МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ПІДРУЧНИК

з дисципліни

“ТЕХНІЧНІ, МАТЕМАТИЧНІ ТА АЛГОРИТМІЧНІ ОСНОВИ КОМП’ЮТЕРНОЇ ГРАФІКИ”

з дисципліни “Комп’ютерна графіка”
для студентів базових напрямків
“Комп'ютерні науки” та “Видавничо-поліграфічна справа”

Львів 2004

Технічні, математичні та алгоритмічні основи комп’ютерної графіки: Методичні вказівки до вивчення курсу лекцій з дисципліни “ Комп’ютерна графіка” /Укл.: Різник О.Я. – Львів.

Укладач: Різник Олег Яремович, канд. техн. наук, доц.

Комп'ютерна верстка: Різник Олег Яремович, канд. техн. наук, доц.

Відповідальна за випуск: Дронюк Іванна Мирославівна, канд. фіз.-мат. наук, доц.

Рецензенти: Різник Володимир Васильович, докт. техн. наук, проф.
Вальковський Володимир Олександрович, докт. фіз.-мат. наук, проф.

ЗМІСТ

Вступ **5**
Лекція 1. Історія машинної графіки. Засоби графічної взаємодії ЕОМ і людини. Класифікація і коротка характеристика **7**
1.1. Історія машинної графіки **7**
1.2. Синтез зображення **8**
1.3. Видалення схованих поверхонь **9**
1.4. Зафарбування **10**
1.5. Приклад, який ілюструє синтез зображення **12**
Лекція 2. Апаратні засоби комп'ютерної графіки **14**
2.1. Дисплеї (монітори) **14**
2.2. Світлове перо (light pen) **21**
2.3. Маніпулятор "миша" **22**
2.4. Джойстики **23**
2.5. Трекбол (Trackball) **24**
2.6. Дигитайзери **25**
Лекція 3. Математичні основи комп'ютерної графіки. Відображення, обертання, масштабування. Представлення і перетворення точок і відрізків **27**
3.1. Представлення і перетворення точок **27**
3.2. Перетворення прямих ліній **28**
3.3. Обертання **29**
3.4. Відображення **30**
3.5. Зміна масштабу **30**
3.6. Двовимірний зсув і однорідні координати **31**
Лекція 4. Однорідні координати. Тривимірні перетворення і стандартні проекції. Перспективні перетворення **32**
4.1. Тривимірні перетворення **32**
4.1.1. Вступ **32**
4.1.2. Тривимірна зміна масштабу **34**
4.1.3. Тривимірний зсув **34**
4.1.4. Тривимірні обертання **38**
4.1.5. Відображення в просторі **39**
4.1.6. Просторовий перенос **39**
4.1.7. Тривимірне обертання навколо довільної осі **40**
4.1.8. Афінна і перспективна геометрія **41**
4.2. Тривимірні проекції **41**
4.2.1. Вступ **42**
4.2.2. Центральні проекції **42**
4.2.3. Паралельні проекції **44**
4.2.4. Математичний опис плоских геометричних проекцій **48**
Лекція 5. Растрова графіка. Представлення точок і зображень. Алгоритми креслення відрізків і кіл **53**
5.1. Алгоритм креслення відрізків **53**
5.2. Цифровий диференціальний аналізатор **54**

