначити, що око людини не володіє високою роздільною здатністю, воно усереднює і змішує відтінки сусідніх пікселів, сприймаючи сукупний колір області як деякий новий колір. Так зелений колір можна отримати чергуванням жовтих і синіх точок, а відтінки сірого можна імітувати змішуванням чорних і білих точок (поставити експеримент).

На цій властивості базується інтерполяція (або псевдотонування) кольору, яка полягає в заміні кольорів пікселів конфігураціями точок з доступними кольорами. Тим самим можна збільшувати колірне охоплення обмежених палітр, створювати більш реалістичні ефекти, імітувати високохудожні техніки та ін.

4.4. 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.

Чорно-білі зображення мають глибину кольору 1. Зображення у відтінках сірого (напівтонові) кодуються 8 бітами. Тут піксель може приймати 256 різних значень від білого (255) до чорного (0).

Зображення з індексними кольорами кодуються 4 або 8 бітами у вигляді таблиці, у якій визначені кольори мають свої індекси (палітри).

Повноколірні зображення мають глибину кольору 24 біти, які розподіляються по 8 на кожний базовий колір R, G, B. Для моделі RGB кожну 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). Тепер 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 з трьох компонент: 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: по одному байту для червоної, зеленої, синьої складових кольору (рис. 4.18) або упаковуються в довге ціле – 32 біти (так, наприклад, зроблено в API Windows).

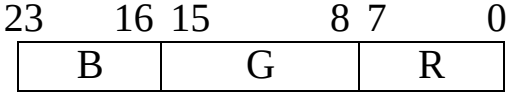


Рис. 4.18. 3-байтовий код

У 32-бітній моделі RGBA старші 8 бітів використовуються для задання компоненти маски (A-компоненти, або Alpha-каналу). Маска створюється деякими графічними програмами для спеціальних ефектів, зокрема 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, як-от 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. 4.19). Цей режим називається Hight Color.

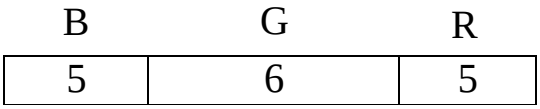


Рис. 4.19. 2-байтовий код

У випадку True Color/Hight 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 браузери (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 із 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. 4.1).

Таблиця 4.1

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	

При 8-бітній глибині кольору можна задати 256 кольорів. Очевидно, що тут можна виділити під кожну 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 × 256 × 256 кольорів вибираються довільні 256 кольорів, які нумеруються індексами від 0 до 255. Вибрані кольори записуються в таблицю кольорів (палітру), а у відеопам'ять 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.

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.

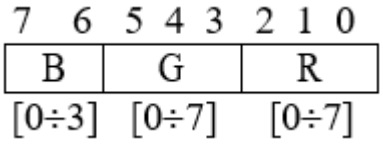


Рис. 4.20. Підіндекси компонент R, G, B

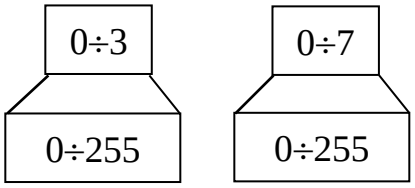


Рис. 4.21. Відображення інтервалів

Розв'язування. Розіб'ємо поле індексу справа наліво на 3 підполя довжиною 3, 3 і 2 біти, відповідно (рис. 4.21). 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). Отже, індекс кольору в палітрі I_c можна подати у вигляді

$$I_c = I_B \cdot 2^6 + I_G \cdot 2^3 + I_R, \qquad (4.1)$$

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. 4.19). 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 (4.1). 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 = [255×6/7] = [218,57] = 218, G = [255×2/7] = [72,86] = 72, B = [255×3/3] = 255. Отже, індексу 214 в рівномірній палітрі з 256 кольорів відповідає колір (218, 72, 255).

4.5. Передача даних про колір

Значення координат кольору в системах RGB, CMYK – це математична абстракція, яка може дати точне описання кольору лише на ідеальних пристроях.

На практиці результати відображення одного й того ж кольору на різних пристроях можуть суттєво відрізнятися, оскільки комп'ютерне устаткування має різні характеристики, принципи дії і програмне забезпечення.

Кожний сканер і монітор мають свої кольорові простори RGB, а кожний принтер – власний CMYK-простір, причому ці простори різні. Скановані кольори виглядають на екрані не так, як в оригіналі, екранні кольори не збігаються з роздрукованими на принтері. Кольори, які зберігаються в графічних файлах виводяться на екран і на друк по-різному. Робота в різних кольорових просторах приводить до виникнення помилок при передачі кольорів від однієї моделі до іншої. Тому необхідно мати відповідні засоби для механізму узгодження кольорів.

Для забезпечення якості кольорів на всіх етапах обробки кольорових зображень використовують апаратно-програмний комплекс, який об'єднує всі засоби управління кольором і називається системою управління кольором CMS (Color Management System).

CMS – це система стандартів, що забезпечує відтворення одного й того ж кольору з одного й того ж графічного файлу практично однаково на різних пристроях виведення. Існує кілька видів таких систем, деякі з них реалізовані на рівні операційної системи. На платформі Windows – це система Image Color Management (ICM).

Суть цієї технології полягає в тому, щоб колір передавався від одного етапу обробки зображень до іншого без спотворень, тобто щоб забезпечити видиму однорідність простору кольорів для всіх периферійних пристроїв і додатків, що працюють у системі. Системи CMS впливають на формування кольору на різних етапах обробки зображень. Задача CMS – це управляти перетворенням кольорів між різними моделями з урахуванням кольорової гамми пристрою, оскільки будь-який пристрій має кольорову гамму, яка завжди вужча, ніж кольоровий охват моделі кольорів. Наприклад, звичайний чотири-

фарбний принтер не може друкувати кольори типу металік. Тому в таких випадках використовують методи імітації кольорів, яких немає в кольоровій гамі пристрою.

Нині жодний кваліфікований комп'ютерний дизайнер не може обійтись без знання основ цієї технології.

В основі сучасних систем CMS лежать дві базові концепції: калібровка і профілювання. *Калібровка* – це зміна (настроювання) параметрів пристрою відповідно до визнаних стандартів. Існують апаратно-програмні та чисто програмні засоби калібровки. *Профілювання* полягає у вимірюванні характеристик пристроїв відображення і збереженні цих даних.

Для узгодження відображення кольорів на різних пристроях вони повинні мати власний профіль, що описує відмінності в представленні кольору на пристрої і в моделі кольорів. *Профіль* – це документ, який описує властивості пристроїв (сканерів, моніторів, принтерів) при передачі і відображенні кольору. Для систем управління кольором профіль дає зведення про правила інтерпретації кольорових координат зображення. Профіль є основним джерелом даних про особливості відтворення кольорів технічними пристроями.

Наприклад, при скануванні кольорового зображення в растровий редактор буде передано значний масив RGB-даних. Щоб редактор правильно інтерпретував кольори, йому необхідно повідомити дані про властивості сканера. Ці відомості надає профіль сканера. Далі щоб друкована версія зображення відповідала оригіналу, потрібно використовувати профіль вибраного принтера. Обидва профілі повідомляють CMS дані для перекодування інформації про колір. Спочатку CMS за профілем сканера встановить істинні значення RGB-оригінала, а потім, використовуючи профіль принтера, переведе їх у відповідні величини CMYK.

Існують стандартні профайли, які постачаються разом із пристроями, а також замовні профайли у вигляді програмних утиліт і професійного ПЗ.

4.6. 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 кольори по-різному діють на психіку людини, тобто колір – це відчуття.

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 – неспокій (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;
- лимонно-жовтий на малиновому;
- чорний на білому;
- темно-синій на оранжевому (жовтогарячому) і т. д.

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 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 запитання та завдання

1. Що таке колір? Як він утворюється?
2. Сформулюйте основні закони колориметрії.
3. В чому суть трикомпонентної теорії кольору?
4. На які основні кольори розкладається біле світло?
5. Що таке колірна модель? Які ви знаєте колірні моделі? Яке призначення кожної колірної моделі?
6. Які кольори називаються доповнюваними?
7. Що таке колірне охоплення? Порівняйте цю властивість для різних моделей кольору.
8. Які атрибути використовує людина для оцінки кольору?
9. Що таке тон кольору? У чому полягає тонова корекція зображення?
10. Що характеризує яскравість/насиченість кольору?
11. Як можна змінити вміст певного кольору?
12. Що таке глибина кольору?
13. Яка максимальна кількість кольорів отримується при глибині кольору 16/24 біти?
14. Що таке палітра кольорів? Для чого вона використовується? Як утворити рівномірну палітру?
15. Скільки різних градацій сірого кольору має режим True Color?
16. Скільки різних градацій зеленого, червоного, синього може бути одержано в режимі Hight Color?
17. 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. Побудувати 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 у 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 у повнокольоровій системі (24 біти на піксель) є буфер кадра 512×512. Скільки кольорів можна відобразити одночасно на екрані?

Розділ 5. Растрові алгоритми генерування ліній

Графічні пристрої, що використовуються в КГ, є в основному растровими. ***Растр*** – це матриця точок заданих на дискретній решітці. Процес генерації растрового зображення геометричного об'єкта називається ***раструванням***.

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 інших растрових алгоритмів.

Зобразити криву на дискретній поверхні означає знайти таку її цілочислову апроксимацію, яка дасть можливість відтворити неперервну криву. Наприклад, для відрізка прямої потрібно отримати послідовність дискретних точок (пікселів), що апроксимує неперервну лінію (рис. 5.1).

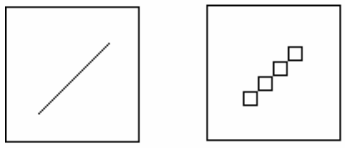


Рис. 5.1. Наближення пікселями неперервної лінії

Растрові методи побудови ліній 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 $0 \le y_2 - y_1 < x_2 - x_1$, тобто цикл у програмі буде здійснюватись по x.

За цих припущень наведемо приклад запису простішого алгоритму побудови прямої лінії мовою C++.

```
void drawLine1(int x1, int y1, int x2, int y2)//Оголошення функції
{
    float a; float b;//Оголошення змінної дійсного типу
    int x;//Оголошення змінної цілого типу
    a = (y2 - y1) / (x2 - x1); //Визначення параметра a
    b = y1 - a * x1; //Визначення параметра b
    for (x = x1;x < x2;x++)//Цикл від x=x1 до x2
    {
     SetPixel(device_context, x, int((a * x) + b), white); //Встановлення пікселя. Перший параметр прив’язка до консолі, потім координати, потім колір.
    }
}
```

Недоліком такої програми є те, що в циклі виконується операція 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:

```
void drawLine2(int x1, int y1, int x2, int y2)//Оголошення функції
 {
        float a; float y;//Оголошення змінної дійсного типу

    int x;//Оголошення змінної цілого типу
    a = (y2 - y1) / (x2 - x1); //Визначення параметра a
    y = y1;
    for (x = x1; x <= x2; x++)//Цикл від x=x1 до x2
    {
    SetPixel(device_context, x, int(y), white);//Встановлення пікселя. Перший параметр прив’язка до консолі, потім координати, потім колір
        y = y + a; //Зміна y при зміні x на одиницю
    }
}
```

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. 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 (метод серединної точки).*

У цьому алгоритмі для обчислення координати y у тілі циклу є тільки одна операція '+', тому, порівнюючи два 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 того, оскільки для обчислення y в останньому варіанті використовується операція $y := y + a$, то з кожним кроком циклу будуть накопичуватися похибки за рахунок неточного представлення дробових чисел у комп'ютері та за рахунок похибки арифметичних операцій із плаваючою точкою. 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 недоліки:

1) використання операцій із плаваючою точкою та операцій множення або ділення в циклі зумовлює зменшення швидкості обчислень, хоча це суттєво залежить від процесора (у сучасних комп'ютерах, де процесори використовують ефективні засоби прискорення, наприклад, конвеєр арифметичних операцій із плаваючою точкою, 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 до створення інкрементних методів (тоді ці методи використовувались для графопобудівників).

Основною ідеєю інкрементних методів є також рекурентні співвідношення для обчислення y, але в них уже не застосовуються дійсні обчислення й операції множення та ділення, а застосовуються лише цілочислові операції додавання віднімання і зсуву (так можна реалізувати швидкі обчислення).

Застосування операцій множення, ділення, а також операцій над числами з плаваючою точкою не бажані, оскільки ці операції зменшують швидкодію обчислень і збільшують апаратні витрати на їх виконання.

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) або (x, y+1). У простішому випадку рух у точку (x+1, y+1) не розглядаємо.

Уведемо 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, 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 $f + m$ ближче 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 $f - n$ ближче 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 $f + m$ ближче 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 $f - n$ ближче до нуля (рис. 5.2, *г*), а $S = 2f + m - n > 0$, то рух відбувається в точку (x, $y + 1$);

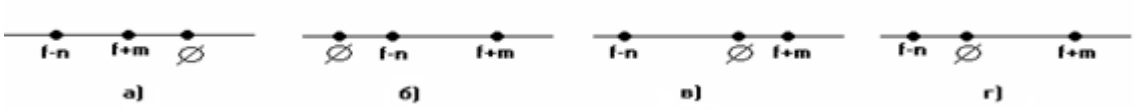


Рис 5.2. Розміщення точок

Отже, вибір наступної точки визначається знаком функції похибок S: якщо $S > 0$, то наступною є точка $(x, y+1)$, а якщо $S \le 0$, 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

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

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

Отже, маємо такі рекурентні формули для S:

$S = S + 2m$, якщо $S \le 0$;

$S = S - 2n$, якщо $S > 0$.

Варіант алгоритму Брезенхема в простішому випадку має вигляд

```
x= 0; y= 0; SetPixel(device_context, x, y, white);
m2=m+m; n2=n+n; S=m-n;
for (i=1; m+n; i++) {
 if (S <= 0) {S += m2;x += 1;}
  else { S=S-n2 ; y+=1; }
  SetPixel(device_context, x, y, white);}.
```

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).

```
void drawLine3(int x1, int y1, int x2, int y2) {
int k; int l; int dy; int S; int m; int n;//Оголошення
змінних цілого типу
        if (x1 > x2)//Розгалуження
    {         S = x1; x1 = x2; x2 = S;
        S = y1; y1 = y2; y2 = S;
    }
    if (y1 > y2)//Розгалуження
    {
        dy = -1;
    }
```

```
    else
    {
        dy = 1;
    }
    m = abs(y2 - y1);//модуль
    n = x2 - x1;
    S = m - n;
    if (m > n)//Розгалуження
    {
        l = m;
    }
    else
    {
        l = n;
    }
    m += m;
    n += n;
SetPixel(device_context, x1, y1, white);//Встановлення
пікселя. Перший параметр привязка до консолі, потім
координати, потім колір
    for (k = 1; k < l; k++)//Цикл від k=1 до l
    {
        if (S <= 0)//Розгалуження
        {
            S += m;
            x1 += 1;
        }
        if (S > 0)//Розгалуження
        {
            S -= n;
            y1 += dy;
        }
    SetPixel(device_context, x1, y1, white);//Встановлення
пікселя. Перший параметр привязка до консолі, потім
координати, потім колір
    }
}
```

Зауваження 2. Для побудови широкої лінії можна її нарисувати кілька 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 неодноразово перефарбовується (рис. 5.3).

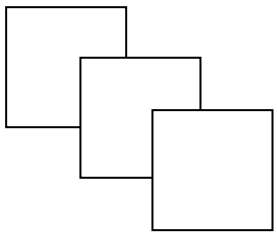


Рис. 5.3. Побудова широкої лінії

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 рисування 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 багатокутника можна встановити яскравість пікселів, пропорційну 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 (xc, yc) створимо таку процедуру:

```
    void circle_8(int x, int y, int xc, int yc) //Процедура
побудови восьми симетричних точок кола з центром у точці
(xc,yc)
    {
HWND console_handle = GetConsoleWindow();
HDC device_content = GetDC(console_handle);
const COLORREF white = RGB(255, 255, 255);
HPEN pen = CreatePen(PS_SOLID, 5, white);
SelectObject(device_content, pen);
SetPixel(device_content, xc + x, yc + y, white);
SetPixel(device_content, xc + y, yc + x, white);
SetPixel(device_content, xc - x, yc + y, white);
SetPixel(device_content, xc - y, yc + x, white);
SetPixel(device_content, xc + x, yc - y, white);
SetPixel(device_content, xc + y, yc - x, white);
SetPixel(device_content, xc - x, yc - y, white);
SetPixel(device_content, xc - y, yc - x, white);
ReleaseDC(console_handle, device_content);
}
```

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 зовнішня до кола, то $D(P) = x^2 + y^2 – R^2$, а якщо внутрішня, 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 $S_1(1, R)$ або $T_1(1, R–1)$ (рис. 5.4), яка найкраще наближає коло.

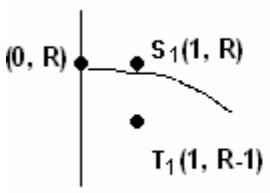


Рис. 5.4

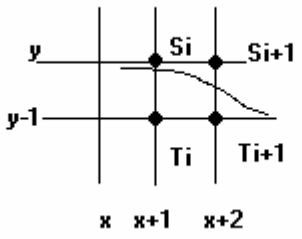


Рис. 5.5

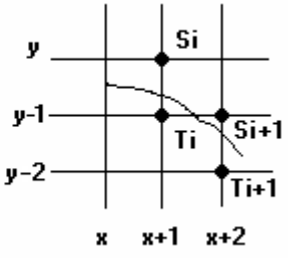


Рис. 5.6

Вибір здійснюється тільки серед двох точок тому, що кутовий коефіцієнт дотичної до кола в цьому октанті, належить проміжку [–1; 0]. 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.

```
void BresenhamCircle() {
    int R, xc, xy;
    std::cout << "Choosing center point." << std::endl;
    std::cout << "Choose center-X, center-Y, radius (it's
one number for all three of them): ";
    std::cin >> R;
    xc = xy = R;
    /*  std::cout << "Input center-X: ";
    std::cin >> xc;
    std::cout << "Input center-Y: ";
    std::cin >> xy;
    std::cout << "Input circle radius: ";
    std::cin >> R;*/
    int x = 0, y = R, d = 3 - 2 * R;
    while (x <= y) {
        circle_8(x, y, xc, xy);
        if (d < 0)d += 4 * x + 6;
        else {
            d += 4 * (x - y) + 10;
            y--;
        }
        x++;
      }
}
```

<u>Зауваження 5.</u> Щоб одержати широке коло, необхідно нарисувати декілька кіл усередині даного кола, послідовно зменшуючи радіус на одиницю, або зовні даного кола, відповідно збільшуючи радіус.

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

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

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

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 підібрали 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 $S(x, y) > 0$.

а *б*

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 (втрачається 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

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

Отже, одержуємо таку процедуру побудови еліпса:

```
void Ellips(int x0, int y0, int Rx, int Ry)
{
    long int S, P, Q, m, n, dp, dq;
    int a, b, x, y;
    b = Ry; a = Rx;
    n = a * a;
    m = (2 * b + 1)*(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;
    SetPixel(dc, x0, y0 + Ry, deep_pink);
    SetPixel(dc, x0, y0 - Ry, deep_pink);
    for (y = Ry; y >= 0; y--)
```

```
    {
        if (S < 0)
        {
            while (S < 0)
            {
                x += 1;
                S += P;
                P += dp;
                P4(x0, y0, x, y);
            }
        }
        else
        {
            S += Q;
            Q += dq;
            P4(x0, y0, x, y);
        }
      }
}
```

Наведемо процедуру побудови симетричних точок еліпса відносно осей координат.

```
void P4(int x0, int y0, int x, int y)
{
    SetPixel(dc, x0 + x, y0 + y, deep_pink);
    SetPixel(dc, x0 + x, y0 - y, deep_pink);
    SetPixel(dc, x0 - x, y0 + y, deep_pink);
    SetPixel(dc, x0 - x, y0 - y, deep_pink);
}
```

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

Hor ($x0 - x$, $x0 + x$, $y0 + y$, c);

Hor ($x0 - x$, $x0 + y$, $y0 - y$, c);

де Hor – процедура відображення горизонтальної лінії, *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 інші – 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).

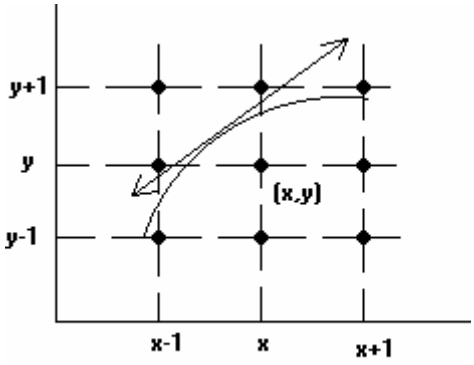


Рис. 5.9. Сусідні точки

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 Δx, Δy, 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:

якщо ($f_y < 0$ і $d=1$) або ($f_y \geq 0$ і $d = 0$), то $\Delta x = 1$;
якщо ($f_y \geq 0$ і $d=1$) або ($f_y < 0$ і $d = 0$), то $\Delta x = -1$; (5.3)
якщо ($f_x \geq 0$ і $d=1$) або ($f_x < 0$ і $d = 0$), то $\Delta y = 1$;
якщо ($f_x < 0$ і $d=1$) або ($f_x \geq 0$ і $d = 0$), то $\Delta y = -1$.

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

якщо $|f^x| < |f^y|$ і $|f^x| < |f^{xy}|$, то ($y := y$, $x := x + \Delta x$);
якщо $|f^y| < |f^x|$ і $|f^y| < |f^{xy}|$, то ($x := x$, $y := y + \Delta y$); (5.4)
якщо $|f^{xy}| < |f^x|$ і $|f^{xy}| < |f^y|$, то ($x := x + \Delta x$, $y := y + \Delta y$).

Для знаходження значень функції використаємо розклад у ряд Тейлора

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

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

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

Частинні похідні в наступних точках визначаємо співвідношеннями

$$
\begin{aligned}
f_x(x+\Delta x, y) &= f_x + f_{x^2}\Delta x + \frac{1}{2} f_{x^3}\Delta x^2 + \ldots \\
f_y(x+\Delta x, y) &= f_y + f_{xy}\Delta x + \frac{1}{2} f_{x^2y}\Delta x^2 + \ldots \\
f_{x^2}(x+\Delta x, y) &= f_{x^2} + f_{x^3}\Delta x + \frac{1}{2} f_{x^4}\Delta x^2 + \ldots \\
f_{y^2}(x+\Delta x, y) &= f_{y^2} + f_{xy^2}\Delta x + \frac{1}{2} f_{x^2y^2}\Delta x^2 + \ldots \\
f_{xy}(x+\Delta x, y) &= f_{xy} + f_{x^2y}\Delta x + \frac{1}{2} f_{x^3}\Delta x^2 + \ldots
\end{aligned}
\qquad (5.6)
$$

Оскільки Δx, Δy можуть набувати значень ± 1, 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 Δx, Δy 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 (xk, yk) останньої точки на кривій, то критерієм зупинки роботи цього алгоритму може бути умова

$$|x - xk| \le 1 \text{ і } |y - yk| \le 1, \text{ або } y = yk \text{ і } |x - xk| \le 1,$$
$$\text{або } x = xk \text{ і } |y - yk| \le 1.$$

Зауваження 8. Для відрізків прямих та кривих другого порядку більш ефективні 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 запитання 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 інкрементних методів.
5. Запишіть простіший алгоритм Брезенхема для відрізка.
6. Який вигляд має функція похибок в алгоритмі Брезенхема для відрізка?
7. Запишіть алгоритм Брезенхема для побудови кола.
8. Яка частина кола формується алгоритмом Брезенхема? 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. Запишіть інкрементний алгоритм Жордана.
12. В чому 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.

Розділ 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*).

а *б*

Рис. 6.1. Способи задання областей

Якщо область задається пікселями, що лежать усередині області, то всі пікселі з області мають одне й теж саме 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*.

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 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 указати 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.

Важливим поняттям растрових областей є їх зв’язність – можливість з’єднати два пікселі області растровою лінією. 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

$$|x1 - x2| + |y1 - y2| \le 1.$$

Такі пікселі називають ще околом фон Неймана.

У 8-зв'язних областях до цих напрямків додаються ще й діагональні, тобто *8-зв'язними* називаються області, в яких кожний 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

$$|x1 - x2| \le 1 \text{ і } |y1 - y2| \le 1.$$

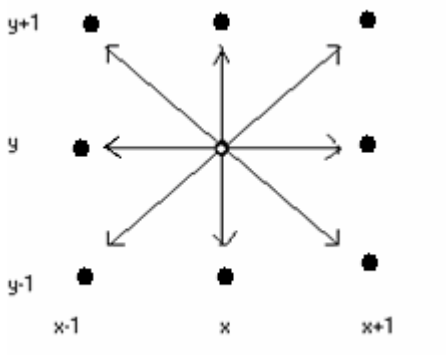


Рис. 6.2. Сусідні пікселі для 8-зв'язної області (окіл Мура)

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 прямокутник, а алгоритм для 8-зв'язних областей заповнить два прямокутники.

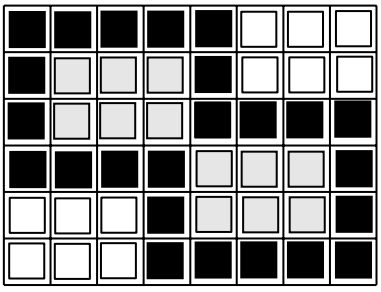


Рис. 6.3. 8-зв'язна область

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 відшуковуються 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.

У методах растрової розгортки в порядку сканування рядків визначають, чи точка лежить всередині контуру і не зафарбована.

6.2. Рекурсивні алгоритми зафарбовування областей

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

```
void Pixel_Fill_4(int x, int y, COLORREF bound, COLORREF new)
    {// (x,y) – координати початковоі точки, з якої почнеться
зафарбовування, bound – колір границі, new – новий колір
    if (GetPixel(dc,x, y) != bound && GetPixel(dc,x, y) !=
new) // якщо точка (x, y) не належить контуру і не
зафарбована в колір new,то{
SetPixel(dc, x, y, new);//зафарбовуємо піксель в заданий колір
    //рекурсивний виклик для сусідніх точок
    Pixel_Fill_4(x + 1, y, bound, new);
    Pixel_Fill_4(x, y + 1, bound, new);
    Pixel_Fill_4(x, y - 1, bound, new);
    Pixel_Fill_4(x - 1, y, bound, new);
 }
}
```

Аналогічний алгоритм маємо для внутрішньо заданої області.

```
void Flood_Fill_4(int x, int y, COLORREF old, COLORREF _new) {
    //(x,y) – координати затравочної точки, old – значення
атрибуту пікселів, що задають область, _new – нове значення
атрибуту пікселя
if (GetPixel(dc, x, y) == old) // якщо точка (x, y)
зафарбована в колір old,то {SetPixel(dc, x, y, _new);
//зафарбовуємо піксель в заданий колір
    //рекурсивний виклик для сусідніх точок
    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);
 }
}
```

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.

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

6.3. Пострічковий алгоритм зафарбовування із затравкою

Розглянемо версію одного з найпопулярніших алгоритмів зафарбовування з затравкою. Ідея цього алгоритму полягає в тому, що для заданої точки (*x, y*) з області визначається і заповнюється максимальний горизонтальний відрізок [*xl, xr*], що лежить усередині області і містить цю точку. Після цього перевіряються точки відрізків області, що лежать вище і нижче знайденого відрізка. Якщо знаходяться такі пікселі, що не належать межі і не зафарбовані, 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, що належить області;
Знайти крайню праву точку *xr* інтервалу, що належить області;
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.

Наведемо ще програму пострічкового алгоритму мовою С++.

```
  //Пострічковий алгоритм зафарбовування із затравкою
void Line_Fill(int x, int y, COLORREF bound, COLORREF_new)
  {
      int xr = x;             //права границя
      int xl = x;             //ліва границя
      //знаходження лівої границі по х
      while (GetPixel(dc, xl, y) != bound) {
          xl--;
      }
      xl++;
      //знаходження правої границі по х
      while (GetPixel(dc, xr, y) != bound) {
          xr++;
      }
      xr--;
      //малюємо лінію від xl до xr заданим кольором
      int i = xl;
      while (i <= xr) {
          SetPixel(dc, i, y, _new);
          i++;
      }
  //перевірка нижніх пікселів відносно намальованої лінії
      i = xl;
      while (i <= xr){
          if (GetPixel(dc, i, y + 1) != bound &&
  GetPixel(dc, i, y + 1) != _new) {
              Line_Fill(i, y + 1, bound, _new);
          }
          i++;
      }
  //перевірка верхніх пікселів відносно намальованої лінії
      i = xl;
      while(i <= xr){
          if (GetPixel(dc, i, y - 1) != bound &&
  GetPixel(dc, i, y - 1) != _new) {
              Line_Fill(i, y - 1, bound, _new);
          }
          i++;
        }
 }
```

Цей алгоритм також рекурсивний, 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).

<u>Зауваження 1.</u> Для зафарбовування простіших фігур (прямокутник, еліпс) можна використати алгоритми виведення контуру, які розглядались 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).

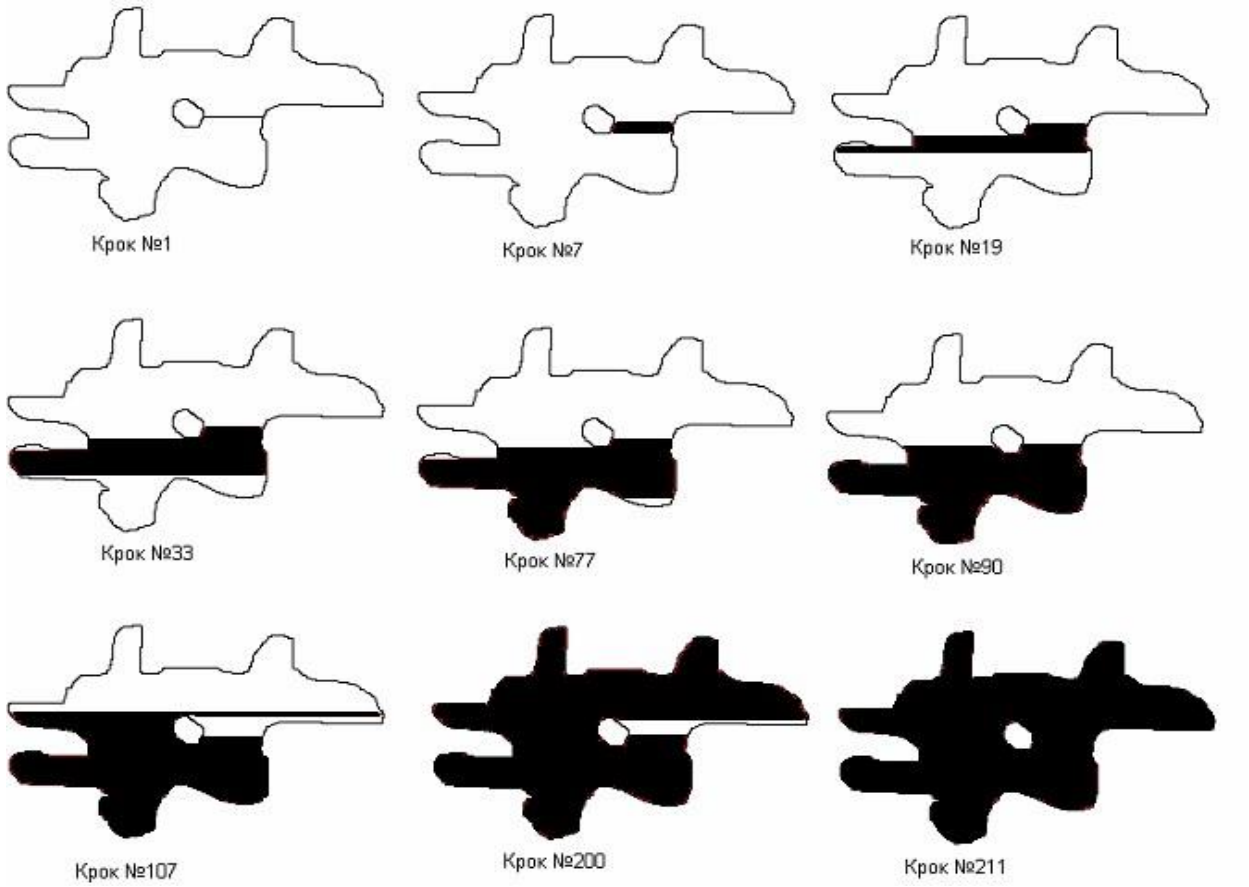


Рис. 6.4. Пострічковий алгоритм зафарбовування областей

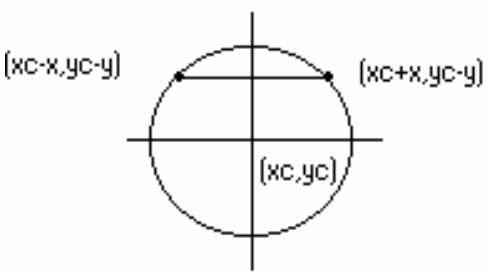


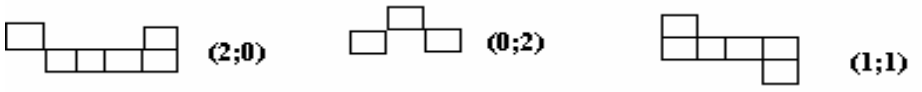
Рис. 6.5. Зафарбовування круга

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 внутрішні чи зовнішні 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, 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*).



а *б* *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.

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 $x \le x_{max}$ *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* ≠ 2, *то* ознака помилки
кінець
кінець
кінець.

Функцію Link визначимо так.

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 полі-гонів. 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 багатокутником (полігоном). Основна ідея алгоритму – зафарбовування полігона відрізками горизонталей.

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

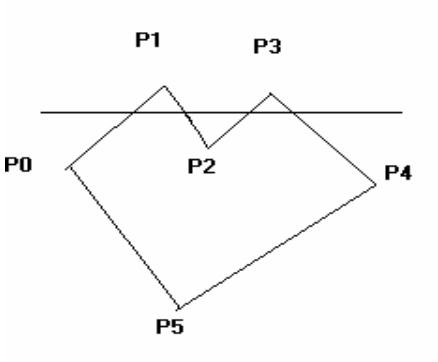


Рис. 6.7. YX-алгоритм зафарбовування полігона

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 усіх вершин P_i;

Для y від $y = y_{min}$ *до* $y = y_{max}$

виконувати початок

Знайти точки перетину всіх відрізків контуру з горизонталлю y;

Координати точок перетину записати в масив X;

Упорядкувати масив X за зростанням координат x_i;

Вивести горизонтальні відрізки з координатами кінців

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

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_iP_{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 досягти зменшення кількості обчислень.

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 приблизно у 2 рази менше). Аналогічно для діапазону [y_{cp}, y_{max}] також складаємо список ребер (їх приблизно вдвічі менше).

Контур можна ділити не навпіл, а на менші частини. Так підвищити швидкодію ефективно 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 й 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 у випадковий спосіб, тобто значення *color = random*( ), то можна створити ілюзію матової поверхні (рис. 6.8), а якщо за *f(x, y)* взяти функцію

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

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

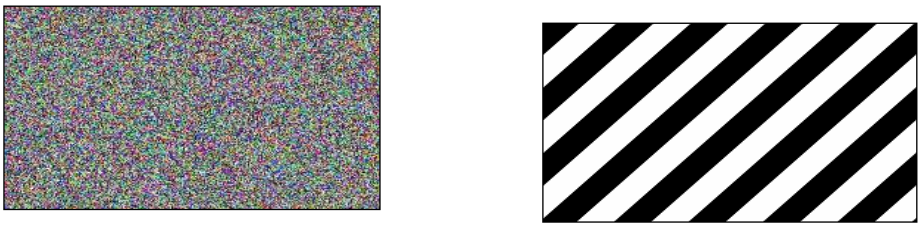


Рис. 6.8. Матова та штрихова фігури

Тут параметр T задає товщину штрихів, S – період нанесення штрихів, $C_ш$ – колір штрихових ліній, $C_ф$ – 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 задає зразок текстури. При цьому координати пікселів структури $x_т$, $y_т$ можуть набувати значень $x_т = 0, 1, ..., m - 1$, $y_т = 0, 1, \ldots, n - 1$. Заповнення фігури можна забезпечити шляхом циклічного копіювання фрагмента зразка всередину 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 ($x_т$, $y_т$) зразка заповнення;
- за координатами ($x_т$, $y_т$) визначити колір *color* пікселя зразка текстури;
- вивести піксель із координатами (*x, y*) і з кольором *color*.

Обчислення координат ($x_т$, $y_т$) через координати (*x, y*) можна здійснити за формулами $x_т = x \bmod m$, $y_т = y \bmod n$, де m, n –

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

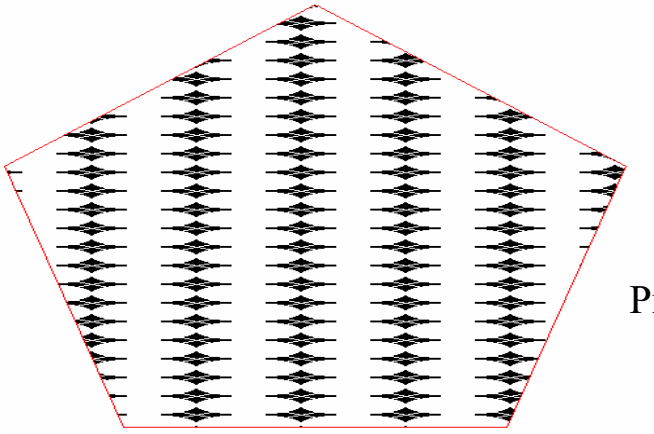


Рис. 6.9. Заповнення області деякими зразками

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:

– 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 у 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 із координатами (i, j), тоді $A = \{ a_{ij} \}$, $i = 0, 1,\ldots, m$, $j = 0, 1, \ldots, n$ – 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 $a_{ij} \in \{0, 1, \ldots, N - 1\}$, то растрове зображення називається напівтоновим і кожний піксель може набувати 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 $x_{ij}^k \in \{0,\ 1,\ \ldots,\ N-1\}$, то зображення називається кольоровим. Колір кожного пікселя визначається координатами (x^1, x^2, x^3) в просторі кольорів.

Розглянемо алгоритми обробки бінарних і напівтонових зображень.

Точкові алгоритми. Точкові алгоритми прості і найбільш часто застосовуються в алгоритмах обробки зображень, зокрема застосовуються для бінарних і напівтонових зображень. В цих алгоритмах для зміни яскравості пікселя 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 реалізувати на основі таблиць перетворення. Якщо перетворення в точкових алгоритмах враховує і розміщення пікселів, то воно задається формулою.

Найпростіші алгоритми цього класу – 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 бінарзації напівтонових зображень, тобто перетворення їх в чорно-біле (бінарне) зображення. Найпротіші алгоритми базуються на пороговій обробці напівтонових зображень шляхом розділення всіх пікселів на два класи за ознакою яскравості: пікселі об’єкта і фону. Формула перетворення яскравості в цих алгоритмах 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).

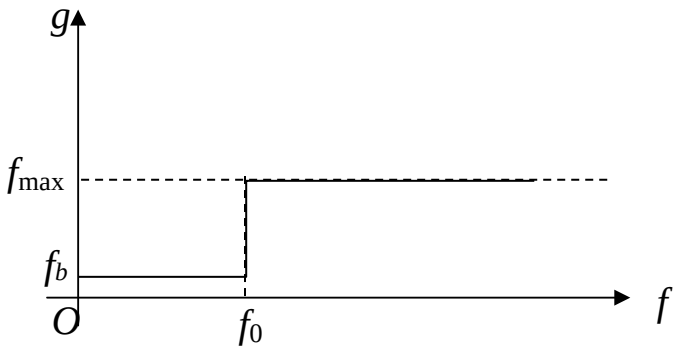


Рис. 6.10. Графік функції яскравості

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.

Наприклад, якщо ми маємо напівтонове зображення, точки якого набувають значення від 0 до 1, де нульова яскравість відповідає чорному кольору, а одинична – білому. Тоді для приведення такого зображення до монохромного вибираємо порогове значення (зазвичай 0,5) і всі точки, яскравість яких більша за порогове значення, стають білими, інші – чорними. Але такий простий алгоритм бінаризації дає не дуже якісні результати.

Існують ще й інші алгоритми бінаризації (псевдотонування) зображень. Більш довершені алгоритми псевдотонування розподіляють чорні та білі пікселі так, щоб на кожному участку зображення яскравість білих точок збігалася з яскравістю цієї ділянки у вихідному зображенні. Один з таких алгоритмів – це алгоритм упорядкованого псевдотонування. У цьому алгоритмі растрове зображення розбивається на блоки (наприклад, 3 × 3). Потім в кожному блоці знаходиться середня яскравість. Відповідно до цього значення вибирається кількість білих пікселів у відповідному блоці монохромного зображення. Ці білі пікселі упорядковуються згідно з деяким шаблоном. Такий алгоритм забезпечує більш якісні результати.

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 яскра-

вості. 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 беруть 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 згортки, 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. Операція згортки називається фільтрацією.

Цей процес повторюється для кожного пікселя зображення (так фільтрується зображення). Коефіцієнти ядра згортки 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):

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

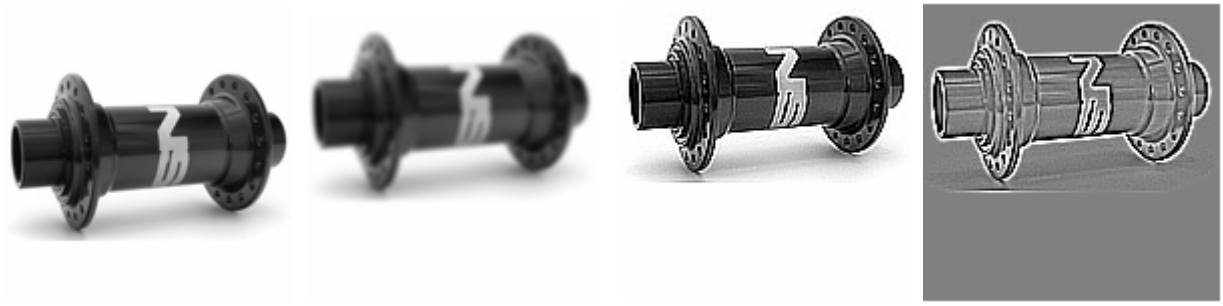
Знову, як і раніше у випадку кольорового зображення червона, зелена і синя складові обробляються маскою окремо, а потім об'єднуються, щоб сформувати 24-бітне значення кольору.

Алгоритми тиснення перетворюють зображення так, що об'єкти сцени виглядають видавленими на деякій поверхні (рис. 6.11, *г*). Тиснення виконується за допомогою ядра згортки, як і раніше. Кожен піксель зображення обробляється ядром тиснення.

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 дорівнюватиме нулю (чорний колір).



а *б* *в* *г*

Рис. 6.11. Обробка зображень: *а* – вихідне зображення; *б* – розмивання зображення; *в* – збільшення чіткості; *г* – тиснення

Після обробки пікселя ядром тиснення до одержаної яскравості для кольорового зображення до кожної складової R, G, B добавляють число 128. Суми, що перевищують 255, заокруглюються до 255. Аналогічно конструюються маски для акварельних зображень.

До просторових алгоритмів належать 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 вивчатимуться далі.

Контрольні запитання та завдання

1. Які є способи задання областей?
2. Які області називаються 4-зв’язними/8-зв’язними?
3. Поясніть 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 YX-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. Визначити 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
$$u=a_1x+b_1y+c_1y+d_1,\ \ v=a_2x+b_2x+c_2y+d_2.$$
8. Розробити алгоритм створення ефекту хвиль на чорно-білому зображенні.
9. Розробити алгоритм побудови контурів предметів на чорно-білому зображенні.
10. Реалізувати алгоритм упорядкованого псевдотонування для растрового напівтонового зображення.
11. Розробити алгоритм морфологічного розширення зображень A та B. Під морфологічним розширенням розуміють операцію $A(+)B = \{t \in R^2: t=a+b, a \in A, b \in B\}$.

Розділ 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 $x_0, x_1, \ldots, x_m$ таких, що $a \le x_0, < x_1 < \ldots < x_m \le b$ (ці точки називаються *вузлами*) відомі значення функції $y_i = f(x_i)$, $i = 0, 1, \ldots, m$, тобто на відрізку $[a, b]$ таблично задана функція $y=f(x)$

x	x_0	x_1	…	x_m
y	y_0	y_1	…	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), $i = 0, 1, \ldots, m$.

Функція $\varphi(x)$ називається *інтерполяційною* для $f(x)$ на $[a, b]$, якщо її значення в заданих точках збігаються з наперед заданими значеннями, тобто $\varphi(x_0) = y_0$, $\varphi(x_1) = y_1$, …, $\varphi(x_m) = y_m$, а для решти $x \in [a, b]$ $f(x) \approx \varphi(x)$. Як функцію $\varphi(x)$ найчастіше використовують інтерполяційні багаточлени $P_m(x)$ 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 разі не можна застосовувати інтерполяційний підхід, оскільки функція $\varphi(x)$ відображатиме всі похибки експериментальних даних. Тому для задачі відновлення функції за експериментальними даними, щоб зменшити вплив випадковостей у результатах вимірювання, використовують процедури згладжування.

Задача згладжування полягає у відновленні гладкої функції $\varphi(x)$, для якої $\varphi(x_i) \approx y_i$, $i = 0, 1, \ldots, m$, тобто потрібно побудувати криву, що проходить поблизу опорних точок (x_i, y_i), 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 багаточленами полягає у побудові багаточлена $P_m(x) = a_0 + a_1x + \ldots a_mx_m$, значення якого в точках x_i відомі, тобто $P_m(x_i) = y_i$, $i = 0, 1, \ldots, m$. Для обчислення коефіцієнтів $a_0, a_1, \ldots, a_m$ одержуємо систему лінійних рівнянь вигляду

$$\begin{cases} a_0 + a_1x_0 + \ldots + a_mx_0^m = y_0, \\ a_0 + a_1x_1 + \ldots + a_mx_1^m = y_1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_0 + a_1x_m + \ldots + a_mx_m^m = y_m, \end{cases}$$

визначник якої відмінний від нуля, як визначник Вандермонда.

7.2.1. Інтерполяційний багаточлен Лагранжа

Інтерполяційний багаточлен $P_m(x)$ можна записувати по-різному. Найчастіше використовуються інтерполяційні багаточлени у формі Лагранжа та Ньютона. Інтерполяційний багаточлен Лагранжа (ІБЛ) має вигляд

$$L_m(x) = \sum_{k=0}^{m} c_k(x)\, f(x_k),$$

де $c_k(x) = \dfrac{(x - x_0)(x - x_1)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_m)}{(x_k - x_0)(x_k - x_1)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_m)}$.

Переваги ІБЛ у тому, що він легко конструюється і його графік проходить через кожну точку (x_i, y_i), $i = 0, 1, \ldots, m$.

Недоліки ІБЛ такі:

1. Степінь ІБЛ прямо залежить від кількості вузлів і чим більша ця кількість, тим вищий степінь ІБЛ, а отже, при обчисленні значень ІБЛ необхідно виконувати більше операцій, що приводить до накопичення похибок.
2. Зміна хоча б однієї точки в масиві вимагає повторного переобчислення коефіцієнтів ІБЛ.
3. Додавання нової точки до масиву збільшує степінь ІБЛ на одиницю і теж зумовлює необхідність його переобчислення.
4. При збільшенні кількості точок ІБЛ може неправильно відтворювати функцію $f(x)$.

7.2.2. Інтерполяційні сплайни

Інтерполювання багаточленом Лагранжа на відрізку $[a, b]$ при збільшенні кількості вузлів часто призводить до неточного наближення та небажаних осциляцій, що пояснюється накопиченням

похибок у процесі обчислень та розбіжністю інтерполяційного процесу. Щоб уникнути великих похибок, відрізок $[a, b]$ розбивають на окремі відрізки, на кожному з яких наближено замінюють функцію $f(x)$ багаточленом невисокого степеня. Це *кусково-поліноміальна інтерполяція*. Одним із варіантів кусково-поліноміальної інтерполяції є кусково-лінійна інтерполяція, при якій апроксимуюча функція лінійна на кожному з відрізків $[x_i, x_{i+1}]$, $i = 0, 1, \ldots, m - 1$. В цьому випадку точки (x_i, y_i) послідовно з'єднуються прямолінійними відрізками (рис. 7.1).

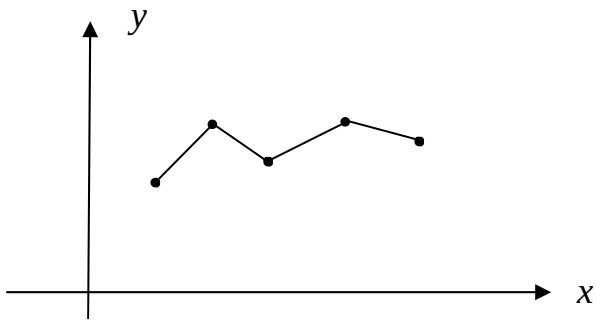


Рис. 7.1. Кусково-лінійна інтерполяція

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.

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

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

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

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

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

Оскільки на кожному відрізку $[x_i, x_{i+1}]$, $i = 0, 1, \ldots, m - 1$ сплайн є багаточленом 3-го степеня, то він визначається однозначно чотирма коефіцієнтами $a_0^i, a_1^i, a_2^i, a_3^i$, $i = 0, 1, \ldots, m - 1$.

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 стрибок прискорення. За другим законом Ньютона стрибок прискорення означає стрибок 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 $S_3''(x_i) = M_i$, $i = 0, 1, \ldots, m$. Тоді $S_3''(x)$ на кожному з відрізків $[x_i, x_{i+1}]$, $i = 0, 1, \ldots, m - 1$ є багаточленом першого степеня, тобто

$$S_3''(x) = M_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + M_{i+1} \frac{x - x_i}{x_{i+1} - x_i},\ x \in [x_i, x_{i+1}].$$

Інтегруючи $S_3''(x)$, знаходимо $S_3'(x)$ і $S_3(x)$. В результаті одержуємо

$$S_3(x) = M_{i+1} \frac{(x - x_i)^3}{6h_i} + M_i \frac{(x_{i+1} - x)^3}{6h_i} + C_1 x + C_2,\ h_i = x_{i+1} - x_i.$$

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,

$$S_3(x) = \frac{M_{i+1}(x-x_i)^3}{6h_i} + \frac{M_i(x_{i+1}-x)^3}{6h_i} + \left(y_i - \frac{M_i h_i^2}{6}\right)\frac{x_{i+1}-x}{h_i} +$$

$$+\left(y_{i+1} - \frac{M_{i+1}h_i^2}{6}\right)\frac{x-x_i}{h_i}, \quad x \in [x_i, x_{i+1}]. \qquad (7.1)$$

Невідомі M_i знаходимо з умов неперервності першої похідної сплайна у вузлах x_i, $i = 1, 2, \ldots, m-1$ та з крайових умов.

Для крайових умов 1-го типу для знаходження M_i одержуємо систему

$$\begin{cases} 2M_0 + M_1 = d_0, \\ \lambda_i M_{i-1} + 2M_i + \mu_i M_{i+1} = d_i, \quad i = \overline{1, m-1}, \\ M_{m-1} + 2M_m = d_m, \end{cases} \qquad (7.2)$$

де $\lambda_i = \dfrac{h_{i-1}}{h_{i-1}+h_i}$, $\mu_i = \dfrac{h_i}{h_{i-1}+h_i}$, $d_i = \dfrac{6}{h_i + h_{i-1}}\left(\dfrac{y_{i+1}-y_i}{h_i} - \dfrac{y_i - y_{i-1}}{h_{i-1}}\right)$,

$$d_0 = \frac{6}{h_0}\left(\frac{y_1 - y_0}{h_0} - f'(a)\right), \quad d_m = \frac{6}{h_{m-1}}\left(f'(b) - \frac{y_m - y_{m-1}}{h_{m-1}}\right). \qquad (7.3)$$

Оскільки $\lambda_i + \mu_i = 1$, $i = 1, 2, \ldots, m-1$, 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 $h_i = h$, $i = 0, 1, \ldots, m-1$, то система (7.2) має вигляд

$$\begin{cases} 2M_0 + M_1 = d_0, \\ M_{i-1} + 4M_i + M_{i+1} = \dfrac{6}{h^2}\left(y_{i+1} - 2y_i + y_{i-1}\right), \quad i = \overline{1, m-1}, \\ M_{m-1} + 2M_m = d_m. \end{cases}$$

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*

$$S_3(t) = y_i(1-t) + y_{i+1}t - \frac{h_i^2}{6}t(1-t)\left[(2-t)M_i + (1+t)M_{i+1}\right], \quad t \in [0, 1]. \quad (7.4)$$

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, $i = 0,1,\ldots, m$ *функцію* $f(x) \in C^2[a, b]$, *тоді справедливі такі оцінки:*

$$\|f(x) - S(x)\|_C = \max_{x\in[a,b]} |f(x) - S(x)| = O(h^3),$$

$$\|f'(x) - S'(x)\|_C = O(h^2), \ \|f''(x) - S''(x)\|_C = O(h).$$

Англійське слово spline перекладається як гнучка лінійка й пов'язане це з такою обставиною: бажаючи провести плавну лінію через задані точки площини, в усіх цих точках фіксують гнучку пружну лінійку; тоді під впливом пружних сил вона набуває потрібної форми, що забезпечує мінімум потенціальної енергії. Саме функція (7.1) задає криву, що описує деформацію гнучкої лінійки, зафіксованої в окремих точках, тобто сплайн і є математичною моделлю профіля гнучкої лінійки (рис. 7.2).

Функція $S(x)$, що описує форму лінійки, між двома сусідніми опорами є багаточленом 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), $i = 0, 1,\ldots, m$ і належать простору $C^2[a, b]$, саме кубічний сплайн $S(x)$, що задовольняє крайові умови $S''(a) = S''(b) = 0$, забезпечує екстремум (мінімум) функціоналу $I(f)$.

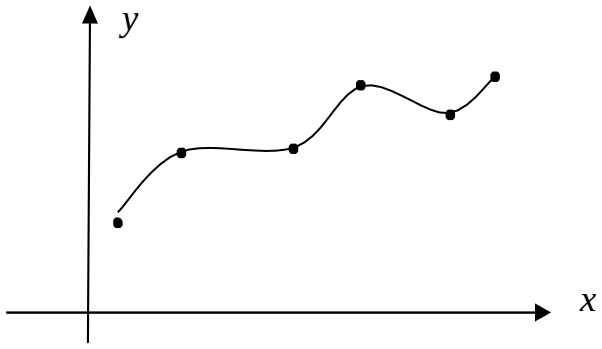


Рис. 7.2. Сплайн-функція

7.3. Згладжуючі кубічні сплайни

На практиці часто значення функції y_i вимірюються експериментально, тому вони містять похибку. А це означає, що при роз-

в'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, крім того, ця функція повинна володіти певними властивостями.

Визначення. *Згладжуючим кубічним сплайном* $S(x)$ на $[a, b]$ *називається функція, яка*

1) *на кожному відрізку* $[x_i, x_{i+1}]$, $i = 0, 1, \ldots, m-1$ *є багаточленом 3-го степеня;*
2) *двічі неперервно диференційована на* $[a, b]$;
3) *доставляє мінімум функціоналу*

$$I(f) = \int_a^b \left(f''(x) \right)^2 dx + \sum_{i=0}^{m} p_i \left(f(x_i) - y_i \right)^2,$$

де $p_i > 0$ *– задані вагові коефіцієнти, від величини яких залежить відхилення значення згладжуючого сплайна в точках* x_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 біля цієї точки.

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 утруднює 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 $P = \{P_0, P_1, \ldots, P_m\}$ 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 називаються *граничними* (кінцевими), а вершини P_i, $i = 1, 2,\ldots, m-1$ – *внутрішніми*.

Розміщення точок P_i, $i = 0, 1,\ldots, m$ 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

$$x = x(t), \quad y = y(t), \quad t \in [a, b]. \tag{7.5}$$

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, зокрема тому, що параметрична форма дозволяє зобразити замкнені криві.

Сплайнові криві шукають у вигляді

$$R(t) = \sum_{i=0}^{m} a_i(t) P_i, \tag{7.6}$$

де $a_i(t)$ – деякі функціональні коефіцієнти, а

$$R(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \; t \in [a, b].$$

Якщо кількість вершин у множині P досить 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 багаточлени однакового степеня, раціональні дроби тощо).

На практиці для опису елементарних фрагментів використовуються багаточлени невисокого степеня, найчастіше третього. Це пов'язане з тим, що багаточлени нижчих порядків не забезпечують необхідної точності апроксимації кривих, а багаточлени вищого степеня можуть викликати небажані коливання згладжуючої кривої. Параметричні кубічні криві – це криві в яких координати точок кривої є багаточленами третього порядку від деякого параметра *t*. 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. Багаточлени 3-го степеня легко обчислюються.

Параметрична крива γ називається C^k-*гладкою,* якщо вектор-функція $R(t)$ є 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 (плавність руху).

Зауважимо, що одиничний вектор дотичної до кривої γ у точці $R(t)$ обчислюється за формулою

$$T(t) = \frac{1}{|\dot{R}(t)|}\begin{pmatrix}\dot{x}(t)\\ \dot{y}(t)\end{pmatrix}, \quad t \in [a, b].$$

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

$$N(t) = \frac{1}{|\dot{R}(t)|}\begin{pmatrix}-\dot{y}(t)\\ \dot{x}(t)\end{pmatrix}, \quad t \in [a, b].$$

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. Із кількох 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 $P = \{P_0, P_1, \ldots, P_m\}$ ***крива Безьє*** степеня m визначається за формулою

$$R(t) = \sum_{i=0}^{m} B_i^m(t) P_i, \quad t \in [0, 1]. \tag{7.7}$$

де $B_i^m(t) = C_m^i t^i (1-t)^{m-i}$ – поліноми Берштейна, $C_m^i = \dfrac{m!}{i!(m-i)!}$, $P_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$

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, ..., P_{m-1} – *контрольними*, Ламана $P_0P_1 \ldots P_m$ іменується *контрольною (опорною)*.

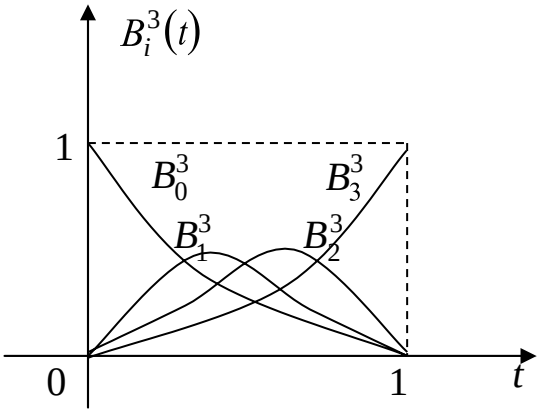


Рис. 7.3. Вагові функції Безьє/Бернштейна при $m = 3$

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 вигляду (випливає 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

$$R(t) = PMT,$$

де $R(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$, $P = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \end{pmatrix}$, $M = \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T = \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$.

Кожне з рівнянь для x, y є багаточленом третього порядку відносно параметра t.

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.

7.5.2. Властивості кривих Безьє

Для згладжувальних кривих Безьє за вагові коефіцієнти беруться *багаточлени Бернштейна*

$$B_i^m(t) = \frac{m!}{i!(m-i)!} t^i (1-t)^{m-i}.$$

Властивості багаточленів Бернштейна суттєво впливають на поведінку кривих Безьє. Наведемо деякі з них:

1. Багаточлени Бернштейна набувають невід'ємних значень.
2. В сумі вони дають одиницю:

$$\sum_{i=0}^{m} B_i^m(t) = 1. \qquad (7.8)$$

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 багаточлена $R(t)$, що визначає криву Безьє, на одиницю менший від кількості точок масиву, тобто крива Безьє побудована на $m+1$ точці задається багаточленом степеня m.
2. Порядок точок у масиві P суттєво впливає на вигляд кривої Безьє. Форма кривої Безьє повторює хід ламаної $P_0P_1\ldots P_m$. При зміні порядку точок повністю змінюється форма кривої (рис. 7.4).

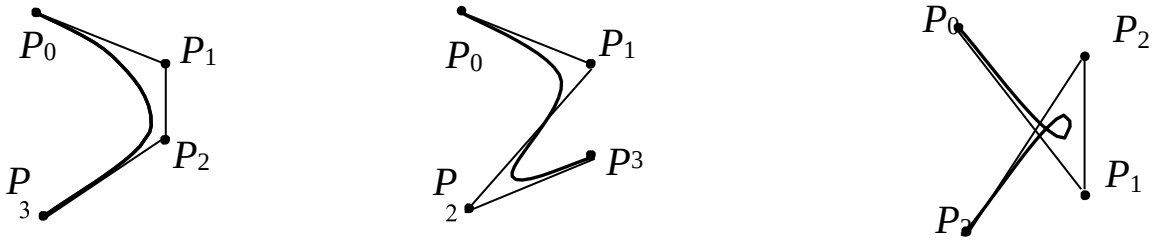


Рис. 7.4. Кубічні криві Безьє при зміні порядку точок (m = 3)

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
$$\dot{R}(t) = m\sum_{i=0}^{m-1}\left(P_{i+1} - P_i\right)B_i^{m-1}(t).$$
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 перетворення над опорними вершинами, а потім побудувати криву Безьє на нових вершинах. Результати будуть однаковими. 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. Але крива Безьє не є проєктивно інваріантною.
8. Якщо в масив P додати хоча б одну вершину, то необхідно повністю переобчислити параметричні рівняння кривої Безьє.
9. Зміна хоча б однієї точки в масиві P приводить до помітної зміни всієї кривої Безьє (рис.7.5).
10. У рівнянні (7.7), що описує елементарну криву Безьє, немає вільних параметрів, тобто заданий масив однозначно визначає криву Безьє. Тому немає можливостей якось впливати на форму кривої Безьє.

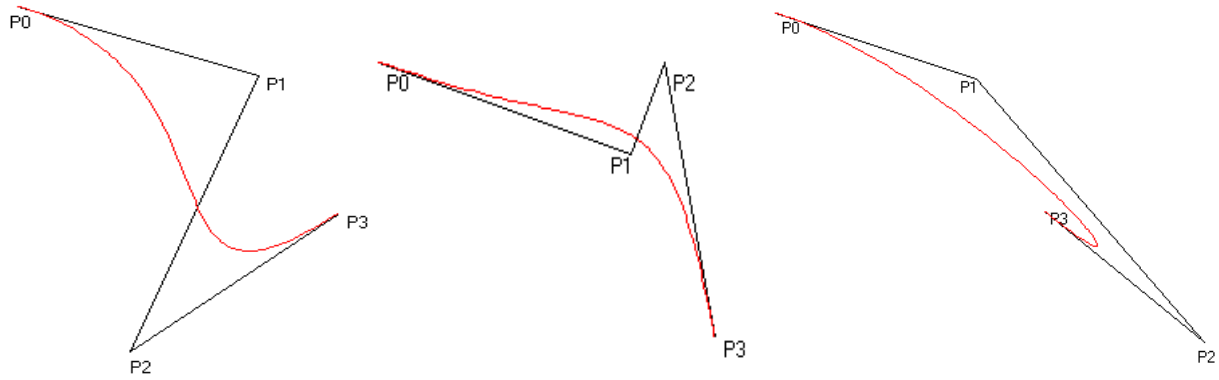


Рис. 7.5. Зміна форми кривої Безьє при зміні однієї точки P_2

Останній недолік можна усунути, якщо ввести до розгляду ***раціональні*** *криві Безьє*, які визначаються формулою

$$R(t) = \frac{\sum_{i=0}^{m} \omega_i B_i^m(t) P_i}{\sum_{i=0}^{m} \omega_i B_i^m(t)}, \quad t \in [0, 1], \qquad (7.9)$$

де $\omega_i \geq 0$ – вагові коефіцієнти, сума яких строго додатна.

Змінюючи параметри ω_i, можна керувати формою раціональних кривих Безьє (рис. 7.6). Якщо значення ω_i велике, то крива $R(t)$ проходить близько до точки P_i , якщо ω_i мале, то $R(t)$ проходить дальше від точки P_i (рис. 7.6). Якщо всі ω_i рівні між собою, то одержується звичайна крива Безьє.

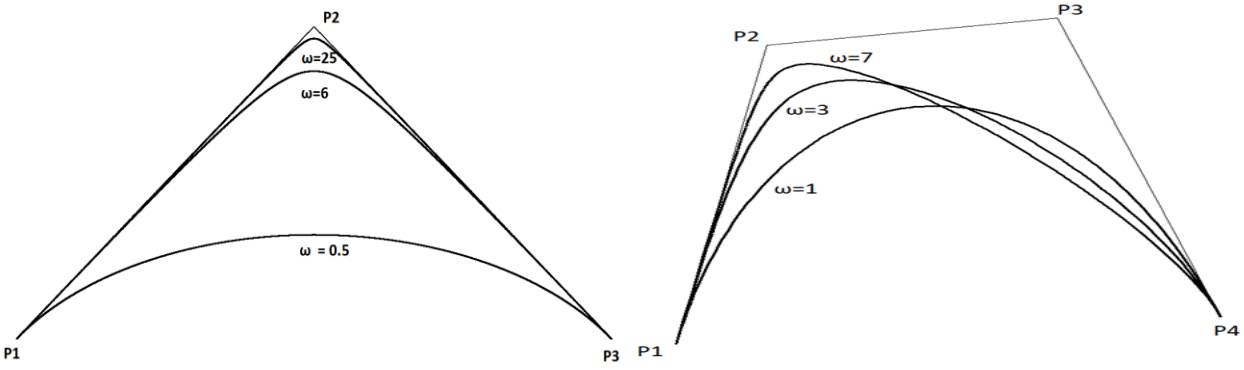


Рис. 7.6. Раціональні криві Безьє

Слово «раціональні» означає, що математичний вираз, що описує криву є відношенням двох багаточленів. Раціональні криві порівняно з нераціональними мають додаткові важливі властивості, хоча при цьому ускладнюється інструментарій їхньої побудови.

1. Вони є проекційно інваріантними, а нераціональні криві тільки афінно інваріантні.

2. Їх можна використовувати для моделювання кривих будь якого вигляду, зокрема для побудови конічних перерізів.

Ці властивості досягаються завдяки тому, що тривимірна точка має чотири координати (*x*,*y*,*z*,*w*), де остання координата *w* означає вагу точки і відображає вплив на поведінку кривої. Зі збільшенням цього значення для точки збільшується степінь її впливу на форму кривої і вона сильніше приближається до точки.

7.5.3. Складені криві Безьє

Кількість точок множини *P* жорстко визначає порядок багаточлена 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.

Зупинимося тепер на побудові складених кубічних кривих Безьє.

Складена *кубічна крива Безьє* – це неперервна крива γ, що є об'єднанням елементарних кубічних кривих γ^0, γ^1, …, γ^l, тобто

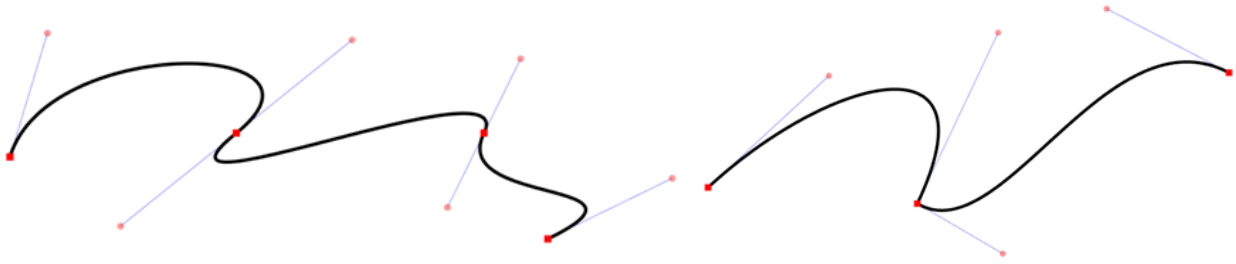
$$\gamma = \gamma^0 \cup \gamma^1 \cup \ldots \cup \gamma^l.$$

Для побудови складеної кривої Безьє розбиваємо масив точок P_0, P_1, …, 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

$$\{P_0, P_1, P_2, P_3\}, \{P_3, P_4, P_5, P_6\}, \ldots, \{P_{3l}, P_{3l+1}, P_{3l+2}, P_{3l+3}\}.$$

На кожній такій підмножині можна побудувати елементарну криву Безьє, тобто окремий фрагмент складеної кривої Безьє. Об'єднання фрагментів дає складену кубічну криву Безьє, яка належить до класу неперервних функцій (рис.7.7 *б*). 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 (рис.7.7 *а*). Якщо у вихідному масиві немає колінеарних трійок вершин, їх завжди можна добавити так, щоб побудована складена крива була C^1-гладкою. C^2-гладкості можна добитися для раціональних складених кривих Безьє за рахунок вагових коефіцієнтів ω_i.



а *б*

Рис. 7.7. Складені криві Безьє

Зауваження 3. Принцип складених кривих Безьє покладено в основу редагування векторних об'єктів у графічних редакторах. 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 такому.

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. 7.8).