5.3. Приклад, який ілюструє роботу алгоритму цифрового диференціального аналізатора в першому квадранті **54**
5.4. Приклад, який ілюструє роботу алгоритму цифрового диференціального аналізатора в третьому квадранті **56**
5.5. Алгоритм Брезенхема **57**
5.6. Приклад, який ілюструє роботу алгоритму Брезенхема в першому октанті **60**
5.7. Загальний алгоритм Брезенхема **61**
5.8. Приклад, який ілюструє роботу узагальненого алгоритму Брезенхема **62**
5.9. Алгоритм Брезенхема для генерації кола **63**
5.10. Приклад, який ілюструє роботу алгоритму Брезенхема для кола **70**
Лекція 6. Растрова розгортка суцільних областей. Алгоритми заповнення контурів. Алгоритми зафарбування багатокутників **72**
6.1. Растрова розгортка суцільних областей **72**
6.2. Заповнення багатокутників **72**
6.3. Растрова розгортка багатокутників **73**
6.4. Простий алгоритм з упорядкованим списком ребер **76**
6.5. Приклад, який ілюструє роботу простого алгоритму з упорядкованим списком ребер **76**
6.6. Більш ефективний алгоритм з упорядкованим списком ребер **77**
6.7. Приклад, який ілюструє роботу більш ефективного алгоритму з упорядкованим списком ребер **78**
6.8. Алгоритм з упорядкованим списком ребер, що використовує список активних ребер **78**
6.9. Приклад, який ілюструє роботу алгоритму впорядкованого списку ребер разом зі списком активних ребер **79**
Лекція 7. Алгоритми заповнення **82**
7.1. Алгоритми заповнення з запалом **82**
7.2. Простий алгоритм заповнення з запалом **83**
7.3. Приклад, який ілюструє роботу простого алгоритму заповнення з запалом **84**
7.4. Приклад, який ілюструє роботу алгоритму заповнення з запалом для багатокутника з діркою **85**
7.5. Порядковий алгоритм заповнення з запалом **85**
7.6. Приклад, який ілюструє роботу порядкового алгоритму заповнення з запалом **88**
Лекція 8. Основні методи усунення ступінчастості **90**
8.1. Основи методів усунення ступінчастості **90**
8.2. Простий метод усунення сходового ефекту **91**
8.3. Згортка й усунення ступінчастості **94**
8.4. Апроксимація півтонами **96**
Лекція 9. Відсічення: двовимірні, тривимірні відсічення багатокутників **100**
9.1. Відсічення **100**

9.2. Двовимірне відсічення **100**
9.3. Приклад, який ілюструє просте двовимірне відсічення **104**
9.4. Алгоритм відсічення Сазерленда - Коена, що ґрунтується на розподілі відрізка **108**
9.5. Приклад, який ілюструє роботу алгоритму відсічення Сазерленда - Коена **110**
9.6. Внутрішнє і зовнішнє відсічення **111**
9.7. Приклад, який ілюструє роботу алгоритму Кіруса - Бека **113**
9.8. Тривимірне відсічення **115**
9.9. Відсічення багатокутників **117**
9.10. Приклад, який ілюструє визначення положення точки щодо площини **120**
9.11. Алгоритм Сазерленда-Ходжмена для відсічення багатокутника **121**
Лекція 10. Видалення невидимих ліній і поверхонь **124**
10.1. Видалення невидимих ліній і поверхонь **124**
10.2. Алгоритм плаваючого горизонту **126**
10.3. Приклад роботи алгоритму плаваючого горизонту **136**
10.4. Алгоритм, що використовує z-буфер **139**
10.5. Приклад, який ілюструє роботи алгоритму, що використовує z-буфер **142**
10.6. Алгоритм, що використовує список пріоритетів **143**
10.7. Приклад, який ілюструє роботу алгоритму, що використовує список пріоритетів **146**
10.8. Приклад, який ілюструє роботу алгоритму, що використовує кластерні пріоритети **149**
10.9. Алгоритм, що використовує трасування променів **150**
10.10. Приклад, який ілюструє роботу алгоритму, що використовує трасування променів **156**
Література **160**

Вступ

Курс лекцій з дисципліни “Комп'ютерна графіка” для студентів базових напрямків “Комп'ютерні науки” та “Видавничо-поліграфічна справа” складається з окремих частин, присвячених основним розділам комп'ютерної графіки:

1. Технічні основи комп'ютерної графіки.
2. Математичні основи комп'ютерної графіки.
3. Алгоритмічні основи комп'ютерної графіки: растрова графіка.
4. Алгоритмічні основи комп'ютерної графіки: обробка зображень.
5. Алгоритмічні основи комп'ютерної графіки: відсічення.
6. Алгоритмічні основи комп'ютерної графіки: видалення невидимих ліній та поверхонь.
7. Алгоритмічні основи комп'ютерної графіки: побудова реалістичних зображень.
8. Формати графічних файлів.
9. Алгоритми стиснення зображень.
10. Комп'ютерна графіка в поліграфії та відеографіка.