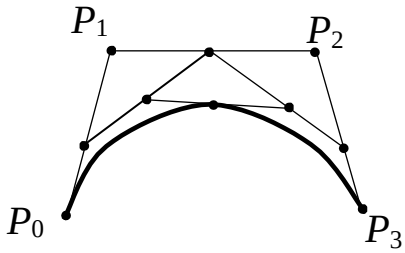


Рис. 7.8. Геометричний алгоритм для кривих Безьє

7.6. B-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 багаточлена Безьє (див. формулу (7.7)). Наприклад, чотири вершини ламаної визначають кубічну криву Безьє, шість вершин ламаної визначають багаточлен 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 (багаточленів Бернштейна B^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 B-сплайна. B-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 B-сплайни, в яких зміна кривизни неперервна. Тому цей тип кривих оптимальний для анімації і створення 2D-форм.

Розглянемо ці специфічні сплайни, що називаються B-сплайнами. Найпоширеніші серед них – B-сплайни третього порядку.

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 B-сплайна.

Функція $B(x)$ є поліномом третього степеня на кожному інтервалі $x_i \le x < x_{i+1}$, 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 задають базові функції *B*-сплайна. Вони мають вигляд

$$n_1(t) = \frac{3t^3 - 6t^2 + 4}{6},$$

$$n_0(t) = \frac{(1-t)^3}{6}, \quad n_2(t) = \frac{-3t^3 + 3t^2 + 3t + 1}{6}, \quad n_3(t) = \frac{t^3}{6}, \ t \in [0, 1].$$

Графіки функцій $n_i(t)$ наведені на рис. 7.9. На кінцях одиничного інтервалу три функції з чотирьох відмінні від нуля.

Отже, елементарну кубічну В-сплайнову криву можна визначити векторним рівнянням:

$$R(t) = \frac{(1-t)^3}{6} P_0 + \frac{3t^3 - 6t^2 + 4}{6} P_1 + \frac{-3t^3 + 3t^2 + 3t + 1}{6} P_2 + \frac{t^3}{6} P_3, \ t \in [0, 1]. \tag{7.10}$$

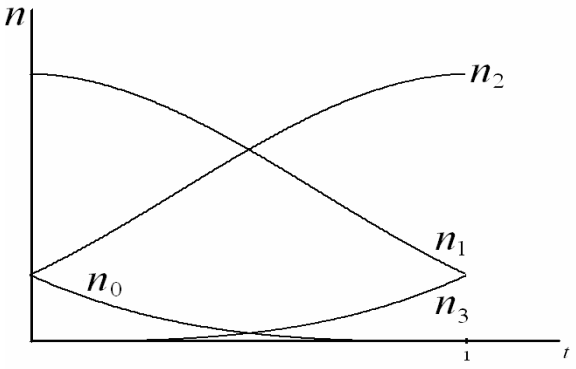


Рис 7.9. Вагові коефіцієнти елементарних В-сплайнів

Матричний запис параметричного рівняння елементарної кубічної В-сплайнової кривої має вигляд

$$R(t) = PMT,$$

де $R(t)=\begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$, $M=\frac{1}{6}\begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 0 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $T=\begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$,

$$P=\begin{pmatrix} P_0 & P_1 & P_2 & P_3 \end{pmatrix}=\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \end{pmatrix}.$$

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. 7.9), 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. 7.10).

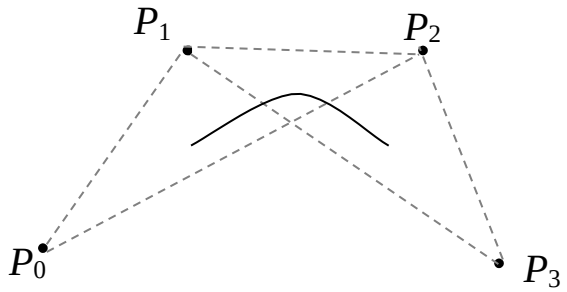


Рис. 7.10. Елементарна кубічна В-сплайнова крива

Дотична в початковій точці $R(0)=\frac{1}{6}(P_0+4P_1+P_2)$ паралельна відрізку 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, ..., P_m, називається крива γ, яку можна подати у вигляді об'єднання елементарних В-сплайнових кривих γ^1, γ^2, ..., γ^{m-2},

$$\gamma = \gamma^1 \cup \gamma^2 \cup \ldots \cup \gamma^{m-2},$$

де γ^i будується як елементарний B-сплайн на точках P_{i-1}, P_i, P_{i+1}, P_{i+2}. Сформулюємо *властивості складеної кубічної B-сплайнової кривої.*

1. Складена кубічна B-сплайнова крива, що породжується масивом вершин P_0, P_1, ..., P_m, є C^2-гладкою кривою, тобто форма кубічної B-сплайнової кривої більш гладка, ніж крива Безьє.
2. Форма кривої визначається контрольними точками, що не лежать на кривій (рис. 7.11). 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}, $i = 1, 2, \ldots, m-2$.
4. Якщо всі опорні вершини лежать на одній прямій, то складена кубічна B-сплайнова крива лежить на цій прямій між точками P_0, P_m.
5. Складена B-сплайнова крива не глобальна, тому локальна зміна однієї точки в масиві призводить до зміни тільки частини кривої; при зміні вершини P_i необхідно переобчислити параметричні рівняння тільки чотирьох елементарних кривих γ^{i-2}, γ^{i-1}, γ^i, γ^{i+1}.
6. Складена кубічна B-сплайнова крива афінно інваріантна.

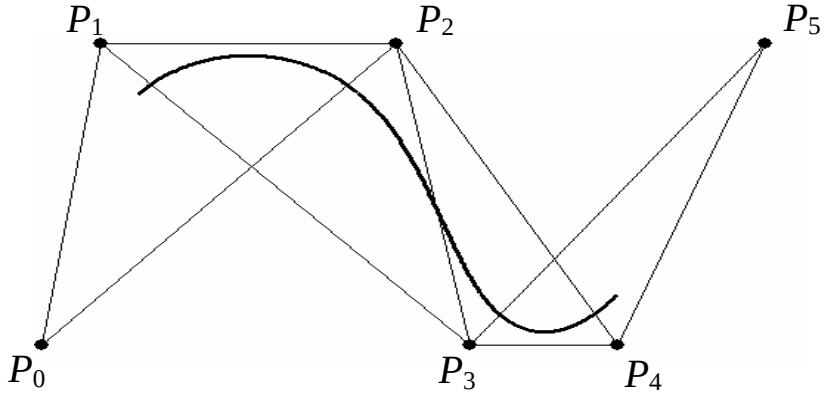


Рис. 7.11. Складена B-сплайнова крива

Зауваження 4. Складена кубічна B-сплайнова крива, як правило, не проходить через жодну з вершин P_0, P_1, ..., 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 (7.10) 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 $t \in [t_i, t_{i+1}]$, $i = 1, 2, \ldots, m - 2$.

Значення t_i визначають точку стику або *вузол* між сусідніми сегментами. Тому t_i називаються ***вузловими значеннями***. Загальна кількість вузлів на складеній кривій дорівнює $m - 1$.

Якщо всі вузли розміщені на рівній відстані за значенням параметра t, то такі B-сплайни називаються ***однорідними***. Без втрати загальності можна вважати, що $t_1 = 0$, $t_{i+1} = t_i + 1$, $i = 1, 2, \ldots, m - 2$, наприклад вектор вузлових значень взяти у вигляді (0,1,2,3,4). У цьому випадку параметричне рівняння i-ї елементарної кубічної B-сплайнової кривої матиме вигляд

$$R^i(t) = \left(P_{i-1}, P_i, P_{i+1}, P_{i+2}\right) M \begin{pmatrix} 1 \\ t - t_i \\ (t - t_i)^2 \\ (t - t_i)^3 \end{pmatrix}, \quad t \in [t_i, t_{i+1}].$$

B-сплайни, в яких відстані між вузловими значеннями нерівні, але задовольняють співвідношення $t_i \le t_{i+1}$, називаються ***неоднорідними*** B-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 B-сплайнів, для неоднорідних B-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

$$N_i^0(t) = \begin{cases} 1, t_i \le t < t_{i+1}, \\ 0, t < t_i, t \ge t_{i+1}, \end{cases}$$

$$N_i^1(t) = \frac{t - t_i}{t_{i+1} - t_i} N_i^0(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} N_{i+1}^0(t), \qquad (7.11)$$

……

$$N_i^k(t) = \frac{t - t_i}{t_{i+k} - t_i} N_i^{k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1}^{k-1}(t), \quad k = 1, 2, 3, \ldots .$$

Графіки базових функцій різного порядку наведені на рис. 7.12.

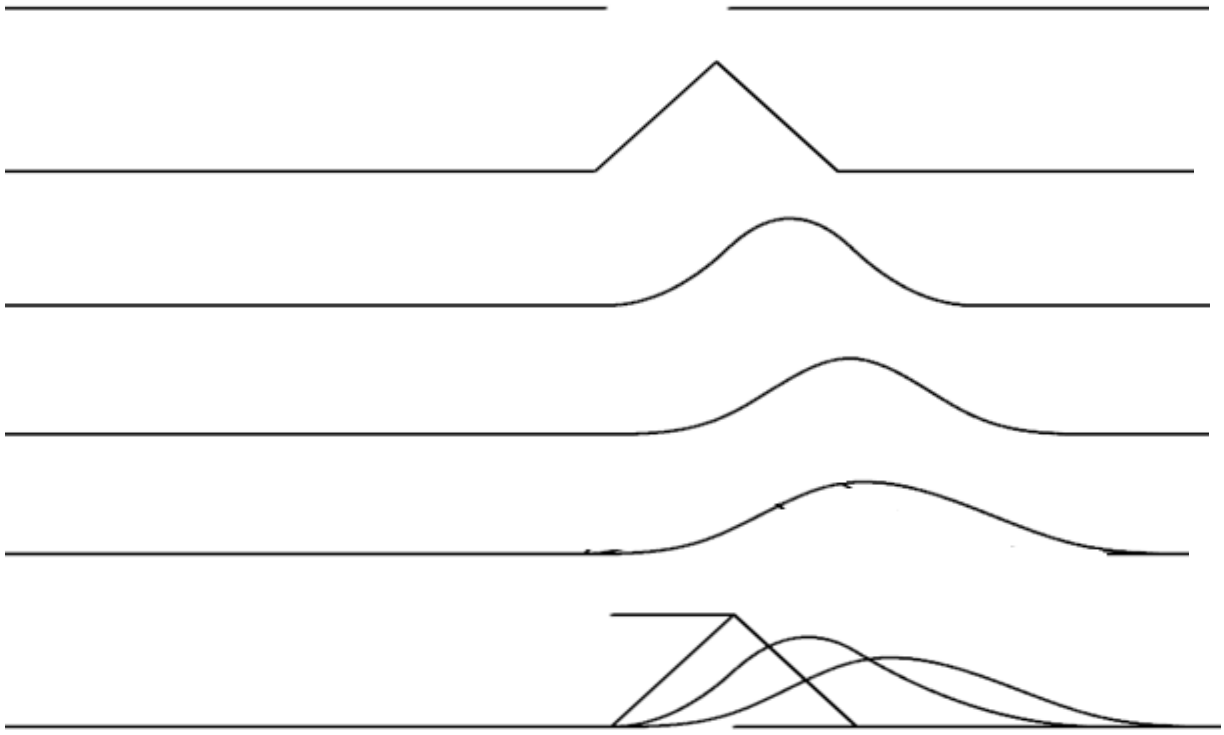


Рис. 7.12. Графіки нормалізованих функцій $N_i^k(t)$
при k = 0, 1, 2, 3, 4, i = 5

Тоді B-сплайн побудований на множині точок $P = \{P_0, P_1, \ldots, P_m\}$ з вектором вузлових значень $t_0, t_1, \ldots, t_k, \ldots, t_m, \ldots, t_{m+\kappa+1}$ має вигляд

$$R(t) = \sum_{j=i-k}^{i} N_j^k(t) P_j, \quad t_i \le t \le t_{i+1}, \quad i = k, k+1, \ldots, m, \quad 1 \le k \le m-1. \quad (7.12)$$

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 $N_i^k(t) = 0$ (рис. 7.12). 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 (7.12), завжди належатиме до опуклої оболонки масиву *P*.

Неоднорідні B-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 B-сплайнами можна ввести додатковий вузол і змінити форму кривої.

Раціональні кубічні B-сплайнові криві. За заданим масивом точок $P=\{P_0, P_1, \ldots, P_m\}$ раціональна кубічна B-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 $\bar{N}_i^3(t) = \frac{\omega_i N_i^3(t)}{\sum_{i=0}^{3} \omega_i N_i^3(t)} > 0$, $\sum_{i=0}^{m} \bar{N}_i^3(t) = 1.$

Тому раціональні B-сплайни – це узагальнення нераціональних B-сплайнів і їх базису. І оскільки не накладається жодних обмежень на розміщення вузлів, то ця функція належить до класу нерівномірних раціональних B-сплайнів. Нерівномірні раціональні B-сплайни в літературі з комп'ютерної графіки називаються ще NURBS-кривими (скорочено від nonuniform rational B-spline). NURBS-криві зберігають властивості B-сплайнів, зокрема вони володіють властивістю афінної інваріантності. Але, крім цього, раціональні B-сплайни інваріантні відносно перспективного перетворення, нераціональні B-сплайни зберігають інваріантність тільки відносно масштабування, повороту і зсуву.

Отже, 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 B-сплайни є стандартом обміну проєктної інформації між системами машинного проєктування, тобто вони використовуються в універсальних форматах подання геометричних даних. Раціональні B-сплайни застосовуються в системах геометричного моделювання, сучасних CAD-системах і 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*. 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

$$R(t) = H_0(t)P_0 + H_1(t)P_1 + H_2(t)Q_0 + H_3(t)Q_1,\ 0 \le t \le 1,$$

де $H_i(t)$, $i = 0,1,2,3$ – ермітові базисні функції (ермітові багаточлени):

$$H_0(t) = (1 + 2t)(1 - t)^2 = 2t^3 - 3t^2 + 1,$$
$$H_1(t) = (3 - 2t)t^2 = -2t^3 + 3t^2,$$
$$H_2(t) = (1 - t)^2 t = t^3 - 2t^2 + t,$$
$$H_3(t) = (t - 1)\, t^2 = t^3 - t^2.$$

Графіки ермітових базисних функцій наведені на рис 7.13.

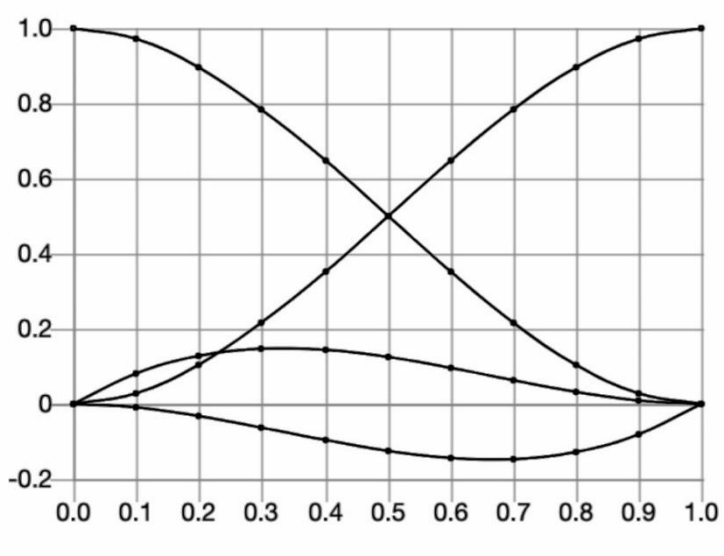


Рис. 7.13. Графіки багаточленів Ерміта при $0 \le t \le 1$

Матричний запис рівняння кривої Ерміта має вигляд

$$R(t) = GMT,\ 0 \le t \le 1,$$

де

$$G = \begin{pmatrix} P_0 & P_1 & Q_0 & Q_1 \end{pmatrix} = \begin{pmatrix} x_0 & x_1 & u_0 & u_1 \\ y_0 & y_1 & v_0 & v_1 \end{pmatrix},\ R = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix},$$

$$M = \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix},\ T = \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}.$$

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. 7.14). Зауважимо, що якщо збільшувати тільки довжину вектора Q_0, то крива Ерміта витягується вправо.

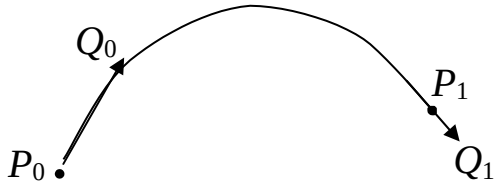


Рис. 7.14. Елементарна крива Ерміта

Складена кубічна крива Ерміта. Складена кубічна крива Ерміта, що визначається масивом точок

$$P_0, P_1, \ldots, P_m, \quad m \ge 1$$

і парою ненульових векторів Q_0, Q_m, – це крива γ, яку можна зобра-

зити у вигляді об'єднання кубічних кривих Ерміта γ^1, γ^2, ..., γ^m, тобто

$$\gamma = \gamma^1 \cup \gamma^2 \cup \ldots \cup \gamma^m.$$

При цьому i-та крива γ^i визначається параметричним рівнянням вигляду

$$R^i(t) = (P_{i-1}\ P_i\ Q_{i-1},\ Q_i)MT, \qquad 0 \le t \le 1,\ i = 1, 2, \ldots, m,$$

де M – базова матриця ермітового сплайна, а вектори $Q_1, \ldots, Q_{m-1}$ 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 $P_0, P_1, \ldots, P_m$.
3. Дотичні вектори Q_i у внутрішніх точках P_i, $i = 1, 2, \ldots, m-1$ 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.

<u>Зауваження 6.</u> За допомогою елементарних кубічних кривих Ерміта складену криву можна побудувати за заданим масивом вершин $P_0, P_1, \ldots, P_m$ і довільним набором ненульових векторів $Q_0, Q_1, \ldots, Q_m$. 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.

<u>Зауваження 7.</u> 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. TCB-сплайни

У задачах анімації при моделюванні руху за допомогою ключових кадрів задається послідовність положень об'єкта, тобто точок P_i, $i = 0, 1,\ldots, m$. 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'язування цієї задачі використовують модифіковані ермітові сплайнові криві. Вони називаються *TCB-сплайнами*.

Розглянемо ці модифікації.

1. У точках P_i, $i = 1, 2, \ldots, m - 1$ потрібно визначити вектори швидкостей, які задаватимуть напрям та швидкість зміни кривої в цих точках.

Наприклад, для напряму можна взяти

$$\boldsymbol{T_i} = \frac{P_{i+1} - P_{i-1}}{2} = \frac{P_{i+1} - P_i + P_i - P_{i-1}}{2}.$$

Окрім цього, зберігаючи напрям вектора, можна регулювати його довжину. Наприклад, помноживши вектор T_i на параметр $1 - t$, де $t \in [-1, 1]$, матимемо

$$\boldsymbol{T_i} = (1 - t)\frac{P_{i+1} - P_{i-1}}{2} = \frac{(1-t)(P_{i+1} - P_i) + (1-t)(P_i - P_{i-1})}{2}.$$

Швидкість $\boldsymbol{T_i}$ визначає натяг кривої (tension). На рис. 7.15 зображені вектори швидкостей для різних значень t. При $t = 0$ вектор $\boldsymbol{T_i}$ – середня лінія $\Delta P_{i-1}P_iP_{i+1}$. При $t \to -1$ швидкість зростає, натяг спадає, при $t \to 1$ швидкість спадає, натяг збільшується.

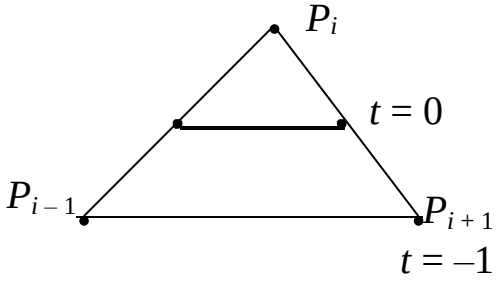


Рис. 7.15. Вектор натягу

2. Вектори $P_{i-1}P_i$, P_iP_{i+1} можуть робити різний внесок у формування вектора швидкості $\boldsymbol{T_i}$. Для врахування цього чинника можна ввести вагові коефіцієнти $1 - \beta$, $1 + \beta$, які описують внесок кожного з векторів P_iP_{i+1}, $P_{i-1}P_i$ у напрямний вектор $\boldsymbol{T_i}$, тобто вважати, що

$$\boldsymbol{T_i} = ((1 - \beta)(P_{i+1} - P_i) + (1 + \beta)(P_i - P_{i-1}))/2.$$

При $\beta = 0$ вектор $\boldsymbol{T_i}$ з'єднує середини відрізків P_iP_{i+1} та $P_{i-1}P_i$. При $\beta \to 1$ внесок вектора $P_{i-1}P_i$ збільшується (вектор швидкості $\boldsymbol{T_i}$ наближається до вектора $P_{i-1}P_i$), а при $\beta \to -1$ збільшується

внесок вектора P_iP_{i+1} (вектор швидкості наближається до вектора P_iP_{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 розрізняти вектори $\boldsymbol{T_i}$ зліва і справа від точки P_i, тобто слід ввести до розгляду вектори $\boldsymbol{T}_i^-$, $\boldsymbol{T}_i^+$.

Аналогічно п.2 вводимо параметр c і задаємо вектори швидкостей за формулами

$$\boldsymbol{T}_i^+ = ((1-c)(P_{i+1}-P_i) + (1+c)(P_i - P_{i-1}))/2,$$

$$\boldsymbol{T}_i^- = ((1+c)(P_{i+1}-P_i) + (1-c)(P_i - P_{i-1}))/2.$$

Параметр c визначає неперервність (continuity) вектора швидкості. При $c = 0$ вектори $\boldsymbol{T}_i^+$ і $\boldsymbol{T}_i^-$ 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 *TCB-сплайнами*. Маючи вектори, на кожній парі точок будуємо ермітові криві, які разом складають TCB-сплайн. Графік TCB-сплайна наведений на рис. 7.16.

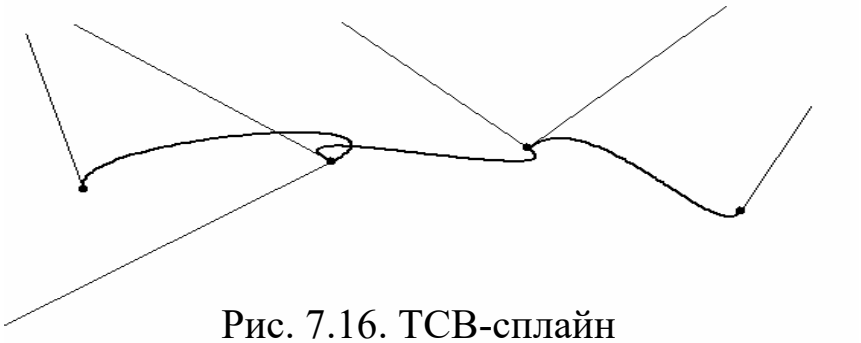


Рис. 7.16. TCB-сплайн

Зауваження 8. Сплайнові криві у випадку простору задаються такими ж самими формулами, тільки як R(t), P_i потрібно розглядати

$$R(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}, \quad P_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}.$$

Зауваження 9. Як правило, всі векторні редактори мають схожі інструменти для редагування кривих. Щоб задати форму кривої, користувач повинен вказати розташування вузлів і кінці дотичних до нього векторів. Щоб крива не мала зламів (була C^1гладкою), дотичні вектори мають бути колінеарними.

Контрольні запитання та завдання

1. Як ставиться задача інтерполяції та задача згладжування?
2. Що таке інтерполяційний сплайн, які він має переваги над інтерполяційним багаточленом? 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 гладкі в точках стику?
12. Вкажіть переваги та недоліки кривих Безьє?
13. Як визначаються B-сплайни третього порядку?
14. Які властивості мають елементарні кубічні B-сплайнові криві?
15. Які властивості мають складені кубічні B-сплайнові криві?
16. Що таке однорідні/неоднорідні B-сплайни?
17. Як визначаються раціональні кубічні B-сплайнові криві? Як можна управляти їх формою?
18. Що таке NURBS-криві, в чому їх переваги?
19. Наведіть визначення кривих Ерміта.
20. Назвіть властивості складеної кубічної кривої Ерміта.
21. Порівняйте криві Безьє, B-сплайни, криві Ерміта в плані інтерполяції контрольних точок і гладкості кривих у точках стику.
22. Що таке TCB-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 четверті. *Вказівка. Для побудови кубічної кривої Безьє потрібно мати чотири точки. Дві крайніх точки уже відомі – це кінці дуги, що задає четверть 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 B-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. Використовуючи квадратурні криві Безьє, сконструювати контур гліфа для букв *u*, *a*, *G*.
7. 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.
8. Вивести параметричні рівняння кривої Ерміта.
   Вказівка. Рівняння необхідно шукати у вигляді
$$R(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} a_1 t^3 + b_1 t^2 + c_1 t + d_1 \\ a_2 t^3 + b_2 t^2 + c_2 t + d_2 \end{pmatrix}.$$
9. Побудувати рівняння кривої Ерміта, що проходить через точки з координатами (0, *a*), (1, *b*), а похідні в початковій та кінцевій точках дорівнюють $y'(0) = u$, $y'(1) = v$.
10. Які властивості має елементарна кубічна B-сплайнова крива, побудована на масиві точок P_0, P_1, P_2, P_3 у випадку, якщо а) $P_1 = P_2$, б) $P_1 = P_2 = P_3$?
11. Де буде починатися/закінчуватися B-сплайнова крива, якщо до масиву точок P_0, P_1, P_2, P_3 добавити опорні вершини P_{-2}, P_{-1}, P_4, P_5 так, що $P_{-2} = P_{-1} = P_0$, $P_3 = P_4 = P_5$?

Розділ 8. Математичні моделі поверхонь

Поверхні та їх опис відіграють важливу роль у конструюванні та виробництві. Наглядні приклади – проєктування 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 галузях науки і техніки.

Лінії та поверхні часто будуються на екрані комп'ютера кінематичним методом. Згідно з цим методом лінія утворюється неперервним рухом точки по заданій траєкторії, а поверхня – в результаті руху в просторі одних ліній, які називаються твірними, вздовж інших ліній, які називаються направляючими.

За допомогою програми MathCAD можна сформувати кінематичним методом просторові криві і поверхні. Застосування універсального пакету MathCAD дозволяє користувачу розробляти моделі будь-яких ліній та поверхонь за своїм алгоритмом і отримувати їх зображення на екрані комп'ютера. Оволодівши методикою конструювання ліній і поверхонь за параметричними рівняннями, студенти зможуть розробляти моделі складних об'єктів та їх реалізації.

Для комп'ютерної графіки і автоматизованого проєктування 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

$$x = F_x(u, v),\ y = F_y(u, v),\ z = F_z(u, v),\ (u, v) \in D, \qquad (8.1)$$

а також векторна $r = r(u, v)$, $(u, v) \in D$, де D – деяка область параметрів u, v.

Наведемо приклади деяких поверхонь та їхні параметричні рівняння.

Еліпсоїд: $x = a \cos u \sin v,\ 0 \le u \le 2\pi,$

$y = b \sin u \sin v,\ \ 0 \le v \le 2\pi, z = c \cos v.$

Однополий гіперболоїд:

$x = a \cos u \operatorname{ch} v, 0 \le u \le 2\pi,$

$y = b \sin u \operatorname{sh} v,\ -\pi \le v \le \pi,$

$z = c \operatorname{sh} v.$

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 із числовим програмним управлінням. Різець при цьому рухається за законом, що задається в параметричній формі.

Але параметричні поверхні мають і ряд недоліків, зокрема значні обчислювальні затрати, що пояснюється необхідністю застосування числових, а не аналітичних методів. Ілюстрації параметрично заданих поверхонь, як правило, не мають тіні, не передають прозорості й дзеркального відбиття сусідніх об’єктів. Це зумовлено 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 суміжними полігонами.

Особливий клас поверхонь – це бікубічні поверхні. Бікубічні сплайни разом із полігональними сітками є найбільш поширеним способом зображення тривимірних поверхонь у просторі. За допомогою бікубічних сплайнів 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

$$r(t) = tr(0) + (1 - t)r(1),\ t \in [0, 1].$$

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 (рис 8.1).

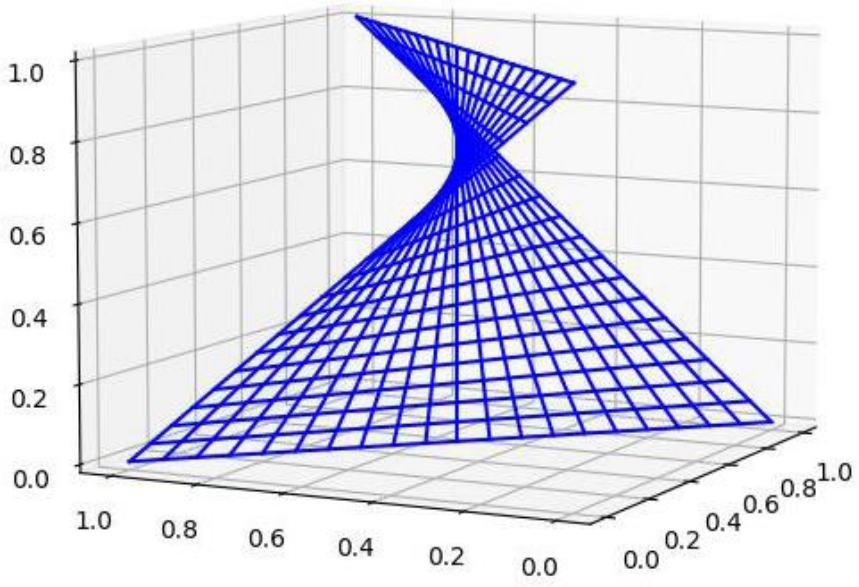


Рис. 8.1. Білінійна поверхня побудована на точках (0,0,1), (1,1,1), (0,1,0), (1,0,0)

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 (рис. 8.2) 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 білінійну 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.

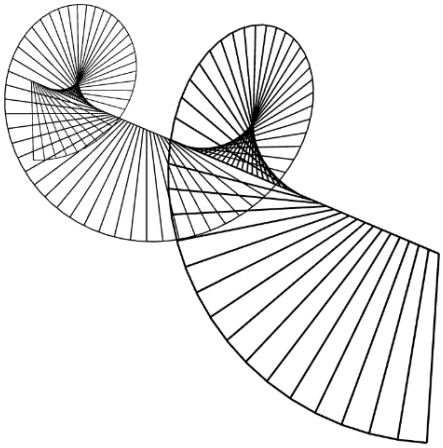


Рис. 8.2. Гвинтова поверхня

8.2. Інтерполяційні бікубічні сплайни

Нехай на площині *xy* задано набір з $(n + 1)(m + 1)$ точок $(x_i, y_j) \in D$, $i = 0, 1, \ldots, n$, $j = 0, 1, \ldots, m$ $(x_0 < x_1 < \ldots < x_n)$, 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.

Визначення. *Інтерполяційним бікубічним сплайном називають функцію* $z = S(x, y)$, *що має такі властивості:*

1) *для всіх точок* $(x_i, y_j) \in D$, $i = 0, 1, \ldots, n$, $j = 0, 1, \ldots, m$ *справедлива рівність* $S(x_i, y_j) = z_{ij}$;
2) *на кожному прямокутнику* $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$, $i = 0, 1, \ldots, n - 1$, $j = 0, 1, \ldots, m - 1$ $S(x, y)$ *є багаточленом 3-го степеня, тобто*

$$S^{ij}(x, y) = \sum_{i=0}^{3} \sum_{k=0}^{3} a_{lk}^{ij} (x - x_i)^l (y - y_j)^k ;$$

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 16 mn 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.

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 неточно, 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, які описують координати точок на викривленій поверхні за допомогою трьох рівнянь, кожне з яких має два параметри, причому показники степеня при них не вищі за третій (звідси назва бікубічний). Найрозповсюдженіші сплайн-поверхні – це поверхня Безьє та B-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. Межі бікубічних фрагментів поверхонь – це бікубічні криві. Для представлення поверхонь із заданою точністю потрібно значно менше бікубічних сплайнів, ніж при апроксимації полігональною сіткою. Проте алгоритми для роботи з бікубічними сплайнами складніші за алгоритми для роботи з полігональною сіткою.

Зробимо узагальнення від просторових сплайнових кривих до тривимірних криволінійних поверхонь.

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

$$r(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} a_i^n(u) b_j^m(v) P_{ij}, \quad \text{i} = 0,1,\dots,n, j = 0,1,\dots,m, \quad (8.2)$$

де P_{ij} – опорні точки-орієнтири тривимірного простору, $r(u, v)$ = colon($x(u, v)$, $y(u, v)$, $z(u, v)$). Якщо ці точки відповідно з'єднати прямолінійними відрізками, то одержимо контрольну (опорну) багатогранну поверхню (рис. 8.3).

Оскільки співвідношення (8.2) можна переписати у вигляді

$$r(u,v)=\sum_{i=0}^{n} a_i^n(u)s_i(v),$$

де $s_i(v)=\sum_{j=0}^{m} b_j^m(v)P_{ij}, i=0,1,\dots,n,$

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:

$$a_i^n(u)=B_i^n(u)=C_n^i u^i(1-u)^{n-i}, \qquad C_n^i=\frac{n!}{i!(n-i)!},$$
$$b_j^m(v)=B_j^m(v)=C_m^j v^j(1-v)^{m-j}, \qquad u, v\in[0,1],$$

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

$$\begin{pmatrix}x(u,v)\\ y(u,v)\\ z(u,v)\end{pmatrix}=(1\ \ u\ \ u^2\ \ u^3)M^T\begin{pmatrix}P_{00}&P_{01}&P_{02}&P_{03}\\ P_{10}&P_{11}&P_{12}&P_{13}\\ P_{20}&P_{21}&P_{22}&P_{23}\\ P_{30}&P_{31}&P_{32}&P_{33}\end{pmatrix}M\begin{pmatrix}1\\ v\\ v^2\\ v^3\end{pmatrix}, \qquad (8.3)$$

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. 8.3).

Елементарна бікубічна поверхня Безьє успадковує багато властивостей 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 із 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.

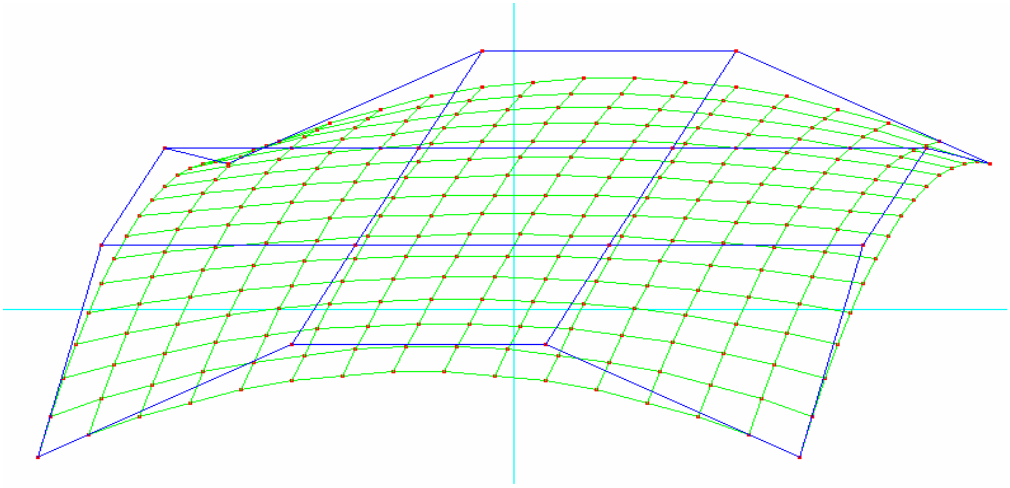


Рис. 8.3. Елементарна бікубічна поверхня Безьє та її контрольна поверхня

8.3.2. B-сплайнові поверхні

Векторне параметричне рівняння елементарного фрагмента бікубічної B-сплайнової поверхні (рис.8.4), що задана на масиві точок P_{ij}, $i, j = 0, 1, 2, 3$, має вигляд

$$r(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} n_i^3(u) n_j^3(v) P_{ij}\,, \qquad u,v \in [0,1],$$

де $n_0^3(u) = \frac{1}{6}(1-u)^3$, $n_1^3(u) = \frac{1}{6}(3u^3 - 6u^2 + 4)$,

$n_2^3(u) = \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1)$, $n_3^3(u) = \frac{1}{6}u^3$, $u \in [0,1]$.

Змінюючи обидва параметри u, v від 0 до 1 можна визначити всі точки для фрагмента поверхні. Якщо один з параметрів зафіксувати, а інший змінювати від 0 до 1, то в результаті отримаємо кубічну B-сплайнову криву. Як і бікубічна поверхня Безьє, елементарна бікубічна B-сплайнова поверхня наслідує багато властивостей елементарної кубічної B-сплайнової кривої. Об'єднуючи елементарні бікубічні B-сплайни, одержуємо складену B-сплайнову поверхню, для якої на границях двох фрагментів автоматично забезпечується належність класу C^2.

Раціональна (елементарна) бікубічна B-сплайнова поверхня за заданим масивом точок P_{ij}, $i, j = 0, 1, 2, 3$ визначається рівнянням

$$r(u,v) = \frac{\sum_{i=0}^{3}\sum_{j=0}^{3} \omega_{ij} n_i^3(u) n_j^3(v) P_{ij}}{\sum_{i=0}^{3}\sum_{j=0}^{3} \omega_{ij} n_i^3(u) n_j^3(v)}, \qquad u,v \in [0,1], \qquad (8.4)$$

де $\omega_{ij} \geq 0$ $\left(\sum_{i=0}^{3}\sum_{j=0}^{3} \omega_{ij} > 0\right)$ – вагові коефіцієнти.

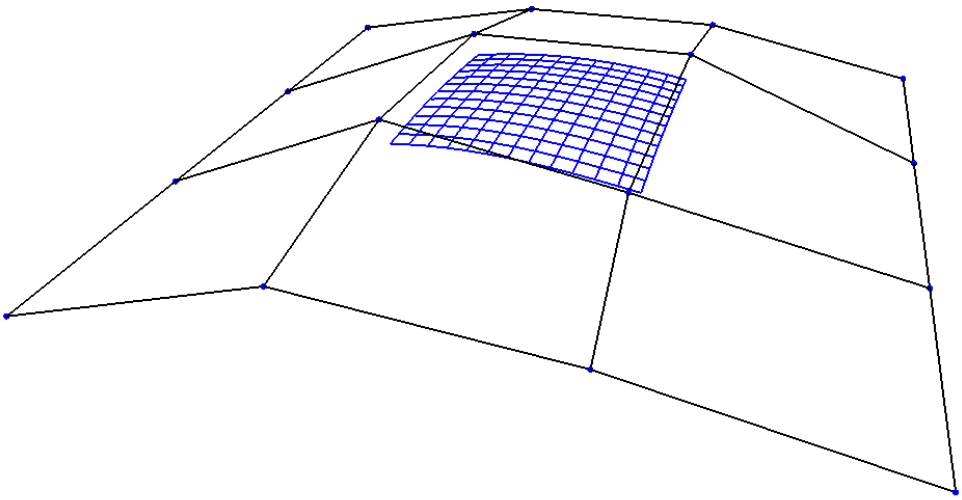


Рис. 8.4. Елементарна B-сплайнова поверхня

Раціональні B-сплайнові поверхні, які в англомовній 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 непроста 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

$$u = u_0, u_1, \ldots, u_n, v = v_0, v_1, \ldots, v_m$$

і організація вкладених циклів виведення каркасних ліній $r(u_i, v)$ й $r(u, v_j)$. Перетин ліній дає комірки трикутної або чотирикутної форми, які апроксимують поверхню.

Контрольні запитання 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. За яких умов на граничні контрольні точки складена поверхня Безьє належить класу C^1/ C^2?
10. Як визначається елементарна бікубічна B-сплайнова поверхня?
11. Які властивості має складена бікубічна B-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:

$$P_{00}(0,0,0)\ P_{01}(1,1,0)\ P_{02}(2,1,0)\ P_{03}(3,0,0)$$
$$P_{10}(0,1,1)\ P_{11}(1,2,1)\ P_{12}(2,2,1)\ P_{13}(3,1,1)$$
$$P_{20}(0,1,2)\ P_{21}(1,2,2)\ P_{22}(2,2,2)\ P_{23}(3,1,2)$$
$$P_{30}(0,0,3)\ P_{31}(1,1,3)\ P_{32}(2,1,3)\ P_{33}(3,0,3).$$

4. Розробити алгоритм обчислення вектора нормалі до поверхні Безьє в точці $P(u,v)$.
5. Записати параметричні рівняння елементарної бікубічної B-сплайнової поверхні в матричній формі.
6. Побудувати параметричне рівняння B-сплайнової поверхні за контрольними точками з прикладу 3.
7. Узагальнити рівняння кривих Ерміта на поверхні Ерміта.
8. Задати квадратичну поверхню в матричному вигляді. Як зміниться матриця квадратичної поверхні, якщо над нею здійснити афінні перетворення з матрицею F.

Розділ 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. Об'єктами вивчення обчислювальної геометрії є геометричні об'єкти: точка та відрізок прямої в двовимірному просторі та поліедри в тривимірному 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 $\boldsymbol{N} = (A, B)$. Вектор $\boldsymbol{V} = (-B, A)$ задає напрямний вектор прямої, бо $(\boldsymbol{N} \cdot \boldsymbol{V}) = 0$.

Оскільки $\boldsymbol{V} = \boldsymbol{N} \cdot R(90^0) = (A \;\; B) \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = (-B \;\; A)$, то це означає, що при русі точки по лінії в напрямку $\boldsymbol{V}$ вектор $\boldsymbol{N}$ направлений вправо від вектора $\boldsymbol{V}$ (вектор $\boldsymbol{V}$ – вліво від $\boldsymbol{N}$) (рис. 9.1). Тут $R(\varphi)$ є матрицею повороту простору проти годинникової стрілки на кут φ.

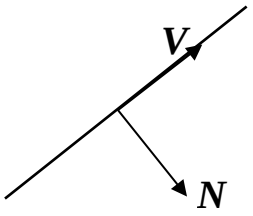


Рис. 9.1. Орієнтація нормалі

Іншими словами, якщо рухатися вздовж вектора (A, B), то вектор $(-B, A)$, направлений вліво, а вектор $(B, -A)$ – 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 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.

Параметричне рівняння прямої, що проходить через точку $p_0(x_0, y_0)$ в напрямку вектора $\boldsymbol{V}=(V_x, V_y)$, має вигляд

$$x = x_0 + V_x t, \quad y = y_0 + V_y t, \quad \text{або} \quad p = p_0 + Vt, \quad \text{де} \quad p = (x, y).$$

Завдяки правій орієнтації вектора $\boldsymbol{N} = (V_y, -V_x)$ відносно вектора $\boldsymbol{V}$ функція $f1(p) = ((p - p_0) \cdot N) = ((p - p_0) \cdot (V_y, -V_x)) = \begin{vmatrix} p - p_0 \\ V \end{vmatrix}$ дозволяє визначити положення точки відносно прямої, що проходить через точку p_0 уздовж вектора $\boldsymbol{V}$ (рис. 9.2):

- при $f1(p) > 0$ точка p лежить справа від прямої, тобто зі сторони нормалі (кут між векторами $p - p_0$ і $\boldsymbol{N}$ гострий);
- при $f1(q) < 0$ точка q лежить зліва (кут між векторами $q - p_0$ і $\boldsymbol{N}$ тупий).

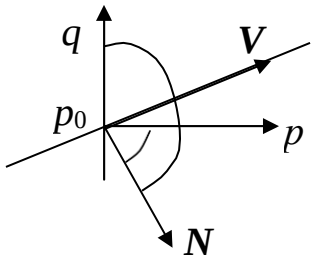


Рис. 9.2. Орієнтація точки

9.1.3. Тест напряму обходу трьох точок

Нехай задано три точки $a(a_x, a_y)$, $b(b_x, b_y)$, $p(x,y)$. Як видно з п. 9.1.2 функція

$$f2(p) = \begin{vmatrix} p-a \\ b-a \end{vmatrix} = \begin{vmatrix} x-a_x & y-a_y \\ b_x-a_x & b_y-a_y \end{vmatrix}$$

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:

$$f2(p) = -\begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ x & y & 1 \end{vmatrix}, \text{ оскільки } \begin{vmatrix} x-a_x & y-a_y \\ b_x-a_x & b_y-a_y \end{vmatrix} = -\begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ x & y & 1 \end{vmatrix}.$$

Отже, якщо задано три точки a, b, c, то визначник

$$D = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix}$$

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 або проти годинникової стрілки. Щоб реалізувати тест опуклості полігона, достатньо підрахувати значення визначників

$$D_i = \begin{vmatrix} x_{i-1} & y_{i-1} & 1 \\ x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \end{vmatrix}, \; i = 2,3,...,n-1, \; D_1 = \begin{vmatrix} x_n & y_n & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}, \; D_n = \begin{vmatrix} x_{n-1} & y_{n-1} & 1 \\ x_n & y_n & 1 \\ x_1 & y_1 & 1 \end{vmatrix}.$$

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 оболонок*. 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 $P = \{P_1\ P_2\ ...\ P_n\ P_1\}$ та точка $Q(x, y)$. Визначимо мінімальні та максимальні координати вершин P_i (i = 1, 2,…, 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), і якщо ці тіла не перетинаються, то й 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 $p = Q + \mathbf{V}t$, $t \geq 0$ 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 *а*, 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.

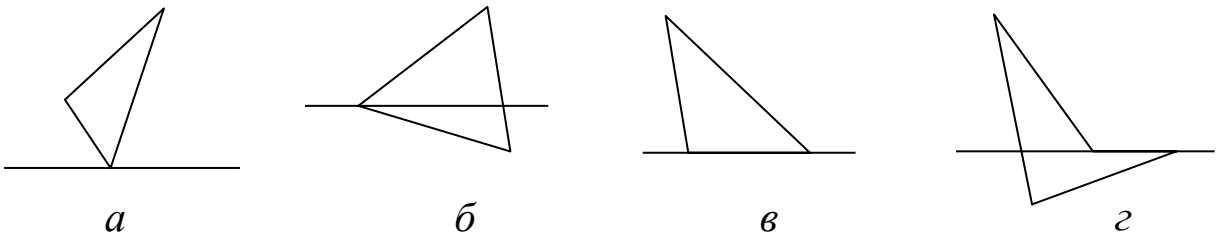


Рис. 9.3. Різні випадки перетину променя з полігоном

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, маємо 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 1. У випадках б, г, коли промінь зустрічається з горизонтальним ребром P_iP_{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 $\delta_i = \angle(\boldsymbol{V_i}, \boldsymbol{V_{i+1}})$ між суміжними векторами $\boldsymbol{V_i} = P_i - Q$, 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 $Q_в$, здійснює навколо себе повний оберт (рис. 9.4, *а*), а з зовнішньої точки $Q_з$ – жодного оберту (рис. 9.4, *б*).

Тоді при всіх ненульових векторах $\boldsymbol{V_i}$ радіанний варіант кутового тесту матиме вигляд

$$\left|\sum_{i=1}^{n} ang(V_i, V_{i+1})\right| = \begin{cases} 0 < \pi, Q_з \notin P, \\ 2\pi > \pi, Q_в \in P, \end{cases} \tag{9.1}$$

де $ang(\boldsymbol{V}, \boldsymbol{W})$ – це функція обчислення кута між векторами $\boldsymbol{V}$ і $\boldsymbol{W}$, як кута найкоротшого повороту від $\boldsymbol{V}$ до $\boldsymbol{W}$, тобто

$$ang(V,W) = if\left(d = \begin{vmatrix} V \\ W \end{vmatrix} \neq 0 : sign(d); 1\right) \arccos\left(\frac{V \cdot W}{|V| \cdot |W|}\right).$$

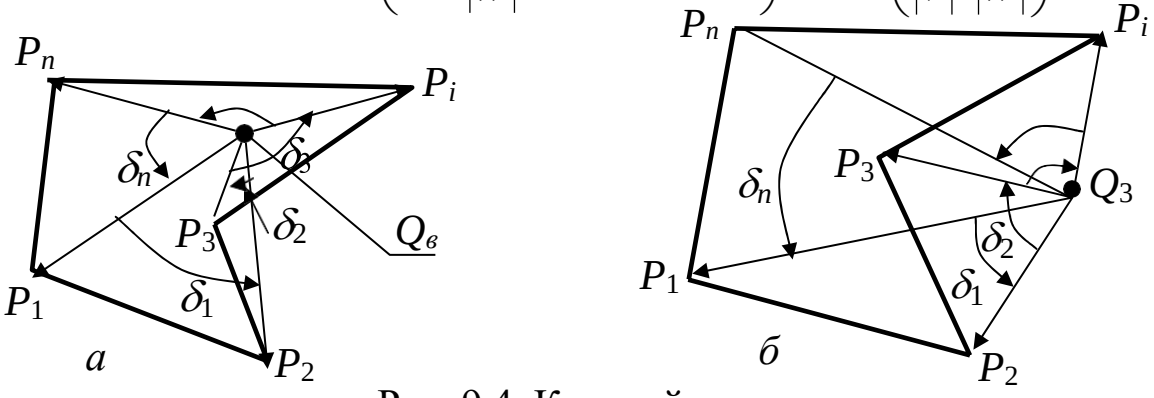


Рис. 9.4. Кутовий тест

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. 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_iP_{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 $p_1p_2p_3$. Для цього достатньо записати розклад вектора qp_1 за векторами сторін трикутника, тобто

$$q - p_1 = \boldsymbol{V}t + \boldsymbol{W}\tau, \tag{9.2}$$

де $\boldsymbol{V} = p_2 - p_1$, $\boldsymbol{W} = p_3 - p_1$.

Далі систему (9.2) необхідно розв'язати відносно τ, t або скористатися методом Грамма і скласти нову систему двох лінійних рівнянь зі скалярних добутків

$$(q - p_1)\boldsymbol{V} = t(\boldsymbol{V} \cdot \boldsymbol{V}) + \tau(\boldsymbol{W} \cdot \boldsymbol{V}),$$
$$(q - p_1)\boldsymbol{W} = t(\boldsymbol{V} \cdot \boldsymbol{W}) + \tau(\boldsymbol{W} \cdot \boldsymbol{W})$$

та знайти розв'язок

$$(t, \tau) = ((q - p_1)V\ (q - p_1)W)\begin{bmatrix}(V \cdot V) & (V \cdot W)\\ (V \cdot W) & (W \cdot W)\end{bmatrix}^{-1}. \tag{9.3}$$

Тоді при $\{t > 0\} \bigcap \{\tau > 0\} \bigcap \{t + \tau < 1\}$ точка лежить усередині трикутника, при $\{t < 0\} \bigcup \{\tau < 0\} \bigcup \{t + \tau > 1\}$ – зовні трикутника, а в решті випадків – на його границі.

Матриця, що фігурує в (9.3), невироджена, оскільки визначник

$$|\boldsymbol{V}|^2 \cdot |\boldsymbol{W}|^2 - (\boldsymbol{V} \cdot \boldsymbol{W})^2 = |\boldsymbol{V}|^2 \cdot |\boldsymbol{W}|^2 \sin^2(\angle(\boldsymbol{V}, \boldsymbol{W})) > 0$$

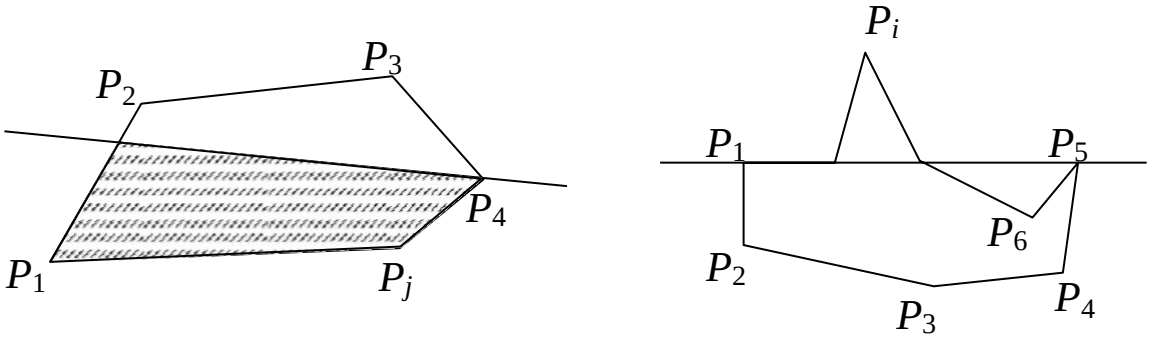
і дорівнює нулю лише при колінеарності вершин трикутника $p_1p_2p_3$.

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.



а *б*

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 $i = 1, 2, \ldots, n$ обчислюємо значення $s_i = f(p_i)$. У внутрішньому циклі $j = 1, 2, \ldots, i - 1$, $i > 1$ 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 $i = 1, 2, \ldots, n$ 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 $i = 1, 2, \ldots, n$ 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 із цим полігоном. Блок-схему цього алгоритму подано на рис. 9.6.

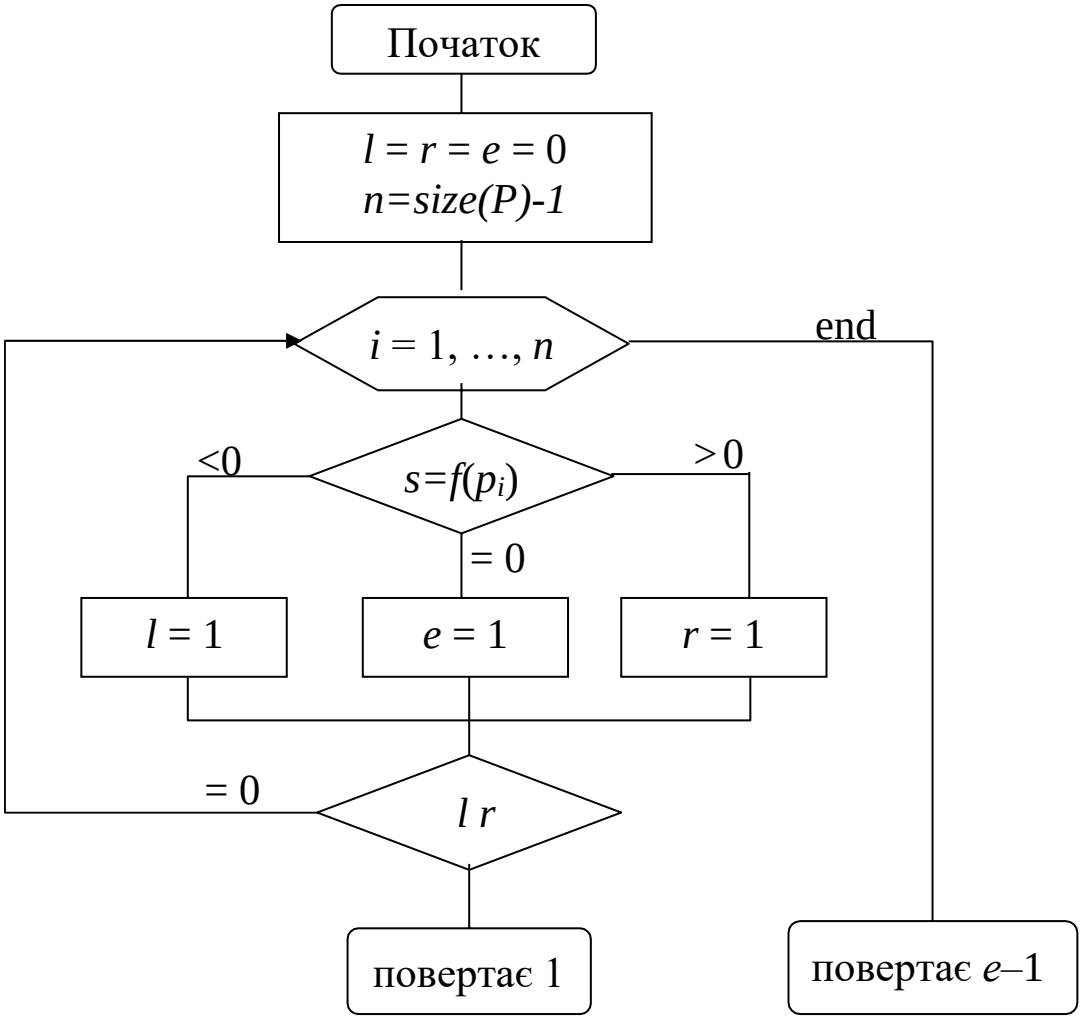


Рис. 9.6. Блок-схема тесту перетину прямої з полігоном

9.4.2. Тест перетину відрізків

Нехай задано відрізки [*a*, *b*] та [*c*, *d*] параметричними рівняннями

$$x = x_a + (x_b - x_a)t, \qquad x = x_c + (x_d - x_c)\tau,$$
$$y = y_a + (y_b - y_a)t, \quad t \in [0, 1]; \qquad y = y_c + (y_d - y_c)\tau, \quad \tau \in [0, 1].$$

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

$$(t \quad \tau) = (x_c - x_a \quad y_c - y_a)\,M^{-1}, \text{ де } M = \begin{pmatrix} x_b - x_a & x_c - x_d \\ y_b - y_a & y_c - y_d \end{pmatrix}.$$

Якщо $|M| = 0$, то відрізки паралельні. У випадку непаралельності відрізків умови $0 \le t \le 1$, $0 \le \tau \le 1$ 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:

cross_segm(*ab*, *cd*, *t*, *τ*, *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 $(0 \le t \le 1) \bigcap (0 \le \tau \le 1)$ // 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***.

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_segm 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).

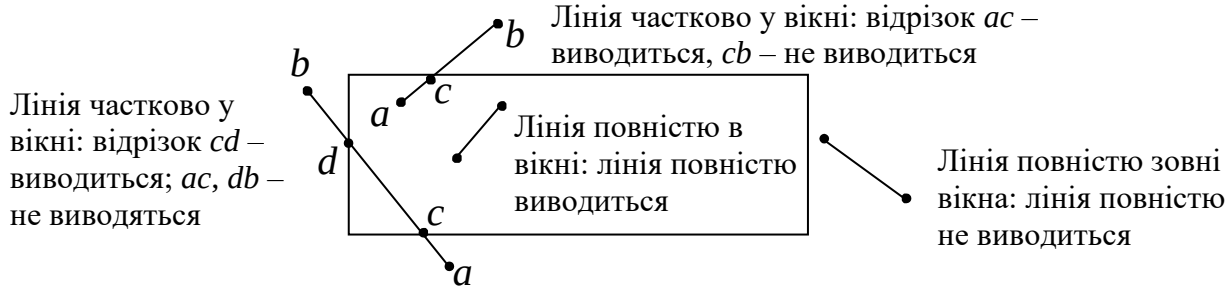


Рис. 9.7. Двовимірне відсікання прямокутним вікном

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 свій код (рис. 9.8). Ці коди використовуються для визначення того чи знаходиться відрізок повністю у вікні чи зовні вікна.

Отже, маємо такий алгоритм відсікання відрізків.

- Формуємо коди (*код*1 та *код*2) для кінців відрізка.
- Якщо два коди для кінців відрізка дорівнюють 0000, то відрізок повністю знаходиться у вікні.
- Відрізок цілком відсікається, якщо *код*1 and *код*2 ≠ 0000.

0011	0010	0110
0001	**0000**	0100
1001	1000	1100

Рис. 9.8. Чотирибітні коди

- Якщо *код*1 and *код*2 = 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

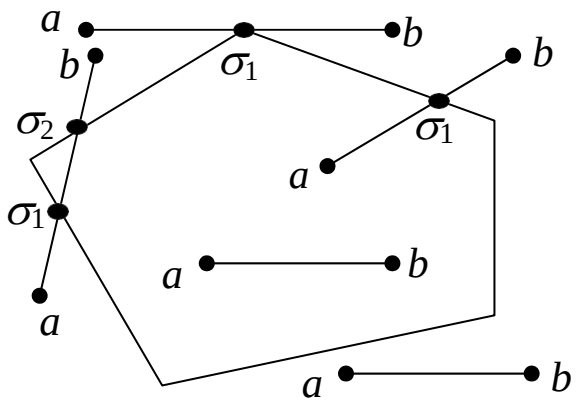


Рис. 9.9. Відсікання полігоном

Розглянемо задачу зовнішнього відсікання відрізка ab опуклим полігоном $P = \{P_1\, P_2\, \ldots, P_n\, P_1\}$.

Необхідно відсікти ті частини відрізка, які знаходяться зовні полігона (рис. 9.9).

Запропонований алгоритм базується на розрахунках перетину прямої ab із відрізками P_iP_{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 набуває 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 $i = 1, 2,\ldots, n$ ($P_{n+1}=P_1$) 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 $0 \le \tau \le 1$), 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 $\{\sigma_1 < 0, \sigma_2 > 1\}$ або $\{\sigma_1 > 1, \sigma_2 < 0\}$ 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 $\{0 \le \sigma_1 \le 1,\ \sigma_2 < 0\}$ повертають відрізок $[a,\ a + V\sigma_1]$;
- значення $\{0 \le \sigma_1 \le 1,\ \sigma_2 > 1\}$ повертають відрізок $[a + V\sigma_1,\ b]$;
- значення $\{\sigma_1 < 0,\ 0 \le \sigma_2 \le 1\}$ повертають відрізок $[a,\ a + V\sigma_2]$;
- значення $\{\sigma_1 > 1,\ 0 \le \sigma_2 \le 1\}$ повертають відрізок $[a + V\sigma_2,\ b]$;
- значення $\{0 \le \sigma_1 \le 1,\ 0 \le \sigma_2 \le 1\}$ 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 зображено на рис. 9.10.

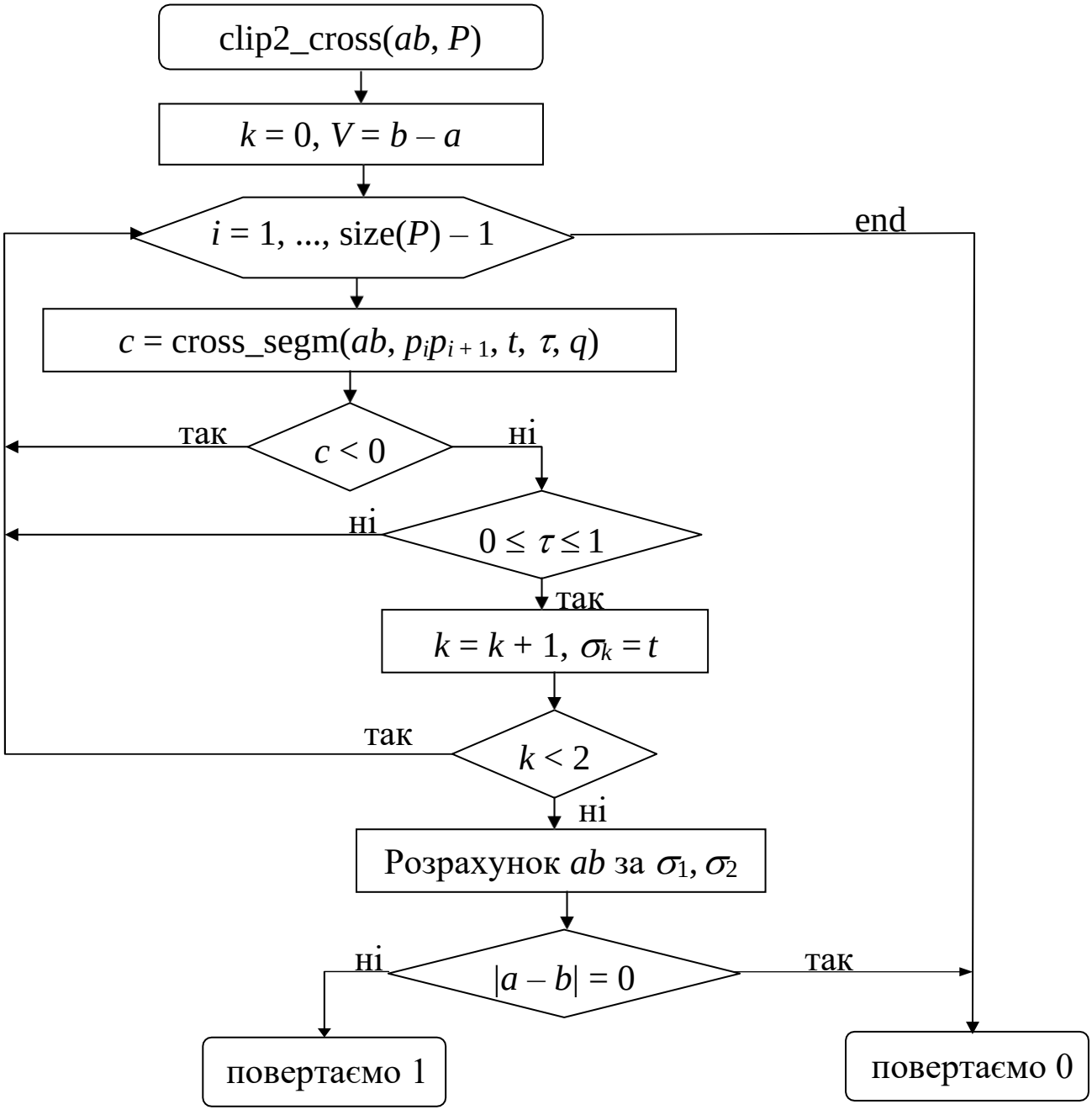


Рис. 9.10. Блок-схема алгоритму відсікання

9.5.3. Перетин та об’єднання опуклих полігонів

Розглянемо задачу побудови перетину опуклих полігонів $A = \{A_1\ A_2\ \ldots\ A_n\ A_1\}$ та $B = \{B_1\ B_2\ \ldots\ B_m\ B_1\}$. Необхідно знайти спільну частину полігонів, що належить полігонам A та B, тобто їхній логічний добуток $A \bigcap B$.

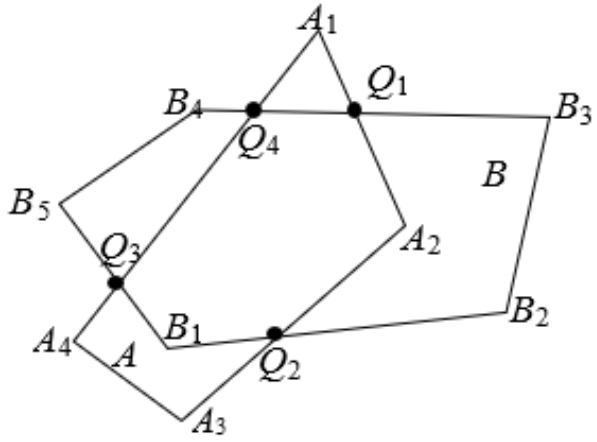


Рис. 9.11. Перетин полігонів

Алгоритм знаходження перетину полігонів пояснимо на прикладі опуклих полігонів, які зображені на рис. 9.11.

- Спочатку функцією clip2 cross (A_iA_{i+1}, B), $i = 1, 2, \ldots, n$ виконуємо зовнішнє відсікання ребер полігона A полігоном B. У результаті одержуємо список відрізків $Ls = \{Q_1A_2,\ A_2Q_2,\ Q_3Q_4\}$.
- Далі функцією clip2_cross(B_jB_{j+1}, A), $j = 1, 2, \ldots, m$ виконуємо зовнішнє відсікання ребер полігона B полігоном A і додаємо в список Ls нові відрізки B_1Q_2, Q_1Q_4, Q_3B_1.
- З елементів одержаного списку

$$Ls = \{Q_1A_2,\ A_2Q_2,\ Q_3Q_4,\ B_1Q_2,\ Q_1Q_4,\ Q_3B_1\}$$

збираємо замкнутий контур шляхом з’єднання відрізків збіжними кінцями. Для цього беремо перший відрізок Q_1A_2 і серед інших елементів списку Ls шукаємо відрізок, кінець якого збігається з A_2. Це відрізок Q_2A_2, який разом із відрізком Q_1A_2 утворює ламану $C = \{Q_1A_2Q_2\}$. Далі знаходимо продовження ламаної B_1Q_2 і одержуємо ламану $C = \{Q_1A_2Q_2B_1\}$ і т. д., поки не одержимо замкнений контур $C = \{Q_1A_2Q_2B_1Q_3Q_4Q_1\}$. Якщо полігони не перетинаються, то після виконання зовнішнього відсікання список Ls буде порожнім.

9.6. Інші алгоритми відсікання відрізків

У багатьох випадках більшість відрізків лежить цілком усередині 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:

– 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, Вейлера–Азертона 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 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 так

$$LineCode(ab) = Code(a)* 10 + Code(b),$$

де $Code(a)$ – 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 варіанта, 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, 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 – на рис. 9.14.

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.

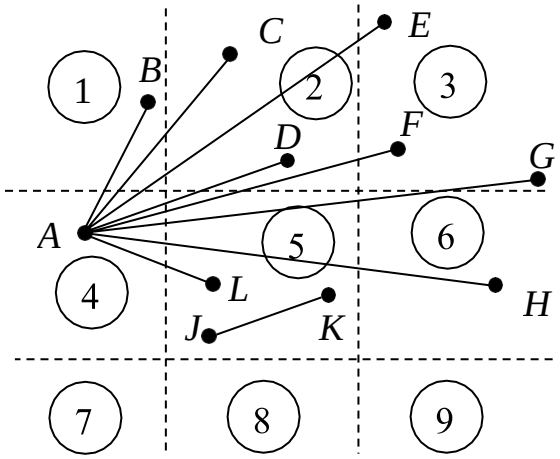


Рис. 9.13. Варіанти розміщення відрізка для некутових областей

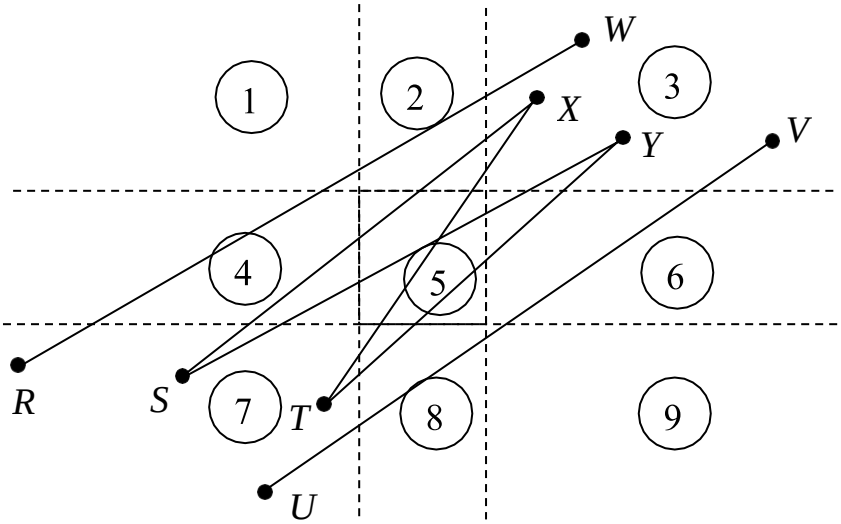


Рис. 9.14. Варіанти розміщення відрізка для кутових областей

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 обчислюємо точку перетину відрізка *AD* з верхньою межею вікна й зображаємо 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 й 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*, *TY*, *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. Обчисляємо 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 *SY*. Далі, якщо $y_{n2} < y_{верхн}$, то це відрізок *SY* 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*, *TY*. Якщо $y_{n2} < y_{верхн}$, то це відрізок *TY* і ми можемо зобразити його видиму частину. Інакше це відрізок *TX*, тому обчисляємо для нього точку перетину з верхньою стороною вікна і теж відображаємо його видиму частину.

Зауваження 2. 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 багатокутником, 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 багатокутника, а також на методі визначення місцезнаходження точки відрізка відносно вікна – всередині, на границі або зовні вікна. Для цього в алгоритмі використовується вектор внутрішньої нормалі до ребра вікна.

Розглянемо опуклу двовимірну область P, тобто опуклий багатокутник. Внутрішньою нормаллю $\boldsymbol{n}$ до сторони вікна називається нормаль, яка направлена в бік області, що складає вікно відсікання. Як видно з рис. 9.15, внутрішня нормаль $\boldsymbol{n}$ у точці a, що лежить на границі, задовольняє умову

$$(\boldsymbol{n} \cdot (b - a)) \geq 0, \qquad (9.4)$$

де b – будь-яка інша точка границі опуклої області P.

Скалярний добуток в (9.4) для внутрішньої нормалі $\boldsymbol{n}$ невід'ємний тому, що $\theta \in \left[0, \frac{\pi}{2}\right]$, тобто $\cos\theta \geq 0$.

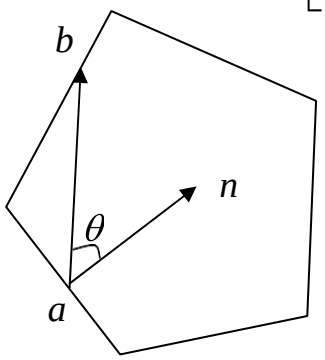
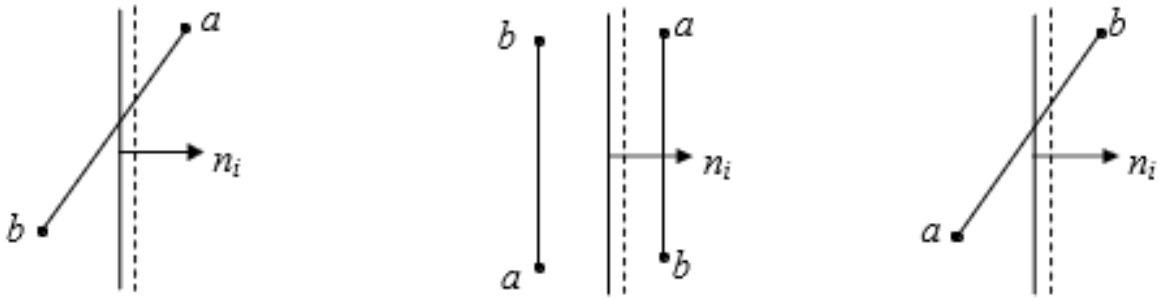


Рис. 9.15. Визначення місцезнаходження точки відносно вікна відсікання в алгоритмі Кіруса–Бека

Зазначимо, що внутрішні нормалі можна обчисляти й іншим способом. Для цього необхідно полігон обходити за годинниковою стрілкою і вибирати нормалі, які направлені вправо від кожного ребра (див. п. 9.1.1).

Нехай $\boldsymbol{n_i}$ – внутрішня нормаль до i-ї граничної лінії багатокутного вікна, а $\boldsymbol{V} = b - a$ – вектор, що визначає орієнтацію відрізка ab, який відсікається вікном P. Тоді орієнтація відрізка ab відносно i-ї сторони вікна визначається знаком скалярного добутку $r_i = (\boldsymbol{n_i} \cdot \boldsymbol{V})$:

– при $r_i < 0$ відрізок ab направлений з внутрішньої сторони на зовніш- ню сторону i-ї граничної лінії вікна (рис. 9.16, *а*).
– 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*).



а *б* *в*

Рис. 9.16. Орієнтація відрізка *ab* відносно сторони вікна:
а – 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 $\boldsymbol{n_i}$ на вектор $\boldsymbol{Q_i} = V(t) - P_i$, що починається в початковій точці P_i ребра вікна і закінчується в деякій точці $V(t)$ відрізка *ab*, а саме

$$q_i = (\boldsymbol{n_i} \cdot \boldsymbol{Q_i}),\ i = 1, 2, \ldots \qquad (9.5)$$

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.

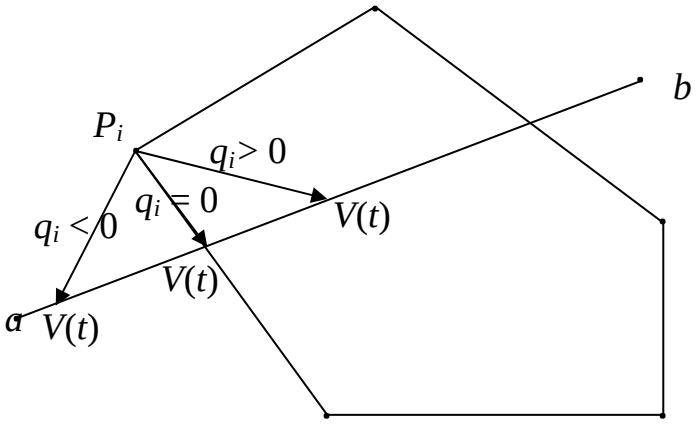


Рис. 9.17. Розміщення точки відносно границі вікна

Згадаємо векторно-параметричне рівняння відрізка *ab*

$$V(t) = a + (b - a)\,t,\ t \in [0, 1]. \qquad (9.6)$$

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

$$(\boldsymbol{n_i} \cdot (a + (b - a)t - P_i))) = 0,$$

або

$$(\boldsymbol{n_i} \cdot (a - P_i)) + (\boldsymbol{n_i} \cdot (b - a)t) = 0.$$

Звідки

$$t = -\frac{(n_i \cdot (a - P_i))}{(n_i \cdot (b - a))} = -\frac{(n_i \cdot Q_i)}{(n_i \cdot V)} = -\frac{q_i}{r_i}. \qquad (9.7)$$

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

$$r_i = (\boldsymbol{n_i} \cdot (b - a)).$$

Якщо $r_i = 0$, то відрізок *ab* або вироджується в точку, або паралельний *i*-й стороні вікна. При цьому проаналізуємо значення $q_i = (\boldsymbol{n_i} \cdot (a - P_i))$. 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*. У цьому випадку визначається min *t* серед всіх одержаних *t* (рис. 9.18). Воно дає значення t_1 для кінцевої точки шуканого відрізка

$$t_1 = \min\left(\left\{-\frac{q_i}{r_i}\middle| r_i < 0, i = 1, 2, \ldots\right\}, 1\right).$$

Якщо поточне значення t_1 стане меншим за t_0, товідрізок *ab* відкидається, оскільки не виконується умова $t_0 \le t_1$.

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

$$t_1 = \max\left(\left\{-\frac{q_i}{r_i}\middle| r_i > 0, i = 1, 2, \ldots\right\}, 0\right).$$

Якщо поточне значення t_0 стане більшим за t_1, то відрізок відсікається.

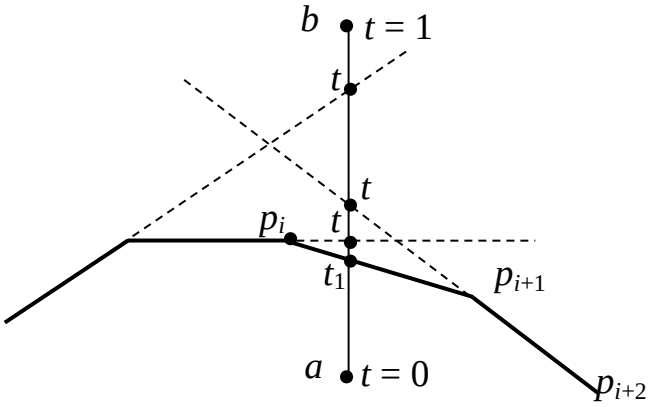


Рис. 9.18. Визначення кінцевої точки $V(t_1)$

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 зайві.

9.6.3. Відсікання полігонів. Алгоритм Вейлера–Азертона

Багатокутник 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 багатокутників. До таких алгоритмів належать алгоритм Сазерленда–Ходжмена для відсікання опуклих багатокутників та більш сильний алгоритм Вейлера–Азертона для відсікання довільних областей.

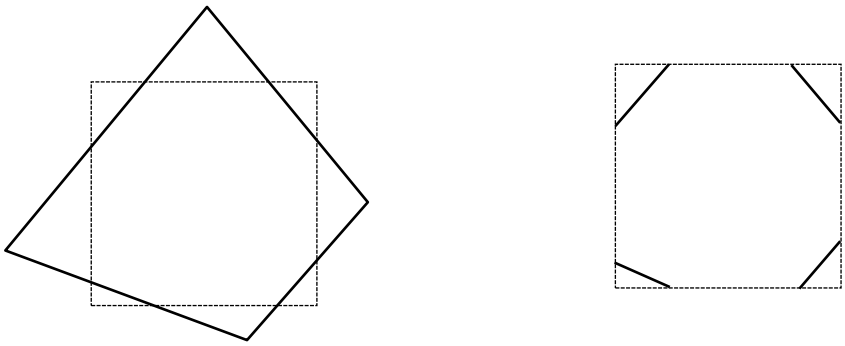


Рис. 9.19. Відсікання вікном ребер багатокутника

На відміну від інших алгоритмів відсікання, алгоритм Вейлера–Азертона дозволяє відсікати довільний неопуклий багатокутник з дірками іншим неопуклим багатокутником з дірками, причому можливе як зовнішнє, так і внутрішнє відсікання. Розглянемо його.

Кожний із багатокутників у цьому алгоритмі задається циклічним списком вершин, причому вершини зовнішньої границі кожного багатокутника обходяться за годинниковою стрілкою, а внутрішня границя (дірки) – проти годинникової стрілки так, щоб внутрішня область багатокутників завжди залишалась справа від границі.

Границі багатокутника, що відсікається, і вікна можуть перетинатися або не перетинатися. Якщо вони перетинаються, то виникають точки перетину ребер двох типів:

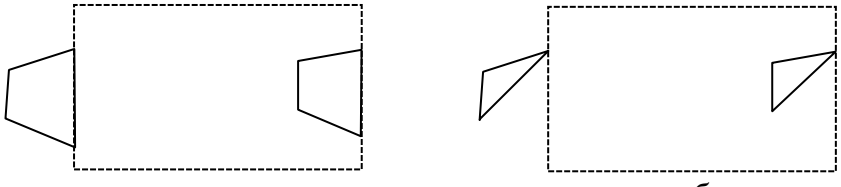
– вхідні точки, коли орієнтовані ребра багатокутника, що відсікається, входять у вікно;
– вихідні точки, коли ребра багатокутника, що відсікається, виходять із внутрішньої сторони вікна на зовнішню.

Точки перетину заносяться в циклічні списки вершин двох многокутників так, щоб не порушити правила обходу границі багатокутників. Тобто якщо точка перетину I_k знаходиться на ребрі, що з'єднує вершини P_i, P_{i+1}, то послідовність P_iP_{i+1} перетворюється в послідовність $P_iI_kP_{i+1}$.

Алгоритм починається з точки перетину вхідного типу. Далі він простежує модифікований список вершин (границю багатокутника) 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 алгоритму Вейлера–Азертона має такий вигляд:

– побудувати циклічні списки вершин багатокутника, що відсікається та вікна, згідно з наведеним вище правилом обходу;
– відшукати всі точки перетину вікна і багатокутника (при цих обчисленнях дотикання не вважається перетином (рис. 9.20, 9.21));
– додати ці точки перетину до циклічних списків вершин багатокутників. При цьому встановлюються двосторонні зв'язки між однойменними точками перетину в двох списках;



а *б*

Рис. 9.20. Випадки дотикання (не вважається перетином): *а* – дотикання ребром; *б* – дотикання вершиною

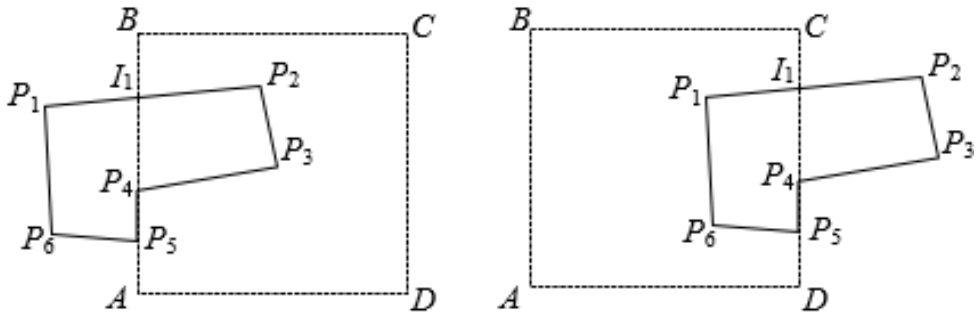


Рис. 9.21. Окремі випадки перетину

– 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.

Пошук 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 $P_1P_2P_3P_3P_4P_5$ (рис. 9.22).

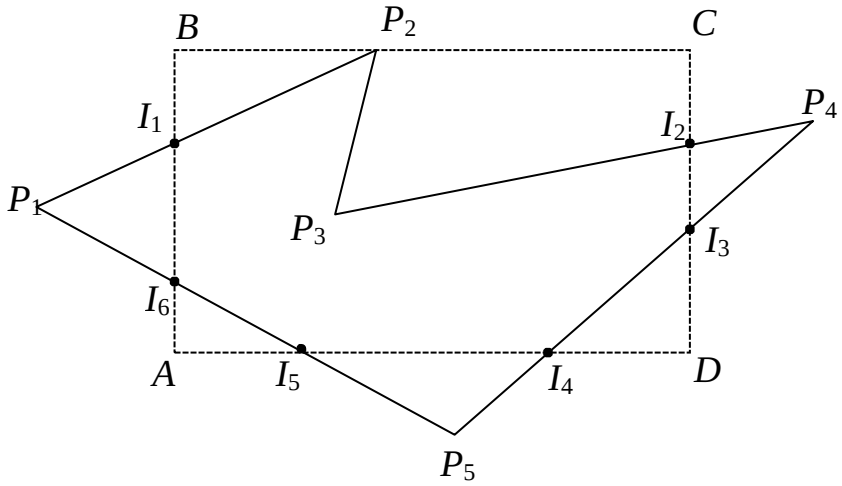
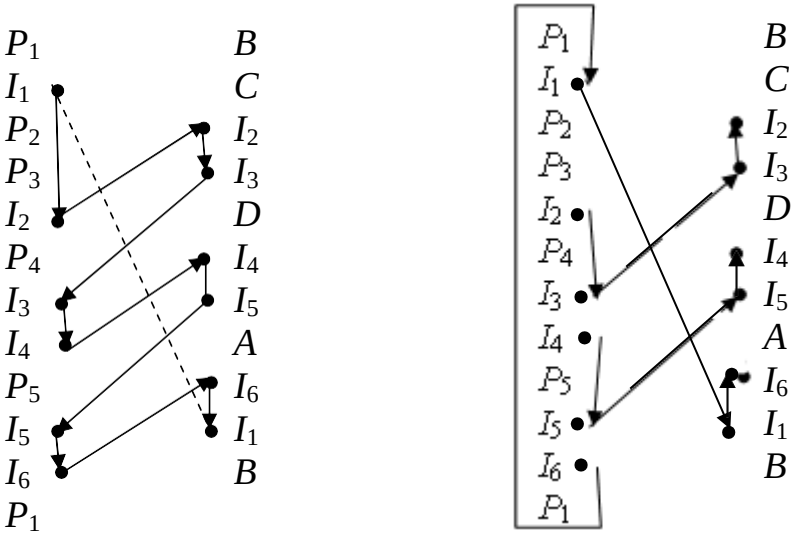


Рис. 9.22. Відсікання багатокутника вікном

Для внутрішнього відсікання циклічні списки вершин і порядок роботи алгоритму зображені на рис. 9.23, *а*. Як вхідна вибрана точка I_1. Пройдені точки формують результат $I_1P_2P_3I_2I_3I_4I_5I_6I_1$.



а *б*

Рис. 9.23. Схема алгоритму Вейлера–Азертона: *а* – внутрішнє відсікання; *б* – зовнішнє відсікання

Для виконання зовнішнього відсікання (рис. 9.23, *б*) як початкову беремо вихідну точку перетину, наприклад I_2. Далі діємо згідно з алгоритмом, проходячи вершини вікна знизу вверх.

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 $P = \{P_1, P_2, \ldots, P_n\}$ з координатами $P_i(x_i, y_i)$, $i = 1, 2, \ldots, n$, $n \geq 2$.

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 кілька, 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 $\boldsymbol{V}$ уздовж осі x та індекс вершини оболонки $k = 1$. Далі організовуємо зовнішній цикл по k для 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 $\boldsymbol{W} = P_i - P_k$. При ненульовій довжині цього вектора $r = |\boldsymbol{W}| \neq 0$ (така перевірка дозволяє відкинути крайні точки) знаходимо

$$c = \cos\varphi = \frac{\boldsymbol{V}\cdot\boldsymbol{W}}{|\boldsymbol{V}|\cdot|\boldsymbol{W}|}.$$

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 кілька, 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 $\boldsymbol{V} = P_k - P_{k-1}$.

Зовнішній цикл і побудова оболонки закінчується при збігові точки P_k із початковою точкою P_1.

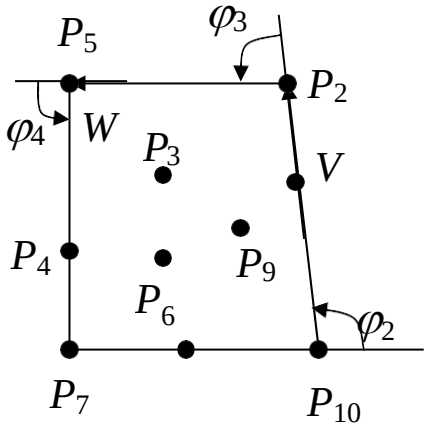


Рис. 9.24. Опукла оболонка

Алгоритм повертає субмасив вершин оболонки $\{P_1, P_2, \ldots, P_k\}$. Робота алгоритму зображена на рис. 9.24. Цей алгоритм називають алгоритмом загортання подарунка. 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 утворять шукану опуклу оболонку. Попереднє сортування точок спрощує побудову опуклої оболонки.

Для вирішення питання, 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 $P_{i-1}\,P_i\,P_{i+1}$ не відповідає повороту вліво, то точка P_i вилучається з опуклої оболонки, і 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, *д*). Оскільки обхід $P_4P_5P_6$ не лівий, то P_5 вилучається з поточної оболонки. Далі перевіряємо обхід $P_3P_4P_6$ – він теж не лівий. 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 $O(nlgn)$ арифметичних операцій.

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 опуклу оболонку.

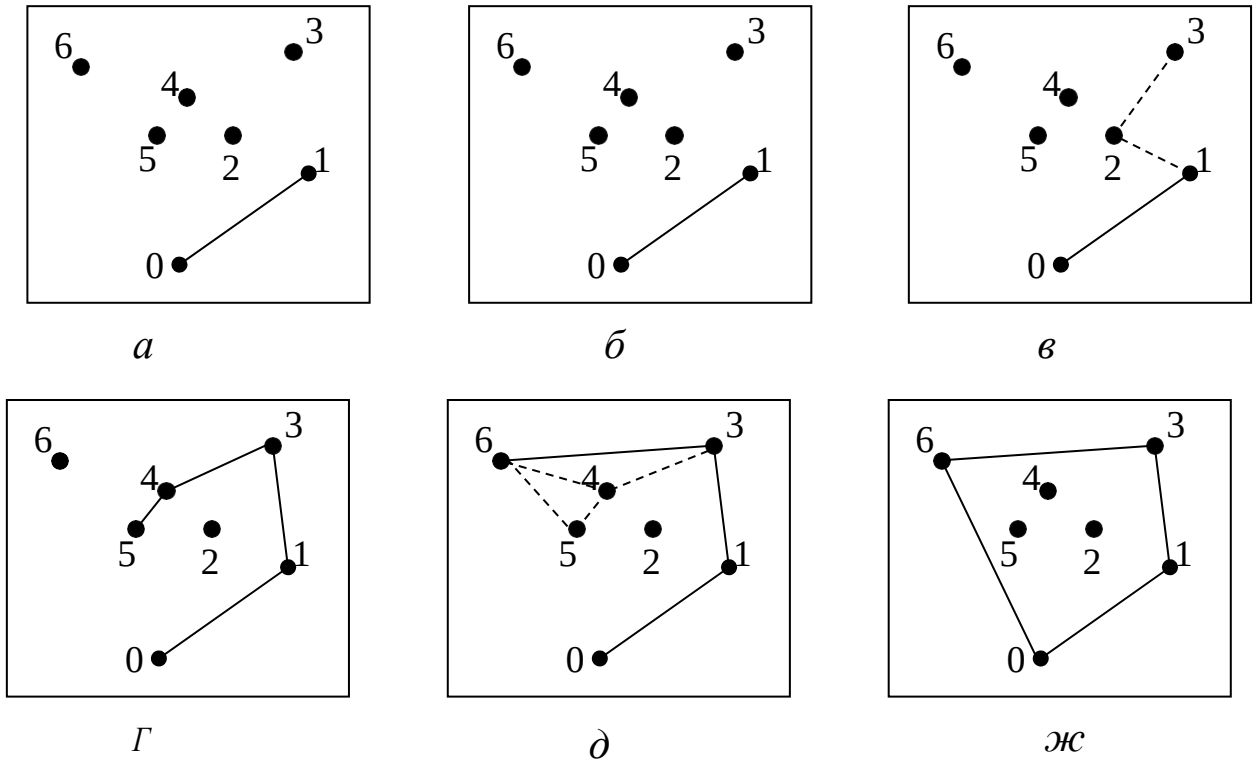


Рис. 9.25. Робота алгоритму обходу Грехема (точки P_i, $i = 0, 1,\ldots, 6$ вже заздалегідь відсортовані за полярним кутом)

Зауваження 4. В алгоритмі Грехема використовуються тригонометричні функції, а їх обчислення вимагають великих обчислювальних затрат в часі. Тому для швидкодіючих 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 цю 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 й називаються динамічними.

9.8. Тріангуляція полігонів і точкових множин

Широко розповсюдженою задачею обчислювальної геометрії є задача тріангуляції полігонів і множини точок. ***Тріангуляцією полігонів*** 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 фігурою і завжди опуклим, а в цьому випадку задачі розв'язуються значно простіше. Наприклад, для визначення належності точки неопуклому полігонові 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 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.

Тріангуляція множини точок P на площині – це розбиття площини на трикутники, яке визначається максимальною кількістю неперетинних ребер, множина вершин яких належить P. Для точок множини P ребром називають відрізок, який має дві точки множини P як кінцеві.

Будь-який набір точок P, за винятком деяких тривіальних випадків, допускає більше одного способу тріангуляції, при цьому одержується одна і та ж сама кількість трикутників. За теоремою Ейлера, кількість трикутників не перевищує $2n$, де n – кількість точок масиву P.

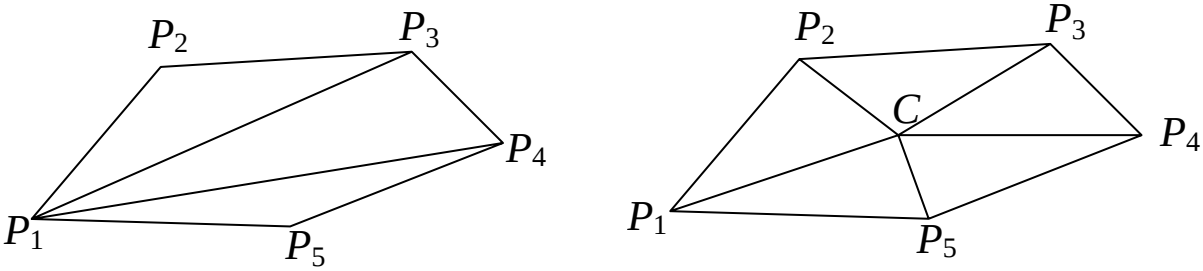
Унаслідок великої кількості можливих тріангуляцій для однієї і тієї самої точкової множини існує багато критеріїв, за якими може бути здійснена тріангуляція. Серед важливих видів тріангуляції виділяють:

- тріангуляцію Делоне, в якій для будь-якого трикутника тріангуляції всі точки множини P, за винятком вершин, лежать поза колом, описаним навколо цього трикутника.
- оптимальну зважену тріангуляцію, для якої сумарна довжина всіх ребер мінімальна серед усіх можливих тріангуляцій, побудованих на тих самих даних та ін.

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, …, 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.



а *б*

Рис. 9.26. Тріангуляція опуклих полігонів

Опуклий полігон можна розрізати на n трикутників ще й в інший спосіб (рис. 9.26, *б*):

$$CP_1P_2 \cup CP_2P_3 \cup \ldots \cup CP_nP_1,$$

ввівши додаткову вершиною C, яку можна обчислити як геометричний центр полігона

$$C = \frac{1}{n}\sum_{i=1}^{n} P_i.$$

У 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 вирішити, 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.

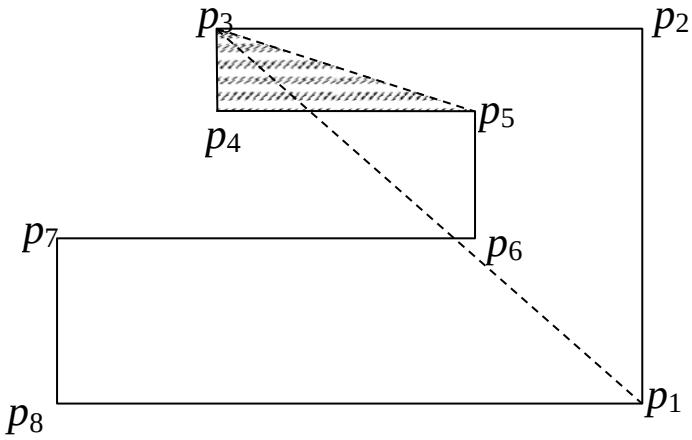


Рис. 9.27. Тріангуляція неопуклого полігона

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 самі операції, що й для вихідного полігона і т. д., поки полігон, який залишається, не стане трикутником.

Зауваження 6. Задачу тріангуляції 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. Тріангуляція точкової множини

При тріангуляції всі точки набору P повинні лежати у вершинах трикутників. Ребра трикутників не повинні перетинатися. Усі точки набору P 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 тріангуляції, – ребра оболонки і внутрішні ребра.

Розглянемо кілька алгоритмів тріангуляції множини точок у простішому випадку, коли жодні три точки не лежать на одній прямій. Спочатку здійснимо тріангуляцію через побудову опуклої оболонки (рис. 9.28). Таку тріангуляцію можна здійснити за допомогою таких кроків.

1. Знаходимо опуклу оболонку множини P.
2. Проводимо тріангуляцію опуклого багатокутника (внутрішні точки поки що ігноруємо).

3. Унаслідок припущення про розміщення точок, кожна внутрішня точка знаходиться всередині трикутника, а не на його ребрах. Тому обираємо внутрішню точку і проводимо з неї ребра до вершин трикутника, в якому вона лежить
4. Повторюємо крок 3 для кожної внутрішньої точки.

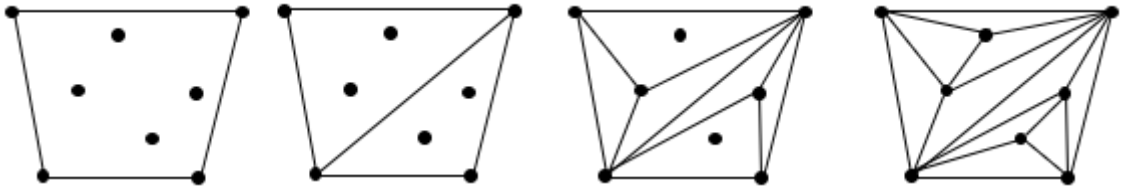


Рис. 9.28. Тріангуляція через опуклу оболонку

Іншим прикладом тріангуляції є інкрементний алгоритм тріангуляції (рис. 9.29).

1. Відсортуємо точки множини P за координатою x і помістимо його в список S.
2. Перші три точки в S формують трикутник тріангуляції.
3. Розглянемо наступну точку p є S і з'єднаємо її з усіма попередніми точками, які видимі з точки p.
4. Повторюємо крок 3 для кожної точки множини S.

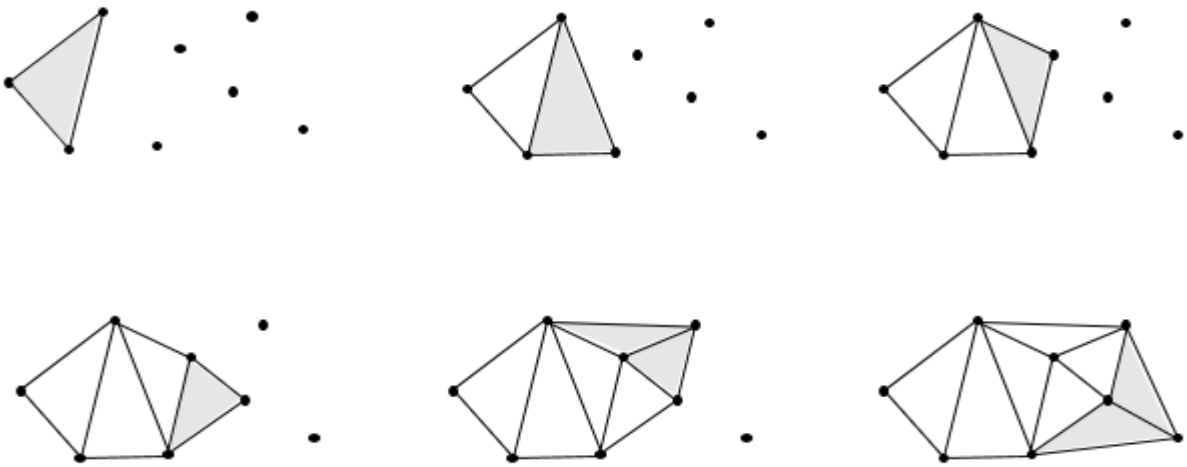


Рис 9.29. Інкрементна тріангуляція

Наведемо ще алгоритм жадібної тріангуляції. Нагадаємо, що алгоритм називається жадібним, якщо на кожному кроці не відмінюється зроблене на попередніх кроках.

1. Формуємо список з $n(n\text{-}1)/2$ ребер, тобто всіх можливих відрізків, що з'єднують пари вихідних точок. Отриманий список упорядковуємо за довжиною відрізків.

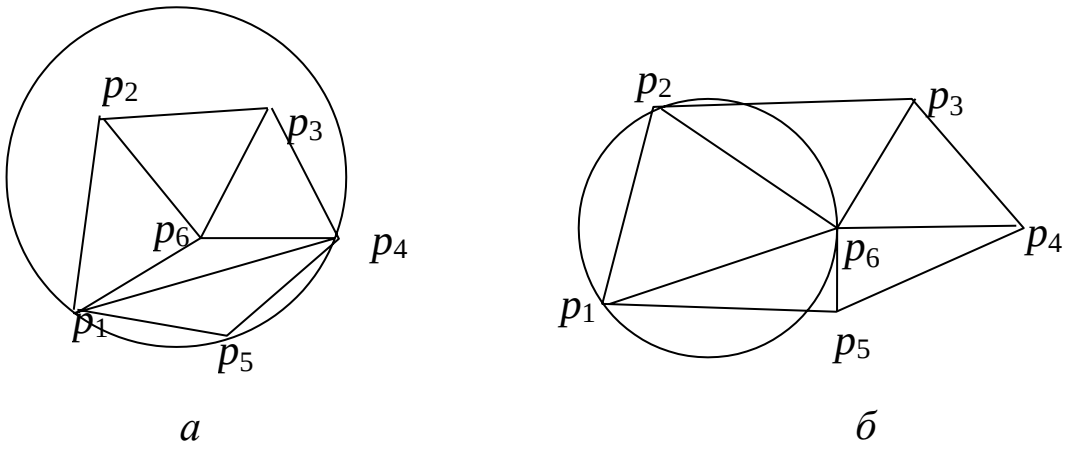
2. Виконується вставка ребер у тріангуляцію від найкоротшого до найдовшого. Якщо це ребро не перетинає жодне з ребер, які ввійшли до тріангуляції, то воно включається до тріангуляції, інакше ребро виключається з подальшого розгляду. Тріангуляція закінчується, коли всі ребра множини S розглянуті, або тріангуляція вже побудована.

9.8.4.Тріангуляція Делоне

Серед усіх можливих тріангуляцій виділяється спеціальний вид – *тріангуляція Делоне* [21]. Розглянемо задачу тріангуляції Делоне набору точок P на площині. Задача побудови тріангуляції Делоне є однією з базових задач обчислювальної геометрії, це найвідоміша тріангуляція множин точок. Вона широко використовується в комп'ютерній графіці для моделювання поверхонь.

Ця тріангуляція характеризується тим, що утворені трикутники наближаються до максимально правильних, тобто тріангуляція Делоне максимізує мінімальний кут, а не довжину ребра трикутника.

Визначення. *Тріангуляція набору точок P називається тріангуляцією Делоне, якщо всередині кола, описаного навколо кожного трикутника, немає точок набору P, окрім їх вершин, тобто всі точки множини P лежать поза колом*.



а *б*

Рис. 9.30. Тріангуляція набору точок

Тріангуляцію, що зображена на рис. 9.30, *а*, не можна віднести до тріангуляції Делоне, а тріангуляція на рис. 9.30, *б* є тріангуляцією Делоне.

Для заданої множини точок, в якій жодні чотири точки не знаходяться на одному колі існує рівно одна триангуляція Делоне.

Використовуючи означення тріангуляції Делоне, можна розробити алгоритм її побудови, час роботи якого прямо пропорційний

кількості точок масиву *S*, і виконана побудова однозначна [4; 21; 30; 40; 44].

Тут ми тільки вкажемо, що центри кіл трикутників Делоне є вершинами діаграми Вороного, побудову, якої проілюструємо в п. 9.8.5.

Якщо два трикутники мають спільне ребро в тріангуляції Делоне, то їх центри описаних кіл будуть з'єднані на діаграмі Вороного, тобто з'єднання центрів описаних кіл утворює діаграму Вороного.

Задачі тріангуляції 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.

9.8.5. Діаграма Вороного

Діаграма Вороного (або теселяція Вороного, або мозаїка Вороного) множини *n*-точок $P = \{P_1, P_2, \ldots P_n\}$ на площині – це таке розбиття площини на області (*комірки Вороного*) $V_1, V_2, \ldots V_n$ так, що V_i утворена з усіх точок, ближчих до P_i , ніж до будь-якої іншої точки системи точок *P* (рис. 9.31). Вершини діаграми називають *вершинами Вороного,* а її ребра – *ребрами Вороного*. Зазначимо, що вершини Вороного в загальному випадку не є точками множини *P*. (Георгій Вороний – видатний український математик).

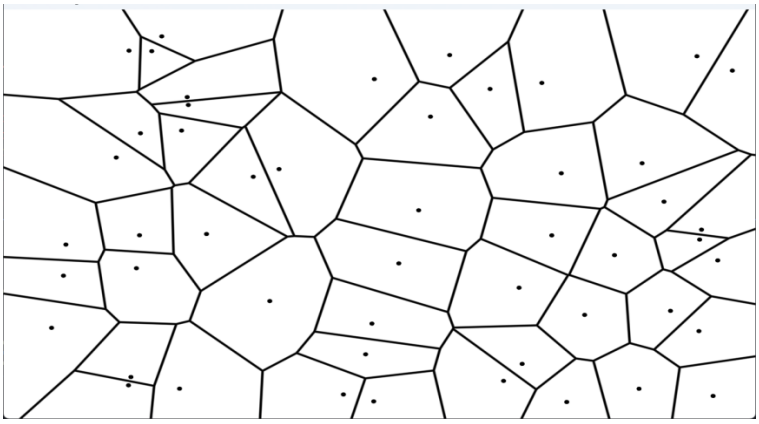


Рис. 9. 31. Діаграма Вороного

Діаграма Вороного розв'язує задачу пошуку найближчого сусіда або області близькості. Вона має широке практичне застосування у різних галузях: архітектурі, астрономії, дизайні, комп'ютерній графіці.

Наприклад, розв'язування задач алелопатії в садівництві, розміщенні установ для розбудови інфраструктури міст здійснюється за допомогою діаграми Вороного. В припущенні, що жодні чотири точки не лежать на колі, правильні нижче наведені твердження.

Твердження 1. Будь-яка вершина Вороного має степінь 3.

Твердження 2. Для будь-якої вершини Вороного *v,* що належить комірці *Vi*, коло радіуса $P_i v$, проведене з точки P_i, не містить всередині точок множини *P*.

Твердження 3. Діаграма Вороного містить максимум $2n-3$ вершин, $3n-5$ ребер та n комірок.

Твердження 4. Комірка Вороного не має границі тоді і тільки тоді, якщо відповідна точка множини *P* є вершиною опуклої оболонки масиву *P*.

Розглянемо рекурсивний алгоритм побудови діаграми Вороного методом "розділяй і володарюй".

- Множину точок *P* сортуємо за координатою *x* і ділимо на дві приблизно однакових за кількістю підмножини.
- Кожну з підмножин знову ділимо на дві і т. д., поки в кожній з підмножин залишиться не більше двох точок. Далі для кожної одержаної підмножини будуємо діаграму Вороного в оберненому порядку поділу точок, після чого ці діаграми об'єднуємо в одну. Відзначимо, що для множини з двох точок діаграма Вороного є серединний перпендикуляр.

Припустимо, що для кожної підмножини побудована діаграма Вороного. Виконаємо об'єднання цих множин та їх діаграм. Це можна зробити так.

1. Для кожної з підмножин знайдемо опуклу оболонку. На кожному кроці об'єднання діаграм Вороного ми об'єднуємо також і опуклі оболонки цих множин.
2. Тепер, коли ми маємо дві опуклі оболонки двох підмножин, знайдемо верхню і нижню межі цих множин, тобто ми повинні знайти два відрізки, які об'єднують дві знайдені опуклі оболонки в одну (звичайно вона буде опуклою).
3. З одержаних на кроці 2 відрізків вибираємо один, наприклад верхній, позначимо його через *L* (нижній відрізок позначимо через *Q*) і через його середину перпендикулярно випускаємо промінь і шукаємо його перетин з тими комірками Вороного, центрами яких є кінці відрізка, перпендикулярно якому ми випускали промінь. Нам потрібно вибрати серед них той перетин, що зустрічається раніше. Позначимо цю точку перетину як *M*, а

комірку, яку ми перетнули, запам'ятаєм і позначимо через *V*. Далі той кінець відрізка *L,* що є центром для неперетинної комірки Вороного, залишаємо на місці, а той, що був центром комірки, яку перетнули, обновляємо: ми перетнули одну зі сторін комірки Вороного, тоді новим кінцем відрізка *L* стане центр комірки Вороного, яка є суміжною по цій стороні з перетнутою коміркою. В спеціальній множині *S* (в ній зберігається межа двох діаграм Вороного) потрібно добавити ту частину променя, яка розміщена до перетину зі стороною комірки. Крок 3 повторюємо доти, поки значення кінців відрізка *L* не стануть дорівнювати значенням кінців відрізка *Q*. В результаті в множину *S* добавиться неперервний ламаний промінь.

4. На кроці 3 ми отримали неперервний ламаний промінь, який є межею, що з'єднує діаграми Вороного двох підмножин. Для отримання фінального результату, потрібно для діаграм Вороного лівої множини затерти ті відрізки, що знаходяться справа від одержаного променю, а для діаграми Вороного правої підмножини затерти ті – що зліва. Для цього потрібно враховувати той факт, що коли ми перетинаємо комірку, то спочатку промінь у неї входить, а потім виходить. І в залежності від того, з діаграмою якої множини ми працюємо, видалити відповідні ланцюжки ребер комірки Вороного, яку ми перетнули променем і добавити туди потрібну частину з множини *S*.

Роботу алгоритму "розділяй і володарюй" в графічній формі можна побачити на сайтах [56; 57]. Він здійснює побудову діаграми Вороного за O(n log(n)) операцій і є свого роду класикою в програмуванні. З'єднуючи ребрами суміжні центри описаних кіл трикутників Делоне, формуємо вершини діаграми Вороного. Маючи діаграму Вороного, проводимо серединні перпендикуляри до ребер Вороного. Як наслідок отримуємо тріангуляцію Делоне.

9.9. Тести властивостей поліедра

Вище наведені алгоритми для полігонів можна узагальнити на випадок поліедрів.

9.9.1. Тест перетину площини та поліедра

Задана площина $f(p)=0$ і поліедр$\{P,G\}$ списками своїх вершин $P=\{p_1, p_2, \ldots p_n\}$ і граней $G=\{G_1,G_2, \ldots,G_m\}$. Потрібно визначити, чи площина перетинає поліедр.

Цей тест ідентичний двовимірному випадку. Очевидно, що площина перетинає поліедр, якщо існує хоча б одна пара вершин

(p_i, p_j), що лежать по різні боки від площини. Різні знаки значень $f(p_i)$, $f(p_j)$ свідчать про те, що вершини p_i, p_j лежать по різні боки від площини. Тобто площина перетинає поліедр, якщо

$$\{1 \le \boldsymbol{j} < \boldsymbol{i} \le \boldsymbol{n}: f(p_i) \cdot f(p_j < 0\}.$$

У випадку, коли для значень $f(p)$ у вершинах досягаються нульові значення, то площина ще може дотикатися до поліедра вершиною, ребром і гранню.

9.9.2. Тест опуклості поліедра

В опуклого поліедра не існує ні однієї пари вершин, що лежать по різні сторони від будь-якої площини грані. І навпаки в неопуклому поліедрі завжди існує грань G_k і пара вершин (p_i, p_j), що знаходяться по різні боки від площини цієї грані. При цьому розглядаються вершини, що не належать грані G_k.

Це означає, що в алгоритмі тестування опуклості поліедра організовується зовнішній цикл по гранях G_i, i = 1, 2, …, m і внутрішній цикл обходу вершин p_j, j = 1, 2, …, n.

Якщо $f(p_j)$=0, то таку вершину p_j ігноруємо. Виловлюємо тільки випадок різних знаків для $f(p_j)\neq 0$. Якщо така пара знайдена, то алгоритм припиняємо і видаємо повідомлення про неопуклість поліедра.

Якщо по відношенню до всіх граней для $f(p_j)\neq 0$ спостерігався один і той самий знак для всіх пар точок, то поліедр – опуклий. За аналогією з полігонами для перевірки знаків можна організувати прапорці, які сигналізуватимуть про розміщення вершин (див. п. 9.4.1).

9.9.3. Тести орієнтації точки відносно поліедра

Опуклий тест. Задана точка q і опуклий поліедр $\{P,G\}$, визначити розміщення точки відносно поліедра (внутрішня чи зовнішня).

Припустимо, що для кожної грані маємо однаково орієнтовані нормалі, наприклад усі зовнішні, тоді точка q буде внутрішньою, якщо $f_i(q)$ одного знаку по відношенню до усіх площин граней. Внутрішні точки лежать по одну сторону від усіх граней поліедра. Для будь-якої зовнішньої точки завжди існує пара граней відносно яких точка q розміщена зрізних боків.

Точки, які не ідентифіковані як внутрішні і зовнішні будуть граничними, тобто лежать на гранях, ребрах або вершинах опуклого поліедра.

Габаритний тест. Задані поліедр $\{P,G\}$ і точка $q(x, y, z)$. Визначимо координати габаритних параметрів поліедра. Для цього

знайдемо мінімальні та максимальні координати вершин поліедра. Позначимо їх

$$x_{\min}, x_{\max}, y_{\min}, y_{\max}, z_{\min}, z_{\max}.$$

Тоді якщо

$$x \notin [x_{\min}, x_{\max}] \cup y \notin [y_{\min}, y_{\max}] \cup z \notin [z_{\min}, z_{\max}],$$

то точка q гарантовано не належить довільному поліедру $\{P,G\}$.

Якщо хоч одна з умов не виконана, то точка q не може бути однозначно ідентифікована як внутрішня чи зовнішня. Тобто габаритний тест повністю задачу орієнтації точки відносно полігона не розв'язує, він розпізнає лише ті точки, які лежать поза габаритами поліедра.

Променевий тест аналогічний варіанту з полігонами. А саме

- з точки q у довільному напрямку V випускаємо промінь $p(t)=q+Vt$;
- записуємо рівняння площини для i грані: $f_i(p) = 0$, $i=1, 2, \ldots, n$;
- обчисляємо значення параметра t з рівняння $f_i(q+Vt_i) = 0$ і точку перетину променю з площиною i-тої грані $c_i=q+Vt_i$;
- виконуємо тест орієнтації точки c_i по відношенню до полігона i-тої грані, тобто визначаємо точки, що належать грані;
- якщо при $t_i>0$ кількість таких точок парна або дорівнює нулю, то точка q не належить поліедру, якщо ця кількість непарна, то точка лежить у поліедрі. Точка лежить на межі поліедра, якщо хоча б одне з $t_i=0$. Якщо промінь перетне ребро або вершину поліедра, то потрібно тест повторити з новим напрямком променю V.

Кутовий тест. Просторового варіанта кутового тесту не існує.

Контрольні запитання 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 трикутник лежить усередині іншого?
9. В чому 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 багатокутника?
14. Як знайти перетин опуклих полігонів?
15. Як класифікуються алгоритми відсікання?
16. Як утворюється коди відрізків у *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–Азертона?
22. Наведіть загальну схему алгоритму Вейлера–Азертона.
23. Як визначаються вершини опуклої оболонки в методі загортання подарунка?
24. Опишіть роботу алгоритма Грехема на конкретному прикладі.
25. Що таке тріангуляція полігонів?
26. Як виконується тріангуляція неопуклих полігонів?
27. Розкрийте поняття тріангуляції Делоне.
28. Які перевірки потрібно здійснити, щоб дізнатися чи площина $f(p)=0$ перетинає поліедр?
29. Сформулюйте основну ідею тесту опуклості поліедра.
30. Які точки по відношенню до поліедра розпізнає габаритний тест?
31. Опишіть роботу алгоритма променевого тесту перевірки належності точки до поліедра.

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 променю з багатокутником у тривимірному просторі.
6. Розробити алгоритм визначення належності точки поліедру.

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 $\boldsymbol{V}$.
13. Розробити тест перетину/ об'єднання двох плоских полігонів.
14. Розробити алгоритм відсікання опуклого/неопуклого полігона півплощиною (рис. 9.5, *а*).
15. Навести приклад відсікання неопуклого багатокутника з отворами іншим неопуклим багатокутником з отвором. На цьому прикладі проілюструвати роботу алгоритма Вейлера-Азертона.
16. Розробити алгоритм розрізання неопуклого багатокутника на опуклі багатокутники, використовуючи операції повороту.
17. Розробити алгоритм тріангуляції монотонних полігонів.
18. Визначити чи знаходиться точка D у колі, описаному навколо трикутника ABC.
19. Задано масив точок P. Розробити ефективний алгоритм визначення належності будь-яких трьох точок одній прямій.
20. Задано масив точок P. Розробити ефективний алгоритм визначення належності будь-яких чотирьох точок одному колу.
21. Розробити алгоритм відсікання зафарбованої області.
22. Розробити програму для виконання бінарних операцій над багатокутниками, що задаються в інтерактивному режимі.
23. Задано полігон цілочисловими координатами своїх вершин. Знайти скільки точок з цілими координатами лежить усередині полігона. ***Вказівка.*** Для будь-якого полігона правильна формула Піка: $S=n+m/2-1$, де S – площа полігонаа; n – кількість цілих точок, що лежить строго в полігоні; m – кількість цілих точок, що лежить на межі полігона.
24. Розробити алгоритм перетину променя з площиною та алгоритм перетину променя зі сферою.
25. Розробити алгоритм перетину лінії/відрізка з плоским 3D-полігоном.
26. Розробити алгоритм зовнішнього відсікання відрізка $[a,b]$ опуклим поліедром $\{P,G\}$.

Розділ 10. Фрактали в комп'ютерній графіці

10.1. Поняття фрактала

При побудові моделей, що описують навколишній світ донедавна використовувалися такі геометричні об'єкти, як лінія, квадрат, круг, сфера, піраміда, конус та ін. Ці фігури не можуть адекватно описати природні об'єкти, тому що хмари – це не зовсім сфери, гори – це не піраміди. Математичний опис нескінченно ділимих об'єктів рівняннями ліній або поверхонь надзвичайно громіздкий.

Щоб обійти цю проблему, математиками знайдено спосіб зображення складних природних об'єктів. Вони розробили нові засоби, які дозволяють описати складні на перший погляд природні об'єкти, зокрема введено поняття фрактала.

Виявилося, що фрактальні структури в природі зустрічаються досить часто, вони незамінні при побудові зображень контурів хмар, гір, поверхонь морів, ландшафтів, русла річок, рослин, кристалів, тріщин у матеріалах, бронхів і багатьох інших об'єктів.

Поняття фрактала, фрактальної геометрії з'явилося в кінці 70-х років минулого століття і стало широко застосовуватися математиками та художниками. Фрактали знайшли широке застосування в комп'ютерній графіці завдяки компактності математичного апарату, необхідного для їх відтворення на екрані комп'ютера.

Слово *фрактал* походить від латинського fractus, що в перекладі означає "складений із фрагментів''. Цей термін запропонований математиком Бенуа Мандельбротом у 1975 р. для позначення самоподібних структур. У 1977 р. ним опублікована фундаментальна праця "Фрактальна геометрія природи" [45] ("The Fractal Geometry of Nature"), в якій він стверджував, що природні форми, які є складними геометричними фігурами, складаються з менших фігур, схожих на саму фігуру. Тому саме Мандельброта вважають засновником фрактальної геометрії.

Відтоді теорія фракталів отримала значний розвиток. За останній час з'явилося багато праць (особливо на Заході), присвячених фракталам (наприклад, [44]). В цих працях вивчаються математичні основи фракталів, а також наводяться оригінальні графічні зображення фракталів, одержаних за допомогою комп'ютерів.

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 визначення 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. Частина фрактала містить інформацію про весь фрактал.

Багато об'єктів навколишнього світу володіють властивістю самоподібності, або масштабної інваріантності, – точної або наближеної подібності об'єктів з їх окремими елементами. Зокрема, будова Всесвіту від галактик до атома описується моделями типу "ядро–електрони".

Існують *конструктивні (геометричні) та динамічні (алгебраїчні) фрактали.* Найбільш наочні *геометричні фрактали*, оскільки їхня форма може бути описана як послідовність простих геометричних операцій. На площині їх отримують за допомогою деякої ламаної, яка називається ***генератором*** *S* (шаблоном). Шаблон *S* – це список відрізків, які заміняють одиничний відрізок на осі *Ox.* Шаблон із відрізків однакової довжини називається *однорідним***.**

Спочатку кожен із відрізків основи *O* замінюється набором більш коротких відрізків із шаблону *S і*з можливістю чергування сторони їх розміщення в залежності від номера ітерації та номера відрізка генератора. Основою *O* може бути відрізок *ab* або правильний *n*-кутник $\{p_1, p_2, \dots, p_n, p_1\}$.

Далі ***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 у більшості випадків носять рекурсивний характер. Рекурсивність зумовлює властивість самоподібності фрактала.

Методи побудови фрактальних об'єктів можуть бути як ***рекурсивними*** так і ***ітераційними,*** причому рекурсивні простіші в програмуванні. Теоретично глибина рекурсії може бути нескінченною. На практиці ми обмежуємся деяким наближенням атрактора.

Прикладом геометричного фрактала є крива Коха. Послідовні наближення до фрактала Коха зображено на рис. 10.1.

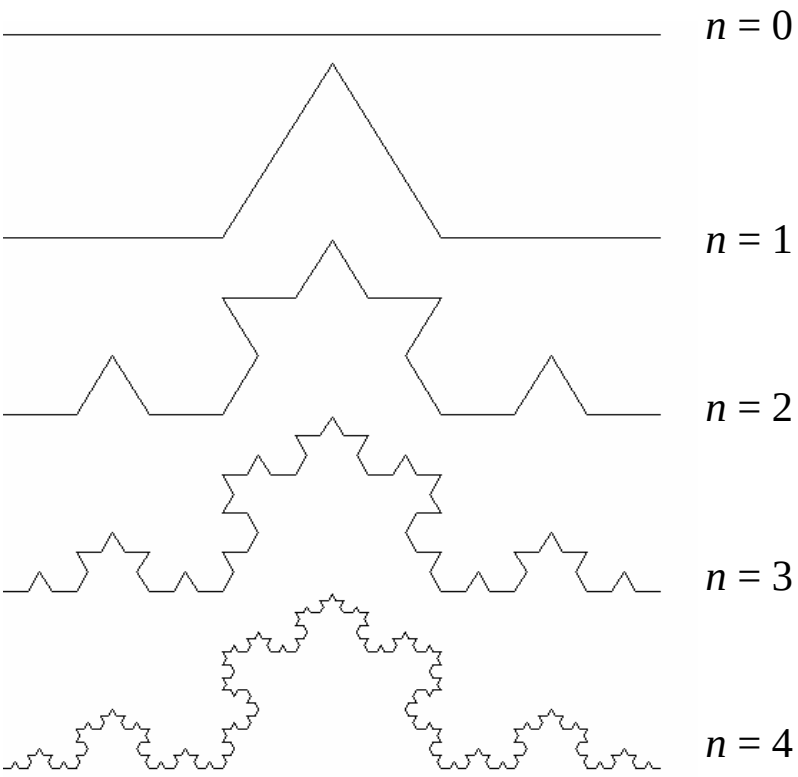


Рис. 10.1. Фрактал Коха

Досить відомим прикладом конструктивного фрактала може бути дерево, стовбур якого розділений на менші гілки, а кожна із цих гілок ділиться на ще 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 подібна на весь фрактал (рис. 10.2).

Поряд із конструктивними фракталами були відкриті множини, схожі на фрактали (вони можуть володіти масштабною інваріантністю 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 фазовий портрет цієї системи.

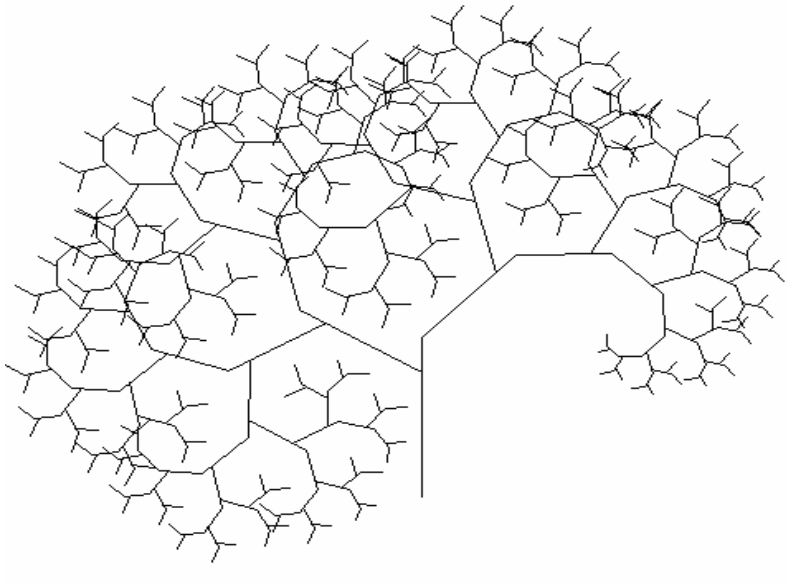


Рис. 10.2. Фрактал „дерево”

Динамічні фрактали одним із перших описав у 1918 р. французький математик Г. Жуліа. На той час ще не було зображень фракталів. Комп’ютери зробили видимим те, чого не уявляли в часи Жуліа.

Крім детермінованих фракталів, існують ще стохастичні фрактали. Вони дають ще більше різноманіття форм.

10.2. Конструктивні фрактали

Фрактали, в яких фрагменти повторюються при зменшенні масштабу, називаються *конструктивними фракталами*. Розглянемо математичні основи конструктивних фракталів. Вивчення геометричних фракталів почнемо з формування плоскої лінії Коха.

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 *ab*. Для цього відрізок прямої розбиваємо на три рівних частини точками *c* і *d*. Далі усуваємо середню третю частину відрізка, тобто *cd* і замінюємо її на сторони рівностороннього трикутника *ced* (рис.10.3). У результаті відрізок перетворюється на ламану, довжина якої в 4/3 разів більша від довжини вихідного відрізка. Далі ця операція повторюється багатократно для кожного відрізка ламаної (рис. 10.1), поки довжина відрізка ламаної не стане меншою δ або кількість операцій розбиття не досягне максимального числа *level*.

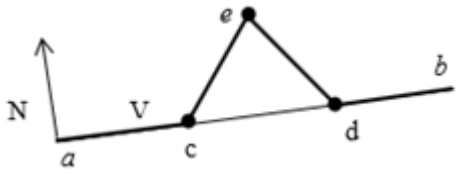


Рис. 10.3. Генератор фрактальної кривої Коха

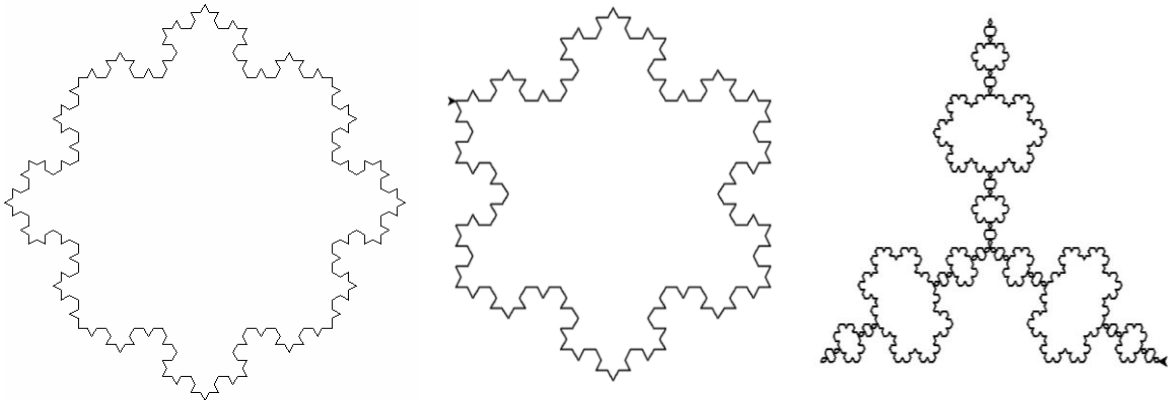
Якщо основа має одиничну довжину, то ламана при $n = 1$ складається з 4 відрізків (кожен довжиною 1/3). При $n = 2$ одержуємо ламану з 16 відрізків загальною довжиною $16/9 = (4/3)^2$.

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.

Оскільки вже після першого поділу відрізок перетворюється в ламану, то спосіб побудови фрактала Коха на відрізку легко можна узагальнити на довільну полілінію, що задана списком вершин $P = p_1p_2\ldots p_n$, зокрема й замкнену (полігон) $P = p_1p_2\ldots p_np_1$.

Якщо за основу кривої Коха взяти сторони правильного багатокутника, 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. 10.4, *а* ($p = 4$), а якщо – правильний трикутник, то сніжинку Коха (рис. 10.4, *б*). Якщо генератор Коха орієнтований усередину трикутника, то сніжинка Коха має вигляд наведений на рис 10.4 *в*.



а *б* *в*

Рис. 10.4. *а* – острів Коха; *б, в* – сніжинки Коха

Зауважимо, що фрактальна сніжинка на p-тій ітерації приростає площами $3\cdot4^{p-1}$ трикутників, площа кожного з яких становить

$S_0/9^p$, де S_0 – площа вихідного трикутника. Якщо знайти суму ряду

$$1+3/9+12/9^2+ \ldots + 3\cdot 4^{p-1}/9^p+ \ldots = 8/5,$$

то видно, що гранична площа фрактала становить $8/5\ S_0=1{,}6\ S_0$.

Приклад програми, що будує фрактал Коха на відрізку, який задається точками (startx, starty) та (endx, endy) з глибиною рекурсії *level*, наведений на лістингу. Тут реалізований відносно простий рекурсивний алгоритм ***koch*** побудови лінії Коха. Рекурсивний метод може бути застосований завдяки самоподібності фрактала Коха.

```
#include <iostream>//Підключення бібліотек
#include <Windows.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159265 // Визначеннння константи π
HWND hwnd = GetConsoleWindow();
HDC dc = GetDC(hwnd);
const COLORREF white = RGB(255, 255, 255);//Задання кольорів
const COLORREF yellow = RGB(255, 255, 0);
const COLORREF green = RGB(0, 255, 0);
const COLORREF deep_pink = RGB(255, 20, 147);
HPEN pen = CreatePen(PS_SOLID, 1, deep_pink); //Створення ручки

void koch(float startx, float starty, float endx, float endy,
int level)
{
        float x1, x2, x3, y1, y2, y3; //Оголошення змінних
        if (level == 1)
        {
              MoveToEx(dc, int(startx), int(starty),
NULL);//Переміщення на координати int(startx), int(starty)
              SelectObject(dc, pen); //Вибір ручки
              LineTo(dc, int(endx), int(endy)); //Лінія до
координат int(endx), int(endy)
        }
        else
        {
        x1 = startx+(endx - startx) / 3; //Точки розбиття
відрізка
     x3 = startx+2*(endx - startx) / 3;
     y1 = starty+(endy - starty) / 3;
     y3 = starty+2*(endy - starty) / 3;
     x2 = (endx - x3)*cos(2*PI/3)+(endy - y3)*sin(2*PI/3) + x3;
     y2 = (endy - y3)*cos(2*PI/3)-(endx - x3)*sin(2*PI/3) + y3;
          koch(startx,starty, x1, y1, level - 1);
          koch(x1,y1,x2,y2, level - 1);
          koch(x2,y2, x3, y3, level - 1);
```

```
            koch(x3, y3,endx, endy, level - 1);
             }
}
int main()
{
        koch(100,200,500,200,10);
        ReleaseDC(hwnd, dc);
        _getch();
}
```

Для розрахунку проміжних точок використовувалися формули поділу відрізка у заданому співвідношенні та формули повороту точки. Відображення, що описує поворот точки (x_2, y_2) на кут α (проти годинникової стрілки) відносно довільної точки (x_1, y_1), має вигляд

$$x_3 = (x_2 - x_1)\cos\alpha - (y_2 - y_1)\sin\alpha + x_1, \qquad (10.1)$$
$$y_3 = (x_2 - x_1)\sin\alpha + (y_2 - y_1)\cos\alpha + y_1.$$

Розглянемо ще ітераційний алгоритм ***KOCHit***(*P*, *D*, *δ*, *num*) на ламаній лінії $P=p_1p_2\ldots p_n$ з орієнтацією нових точок (справа або зліва від ламаної) $D=\{-1,1\}$, мінімальною довжиною поділу відрізка δ і максимальною кількістю ітерацій *num*.

Для розрахунку точки *e* генератора сформуємо матрицю обертання Rot = $R(D\cdot\pi/3)$ на кут 60º в напрямку *D*. Список точок кривої Коха позначимо через *Lp*.

- Далі в зовнішньому циклі ітерацій по $i=1, 2,\ldots,$ *num* ініціалізуємо список $Lp=\{p_1\}$ і для поділу відрізків ламаної $p_к\ p_{к+1}$ перейдемо до внутрішнього циклу по $k=1, 2,\ldots, n-1$, де n = size(P) – обновлювана кількість вершин полілінії.
- Знаходимо третю частину вектора напрямку відрізка $p_к\ p_{к+1}$, $V= (p_{к+1} - p_к)/3$. Якщо довжина k-того відрізка $|p_{к+1} - p_к|<\delta$, то досягнута межа для поділу відрізка і в список *Lp* додається точка $p_{к+1}$ (кінець відрізка). Дальше цикл виконується для наступного значення k++.
- Інакше при $|p_{к+1}-p_к|>\delta$ в список *Lp* додаються три нових точки $c = p_к+V$, $e=c+V\cdot$Rot, $d = c+V$ (рис. 10.7) і кінцева точка $p_{к+1}$. Точку *e* можна одержати також іншим способом. Для цього потрібно рухатись вздовж вектора нормалі $N = V\cdot R(D\cdot\pi/2)$, тобто $e=c+0.5(V+\sqrt{3}\cdot N)$

При завершенні циклу по k, тобто при завершенні обходу всіх відрізків ламаної $P = p_1p_2\ldots p_n$, лінія Коха на i-тій ітерації побудована. Список *Lp* перейменовується в *P*, а цикл ітерацій продовжується для наступного значення i++ і оновленої ламаної *P*.

При досягненні значення $i = num$ алгоритм завершує роботу і в списку P містяться вершини лінії Коха в порядку їх з'єднання. Готовий фрактал зображаємо на екрані.

Змінюючи параметри δ та num, можемо керувати виглядом кривої Коха. Наприклад при $\delta=0$ кожен відрізок ламаної буде поділений на num частин незалежно від його довжини. Якщо взяти num достатньо великим, то всі відрізки будуть дробитися доти, поки всі їх довжини не стануть меншими за $\delta>0$.

Для того щоб алгоритм не працював у "холосту", коли довжини всіх відрізків стануть меншими за δ а значення $i = num$ ще не досягнуто, в алгоритмі можна використати ознаку q, яка обнуляється на початку кожної ітерації і змінюється на $q=1$ після першого розбиття. Якщо на всій ітерації не було жодного розбиття, то за ознакою $q = 0$ алгоритм достроково припиняє роботу і повертає сформований список P.

Блок схема ітераційного алгоритму наведена на рис 10.5.

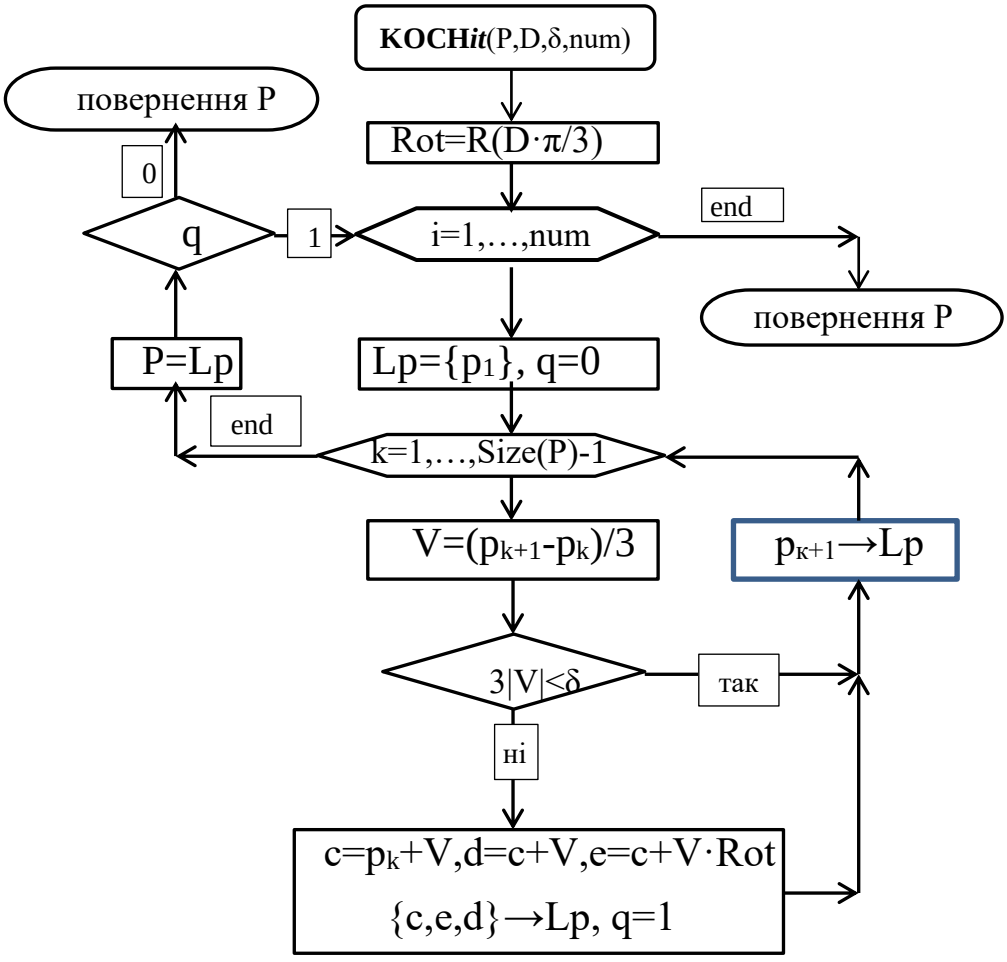


Рис. 10.5. Блок схема ітераційного методу побудови кривої Коха

Якщо зробити у функції ***KOCHit*** невеликі зміни, а саме Rot=$R(D\cdot\pi/2)$, $g=e+V$, $\{c, e, g, d\}\rightarrow Lp$, то одержимо заміну середньої частини не на рівносторонній трикутник, а на квадрат (рис. 10.6). У результаті можна одержати цікаві фрактальні узори. На рис. 10.7 фрактали створені на сторонах квадрата $P = p_1p_2p_3p_4p_1$ за $num = 3$ ітерацій з орієнтацією генератора назовні та всередину квадрата.