В даних методичних вказівках до вивчення курсу лекцій розглядаються основні матеріали до частин дисципліни “Комп'ютерна графіка” – “Технічні основи комп'ютерної графіки”, “Математичні основи комп'ютерної графіки” та “Алгоритмічні основи комп'ютерної графіки”.

Методичні вказівки до вивчення курсу лекцій базуються на фундаментальних роботах відомих учених у галузі машинної графіки, таких як американець Роджерс Д., українці Грицик В.В. і Паленичка Р.М., росіяни Баяковський Ю.М., Ватолін Д.С., і Вельтмандер П.В., іншій літературі з питань комп'ютерної графіки та аналізу, обробки й архівування зображень.

Актуальність більшої уваги до цієї частини комп'ютерної графіки на сьогоднішній день викликана бурхливим розвитком поліграфічних послуг, рекламної справи у різних сферах життя. Зараз машинну графіку розглядають як "засіб від усіх відомих технічних і програмних хвороб", що забезпечує тісний взаємозв'язок між людиною і комп'ютером, змушуючи комп'ютер говорити з людиною мовою зображень. Машинна графіка стала основним засобом зв'язку між людиною і комп'ютером, що постійно розширює сфери свого застосування.

В методичних вказівках всі бажаючі мають можливість познайомитись з історією та методами комп'ютерної анімації, оглядом основних програм для створення і перегляду 3D графіки й анімації, поняттям віртуальної реальності, відеостандартами та відеоформатами, цифровим відео, історією та видами відеомонтажу, генераторами ландшафтів та персонажів. Наведені ілюстрації допомагають краще зрозуміти викладений матеріал. Для додаткового читання рекомендована спеціальна література з питань машинної графіки.

Вивчення цієї частини дисципліни “Комп'ютерна графіка” дає змогу професійно підійти до використання відомих програмних продуктів щодо задач обробки і генерації зображень у поліграфії та анімації, створення рекламних роликів, здійснення відеомонтажу.

Лекція 1
Історія машинної графіки. Засоби графічної взаємодії ЕОМ і людини. Класифікація і коротка характеристика

1.1. Історія машинної графіки

Комп'ютерна графіка - авангард комп'ютерних технологій - нараховує у своєму розвитку не більше десяти років. А її комерційні додатки - і того менше. Андріес ван Дам вважається одним із засновників комп'ютерної графіки, а його книги - фундаментальними підручниками з усього спектру технологій, покладених в основу машинної графіки. Також у цій області відомий Айвен Сазерленд, чия докторська дисертація стала теоретичною основою машинної графіки.

Донедавна дослідження з використання можливостей інтерактивної машинної графіки - можливостей, що здаються магічними, - було привілеєм лише невеликого кола фахівців, переважно вчених та інженерів, що займаються питаннями автоматизації проектування, аналізу даних і математичного моделювання. Тепер же дослідження реальних і уявлюваних світів через "призму" комп'ютерів стало доступним ширшому колу людей.

Така зміна ситуації зумовлена декількома причинами. Насамперед, вона стала результатом різкого поліпшення співвідношення вартість/продуктивність для деяких компонентів апаратури комп'ютерів. Крім того, стандартне програмне забезпечення високого рівня для графіки стало широкодоступним, що спрощує написання нових прикладних програм, які можна переносити з комп'ютерів одного типу на інші.