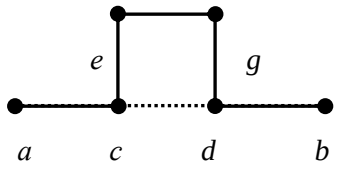


Рис. 10.6. Генератор для узагальнення кривої Коха

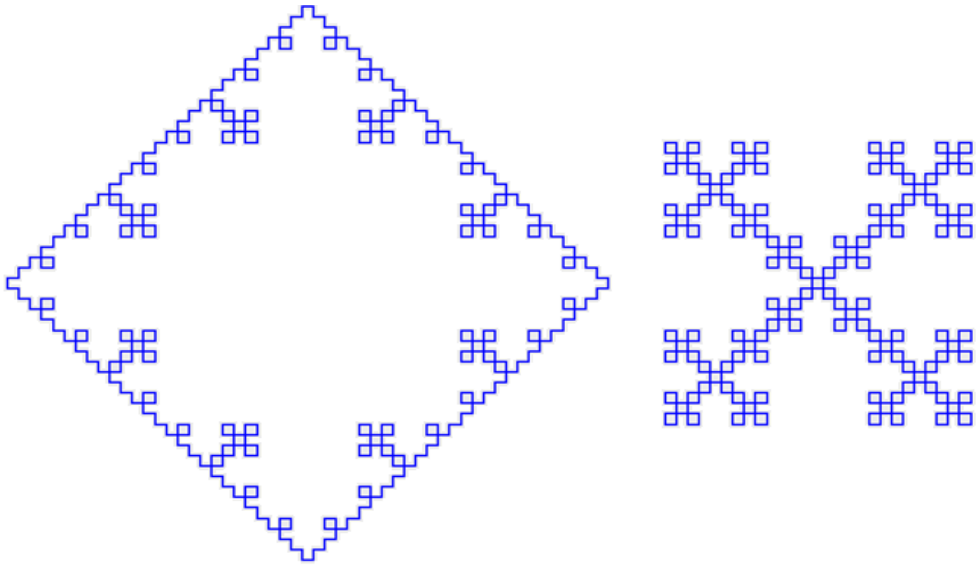


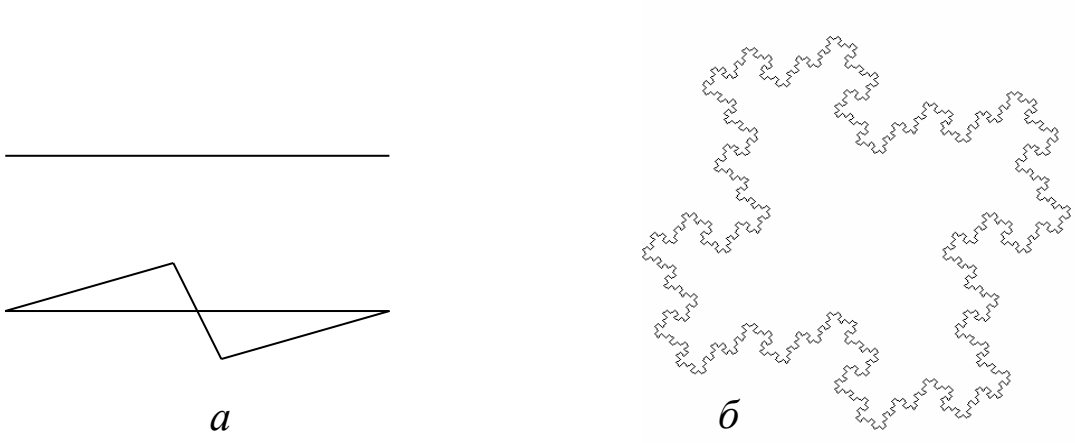
Рис. 10.7. Узагальнений фрактал Коха. Глибина рекурсії 3

Зауважимо, що граничні форми фрактальних кривих вимагають великих обсягів пам'яті як для їх розрахунку так і для зображення, оскільки вивід фрактала здійснюється після його повного обчислення. Обсяг оперативної пам'яті, що зберігає всі вершини фрактальної кривої, зростає у геометричній прогресії від заданої кількості ітерацій *num* або рекурсії *level*. В ітераційному алгоритмі можливості зменшити обсяг даних, які треба зберегти немає, а в рекурсивному алгоритмі можливі модифікації для зменшення обсягу збережуваних даних.

Розглянемо загальну схему побудови подібних фракталів. Припустимо, що основа складається з u відрізків, а фрагмент – із v відрізків, тоді при заданому порядку апроксимації p із кожного відрізка основи одержується ламана з $v^p - 1$ вершинами.

Наведемо приклади інших фракталів.

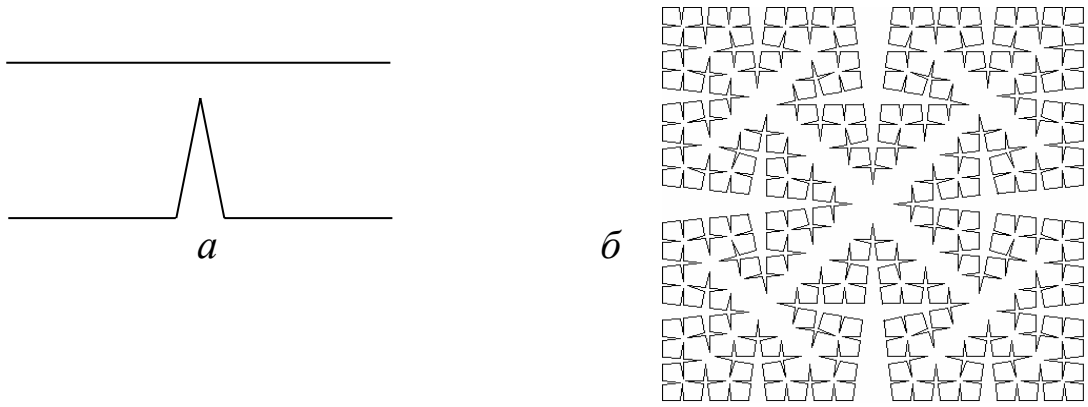
1. Основою є квадрат з одиничною довжиною. Відрізок [0, 1] замінюється фрагментом з проміжними точками (0,4; 0,2), (0,6; - 0,2) (рис. 10.8, *а*). Одержимо острів Мінковського (рис. 10.8, *б*).



а *б*

Рис. 10.8. 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. 10.9, *а*). Фрактал (різаний квадрат) зображений на рис. 10.9, *б*.



а *б*

Рис. 10.9. Різаний квадрат (p = 4)

10.2.2. Гілка папороті та дендрити

Одними з найпопулярніших геометричних фракталів є зображення, що мають вигляд рослин, наприклад гілка папороті. Розробимо рекурсивний алгоритм створення цього фрактального зображення. Застосування рекурсії обґрунтовується самоподібністю.

Усі 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, повернута на кут α відносно точки P_1 (рис. 10.10). Це перетворення задається формулою (10.1). У випадку $\alpha = 0$ стовбур і гілки прямі.

Потім знаходимо точку P_4, що одержується з P_3 за допомогою перетворення стиску відносно точки P_1 з коефіцієнтом стиснення $0 < k < 1$.

Координати точки P_4 знаходяться за формулами

$$x_4 = x_1 k + x_3 (1 - k), \qquad y_4 = y_1 k + y_3 (1 - k).$$

Із точки P_4 будуть виходити нові гілки. Зауважимо, що у більш загальному випадку нові гілки можуть виходити з різних точок. Побудуємо ці гілки. Для цього спочатку знаходимо точку P_5 із відрізка P_3P_4 за формулами стиснення відносно довільної точки

$$x_5 = x_4(1 - k_1) + x_3 k_1,\ y_5 = y_4(1 - k_1) + y_3 k_1\ , \quad 0 < k_1 < 1.$$

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

$$x_6 = (x_5 - x_4)\cos\beta - (y_5 - y_4)\sin\beta + x_4;$$
$$y_6 = (x_5 - x_4)\sin\beta + (y_5 - y_4)\cos\beta + y_4;$$
$$x_7 = (x_5 - x_4)\cos\beta + (y_5 - y_4)\sin\beta + x_4;$$
$$y_7 = -(x_5 - x_4)\sin\beta + (y_5 - y_4)\cos\beta + y_4.$$

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. Величини α, β, k, k_1 є параметрами, які визначають вигляд фрактала в цілому (рис. 10.11).

Алгоритм побудови фрактала можна записати у вигляді рекурсивної процедури ***branch***.

```
#include <iostream>
#include <Windows.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159265
HWND hwnd = GetConsoleWindow();
HDC dc = GetDC(hwnd);
const COLORREF white = RGB(255, 255, 255);
const COLORREF yellow = RGB(255, 255, 0);
const COLORREF green = RGB(0, 255, 0);
HPEN pen = CreatePen(PS_SOLID, 1, green);
const float alpha = 4 * PI/180;
const float beta = 70 * PI / 180;
const float k = 0.4;
const float k1 = 0.7;
const int l_min = 5;
void branch(float x1, float y1, float x2, float y2, int num)
{
   if (num != 0 && (pow(x1 - x2, 2) + pow(y1 - y2, 2)) > l_min) {
     float x3, x4, x5, x6, x7, y3, y4, y5, y6, y7;
```

```
        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;
        y4 = (1 - k) * y1 + k * y3;
        x5 = (1 - k1) * x4 + k1 * x3;
        y5 = (1 - k1) * y4 + k1 * y3;
        x6 = (x5 - x4) * cos(beta) - (y5 - y4) * sin(beta) + x4;
        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;
        MoveToEx(dc, x1, y1, NULL);
        CreatePen(0, 1, yellow);
        SelectObject(dc, pen);
        LineTo(dc, x4, y4);
        branch(x4, y4, x3, y3, num);
        branch(x4, y4, x6, y6, num - 1);
        branch(x4, y4, x7, y7, num - 1);
        }
}
int main()
{
        branch(300, 500, 300, 50, 3);
        ReleaseDC(hwnd, dc);
        _getch();
}
```

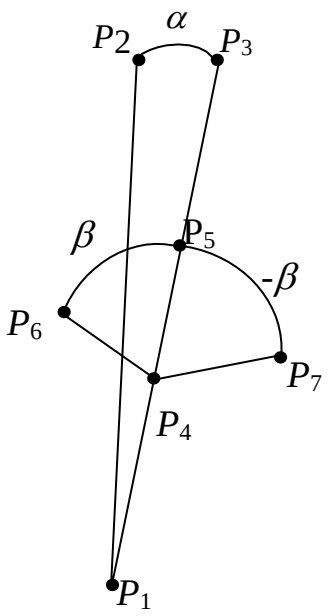


Рис. 10.10. Схема фрактала

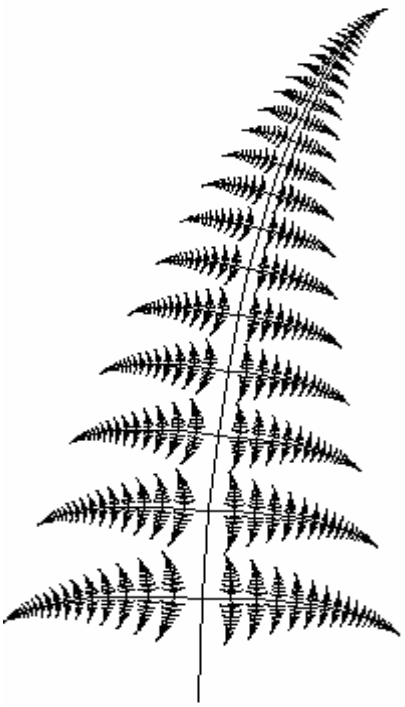


Рис. 10.11. Фрактал „гілка папороті”

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. Це підвищить реалістичність зображення.

У даному алгоритмі завершення циклів ітерації відбувається ще й тоді, коли довжина гілки стає меншою від деякої величини $l_{\min}$.

Якщо в алгоритмі $l_{\min} = 0$, то гілка папороті буде побудована за *num* рекурсій, при цьому загальна кількість відрізків, що складають зображення, дорівнює

$$1 + 3(1 + 3(1 + \cdots + 3(1 + 3))) = \sum_{i=1}^{num} 3^{i-1} = (3^{num} - 1)/2.$$

Зокрема, при *num* = 5 кількість гілок папороті дорівнює 121. Ці параметри дозволяють регулювати густину гілок, глибину рекурсій і необхідні машинні ресурси.

Задавши $l_{\min}$, можна оцінити максимум глибини рекурсії:

$$k^{num} \cdot h < l_{\min} \rightarrow num > \log(l_{\min}/h)/\log k,\ h = |P_1 - P_2|.$$

В алгоритм побудови гілки папороті можна вносити випадкові збурення, зокрема випадково зміщувати точки розгалуження або кути, використовуючи функцію *rnd*(*x*) генерування випадкового числа, рівномірно розподіленого на інтервалі (0, *x*).

Об’єкти типу дерева, кущі, листя, які розгалужуються на елементи, що подібні цілому, називаються дендритами (від грецького, що означає дерево). Їх зображення одержують за допомогою алгоритмів, подібних до алгоритму формування гілки папороті ***branch.*** Необхідно тільки вибрати параметри, що визначають форму рослини. Наприклад для гілки папороті використовується всього 5 параметрів: $l_{\min}$, k, k1, α, β. Разом з аргументами функції *branch* їх всього 10.

Зазначимо, що такі складні форми дендритів створюються за допомогою простих програм і надзвичайно малого обсягу вхідних даних. Звідси виникають значні практичні можливості фрактальної геометрії: описуючи об’єкти мовою фракталів, ми істотно зменшуємо обсяг інформації, яка необхідна для побудови зображення та його збереження в пам’яті комп’ютера.

10.2.3. 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 одер-

жимо зірковий фрактал. Задавши деяку кількість кроків p, одержуємо наближення зіркового фрактала. На рис. 10.12 зображено наближення зіркового фрактала при $p = 2$. Виконаємо нумерацію відрізків фрактала від 0 до 19 .

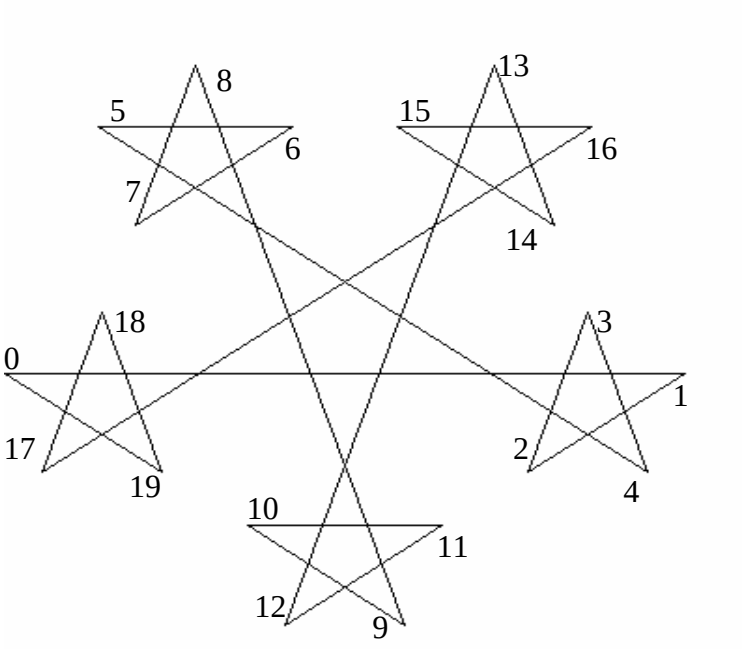


Рис 10.12. Зірковий фрактал при $p = 2$

Зірковий фрактал можна побудувати як замкнену ламану лінію, послідовні відрізки якої завжди перетинаються під одним і тим самим кутом $\alpha = \pi - (\pi/5) = 4\pi/5$, тому що зірка правильна фігура 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. 10.13.

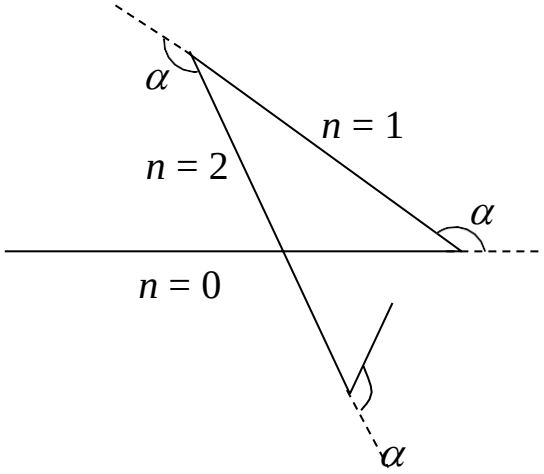


Рис. 10.13. Схема побудови ламаної

При побудові зіркового фрактала ми повинні виробити правило, за яким визначається довжина n-го відрізка ламаної, якщо відома довжина першого відрізка.

Зображений фрактал на рис. 10.12 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. 10.12 $r = 0{,}35$. Як видно з рис. 10.12, 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. 10.14) правило довжин таке:
$n = 0, 16, 32,\ldots$ – довжина $1 = r^0 = r^{p-3}$;
$n = 4, 8, 12, 20,\ldots$ – довжина $r = r^1 = r^{p-2}$;
$n = 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15,\ldots$ – довжина $r^2 = r^{p-1}$.

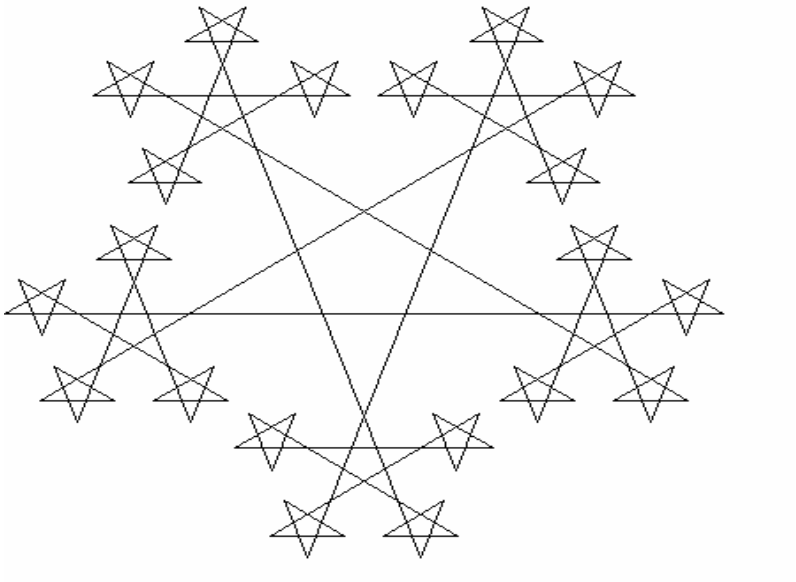


Рис. 10.14. Зірковий фрактал при $p = 3$

Отже, довжина відрізка з індексом n залежить від кількості множ- ників 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 кількість 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.

10.3. Аналіз конструктивних фракталів

Геометричні фрактали можна задавати і лінійними афінними дискретними перетвореннями деякого геометричного об’єкта.

Фрактали можна визначити як множини точок інваріантних відносно перетворення поворот-стиск. У простішому випадку стиск з поворотом (або без нього) – це масштабне зменшення, що

задається лінійним відображенням площини (x, y) в (x', y'), тобто

$$x' = ax + by,\ \ y' = cx + dy. \qquad (10.2)$$

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/стисненння відносно початку координат (центральне перетворення) задається формулами $x' = cx$, $y' = cy$, а відносно точки (x_0, y_0) – формулами

$$x' = c(x - x_0) + x_0,\ y' = c(y - y_0) + y_0.$$

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/стискування відносно початку координат має вигляд $x' = ax - by$, $y' = by + ax$, а відносно довільної точки:

$$x' = ax - by + c,\ y' = ay + bx + d. \qquad (10.3)$$

Величина $Q = a^2 + b^2$ характеризує величину розтягу. При $Q > 1$ маємо розтяг, при $Q < 1$ – стиск.

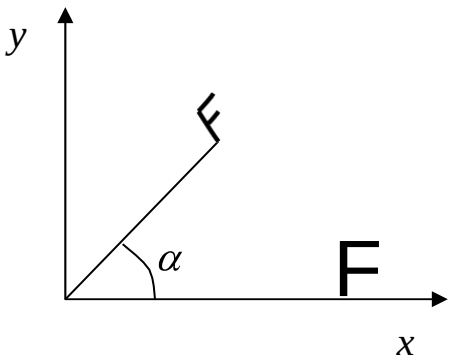


Рис. 10.15. Перетворення поворот-стиск

Перетворення поворот-стиск зображено на рис. 10.15. 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. 10.16, *а*. Наближення фрактала визначається своїми вершинами, тому потрібно на кожному кроці знайти вершини ламаної і з'єднати їх прямими лініями (рис. 10.16, *б*).

З рис. 10.16, *а* 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 $1/2^{½}$.

Друге перетворення R має центр у точці (1, 0), кут повороту – $\pi/4$, коефіцієнт стискування $1/2^{½}$. Перетворення L та R задаються відповідно формулами (випливає з (10.1))

$$L\colon x' = (x - y)/2,\quad y' = (y + x)/2, \qquad R\colon x' = (x + y + 1)/2,\quad y' = (y - x + 1)/2.$$

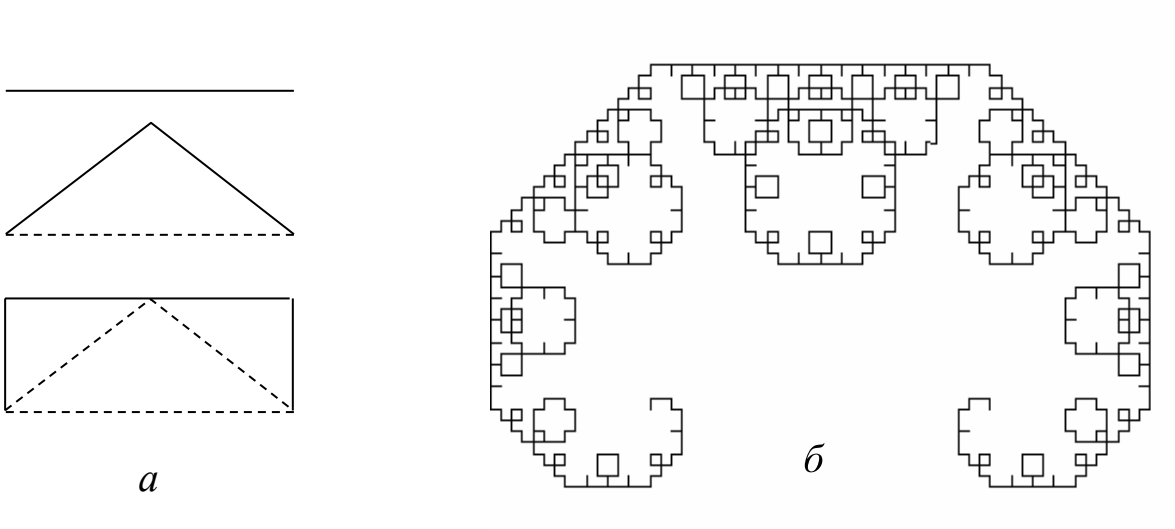


Рис. 10.16. Фрактал Леві (n = 10)

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. На рис. 10.17 побудовано дерево ітерацій генерування точок фрактала Леві.

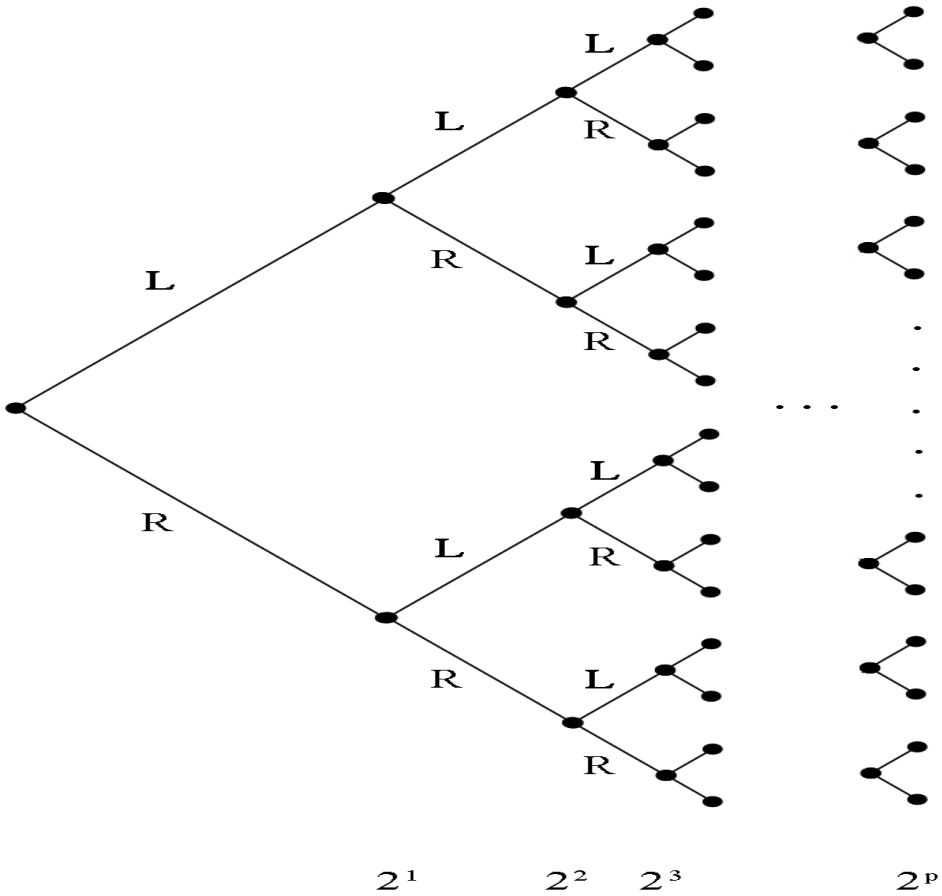


Рис. 10.17. Двійкове дерево ітерацій фрактала Леві

Розглянемо такі перетворення.

$$L: x' = ax - by, \quad y' = ay + bx, \qquad R: x' = cx - dy + 1 - c, \quad y' = cy + dx - 1. \tag{10.4}$$

Перетворення (10.4) є окремим випадком більш загальних перетворень повороту-стиску (10.3). Перетворення L – це поворот-стиск відносно точки (0, 0) з коефіцієнтом $(a^2+b^2)^{½}$, а R – поворот-стиск відносно точки (1, 0) з показником $(c^2 + d^2)^{½}$.

Наведемо конкретні приклади таких фракталів:

а) $a = d = 0$, $b = c = 0.7$, $p = 17$ (рис. 10.18, *а*);

б) $a = b = 0.6$, $c = 0.53$, $d = 0$, $p = 18$ (рис. 10.18, *б*).

Метод побудови фракталів за допомогою LR-перетворень можна узагальнити на випадок, коли замість початкової точки береться ціла множина точок і на кожну з цих точок діють багатьма матричними перетвореннями.

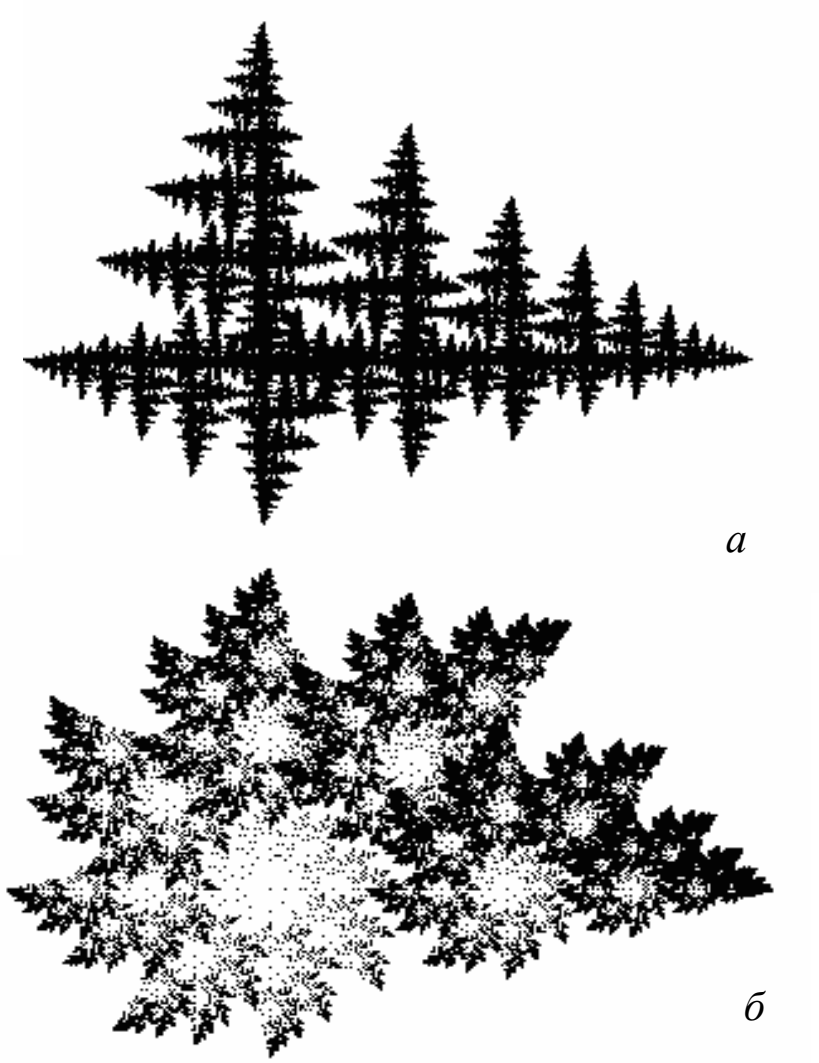


Рис. 10.18. Фрактал, одержаний за допомогою повороту-стиску

Нехай початкова множина складається з n_0 точок $P^0=\{p_1^0, p_2^0, \dots, p_{n_0}^0\}$, а $\{C_1, C_2, \dots, C_m\}$ – множина матриць афінних перетворень. На першій ітерації до кожної з точок множини P^0 застосуємо m перетворень, заданих матрицями $C_1, C_2, \dots, C_m$. Одержимо множину з $n_1 = m \cdot n_0$ точок $P^1 = \{p_1^1, p_2^1, \dots, p_{n_1}^1\}$. Далі до кожного елемента множини P^1 теж застосуємо перетворення матрицями $C_1, C_2, \dots, C_m$. Продовжуючи цей ітераційний процес одержимо послідовність множин P^i , де

$$P^{i+1}=\bigcup_{j=1}^{m} P^i C_j,\ i=1,2, \dots .$$

Ця послідовність точок для кожного набору матриць $C_1, C_2, \dots, C_m$ при $i\to\infty$ дає граничне зображення, яке називається атрактором. Цікаво, що форма атрактора не залежить від початкового набору P^0 , а залежить лише від набору матриць $C_1, C_2, \dots, C_m$.

Ітерації будуть збігатися до атрактора, якщо відображення, що задаються матрицями стискуючі, тобто визначник $-1<|C_j|<1$, $j=1,\dots,m$.

На рис. 10.19 побудовано дерево двох ітерацій генерування множин P^1, P^2 з P^0. Множини $P_1^1, P_2^1, \dots, P_m^1$ утворюються при перетворенні множини P^0 відповідно матрицями $C_1, C_2, \dots, C_m$. На другій ітерації матрицями $C_1, C_2, \dots, C_m$ діємо на множини $P_1^1, P_2^1, \dots, P_m^1$, одержуємо набір $P_{j_1 j_2}^2$, $j_1\ j_2=1, 2,\dots, m$. В загальному випадку на i-тій генерується множина $P_{j_1 \dots j_i}^i$, $j_1\ j_2 \dots j_i=1, 2,\dots, m$. Об‘єднання цих множин по $j_1 j_2 \dots j_i$ дає колаж i-тої ітерації, який при $i\to\infty$ збігається до атрактора.

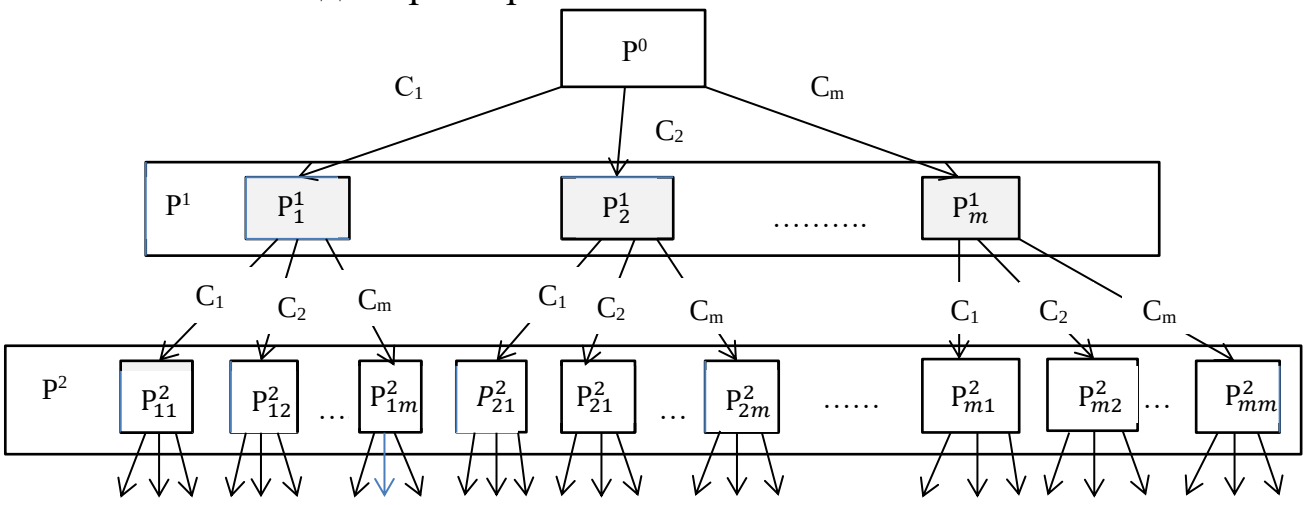


Рис.10.19. Дерево ітерацій генерування множин

Точки, що обчислюються за ітераційними формулами, можна виводити в двох режимах: 1) ***режим заміни***, при якому зображуються тільки точки одержані на останній i-тій ітерації;

2) ***режим додавання***, при якому зображуються точки на всіх ітераціях у міру їх обчислення.

Наведений метод генерування фракталів називається *методом систем ітеративних функцій* (скорочено СІФ, анг. Iterated Function System, IFS). Метод IFS з'явився у середині 80-х років як простий засіб одержання фрактальних самоподібних структур. СІФ генерує самоподібні множини. Фрактали СІФ складаються з декількох власних копій, кожна з яких перетворюється функцією. Функції, як правило, набір стискувальних зображень. Приклади фракталів, які можуть бути одержані за допомогою СІФ, ще описані у 1884 р. – це набір Кантора. Нині СІФ представляють моделі для певних рослин і багатьох розгалужених структур природи, завдяки самоподібності.

Координати нових точок фрактала знаходяться за формулами

$$x_{k+1}= F_1(x_k, y_k),\ y_{k+1}= F_2(x_k, y_k),\ \ k = 1,2,\dots ,$$

де F_1, F_2 – функції перетворення координат, наприклад лінійні афінні перетворення, що задаються матрицями.

Приклад. Проілюструємо застосування методу систем ітеративних функцій для побудови зображення кленового листа на основі афінних перетворень.

На реальній фотографії кленового листа обирають три опорні точки, наклавши лист на деяку систему координат і прочитавши координати цих точок. Нехай це будуть точки p_1, p_2, p_3, які утворюють трикутник, що нагадує форму листа (рис. 10.20).

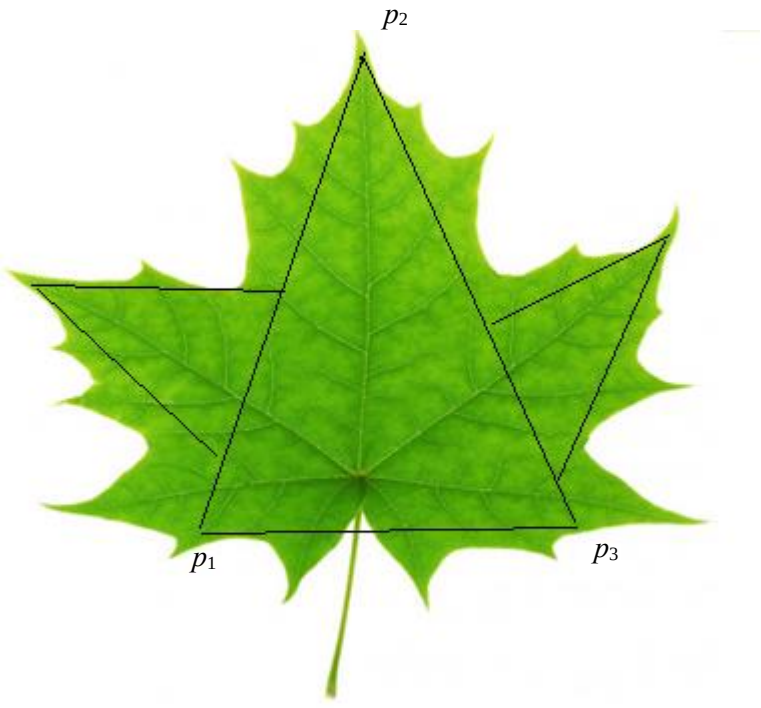


Рис. 10.20. Зображення кленового листа

Дальше масштабуючи це зображення, будемо накладати зменшені копії листа на подібні елементи листа шляхом перенесення і повороту, прагнучи максимально покрити всю площу листа. Тобто одним масштабованим трикутником покриємо центральний пелюсток, другим – лівий пелюсток, третім – правий пелюсток і четвертим – стебло листа (рис. 10.20). В такий спосіб одержимо чотири трійки точок, у які будуть відображатися три початкові точки.

Для цих відображень методом парних точок побудуємо матриці афінних перетворень, які початкові координати будуть відображати у координати точок елементів листа. Такі матриці в конкретному випадку розраховані й мають вигляд

$$C_1=\begin{pmatrix}0.456 & -0.006 & 0\\ 0.019 & 0.683 & 0\\ 2.118 & 2.243 & 1\end{pmatrix},\ C_2=\begin{pmatrix}0.327 & 0.293 & 0\\ -0.534 & 0.591 & 0\\ 2.381 & -0.453 & 1\end{pmatrix},$$

$$C_3=\begin{pmatrix}-0.354 & 0.351 & 0\\ 0.618 & 0.628 & 0\\ 4.782 & -0.748 & 1\end{pmatrix},\ C_4=\begin{pmatrix}-0.002 & -0.053 & 0\\ 0.006 & 0.628 & 0\\ 4.782 & -0.748 & 1\end{pmatrix}.$$

Оскільки визначники цих матриць по модулю менші за одиницю, то афінні відображення, що задаються цими матрицями, є стискувальними. На рис. 10.21 показано зображення кленового листа за 10 ітерацій у режимі заміни з початкової множини P_0, що складається з однієї точки p_1 = (4,14; 7; 1). На першій ітерації на точку p_1 подіяли матрицями C_1, C_2, C_3, C_4, одержали 4 точки. Далі на кожну з 4 точок знову подіяли матрицями, одержали 4^2=16 точок і т. д. Кількість точок на 10 ітерації становить 4^{10}=1048576.

Задача. Написати програму побудови зображення кленового листа.

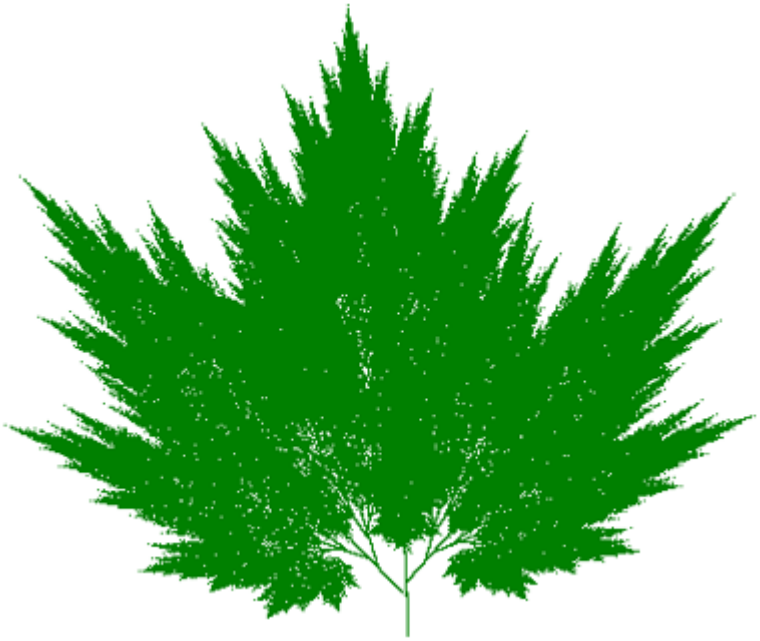


Рис. 10.21. Фрактал кленового листа

Зауважимо, що 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 цифрового графічного зображення необхідно підібрати коефіцієнти афінного перетворення, які за допомогою ітерацій створюють зображення, візуально схоже на оригінал. Цей метод використовується для запису кольорових фотографій у файли зі стискуванням у 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). Межі областей притягування (збіжності) коренів нелінійних рівнянь мають ***фрактальну структуру***. 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 (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

$$z_{к+1} = z_к^2 + c,\ к = 0, 1, 2, \ldots, (z = x + iy,\ c = a + ib,\ i = \sqrt{-1}),$$

де z_0 – різні точки комплексної площини; c – деяка фіксована комплексна точка. Фрактал Мандельброта одержується за цією ж ітераційною формулою $z_{к+1} = z_k^2 + c,\ к = 0, 1, 2, \ldots$, 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

$$z_{к+1} = f(z_к),\ к = 0, 1, 2, \ldots,$$

де $f : C \to C$ – поліном степеня $n \geq 2$.

Послідовність точок z_k з ростом номера ітерації k демонструє поведінки трьох типів: вона або прямує в нескінченність, або збігається до скінченної точки, або завжди залишається в замкненій множині, здійснюючи періодичні рухи.

Позначимо через f^k k-ту ітерацію функції f, тобто $f^k = f(f(\ldots(f(z))))$. 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, а множина точок $\{\omega, f(\omega), f^2(\omega), \ldots, f^p(\omega)\}$ – 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 поводяться хаотично на J і J, як правило, є фракталом. Зауважимо, що інколи фракталом Жуліа називають усю область, що обмежена множиною $J(f)$.

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 $f^k(z) \to 0$ при $k \to \infty$ і $f^k(z) \to \infty$ при $k \to \infty$, 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. 10.22, *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 $f^k(z) \to z^*$ при $k \to \infty$, якщо $|z|$ достатньо малі, (z^*– нерухома точка, близька до нуля), і $f^k(z) \to \infty$, якщо $|c|$ достатньо велике. Отже, множина Жуліа – це границя між двома типами поведінки, тепер J вже є фракталом, схематично зображеним на рис. 10.22, *б*.

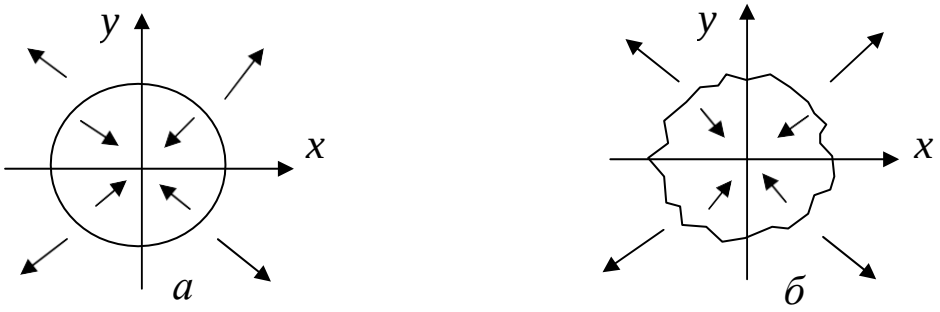


Рис. 10.22. Множини Жуліа: *а* – $c = 0$; *б* – $c \neq 0$

10.4.2. Фрактали Жуліа

Як відомо, фрактали Жуліа на множині комплексних чисел описуються рекурентним співвідношенням $z_{к+1} = z_к^2 + c$, яке після виділення дійсної та уявної частин перепишеться у вигляді

$$x_{k+1} = x_k^2 - y_k^2 + a, \quad y_{k+1} = 2x_k y_k + b, \; k = 0, 1, 2, \ldots \qquad (10.5)$$

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 для зменшення обчислень).

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 підійшла до атрактора. Умовою того, що орбіта прямує до нескінченності, є виконання нерівності $x^2 + y^2 > 4$ при $k < K_{\max}$.

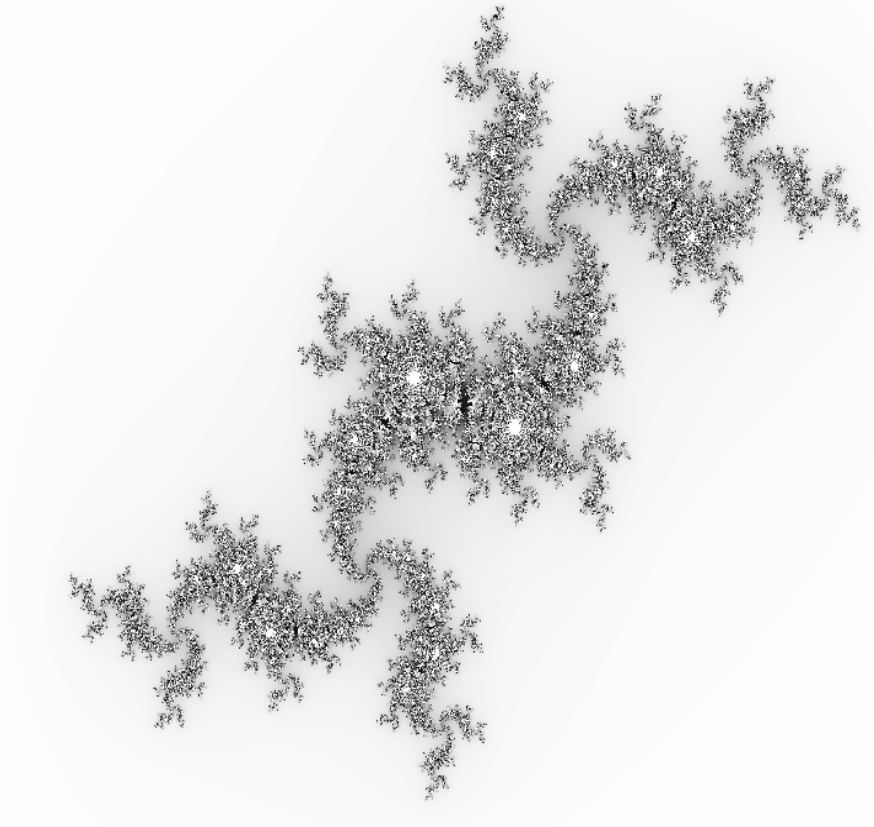


Рис. 10.23. Фрактал Жуліа при $c = -0{,}22 - 0{,}74i$

Якщо при $k = K_{\max}$ $x^2 + y^2 \le 4$ (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. 10.23); 2) $a = 0{,}11$; $b = 0{,}66$. Для того, щоб отримати зображення в растрі, необхідно перераховувати координати x_0, y_0 в піксельні.

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.

10.4.3. Фрактали Мандельброта

Множина всіх точок (a, b), для яких послідовність z_k не прямує до нескінченності (не є розбіжною), складає множину Мандельброта $\mathbb{M}$, тобто точка $c = a+ib$ належить множині Мандельброта тоді і тільки тоді, коли всі точки рекурентної послідовності $z_{k+1}=z_k^2+c$ скінченні й утворюють збіжну послідовність. На рис. 10.24 ця область знаходиться в центрі й зафарбована в чорний колір. Інколи множиною Мандельброта називають тільки межу області $\mathbb{M}$, яка має фрактальну структуру.

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.

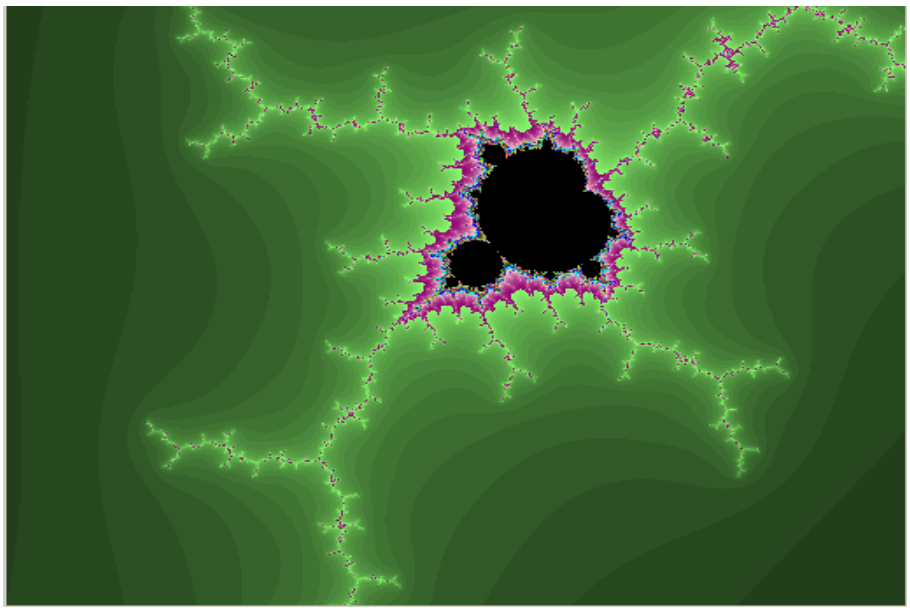
Графічна інтерпретація множини Мандельброта $\mathbb{M}$ дивовижно красива і багатоманітна. Границею цієї множини $\mathbb{M}$ є фрактал Мандельброта, оскільки в границі наявні фрагменти, подібні до усього фрактала (рис. 10.24).

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.

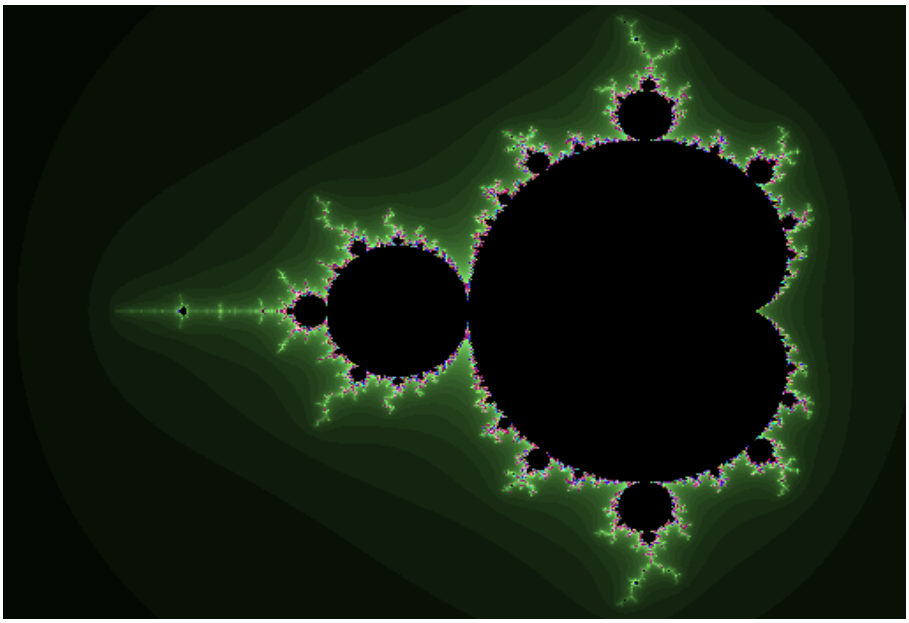
Якщо за максимальну кількість ітерацій K_{max} (K_{max} вибираємо

заздалегідь) орбіта не залишає круга $x^2 + y^2 \le 4$, 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 ($\lim\limits_{k\to\infty} |z_k| = \infty$). Цей факт математично строго доведений.



$a \in [-0{,}19920; -0{,}12954]$, $b \in [1{,}01480; 1{,}06707]$



$a \in [-2{,}4; 0{,}8]$, $b \in [-1{,}2; 1{,}2]$

Рис. 10.24. Фрактал Мандельброта

При цьому обчислення дальніших ітерацій не проводимо, а за значенням k визначаємо колір точки (a, b). Колір точки (a, b) вибираємо, наприклад, за формулою $k \bmod 256$, якщо використовуємо 256 кольорів, або шляхом задання палітри з K_{max} кольорів. Наприклад, можна вибрати палітру, в якій кольори і відтінки від червоного до фіолетового розподілені по діапазону $[0, K_{max}]$ лінійно або за якомось іншим законом.

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 або гірська печера тощо.

Моделювання об'ємних фракталів можна здійснювати, використовуючи гіперкомплексні числа, що задаються набором із дійсного і кількох комплексних чисел.

10.4.4. Фрактали Ньютона

Для знаходження комплексних коренів нелінійного рівняння $f(z) = 0$ можна використовувати алгоритм Ньютона, який має вигляд

$$z_{к+1} = z_к - f(z_к) / f'(z_к),\ к = 0, 1, 2, \ldots \qquad (10.6)$$

Якщо вдало підібрати z_0, то за допомогою попередньої формули одержуємо послідовність $z_0, z_1, z_2, \ldots$, яка збігається до кореня рівняння.

Якщо за $f(z)$ взяти поліном $z^n - a$, то (10.6) матиме вигляд

$$z_{к+1} = \frac{(n-1)z_к^n + a}{nz_к^{n-1}}, \quad к = 0, 1, 2,.. \qquad (10.7)$$

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

$$z_{k+1} = \sqrt[n]{a}\left(\cos\frac{2k\pi}{n} + i\sin\frac{2k\pi}{n}\right),\ k = 0,1,\ldots,n-1.$$

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 притягування нерухомих точок відображення (10.7). Границя областей притягування 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 (зокрема при $|z^n - 1| < 0{,}01$) або при досягненні $k = K_{max}$.

При $a = 1$, $n = 3$ формули (10.7) набувають вигляду

$$z_{к+1} = \frac{2z_к^3 + 1}{3z_к^2}, \quad к = 0, 1, 2, \ldots.$$

Відокремивши дійсну та уявну частини, одержуємо

$$x_{k+1} = \frac{2}{3}x_k + \frac{x_k^2 - y_k^2}{3(x_k^2 + y_k^2)2}, \qquad y_{k+1} = \frac{2}{3}y_k\left(1 - \frac{x_k}{(x_k^2 + y_k^2)2}\right).$$

Піксель, що відповідає значенню (x_k, y_k), зафарбовуємо в колір k mod 256, де k – номер ітерації, починаючи з якого виконується нерівність $|z_k^n - 1| < 0{,}01$ (рис. 10.25).

Рис. 10.25. Фрактал Ньютона при $n = 3$

Отже, моделювання ітераційних процесів обчислювальної математики практично довело фрактальність областей збіжності ітерацій до коренів нелінійних рівнянь.

Можна придумати багато інших ітераційних процедур вигляду

$$z_{к+1} = F(z_к, c), \quad к = 0, 1, 2, \ldots,$$

що породжують різноманітні фрактальні зображення в залежності від вибору параметра c.

10.5. Застосування фракталів

За останні 30 років фрактали стали досить популярними. Дослідження фракталів відкриває широкі перспективи як у галузі чистої математики, так і в практичному застосуванні. Фрактали 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). Фрактальна графіка незамінна при ілюстрації складних неевклідових об'єктів, а також у живописі, архітектурі, поезії, музиці.

Структури, що схожі на фрактали, можна знайти в навколишньому середовищі: контури хмар, гір, морських берегів, ландшафти, рослинність, турбулентні потоки в рідинах, тріщини в породах, зображення структур деяких речовин, отриманих під мікроскопом, тощо.

Більшість природних об'єктів фрактальні й саме за допомогою фракталів їх можна добре зобразити. Фрактали описують природні форми більш витончено і точніше, ніж об'єкти евклідової геометрії.

Уведення понять фрактала і фрактальної геометрії дозволяє виділити приховані закономірності в будові природних об'єктів. Фрактальна графіка має когнітивні функції. За допомогою фракталів можна одержати нове знання, тобто з'являється образне мислення, більш ефективне, ніж мислення, що опирається на символи.

Останнім часом методи ітеративних функцій застосовуються не тільки для створення зображень, але й для ефективного стиску графічних зображень при записі у файли.

Графічні зображення кодуються кількома простими перетвореннями, тобто коефіцієнтами цих перетворень. Наприклад, якщо деяке зображення закодуємо двома афінними перетвореннями вигляду

$$x' = ax + by + c, \qquad y' = dx + ey + f,$$

то ми його можемо однозначно визначити за допомогою 12 коефіцієнтів.

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 цифрового зображення підібрати

коефіцієнти перетворення. Але проблема в тому, що досить важко знайти ці перетворення.

Такі методи стиску графічної інформації називають методами фрактального стиску. Ці алгоритми дозволяють стискати інформацію в сотні разів без помітного погіршення зображення.

Контрольні запитання та завдання

1. Дайте визначення фрактала за Мандельбротом.
2. Назвіть типи фракталів. Опишіть їх властивості.
3. Які фрактали називаються конструктивними? Наведіть приклади.
4. Що таке генератор-ламана?
5. Які фрактали називаються динамічними? Наведіть приклади.
6. Опишіть принципи побудови геометричних фракталів, зокрема фракталів Коха, Леві, Мінковського.
7. Опишіть структуру зіркових фракталів.
8. Які афінні перетворення використовуються при побудові зіркових фракталів?
9. Що таке система ітераційних функцій?
10. Запишіть алгоритм побудови фрактала „гілка папороті”.
11. Поясніть властивість рекурентності алгоритму побудови фрактала.
12. Як описуються множини Жуліа та Мандельброта? Яка між ними різниця? Як їх побудувати?
13. Опишіть процес побудови фракталів Жуліа та Мандельброта на комп’ютері.
14. Опишіть процес побудови фракталів Ньютона на екрані комп’ютера.
15. За якою ознакою зупиняють обчислювальний процес при побудові фракталів Ньютона?
16. Назвіть області застосування фракталів та об’єкти, що володіють фрактальними властивостями.
17. Як, на вашу думку, можна будувати стохастичні фрактали?

Вправи і задачі для самостійного виконання

1. Модифікувати алгоритм побудови фрактала „гілка папороті” з урахуванням товщини стовбура.
2. Розробити алгоритм побудови різнокольорового фрактала „гілка папороті”.
3. Написати програму побудови фрактала, який одержується за ітераційною формулою $z_{к+1} = i(z_к^2 + 1)$, $к = 0, 1, 2, \ldots$

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. Придумати одновимірні комплексні відображення, які, на вашу думку, можуть породити цікаві динамічні фрактали.
6. Побудувати систему ітеративних функцій (IFS) для трикутника Серпінського. ***Вказівка***. *Задати три вершини правильного трикутника в однорідних координатах* $p_i = (x_i, y_i, 1)$ *і підібрати матриці* C_i *, які перетворять вершини великого трикутника у вершини малих трикутників.*
7. Написати фрагмент програми, яка для систем ітеративних функцій буде у випадковий спосіб вибирати функцію зі списку *n* функцій.
8. Побудувати фрактали Жуліа та Мандельброта у випадку коли *z* – гіперкомплексне число.

Розділ 11. Моделювання 2D/3D-перетворень

Конструювання та виконання різноманітних дій із 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 і для створення зображень шляхом перетворень з однієї системи кординат в іншу. При цьому використовуються двовимірні та тривимірні системи координат.

У комп'ютерній графіці все, що стосується двовимірного випадку, позначають символом 2D, а тривимірного – 3D (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

$$x' = ax + by + m, \quad y' = cx + dy + n, \tag{11.1}$$

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 $M'(x', y')$.

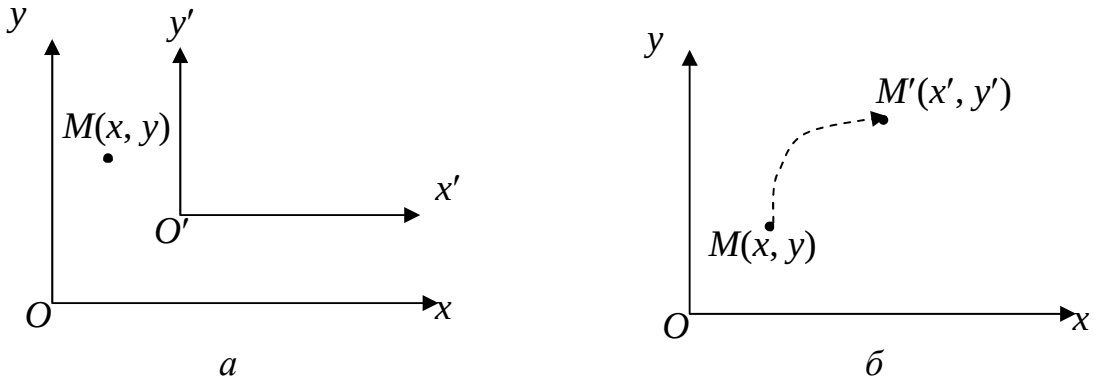


Рис. 11.1. Перетворення координат

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

$$x' = x + m, \quad y' = y + n; \tag{11.2}$$

- зсув системи координат на вектор (m, n) (рис. 11.2, *б*) задається формулами

$$x' = x - m, \quad y' = y - n.$$

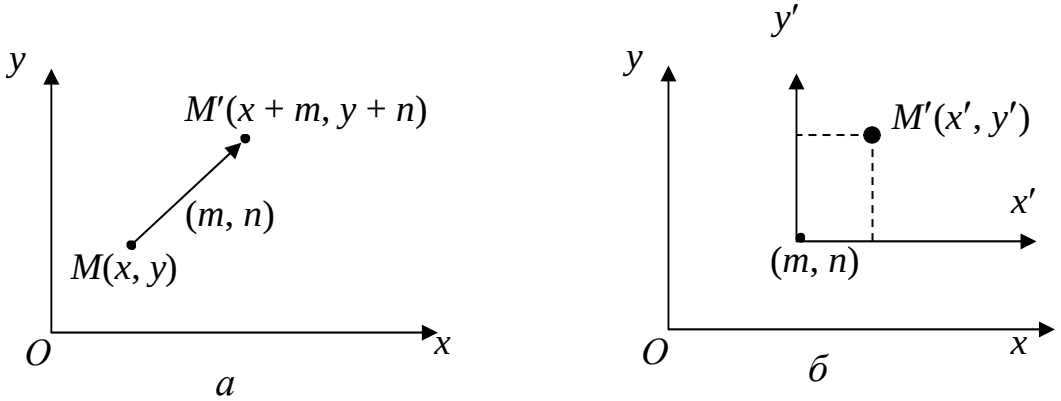


Рис. 11.2. Перетворення зсуву

- поворот точки відносно початку координат на кут φ проти годинникової стрілки (рис. 11.3, *а*) описується формулами

$$x' = x\cos\varphi - y\sin\varphi, \quad y' = x\sin\varphi + y\cos\varphi; \tag{11.3}$$

- 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

$$x' = ax, \quad y' = dy. \tag{11.4}$$

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).

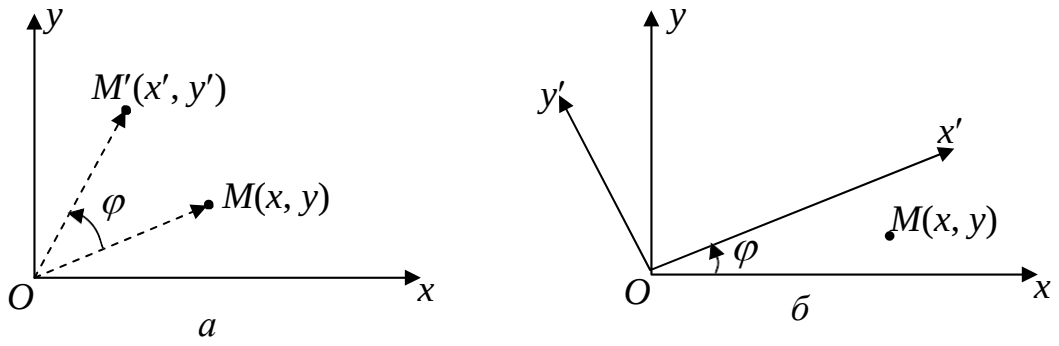


Рис. 11.3. Перетворення повороту

Для ефективного використання формул (11.2) – (11.4) у задачах КГ переходять до матричного запису цих формул. Наприклад, для перетворення (11.3) маємо:

$$X' = X \cdot R = (x\ y)\begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix}. \tag{11.5}$$

Тобто перетворення повороту точки відносно початку координат у додатному напрямку на довільний кут φ задається матрицею

$$R = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix}, \qquad \det R = 1 \neq 0.$$

Зауважимо, що повороти вважаються додатними, якщо вони здійснюються проти годинникової стрілки.

У матричному вигляді можна записати й 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 досить важливий 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 виг- ляді (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).

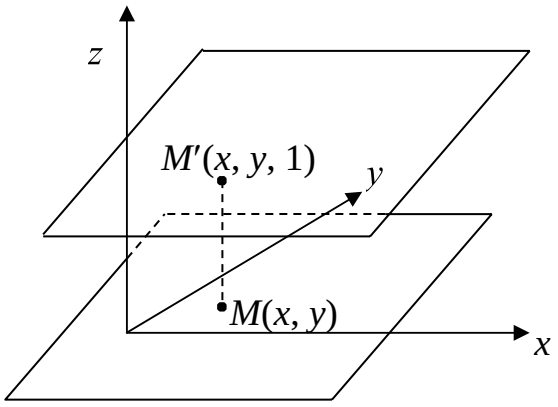


Рис. 11.4. Однорідні координати

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

$$(x_1, y_1, h_1) + (x_2, y_2, h_2) \sim \left(\frac{x_1}{h_1}, \frac{y_1}{h_1}, 1\right) + \left(\frac{x_2}{h_2}, \frac{y_2}{h_2}, 1\right) =$$

$$\left(\frac{x_1}{h_1} + \frac{x_2}{h_1}, \frac{y_1}{h_2} + \frac{y_2}{h_2}, 1\right) \sim (x_1 h_1 + x_2 h_1, y_1 h_2 + y_2 h_2, h_1 h_2);$$

- множення вектора на скаляр

$$a(x, y, h) \sim a\left(\frac{x}{h}, \frac{y}{h}, 1\right) = \left(\frac{ax}{h}, \frac{ay}{h}, 1\right) \sim (ax, ay, ah).$$

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 оперувати точкою, що знаходиться в нескінченності, уникаючи переповнення розрядної сітки ЕОМ, завдяки нормалізації чисел.

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 $X' = X \cdot M$, тобто

$$(x' \;\; y' \;\; 1) = (x \;\; y \;\; 1)\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{pmatrix}, \qquad (11.7)$$

де елементи a, b, c, d визначають масштабування і поворот, а m, n – сумарний зсув. Отже, всі перетворення зсуву, масштабування, повороту відносно початку координат та їх композиції в однорідних координатах мають однакову форму – це добуток вектора вихідних координат на матрицю перетворення.

Структура матриці загального перетворення дозволяє записати ці перетворення у скалярній формі

$$x' = ax + cy + m, \qquad y' = bx + dy + n.$$

Випишемо матриці основних елементарних перетворень:

- матриця повороту (rotation) точки відносно початку координат у додатному напрямку:

$$R = R(\varphi) = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}; \qquad (11.8)$$

- матриця розтягу (стиску) (dilation) відносно початку координат:

$$D = D(a,d) = \begin{pmatrix} a & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{pmatrix}; \qquad (11.9)$$

- матриця перенесення (translation) точки на вектор (m, n):

$$T = T(m,n) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{pmatrix}; \qquad (11.10)$$

- матриця дзеркального відображення (reflection) відносно осі x:

$$Ref = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \qquad (11.11)$$

Для простоти запису аргументи матриць іноді опускаються. 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).

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 відповідають елементарним перетворенням.

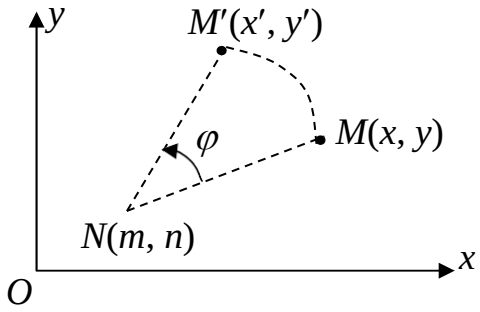


Рис. 11.5. Поворот точки M у додатному напрямку відносно точки N

Можна показати, що для двох послідовних операцій: перенесення-перенесення, масштабування-масштабування, обертання-обертання має місце комутативність. У цих випадках можна змінювати послідовність множення матриць. Розглянемо приклади складніших 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
$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{pmatrix};$$
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 виписана в (11.10).

Отже, для знаходження результуючого повороту точки $M(x, y)$ відносно точки $N(m, n)$ потрібно перемножити матриці T^{-}, R, T за вказаним порядком.

У результаті одержимо шукане перетворення у вигляді
$$(x' \ y' \ 1) = (x \ y \ 1)\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{pmatrix}\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{pmatrix} =$$
$$= (x \ y \ 1)\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ -m\cos\varphi + n\sin\varphi + m & -m\sin\varphi - n\cos\varphi + n & 1 \end{pmatrix}.$$

Зокрема, поворот точки на 90^0 у додатному напрямку відносно точки $N(m, n)$ виконується перетворенням
$$(x' \ y' \ 1) = (x \ y \ 1)\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ m+n & n-m & 1 \end{pmatrix}.$$

Зауважимо, що для генерації кадрів зображень об'єкта, коли наступний відрізняється від попереднього обертанням на кілька градусів, доцільно застосовувати спрощені перетворення. Рівняння (11.3) можна спростити, скориставшись тим, що кут φ малий і cos φ приблизно дорівнює одиниці. Тоді рівняння (11.3) набудуть вигляд
$$x' = x - y\sin\varphi, \quad y' = x\sin\varphi + y.$$

Кращу апроксимацію можна одержати, якщо у другому рівнянні використати x' замість x:

$$x' = x - y\sin\varphi, \qquad y' = x' \sin\varphi + y.$$

Треба врахувати, що в разі багаторазового використання спрощених формул їх похибка буде накопичуватися. Щоб зменшити вплив похибок можна зберігати початкові координати об'єкта (x, y) і після кожного обертання об'єкта на 360° для наступного обертання брати початкові дані.

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 має вигляд (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 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

$$\begin{pmatrix} x_1' & y_1' & 1 \\ x_2' & y_2' & 1 \\ x_3' & y_3' & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 6 & 1 & 1 \\ 6 & 4 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}\begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}\times$$

$$\times\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 1 \\ 0 & 7 & 1 \\ 3 & 7 & 1 \end{pmatrix}.$$

Отже, декартові координати вершин симетричного трикутника $A'B'C'$ мають значення $A'(0;\ 2)$, $B'(0;\ 7)$, $C'(3;\ 7)$.

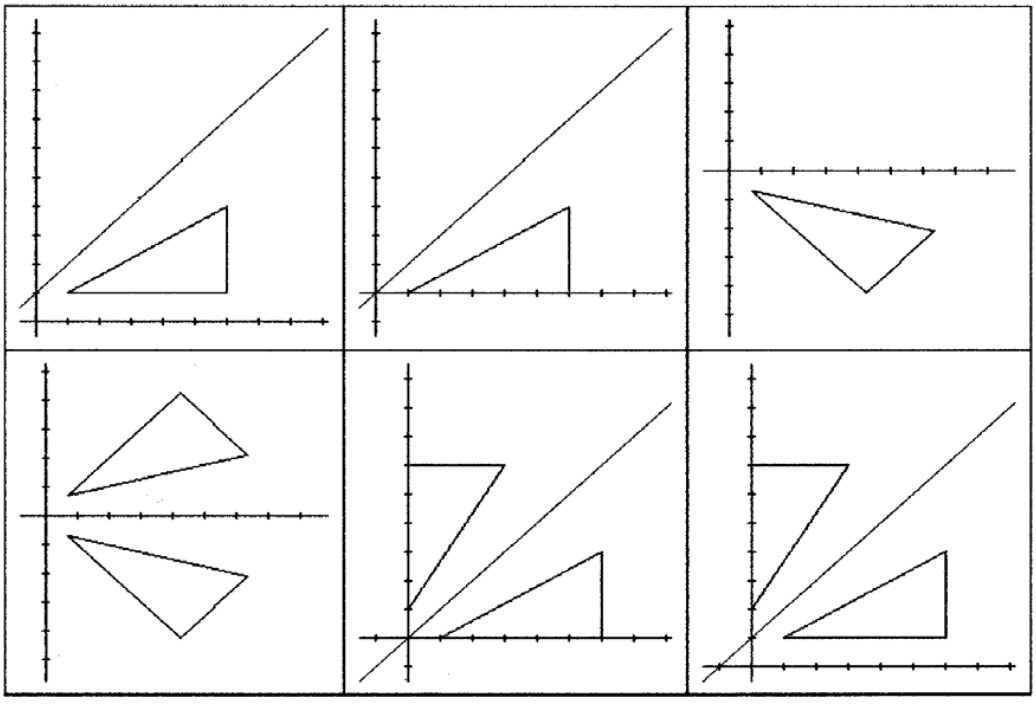


Рис. 11.6. Побудова симетричного трикутника

11.2. 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

$$(x'\ y'\ z'\ 1) = (x\ y\ z\ 1)\ A,$$

де A – матриця афінного перетворення, для якої $\det A \neq 0$.

Зауважимо, що у випадку $\det A = 0$ перетворення A називається *проєктивним*.

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*.

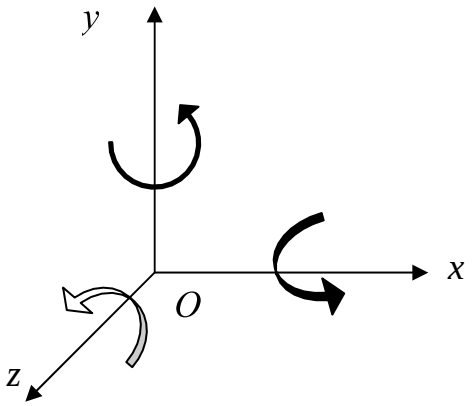


Рис. 11.7. Додатні напрями поворотів

Поворот простору відносно осі *z* на кут χ проти годинникової стрілки відповідає додатному повороту в площині *xy* (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

$$R_z(\chi) = \begin{pmatrix} \cos\chi & \sin\chi & 0 & 0 \\ -\sin\chi & \cos\chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (11.12)$$

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 ($x \to y$, $y \to z$, $z \to x$), тому переставляючи рядки і стовпці матриці (11.12) одержуємо матрицю повороту відносно осі x

$$R_x(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (11.13)$$

Аналогічно матриця повороту точки відносно осі y на кут ψ проти годинникової стрілки має вигляд

$$R_y(\psi) = \begin{pmatrix} \cos\psi & 0 & -\sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (11.14)$$

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

$$D = \begin{pmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (11.15)$$

Якщо $\alpha = \beta = \gamma = s$, то маємо однорідне масштабування: при $s > 1$ – однорідне розтягування, при $s < 1$ – однорідний стиск.

в) У тривимірному просторі дзеркальне відображення відбувається відносно площин. При відображенні відносно координатної площини xy змінюється тільки значення z-координати вектора об’єкта, а саме, вони змінюють тільки знак.

Матриці дзеркального відображення відносно координатних площин *xy*, *yz*, *zx* мають вигляд

$$M_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad M_{yz} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$M_{xz} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \qquad (11.16)$$

г) Перенесення простору на вектор (*l*, *m*, *n*) описується матрицею

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{pmatrix}, \qquad (11.17)$$

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 самий кут – спочатку відносно осі *x*, а потім відносно осі *y* і навпаки. Використовуючи матриці (11.13) та (11.14) при $\psi = \varphi$, одержимо

$$R_{xy} = R_x(\varphi) \cdot R_y(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times$$

$$\times \begin{pmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ \sin^2\varphi & \cos\varphi & \cos\varphi\sin\varphi & 0 \\ \cos\varphi\sin\varphi & -\sin\varphi & \cos^2\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

З іншого боку, поворот спочатку відносно осі *y*, а потім відносно осі *x* з кутом $\psi = \varphi$ дає матрицю

$$R_{yx} = R_y(\varphi) \cdot R_x(\varphi) = \begin{pmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times$$

$$\times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\varphi & \sin^2\varphi & -\cos\varphi\sin\varphi & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ \sin\varphi & -\cos\varphi\sin\varphi & \cos^2\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

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 2*D*-випадку, для 3*D*-простору правильна наступна теорема.

Теорема. *Довільне афінне перетворення в 3D-просторі можна подати у вигляді послідовності елементарних перетворень а) – г).*

Нагадаємо, що загальна форма афінних перетворень у просторі задається системою рівнянь

$$x' = ax + by + ez + m,$$
$$y' = cx + dy + fz + n,$$
$$z' = gx + hy + pz + l,$$

де (x', y', z') – точки, що одержуються в результаті цих перетворень.

Розглянемо приклади складніших перетворень, які подамо у вигляді композиції елементарних перетворень.

11.3. Приклади складніших 3*D*-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^{-} – обернена матриця до матриці переміщення.

Використовуючи формули (11.13), (11.17), одержуємо

$$T \cdot R_x \cdot T^{-} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -m & -n & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & m & n & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & m(1-\cos\varphi)+n\sin\varphi & n(1-\cos\varphi)-m\sin\varphi & 1 \end{pmatrix}.$$

Вправа. Знайти координати точки P', яка одержується поворотом точки $P(1, 1, 1)$ на кут 45^0 відносно прямої, що паралельна осі Ox і проходить через точку (1, 2, 2).

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).*

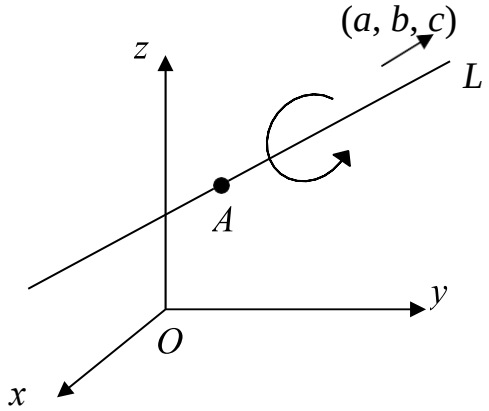


Рис. 11.8. Обертання відносно довільної прямої

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

$$T^{-} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{pmatrix}.$$

Унаслідок цього 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 визначати кут повороту φ відносно осі *x*, спершу спроєктуємо на площину *Oyz* напрямний вектор (*a*, *b*, *c*) (рис. 11.9, *а*). Спроєктований вектор має координати (0, *b*, *c*). Довжина його проєкції $d = \sqrt{b^2 + c^2}$.

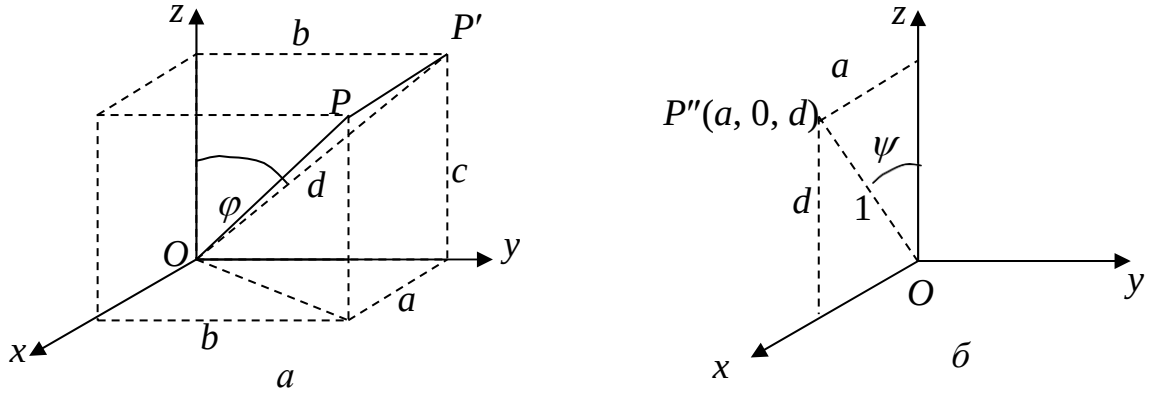


Рис. 11.9. Суміщення вектора *OP* з віссю *z*

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

$$\sin\psi = a,\ \cos\psi = d.$$

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

$$M = \begin{pmatrix} a^2 + (1-a^2)\cos\theta & a(1-\cos\theta)b + c\sin\theta & a(1-\cos\theta)c - b\sin\theta & 0 \\ a(1-\cos\theta)b - c\sin\theta & b^2 + (1-b^2)\cos\theta & b(1-\cos\theta)c + a\sin\theta & 0 \\ a(1-\cos\theta)c + b\sin\theta & b(1-\cos\theta)c - a\sin\theta & c^2 + (1-c^2)\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Розглядаючи довільні перетворення в просторі, в результаті їх комбінування одержуємо матриці вигляду

$$A = \left(\begin{array}{ccc:c} \alpha_1 & \alpha_2 & \alpha_3 & 0 \\ \beta_1 & \beta_2 & \beta_3 & 0 \\ \gamma_1 & \gamma_2 & \gamma_3 & 0 \\ \hdashline l & m & n & 1 \end{array}\right).$$

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 багатогранника. Опуклий багатогранник однозначно визначається набором своїх вершин $X_i(x_i, y_i, z_i)$, що утворюють матрицю X. Далі на матрицю X діємо невиродженою матрицею A і одержуємо набір вершин $X' = X \cdot A$ нового багатогранника, який є образом вихідного.

Вправа. Знайти координати точки P', яка одержується поворотом точки $P(2, 2, 2)$ на кут 30^0 відносно лінії, що проходить через початок координат і має напрямний вектор (1, 2, 2).

Приклад 6. *Побудувати матрицю дзеркального відображення точки P відносно площини Q, що проходить через точку $A(x_0, y_0, z_0)$ з нормальним вектором (a, b, c).*

Перетворення з допомогою матриць M_{xy}, M_{yz}, M_{xz} з (11.16) задають відображення відносно координатних площин $z = 0$, $x = 0$, $y = 0$, відповідно. Але на практиці часто виникає задача дзеркального відображення об'єкта відносно довільної площини. Це можна зробити за допомогою комбінації матриць зсуву і обертання. Один з можливих методів полягає у виконанні таких операцій:

1) Перенесення простору так, щоб точка $A(x_0, y_0, z_0)$ попала в початок координат. Це можна зробити за допомогою матриці $T^{-}(-x_0,-y_0,-z_0)$.
2) Поворот вектора нормалі (a, b, c) до площини відображення так, щоб його напрям збігся з додатним напрямом осі OZ. Це здійснюють матриці R_x і R_y^- з *прикладу 5*. Тепер площина відображення буде збігатися з координатною площиною $z = 0$.
3) Відобразимо об'єкт відносно координатної площини $z = 0$ (матриця M_{xy}).
4) Далі виконуємо обернені перетворення.

Тоді загальне перетворення, тобто відображення простору відносно довільної площини, опишеться матрицею

$$M = T^{-} \cdot R_x \cdot R_y^{-} \cdot M_{xy} \cdot R_y \cdot R_x^{-} \cdot T.$$

Знаходження симетричної точки виконуємо за формулою $P' = P \cdot M$.

Вправа. Знайти дзеркальне відображення точки $P(2, 1, 2)$ відносно площини, що проходить через три точки $C(3, 1.5, 2)$, $D(2.5, 2, 2)$, $B(3, 2, 1.5)$.

11.4. 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, $i = 1, 2, \ldots, n$. 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.

В цьому методі матрицю складних перетворень C знаходимо з системи рівнянь $X' = X \cdot C$, а саме $C = X^{-1} \cdot X'$.

Зауважимо, що застосування методу парних точок, в якому задаються лише початкові та кінцеві стани об'єктів, дозволяє створювати алгоритми складних перетворень без відтворення проміжних елементарних перетворень, побудови їх матриць, урахування особливих випадків орієнтації об'єктів відносно системи

координат та ін. А це важливо для коректної роботи програм візуалізації.

Приклад 7. *Знайти матрицю перетворення трикутника* $P_1P_2P_3$ *у трикутник* $P_1'P_2'P_3'$ *(рис. 11.10).*

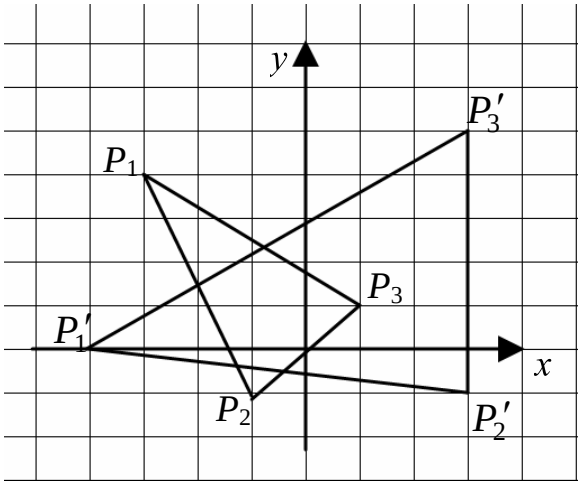


Рис. 11.10. Перетворення трикутників

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 $P_1'P_2'P_3'$ – 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 $P_1'P_2'P_3'$ і переконатися, що такий спосіб побудови матриці C у цьому разі складніший.

Приклад 8. *Знайти матрицю C пропорційного відображення прямокутного вікна* $[x_{min} \div x_{max}, y_{min} \div y_{max}]$ *у площині* xy *у вікно екранної системи координат* $[left \div right,\ up \div down]$ (рис.11.11).

На початку визначимо, чи прямокутні вікна пропорційні. Для цього обчислимо коефіцієнти прямокутності вікон

$$k_1 = \frac{y_{max} - y_{min}}{x_{max} - x_{min}}, \quad k_2 = \frac{up - down}{right - left}.$$

Якщо $k_1 = k_2$, то вікна пропорційні й тоді матрицю C можна просто знайти методом парних точок, як це зроблено в прикладі 7. Для цього виберемо три відповідних вершини прямокутних вікон p_1, p_2, p_3 та p_1', p_2', p_3' (рис. 11.11)

$p_1 = (x_{min}\ y_{min}\ 1) \rightarrow p_1' = (left\ down\ 1)$,

$p_2 = (x_{min}\ y_{max}\ 1) \rightarrow p_2' = (left\ up\ 1)$,

$p_3 = (x_{max}\ y_{max}\ 1) \rightarrow p_3' = (right\ up\ 1)$.

Маємо

$$C = \begin{pmatrix} x_{min} & y_{min} & 1 \\ x_{min} & y_{max} & 1 \\ x_{max} & y_{max} & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} left & down & 1 \\ left & up & 1 \\ right & up & 1 \end{pmatrix}.$$

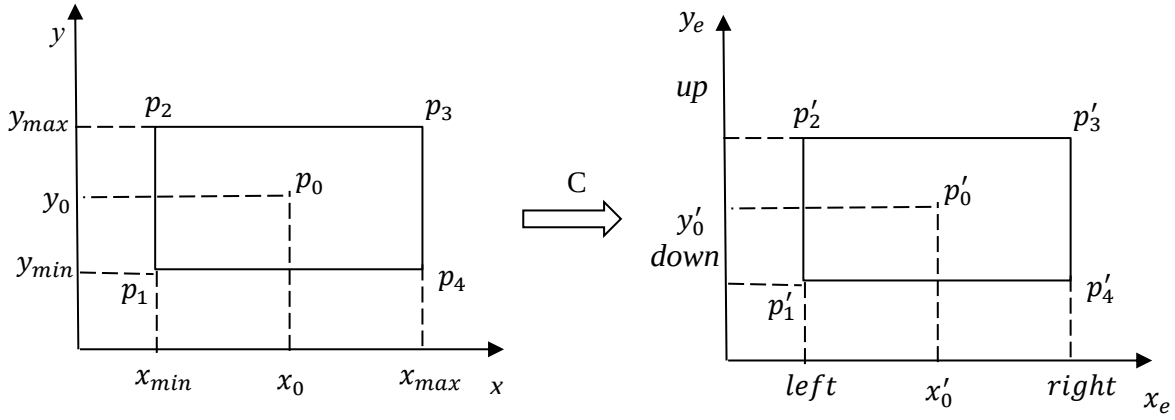


Рис. 11.11. Відображення прямокутних вікон

Масштабний коефіцієнт відображення вікон при цьому дорівнює

$$\frac{right - left}{x_{max} - x_{min}} = \frac{up - down}{y_{max} - y_{min}}.$$

У випадку непропорційності вікон ($k_1 \neq k_2$) матрицю C знайдемо методом декомпозиції рухів, виконавши ланцюжок елементарних перетворень.

1. Перенесення центра p_0 прямокутного вікна в початок координат системи xy матрицею T_1, де

$$T_1 = T(-x_0, -y_0), \text{ де } x_0 = \frac{x_{min} + x_{max}}{2},\ y_0 = \frac{y_{min} + y_{max}}{2}.$$

2. Пропорційне масштабування цього вікна по осях координат матрицею

$$D = diag(\mu_1, \mu_2, 1), \text{ де } \mu_1 = \frac{right - left}{x_{max} - x_{min}}, \quad \mu_2 = \frac{up - down}{y_{max} - y_{min}}.$$

3. Перенесення центра вікна з початку координат у точку p_0' матрицею

$$T_2 = T({x_0}', {y_0}'),\ \text{де}\ {x_0}' = \frac{left+right}{2},\ {y_0}' = \frac{down+up}{2}.$$

Результуюча матриця відображення прямокутних вікон має вигляд

$$C = T_1 D T_2 = \begin{pmatrix} \mu_1 & 0 & 0 \\ 0 & \mu_2 & 0 \\ {x_0}' - x_0\mu_1 & {y_0}' - y\mu_2 & 1 \end{pmatrix}.$$

Звідси безпосередньо знаходимо рівняння для обчислення координат з вихідної системи координат xy в екранну

$$x_e = {x_0}' + \mu_1(x - x_0),\ \ y_e = {y_0}' + \mu_2(y - y_0).$$

Обернена матриця C^{-1} відображає прямокутну область з екранної системи координат у вихідну систему координат xy.

Зробимо ще декілька зауважень.

Зауваження 1. Якщо екранна система координат має початок координат у лівому верхньому куті екрана і вісь y направлена вниз, то замість μ_2 потрібно у наведених вище викладеннях всюди поставити протилежне значення $-\mu_2$.

Зауваження 2. Для того, щоб вихідне вікно відображалося в екранне без спотворень, тобто з однаковим коефіцієнтом пропорційності по двох осях, потрібно підставити замість μ_1 і μ_2 значення μ, де

$$\mu = \begin{cases} \mu_1, & \text{якщо } k_2 \geq k_1, \\ \mu_2, & \text{якщо } k_2 < k_1. \end{cases}$$

При цьому екранне вікно буде заповнено максимально повно й симетрично відносно центра p_0'.

***Приклад 9**. Розглянути задачу про зміну просторової орієнтації системи координат, як це зображено на рис. 11.12, 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 $\boldsymbol{V}(a, b, c)$ (рис. 11.12) (розміщення двох інших осей не регламентується).

Розв'язок задачі досягається за допомогою двох перетворень:

1. Перенесення початку системи координат *xyz* у точку *O*'. Це перетворення описується матрицею $T(-m, -n, -l) = T^{-}$.

2. Поворот системи координат так, щоб вісь z мала напрям вектора $\boldsymbol{V}$, що виходить уже з початку перенесеної системи координат. Позначимо матрицю цього перетворення $A(V)$, тоді матриця повного перетворення $C = T^{-} \cdot A(V)$.

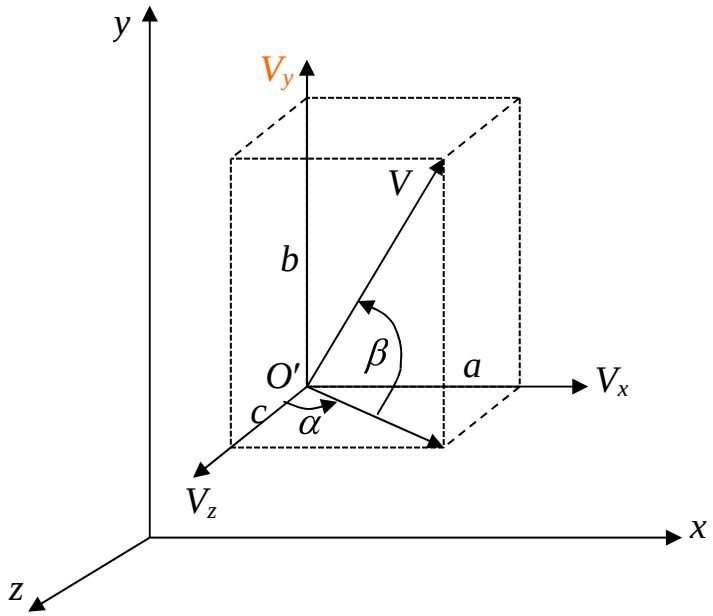


Рис. 11.12. Зміна просторової орієнтації системи координат

Виразимо матрицю $A(V)$ через матриці елементарних поворотів.

- Перший поворот системи координат виконуємо відносно осі y проти годинникової стрілки на кут α, для якого

$$\cos\alpha = \frac{c}{d},\ \sin\alpha = \frac{a}{d},\ d = \sqrt{a^2 + c^2} \neq 0,\ \alpha \in \left[-180^0, 180^0\right].$$

Це перетворення виконує матриця повороту $R_y(-\alpha)$.

- Друге перетворення – 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 $\boldsymbol{V}$ обчислюється за формулою

$$A(V) = R_y(-\alpha) \cdot R_x(\beta).$$

Якщо $d = 0$ (тобто $\boldsymbol{V} \| Oy$), то кут α невизначений і суміщення осі z із вектором $\boldsymbol{V}$ досягається не за два, а за одне обертання системи координат відносно осі x матрицею

$$A(V) = R_x(sign(V_y)90^0).$$

Зауваження 3. Пряма підстановка

$$\cos\alpha = \frac{c}{d}, \qquad \sin\alpha = \frac{a}{d}, \quad \cos\beta = \frac{d}{|V|}, \quad \sin\beta = \frac{b}{|V|}$$

у матриці поворотів $R_y(-\alpha)$, $R_x(\beta)$ та їх множення призводить 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 $\boldsymbol{V}$, є прообразом z^0. У випадку $\boldsymbol{V} \neq z^0$ виберемо прообрази y^0 і x^0 на кінцях векторів нормалі $\boldsymbol{N} = z^0 \times \boldsymbol{V}$ (× – означає векторний добуток) і бінормалі $\boldsymbol{W} = \boldsymbol{N} \times \boldsymbol{V}$. 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

$$C^{-1} = \begin{pmatrix} N \times V \\ N \\ V \end{pmatrix}^{-1}.$$

Отже, метод парних точок дозволяє знайти матрицю повного перетворення системи координат без розкладу на проміжні перетворення.

Приклад 10. *Побудуємо ще в 3D-просторі коло радіусом r з центром у точці $p_0 \in R^3$ із нормаллю N до площини кола.*

За методом розкладу сформуємо елементарні рухи, що дозволять отримати довільне розміщення кола в просторі:

- одиничне коло $r(t)=(\cos t, \sin t, 0)$, що лежить в площині (xy) з нормаллю $z^0=(0, 0, 1)$ масштабуємо до радіуса r;
- сумістимо нормаль $z^0=(0, 0, 1)$ з вектором нормалі N до площини кола;
- перенесемо центр кола (0, 0, 0) в точку p_0.

Виконавши ці перетворення, одержимо рівняння кола в просторі у векторній формі

$$p(t)=r(\cos t \ \sin t \ 0)A^{-1}(N)+ p_0,$$

де $A^{-1}(N)$ – це обернена матриця (оскільки повертається нормаль, а не координатна вісь) перетворення системи координат $A(V)$ з аргументом $V=N$, що обчислена в *прикладі 9*.

Контрольні запитання 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. Виписати матриці елементарних перетворень у просторі.
6. Які перетворення в просторі є комутативними/некомутативними?
7. Як побудувати матрицю повороту відносно довільної осі? Які повороти називаються додатними?
8. Назвати та порівняти методи знаходження матриць складних афінних перетворень.
9. Сформулювати 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. Довести, 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 $D=D(a_1a_2, b_1b_2)$.
5. Побудувати матрицю такої послідовності рухів точки:
   a) зсув по осі x на Δx;
   b) поворот відносно попереднього положення точки на кут α;
   c) зсув по осі y на Δy.
6. Довести, що відображення відносно прямої в площині xy, що проходить через початок координат, еквівалентне повороту відносно початку координат. Розглянути випадок прямої $y = x$.

7. Побудувати матрицю симетричного відображення відносно прямої, що проходить через точку $P(m, n)$ із напрямним вектором $\boldsymbol{V}$.
8. Побудувати матрицю симетричного відображення відносно точки $P(m, n)$.
9. Побудувати матрицю симетричного відображення відносно прямої 1) $x=a$; 2) $y=b$.
10. Побудувати матрицю загального повороту в 3D-просторі відносно трьох координатних осей, тобто матрицю $R_{xyz} = R_z(\alpha) \cdot R_y(\alpha) \cdot R_x(\gamma)$. Знайти обернену матрицю цього перетворення.
11. Задано три точки P_1, P_2, P_3 у просторі. Побудувати матрицю перетворення, в результаті якого вектор P_1P_2 буде направлений уздовж осі z, а вектор P_1P_3 лежатиме в площині $x = 0$.
12. 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 призми, повернутої відносно ребра CB на кут 45^0.
13. Побудувати матрицю масштабування в 3D-просторі відносно точки $P(m, n, l)$.
14. Побудувати матрицю симетричного відображення точки відносно точки $P(m, n, l)$.
15. Побудувати матрицю симетричного відображення точки відносно прямої лінії, що проходить через точку $P(m, n, l)$ з напрямним вектором $\boldsymbol{V}$.
16. Записати векторну форму рівняння кола в просторі через координатні функції.
17. Використовуючи афінні перетворення, побудувати в 3D-просторі рівняння кола радіусом r із центром у точці p_0 і нормальним вектором N до площини кола. ***Вказівка.*** Виконати над колом одиничного радіуса $c(t) = (\cos(t), \sin(t)) \in (x,y)$ з нормаллю $z^0=(0, 0, 1)$ три перетворення:
    - масштабування одиничного кола до радіуса r;
    - суміщення нормалі z^0 із вектором N;
    - перенесення центра кола $(0, 0, 0)$ у точку p_0.

Розділ 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 проєктування. Ці прямі проходять через кожну 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 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. Відмінність між ними визначається співвідношенням між центром проєкції і проєкційною площиною.

У випадку *центрального проєктування* всі прямі-проєктори виходять з однієї точки – центру проєктування, 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 проєктування знаходиться на нескінченності, то всі проєктори паралельні між собою і при проєктування одержується *паралельна проєкція* (рис. 12.1, *а*).

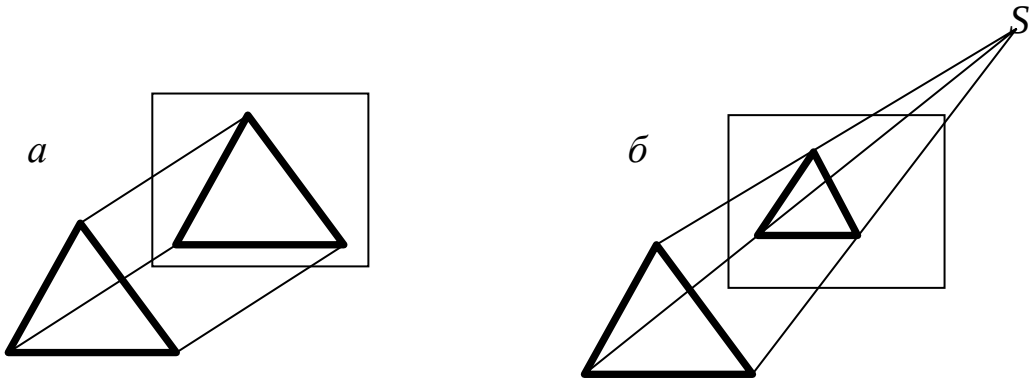


Рис. 12.1. Паралельна і центральна проєкції

До важливих властивостей методу проєктування належить ступінь достовірності сприйняття об’єкта спостерігачем за його проєкціями, тобто розпізнавання об’єкта за його плоским зображенням. Призначення того чи іншого методу проєктування полягає в змен-

шенні невизначеності найбільш важливих властивостей об’єкта (будь-яка проєкція завжди містить у собі похибку, оскільки 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 реалістичних зображеннь тривимірних об’єктів.

Центральна проєкція породжує візуальний ефект, аналогічний до того, який виникає при спостереженні людиною, тобто досягається деякий ступінь реалістичності завдяки ефекту перспек

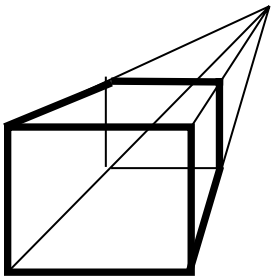


Рис. 12.2. Точка збігу

тивного вкорочування. При центральному проектуванні 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 і розміри об’єкта.

Центральні проєкції довільної сукупності паралельних прямих, не паралельних до проєкційної площини, не паралельні і 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. Наприклад, проєкції каркасних кубів мають одну, дві або три точки збігу і, відповідно, називаються *одноточковими*, *двоточковими* або *триточковими* перспективами.

Паралельні проєкції розділяються на типи в залежності від співвідношень між напрямом проєктування і нормаллю до КП.

У ортографічних проєкціях ці напрями співпадають, у косокутніх вони не співпадають. Найбільш поширеними видами ортографічних проєкцій, в яких КП перепендикулярна координатним осям, є вигляд спереду, вигляд зверху і вигляд збоку.

У разі аксонометричних проєкцій КП не перпендикулярні координатним осям, тому на них відображаються декілька сторін об’єкта. При аксонометричних проєкціях зберігається паралель-

ність прямих, а відстані можна вимірювати вздовж координатних осей з різними масштабними коефіцієнтами.

В косокутній проєкції картинна площина перпендикулярна координатній осі, а напрям проєктування не перпендикулярний до КП

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 проєктуванні КП збігається з однією із координатних площин або паралельна їй. Якщо проєктори паралельні осі z, то точка (x, y, z) проєктується в точку $(x, y, 0)$. Матриця ортогонального проєктування вздовж осі z має вигляд

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

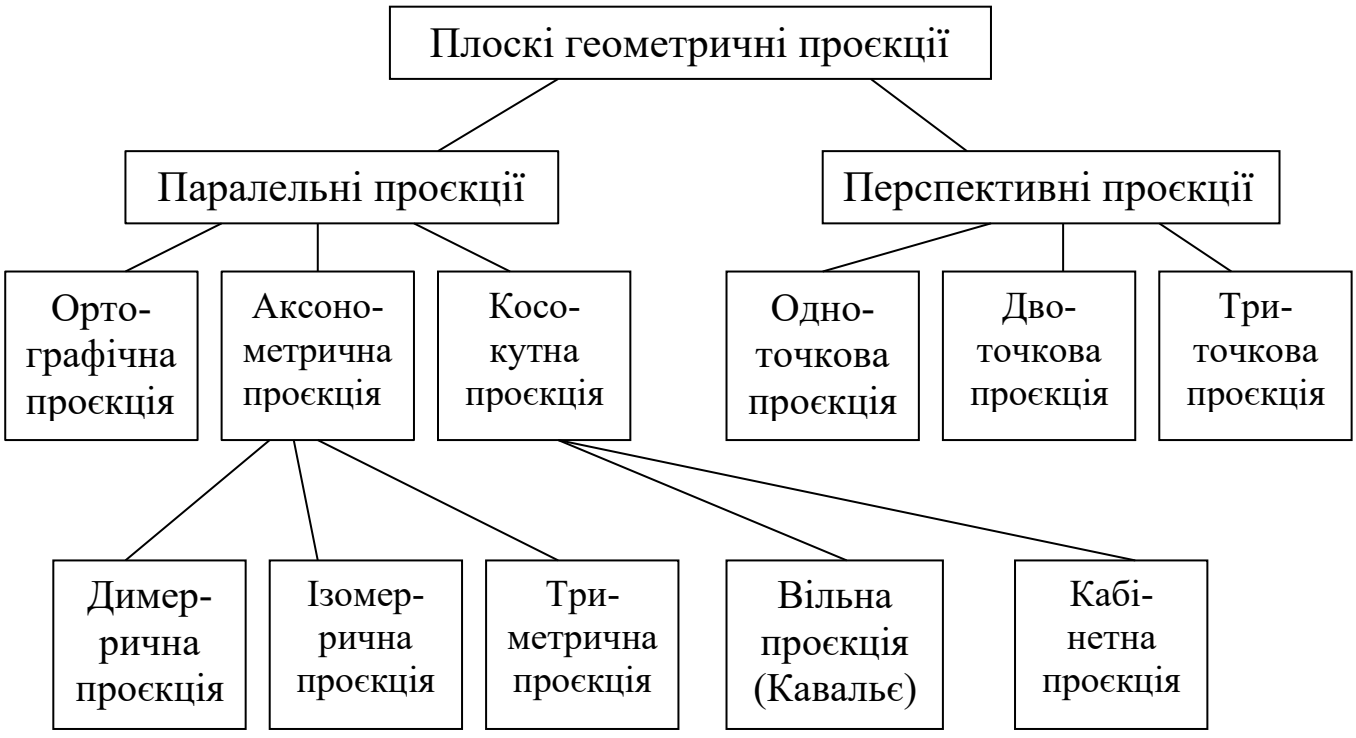


Рис. 12.3. Класифікація проєкцій

Аналогічно виписуються матриці ортогонального проєктування вздовж двох інших координатних осей

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

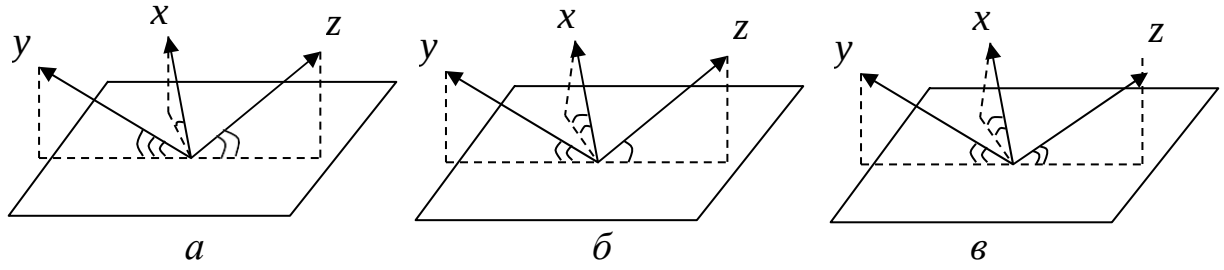
Зауваження 1. Для адекватного відтворення форми об'єкта недостатньо однієї ортогональної проєкції, оскільки матриці ортогонального проєктування вироджені. Необхідно мати кілька ортогональних проєкцій, а саме проєкції на площини z = 0, x = 0, y = 0. Вони є видами об'єкта спереду (фронтальний), справа (поздовжній) і зверху (горизонтальний). В технічних кресленнях за основні площини проєкцій приймають шість граней куба. Зазначимо, що всі проєкції можуть бути одержані комбінуючи перетворення відображення, повороту, перенесення з ортогональним проєктуванням на площину z = 0.

12.3. Аксонометрична проєкція

Одна ортографічна проєкція не може дати уяви про загальну форму тривимірного об'єкта. Це обмеження можна усунути за допомогою аксонометричних проєкцій. Аксонометрична проєкція (від грецького axon – вісь, metreo – вимірювати) утворюється маніпулюванням об'єкта шляхом поворотів так, щоб були видимими принаймні три сусідні грані.

При аксонометричній проєкції проєктори теж перпендикулярні КП, але вже не паралельні координатних осям. Відповідно до взаємного розміщення КП і координатних осей розрізняють три види аксонометричних проєкцій:

- 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*).



а *б* *в*

Рис. 12.4. Три види аксонометричної проєкції: *а* – триметрія; *б* – диметрія; *в* – ізометрія

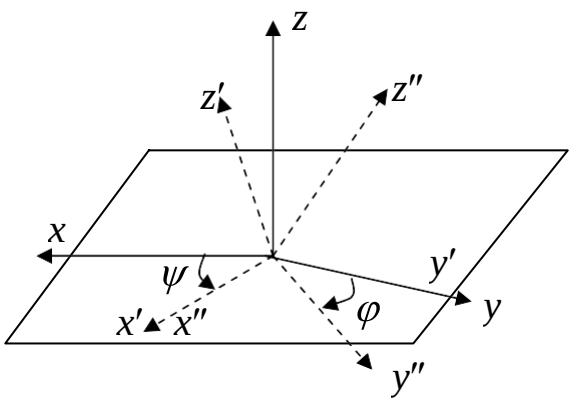


Рис. 12.5. Повороти системи координат

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 проєктування вздовж осі z''. В результаті одержуємо *матрицю аксонометричного проєктування*

$$A = R_y^-(\psi)\cdot R_x(\varphi)\cdot P_z = \begin{pmatrix} \cos\psi & 0 & \sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times$$

$$\times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos\psi & -\sin\psi\sin\varphi & 0 & 0 \\ 0 & \cos\varphi & 0 & 0 \\ -\sin\psi & -\cos\psi\sin\varphi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Триметрична проєкція здійснюється довільними поворотами, що виконуються в будь-якому порядку навколо координатних осей із наступним ортографічним проєктуванням на КП, тобто φ та ψ у матриці *A* для триметрії є довільними. Кількість триметричних проєкцій нескінченна.

На рис. 12.6 зображено триметричні проєкції зрізаного куба для різних значень кутів φ та ψ. Куб зі зрізаною вершиною задається матрицею *X*, де

$$X = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0{,}5 & 1 & 1 \\ 0{,}5 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0{,}5 & 1 \end{pmatrix}.$$

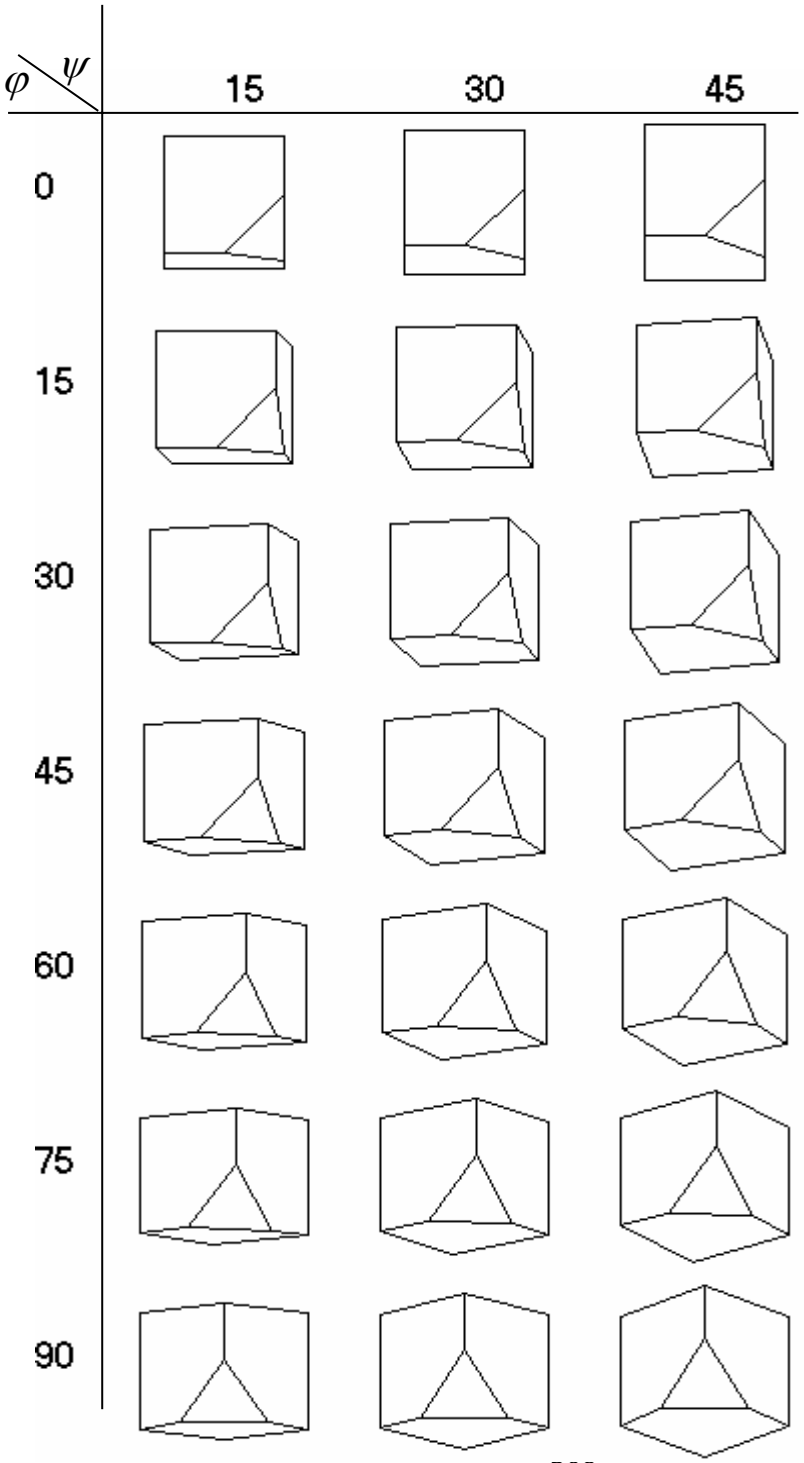


Рис. 12.6.
Триметричні проєкції
зрізаного куба

Координати триметричної проєкції зрізаного куба обчислюємо за формулою $X' = X \cdot A$, де A – матриця аксонометричного проєктування.

Для диметричної проєкції 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 проєктування A. Маємо

$$X^* = X \cdot A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \cos\psi & -\sin\psi\sin\varphi & 0 & 0 \\ 0 & \cos\varphi & 0 & 0 \\ -\sin\psi & -\cos\psi\sin\varphi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos\psi & -\sin\psi\sin\varphi & 0 & 1 \\ 0 & \cos\varphi & 0 & 1 \\ -\sin\psi & -\cos\psi\sin\varphi & 0 & 1 \end{pmatrix}.$$

Квадрат довжини проєкції одиничного орта вздовж осі x (коефіцієнт спотворення) дорівнює

$$k_x^2 = \cos^2\psi + \sin^2\psi\sin^2\varphi = 1 - \sin^2\psi + \sin^2\psi\sin^2\varphi =$$

$$= 1 - \sin^2\psi\cos^2\varphi \le 1. \qquad (12.1)$$

Аналогічно квадрати довжин проєкцій одиничних ортів вздовж осей y та z дорівнюють, відповідно,

$$k_y^2 = \cos^2\varphi \le 1, \qquad (12.2)$$

$$k_z^2 = \sin^2\varphi + \cos^2\varphi\sin^2\psi = 1 - \cos^2\varphi\cos^2\psi \le 1. \qquad (12.3)$$

Масштабні коефіцієнти k_x, k_y, k_z є довжинами проєкцій одиничних ортів на КП. Із формул (12.1), (12.2), (12.3) випливає умова взаємозв'язку між коефіцієнтами k_x, k_y, k_z:

$$k_x^2 + k_y^2 + k_z^2 = 1 - \sin^2\psi\cos^2\varphi + \cos^2\varphi + 1 - \cos^2\varphi\cos^2\psi = 2 \qquad (12.4)$$

Часто для аксонометричної проєкції задають не кути, а масштаби k_x, k_y, k_z (два з цих параметрів вільні), тоді з формул (12.2), (12.3) можна знайти параметри φ та ψ, тобто

$$\cos^2\varphi = k_y^2, \qquad \cos^2\psi = \frac{1 - k_z^2}{k_y^2}.$$

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

$$c = \sqrt{\frac{2}{m_x^2 + m_y^2 + m_z^2}}. \qquad (12.5)$$

Диметрія характеризується тим, що довжини двох проєкцій 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

$$\sin^2\psi\,(1 - \sin^2\varphi) = \sin^2\varphi, \qquad (12.6)$$

або

$$\sin^2\psi = \operatorname{tg}^2\varphi. \qquad (12.7)$$

Фіксуючи один із кутів, наприклад ψ, з (12.7) знаходимо

$$\varphi = \operatorname{arctg}(\pm\sin\psi).$$

Тоді, наприклад, якщо вибрати $\psi = 60^0$, 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 проєкція, 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

$$\cos^2\varphi = k_y^2 = \frac{8}{9}, \qquad \cos^2\psi = \frac{1 - k_z^2}{k_y^2} = \frac{7}{8},$$

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 проєктування має вигляд

$$A_{\text{дим}}^{\text{ст}} = \begin{pmatrix} \sqrt{\frac{7}{8}} & -\sqrt{\frac{1}{72}} & 0 & 0 \\ 0 & \sqrt{\frac{8}{9}} & 0 & 0 \\ -\sqrt{\frac{1}{8}} & -\sqrt{\frac{7}{72}} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \approx \begin{pmatrix} 0{,}935 & -0{,}118 & 0 & 0 \\ 0 & 0{,}943 & 0 & 0 \\ -0{,}354 & -0{,}312 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Умова $k_x = k_y \neq k_z$ для диметричної проєкції дозволяє знаходити розміри спроєктованого 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 спроєктованого об'єкта.

Цей недолік усуває ізометрична проєкція. 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

$$\sin^2\psi = \frac{\sin^2\varphi}{1-\sin^2\varphi}, \qquad (12.8)$$

аналогічно з умови $k_x^2 = k_y^2$ одержуємо

$$\sin^2\psi = \frac{1-2\sin^2\varphi}{1-\sin^2\varphi}. \qquad (12.9)$$

Прирівнюючи праві частини (12.8) і (12.9), знаходимо $\sin^2\varphi = \frac{1}{3}$, або $\varphi = \arcsin(\pm\frac{1}{\sqrt{3}}) \approx \pm 35{,}26^0$. Тоді з (12.8) $\sin^2\psi = \frac{1}{3} : (1 - \frac{1}{3}) = \frac{1}{2}$, а отже, $\psi = \pm 45^0$.

Коефіцієнт спотворення для ізометричної проєкції $k_y = \sqrt{\cos^2\varphi} = \sqrt{2/3} \approx 0{,}8165$ 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 проєктування, що одержується при $\psi = 45^0$ та $\varphi = 35{,}26^0$, має вигляд

$$A_{\text{ізо}} = \begin{pmatrix} \sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{6}} & 0 & 0 \\ 0 & \sqrt{\frac{2}{3}} & 0 & 0 \\ -\sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{6}} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0{,}707 & -0{,}408 & 0 & 0 \\ 0 & 0{,}816 & 0 & 0 \\ -0{,}707 & -0{,}408 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Якщо взяти куб із відрізаною вершиною, який задається матрицею X, яка була наведена вище, і подіяти на неї матрицею ізометричного проєктування $A_{ізо}$, то одержимо координати ізометричної проєкції куба, тобто

$$X' = X \cdot A_{\text{ізо}} = \begin{pmatrix} -0{,}707 & -0{,}408 & 0 & 1 \\ 0 & -0{,}816 & 0 & 1 \\ 0 & -0{,}408 & 0 & 1 \\ -0{,}354 & 0{,}204 & 0 & 1 \\ -0{,}707 & 0{,}408 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0{,}707 & -0{,}408 & 0 & 1 \\ 0{,}707 & 0{,}408 & 0 & 1 \\ 0 & 0{,}816 & 0 & 1 \\ 0{,}354 & 0{,}204 & 0 & 1 \end{pmatrix}.$$

На рис. 12.8 зображено чотири можливі ізометричні проєкції зрізаного куба з кутами $\psi = \pm 45^{\circ}$, $\varphi = \pm 35{,}26^{\circ}$.

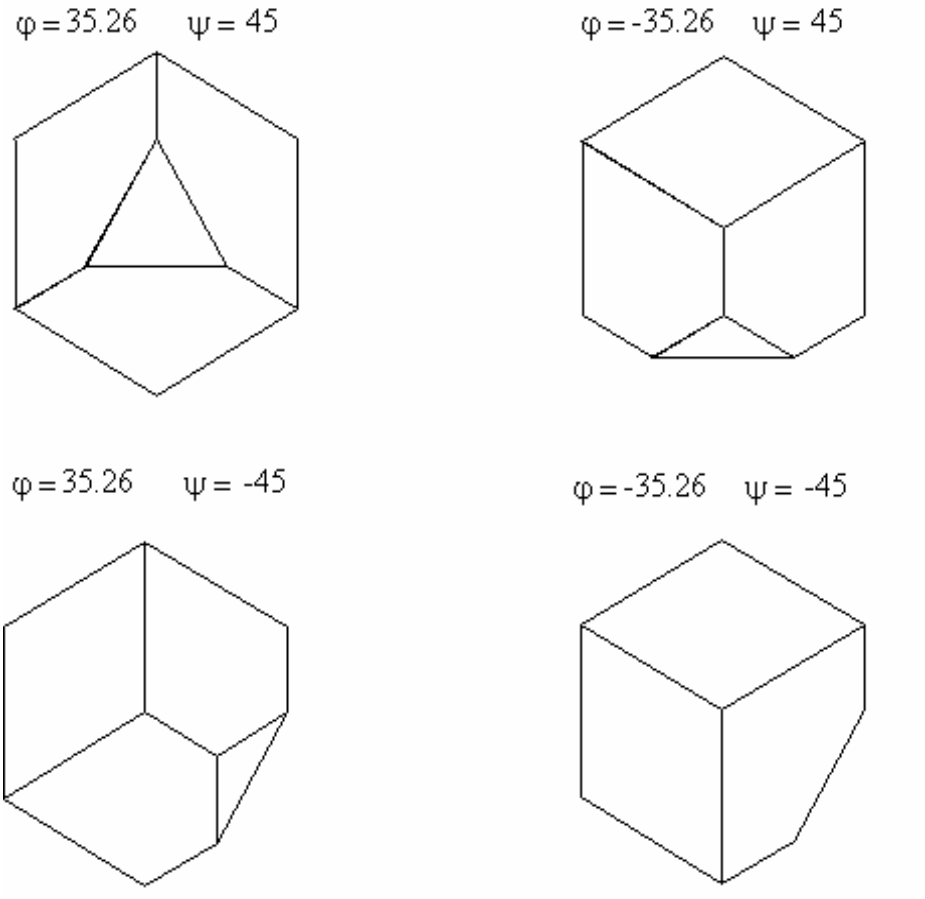


Рис. 12.8. Ізометричні проєкції зрізаного куба

12.4. Косокутна проєкція

На відміну від ортографічної і аксонометричної проєкцій, для яких проєктори перпендикулярні до КП, косокутна проєкція формується паралельними проєкторами, що проходять не під прямим кутом до КП.

Косокутні проєкції 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 проєктуванні 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.

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 $\overrightarrow{PP_1}$ 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 $\overrightarrow{PP_1} = (l\cos\alpha, l\sin\alpha, -1)$.

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, $l = \operatorname{ctg}\beta$ ($\operatorname{arcctg} l = \beta$), де β – кут нахилу косих проєкторів до площини проєкції $z = 0$.

Нехай тепер задано довільну точку $Q(x, y, z)$. Знайдемо матричне перетворення косокутного проєктування, тобто перетворення, яке точку Q перетворює в точку Q'.

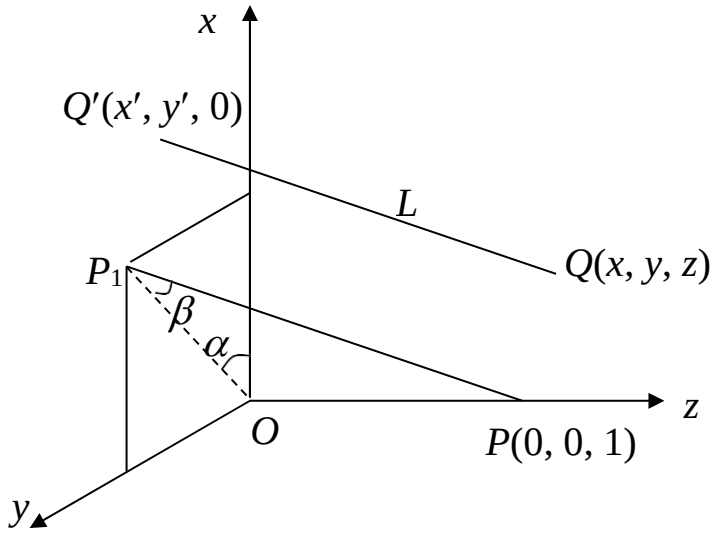


Рис. 12.9. Напрям косокутного проєктування

Рівняння прямої, що проходить через точку Q паралельно вектору $\overrightarrow{PP_1}$, 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

$$x' = x + lz\cos\alpha,$$
$$y' = y + lz\sin\alpha,$$

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 проєктування. Змінюючи значення α, β, одержуємо різні варіанти косокутної проєкції. Надалі для спрощення запису для матриці косокутного проєктування будемо виписувати тільки верхню ліву частину розміром 3×3.

Зауваження 2. *Якщо напрям проєктування* $\overrightarrow{PP_1}$ *задати вектором* (p_x, p_y, p_z), *то*

$$l = \operatorname{ctg}\beta = \frac{\sqrt{p_x^2 + p^2{}_y}}{p_z},\ \sin\alpha = \frac{p_y}{\sqrt{p_x^2 + p^2{}_y}},\ \cos\alpha = \frac{p_x}{\sqrt{p_x^2 + p^2{}_y}},$$

і матриця косокутного проєктування 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 проєктування 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 $\overrightarrow{PP_1}$ дають стандартні косокутні проєкції. Особливий інтерес являють собою дві косокутні проєкції – проєкція Кавальє (вільна проєкція) і кабінетна проєкція. *Проєкція Кавальє* (у вітчизняній літературі горизонтальна косокутна ізометрія) одержується тоді, коли кут між проєкторами і площиною проєкції дорівнює 45^0 ($l = 1$), однак результат цієї проєкції виглядає неприродно потовщеним (рис. 12.10).

Цього недоліку немає в *кабінетній проєкції* (фронтальна косокутна диметрія), в якій коефіцієнт спотворення $l = \frac{1}{2}$, тобто кут між проєкторами і КП β = arcctg(1/2) = 63,43^0.

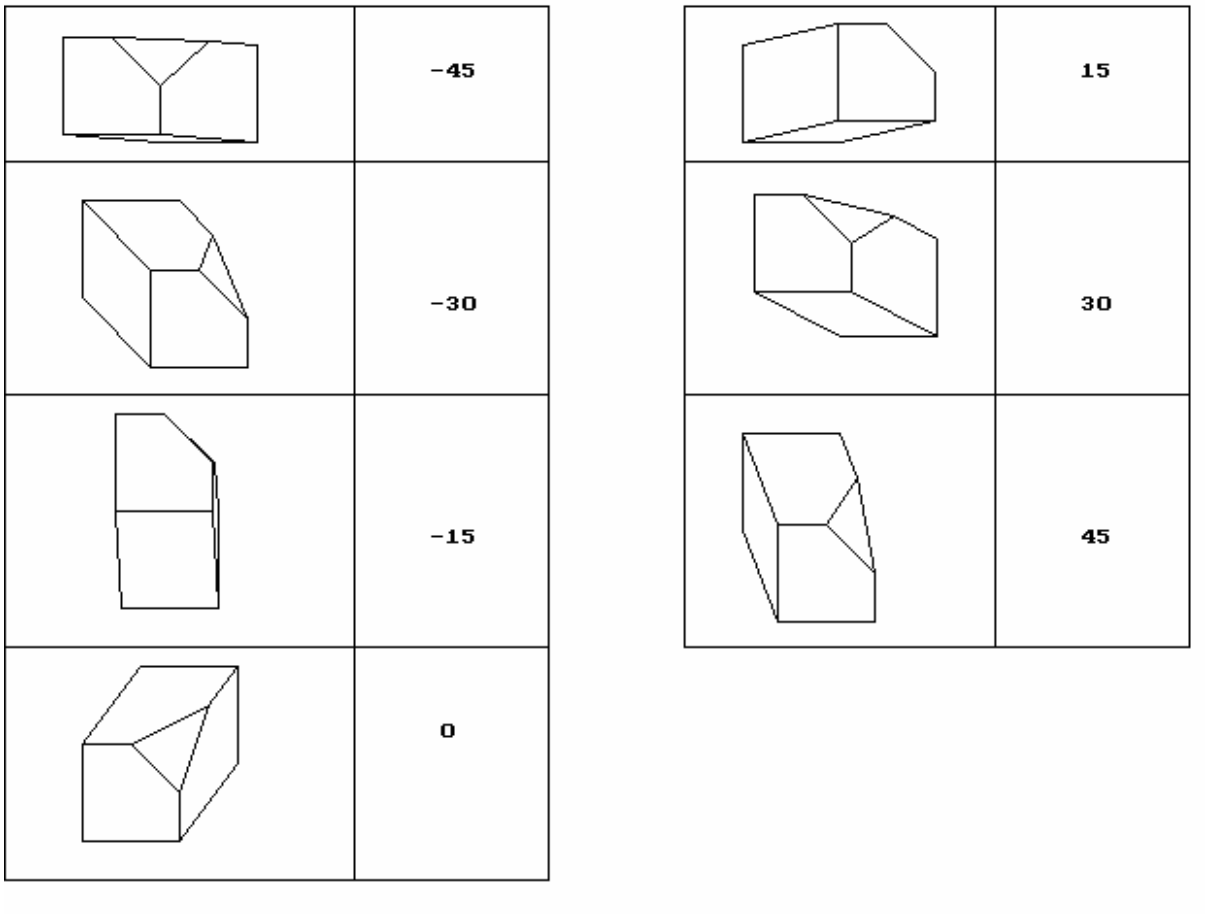


Рис. 12.10. Проєкція Кавальє

Проєкція Кавальє визначається вектором матрицею $\overrightarrow{PP_1}$ = (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 проєкція визначається вектором (1, 1, 2 $\sqrt{2}$) і задається матрицею

$$K_{\text{каб}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & 0 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0{,}354 & 0{,}354 & 0 \end{pmatrix}.$$

Проєкція Кавальє зрізаного куба при різних кутах α наведена на рис. 12.10, а кабінетна проєкція – на рис. 12.11.

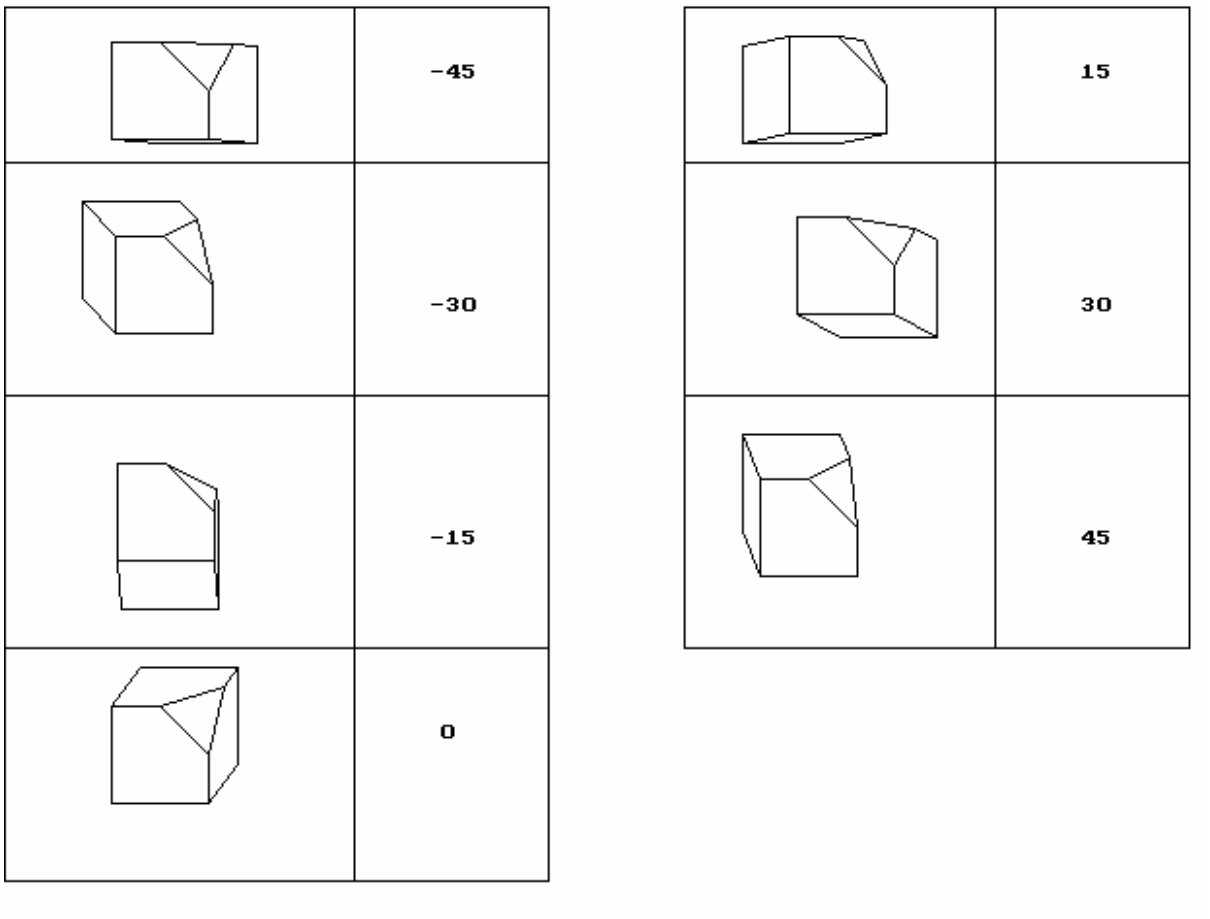


Рис. 12.11. Кабінетна проєкція

12.5. Одноточкова (однофокусна) перспективна проєкція

Спочатку розглянемо простіший випадок центрального проєктування. 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

$$\frac{c}{z} = \frac{y'}{y}, \quad \frac{c}{z} = \frac{x'}{x},$$

матричною формулою $P' = P \cdot M_1$, де

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/c \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

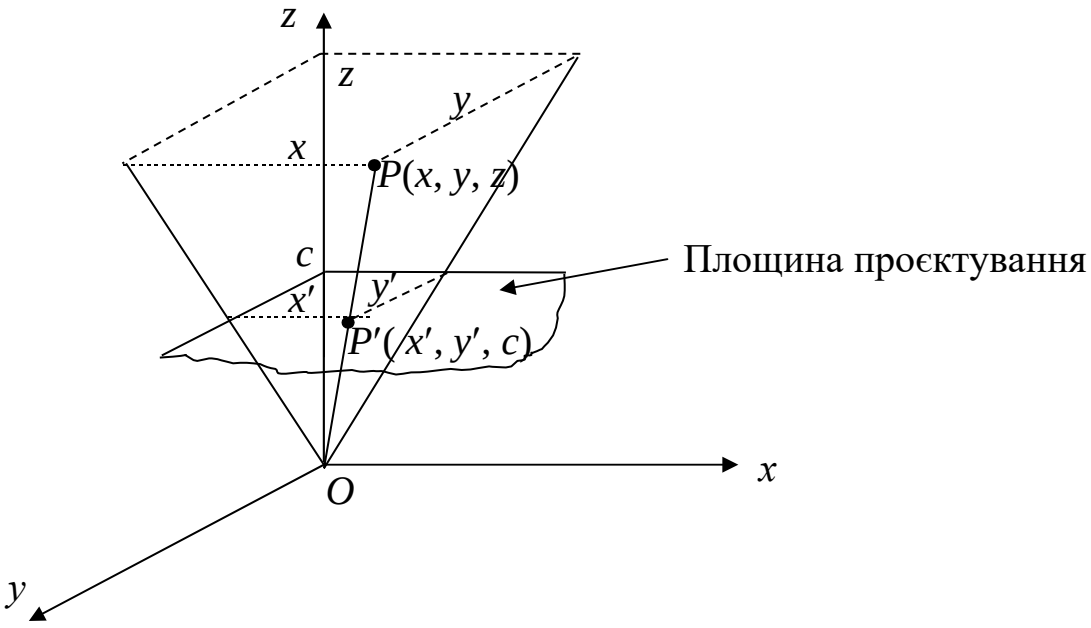


Рис. 12.12. Перспективна проєкція

Розглянемо інший варіант центрального проєктування. Нехай КП знаходиться у площині $z = 0$, а камера – в точці $C(0, 0, c)$ (рис. 12.13).