Наступна причина зумовлена впливом дисплеїв на якість інтерфейсу - засобу спілкування між людиною і машиною, - забезпечуючи максимальні зручності для користувача. Нові, зручні для користувача системи побудовані в основному на підході WYSIWYG (абревіатура від англійського виразу "What you see is what you get" - "Що бачите, те і маєте"), відповідно до якого зображення на екрані повинне бути подібним на результуючий друк.

Більшість традиційних додатків машинної графіки є двовимірними. Останнім часом, однак, відзначається зростаючий комерційний інтерес до тривимірних додатків. Він викликаний значним прогресом, який був досягнутий у минулому десятилітті у вирішенні двох взаємозалежних проблем: моделювання тривимірних сцен і побудови якнайбільш реалістичного зображення (рис.1.1).

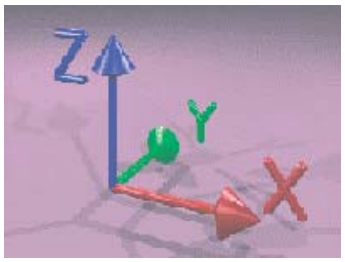


Рис.1.1. Побудова реалістичного зображення

Наприклад, в імітаторах польоту особливе значення надається часу реакції на команди, що вводяться пілотом та інструктором. Для створення ілюзії плавного руху імітатор повинен породжувати надзвичайно реалістичну картину динамічно змінного "світу" з частотою як мінімум 30 кадрів за секунду. На противагу цьому, зображення, застосовувані в рекламі та індустрії розваг, обчислюють автономно, нерідко протягом годин, з метою досягти максимального реалізму або викликати сильне враження (рис.1.2).

Рис.1.2. Генерація фотореалістичного зображення (зліва внизу фото)

1.2. Синтез зображення

Синтез зображення за допомогою комп'ютера можна представити у вигляді послідовності кроків, хоча реально ці кроки в програмі часто переплітаються (рис 1.3):

- задання об'єкта;
- видалення схованих поверхонь;
- зафарбування.

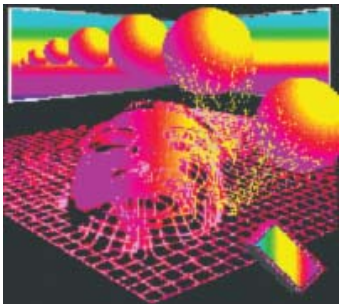


Рис.1.3. Синтез зображення за допомогою комп’ютера

Об'єкти можуть бути задані, як:

1. Двовимірні об'єкти.

Вони моделюються за допомогою таких примітивів, як:

- відрізки (задані двома кінцевими точками);
- багатокутники (визначені списком вершин і, можливо, заповнені візерунком);
- кола (задані центром, радіусом і, можливо, заповнені візерунком);
- поліноміальні криві (задані своїми коефіцієнтами).

2. Тривимірні об'єкти.

У випадку трьох вимірів відповідні примітиви визначаються шляхом додавання координати z.

Можна також ввести примітиви, що існують тільки у тривимірному просторі:

- багатогранники;
- піраміди;
- сфери;
- циліндри;
- поверхні, які описуються деякими поліноміальними функціями.

Системи моделювання тіл породжують тривимірні об'єкти, ґрунтуючись на:

1. інтерактивному заданні параметрів (при взаємодії з користувачем);
2. автономному заданні параметрів.

При автономному заданні параметри можна вносити у файли даних, створених іншою програмою, чи за допомогою текстового редактора. З іншого боку, можна скористатися процедурним описом, аналогічним до того, що застосовується для генерації фрактальних кривих і ландшафтів. Об'єкт може бути також змодельований безпосередньо, як тверде тіло, або посередньо, як об’єм, обмежений поверхнею.

У системах, побудованих на основі конструктивної геометрії суцільних тіл, об'єкти формуються з твердих тіл - примітивів, таких, як блоки, циліндри і сфери.