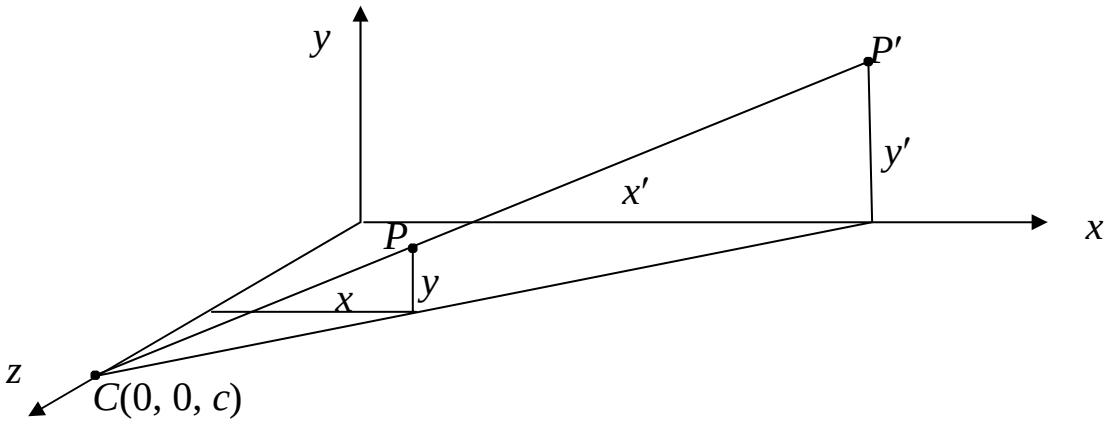


Рис. 12.13. Інший варіант перспективної проєкції

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

$$X = xt,\ Y = yt, \qquad Z = c + (z - c)t.$$

Знайдемо координати точки перетину побудованої прямої з площиною $z = 0$. З умови $Z = 0$ маємо $t = \frac{c}{c-z}$, а отже, $x' = \frac{xc}{c-z} = \frac{x}{1-\frac{z}{c}}$, $y' = \frac{yc}{c-z} = \frac{y}{1-\frac{z}{c}}$, тобто одержуємо точку $P'(\frac{x}{1-\frac{z}{c}}, \frac{x}{1-\frac{z}{c}}, 0, 1) = P'(x,y,0,1 - \frac{z}{c})$.

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 $z = 0$:

$$M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/c \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

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

$$(x'\ y'\ z'\ 1) = (x\ y\ z\ 1) \cdot P_c = (x\ y\ z\ 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/c \\ 0 & 0 & 0 & 1 \end{pmatrix} = (x\ \ y\ z\ \ 1 - \frac{z}{c}).$$

Отже, координати перспективного перетворення точки (x, y, z) в цьому випадку проєктування визначаються за формулою

$$(x', y', z', 1) = (\frac{x}{1 - z/c}, \frac{y}{1 - z/c}, \frac{z}{1 - z/c}, 1).$$

Тепер перспективне проєктування на КП може бути одержане як послідовне виконання операцій перспективного перетворення та ортогонального проєктування. Наприклад, перспективне проєктування на площину $z = 0$ задається матрицею $M_2 = P_c \cdot P_z$, де P_c – матриця перспективного перетворення, а P_z – матриця ортогонального проєктування 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, тому центральне проєктування 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 $M(a,\ b,\ 0)$ свого 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

$$(a\ b\ t\ 1)\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & -1/c\\ 0 & 0 & 0 & 1\end{pmatrix} = (a\ b\ t\ 1-\frac{t}{c}) = (\frac{a}{1-\frac{t}{c}}\quad \frac{b}{1-\frac{t}{c}}\quad \frac{t}{1-\frac{t}{c}}\quad 1).$$

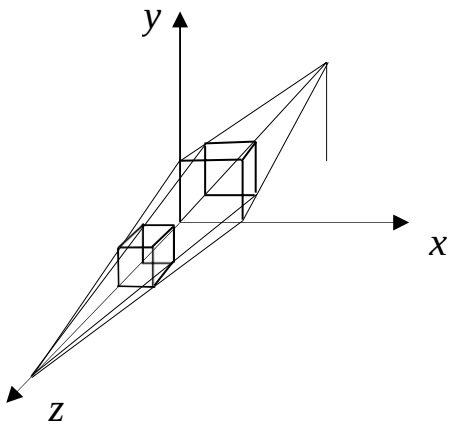


Рис. 12.14. Проєкція з однією точкою збігу

При цьому точки образу $(\frac{a}{1-\frac{t}{c}}, \frac{b}{1-\frac{t}{c}}, \frac{t}{1-\frac{t}{c}}, 1)$ при $t \to \infty$ 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 розглядатися 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

$$(x'\ y'\ z'\ 1) = (x\ \ y\ \ z\ \ 1)\begin{pmatrix} 1 & 0 & 0 & -1/a \\ 0 & 1 & 0 & -1/b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = (x\ \ y\ z\ \ 1 - \frac{x}{a} - \frac{y}{b}) =$$

$$= \left(\frac{x}{1-\frac{x}{a}-\frac{y}{a}} \quad \frac{y}{1-\frac{x}{a}-\frac{y}{a}} \quad \frac{z}{1-\frac{x}{a}-\frac{y}{a}} \quad 1 \right).$$

Зауважимо, що двоточкове перспективне перетворення можна одержати шляхом об’єднання двох одноточкових перспективних перетворень, тобто

$$P_{ab} = \begin{pmatrix} 1 & 0 & 0 & -\frac{1}{a} \\ 0 & 1 & 0 & -\frac{1}{b} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 0 & 0 & -1/a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1/b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = P_a \cdot P_b.$$

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 $z = 0$.

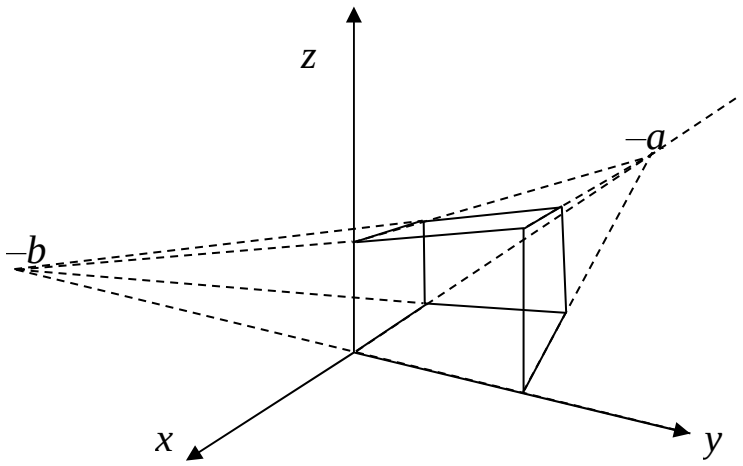


Рис. 12.15. Двоточкове центральне перетворення

Двоточкова центральна проєкція широко застосовується в архітектурному, інженерному та промисловому проєктуванні, в комп'ютерній графіці.

Триточкова перспектива задається матрицею, в якої три перших елементи четвертого стовпчика матриці перетворення не дорівнюють нулю. Триточкове перспективне перетворення

$$(x'\ y'\ z'\ 1) = (x\ \ y\ \ z\ \ 1)\begin{pmatrix} 1 & 0 & 0 & -1/a \\ 0 & 1 & 0 & -1/b \\ 0 & 0 & 1 & -1/c \\ 0 & 0 & 0 & 1 \end{pmatrix} = (x\ \ y\ z\ \ 1 - \frac{x}{a} - \frac{y}{b} - \frac{z}{c})$$

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 кожну 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 проєктуванням на площину $z = 0$ із центром проєкції в точці $z = c$ має вигляд

$$M = R_y \cdot M_2 = \begin{pmatrix} \cos\psi & 0 & -\sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/c \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos\psi & 0 & 0 & \frac{\sin\psi}{c} \\ 0 & 1 & 0 & 0 \\ \sin\psi & 0 & 0 & -\frac{\cos\psi}{c} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

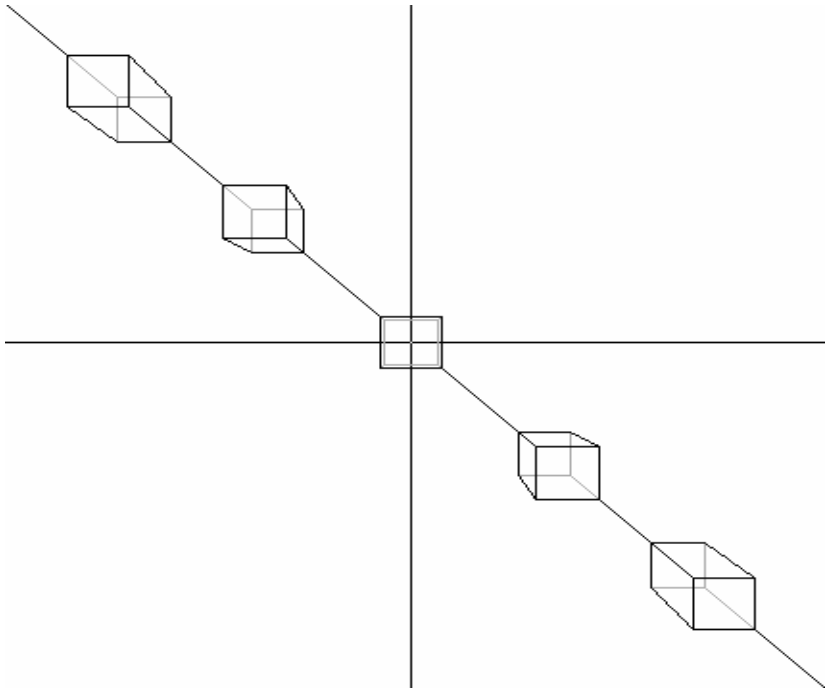


Рис. 12.16. Одноточкова перспективна проєкція перенесеного об'єкта в x, y-напрямках

Звідси видно, що обертання навколо осі x (аналогічно y) з подальшим проєктуванням 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 проєктування на площину $z = 0$. Таке перетворення задається матрицею $M = R_y \cdot R_x \cdot M_2$.

Контрольні запитання та завдання

1. Як будуються проєкції? Дайте їх класифікацію.
2. Виписати матриці ортогональних перетворень.
3. В чому полягають переваги та недоліки перспективних проєкцій?
4. Як утворюється аксонометрична проєкція? Назвіть її різновиди.
5. Виписати матрицю аксонометричного проєктування.
6. Які властивості має диметрична/ізометрична проєкції?
7. Як вибирається напрям косокутного проєктування?
8. Який вигляд має матриця косокутного проєктування?
9. Назвати дві особливі косокутні проєкції. Порівняйте їх.
10. Як побудувати матрицю одноточкової перспективної проєкції?
11. Які властивості має матриця перспективного перетворення?
12. Що таке точка збігу? Як довести її існування?
13. Як задаються двоточкові перспективні перетворення?
14. Які властивості має двоточкова перспективна проєкція?
15. Виписати матрицю триточкової перспективної проєкції. Вкажіть центри проєктування та точки збігу.
16. Обчислити матрицю $M = R_y \cdot R_x \cdot M_2$.
17. 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^o 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 відобра-

зити відносно площини, що проходить через три задані точки.

4. Побудувати матрицю триметричного проєктування при $\psi = 30^o$, $\varphi = 45^o$. Обчислити координати проєкції зрізаного куба.
5. Знайти координати проєкції Кавальє зрізаного куба при $\alpha = 45^o$.
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.
9. Побудувати матрицю аксонометричного проєктування куба при співвідношенні масштабів k_x:k_y: k_z=6:5:4.

Розділ 13. Системи координат та їх перетворення

13.1. Системи координат

У комп'ютерній графіці використовуються кілька 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, зв'язану 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 екрана (вікно або область виведення), які мають свої системи координат.

Розглянемо більш детально ці системи координат і 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 уже 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

$$P = P_0 \cdot M_0. \qquad (13.1)$$

- *Видова система координат* (ВСК). Це система координат спостерігача. Вважається, що спостерігач знаходиться в початку координат ВСК, 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* – вверх, тобто видова система координат лівостороння, а це відповідає вибору напряму осей екранної системи координат (вісь *z* направлена вглиб екрана). З видовою системою координат пов’язане *видове перетворення*, яке здійснює відображення світових координат у видові координати.

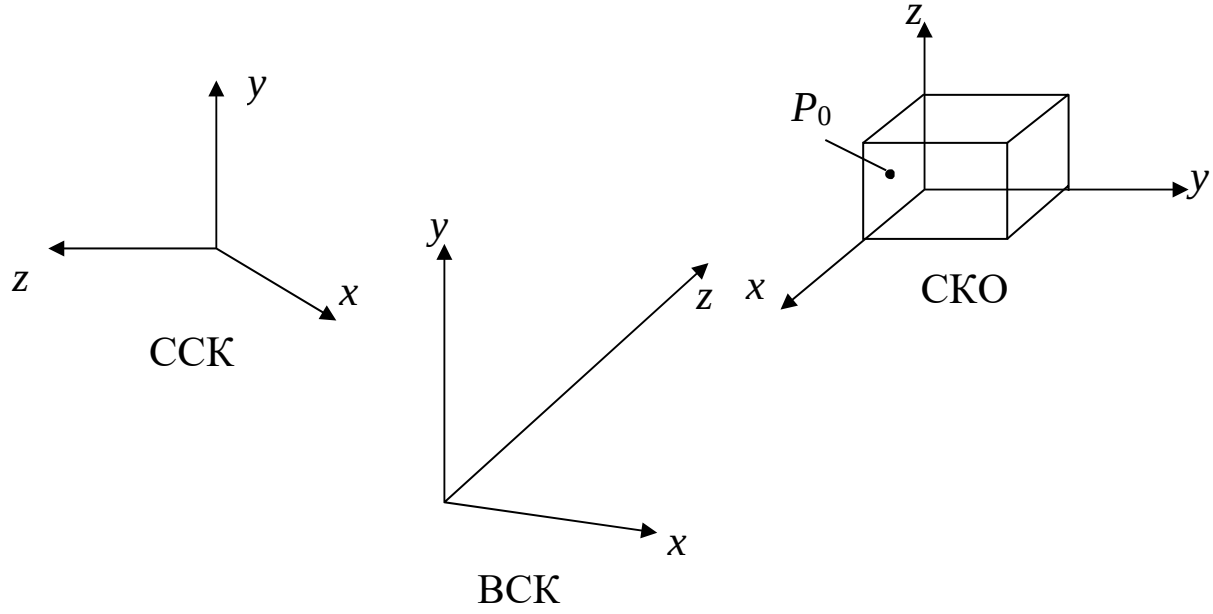


Рис. 13.1. Локальна (СКО), світова (ССК) та видова (ВСК) системи координат

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

$$P_v = P \cdot M_v. \qquad (13.2)$$

Із рівностей (13.1), (13.2) маємо ще один вираз для координат точки P_v у видовій системі координат

$$P_v = P_0 \cdot M_0 \cdot M_v, \qquad (13.3)$$

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 проєктуванні або у вигляді паралелепіпеда при ортографічному проєктуванні. Кінцевою метою графічної системи є відображення тривимірної піраміди видимості на двовимірне вікно екрана.

- *Система координат виведення зображення* (СКВЗ). Ця система координат чотиривимірного простору, точки якого являють собою однорідні координати тривимірного простору в лівосторонній сис-

темі 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 проєктуванні відобразиться в точку $P_c(x_c, y_c, z_c, h_c)$ за допомогою перетворення

$$P_c = P_v \cdot M_{pr}, \qquad (13.4)$$

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 вверх, а вісь z_p – углиб екрана (рис. 13.2).

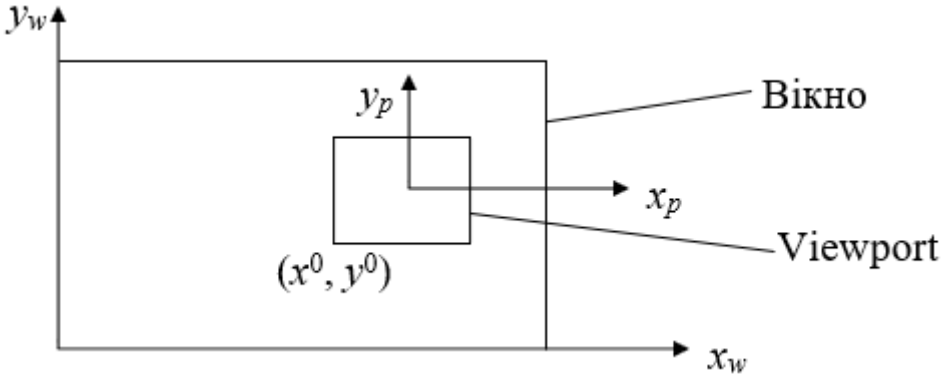


Рис. 13.2. Віконна система координат x_w, y_w та система координат області виведення (Viewport) $x_p y_p$

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

$$P_p = P_d \cdot M_{vp}. \qquad (13.5)$$

• *Віконна система координат* (ВСК). Початок цієї системи координат розміщується в лівому нижньому куті вікна, вісь x_w направлена горизонтально вправо, а вісь y_w – вверх. Отже, ця система координат одержується із системи координат області виведення шляхом перенесення початку координат. Перенесення системи координат теж можна записати в матричній формі:

$$P_w = P_p \cdot M_w, \qquad (13.6)$$

де M_w – матриця перенесення.

На рис. 13.3 схематично показана вся описана вище послідовність перетворень координат.

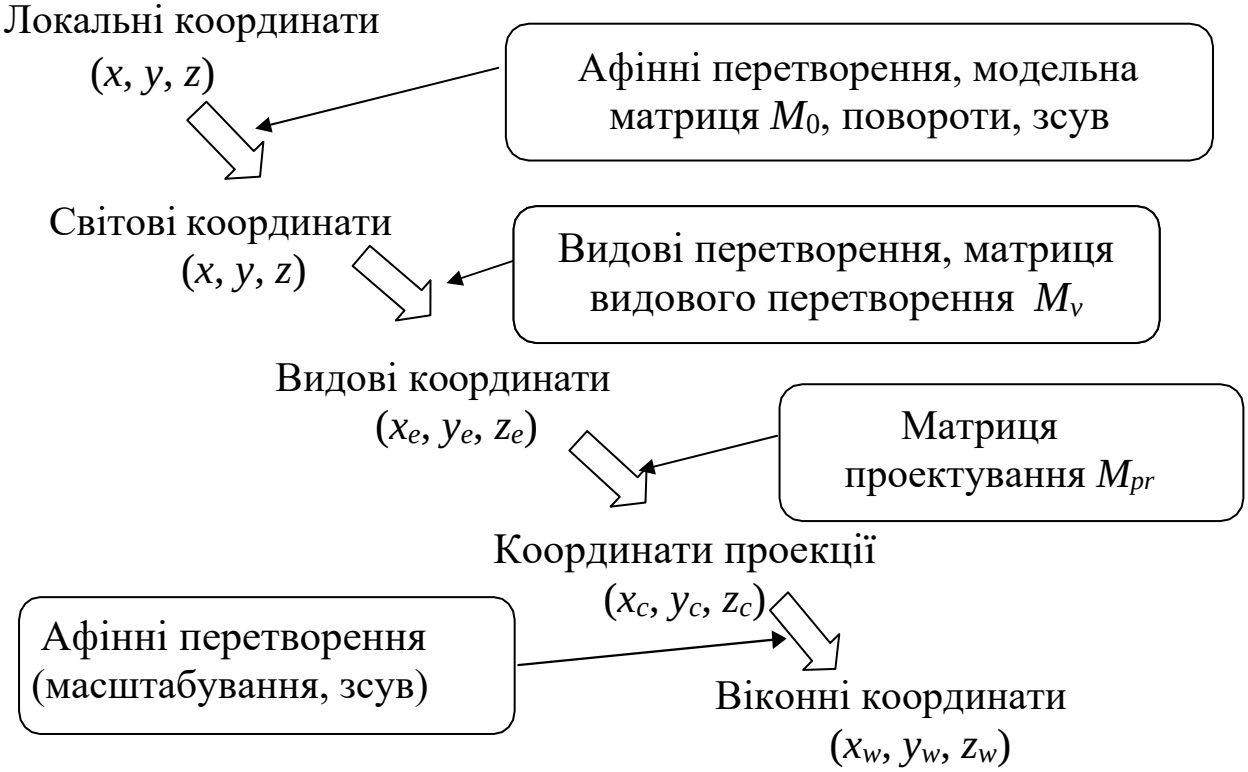


Рис. 13.3. Послідовність перетворень координат

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 спостерігача E і об’єкт у світовій системі координат, яка завжди правостороння. 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 її сферичні координати.

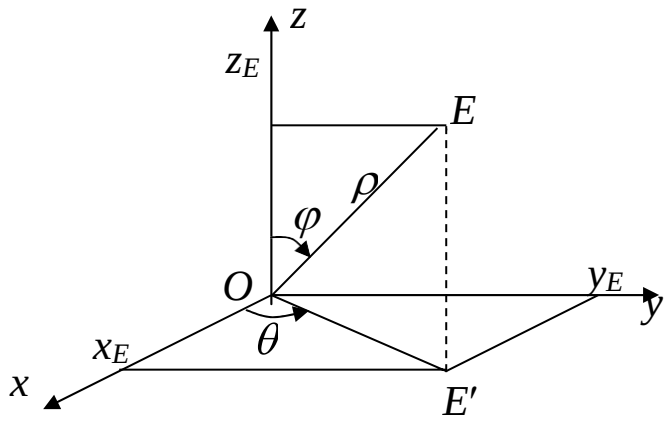


Рис. 13.4. Сферичні координати точки спостерігача E

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 екрана). 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.

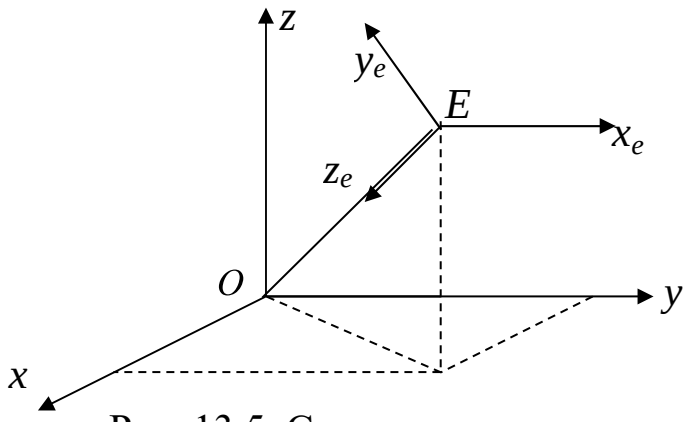


Рис. 13.5. Система видових координат

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

$$T^{-} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_E & -y_E & -z_E & 1 \end{pmatrix}.$$

2) Поворот системи навколо осі z. Повернемо систему координат навколо осі z на кут $\left(\frac{\pi}{2} - \theta\right)$ 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

$$R_z = \begin{pmatrix} \cos\left(\frac{\pi}{2} - \theta\right) & \sin\left(\frac{\pi}{2} - \theta\right) & 0 & 0 \\ -\sin\left(\frac{\pi}{2} - \theta\right) & \cos\left(\frac{\pi}{2} - \theta\right) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \sin\theta & \cos\theta & 0 & 0 \\ -\cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

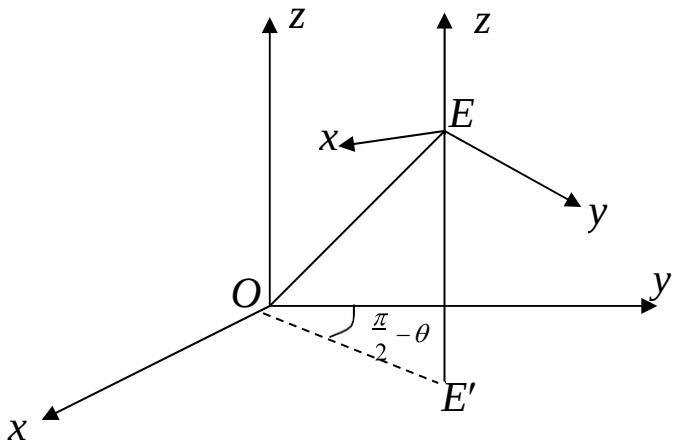


Рис.13.6. Поворот відносно осі Z на кут $\left(\frac{\pi}{2} - \theta\right)$

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

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

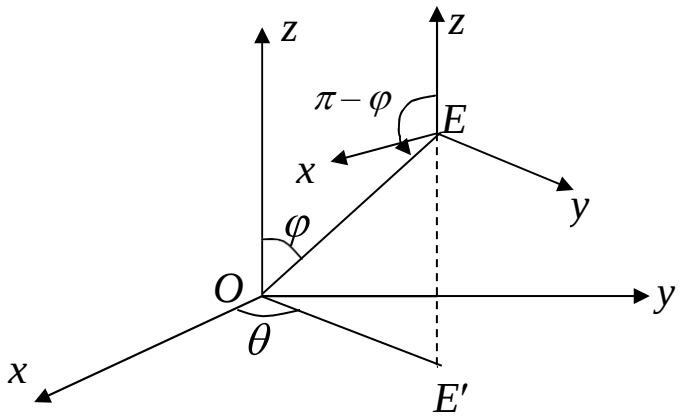


Рис. 13.7. Поворот відносно осі *Ex*

У результаті цього повороту нова вісь *z* буде направлена вздовж *EO*.

4) Зміна напрямку осі *x*. Після трьох перетворень осі *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

$$M_{yz} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Тоді матриця видового перетворення M_v обчислюється як добуток

$$M_v = T^{-} \cdot R_z \cdot R^{-}_{x} \cdot M_{yz}.$$

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). Використавши ці координати, можемо побудувати ортогональну або центральну проєкцію об’єкта. Значення z_e будуть використовуватися для усунення невидимих ліній та граней, наприклад у методі *Z*-буфера.

13.3. Перспективне проєктування

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 *c* від спостерігача, то координати центральної проекції знаходяться 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 самий напрям, що й x_e, y_e, z_e (рис. 13.8).

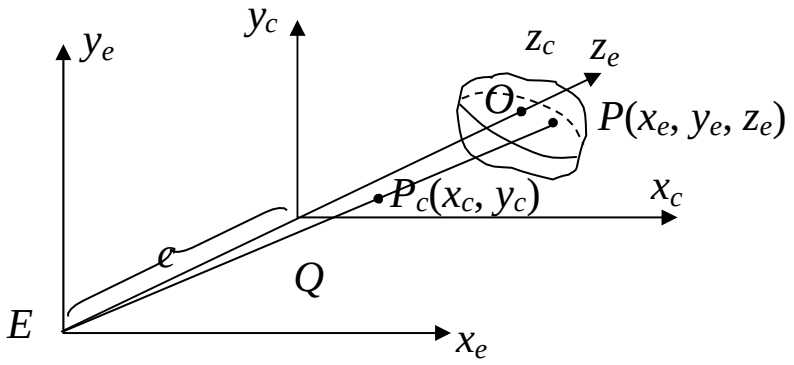


Рис. 13.8. Картинна площина і видові координати

Точка $Q(0, 0, c)$ буде знаходитися в центрі проекції, 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 $x_c, y_c \in [-1, 1]$.

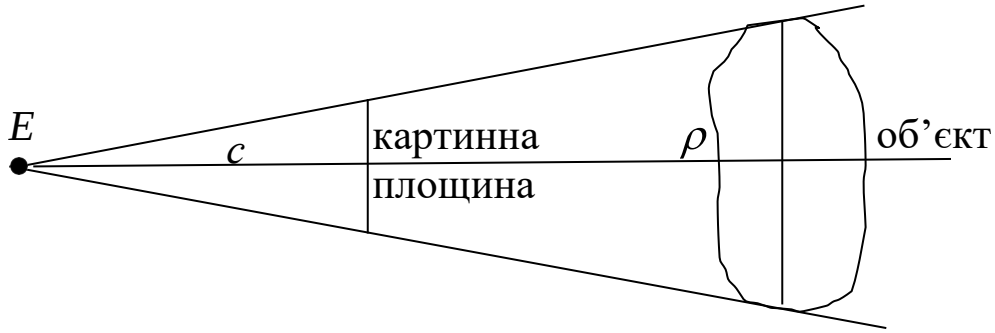


Рис. 13.9. Розмір картини і об’єкта

Окрім цього, розмір картини проєкції можна підбирати за рахунок відповідного вибору числа c – віддалі між точкою спостереження E і картинною площиною x_cQy_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 “центра” 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 $\alpha \approx 150^0$, вертикальний кут $\beta \approx 110^0$ 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

$$\left(x, y, \frac{z-\rho}{c}, \frac{z}{c}\right) = (x, y, z, 1)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/c & 1/c \\ 0 & 0 & -\rho/c & 0 \end{pmatrix} = (x, y, z, 1)M_1'$$

(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).

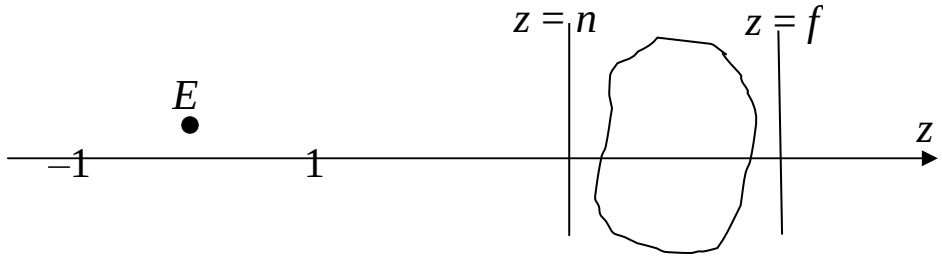


Рис. 13.10. Обмежуючі площини об’єкта

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

$$(x_c^*, y_c^*, z_c^*, 1) = (x, y, z, 1)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & p & q \\ 0 & 0 & r & s \end{pmatrix} = (x, y, pz + r, qz + s) =$$

$$= \left(\frac{x}{qz+s}, \frac{y}{qz+s}, \frac{pz+r}{qz+s}, 1\right).$$

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 $z = 1$:

$$\frac{pf + r}{qf + s} = -1, \quad pf + r = \frac{f}{c}.$$

З двох останніх умов 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 проєктування задається піраміда видимості, тобто той об'єм простору, який необхідно відобразити у вікні. На основі цієї інформації графічна система створює матрицю проєктування. 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 проекції (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

$$X_{\max} = \max(|x_c|), \quad Y_{\max} = \max(|y_c|),$$

тоді $-X_{\max}$, $X_{\max}$, $-Y_{\max}$, $Y_{\max}$ задають границі проєкції сцени (рис. 13.11).

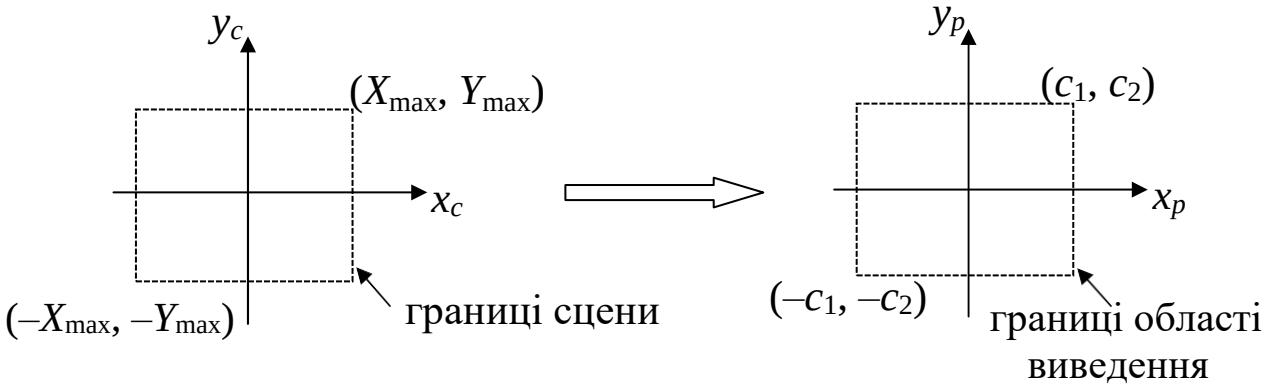


Рис. 13.11. Границі сцени та області виведення

Координати точки (x_p, y_p) у *системі координат області виведення* одержуються за формулами масштабування

$$x_p = k_x x_c, \quad y_p = k_y y_c .$$

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 вибирати однаковий коефіцієнт розтягу/стиску k для всіх координат, тобто

$$k = \min(k_x, k_y).$$

Зауваження 2. Якщо відомі нормалізовані координати проекції 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):

$$x_w = x_p + x^0 + c_1, y_w = y_p + y^0 + c_2.$$

Зауваження 3. Координати z у віконній системі координат не перетворюються, але будуть у подальшому використовуватися для аналізу глибини сцени.

Контрольні запитання та завдання

1. Які системи координат використовують у комп'ютерній графіці?
2. Яке перетворення задається модельною матрицею?
3. Дайте 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 в матричному вигляді.
4. Відобразити вмістиме нормованого квадрата в прямокутну область виведення на екрані.

Розділ 14. Усунення невидимих ліній і граней

14.1. Основні поняття

Усунення невидимих ліній і поверхонь (задача загороджування) – одна з найважливіших та найважчих проблем при створенні на комп'ютері якісного реалістичного зображення 3D-об'єктів.

Нехай задано тривимірний об'єкт і видові параметри, що визначають тип проєкції 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 проєкцій) або вздовж напрямку проектування (для паралельних проєкцій).

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, немає універсального алгоритму, тобто 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 необхідні ще й алгоритми, які генерують зображення з тінями, ефектами освітлення тощо. Усі ці алгоритми 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 багатогранники (кожна грань опуклого багатогранника або повністю видима, або повністю невидима). 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 взаємного розташування геометричних фігур. Вони особливо корисні, коли 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 почергово працюють в об'єктному просторі та у просторі зображень. Найрозповсюдженіші на практиці методи розв'язування задач усунення невидимих ліній та поверхонь – метод прямого перебирання, метод *Z*-буфера, метод поточних горизонтів, метод усунення нелицьових граней, метод розбиття картинної площини, 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 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 у 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 проектування.

Для задання напряму вектора проектування, тобто напряму нормалі до картинної площини, потрібно задати два кути φ, ψ. Нехай вектор нормалі до картинної площини утворює кут ψ із площиною xy, а проєкція нормалі на площину xy із віссю y – кут φ (рис. 14.1). Тоді напрям проектування задається вектором

$$\boldsymbol{n} = (\sin\varphi\cos\psi, \cos\varphi\cos\psi, \sin\psi), \quad \varphi \in [0, 2\pi],\ \psi\left[-\frac{\pi}{2}, \frac{\pi}{2}\right].$$

Для того, щоб виконати ортогональне проектування поверхні на картинну площину, вісь z' нової системи координат $x'y'z'$ повинна збігатися з напрямом проектування (рис. 14.1).

Розглянемо перетворення системи координат xyz 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 xyz 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 $\left(\frac{\pi}{2}-\psi\right)$ у напрямку від z до y. Матриця повороту має вигляд

$$R_x^-\left(\frac{\pi}{2}-\psi\right)=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & \cos\left(\frac{\pi}{2}-\psi\right) & -\sin\left(\frac{\pi}{2}-\psi\right) & 0\\ 0 & \sin\left(\frac{\pi}{2}-\psi\right) & \cos\left(\frac{\pi}{2}-\psi\right) & 0\\ 0 & 0 & 0 & 1\end{pmatrix}=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & \sin\psi & -\cos\psi & 0\\ 0 & \cos\psi & \sin\psi & 0\\ 0 & 0 & 0 & 1\end{pmatrix}.$$

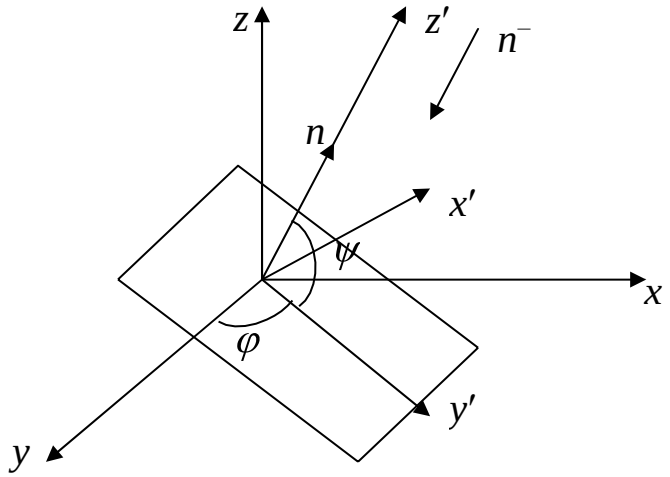


Рис. 14.1. Перетворення системи координат

Перетворення системи координат xyz у $x'y'z'$ задається добутком матриць $R_z^-(\varphi)R_x^-\left(\frac{\pi}{2}-\psi\right)$, тобто матрицею

$$A=\begin{pmatrix}\cos\varphi & -\sin\varphi\sin\psi & \sin\varphi\cos\psi & 0\\ \sin\varphi & \cos\varphi\sin\psi & -\cos\varphi\cos\psi & 0\\ 0 & \cos\psi & \sin\psi & 0\\ 0 & 0 & 0 & 1\end{pmatrix}. \qquad (14.1)$$

Це означає, що точка (x, y, z) з поверхні переходить у точку (x', y', z') перетвореної системи координат, де

$$\begin{aligned} x' &= x\cos\varphi + y\sin\varphi,\\ y' &= -x\sin\varphi\sin\psi + y\cos\varphi\sin\psi + z\cos\psi, \qquad (14.2)\\ z' &= x\sin\varphi\cos\psi - y\cos\varphi\cos\psi + z\sin\psi. \end{aligned}$$

Обернене перетворення задається матрицею $A^{-1} = A^{тр}$, тобто

$$\begin{aligned} x &= x'\cos\varphi - y'\sin\varphi\sin\psi + z'\cos\psi\sin\varphi, \\ y &= x'\sin\varphi + y'\sin\psi\cos\varphi - z'\cos\psi\cos\varphi, \\ z &= y'\cos\psi + z'\sin\psi. \end{aligned} \qquad (14.3)$$

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 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.

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 необхідність побудови такої поверхні, на якій би усувались невидимі лінії.

Шляхом перетину поверхні послідовністю паралельних площин $z = z_i$ ($i = 1,2,...,n$) 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 $y_k^{max}(x)$, і нижня $y_k^{min}(x)$ (два масиви, довжина яких дорівнює кількості точок x у просторі зображення). У цих масивах зберігатимуться поточні значення горизонтів на k-тому кроці.

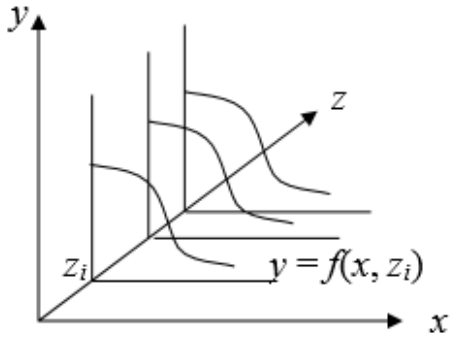


Рис. 14.2. Апроксимація поверхні лініями

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, тобто при k=1 маємо $y_k^{\max}(x) = f(x, z_1)$, $y_k^{\min}(x) = f(x, z_1)$,

Друга лінія теж повністю виводиться, а лінії горизонту корегуються так: верхня лінія горизонту в усіх своїх точках x прирівнюється до максимуму серед значень y для двох уже виведених ліній, а нижня лінія горизонту – до мінімуму ($y_k^{\max}(x) = \max(f(x, z_1), f(x, z_2))$, $y_k^{\min}(x) = \min(f(x, z_1), f(x, z_2), k=2)$.

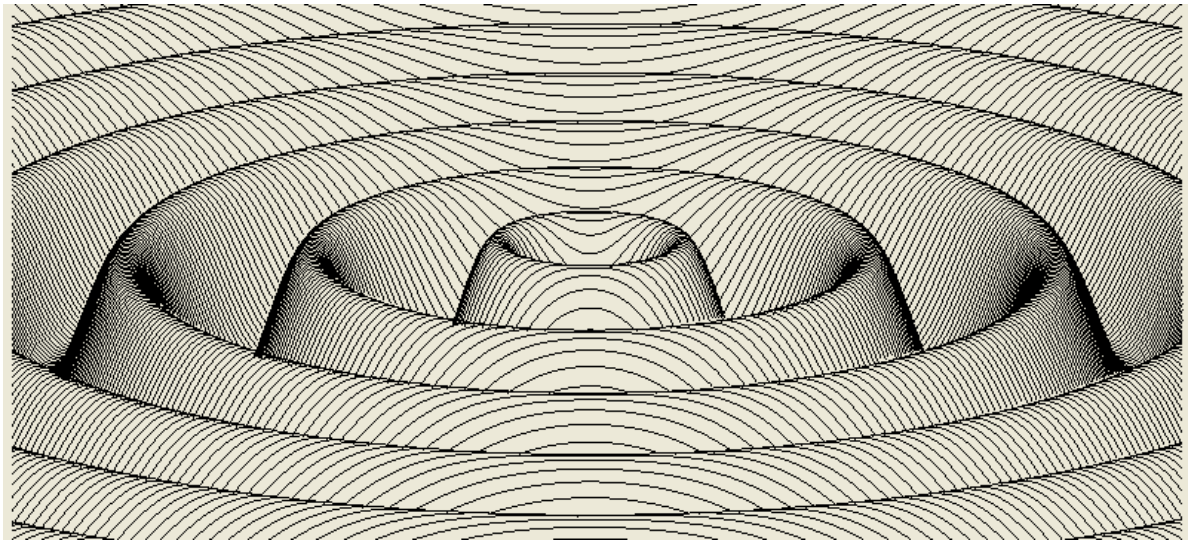
Область екрана між верхньою та нижньою лініями горизонту є проєкцією частини графіка функції $y = f(x, z)$ у смузі $z_1 < z < z_2$. Для інших ліній $y = f(x, z_i)$ при $i > 2$ ті частини ліній, які при проєктуванні попадають в область між двома лініями горизонту, невидимі. Ця область заборонена для подальшого втручання. Наступна лінія буде малюватися лише для тих x, для яких її проєкція лежить зовні області, що задається двома лініями горизонту. Тобто алгоритм тепер такий: якщо при деякому x значення y на поточній кривій більше або менше за відповідне значення верхнього горизонту, то точка y видима, інакше – невидима.

Позначимо проєкцію 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:

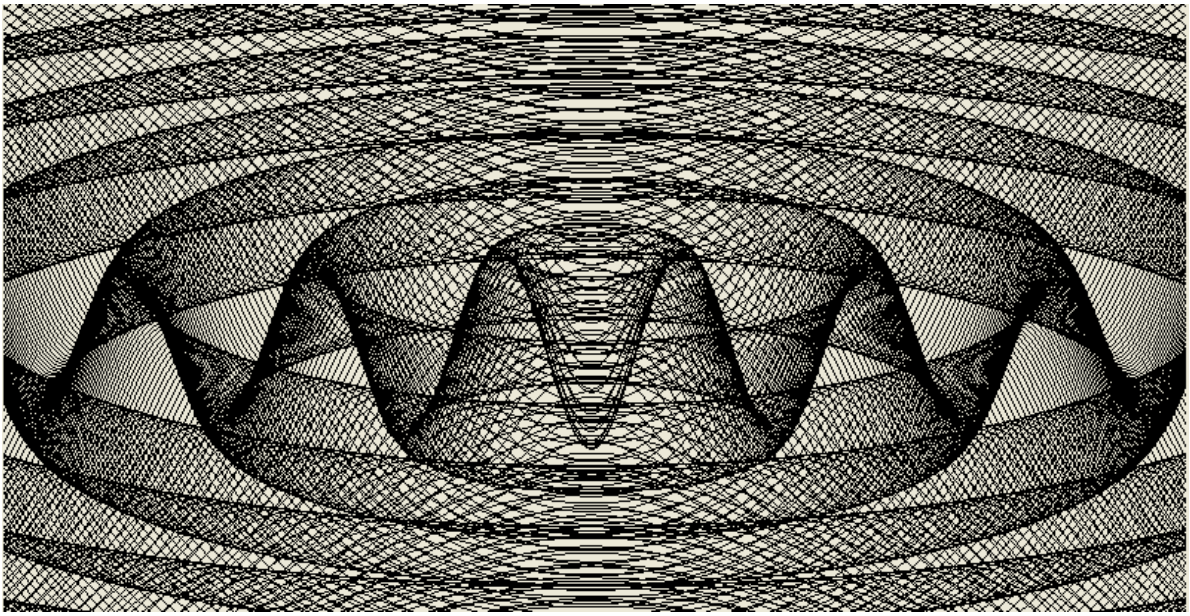
$$y_k^{\max}(x) = \max_{1 \le i \le k} y_i(x), \qquad y_k^{\min}(x) = \min_{1 \le i \le k} y_i(x).$$

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 $y_k^{\min}(x) \le y_k(x) \le y_k^{\max}(x)$, 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 подано 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 – з усуненням невидимих ліній



б – без усунення невидимих ліній

Рис. 14.3. Поверхня $z = f(x,y) = \dfrac{\cos\left(b\sqrt{x^2+y^2}\right)}{1+c\sqrt{x^2+y^2}}$

<u>Зауваження 1</u>. 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, потрібно 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).

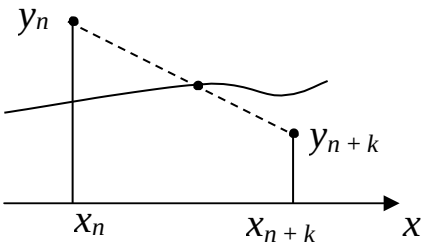


Рис. 14.4. Визначення видимої частини лінії

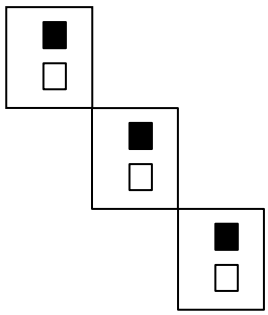


Рис. 14.5. Випадні пікселі

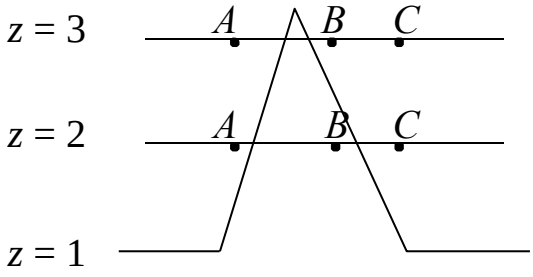


Рис. 14.6. Дефект алгоритму при наявності піків

<u>Зауваження 2.</u> Випадок відрізків ламаної з кутовим коефіцієнтом, за модулем більшим одиниці, вимагає спеціальної обробки відрізків, щоб не з'являлися випадні пікселі (рис. 14.5).

<u>Зауваження 3.</u> Якщо функція задана малою кількістю точок і має досить гострі піки, то наведений алгоритм може дати некоректні результати. Як приклад, розглянемо випадок ліній, зображених 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 уведення в масиви верхнього та нижнього горизонтів ординат, що відповідають штриховим лініям, які утворюються хибними бічними ребрами.

Уведення 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.

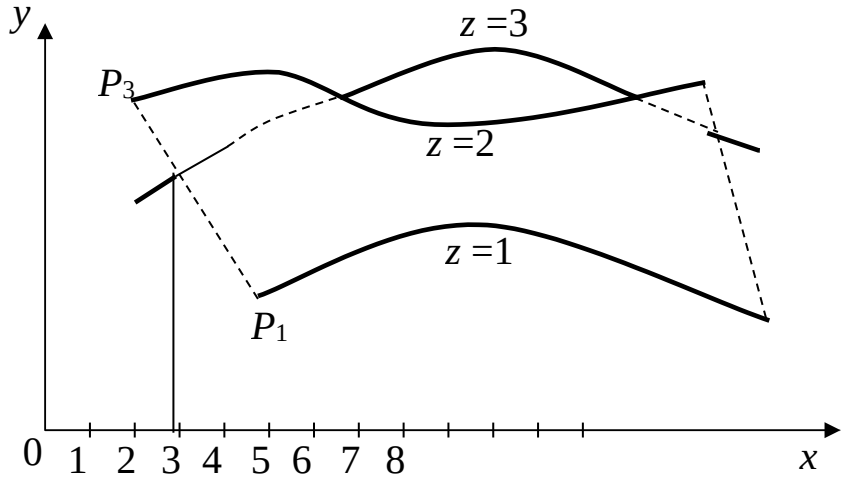


Рис. 14.7. Уведення хибних бічних ребер

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 багатогранних об’єктів, кожна грань яких є опуклим багатокутником, 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 проєкції, 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*) мають бути більшими за нуль або дорівнювати нулю. Якщо ці значення менші нуля, то необхідно зупинити програму. Надалі оперуватимемо тільки координатами у видовій системі координат.

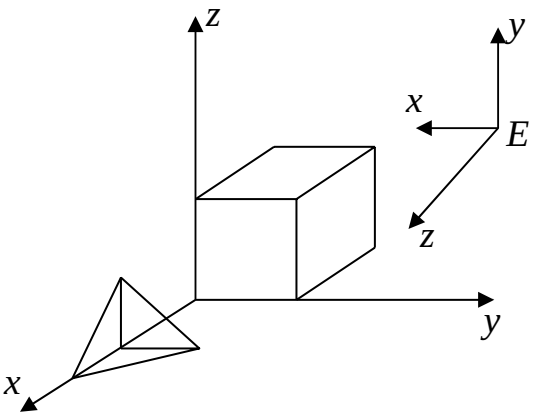


Рис. 14.8. Видові координати

Для цього алгоритму кожну грань об’єкта розбивають на трикутники (щоб програма була простішою, програміст сам розбиває грань на трикутники). Ці трикутники будуть використовуватися для з’ясування того, чи закривають вони відрізки прямих ліній. При цьому номери вершин у кожному трикутнику задаються в такій послідовності, щоб вони складали поворот проти годинникової стрілки, якщо дивитися на об’єкт зовні.

Список трикутників вигідно запам’ятати в деякому масиві – назвемо його *TRIANGLE*, кожний елемент якого матиме структуру

A B C 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

$$a = \begin{vmatrix} y_A & z_A & 1 \\ y_B & z_B & 1 \\ y_C & z_C & 1 \end{vmatrix}, \; b = -\begin{vmatrix} x_A & z_A & 1 \\ x_B & z_B & 1 \\ x_C & z_C & 1 \end{vmatrix}, \; c = \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix}, \; h = \begin{vmatrix} x_A & y_A & z_A \\ x_B & y_B & z_B \\ x_C & y_C & z_C \end{vmatrix}.$$

Розглянемо проєкції точок *A, B, C* на картинну площину. Відомо, що координати перспективної проєкції точки *A* на площину $z = d$ знаходяться за формулами

$$x'_A = \frac{dx_A}{z_A}, \qquad y'_A = \frac{dy_A}{z_A}.$$

Напрям 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

$$D = \begin{vmatrix} \frac{dx_A}{z_A} & \frac{dy_A}{z_A} & 1 \\ \frac{dx_B}{z_B} & \frac{dy_B}{z_B} & 1 \\ \frac{dx_C}{z_C} & \frac{dy_C}{z_C} & 1 \end{vmatrix} = d^2 \begin{vmatrix} x_A & y_A & z_A \\ x_B & y_B & z_B \\ x_C & y_C & z_C \end{vmatrix} / \left(z_A z_B z_C \right) = \frac{hd^2}{z_A z_B z_C}, \quad z_A, z_B, z_C > 0,$$

то *D* і *h* однакового знака. Тоді можливі такі випадки:

1) якщо $h = 0$, то площина *ABC* проходить через точку *E* і Δ *ABC* нічого не закриває;

2) якщо $h < 0$ (рівносильно $D < 0$), то Δ *ABC* – 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 багатогранника 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;

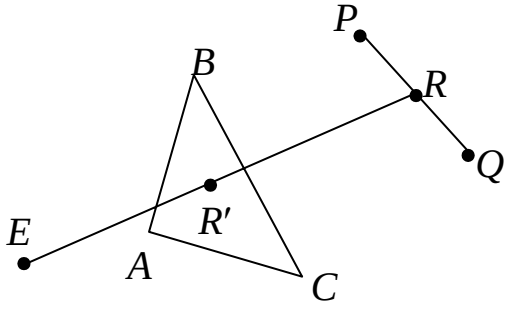


Рис. 14.9. Видимість точки *R*

3) якщо $h > 0$, то Δ *ABC* 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*.

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 з 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.

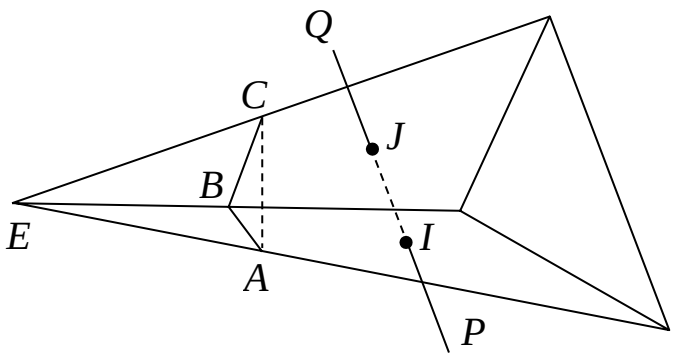


Рис. 14.10. Піраміда видимості

Якщо ΔABC закриває відрізок PQ, то PQ закритий і не повинен зображатися. При цьому немає необхідності виконувати перевірку для решти трикутників.

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, невидиме, 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

$$\begin{aligned} x &= x_P + \lambda(x_Q - x_P), \\ y &= y_P + \lambda(y_Q - y_P), \\ z &= z_P + \lambda(y_Q - z_P). \end{aligned} \qquad (14.4)$$

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

$$C_1x + C_2y + C_3z = 0, \qquad (14.5)$$

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

$$\lambda = -\frac{C_1x_P + C_2y_P + C_3z_P}{C_1(x_Q - x_P) + C_2(y_Q - y_P) + C_3(z_Q - z_P)}. \qquad (14.6)$$

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

$$\begin{aligned} x &= x_A + \mu(x_B - x_A), \\ y &= y_A + \mu(y_B - y_A), \\ z &= z_A + \mu(y_B - z_A), \end{aligned} \qquad (14.7)$$

а також рівняння площини PEQ

$$K_1x + K_2y + K_3z = 0, \qquad (14.8)$$

де

$$K_1 = y_Pz_Q - y_Qz_P, \quad K_2 = x_Qz_P - x_Pz_Q, \quad K_3 = x_Py_Q - x_Qy_P.$$

Із рівнянь (14.7), (14.8) знаходимо

$$\mu = -\frac{K_1x_A + K_2y_A + K_3z_A}{K_1(x_B - x_A) + K_2(y_B - y_A) + K_3(z_B - z_A)}. \qquad (14.9)$$

Отже, відрізок PQ і грань AEB піраміди мають спільну точку I тоді і тільки тоді, коли

$$0 \le \lambda \le 1, \; 0 \le \mu \le 1.$$

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 усередині, зовні піраміди або безпосередньо на піраміді.

Такі перевірки потрібно виконувати для кожної пари відрізок–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 всіх 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

$$ax_P + by_P + cz_P \le h,\ ax_Q + by_Q + cz_Q \le h.$$

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, *а*), отже, ребро PQ – видиме.

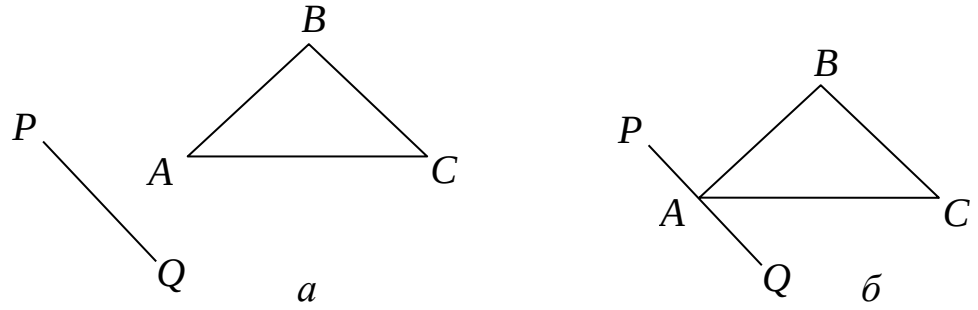


Рис. 14.11. *а* – модуль суми знаків = 3; *б* – модуль суми знаків = 2

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 *Pout* (точка P зовні).

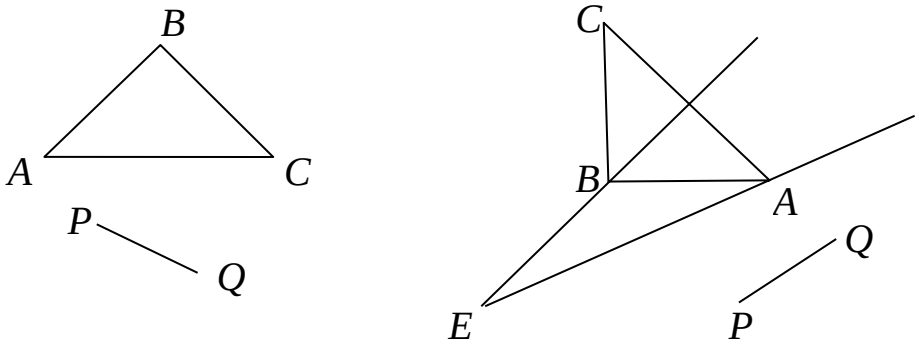


Рис. 14.12. Відрізок *PQ* видимий

Аналогічно для точки *Q* вводимо змінну *Qout*. Якщо дві точки *P*, *Q* лежать за площиною *AEB*, то відрізок *PQ* видимий (рис. 14.13, *а*). Відрізок *PQ* буде видимим, якщо одна з точок лежить у площині *AEB*, а інша точка видима (рис. 14.13, *б*).

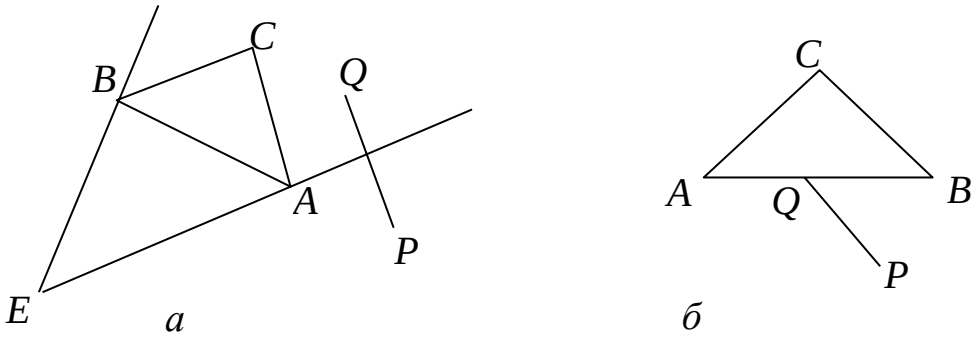


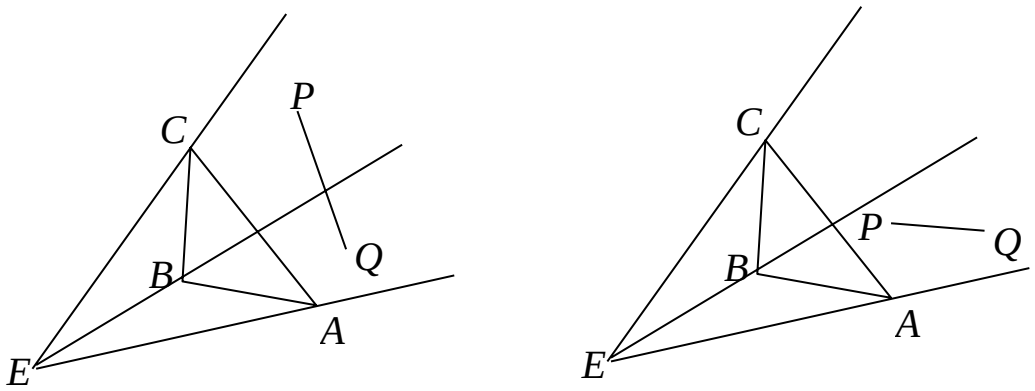
Рис. 14.13. Тест 3: *а* – точки *P, Q* видимі; *б* – точка *P* видима, *Q* належить площині *AEB*

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 ΔABC (рис. 14.14, *а*), а отже, невидимий.

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.

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).



а *б*

Рис. 14.14. Тест 4: *а – PQ* позаду Δ*ABC*; *б – PQ* повністю закривається відрізком *AB*

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

$$0 \le \lambda_i \le 1,\ 0 \le \mu_i \le 1,\ i = 1, 2, 3.$$

Ці значення позначимо $\lambda_{\min}$, $\lambda_{\max}$.

Якщо при деякому $0 \le \lambda_i \le 1$ знайдена з формули (14.4) точка *I* (точка перетину *PQ* з гранню піраміди) лежить перед Δ*ABC*, то *PQ* – видимий, оскільки *PQ* не може проходити через внутрішні точки Δ*ABC*.

Тест 6. Цей тест виконується тоді, коли всі попередні тести не виконалися, тобто коли відрізок *PQ* перетинає піраміду позаду Δ*ABC*.

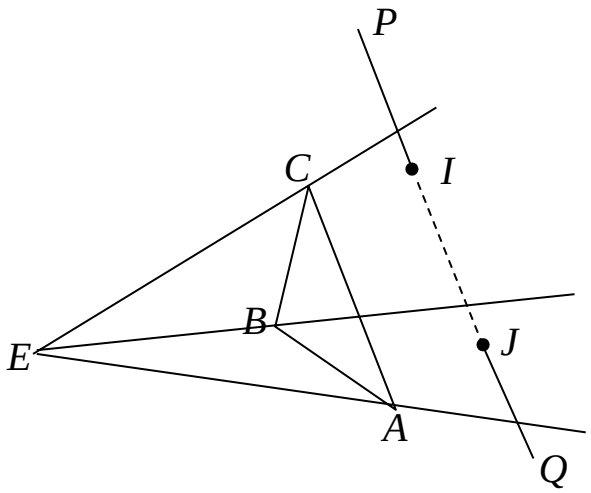


Рис. 14.15. *IJ* – невидима частина відрізка *PQ*

Якщо $\lambda_{\min} \ne \lambda_{\max}$, 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 *Pout, Qout*.

У цьому тесті можуть з’явитися два нових відрізки *PI* і *JQ*, які є видимими відносно Δ*ABC*, тому надалі необхідно двічі звертa-

тися 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 точкою 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 abs(x – a) <= epsilon, де epsilon – мала додатна величина.

Нині в зв'язку з широким розповсюдженням растрових пристроїв найбільш розповсюджені алгоритми, які працюють у 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. Отже, 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.

У 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 такий (варіант програми на мові C++ наведений у додатку):

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 проєкцію 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 $Z_{ij}^{буф}$ присвоюють значення z_{ij}, заносячи 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)*.
5. Беруть наступну необроблену грань, причому порядок не має жодного значення, та переходять до кроку 2.

Досить ефективне в цьому методі суміщення побудови растрової розгортки з виведенням у *Z*-буфер, при цьому для обчислення глибини пікселів можуть застосовуватися інкрементні методи.

Грань об'єкта зображується стрічка за стрічкою; для знаходження необхідних значень використовується лінійна інтерполяція (рис. 14.16).

$$x_a = x_1 + (x_2 - x_1)\frac{y - y_1}{y_2 - y_1}$$

$$x_b = x_1 + (x_3 - x_1)\frac{y - y_1}{y_3 - y_1}$$

$$z_a = z_1 + (z_2 - z_1)\frac{y - y_1}{y_2 - y_1},$$

$$z_b = z_1 + (z_3 - x_1)\frac{y - y_1}{y_3 - y_1}.$$

$$z = z_a + (z_b - z_a)\frac{x - x_a}{x_b - x_a}.$$

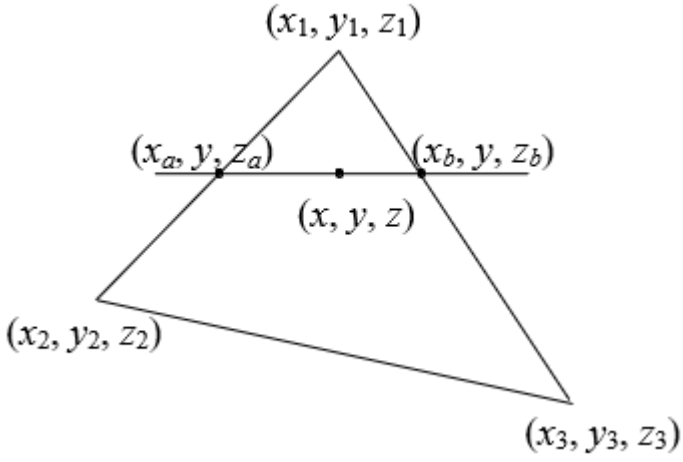


Рис. 14.16. Лінійна інтерполяція грані об'єкта

Рис. 14.17. Приклад роботи методу *Z*-буфера

Метод *Z*-буфера працює виключно в просторі картинної площини, тобто в просторі зображень (рис.14.17). 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 *xy*. Наприклад, буфер кадру розмірності 512×512×24 бітів у комбінації з *Z*-буфером розмірності 512×512×20 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 кількох 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 цей піксель. При використанні

складних моделей освітлення і текстур ці обчислення можуть вимагати багато часових затрат. Тому існують кілька 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 $z_{ij} < Z_{ij}^{буф}$ замінити порівнянням $z^{пер} \le z_{ij} < Z_{ij}^{буф}$, де $z^{пер}$ – глибина шуканого перетину, то автоматично можна одержати зображення перетину поверхні площиною $z = z^{пер}$.

14.5. Огляд деяких інших методів

14.5.1. Метод відсікання нелицьових граней

Розглянемо багатогранник, для кожної грані якого заданий одиничний вектор зовнішньої нормалі ***n*** (рис. 14.18).

Якщо вектор зовнішньої нормалі ***n*** грані утворює з вектором ***l***, що задає напрям проектування, тупий кут, тобто вектор нормалі направлений у 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 $(\boldsymbol{n} \cdot \boldsymbol{l}) < 0$ (рис. 14.19, *а*).

При центральному проектуванні з центром у точці *C* вектор про- ектування для точки *P* визначається зі співвідношення $\boldsymbol{l} = P - C$. Тоді, щоб визначити, чи грань лицьова, достатньо взяти будь-яку точку *P* цієї грані і перевірити виконання умови $(\boldsymbol{n} \cdot \boldsymbol{l}) < 0$ (рис. 14.19, *б*).

Для опуклих багатогранників цей метод дає повний розв’язок задачі усунення невидимих граней (достатньо видалити всі нелицьові грані). Для неопуклих багатогранників додатково аналізують видимість лицьових граней.

<u>Зауваження 5</u>. Зовнішні нормалі граней можна знаходити як векторний добуток відповідних векторів, що лежать у цій грані.

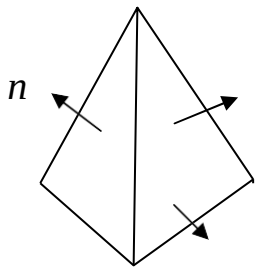


Рис. 14.18. Зовнішні нормалі багатогранника

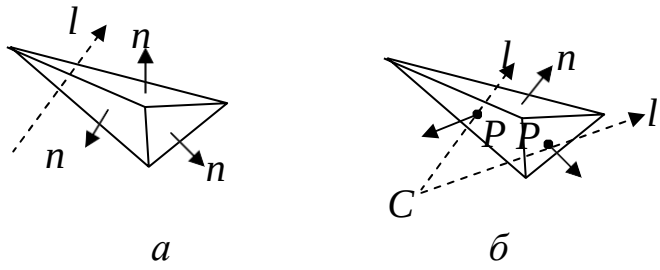


Рис. 14.19. Визначення видимості грані при проектуванні: *а* – паралельному; *б* – центральному

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 із 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, розбиваючи її на вікна меншого розміру.

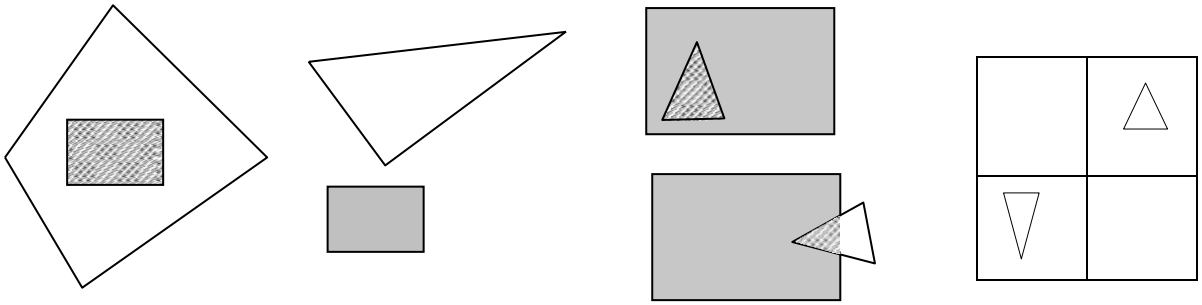
Вміст активного (прямокутника) вікна аналізують на взаємне розташування з проєкціями елементів 3D-сцени (найчастіше з багатокутниками, що є проєкціями граней багатогранників). Процедура прийняття рішення застосовується в кожній малій області. Якщо вікно порожнє або його вміст достатньо простий, то відбувається візуалізація вікна. Якщо ця умова не виконується, то вікно розбивається на частини доти, поки вміст вікна не стане досить простим. По мірі того, як розміри вікна зменшуються, кожну таку область перетинає все менша і менша кількість багатокутників. При такому підході робота ведеться в просторі зображення. При цьому ефективне використання списку тих об’єктів, проєкції 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 проєкцією найближчого до спостерігача багатокутника (багатокутник охоплює вікно), то вікно зафарбовується кольором проєкції цього багатокутника (рис. 14.20, *а*);
- ✓ якщо до вікна не потрапила жодна з проєкцій елементів сцени (всі багатокутники сцени зовнішні відносно вікна), то його зафарбовують кольором фону (рис. 14.20, *б*);
- ✓ якщо існує єдиний внутрішній (багатокутник повністю знаходиться у вікні) або єдиний перетинаючий багатокутник, то все вікно зафарбовується кольором фону, а потім частина вікна, яка відповідає багатокутнику, зафарбовується кольором багатокутника (рис. 14.20, *в*);
- ✓ якщо не виконалася жодна з умов, то вікно розбивається на чотири частини та для кожної з них повторюються попередні тести

(рис. 14.20, *г*). Подрібнення вікна можна виконувати до розміру пікселя.



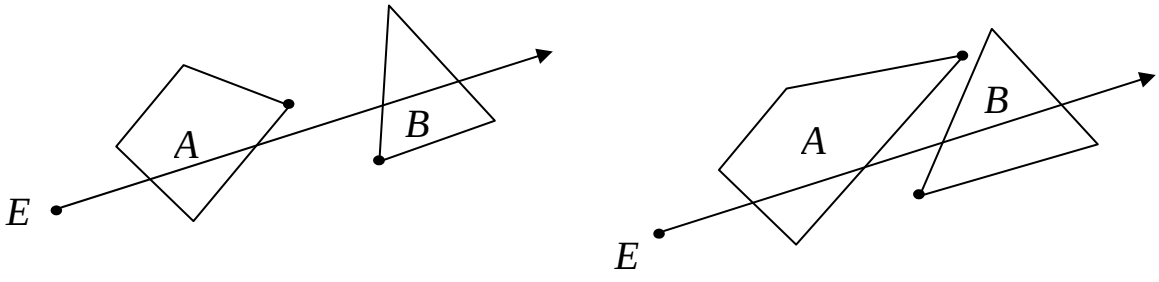
а *б* *в* *г*

Рис. 14.20. Розміщення вікна та проєкції 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 проєктуванні може закривати 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. 14.21, *а*). Це означає, що спочатку виводиться грань *B*, а потім грань *A*. У випадку, коли проектування ведеться вздовж осі *z*, з умови *AB* випливає, що їхні *z*-оболонки не перетинаються.



а *б*

Рис. 14.21. Упорядкування граней за глибиною
(*E* – точка спостерігача)

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) в 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. 14.21, *б*). В цьому випадку грань *A* може виводитися швидше за грань *B*, 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 жодних обмежень на складність сцени і тип багатокутників.

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 багатокутник через *P*, а наступний – через *Q*. Тепер для кожного багатокутника *P* необхідно перевірити його розміщення по відношенню до наступних багатокутників *Q*.

При $z_{max}(Q) \le z_{min}(P)$ (рис. 14.22, *а*) ніяка частина *P* не може закрити *Q*. У цьому випадку багатокутники відсортовані правильно, тому спочатку виводимо багатокутник *P*, а потім *Q*.

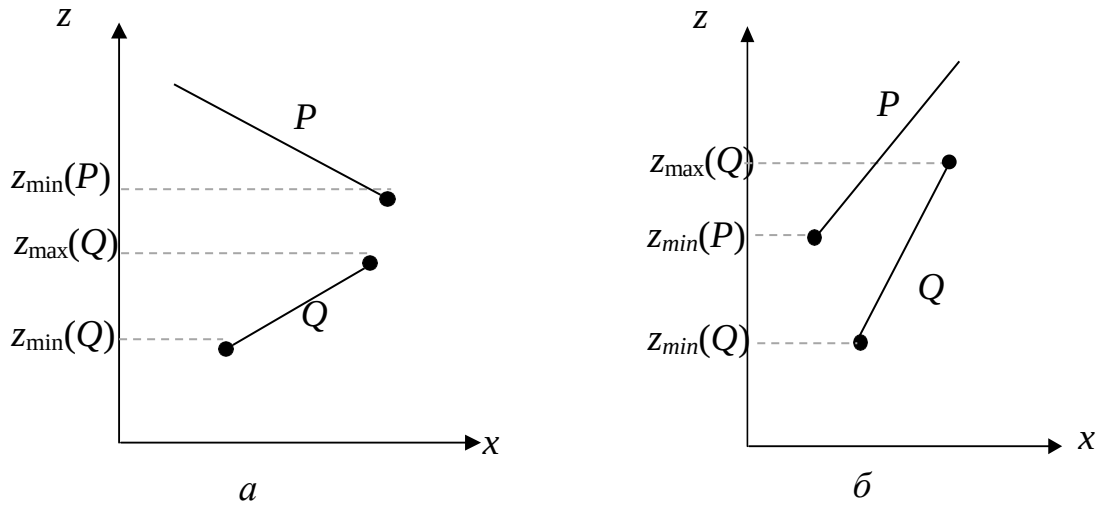


Рис. 14.22. Сортування багатокутників за z_{min} ($z_{min}(P) \ge z_{min}(Q)$)

Для сцени, що зображена на рис. 14.22, *б* ($z_{max}(Q) \ge z_{min}(P)$), 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 може бути не так. У цьому випадку P і Q в списку потрібно поміняти місцями.

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 багатокутників не перетинаються, то й самі багатокутники та їх проєкції не перетинаються;
2) якщо y-оболонки багатокутників не перетинаються, то й самі багатокутники та їх проєкції 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 знаходяться грань P і спостерігач по різні боки від площини, що проходить через грань Q (рис. 14.23, *а*);
4) чи знаходяться грань Q і спостерігач по один бік від площини, що проходить через грань P (рис. 14.23, *б*).

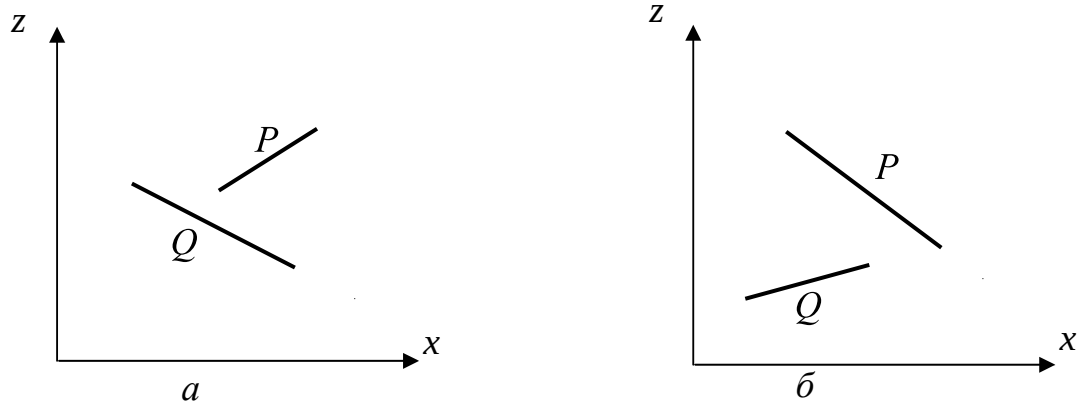


Рис. 14.23. Тести перекриття багатокутників

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, для яких z-оболонки перетинаються.

У випадку, коли жоден із тестів не виконався (рис. 14.24, *а*), міняємо місцями грані P, Q ($P \to Q'$, $Q \to P'$) і знову провіряємо тести, аналогічні 3), 4), тобто

3′) чи знаходяться грань P' і спостерігач по різні боки від площини, що проходить через грань Q';

4′) чи знаходяться грань Q' і спостерігач по один бік від площини, що проходить через грань P'.

Якщо на одне з цих питань отримана позитивна відповідь, то P' передує Q' і грань $P'=Q$ буде виводитися раніше (рис. 14.24, *б*).

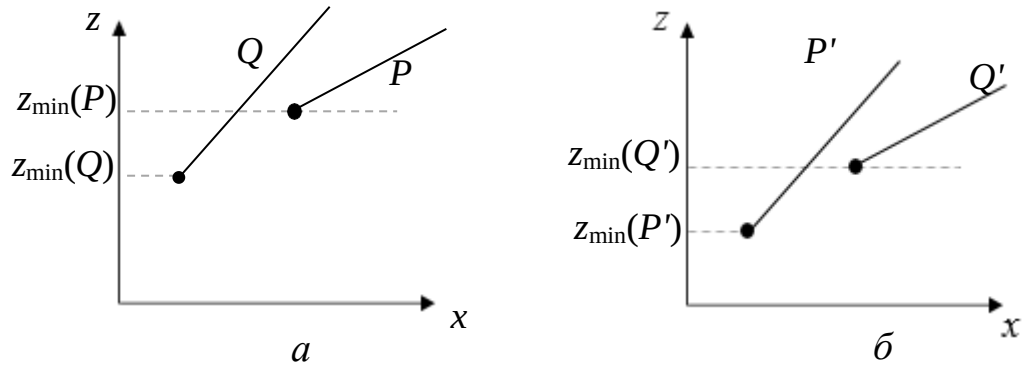


Рис. 14.24. Перевірка тестів 3), 4), 3'), 4')

Водночас зустрічаються випадки, коли задані грані упорядкувати не вдається (рис. 14.25), 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 вирішується за допомогою тестів 3), 4) або 3′), 4′).

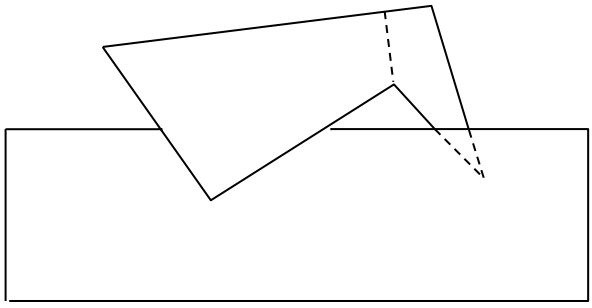


Рис. 14.25. Циклічне перекриття багатокутників

<u>Зауваження 6.</u> Методу 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.

<u>Зауваження 7.</u> Останнім часом з'явилися і більш прийнятні методи розв'язування задач загороджування: метод BSP-дерев, метод октодерев, метод порталів та ін.

Контрольні запитання 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 полягає суть методу *Z*-буфера?
8. Назвіть недоліки та переваги алгоритму *Z*-буфера.
9. Як визначити зовнішні нормалі до граней багатогранника?
10. Поясніть способи розміщення багатокутника відносно вікна.
11. Які тести перевіряє алгоритм Варнока?
12. Як можна легко впорядкувати багатокутники 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 $\{f(x_i, y_j)$, $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m\}$, 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 розміщення багатокутника відносно прямокутного вікна. Розробити такі алгоритми. *Вказівка. Узагальнити габаритний, променевий та кутовий тести на випадок прямокутного вікна*.
5. Розробити тест визначення нормалі грані багатогранника.
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.

Розділ 15. Зафарбовування видимих поверхонь

15.1. Основні поняття

При побудові тривимірних графічних зображень на площині дотримуються певної послідовності дій. Послідовність операцій для побудови високоякісних реалістичних 2D-зображень з 3D-сцени називається ***3D-конвеєром***.

3D-конвеєр реалізується в кілька етапів (стадій). Виділяють етапи опису тривимірних зображень, геометричних перетворень і рендерінгу (рис 15.1). Слово рендерінг вживають для позначення процесу візуалізації, а рендер для позначення готового зображення.

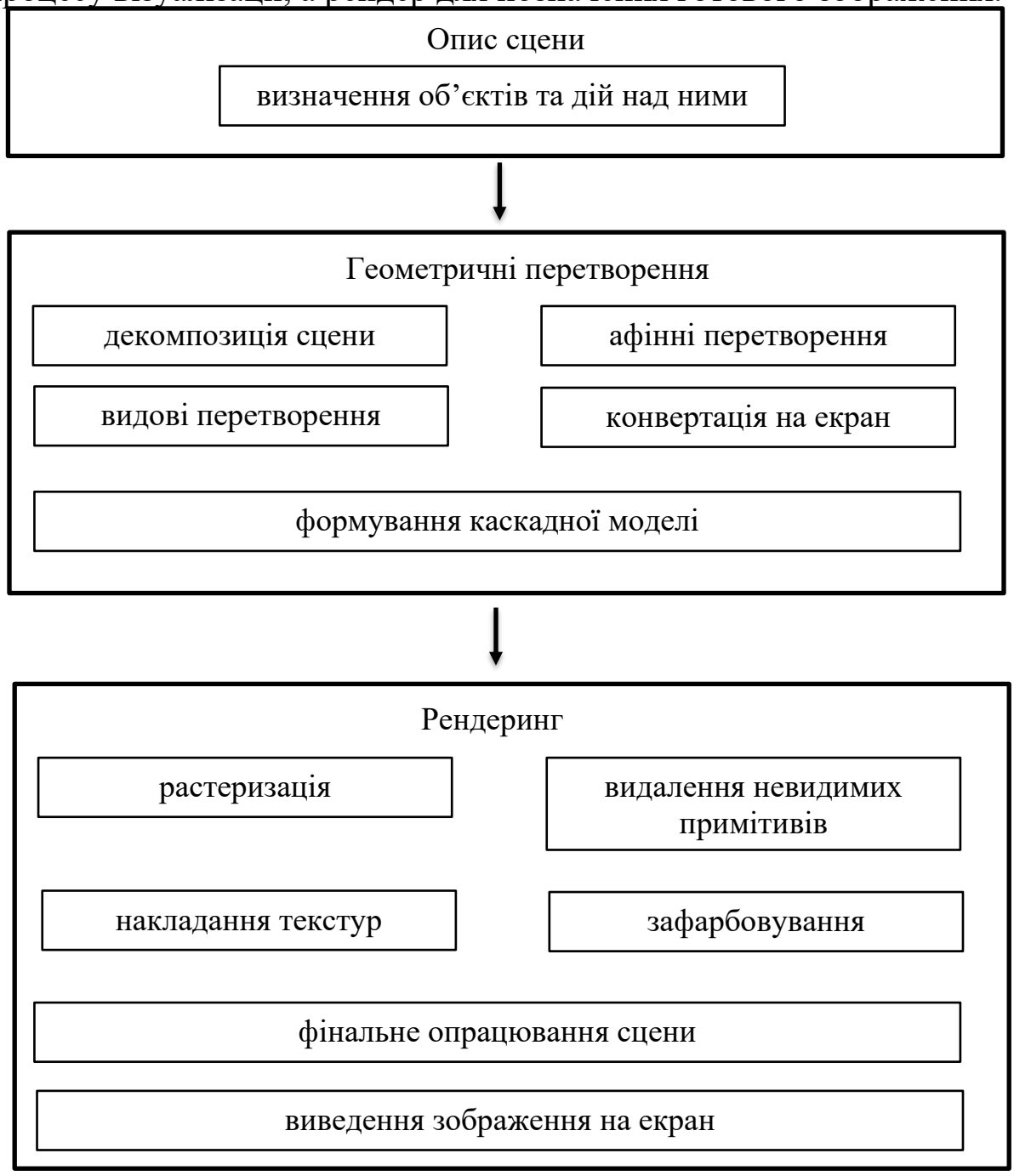


Рис. 15.1. Основні етапи графічного конвеєра

На початку цієї послідовності, тобто на етапі опису сцени, за допомогою математичних моделей визначають 3D-об’єкти, які відображатимуться в сцені, їх взаємне розміщення та наступні дії над об’єктами. Для моделювання об’єктів можна брати сплайнові криві та поверхні, однак частіше за все використовують полігони.

На наступному етапі геометричних перетворень, виходячи з математичної моделі об’єктів, здійснюється декомпозиція сцени та афінні перетворення над об’єктами, створюється зовнішній вигляд об’єктів у вигляді 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 проєктування сцени зі збереженням інформації про відстань до спостерігача.

Кінцевий зовнішній вигляд об’єкта формується на етапі, який називається рендерінгом. ***Рендерінг*** (від англ. ”rendering” – візуалізація, вимальовування) – це процес створення високореалістичних зображень 3D-об’єктів на основі їх математичних моделей. Фундаментом рендерінгу є векторна математика. Слово *рендер* інколи використовують для позначення готового зображення.

Рендерінг дозволяє досягти високого фотореалізму при створенні ігор. Сучасна 3D-анімація неможлива без рендерінгу. 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, накладання текстур, зафарбовування примітивів і фінальна обробка зображення як єдиного цілого. На етапі рендерінгу формуються пікселі зображення, для яких визначені екранні координати та компоненти кольору. Для розв’язування задач рендерінгу використовуються опис світлового потоку в заданій системі. Етап рендерінгу у графічному конвеєрі найбільш трудомісткий, оскільки він пов’язаний із попіксельною обробкою і складними обчисленнями. Рендерінг у загальному обсязі обчислень із формування зображень тривимірних сцен складає 60-80%, тому саме він визначає продуктивність графічної системи. Хто виконує ці стадії? Сама програма (як правило, початок конвеєра), бібліотека прикладного програмування (інтерфейс API) і прискорювач.

На сьогодні розроблено багато алгоритмів візуалізації. Програмне забезпечення може використовувати одночасно кілька алгоритмів для отримання кінцевого зображення.

Існує спеціальне програмне забезпечення, що дозволяє реалізувати процес рендерінгу. Програму, яка виконує рендерінг і створює зображення, теж називають ***рендером***. Рендер обробляє масу даних, які визначають текстуру, колір, матеріал, освітленість, тіні тощо. Без рендерів неможливо створювати ігри, мультфільми, кінофільми з реалістичною графікою. Сюди, наприклад, можна віднести програми Sds Max, Maya (добре моделює персонажі для ігор). Рендери використовують різні спеціалісти, зокрема, архітектори для візуалізації проєктів, маркетологи в рекламних кампаніях тощо.

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. Існують 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. Колір визначається на основі інформації про освітленість і накладання текстур. У результаті на основі попередньої інформації в буфері кадра відбувається відбувається побудова результуючого зображення.

Синтез реалістичних зображень, особливо в режимі реального часу, передбачає великий обсяг обчислень, який визначається складністю геометрії та швидкістю зміни кадрів. Так, при моделюванні зображення винищувача F-14 модель формується з 76 194 полігонів і включає 40 678 вершин. Підраховано, що при частоті зміни кадрів 25 кадрів/с необхідна швидкість обчислень складає122 922 000 операцій/с з плаваючою точкою і приблизно $1{,}9 \cdot 10^{11}$ операцій/с із фіксованою точкою. Тому сучасна фотореалістична графіка – це сплав обчислювальної фізики, математичних методів та витончених алгоритмів.

Спочатку розглянемо деякі фізичні властивості та моделі освітлення.

15.2. 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 повністю поглинає 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 в усіх напрямках, дзеркальне відбиття відбувається вздовж одного променя.

Дійсна теорія світла досить складна 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.

15.2.1. Дзеркальне відбиття світла

До дзеркальних належать 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 $\boldsymbol{S}$ збігається з вектором відбиття $\boldsymbol{R}$ і кут $\alpha = 0$ (рис 15.2).

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 не перевищує 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 вектора, направленого на джерело світла (кута падіння світла θ), а також вектора відбиття світла та вектора, спрямованого на спостерігача, від довжини хвилі λ і від властивостей матеріалу. Існують різні моделі обчислення інтенсивності відбитого світла (відблискової складової).

До простіших моделей дзеркального освітлення (відблискових моделей) відносять емпіричну модель Буі–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. 15.2); 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 відблиски). Великі значення p задають сфокусовані розподіли для металів, малі p – для неметалевих поверхонь. Варто зазначити, що відбитий від металу дзеркальний промінь зберігає властивості падаючого променя. Наприклад, при освітленні блискучої синьої поверхні білим світлом виникають білі, а не сині відблиски.
Відблиски суттєво збільшують реалістичність зображення, оскільки всі реальні поверхні відбивають світло. Тому відблискова складова моделі освітлення важлива, особливо при моделюванні динаміки об’єктів, оскільки за відблисками визначають зміну положення камери або самого об’єкта.

Функція $\omega(\theta, \lambda)$ досить складна, через це її часто заміняють константою k_s, яка називається *коефіцієнтом дзеркального відбиття* і визначається експериментально. Тобто на практиці модель Фонга використовується у формі

$$I_s = I\, k_s \cos^p \alpha, \qquad 0 < k_s < 1. \tag{15.1}$$

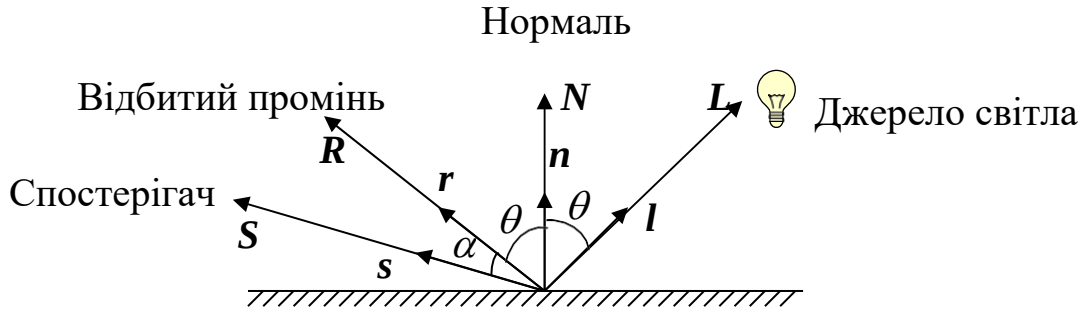


Рис. 15.2. Дзеркальне відбиття світла: ***r***, ***s***, ***n***, ***l*** – одиничні вектори

15.2.2. 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 притаманний паперу, гіпсу, піску. Якщо поверхня слабо шорсткувата, 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 залежить від розміщення джерела світла і нормалі до точки поверхні, а саме пропорційна косинусу кута θ між напрямком $\boldsymbol{L}$ від точки поверхні на джерело світла та нормаллю $\boldsymbol{N}$ до поверхні в цій точці (рис.15.3):

$$I_d = I\ k_d \cos\theta, \quad 0 < \theta < \frac{\pi}{2}, \tag{15.2}$$

де I – інтенсивність точкового джерела світла; k_d – коефіцієнт дифузного відбиття, що залежить від властивостей матеріалу поверхні ($0 \le k_d \le 1$); 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 0^0. Поверхня предметів, що зображається за допомогою моделі освітлення Ламберта (з дифузним відбиттям), виглядає бляклою і матовою. Світло вже не заповнює поверхню однаковим відтінком.

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, прийнятній для конкретного матеріалу поверхні.

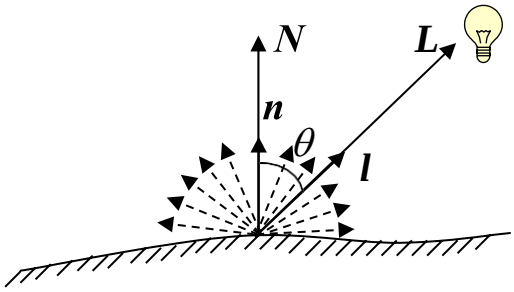


Рис. 15.3. Дифузне відбиття

У цьому випадку модель відбиття світла записують у вигляді суми дифузної і дзеркальної компоненти

$$I^* = I(k_d \cos\theta + k_s \cos^p \alpha). \qquad (15.3)$$

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. 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 фонового 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:

$$I^* = I_a\, k_a + I(k_d \cos\theta + k_s \cos^p \alpha), \qquad (15.4)$$

де I_a – інтенсивність фонового розсіяного світла (постійна для всіх об'єктів сцени), $0 \le k_a \le 1$ – коефіцієнт відбиття поверхнею розсіяного світла, k_d, k_s, як і раніше, коефіцієнти дифузного та 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
полірованесрібло	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 далі від джерела світла має бути темнішим). Тому в комп'ютерній графіці для більшої реалістичності використовують модель освітлення з лінійним затуханням

$$I^* = I_a\, k_a + \frac{I}{d+k}(k_d \cos\theta + k_s \cos^p \alpha), \qquad (15.5)$$

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* $\frac{1}{k + c_1 d + c_2 d^2}$*, де параметри* 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 кілька (наприклад, m), то їх ефекти підсумовуються. В цьому випадку модель інтенсивності освітлення має вигляд

$$I^* = I_a\, k_a + \sum_{j=1}^{m} \frac{I_j}{d_j + k}(k_d \cos\theta_j + k_s \cos^{p_j} \alpha_j). \qquad (15.6)$$

Використовуючи одиничні вектори зовнішньої нормалі $\boldsymbol{n}$, а також одиничні вектори, що визначають напрями на джерело світла (вектор $\boldsymbol{l}$), відбитого променя (вектор $\boldsymbol{r}$), на спостерігача (вектор $\boldsymbol{s}$) (рис. 15.1), функцію зафарбовування (15.5) можна записати у вигляді

$$I^* = I_a\, k_a + \frac{I}{d+k}(k_d\, (\boldsymbol{n} \cdot \boldsymbol{l}) + k_s (\boldsymbol{r} \cdot \boldsymbol{s})^p), \qquad (15.7)$$

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 напряму 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 $\boldsymbol{m}$, напрямленим від джерела світла до точки поверхні (рис. 15.4); 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 формулу

$$I^* = \left(I_a k_a + \frac{I}{d+k}(k_d(\boldsymbol{n}\cdot\boldsymbol{l}) + k_s(\boldsymbol{r}\cdot\boldsymbol{s})^p) \right) \cos^t\varphi.$$

Зауважимо, що існують більш складні моделі освітлення, наприклад моделі Варна та Торренса–Сперроу.

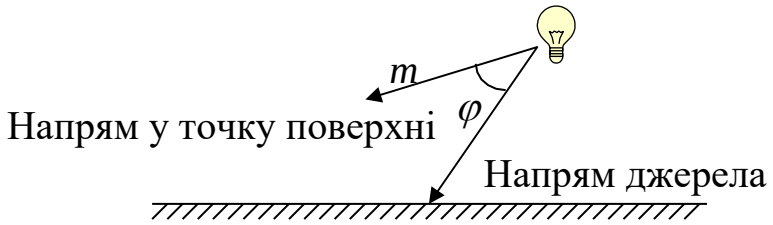


Рис. 15.4. Сфокусоване джерело світла

15.3. Обчислення нормалей до поверхні відбиття світла

Вектори $\boldsymbol{l}$ та $\boldsymbol{s}$ задаються, а вектори $\boldsymbol{n}$ та $\boldsymbol{r}$ 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 багатогранної поверхні вектори зовнішніх нормалей можна знайти тільки для граней, тобто якщо грані поверхні лежать у площинах, що описуються рівняннями

$$a_i x + b_i y + c_i z + d_i = 0,\ i = 1, 2, \ldots, m,$$

то нормальні вектори цих площин (a_i, b_i, c_i) є векторами зовнішніх нормалей для багатогранної 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 у вершинах багатогранника, 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, …, m, тоді за нормальний вектор у вершині багатогранника можна взяти суму векторів, тобто (a, b, c)= $= \sum_{i=1}^{m} (a_i, b_i, c_i)$ (рис. 15.5). Інколи для знаходження вектора нормалі в вершині використовують нормовану зважену суму векторів нормалі граней, яким належить ця вершина.

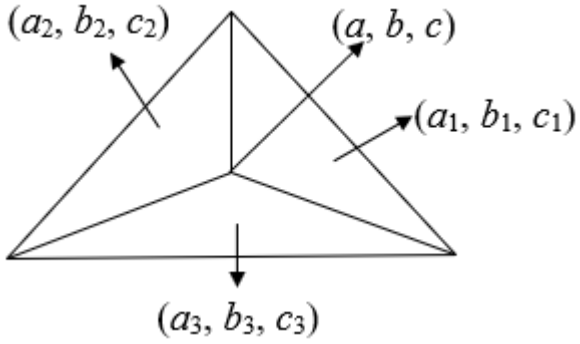


Рис. 15.5. Нормаль у вершині при m = 3

Для визначення напрямку нормалі в точці, що належить ребру багатогранної поверхні, достатньо додати вектори нормалей тих граней, які прилягають до даного ребра. А можна зробити по-іншому, тобто апроксимувати змінний вектор нормалі вздовж ребра багатогранної поверхні можна за допомогою вже знайдених векторів зовнішніх нормалей у вершинах, що утворюють дане ребро. Якщо багатогранна поверхня задана своїми вершинами V_1,V_2,V_3,V_4, то вектор зовнішньої нормалі в її вершинах можна знайти, використовуючи векторні добутки, 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. 15.6).

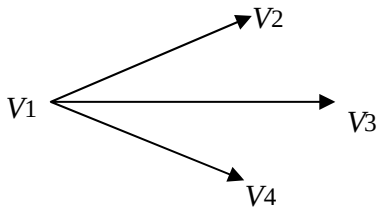


Рис. 15.6. Визначення вектора зовнішньої нормалі

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

$$\cos\theta = \frac{(n \cdot l)}{|n| \cdot |l|}. \qquad (15.8)$$

У випадку, коли джерело світла розміщується в скінченній точці простору (x_c, y_c, z_c), для визначення $\cos\theta$ потрібно спочатку знайти вектор, направлений на джерело світла.

Вивчимо питання про те, як знайти напрямок відбитого променя $\boldsymbol{r}$ та $\cos\alpha$. Будемо вважати, що заданий вектор $\boldsymbol{l}$ напрямлений на джерело світла, а також маємо вектор нормалі $\boldsymbol{n}$.

Як відомо, при дзеркальному відображенні вектори $\boldsymbol{l}$, $\boldsymbol{r}$, $\boldsymbol{n}$ лежать в одній площині, причому кути між векторами $\boldsymbol{l}$, $\boldsymbol{n}$ та $\boldsymbol{n}$, $\boldsymbol{r}$ однакові (рис. 15.2).

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

$$r_x = 2n_x n_z, \qquad r_y = 2n_y n_z, \qquad r_z = 2n_z^2 - 1. \qquad (15.9)$$

Знайдемо вектор $\boldsymbol{r}$ у загальному випадку (рис. 15.7).

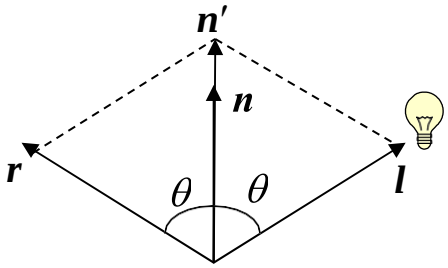


Рис. 15.7. Вектори $\boldsymbol{r}$, $\boldsymbol{l}$, $\boldsymbol{n}$ одиничної довжини

Вектор $\boldsymbol{r} + \boldsymbol{l} = \boldsymbol{n}'$ є діагоналлю ромба. Напрям $\boldsymbol{n}'$ збігається з $\boldsymbol{n}$, причому $|\boldsymbol{n}'| = 2\cos\theta$, тому $\boldsymbol{n}' = 2\boldsymbol{n}\cos\theta$, тобто $\boldsymbol{r} + \boldsymbol{l} = 2\boldsymbol{n}\cos\theta$. Звідси знаходимо одиничний вектор відбитого променя

$$\boldsymbol{r} = 2\boldsymbol{n}\cos\theta - \boldsymbol{l}.$$

Враховуючи, що $\cos\theta = (\boldsymbol{l} \cdot \boldsymbol{n})$, одержуємо

$$\boldsymbol{r} = 2\boldsymbol{n}(\boldsymbol{l} \cdot \boldsymbol{n}) - \boldsymbol{l},$$

а у скалярній формі

$$r_x = 2n_x(l_x n_x + l_y n_y + l_z n_z) - l_x,$$
$$r_y = 2n_y(l_x n_y + l_y n_y + l_z n_z) - l_y,$$
$$r_z = 2n_z(l_x n_y + l_y n_y + l_z n_z) - l_z.$$

Тепер уже можна обчислити $\cos\alpha = (\boldsymbol{s} \cdot \boldsymbol{r})$, де $\boldsymbol{s}$ – одиничний вектор, направлений на спостерігача.

15.4. 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 освітленості вздовж границь об'єктів. Найбільш відомі методи побудови згладжених зображень – це методи Гуро та Фонга. Більш простий із них метод Гуро, він дозволяє сформувати графічне зображення із задовільною реалістичністю. Метод Фонга дозволяє отримати більш реалістичні графічні зображення. Розглянемо ці методи.

15.4.1. 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 уся грань зафарбовується однаковим кольором, що визначається знайденою освітленістю.

Запропонований алгоритм має великий недолік, оскільки побудоване при цьому зображення має яскраво виражений багатогранний 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 багатогранник, 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.

15.4.2. Метод Гуро

Метод Гуро використовується для створення ілюзії гладкої криволінійної поверхні, яка задається у вигляді багатогранників 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 білінійно 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. 15.6);
- 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 полігона.

Нехай задана грань *ABC* (рис. 15.8) в екранних координатах. Використовуючи моделі освітлення, знайдемо інтенсивності відбитого світла у вершинах – нехай це будуть значення I_A, I_B, I_C. Обчислимо освітленість у точці *P*. Зображаючи грань рядок за рядком, будемо знаходити значення інтенсивності в кінцях кожного горизонтального відрізка *MN* шляхом лінійної інтерполяції значень вздовж ребер.

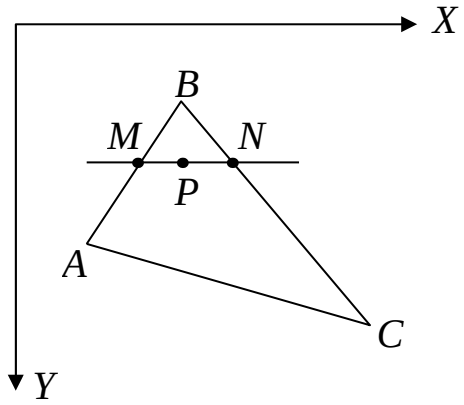


Рис. 15.8. Заповнення грані

Так, освітленість у точці *M* обчислимо за формулою

$$I(M) = I(A)\,t + I(B)\,(1 - t),\ t = \frac{|MA|}{|AB|}.$$

Аналогічно обчислюємо $I(N)$. Маючи значення $I(M)$, $I(N)$, за допомогою лінійної інтерполяції знаходимо $I(P)$:

$$I(P) = I(M)(1 - s) + I(N)s, \quad s = \frac{|PM|}{|MN|}, \text{ або } s = \frac{|X_P - X_M|}{|X_N - X_M|}.$$

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 багатогранної поверхні, але не забезпечує гладкість (диференційованість), оскільки застосовувалася лінійна інтерполяція освітлення. Цей метод дає багато інших дефектів на зображуваній поверхні, однако найоптимальніший в обчислювальному плані, тому його використовують тільки в попередніх розрахунках. Наступний метод забезпечує гладкість зміни освітленості.

15.4.3. 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 потребує значно більше обчислень, оскільки інтерполюються три векторних компоненти, проте одержуються більш реалістичні графічні зображення.

Розглянемо, як можна одержати вектор нормалі, а отже, колір зафарбовування в довільній точці P грані ABC (рис. 15.9).

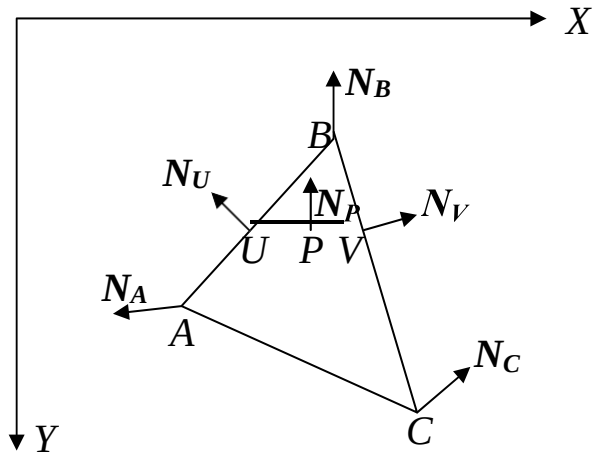


Рис. 15.9. Інтерполяція нормалей

Маючи нормалі прилягаючих граней, шукаємо нормалі в опорних точках (вершинах *A*, *B*, *C*). За допомогою лінійного інтерполювання векторів $\boldsymbol{N_U}$, $\boldsymbol{N_V}$ обчислюємо $\boldsymbol{N_P}$ (рис. 15.9), а для знаходження $\boldsymbol{N_U}$, $\boldsymbol{N_V}$ інтерполюємо вектор нормалі вздовж ребер граней, тобто

$$\boldsymbol{N_U} = (1 - t)\boldsymbol{N_A} + t\boldsymbol{N_B},\ t = \frac{|UA|}{|AB|},$$

$$\boldsymbol{N_V} = (1 - t)\boldsymbol{N_B} + t\boldsymbol{N_C},\ t = \frac{|BV|}{|BC|},$$

$$\boldsymbol{N_P} = (1 - s)\boldsymbol{N_U} + s\boldsymbol{N_V},\ s = \frac{|UP|}{|UV|}.$$

Але оскільки в моделях освітленості використовується одиничний вектор нормалі, то вектори $\boldsymbol{N_U}$, $\boldsymbol{N_V}$, $\boldsymbol{N_P}$ потрібно нормувати.

Зауваження 4. Змінюючи довільно вектор нормалі, можна моделювати поверхні з різними властивостями.

Як бачимо, перевагою моделей зафарбовування Гуро і Фонга є їхня 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 бібліотеки.

Поліпшувати якість зображення дозволяють досконаліші методи, наприклад методи трасування променів.

15.5. Методи трасування променів

Природно, що простіші локальні моделі освітлення мають певні обмеження, оскільки в цьому випадку кожна поверхня аналізується окремо, а отже, неможливо сформувати 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 здійснювати опис освітленості з урахуванням впливу всіх об’єктів тривимірної сцени, тобто потрібно враховувати не тільки прямі промені від джерела світла, але й другорядні (промені, що багаторазово 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 релізації цих алгоритмів для побудови якісного зображення досить складних тривимірних сцен. Але слід мати на увазі, що ці алгоритми вимагають великого обсягу обчислень, тим більше якщо формування

графічних зображень здійснюється в режимі реального часу. Тому використання таких моделей у реальному масштабі часу можливе лише при наявності потужних апаратних засобів.

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 у 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 бере участь у 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, заломлюючись, 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.15.10).

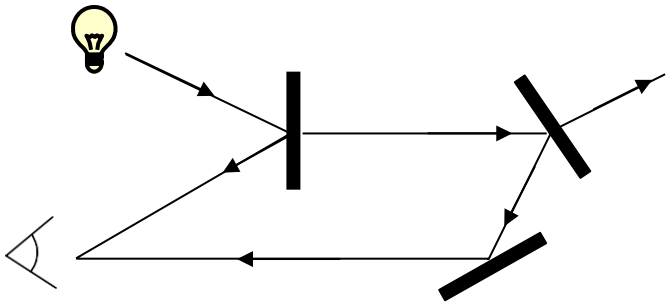


Рис. 15.10. Пряме трасування променів

У результаті численних проходів таких променів деякі з них попадуть в точку спостереження – око людини або оптичну систему камери – і сформують 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.

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 більш ефективний 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.

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 або 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, отримані 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 й заломлені промені.

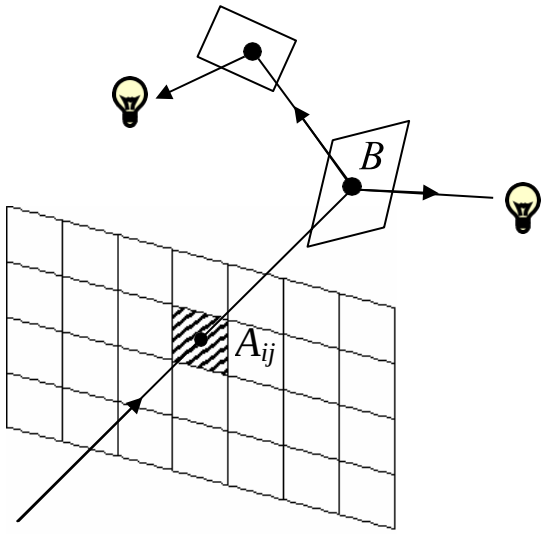


Рис. 15.11. Зворотне трасування променів

Якщо поверхня дзеркальна, то вторинний промінь будуємо 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. 15.11). 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 деревоподібна структура із сотень променів. Гілки такого дерева задають розповсюдження променів у сцені, а вузли – перетин із поверхнями в сцені.

Якщо джерело світла видно з точки перетину, то на початку необхідно обчислити „внесок” 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 тоді 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 збігаються, тобто при дифузному та 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 кольором джерел світла.

Алгоритм трасування рекурсивний, 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 сучасне програмне забезпечення, для того щоб одержати якісне фотореалістичне зображення за прийнятні витрати обчислювальних ресурсів, поєднує в собі кілька алгоритмів.

Значна частина обчислень при реалізації методу трасування променів припадає на визначення перетинів променів із поверхнями об’єктів сцени. Такий метод складно реалізувати для сцен зі 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 зумовлений 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.

Зазначимо, 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 запитання та завдання

1. Назвіть основні етапи 3D-конвеєра.
2. Розкрийте зміст поняття рендерінг/рендер.
3. Як обчислюється інтенсивність відбитого світла для дзеркальних та матових поверхонь?
4. Які моделі освітлення використовують у КГ?
5. Як обчислити координати вектора нормалі до параметрично заданої поверхні?
6. Як розрахувати тон зафарбовування точок поверхні?
7. Опишіть метод постійного зафарбовування. Вкажіть його недолік.

8. Які ефекти фотореалістичної візуалізації тривимірних моделей не можна одержати методом Фонга?
9. Порівняйте кількість обчислень у методах Фонга та Гуро.
10. Запишіть алгоритм методу зворотного трасування променів.
11. Що є критерієм зупинки методу зворотного трасування променів?
12. Чому метод зворотного трасування більш ефективний, ніж метод прямого трасування?
13. Для чого використовується метод оболонок?
14. Як знайти перетин променя з багатогранним об'єктом?
15. 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.
6. Записати базову операцію зворотного трасування променів у вигляді рекурсивної функції.
7. Розробити алгоритм знаходження точки перетину променя зі сферою/ еліпсоїдом.
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.

Розділ 16. Програмування графіки на OpenGL

16.1. Основні поняття

При створенні графічних додатків зручно користуватися спеціальними графічними бібліотеками. Існують десятки графічних бібліотек. Нині найбільш відомими на ринку вважаються два стандарти для роботи з двовимірною та тривимірною графікою: OpenGL (Open Graphics Library) – стандарт для всіх графічних станцій, а також DirectX із 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 із найпопулярніших серед прикладних програмних інтерфейсів API (Application Programming Interface) для розробки графічних додатків, тобто програмісти пишуть програми, які викликають графічні функції з бібліотек. OpenGL – універсальний інтуїтивно зручний потужний інструмент для програмування графіки. Цей стандарт використовується при створенні комп'ютерних ігор, САПР, візуалізації різних об'єктів, створення віртуальної реальності тощо. Він має інтуїтивно зрозумілий інтерфейс, що дозволяє розробляти програми з меншою кількістю рядків коду. Застосунки, які працюють з OpenGL, забезпечують однаковий візуальний результат для різних сучасних операційних систем (Windows, Linux, macOS).

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, яка для малювання 3D-вимірних зображень, що складаються з простіших примітивів, містить більше 300 графічних 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 об'єктів 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, серед яких – Digital, Hewlet Packard, IBM, Intel, Silicon, Microsoft. В основу стандарту покладена бібліотека IRIS GL, розроблена фірмою Silicon Graphics. Нині цей стандарт нараховує кілька – версій від 1.0 до 4.x. Для початкового вивчення програмування з застосуванням OpenGL можна використовувати версію 3.0.

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 повехні, 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; пересувати джерела світла і камеру за заданими траєкторіями;
6) додавати спеціальні ефекти туману, димки, 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 простоту* 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 графічними процесорами. Оскільки OpenGL відкритий стандарт, то виробники графічних прискорювачів самостійно можуть добавляти до OpenGL нові можливості.

Open GL дозволяє реалізувати сучасні графічні можливості відеокарт. 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 необхідно розраховувати проєкційну 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 стандарта і поставляється разом із 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 платформо-незалежна. Але можна задіяти функції конкретної операцій-

ної 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 й 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 точкою	float	GLfloat, GLclampf

D	64-бітне число з плаваючою точкою	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 void	GLuint, GLenum 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)* або 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 вправо, 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 треба певним способом виділити набір вершин, що визначають цей об'єкт. Для цього використовуються процедури *glBegin( )* і *glEnd( ).* Для комп'ютерної графіки на OpenGL ці дві команди найважливіші, одна з них повідомляє системі, що ми будемо рисувати примітив і який саме, а друга, що ми закінчили рисувати примітив. Між функціями *glBegin( )* і *glEnd( )*

можна використовувати тільки певний набір функцій OpenGL (інші функції будуть проігноровані). Серед цих функцій відзначимо *glVtrtex(), glColor(), glNormal()* та ін.

Кількість вершин, які можуть бути задані між *glBegin( )* і *glEnd( ),* необмежена.

Процедура *glBegin(GLenum mode);*. Параметр *GLenum mode* задає тип примітива (рис. 16.1), який може набувати таких символьних 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, тобто перший трикутник буде побудований на першій, другій, третій точках, а другий – на другій, третій, четвертій точках і т. д.;

GL_TRIANGLE_FAN – віяло трикутників (фен);

GL_QUARD_STRIP – чотирикутний стріп, в якому чотирикутник із номером n визначається вершинами з номерами $2n - 1$, $2n$, $2n + 2$, $2n + 1$.

Приклад 16.1. Побудувати трикутний стріп з двох трикутників.

```
glBegin(GL_TRIANGLE_STRIP);//трикутний стріп
    glColor3d(0,0,1);
    glVertex3d(1,2,3);
    glVertex3d(0,2,1.9);
    glVertex3d(-1,2,0);
    glVertex3d(0.1,2,1);
glEnd( );
```

Приклад 16.2.Процедура побудови кола

```
glBegin (GL_LINE_LOOP); //замкнена ламана
for (int i= 0; i<N; i++)
    {
    float angle=2*M_PI* i /N;
    glVertex2f (R*cos(angle), R*sin(angle));
    }
glEnd( );
```

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), координати текстури дорівнюють нулю.

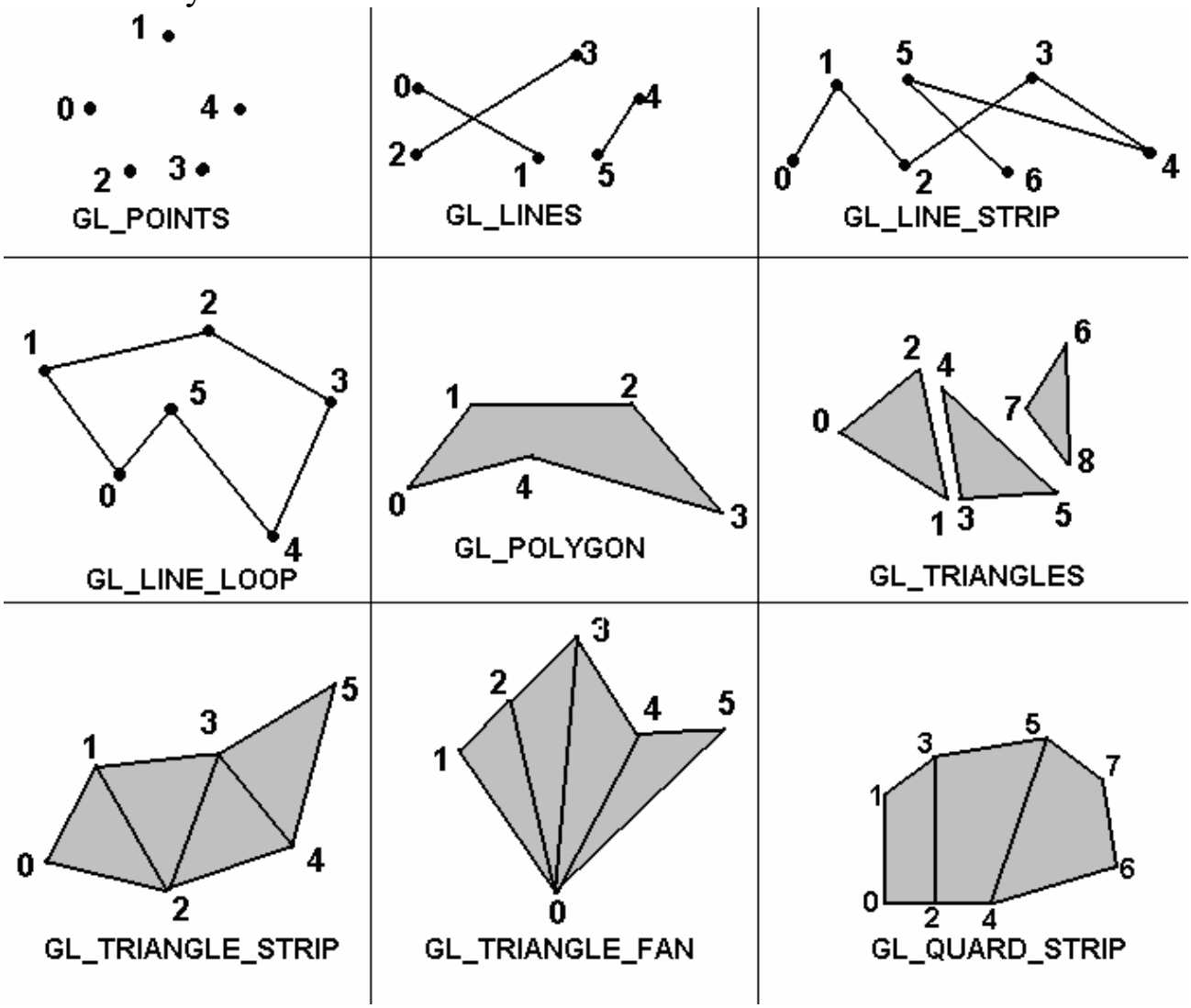


Рис. 16.1. Типи примітивів

Для задання поточного кольору вершини використовується команда *void glColor {3 4} {b s i f ub us ui} (Gltype 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 відрізку [0, 1], при цьому, за замовчуванням, значення alpha-компоненти дорівнює 1.0, що відповідає повній непрозорості. Якщо зазначений тип *ub* (*unsigned char*), то значеннння аргументів повинні лежати на відрізку [0, 255].

Якщо встановити поточний колір, то всі наступні об'єкти будуть зображатися цим самим 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 із 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();
```

Багатокутники можна зображати як заповнені області 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);.

Багатокутник має 2 сторони (грані) – передню (лицеву) і задню (нелицеву) і може бути зафарбований по-різному, в залежності від того, яка зі сторін багатокутника повернута до спостерігача. За замовчуванням, передньою вважається та сторона, вершини якої обходяться проти годинникової стрілки. Напрям обходу вершин лицьових сторін можна змінити викликом команди

void glFrontFace (Glenum mode);

зі значенням *GL_CW* параметра *mode*, а скасувати – з *GL_CCW*.

Щоб задати відображення багатокутників, використовується команда

void glPolygonMode (Glenum face, Glenum mode);.

Параметр *mode* визначає, як будуть відображатися багатокутники, при цьому параметр *mode* може мати 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.

Приклад 16.3. Побудувати заповнений полігон з межами.

```
glPolygonMode (GL_FRONT, GL_LINE);
glPolygonMode (GL_BACK, GL_FILL);
glBegin (GL_POLYGON);
glVertex3f(50.0, 50.0, 0.0);
glVertex3f(60.0, 20.0, 0.0);
glVertex3f(80.0, 30.0, 0.0);
glVertex3f(70.0, 80.0, 0.0);
glEnd();
```

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 граней командою

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 самим 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.*

Приклад 16.4. 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 (їх кількість необмежена), 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 зазнають певних перетворень. 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 тривимірних 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 або 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 кілька 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 й 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 *M* кожну вершину об’єкта й одержати змінені координати цієї вершини,

$$(x', y', z', 1)^T = M * (x , y , z , 1)^T,$$

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 16.5. Зсув прямокутника (рис. 16.2).

```
glMatrixMode(GL_MODELVIEW);
//попередньо модельна матриця одинична
glColor3f (0.0, 0.0, 1.0);
```

glRecti (100, 100, 200, 150);
//відображається правий синій прямокутник
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);.

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.

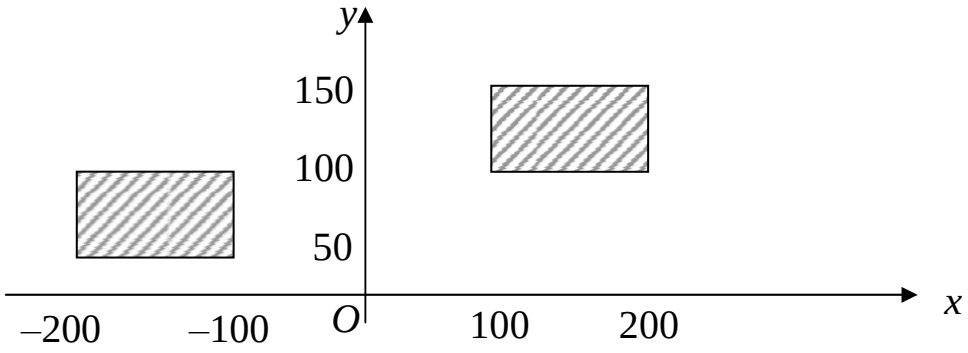


Рис. 16.2. Зсув прямокутника

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 проєктування визначає, як саме будуть проєктуватися 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 проєктування видимим об'ємом є прямокутний паралелепіпед, який обмежується межами відсікання. Для задання паралельного проєктування 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 проєктуванні є зрізаною пірамідою (рис. 16.3).

Для задання перспективного проєктування в OpenGL створено ряд процедур, наприклад

void gluPerspective (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);.

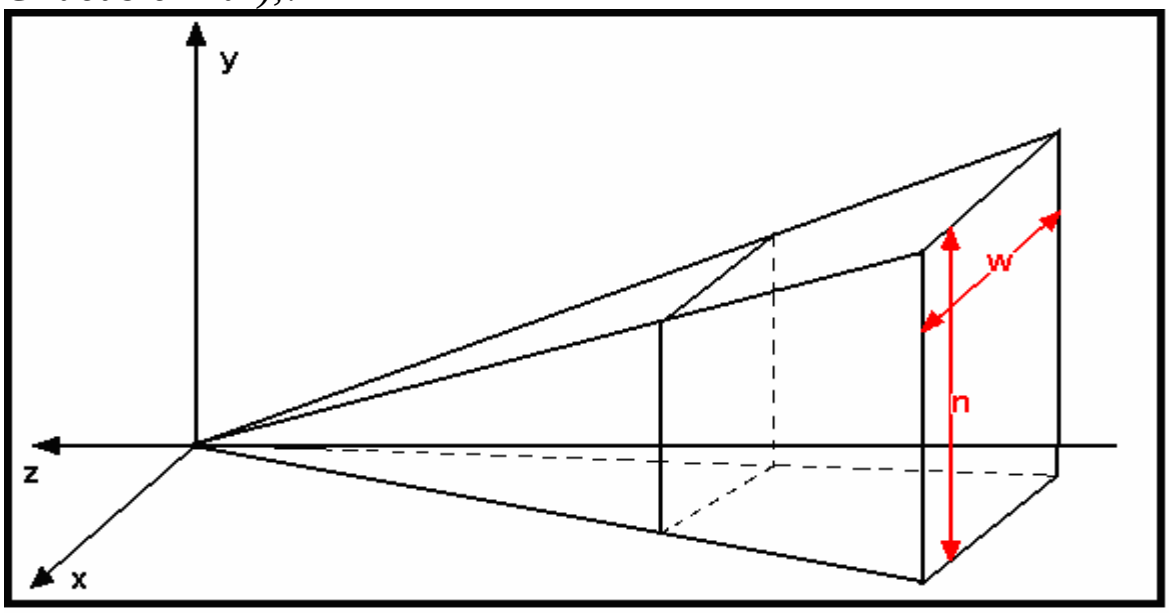


Рис. 16.3. Піраміда видимості при центральному проєктуванні

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 напряму 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 напряму 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 змоделювати камеру, що розташована в точці (*x, y, z*) і направлена на точку (*xp, yp, zp*), – це використати функцію *gluLookAt:*

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 освітлення.

Наприклад, гексаедр (куб) виводиться за допомогою функцій

glutWireCube(edgeLenght); // каркасний куб,

glutSolidCube(edgeLenght); // куб із зафарбованими гранями.

Параметру *edgeLengh* 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 сфе- ричної поверхні чотирикутною сіткою.

Для відображення поверхонь Glu другого порядку доступні три режими відображення. Якщо замість *GLU_LINE* використовувати символьну константу *GLU_POINT* як аргумент функції *gluQuadricDrawStyle*, то поверхня другого порядку відобразиться у вигляді точкового графіка. Якщо задати символьну 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 активізується 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. До цих ефектів належить 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. Крім 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 символьних констант (табл. 16.2).

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 знаходження розміщення джерела світла в OpenGL використовується символьна константа *GL_POSITION*. При цьому в масив *param* потрібно записати однорідні світові координати джерел світла у вигляді (*x*, *y*, *z*, *w*).

Таблиця 16.2

Значення	Значення за замовчуванням	Коментарі
GL_AMBIENT	(0, 0, 0, 1)	фонова RGBA-освітленість
GL_DIFFUSE	(1, 1, 1, 1)	дифузна RGBA-освітленість
GL_SPECULAR	(1, 1, 1, 1)	блікова (Фонга) RGBA-освітленість
GL_POSITION	(0, 0, 1, 0)	(*x*, *y*, *z*, *w*) – координати джерел світла
GL_SPOT_DIRECTION	(0, 0, –1)	(*x*, *y*, *z*) – напрям для конічних джерел світла
GL_SPOT_EXPONENT	(0)	показник степеня у формулі Фонга
…	…	…

Якщо в команді *GL_POSITION* параметр $w = 0$, то джерело світла направлене і світить у напрямку (*x*, *y*, *z*). Якщо $w \neq 0$, 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 увімкнутих джерел світла.

У наступному прикладі світло 1 іде від локального джерела, розміщеного в точці (2.0, 0.0, 3.0), а світло 2 – від направленого 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 установлюється 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 конічною 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 конічне світло так, що вісь конуса проходить уздовж додатного напрямку осі *x.* Кут конуса дорівнює 30^o 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 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 ***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. Напрям 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 символьних констант GL_FRONT, GL_BACKабо 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	Коментарі
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	фоновий колір матеріалу
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	дифузний колір матеріалу
GL_AMBIENT_ AND_DIFFUSE		фоновий і дифузний кольори матеріалу
GL_EMISSION	(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 символьну константу *GL_AMBIENT_AND_DIFFUSE*.

Метод візуалізації поверхні OpenGL задається функцією *glShadeModel (Rendering Metod);*.

Якщо параметру *Rendering Metod* 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 й різні способи накладання текстур. Для задання двовимірної текстури використовують функцію *glTexImage2D*.

Це лише короткий опис основних можливостей OpenGL. Більш ґрунтовні знання можна почерпнути з [36; 37; 39; 54].

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 з OpenGL у Windows використовується поняття контексту відтворення, який зв'язує OpenGL із 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);
BOOL wglMakeCurrent (HDC, HGLRChGLRC);.

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(HGLRChGLRC);.

Приклад програми з використанням функцій OpenGL для виконання операцій з кубом наведений в додатку.

Контрольні запитання та завдання

1. Поясніть різницю між кореневою бібліотекою OpenGL і бібліотеками Glu і Glut.
2. Як в OpenGL задати багатокутник списком його вершин/списком його ребер?
3. Яка команда в OpenGL використовується для того, щоб задати світло-сірий/чорний колір вікна зображення?
4. Як в OpenGL вказати 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 конічною 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 конічних джерел світла.
11. Як задати колір для дифузного і дзеркального освіт- лення?

Вправи і задачі для самостійного виконання

1. Написати код програми, яка будує 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 багатокутника червоним кольором, а його сторони – синім.
5. Написати фрагмент коду для присвоєння трьом вершинам трикутника різних кольорів. Сам трикутник зафарбувати за допомогою лінійної інтерполяції кольорів цих вершин.
6. Написати процедуру побудови прямокутного паралелепіпеда з ребрами паралельними координатним осям.
7. Використовуючи функції OpenGL, написати програму повороту багатокутника відносно деякої зовнішньої точки. Вхідними параметрами процедури є вершини багатокутника, координати центра повороту і кут повороту.
8. Реалізувати програму, в якій би тривимірні процедури геометричних перетворень OpenGL застосовувались до тетраедра і в результаті одержувалися масштабовані, повернуті та зсунуті фігури.
9. Написати процедуру побудови матриці складного двовимірного перетворення, об'єднавши три матриці базових перетворень. Після цього застосувати складне перетворення до трикутника так, щоб трикутник спочатку масштабувався відносно центра ваги, потім повернутий на кут α і зміщений уздовж вектора $(\Delta x, \Delta y)$.
10. Написати програму візуалізації сферичної поверхні з використанням моделі Фонга для освітлення.
11. Написати програму відображення даного текстурного узору на грань куба/тетраедра/на поверхню сфери.
12. Написати програму для відображення сцени, що містить сферу і тетраедр. Сцена освітлюється двома джерелами світла: одне – локальне червоне джерело, друге – віддалене біле джерело. Задати параметри поверхні для дифузного і дзеркального відображення. .
13. Створити проєкт у середовищі C++ для реалізації сфери, що обертається.
14. Написати програму відображення багатогранника з усуненням невидимих граней, використовуючи функції OpenGL. Кожна грань багатогранника повинна зафарбовуватися різними кольорами. Точка спостереження та інші параметри задаються як вхідні параметри. Модифікувати програму так, щоб багатогранник можна було спостерігати з будь-якої точки.
15. Ознайомитися з процедурами OpenGL для генерування кривих Безьє. Використовуючи ці процедури, побудувати різні криві Безьє за чотирма контрольними точками, розрахувавши 100 точок на кривій і з'єднавши їх відрізками.

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 і динамічних об'єктів, у 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 із його фокусів знаходиться на колі з центром у 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 всіх вершин правильного n-кутника між ($n \le 25$). Колір діагоналей при цьому змінюється у випадковий спосіб.
7. Побудувати художній візерунок, якщо в куті він будується так: 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, кожні 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 $j - i$ є 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, що лежить на колі, яке котиться по прямій (циклоїда). Розглянути випадки, коли точка P лежить у 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 у 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, …, 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 мінімальна. Узагальнити задачу на випадок площини.
21. Побудувати діаграму Вороного на множині довільно розташованих на площині опорних точок. *Вказівка*: діаграма Вороного – це об'єднана межа багатокутників Вороного, створених для кожної з

опорних точок. Багатокутником Вороного опорної точки A називається множина всіх точок площини, відстані від яких до точки A менші, ніж до іншої опорної точки.

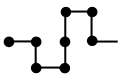
22. На площині задано n-кутник (опуклий або неопуклий). Зобразити розбиття полігона на трикутники.
23. Задано дві півплощини. Побудувати їхній перетин. За півплощину вибираємо ту, яка залишається справа від прямої, якщо рухатися вздовж цієї прямої.
24. На площині задано n точок. Зобразити опуклу оболонку цих точок і 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 $P_1, P_2, \ldots, P_n$.
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
$$\begin{cases} x = (2 + 7\cos(\sin\theta + \sin 121\theta))\cos\theta, \\ y = (2 + 7\cos(\sin\theta + \sin 121\theta))\sin\theta, \end{cases} \quad \frac{\pi}{2800} \le \theta \le 2\pi,$$
вибравши однаковий масштаб по Ox і Oy. Передбачити виведення координати точки і параметра θ при натисканні правої клавіші миші.
30. Побудувати спіраль Архімеда. Рівняння гвинтової лінії при цьому має вигляд $x(t) = (r_0 + r_1 t)\cos(t)$, $y(t) = (r_0 + r_1 t)\sin(t)$, $z = ht$.
31. Побудувати графік торової спіральної кривої. Параметричне рівняння торової спіральної кривої має вигляд
$x(t) = (r\cos(wt) + R)\cos(t)$, $y(t) = r\sin(wt)$, $z = (r\cos(wt) + R)\sin(t)$.
32. Написати процедуру знаходження перетину двох довільних трикутників, заданих на площині.
33. Дано два трикутники, які не перетинаються. В кожному з цих трикутників вибирається по одній точці. Перший трикутник починає рухатися до другого паралельно вектору, що з'єднує ці точки.

Зупинити рух трикутника в момент його дотику до другого трикутника. Узагальнити задачу для довільного полігона.

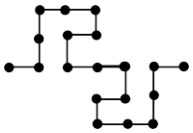
34. Задано квадрат і пряму, які не мають спільних точок. Через центр квадрата проведено перпендикуляр до прямої. Квадрат, обертаючись навколо свого центру проти годинникової стрілки, рухається до прямої так, що центр квадрата залишається на перпендикулярі. Зупинити рух квадрата в момент його дотику з прямою.
35. Побудувати зображення платонових тіл: 1) гексаедра (куба), 2) тетраедра, 3) октаедра, 4) додекаедра, 5) ікосаедра.
36. Розробити тест перетину прямої лінії $Ax + By + C = 0$ з полігоном $P = \{P_1, P_2, \ldots, P_n, P_1\}$.
37. Задано трикутник, одна з вершин якого рухається по інтерактивно введеній кривій (у простішому випадку – по прямій). Побудувати анімацію кіл: *а*) вписаних у рухомий трикутник; *б*) описаних навколо рухомого трикутника.
38. Розробити анімаційну програму, яка малює динамічні кольорові візерунки за допомогою радіальних багатоколірних ліній.
39. Розробити анімаційну програму, яка виконує кольорову анімацію набору різнокольорових концентричних кіл так, щоб за рахунок зміни кольорів і плавної зміни розмірів кіл створити ілюзію руху.
40. Розробити анімаційну програму, яка реалізовує виведення символів у задану позицію екрана та анімацію цього тексту (динамічна зміна його положення, орієнтації в просторі, розмірів).

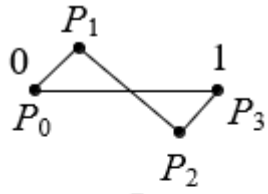
Лабораторна робота № 2. Фрактали

Побудувати нижчеописані 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 характерне задання основи та генератора, який повторюється при зменшенні масштабу. Динамічні фрактали отримуються за допомогою рекурентних формул.

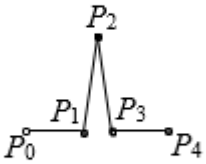
1. Фрактал Коха. Основа – відрізок. Генератор  . Усі ланки генератора мають однакову довжину. Реалізувати інкрементний метод.
2. Фрактал Мінковського. Основа – відрізок. Генератор  . Усі ланки ламаної мають однакову довжину.
   Узагальнити задачу, взявши за основу квадрат.
3. Фрактал Леві. Основа – одиничний відрізок. Генератор (ламана з

проміжною точкою (0,5; 0,5)), тобто відрізок-основу замінюємо половиною квадрата.

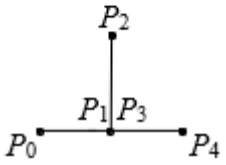
4. Основа – відрізок. Генератор  . Всі ланки ламаної мають однакову довжину.
5. Острів Коха. Основа – правильний трикутник. Генератор Коха (задача 1) орієнтований усередину/назовні трикутника.
6. Острів Мінковського. Основа – одиничний квадрат. Генератор – ламана з проміжними точками:
   а) $P_1(0{,}25;\ 0{,}25)$, $P_2(0{,}75;\ -0{,}25)$;
   б) $P_1(0{,}3;\ 0{,}3)$, $P_2(0{,}7;\ -0{,}3)$.



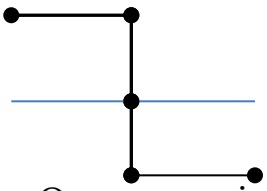
7. Різаний квадрат. Основа – одиничний квадрат. Генератор – ламана з проміжними точками $P_1(0{,}47;\ 0)$, $P_2(0{,}5;\ 0{,}47)$, $P_3(0{,}53;\ 0)$.



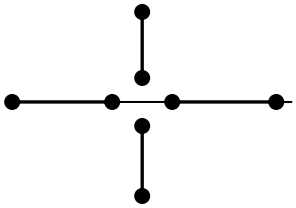
8. Льодовий квадрат. Основа – одиничний квадрат. Генератор – ламана з проміжними точками $P_1(0{,}5;\ 0)$, $P_2(0{,}5;\ 0{,}33)$, $P_3(0{,}5;\ 0)$.



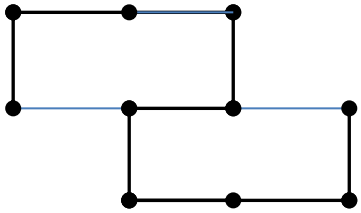
9. Льодовий трикутник 1. Основа – правильний трикутник. Генератор – ламана із задачі 8.
10. Льодовий трикутник 2. Основа – рівносторонній трикутник із вершинами (0, 0), (0,5; 0,85), (0, 1). Генератор – ламана з проміжними точками $P_1(0{,}5;\ 0)$, $P_2(0{,}375;\ 0{,}2165)$, $P_3(0{,}5;\ 0)$.
11. Побудувати фрактал. Основа – квадрат. Генератор складається з 4 ланок. Коефіцієнт зменшення $r = 1/2$.



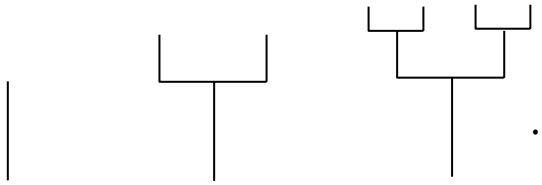
12. Побудувати фрактал із незв’язаним шаблоном. Основа – відрізок. Коефіцієнт зменшення $r = 0{,}45$. Генератор складається з 4 ланок.



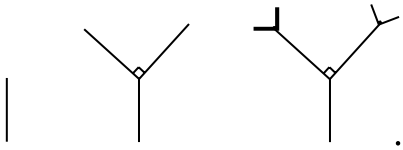
13. Побудувати фрактал. Основа квадрат. Генератор – ламана з довжиною ланки 1/3.

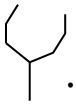


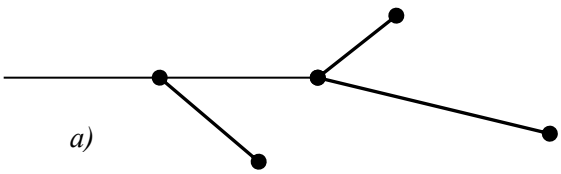
14. H-фрактал. Основа – горизонтальний відрізок. Через кінці горизонтального відрізка перпендикулярно до нього проводяться два коротших відрізки. На кінцях цих вертикальних відрізків будуються горизонтальні відрізки у зменшеному масштабі і т. д. Коефіцієнт зменшення дорівнює $\frac{1}{3}$.
15. У колі 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.
16. Дерево Мандельброта. Основа – відрізок. Генератор  . Кожний із відрізків замінюється генератором у зменшеному масштабі. Коефіцієнт зменшення $\mu \in (0, 1)$.
17. Двійкове дерево. Основа – відрізок (стовбур). Генератор  . Перші наближення мають вигляд:



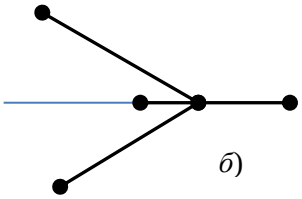
18. Оголене дерево Піфагора. Основа – відрізок (стовбур). Генератор –  . Перші наближення мають вигляд:



19. Реалістичне дерево. Основа – відрізок. Генератор –  .
20. Кущова структура. Основа – відрізок. Генератор –  .
21. Побудувати фрактал дерево. Основа – відрізок (стовбур). Генератор (3 відрізки).

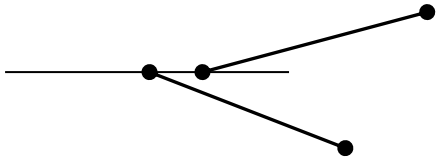


а)

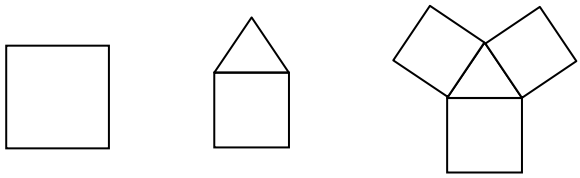


б)

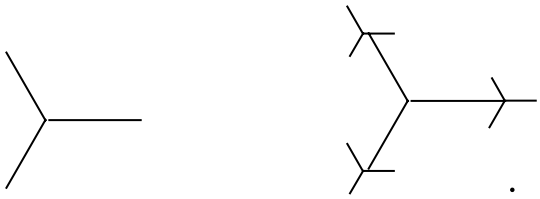
22. Побудувати фрактал–дерево. Основа – відрізок (стовбур). Генератор (2 відрізки).



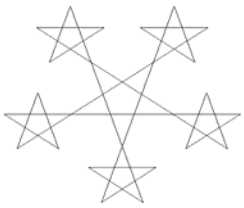
23. Дерево Піфагора. Основа – квадрат. Генератор – правильний трикутник. Перші наближення мають вигляд:



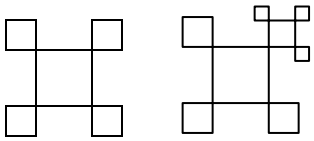
24. Трійкове дерево (дендрит). З однієї точки під кутом 120^0 виходять три відрізки. Кожен із кінців цих відрізків є точкою, звідки також виходять три менших відрізки, і т. д. Перші наближення:



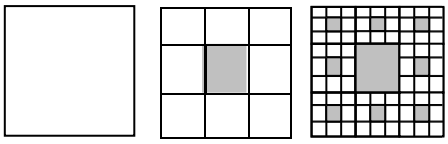
25. Зірковий фрактал. У вершинах правильної п’ятикутної зірки будується 5 менших зірок, далі на вільних кінцях цих зірок будуються ще менші зірки і т. д. Взяти коефіцієнт зменшення $r = 0{,}35$, а кількість кроків рекурсії $n = 5$.



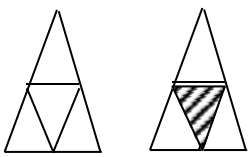
26. Зірковий квадратичний фрактал. У вершинах квадрата будуються 4 менших квадрати, на трьох вільних вершинах цих квадратів будуються ще менші квадрати і т. д.



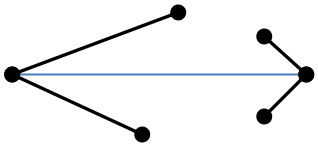
27. Квадрат Серпінського. Заданий квадрат ділиться двома горизонтальними і двома вертикальними лініями на 9 квадратів. Одержаний центральний квадрат зафарбовуємо, а решту вісім квадратів перетворюємо як перший і т. д.



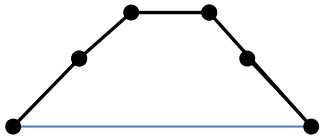
28. Трикутник Серпінського. У рівнобедреному трикутнику проводимо середні лінії. Трикутник, що утворений середніми лініями, зафарбовуємо, а інші трикутники перетворюємо аналогічно до першого і т. д.



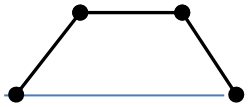
29. Гілка папороті з параметрами: $\alpha = 2^0$, $\beta = 86^0$, $k = 0.14$, $k_1 = 0{,}3$.
30. Побудувати дендрит. Основа – відрізок стовбур. Генератор – чотири відрізки.



31. Побудувати фрактал. Генератор взяти з рис. 10.6. Основа – квадрат. Глибина рекурсії *num* = 3.
32. Побудувати фрактал. Основа правильний п'ятикутник. Генератор із довжиною ланки ламаної r =1/3 має вигляд

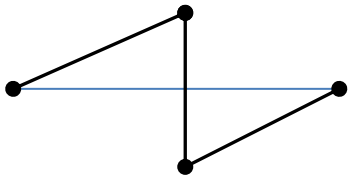


33. Побудувати фрактал. Основа правильний шестикутник. Генератор



Довжина ланки ламаної становить r = 1/2.

34. Побудувати фрактал – острів Госпера. Основа – правильний трикутник. Генератор



Довжина ланки ламаної становить $r = \frac{1}{\sqrt{3}}$.

35. Фрактал Жуліа. Множина Жуліа задається рекурентною формулою

$z_{n+1} = z_n^2 + c$, $n = 0,1,2,\ldots$, $z = x + iy$, $c = a + ib$, $i = \sqrt{-1}$, або

$x_{n+1} = x_n^2 - y_n^2 + a$, $y_{n+1} = 2x_n y_n + b$, $n = 0,1,2,\ldots$

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 1) $a = -0,22$, $b = -0,74$; 2) $a = 0,11$, $b = 0,66$; 3) $a = 0$, $b = 1$; 4) $a = 0,8$, $b = 0,08$.

36. Фрактал Мандельброта. Множина Мандельброта задається рекурентною формулою

$z_{n+1} = z_n^2 + c$, $n = 0,1,2,\ldots$, $z = x + iy$, $c = a + ib$, $i = \sqrt{-1}$,

де z_0 – фіксована точка, а $c \in C$. За вихідні параметри взяти $z_0 = 0$, $a \in [-2,5; 2,5]$, $b \in [-2, 2]$. Передбачити виведення на екран будь-якої прямокутної області цього фрактального зображення.

37. Фрактал, визначається рекурентною формулою

$z_{n+1} = \lambda z_n (1 - z_n)$, $n = 0,1,2,\ldots$, $z = x + iy$.

Для z_0 вибрати область [–2,2; 1,2] × [–1,2; 1,2]. Критерій завершення ітераційного процесу: $x_k^2 + y_k^2 > 5$.

38. Фрактал Ньютона. Фрактал Ньютона задається формулою

$z_{n+1} = z_n - \dfrac{f(z_n)}{f'(z_n)}$. Якщо $f(z) = z^k - a$, тоді

$$z_{n+1} = \frac{(k-1)z_n^k + a}{kz_n^k}, \quad n = 0,1,2,...$$

Розглянути варіанти: 1) $k = 3$, $a = 1$; 2) $k = 5$, $a = 1$.

39. 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 і b так, щоб одержати ефектні кольорові фрактальні зображення. Колір визначається кількістю ітерацій. Критерій завершення ітераційного процесу: $x_k^2 + y_k^2 > 10$.

40. Побудувати фрактал, що дає зображення кленового листа. Необхідні дані взяти з розділу 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 вигляді.
3. Адаптувати алгоритм Брезенхема для відрізка прямої лінії із заданою товщиною лінії. *Вказівка*: для побудови широкої лінії можна перерисовувати зміщений квадрат відповідної орієнтації.
4. Реалізувати програму для побудови прямокутника із 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 із 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 і 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 ТСВ-сплайнів.
21. Написати програму, яка виводить на екран просторову криву Безьє для набору з чотирьох контрольних точок. Для зображення кривої використати ортогональну проєкцію.
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 прямокутником.
27. Реалізувати алгоритм Вейлера–Азертона 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 полігона.

33. Розробити програму, яка для напівтонового растрового зображення дозволить поліпшити 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 $\alpha \neq 90^{o}$.
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 О*x*, яка пересувається фіксованою траєкторією.

6. Побудувати триметричну проєкцію зрізаного куба для $\psi = 15^0, 30^0, 45^0$, $\varphi = 0^0, 15^0, 30^0, 45^0, 60^0, 75^0, 90^0$.
7. Побудувати диметричну проєкцію зрізаного куба, взявши для кутів φ, ψ такі значення: $\psi = \pm 60$, $\varphi = \pm arctg\left(\frac{\sqrt{3}}{2}\right) = \pm 40.9^0$.
8. Побудувати одноточкову та двоточкову центральну проєкцію двох різноорієнтованих кубів в одній сцені.
9. Побудувати аксонометричну проєкцію куба з вписаними в його грані колами при співвідношенні масштабів $k_x : k_y : k_z = 6 : 5 : 4$.
10. Побудувати чотири можливі ізометричні проєкції зрізаного куба. *Вказівка*: ізометрична проєкція одержується при кутах $\psi = \pm 45^0$, $\varphi = \pm \arccos\left(\frac{2}{3}\right) = \pm 35.26^0$.
11. Побудувати стандартну диметричну проєкцію зображення глобуса радіуса r, що містить паралелі з кроком 30^0, меридіани з кроком 45^0 і контур глобуса. Реалізувати нахил зображення вправо відносно осі z на кут $\varphi = 23{,}5^0$. *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 проєкцію Кавальє зрізаного куба з кутом $\beta = 45^0$, а кут змінюється від -30^0 до 30^0 з кроком 15^0.
13. Побудувати кабінетну косокутну проєкцію для зрізаного куба з кутом, який змінюється від -30^0 до 30^0 з кроком 15^0.
14. Підібравши кути α та β, побудувати косокутну проєкцію сцени, яка складається з модельного куба з вписаними в його грані кола- ми, якщо кола, що лежать у площині, яка паралельна до картинної площини, проєктуються в кола, а кола з непаралельних граней до картинної площини – в еліпси з відповідною орієнтацією півосей.
15. Написати програму обертання двох платонових тіл навколо дея- кої осі з усуненням невидимих ребер. У роботі використати одно- точкову перспективну проєкцію та алгоритм художника для усу- нення невидимих граней.
16. Побудувати одноточкову проєкцію куба на площину $z = 0$, попе- редньо змістивши його вздовж прямої $y = x$.
17. Побудувати в одній сцені двоточкову проєкцію куба, тетраедра та октаедра.

18. Розробити програму усунення невидимих граней для центральної проєкції октаедра. Видимі грані зафарбовувати зеленим кольором.
19. Написати програму зображення перетину двох трикутників у 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 текстурного зразка на грані куба.
22. Методом поточного горизонту побудувати такі поверхні:

1) $y = f(x,z) = e^{-a\left(x^2+z^2\right)}\cos(\omega_x x)\cos(\omega_x z)$, де $a = 0{,}02$, $\omega_x = 1$, $\omega_z = 0{,}5$;

2) $z = f(x,z) = \dfrac{\cos\left(b\sqrt{x^2+y^2}\right)}{1+c\sqrt{x^2+y^2}}$ (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.
26. 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.
27. Розробити програму, яка усуває невидимі грані октаедра на основі методу видалення нелицьових граней багатогранників. Передбачити обертання цього октаедра навколо однієї з координатних осей. *Вказівка*: для побудови проєкції об'єкта необхідно однорідні координати об'єкта помножити на відповідну матрицю проектування. Для моделювання процесів обертання, перенесення та масштабування використати відповідні матриці цих перетворень.
28. Написати програму анімації опуклого багатогранника відносно осі, що проходить через об'єкт паралельно до картинної площини. Припускається, що об'єкт цілком лежить перед картинною площиною. Для послідовного відображення проєкцій використати ортографічну проєкцію.
29. Реалізувати алгоритм зафарбовування трикутної грані методами *а)* Гуро, *б)* Фонга, вважаючи, що джерело світла спрямоване в центр трикутника.

Комп'ютерні проєкти

1. Написати програму апроксимації сфери методом рекурсивного розбиття тетраедра в паралельній проєкції. 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 проєкцію.
3. У будь-якій системі тривимірного геометричного моделювання, наприклад 3DStudio MAX або Maya, створити сцену з площини, на якій розміщені примітивні тіла (паралелепіпеди, сфери, конуси, циліндри, еліпсоїди тощо), задати три різні джерела світла і розмістити їх так, щоб у сцені були наявні тіні та взаємні відображення світла від створених об'єктів. Виконати накладання різних матеріалів на поверхні тіл, 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 (TCB-сплайни).
5. Написати програму моделювання криволінійних поверхонь за допомогою елементарних фрагментів поверхонь Безьє, бікубічних B-сплайнових поверхонь і NURBS-поверхонь. Функція двох змінних задається своїми значеннями у вузлах прямокутної сітки. Значення функції у вузлах вводити за допомогою клавіатури.
6. Розробити проєкт для моделювання платонових тіл. Передбачити інтерактивний вибір їх типу, розміру і місцерозміщення, способу зображення, можливості задавати вектор обертання та кутову швидкість тощо. В проєкті використати одноточкове центральне проектування та передбачити вибір видової системи координат.
7. Створити проєкт малювання поверхонь другого порядку. Передбачити задання обмежуючого паралелепіпеда для перегляду нескінченних поверхонь, його переміщення, зміну розмірів та обертання. Поверхні вибирати з меню. Для зображення поверхонь використати ортогональні проєкції та апроксимацію лініями. Розробити

проєкт для малювання конструктивних, динамічних і статистичних фракталів. Для динамічних фракталів передбачити вибір та масштабування прямокутної області фрактала.

8. 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 із 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 проєкцію та алгоритм сортування граней за глибиною для усунення невидимих граней. Координати спостерігача задавати інтерактивно.
9. 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 проєкції з усуненням невидимих ліній. Координати спостерігача задавати інтерактивно.
10. Розробити програму візуалізації 3D-полігональної моделі з нанесенням текстури. За текстуру вибрати растровий образ, відображений на грані об’єкта. Проєкція – паралельна. Світло – паралельний пучок. Модель освітлення – з дифузними і дзеркальними складовими.
11. Розробити проєкт відсікання багатокутників прямокутним та багатокутним вікном. Передбачити інтерактивний вибір методу відсікання й інтерактивне задання багатокутника, який відсікається.
12. Розробити проєкт цифрової обробки зображень. Програма повинна завантажити і відобразити BMP-файл у форматі RGB.
13. 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 цією палітрою. Реалізувати зведення зображення до чорно-білого.
14. Розробити проєкт „Типи проєкцій”. Передбачити вибір об’єктів для побудови проєкцій, способу проєктування та параметрів проєкцій.
15. Розробити анімаційний проєкт 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 : пер. с англ. Київ : ЕвроСИБ, 2002. 512 с.
2. Алексюк А.А. Конструирование кинематических моделей линий и поверхностей в компьютерной графике. *Вестник Евразийской науки*. 2019. №6. URL : https://esj.today/PDF/38ITVN619.pdf
3. Аммерал Л. Машинная графика : В 4 кн. : пер. с англ. Москва : СолСистем, 1992.
4. Анісімов В.А., Терещенко В.М., Кравченко І.В. Основні алгоритми обчислювальної геометрії : навч. посібник. Київ : Київський ун-т, 2002. 82 с. URL : http://cg.unicyb.kiev.ua/
5. Блінова Т.О., Порєв В.М. Комп'ютерна графіка. Київ : Юніор, 2004. 456 с.
6. Василюк А., Мельникова М. Комп'ютерна графіка : підручник. Львів : Львівска політехніка, 2017. 308 с. URL : http://pdf.lib.vntu.edu.ua/books/IRVC/Skorukova_2020_93.pdf
7. Веселовська Г.В., Ходаков В.Є., Веселовський В.М. Комп'ютерна графіка : навч. посібник. Херсон : ОЛДІ-плюс, 2004. 584 с. URL : http://pdf.lib.vntu.edu.ua/books/2020/Veselovska_2008_584.pdf .
8. Власій О.О., Дудка О.М. Комп'ютерна графіка. Обробка растрових зображень : Навчально-методичний посібник. Івано-Франківськ : ДВНЗ «Прикарпатський національний університет імені Василя Стефаника», 2015. 72 с. URL : http://lib.pnu.edu.ua:8080/bitstream/123456789/2518/1/Vlasii_Dudka_Graph.pdf .
9. Гилой В. Интерактивная машинная графика : пер. с англ. Москва : Мир, 1981. 380 с.
10. Горобець С.М. Основи комп'ютерної графіки: навч. посібник. Київ : Центр навчальної літератури, 2006. 232 с.
11. Довгий Б.П., Ловейкін А.В., Вакал Є.С., Вакал Ю.Є. Сплайн-функції та їх застосування. Київ : Видавничо-поліграфічний центр "Київський університет", 2016. 117 с. URL : http://www.matfiz.univ.kiev.ua/uploads/books/spline_ml.pdf
12. Довгий Б.П., Ловейкін А.В., Вакал Є.С., Вакал Ю.Є., Попов А.В. Використання пакета MathCad для розв'язування прикладних задач. Київ : Фітосоціоцентр, 2012. 78 с.
13. Журавчак Л.М., Левченко О.М. Програмування комп'ютерної графіки та мультимедійні засоби : навч. посібник. Львів : Вид-во Львівської політехніки, 2019. 276 с. URL : http://pdf.lib.vntu.edu.ua/books/2020/Zhuravchak_2019_276.pdf

14. Карпенко Н.Ю., Корзун Н.К. Комп'ютерна графіка : конспект лекцій. Харків : ХНУМГ, 2020. 48 с. URL : https://core.ac.uk/download/pdf/356030108.pdf
15. Комп'ютерна графіка : метод. рекомендації та лабораторні завдання / укл. Маценко В.Г., Петришин Я.Р. Чернівці : Рута, 2006. 24 с.
16. Комп'ютерна графіка : навч. посібник, ч.1. / укл. Тотосько О.В., Микитишин А.Г., Стухляк П.Д. Тернопіль : ТНТУ, 2017. 304 с. URL : https://elartu.tntu.edu.ua/bitstream/lib/22337/1/Komp_graf_knyga_1.pdf
17. Коссак О, Мітрулі М., Челакас М. Комп'ютерна графіка: навч. посібник. Львів : Львів. нац. ун-т ім. І. Франка, 2010. 205 с.
18. Мандельброт Д. Фракталы и хаос. Множество Мандельброта и другие чудеса : пер. с англ. Ижевск : НИЦ «Регулярная и хаотическая динамика», 2009. 392 с.
19. Маценко В.Г. Комп'ютерна графіка : навч. посібник. Чернівці : Чернівец. нац. ун-т, 2009. 343 с. URL : https://mmi.stu.cn.ua/wp-content/uploads/2016/09/MatsenkoKompGrafyka.pdf
20. Новожилова М.В., Мироненко В.В. Комп'ютерна графіка : навч.-метод. посібник. Харків : ХНУБтаА, 2015. 71 с. URL : https://kn-it.info/wp-content/uploads/2020/10/Kompyuterna-grafika-2-ch.-KN-11.pdf
21. Ньюмен У., Спрул Р. Основы интерактивной машинной графики : пер. с англ. Москва : Мир, 1976. 574 с.
22. Обчислювальна геометрія в задачах комп'ютерної графіки та комп'ютерного зору : конспект лекцій / укл. Дашкевич А.О. Харків : НТУ "ХПІ", 2018. 46 с. URL : http://repository.kpi.kharkov.ua/bitstream/KhPI-Press/40741/1/prohramy_2018_Obchysliuvalna_heometr_konsp_lek.pdf
23. Основи комп'ютерної графіки : метод. вказівки. Харків : НТУ ХПІ, 2020. 51 с. URL : http://repository.kpi.kharkov.ua/bitstream/KhPI-Press/51033/1/prohramy_2020_Osnovy_kompiuternoi_hrafiky.pdf
24. Павлидис Т. Алгоритмы машинной графики и обработка изображений : пер. с англ. Москва : Радио и связь, 1986. 400 с.
25. Пічугін М. Комп'ютерна графіка : навч. посібник. Київ : Центр учбової літератури, 2013. 346 с.
26. Пратт М. Вычислительная геометрия. Применение в проектировании и в производстве : пер. с англ. Москва : Мир, 1982. 304 с.
27. Різник О.Я. Основи комп'ютерної графіки : курс лекцій. Львів : Видавництво Львівської політехніки, 2012. 220 с.
28. Роджерс Д. Алгоритмические основы машинной графики : пер. с англ. Москва : Мир, 1989. 512 с.
29. Роджерс Д., Адамс Дж. Математические основы машинной графики : пер. с англ. Москва : Мир, 2001. 604 с.

30. Романюк О.Н. Комп'ютерна графіка : навч. посібник. Вінниця : ВДТУ, 2001. 130 с.
31. Сирота С.В., Ліскін В.О. Обчислювальна геометрія та комп'ютерна графіка. Київ : Просвіта, 2015. 36 с. URL : https://ela.kpi.ua/handle/123456789/38362
32. Тмєнова Н.П. Комп'ютерна графіка : навч.-метод. посібник. Київ : ВПЦ "Київський університет", 2017. 111 с. URL : http://pdf.lib.vntu.edu.ua/books/2020/Tmenova_2017_111.pdf
33. Фокс А. Вычислительная геометрия. Применение в проектировании и на производстве : пер. с англ. Москва : Мир, 1982. 304 с.
34. Фоли Дж., ван Дэм А. Основы интерактивной машинной графики : в 2 кн. : пер. с англ. Москва : Мир, 1987.
35. Хатунцев А.Ю. Обчислювальна геометрія та комп'ютерна графіка : навч. посібник. Суми : Сум ДУ, 2009. 137 с. URL : http://essuir.sumdu.edu.ua/handle/123456789/1754
36. Херн Д., Бейкер М. Компьютерная графика и стандарт OpenGL : пер. с англ. Москва : Вильямс, 2005. 1158 с.
37. Хилл Ф. OpenGL. Программирование компьютерной графики : пер. с англ. СПб. : Питер, 2002. 1088 с.
38. Эгрон Д. Синтез изображений. Базовые алгоритмы : пер. с англ. Москва : Радио и связь, 1993. 216 с.
39. Эйнджел Э. Интерактивная компьютерная графика. Вводний курс на базе OpenGL : пер. с англ. Москва : Вильямс, 2001. 592 с.
40. Юань Фень. Программирование графики для Windows : пер. с англ. СПб. : Питер, 2002. 1072 с.
41. Agoston M.K. Computer Graphics and Geometry Modelling Springer, 2005. 407 p.
42. Berg M., Cheong O., Kreveld M. Computational Geometry. Algorithms and Applications. Berlin; Heidelberg : Springer-Verlag, 2008. 386 p. ; Берг М., Чеонг О., Кревельд М., Овермарс М. Вычислительная геометрия. Алгоритмы и решения. СПб. : ДМК Пресс, 2016. 438 с.
43. Laszlo V.J. Computational geometry and computer graphics in C++. Prentice Hall, 1995. 300 p. ; Ласло М. Вычислительная геометрия и компьютерная графика на С++. Москва : Бином, 1997. 304 с.
44. Falconer K. Fractal Geometry : mathematical foundation and application. England : Wiley, 2014. 368 p. URL : books.google.com
45. Mandelbrot B.B. The Fractal Geometry of nature Echo Points Books, 2021. 480 p. ; Мандельброт Д. Фрактальна геометрия природы. Ижевск : РХД, 2002. 480 с.
46. Prepata F., Schamos. M. Computational Geometry. Springer, 1985. 390 p. Препата Ф., Шеймос М. Вычислительная геометрия. Введение. Москва : Мир, 1989. 478 с.

Інтернет-ресурси

47. http://cg.unicyb.kiev.ua – сайт з комп'ютерної графіки Київського національного університету імені Тараса Шевченка.
48. http://graphics.cs.ucdavis.edu – сайт з комп'ютерної графіки інституту аналізу даних і візуалізації Каліфорнійського університету.
49. http://www.cg.tuwien.ac.at/courses/cg2 – сайт Інституту комп'ютерної графіки і алгоритмів Віденського технічного університету.
50. https://psi-technology.net/videos/3d-fraktaly/ – подорож по 3D-фракталах.
51. https://journals.carleton.ca/jocg/index.php/jocg – сайт журналу "Journal of Computational Geometry".
52. Gortler S.J. Foundation of 3D Computer Graphics – books.google.com
53. https://sites.google.com/site/virtualnaaitskolaucitela/grafika-v-programme-paintnet – графіка в програмі Paint.NET.
54. https://paintnet.org.ua/ – український сайт редактора Paint.NET.
55. https://www.glfw.org (OpenGL Extension Wrangler) – багатоплатформна бібліотека з відкритим вихідним кодом для програмування комп'ютерної графіки.
56. http://ooep.kpi.ua/downloads/disc/inf_t/posibn_Krav_Myk.pdf – навчальний посібник "Інформаційні технології : системи комп'ютерної математики" (електронний ресурс, автор Кравченко І.В., Микитенко В.І.).
57. http://sambir.wunu.edu.ua/my_downloads/learning/ek/3kurs/SI/MathCAD.pdf – конспект лекцій з MATHCAD.
58. David M. Mount. Lecture notes for the course CMSC 754 Computational Geometry (https://www.cs.umd.edu/class/fall2014/cmsc754/Lects/cmsc754-fall14-lects.pdf).
59. http://www.nbuv.gov.ua/ – національна бібліотека ім. В.І. Вернадського.
60. http://euro.ecom.cmu.edu/people/faculty/mshamos/1975ClosestPoint.pdf – опис алгоритму побудови діаграми Вороного з доведеннями.
61. https://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/Voronoi/DivConqVor/divConqVor.htm – алгоритм та графічна ілюстрація покрокової побудови діаграми Вороного.

ДОДАТОК

Код програм зафарбовування областей (С++)

```
#include <iostream>
#include <Windows.h>      //підключення бібліотеки для графічних операцій
#include <conio.h>        //підключення бібліотеки для затримки екрана
#include <stdlib.h>       //підключення бібліотеки для можливості очищення екрана
HWND hwnd = GetConsoleWindow();                    //привязка до консолі
HDC dc = GetDC(hwnd);                                    //переміщення на консоль
const COLORREF white = RGB(255, 255, 255);   //створення константи кольору (білий)
const COLORREF green = RGB(0, 255, 0);       //створення константи кольору
(зелений)
const COLORREF black = RGB(0, 0, 0);         //створення константи кольору (чорний)
const COLORREF red = RGB(255, 0, 0);         //створення константи кольору
(червоний)
void Pixel_Fill(int x,int y,COLORREF bound,COLORREF next)
{
      if (GetPixel(dc,x, y) != bound && GetPixel(dc,x, y) != next) {
            SetPixel(dc, x, y, next);              //зафарбовую піксель в заданий
колір
            //рекурсивний виклик для сусідніх точок
            Pixel_Fill(x + 1, y, bound, next);
            Pixel_Fill(x, y + 1, bound, next);
            Pixel_Fill(x, y - 1, bound, next);
            Pixel_Fill(x - 1, y, bound, next);
      }
}
void Flood_Fill_4(int x, int y, COLORREF old, COLORREF _new) {
      if (GetPixel(dc, x, y) == old) {
            SetPixel(dc, x, y, _new);              //зафарбовую піксель в заданий
колір
            //рекурсивний виклик для сусідніх точок
            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);
      }
}
//Пострічковий алгоритм зафарбовування із затравкою
void Line_Fill(int x, int y, COLORREF bound, COLORREF _new) {
      int xr = x;               //права границя
      int xl = x;               //ліва границя
      //знаходження лівої границі по x
      while (GetPixel(dc, xl, y) != bound) {
            xl--;
      }
      xl++;
      //знаходження правої границі по x
      while (GetPixel(dc, xr, y) != bound) {
            xr++;
      }
      xr--;
      //малюю лінію від xl до xr заданим кольором
      int i = xl;
      while (i <= xr) {
            SetPixel(dc, i, y, _new);
            i++;
      }
      //перевірка нижніх пікселів відносно намальованої лінії
      i = xl;
      while (i <= xr){
            if (GetPixel(dc, i, y + 1) != bound && GetPixel(dc, i, y + 1) != _new)
{
```

```
                  Line_Fill(i, y + 1, bound, _new);
            }
            i++;
      }
      //перевірка верхніх пікселів відносно намальованої лінії
      i = xl;
      while(i <= xr){
            if (GetPixel(dc, i, y - 1) != bound && GetPixel(dc, i, y - 1) != _new)
{
                  Line_Fill(i, y - 1, bound, _new);
            }
            i++;
      }
}


void Link(int& x, int y, int& up, int& down,COLORREF bound) {
      //визначаємо проядок граничних точок
      up = down = 0;
      if (GetPixel(dc,x - 1, y + 1) == bound)up++;
      if (GetPixel(dc, x - 1, y - 1) == bound)down++;
      while (GetPixel(dc, x, y) == bound) {
            if (GetPixel(dc, x, y + 1) == bound && GetPixel(dc, x - 1, y + 1) !=
bound)up++;
            if (GetPixel(dc, x, y - 1) == bound && GetPixel(dc, x - 1, y - 1) !=
bound)down++;
            x++;
      }
      if (GetPixel(dc, x-1, y + 1) != bound && GetPixel(dc, x, y + 1) ==
bound)up++;
      if (GetPixel(dc, x-1, y - 1) != bound && GetPixel(dc, x, y - 1) ==
bound)down++;
}

//Алгоритм зафарбовування області за критерієм парності
void Fill_Parity(COLORREF bound,COLORREF _new) {
      int up, down;                                    //змінні для опису характеру
перетину прямої з дугою контуру
      int l;                                           //лічильник який вказує на
кількість перетинів контуру
      //для зменшенн обчислювальних затрат визначаємо прямокутну оболонку яка
містить нашу область
      int x_min = 100, x_max = 400;
      int y_min = 100, y_max = 400;
      int x;
      for (int y = y_min; y <= y_max; y++) {
            l = 0;
            x = x_min;
            while(x <= x_max){
                  if (GetPixel(dc, x, y) != bound) {
                        //якщо l непарне
                        if (l % 2 != 0)
                              SetPixel(dc, x, y, _new);
                        x++;
                  }
                  else {
                        Link(x, y, up, down,bound);     //визначаємо порядок
граничних точок
                        if (up == 1 && down == 1)l++;
                        if (up + down != 2) {
                              std::cout << "error" << std::endl;
      //повідомлення про помилку
                              return;
```

```
                        }
                    }
                }
        }
}
```

Алгоритм Брезенхема для кола (C++)

```
#include <iostream>
#include <Windows.h>
```

Спочатку реалізуємо участок, який відповідає за попіксельне малювання секторів кола

```
void circle_8(int x, int y, int xc, int yc) {
        HWND console_handle = GetConsoleWindow();
        HDC device_content = GetDC(console_handle);
        const COLORREF white = RGB(255, 255, 255);
        HPEN pen = CreatePen(PS_SOLID, 5, white);
        SelectObject(device_content, pen);
        SetPixel(device_content, xc + x, yc + y, white);
        SetPixel(device_content, xc + y, yc + x, white);
        SetPixel(device_content, xc - x, yc + y, white);
        SetPixel(device_content, xc - y, yc + x, white);
        SetPixel(device_content, xc + x, yc - y, white);
        SetPixel(device_content, xc + y, yc - x, white);
        SetPixel(device_content, xc - x, yc - y, white);
        SetPixel(device_content, xc - y, yc - x, white);
        ReleaseDC(console_handle, device_content);
}
```

Алгоритм Брезенхема для кола

```
void BresenhamCircle() {
        int R, xc, xy;
        std::cout << "Choosing center point." << std::endl;
        std::cout << "Choose center-X, center-Y, radius (it's one number for all three of
them): ";
        std::cin >> R;
        xc = xy = R;
        /*      std::cout << "Input center-X: ";
        std::cin >> xc;
        std::cout << "Input center-Y: ";
        std::cin >> xy;
        std::cout << "Input circle radius: ";
        std::cin >> R;*/
        int x = 0, y = R, d = 3 - 2 * R;
        while (x <= y) {
                circle_8(x, y, xc, xy);
                if (d < 0)d += 4 * x + 6;
                else {
                        d += 4 * (x - y) + 10;
                        y--;
                }
                x++;
        }
}
```

Код програми методу поточного горизонту (C++)

```
#include <iostream>
#include <Windows.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159265
using namespace std;
HWND hwnd = GetConsoleWindow();
HDC dc = GetDC(hwnd);
const COLORREF black = RGB(0, 0, 0);      //чорний колір
HPEN pen = CreatePen(PS_SOLID, 1, black);//створюю ручку чорноо кольору
```

```
const float phi = 30 * PI/ 180; //фі - кут повороту
const float psi = 20 * PI/ 180;  //псі - кут вектора нормалі до картинної площини
void Draw(float centerX, float centerY,float minY,float maxY,float minX,float maxX)
{
		float b = 0.09;
		float c = 0.08;
		float x, y, z;
		float newX, newY;
		float oldX, oldY;
		float x1, y1, z1;
		//ініціалізація горизонтів
		float up[1000];
		float down[1000];
		for (int i = 0; i < 1000; i++) {
			up[i] = 0;
			down[i] = 1000;
		}
		//вибираю ручку
		SelectObject(dc, pen);
		//// побудова 1 лінії
		y = maxY;
		x = minX;
		z = cos(b * sqrt(x * x + y * y)) / (1 + c * sqrt(x * x + y * y));
		z *= 100; //для того щоб z зобразити на екрані

		//перевід координат з простору на картинну площину
		oldX = x * cos(phi) + y * sin(phi);
		oldY = -x * sin(phi) * sin(psi) + y * cos(phi) * sin(psi) + z * cos(psi);
		for (x = minX; x <= maxX; x += 1) {
			z = cos(b * sqrt(x * x + y * y)) / (1 + c * sqrt(x * x + y * y));

			z *= 100;//для того щоб z зобразити на екрані
			//перевід координат з простору на картинну площину
			newX = x * cos(phi) + y * sin(phi);
		newY = -x * sin(phi) * sin(psi) + y * cos(phi) * sin(psi) + z *
cos(psi);
			//заповнення горизонту
			up[int(newX + centerX)] = newY;
			down[int(newX + centerX)] = newY;
			//побудова лінії
			MoveToEx(dc, oldX + centerX, oldY + centerY, NULL);
			LineTo(dc, newX + centerX, newY + centerY);
			//переприсвоюю значення координат
			oldX = newX;
			oldY = newY;
		}
	////побудова 2 лінії
		y = maxY - 8;
		x = minX;
		z = cos(b * sqrt(x * x + y * y)) / (1 + c * sqrt(x * x + y * y));
		z *= 100;//для того щоб z зобразити на екрані
		//перевід координат з простору на картинну площину
		oldX = x * cos(phi) + y * sin(phi);
		oldY = -x * sin(phi) * sin(psi) + y * cos(phi) * sin(psi) + z * cos(psi);
		for (x = minX; x <= maxX; x += 1) {
			z = cos(b * sqrt(x * x + y * y)) / (1 + c * sqrt(x * x + y * y));
			z *= 100;//для того щоб z зобразити на екрані
			//перевід координат з простору на картинну площину
			newX = x * cos(phi) + y * sin(phi);
			newY = -x * sin(phi) * sin(psi) + y * cos(phi) * sin(psi) + z *
cos(psi);
			//заповнення горизонтів
			if (up[int(newX + centerX)] < newY) {
				up[int(newX + centerX)] = newY;
			}
			else {
				down[int(newX + centerX)] = newY;
```

```
                    }

                    //побудова лінії
                    MoveToEx(dc, oldX + centerX, oldY + centerY, NULL);
                    LineTo(dc, newX + centerX, newY + centerY);
                    //переприсвоюю значення координат
                    oldX = newX;
                    oldY = newY;
             }
      //побудова всіх інших ліній
             for (y = maxY - 16; y >= minY; y -= 8) {
                    x = minX;
                    z = cos(b * sqrt(x * x + y * y)) / (1 + c * sqrt(x * x + y * y));
                    z *= 100;
                    //перевід координат з простору на картинну площину
                    oldX = x * cos(phi) + y * sin(phi);
                    oldY = -x * sin(phi) * sin(psi) + y * cos(phi) * sin(psi) + z *
cos(psi);
                    for (x = minX + 10; x <= maxX; x += 1) {
                    z = cos(b * sqrt(x * x + y * y)) / (1 + c * sqrt(x * x + y * y));
                           z *= 100;
                           //перевід координат з простору на картинну площину
                           newX = x * cos(phi) + y * sin(phi);
                           newY = -x * sin(phi) * sin(psi) + y * cos(phi) * sin(psi) +
                           z* cos(psi);
                           //перевірка методом поточного горизонту
                           if (up[int(newX + centerX)] < newY) {
                                  up[int(newX + centerX)] = newY;
                                  //побудова лінії
                                  MoveToEx(dc, oldX + centerX, oldY + centerY, NULL);
                                  LineTo(dc, newX + centerX, newY + centerY);
                           }
                           else if (down[int(newX + centerX)] > newY) {
                                  down[int(newX + centerX)] = newY ;
                                  //побудова лінії
                                  MoveToEx(dc, oldX + centerX, oldY + centerY, NULL);
                                  LineTo(dc, newX + centerX, newY + centerY);
                           }
                           //переприсвоюю значення координат
                           oldX = newX;
                           oldY = newY;
                    }
      }

}
int main()
{
      //зміна кольору консолі на білий
      system("color F0");
      //виклик функції поточного горизонту
      Draw(450,250,-300,300,-300,300);
      ReleaseDC(hwnd, dc);
      _getch();
}
```

Код програми алгоритму Z-буфера (C++)

```
#include <iostream>
#include <Windows.h>
#include <conio.h>
#include <math.h>
#define MAXDIST 1000.0 // максимальна глибина сцени
#define MAXYLINES 500 // максимальна кількість ліній в сцені
#define color_ 3; // колір за замовчуванням
typedef struct Point3d POINT3D;
typedef struct Cell CELL;
HWND hwnd = GetConsoleWindow();
```

```
HDC hdc;
// структура для точки в 3-мірному просторі
struct Point3d {
	double x, y, z;
};
// структура комірки, із яких буде складатися z-буфер
struct Cell {
	double z;
	int color;
};
//Клас трикутник
class Triangle {
public:
	int color;
	POINT3D p[3];
	Triangle(POINT3D p1, POINT3D p2, POINT3D p3, int c) {
		p[0] = p1; p[1] = p2; p[2] = p3;
		color = c;
	}
};
//Клас Z-буфера.
class ZBuffer {
public:
	CELL *buff[MAXYLINES];
	int sX, sY;   // розмір Z-Буфера
	ZBuffer(int, int);
	~ZBuffer();
	void PutTriangle(Triangle&);
	void Show();
	void Clear();
};
//Конструктор Z-буфера.
ZBuffer::ZBuffer(int ax, int ay) {
	sX = ax; sY = ay;
	for (int i = 0; i < sY; i++) {
		// виділення пам'яті під комірки
		buff[i] = new CELL[sX * sizeof(Cell)];
		//buff[i] = (struct Cell *)malloc(sX * sizeof(struct Cell));
	}
}
//Деструктор Z-буфера.
ZBuffer :: ~ZBuffer() {
	for (int i = 0; i < sY; i++)
		delete buff[i];
		//free(buff[i]);
}
// Функція, яка виводить на екран вміст буфера
void ZBuffer::Show() {
	for (int j = 0; j < sY; j++)
		for (int i = 0; i < sX; i++) {
			// Виводимо пікселі на екран
			SetPixel(hdc, 50 + i, j + 50, 16711680 / (*(buff[j] + i)).color);
			//std::cout << 16711680 / (*(buff[j] + i)).color << " ";
			//Pixels[50 + i][j] = clBlue / (*(buff[j] + i)).color;
		}
}
//Функція, яка очищує Z-буфер.
void ZBuffer::Clear() {
	for (int j = 0; j < sY; j++)
		for (int i = 0; i < sX; i++) {
			// Ініціалізація комірки Z-буфера
			(*(buff[j] + i)).z = MAXDIST;
			(*(buff[j] + i)).color = color_;
		}
}
void ZBuffer::PutTriangle(Triangle& t) {
	int ymax, ymin, ysc, e1, e, i;
```

```
	int x[3], y[3];
	//Заносим x,y із t в масиви для наступної роботи з ними
	for (i = 0; i < 3; i++)
		x[i] = int(t.p[i].x), y[i] = int(t.p[i].y);
	// Визначаємо макс і мін y
	ymax = ymin = y[0];
	if (ymax < y[1])
		ymax = y[1];
	else if (ymin > y[1])
		ymin = y[1];
	if (ymax < y[2])
		ymax = y[2];
	else if (ymin > y[2])
		ymin = y[2];
	ymin = (ymin < 0) ? 0 : ymin;
	ymax = (ymax < sY) ? ymax : sY;
	int ne;
	int x1, x2, xsc1, xsc2;
	double z1, z2, tc, z;
	// Перебираємо всі рядки сцени і визначаємо глибину кожного пікселя для
відповідного трикутника
	for (ysc = ymin; ysc < ymax; ysc++) {
		ne = 0;
		for (int e = 0; e < 3; e++) {
			e1 = e + 1;
			if (e1 == 3) e1 = 0;
			if (y[e] < y[e1]) {
				if (y[e1] <= ysc || ysc < y[e])
					continue;
			}
			else if (y[e] > y[e1]) {
				if (y[e1] > ysc || ysc >= y[e])
					continue;
			}
			else
				continue;
			tc = double(y[e] - ysc) / (y[e] - y[e1]);
			if (ne)
				x2 = x[e] + int(tc * (x[e1] - x[e])),
				z2 = t.p[e].z + tc * (t.p[e1].z - t.p[e].z);
			else
				x1 = x[e] + int(tc * (x[e1] - x[e])),
				z1 = t.p[e].z + tc * (t.p[e1].z - t.p[e].z),
				ne = 1;
		}
		if (x2 < x1)
			e = x1, x1 = x2, x2 = e, tc = z1, z1 = z2, z2 = tc;
		xsc1 = (x1 < 0) ? 0 : x1;
		xsc2 = (x2 < sX) ? x2 : sX;
		for (int xsc = xsc1; xsc < xsc2; xsc++) {
			tc = double(x1 - xsc) / (x1 - x2);
			z = z1 + tc * (z2 - z1);
			// Якщо отримана глибина пікселя менше тої, що знаходиться вже у
буфері - міняємо її на нову
			if (z < (*(buff[ysc] + xsc)).z)
				(*(buff[ysc] + xsc)).color = t.color,
				(*(buff[ysc] + xsc)).z = z;
		}
	}
}
// Створюимо 6 наборів для 6 трикутників
POINT3D p[6][3] = {
{{  40.0,  120.0,  100.0}, {  120.0,  40.0,  100.0}, { 360.0, 280.0,   0.0}},
{{  160.0,  40.0,   0.0}, {  160.0, 360.0,  100.0}, {  80.0, 320.0,  100.0}},
{{  40.0, 280.0,   0.0}, { 300.0,  120.0,  100.0}, { 360.0, 200.0,  100.0}},
{{  180.0,  20.0, 200.0}, {  20.0,  180.0, 200.0}, { 420.0, 380.0, 200.0}},
{{ 200.0,  100.0, 200.0}, { 200.0, 300.0,   0.0}, { 400.0, 200.0,  100.0}},
```

```
{{ 100.0, 200.0,  100.0}, { 300.0, 300.0, 200.0}, { 300.0,  100.0,   0.0}},
};
// Ініціалізумо трикутники
Triangle      t1(p[0][0], p[0][1], p[0][2], 13),
t2(p[1][0], p[1][1], p[1][2], 14),
t3(p[2][0], p[2][1], p[2][2], 12),
t4(p[3][0], p[3][1], p[3][2], 10),
t5(p[4][0], p[4][1], p[4][2], 11),
t6(p[5][0], p[5][1], p[5][2], 9);
// Функція, яка демонструє роботу алгоритму буфера
void Buffer() {
       ZBuffer * zb;
       zb = new ZBuffer(440, 400); // створюємо буфер з необхідними розмірами
       zb->Clear(); // Готуємо до заповнення
       // Заповнюємо буфер
       zb->PutTriangle(t1);
       zb->PutTriangle(t2);
       zb->PutTriangle(t3);
       zb->PutTriangle(t4);
       zb->PutTriangle(t5);
       zb->PutTriangle(t6);
       // Виводимо на екран
       zb->Show();
}
int main()
{
       hdc = GetDC(hwnd);
       Buffer();
       ReleaseDC(hwnd, hdc);
       _getch();
}
```

Ітераційний метод побудови фрактала Коха (C++)

```
#include <GLFW/glfw3.h>
#include <iostream>
#include <cmath>
#include "windows.h"
using namespace std;
#define M_PI 3.14159265358979323846
double P[100000000][2];
long int p1 = 0, iteration = 0;
void KOCHit(int num,double x1, double y1, double x2, double y2)
{
    int n=0,i=0,k,i1=1,p=0;
    double xV,yV;
    double xstart = x1, ystart = y1;
    double xend = x2, yend = y2;
    double leftX, leftY;
    double rightX, rightY;
    double highX, highY;
    double len;
    n = 2;

    for (i1 = 1; i1 <= num; i1++)
    {
        if (i1 == num)
        {
            iteration = p1;
        }
        xstart = x1;
        ystart = y1;

        P[++p1][0] = xstart;
        P[p1][1] = ystart;
```

```
        for (k = 1; k < n; k++)
        {

            P[++p1][0] = xstart;
            P[p1][1] = ystart;
            xV = (xend - xstart) / 3;
            yV = (yend - ystart) / 3;
            leftX = xstart + xV;
            leftY = ystart + yV;
            rightX = xend - xV;
            rightY = yend - yV;
            highX = (rightX - leftX) * cos(M_PI / 3) - (rightY - leftY) * sin(M_PI / 3)
+ leftX;
            highY = (rightX - leftX) * sin(M_PI / 3) + (rightY - leftY) * cos(M_PI / 3)
+ leftY;
            P[++p1][0] = leftX;
            P[p1][1] = leftY;
            P[++p1][0] = highX;
            P[p1][1] = highY;
            P[++p1][0] = rightX;
            P[p1][1] = rightY;
            P[++p1][0] = xend;
            P[p1][1] = yend;
            xstart = P[p + 1][0];
            ystart = P[p + 1][1];
            xend = P[p + 2][0];
            yend = P[p + 2][1];
            p += 1;
        }
        n = p1;
    }
}
int build(double x1, double y1,double x2, double y2)
{
    glBegin(GL_LINES);
    glColor3d(0, 0, 0);
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
    glEnd();
    return 0;
}
int main(void)
{
    GLFWwindow* window;
    double x1 = -0.6, y1 = 0.5, x2, y2=0.5, x3, y3,a=1.3;
    int num;
    cout << "Enter num: ";
    cin >> num;
   x2 = x1 + a;
   x3 = (x2 - x1) * cos(-M_PI / 3) - (y2 - y1) * sin(-M_PI / 3) + x1;
   y3 = (x2 - x1) * sin(-M_PI / 3) + (y2 - y1) * cos(-M_PI / 3) + y1;
    // Initialize the library
    if (!glfwInit()) {
        return -1;
    }
    // Create a windowed mode window and its OpenGL context
    window = glfwCreateWindow(1080, 1080, "Hello World", NULL, NULL);
    if (!window)
    {
        glfwTerminate();
        return -1;
    }
    // Make the window's context current
    glfwMakeContextCurrent(window);
    // Loop until the user closes the window
    float flag = 0;
```

```
while (!glfwWindowShouldClose(window))
{
           glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    KOCHit(num, x1, y1, x2, y2);
    if (num == 3)
    {
       p1 -=25;
    }
    else if (num == 4)
    {
        p1 -= 300;
    }
    else if (num == 5)
    {
        p1 -= 2620;
    }
    else if (num == 6)
    {
        p1 -= 19820;
    }
    else if (num == 7)
    {
        p1 -= 140000;
    }
    while (p1 > (iteration+1))
    {
        build(P[p1][0], P[p1][1],P[p1-1][0],P[p1-1][1]);
        p1--;
    }
    p1 = 0;
    KOCHit(num, x2, y2, x3, y3);
    if (num == 3)
    {
        p1 -= 25;
    }
    else if (num == 4)
    {
        p1 -= 300;
    }
    else if (num == 5)
    {
        p1 -= 2620;
    }
    else if (num == 6)
    {
        p1 -= 19820;
    }
    else if (num == 7)
    {
        p1 -= 140000;
    }
     while (p1 > (iteration+1))
    {
        build(P[p1][0], P[p1][1],P[p1-1][0],P[p1-1][1]);
        p1--;
    }
    p1 = 0;
    KOCHit(num, x3, y3, x1, y1);
    if (num == 3)
    {
        p1 -= 25;
    }
    else if (num == 4)
    {
        p1 -= 300;
```

```
        }
        else if (num == 5)
        {
            p1 -= 2620;
        }
        else if (num == 6)
        {
            p1 -= 19820;
        }
        else if (num == 7)
        {
            p1 -= 140000;
        }
        while (p1 > (iteration+1))
        {
            build(P[p1][0], P[p1][1],P[p1-1][0],P[p1-1][1]);
            p1--;
        }
        p1 = 0;
        glfwSwapBuffers(window);
        // Swap front and back buffers
        // Poll for and process events
        glfwPollEvents();
    }
    glfwTerminate();
    return 0;
}
```

Код програми мовою Python для фрактала

```
import turtle
def MincIsland(lenght: float, iterations: int):
  if iterations == 0:
    turtle.forward(lenght)                      #проводимо лінію довжиною lenght
  else:

    turtle.left(45)
    MincIsland(lenght/2.5, iterations-1)
    turtle.right(100)
    MincIsland(lenght/1.25, iterations-1)
    turtle.left(100)
    MincIsland(lenght/2.5, iterations-1)
    turtle.right(45)
Len = 100
Iter = 1
turtle.speed(10000)    #обираємо зручну швидкість
turtle.hideturtle()
turtle.up();            #піднімаємо перо та зміщаємо його в точку -100;200
turtle.goto(-100, 200)
turtle.down()           #опускаємо перо
MincIsland(Len, Iter)   #викликаємо функцію та повертаємось на 90 градусів для заміни
сторін квадрата
turtle.right(90)
MincIsland(Len, Iter)
turtle.right(90)
MincIsland(Len, Iter)
turtle.right(90)
MincIsland(Len, Iter)
turtle.mainloop()       #затримуємо екран, щоб після закінчення роботи граф.вікно не
закрилося
```

Завдання: Написати програму для відображення різнокольорового куба в тривимірному просторі з освітленням сцени, можливістю керування камерою та виконання графічних операцій з кубом з використанням функцій OpenGL

Код програми на мові C++:

```
// GLAD - бібліотека, що допомагає автоматично отримати та під'єднати вказівники на функції OpenGL
#include <glad/gl.h>
// Явно вказуємо, що не потрібно використовувати бібліотеки, які було підключено у файлах GLFW
#define GLFW_INCLUDE_NONE
// GLFW - бібліотека, яка використовується для створення контексту відображення OpenGL
// GLM - бібліотека математичних функцій, ящо використовуються в комп'ютерній графіці. Чудово інтегрується з OpenGL
#include <GLFW/glfw3.h>
#include <glm/gtc/type_ptr.hpp>
#include "Camera.hpp"
#include <iostream>
// Розміри вікна
const uint32_t SCR_WIDTH = 1200;
const uint32_t SCR_HEIGHT = 800;
// Довжина половини осі
const int SEMIAXIS = 100;
// Об'єкт камери. Встановюємо її позицію за координатами (-3, 1, -6) першим параметром,
// другий параметр - вектор напряму. Наступні два це поворот в площині XOZ на кут 65 градусів та нахил вперед на -5 градусів
Camera camera(glm::vec3(-3.0f, 1.0f, -6.0f), glm::vec3(0.0f, 1.0f, 0.0f), 65.0f, -5.0f);
float deltaTime = 0.0f, lastFrame = 0.0f;
// Прототипи функцій для малювання основних частин сцени
void drawAxis();
void drawPlane();
void drawCube();
// Прототипи керуючих функцій
void processInput(GLFWwindow* window);
void framebufferSizeCallback(GLFWwindow* window, int width, int height);
void mouseCallback(GLFWwindow* window, double xposIn, double yposIn);
void scrollCallback(GLFWwindow* window, double xoffset, double yoffset);
int main(int argc, char const* argv[]) {
    // Ініціалізуємо бібліотеку GLFW
    if (!glfwInit()) return -1;
    // Створюємо дескриптор вікна відповідного розміру
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "Cute cube", NULL, NULL);
    // Якщо не вдалося створити вікно, аварійно завершуємо програму
    if (window == NULL) {
        glfwTerminate();
        exit(EXIT_FAILURE);
    }
    // Робимо даний контекст відтворення поточним
    glfwMakeContextCurrent(window);
    // Вимикаємо курсор і встановлюємо його по центру, щоб в подальшому змогли керувати напрямком камери
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
    // Під'єднуємо керуючі функції
    glfwSetFramebufferSizeCallback(window, framebufferSizeCallback);
    glfwSetCursorPosCallback(window, mouseCallback);
    glfwSetScrollCallback(window, scrollCallback);
    // Завантажуємо бібліотеку GLAD
    gladLoadGL(glfwGetProcAddress);
    float theta = 0.0;              // Кут повороту
    glEnable(GL_LIGHTING);          // Вмикаємо освітлення
    glEnable(GL_LIGHT0);            // Встановлюємо джерело світла #0
    glEnable(GL_COLOR_MATERIAL); // Дозволяємо відображення кольорів під освітленням
    glEnable(GL_NORMALIZE);      // Нормалізуємо вектори нормалі поверхонь, для плавного переходу тіней
    glEnable(GL_AMBIENT);                   // Вмикаємо режим розсіяного світла
    // Вмикаємо режим змішування кольорів
    glEnable(GL_BLEND);
    // Фактор змішування. Встановлюючи ці значення, ми зможемо використовувати альфа-канал,
    // тобто встановлювати прозорість кольору
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

```
    glEnable(GL_DEPTH_TEST);      // Вмикаємо тест глибини для коректного відображення 3д об'єктів
    // Доки не закриємо вікно...
    while (!glfwWindowShouldClose(window)) {
                        // Поточний час таймера та різниця між поточним та попереднім значенням
                        // Використовується для оновлення стану камери
                        float currentFrame = static_cast<float>(glfwGetTime());
                        deltaTime = currentFrame - lastFrame;
                        lastFrame = currentFrame;
                        // Викликаємо функцію для обробки натиснених клавіш
                        processInput(window);
                        // Встановлюємо колір очищення екрану (колір фону)
                        glClearColor(0.93f, 0.93f, 0.93f, 1.0f);
                        // Вказуємо значення глибини, яке використовує glClear для очистки буфера глибини
                        glClearDepth(1.0f);
                        // Очищуємо буфер кольору та буфер глибини
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
                        // Копіюємо поточну матрицю у вершину стека
                        glPushMatrix();
                                // Встановлюємо режим проекції. Це означає, що всі дії,
                        // що відбуватимуться над матрицею, впливатимуть тільки на матрицю проекції
                        glMatrixMode(GL_PROJECTION);
                        // Функція glm::perspective() повертає проекційну матрицю перспективи розміром 4x4.
                        // Перший параметр - кут огляду по вертикалі, другий - певний аспект,
                        // співвідношення між шириною та висотою прямокутної області, яка буде об'єктом
проектування.
                        // Наступні два параметри - відстань ближньої та дальної площин, що обмежують область
перегляду
                        auto projection = glm::perspective(glm::radians(camera.Zoom), (float) SCR_WIDTH /
SCR_HEIGHT, 0.1f, 1000.0f);
                        // Встановлюємо нову матрицю (проекції)
                        glLoadMatrixf((const GLfloat*) glm::value_ptr(projection));
                        // Аналогічно як для проекції, тільки встановлюємо режим матриці представлення
                        glMatrixMode(GL_MODELVIEW);
                        // Саму матрицю отримуємо з об'єкту камери, методом GetViewMatrix(), де проводяться всі
обчислення
                        auto view = camera.GetViewMatrix();
                        glLoadMatrixf((const GLfloat*) glm::value_ptr(view));
                        // Встановлюєм позицію джерела світла за координатами (-1, 1, -1)
                        GLfloat lightPosition[] = { -1, 1, -1, 0 };
                        glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
                        drawAxis();                     // Малюємо осі
                        drawPlane();        // Малюємо площину (сітку)
                        glRotatef(theta, 0.0f, 1.0f, 0.0f);       // Повертаємо систему координат на кут theta навколо осі Y
                        glTranslatef(0, 1, 1);                                  // Переносимо на вектор (0, 1, 1)
                        drawCube();             // Малюємо куб
                        glPopMatrix();      // Витягуємо збережену матрицю зі стека
                        theta += 0.07;      // Збільшуємо кут обертання
        glfwSwapBuffers(window); // Ця функція міняє місцями передній і задній буфери вказаного вікна
        glfwPollEvents();                   // Функція, що обробляє отримані події
    }
    glfwTerminate();
    return EXIT_SUCCESS;
}
// Функція для малювання координатних осей
void drawAxis() {
            glLineWidth(3);    // Встановлюємо ширину ліній у значення 3
            glBegin(GL_LINES);         // Режим малювання ліній
            // Однаковий вектор нормалі для усіх осей, щоб вони виділялися
            // серед інших ліній під освітленням
            glNormal3f(0, 1, 0);
            // Червона вісь X
            glColor3f(1, 0, 0);   // Задаємо колір в RGB форматі, кожне значення лежить в діапазоні від 0 до 1
            glVertex3f(-SEMIAXIS, 0, 0); // glVertex3f(...) задає вершину у тривимірному просторі
            glVertex3f(SEMIAXIS, 0, 0);
            // Зелена вісь Y
            glColor3f(0, 1, 0);
            glVertex3f(0, -SEMIAXIS, 0);
            glVertex3f(0, SEMIAXIS, 0);
```

```
	// Синя вісь Z
	glColor3f(0, 0, 1);
	glVertex3f(0, 0, -SEMIAXIS);
	glVertex3f(0, 0, SEMIAXIS);
	glEnd();
}
void drawPlane() {
	glLineWidth(3);
	glBegin(GL_LINES);
	glColor4f(1.0f, 1.0f, 1.0f, 0.3f);	// Тут використовуємо додатковий альфа-канал для прозорості
	// Малюємо сітку в квадраті зі стороною 2 * SEMIAXIS
	for (int i = -SEMIAXIS; i < SEMIAXIS; i++) {
		if (i == 0) continue;
		// Для кожної лінії встановлюємо свої нормалі
		glNormal3f(0, 1, -1);
		glVertex3f(i, -0.005, -SEMIAXIS);
		glNormal3f(0, 1, 1);
		glVertex3f(i, -0.005, SEMIAXIS);
		glNormal3f(-1, 1, 0);
		glVertex3f(-SEMIAXIS, -0.005, i);
		glNormal3f(1, 1, 0);
		glVertex3f(SEMIAXIS, -0.005, i);
	}
	glEnd();
}
// Функція для малювання куба
void drawCube() {
	// Координати вершин куба
	GLfloat vertices[] = {
		 0.5f, 0.5f, 0.5f,  -0.5f, 0.5f, 0.5f,  -0.5f,-0.5f, 0.5f,   0.5f,-0.5f, 0.5f, // Передня грань
		 0.5f, 0.5f, 0.5f,   0.5f,-0.5f, 0.5f,   0.5f,-0.5f,-0.5f,   0.5f, 0.5f,-0.5f, // Права грань
		 0.5f, 0.5f, 0.5f,   0.5f, 0.5f,-0.5f,  -0.5f, 0.5f,-0.5f,  -0.5f, 0.5f, 0.5f, // Верхня грань
		-0.5f, 0.5f, 0.5f,  -0.5f, 0.5f,-0.5f,  -0.5f,-0.5f,-0.5f,  -0.5f,-0.5f, 0.5f, // Ліва грань
		-0.5f,-0.5f,-0.5f,   0.5f,-0.5f,-0.5f,   0.5f,-0.5f, 0.5f,  -0.5f,-0.5f, 0.5f, // Нижня грань
		 0.5f,-0.5f,-0.5f,  -0.5f,-0.5f,-0.5f,  -0.5f, 0.5f,-0.5f,   0.5f, 0.5f,-0.5f  // Задня грань
	};
// Масив нормалей для кожної грані
	GLfloat normals[] = {
		 0, 0, 1,   0, 0, 1,   0, 0, 1,   0, 0, 1,  // Передня грань
		 1, 0, 0,   1, 0, 0,   1, 0, 0,   1, 0, 0,  // Права грань
		 0, 1, 0,   0, 1, 0,   0, 1, 0,   0, 1, 0,  // Верхня грань
		-1, 0, 0,  -1, 0, 0,  -1, 0, 0,  -1, 0, 0,  // Ліва грань
		 0,-1, 0,   0,-1, 0,   0,-1, 0,   0,-1, 0,  // Нижня грань
		 0, 0,-1,   0, 0,-1,   0, 0,-1,   0, 0,-1   // Задня грань
	};
	// Масив кольорів
	GLfloat colors[] = {
		 1, 1, 1,   1, 1, 0,   1, 0, 0,   1, 0, 1,  // Передня грань
		 1, 1, 1,   1, 0, 1,   0, 0, 1,   0, 1, 1,  // Права грань
		 1, 1, 1,   0, 1, 1,   0, 1, 0,   1, 1, 0,  // Верхня грань
		 1, 1, 0,   0, 1, 0,   0, 0, 0,   1, 0, 0,  // Ліва грань
		 0, 0, 0,   0, 0, 1,   1, 0, 1,   1, 0, 0,  // Нижня грань
		 0, 0, 1,   0, 0, 0,   0, 1, 0,   0, 1, 1   // Задня грань
	};
	// Вмикаємо використання масиву вершин
	glEnableClientState(GL_VERTEX_ARRAY);
	// І з'єднуємо його з вказівником на наш створений масив вершин
	glVertexPointer(3, GL_FLOAT, 0, &vertices);
	// Аналогічно для масиву кольорів...
	glEnableClientState(GL_COLOR_ARRAY);
	glColorPointer(3, GL_FLOAT, 0, &colors);
	// ...та масиву нормалей
	glEnableClientState(GL_NORMAL_ARRAY);
	glNormalPointer(GL_FLOAT, 0, &normals);
	// Малюємо наш об'єкт відповідно до інформації із заданих масивів
	// GL_QUADS означає що кожні чотири вершини сприйматимуться як один чотирикутник
	glDrawArrays(GL_QUADS, 0, 24);
	// Ці масиви нам більше не потрібні, тому відключаємо їх використання
```

```
	glDisableClientState(GL_VERTEX_ARRAY);
	glDisableClientState(GL_COLOR_ARRAY);
	glDisableClientState(GL_NORMAL_ARRAY);
}
// Функція для обробки подій клавіатури
void processInput(GLFWwindow* window) {
	// При натисненні Escape закриваємо нашу програму
	if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
		glfwSetWindowShouldClose(window, true);
	// Переміщуємо камеру відповідно до натисненої клавіші
	if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
		camera.ProcessKeyboard(FORWARD, deltaTime);
	if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
		camera.ProcessKeyboard(BACKWARD, deltaTime);
	if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
		camera.ProcessKeyboard(LEFT, deltaTime);
	if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
		camera.ProcessKeyboard(RIGHT, deltaTime);
}
// Функція для обробки зміни розміру вікна
void framebufferSizeCallback(GLFWwindow* window, int width, int height) {
	glViewport(0, 0, width, height);
}
// Обробка подій миші
void mouseCallback(GLFWwindow* window, double xposIn, double yposIn) {
	float xpos = static_cast<float>(xposIn);
	float ypos = static_cast<float>(yposIn);
	static bool firstMouse = true;
	static float lastX = SCR_WIDTH / 2.0f;
	static float lastY = SCR_HEIGHT / 2.0f;
	if (firstMouse) {
		lastX = xpos, lastY = ypos;
		firstMouse = false;
	}
	float xoffset = xpos - lastX;
	float yoffset = lastY - ypos;
	lastX = xpos;
	lastY = ypos;
	// Вирахавуємо зміщення миші по обох осях і передаємо відповідні значення
	// у метод об'єкту камери для її переміщення
	camera.ProcessMouseMovement(xoffset, yoffset);
}
// Обробка подій коліщатка миші
void scrollCallback(GLFWwindow* window, double xoffset, double yoffset) {
	camera.ProcessMouseScroll(static_cast<float>(yoffset));
}
```

Файл Camera.hpp:

```
#ifndef CAMERA_H
#define CAMERA_H
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
// Визначає кілька можливих варіантів руху камери
enum CameraMovement { FORWARD, BACKWARD, LEFT, RIGHT };
// Стандартні значення камери
const float YAW = -90.0f;	// Нахил уверх або вниз
const float PITCH = 0.0f;	// Поворот в площині XOZ
const float SPEED = 2.5f;	// Швидкість
const float SENSITIVITY = 0.1f; // Чутливість
const float ZOOM = 45.0f;	// Масштабування
// Клас камери, який обробляє вхідні дані та обчислює відповідні кути Ейлера,
// вектори та матриці для використання в OpenGL
class Camera {
  public:
    // Атрибути камери
    glm::vec3 Position;
    glm::vec3 Front;
    glm::vec3 Up;
    glm::vec3 Right;
```

```
    glm::vec3 WorldUp;
    // Кути Ейлера
    float Yaw;
    float Pitch;
    // Параметри камери
    float MovementSpeed;
    float MouseSensitivity;
    float Zoom;
    // Конструктор з векторами
    Camera(glm::vec3 position = glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f), float yaw =
YAW, float pitch = PITCH)
            : Front(glm::vec3(0.0f, 0.0f, -1.0f)), MovementSpeed(SPEED), MouseSensitivity(SENSITIVITY),
Zoom(ZOOM) {
        Position = position;
        WorldUp = up;
        Yaw = yaw;
        Pitch = pitch;
        updateCameraVectors();
    }
    // Конструктор зі скалярними значеннями
    Camera(float posX, float posY, float posZ, float upX, float upY, float upZ, float yaw, float pitch) :
Front(glm::vec3(0.0f, 0.0f, -1.0f)), MovementSpeed(SPEED), MouseSensitivity(SENSITIVITY), Zoom(ZOOM) {
        Position = glm::vec3(posX, posY, posZ);
        WorldUp = glm::vec3(upX, upY, upZ);
        Yaw = yaw;
        Pitch = pitch;
        updateCameraVectors();
    }
    // Повертає матрицю перегляду, обчислену за допомогою кутів Ейлера та матриці LookAt
    glm::mat4 GetViewMatrix() {
        return glm::lookAt(Position, Position + Front, Up);
    }
    // Обробляє введення, отримане з клавіатури
    // Приймає вхідний параметр у формі визначеного камерою ENUM
    void ProcessKeyboard(CameraMovement direction, float deltaTime) {
        float velocity = MovementSpeed * deltaTime;
        if (direction == FORWARD)
            Position += Front * velocity;
        if (direction == BACKWARD)
            Position -= Front * velocity;
        if (direction == LEFT)
            Position -= Right * velocity;
        if (direction == RIGHT)
            Position += Right * velocity;
    }
   // Обробляє вхідні дані, отримані від системи введення мишею. Очікує значення зсуву в обох напрямках x і y.
    void ProcessMouseMovement(float xoffset, float yoffset, GLboolean constrainPitch = true) {
        xoffset *= MouseSensitivity;
        yoffset *= MouseSensitivity;
        Yaw += xoffset;
        Pitch += yoffset;
        // Переконуємося, що коли кут нахилу виходить за межі, екран не перевертається
        if (constrainPitch) {
            if (Pitch > 89.0f) Pitch = 89.0f;
            if (Pitch < -89.0f) Pitch = -89.0f;
        }
        // Оновлення переднього, правого та верхнього векторів, використовуючи оновлені кути Ейлера
        updateCameraVectors();
    }
    // Обробляє вхідні дані, отримані від події коліщатка прокрутки миші
    void ProcessMouseScroll(float yoffset) {
        Zoom -= (float)yoffset;
        if (Zoom < 1.0f) Zoom = 1.0f;
        if (Zoom > 45.0f) Zoom = 45.0f;
    }
  private:
    // Обчислює фронтальний вектор на основі оновлених кутів Ейлера камери
    void updateCameraVectors() {
```

```
        // Обчислення нового фронтального вектора
        glm::vec3 front;
        front.x = cos(glm::radians(Yaw)) * cos(glm::radians(Pitch));
        front.y = sin(glm::radians(Pitch));
        front.z = sin(glm::radians(Yaw)) * cos(glm::radians(Pitch));
        Front = glm::normalize(front)
        // Також повторно обчислюємо правий та верхній вектори.
        // Нормалізуємо вектори, тому що їх довжина наближається до 0,
        // чим більше камера дивиться вгору або вниз, що призводить до повільнішого руху.
        Right = glm::normalize(glm::cross(Front, WorldUp));
        Up = glm::normalize(glm::cross(Right, Front));
    }
};

#endif
```