Примітиви можна комбінувати за допомогою тривимірних теоретико-множинних операцій:

- об'єднання (з'єднання двох об'єктів);
- перетину (виділення спільної підмножини);
- різниці (узяття всього першого об'єкту, за винятком тих його частин, що є спільними з іншим об'єктом).

Непряме задання об'єктів створюється в системах із граничним представленням. Воно також дає можливість виконувати теоретико-множинні операції, однак при цьому об'єкт визначається як обмежений плоскими гранями, циліндричними гранями чи навіть ділянками поверхні, заданими поліноміальними функціями. Такий опис поверхонь використовується аерокосмічними й автомобілебудівними компаніями.

Симетричний об'єкт можна описати за допомогою поверхні обертання. Ваза чи пляшка задаються своєю твірною (кривою, що описує силует) і віссю обертання.

Операція переносу аналогічна руху повороту: у цьому випадку об’єм формується шляхом переміщення грані довільної форми, включаючи отвори, вздовж просторової кривої.

1.3. Видалення схованих поверхонь

Схована поверхня - поверхня чи її частина, невидима з точки, де знаходиться уявна камера; поверхні, що представляють дальші сторони або поверхні об'єктів, закриті іншими, ближчими поверхнями.

Різні методи видалення невидимих поверхонь можуть бути реалізовані апаратно. В алгоритмах зазвичай передбачається, що екран розташований у проекційній площині z=0, а сцена знаходиться за ним.

Існують такі алгоритми видалення поверхонь:

- алгоритм z-буфера;

У цьому алгоритмі використовується окремий буфер значень координати z, по одному значенню на кожен мікшер. Величина z піксела містить інформацію про глибину відповідної точки, що проектується на мікшер і належить найближчому з багатокутників, що зустрічалися до даного моменту. Коли новий багатокутник перетворюється, відсікається і проектується на площину z=0, значення пікселів, що утворюють багатокутник, по черзі порівнюються із значеннями, записаними в z-буфері. Якщо значення z піксела певного багатокутника виявляється меншим (це означає, що такий багатокутник розташований ближче до екрану, ніж кожний з раніше розглянутих багатокутників), то піксел вважається біжучим "видимим" пікселом і запам'ятовується як у буфері регенерації, так і в z-буфері. Реально ж він стане видимим тільки в тому випадку, якщо не заміниться до того часу, поки не буде розглянутий останній багатокутник.

- алгоритм "художника".

На відміну від алгоритму z-буфера, що може обробляти багатокутники, які надходять у довільному порядку, в алгоритмі "художника" багатокутники спочатку впорядковуються в напрямку від заднього плану до переднього. У випадку, коли пари багатокутників не вдається досить просто упорядкувати, вони підрозділяються на частини доти, доки утворені частини не дозволять це зробити. Потім багатокутники проектуються і "розфарбовуються" у буфері кадрів у порядку від заднього плану до переднього так, щоб багатокутники, які знаходяться ближче до точки спостереження, правильно закривали віддалені багатокутники і при цьому не потребували додаткових обчислень.

1.4. Зафарбування

У моделі зафарбування потрібно брати до уваги як властивості поверхні (її колір, фактуру і відбивну здатність), так і відносне місце розташування, орієнтацію і властивості джерел світла та інших поверхонь. У моделях освітлення для світлових джерел можуть враховуватися такі варіанти, як розсіяне світло, точкові джерела (яскраве сонце чи палаюча лампа) чи розподілені джерела (вікно чи ряд флуоресцентних трубок).

Розсіяне світло моделюється шляхом додавання постійної величини світлової інтенсивності до всіх поверхонь, однак, при такій стратегії не проводиться ніякої диференціації між поверхнями. Відображення точкових джерел світла матовими чи дифузійними поверхнями (тобто такими, котрі розсіюють світло рівномірно в усіх напрямках) описується законом косинусів Ламберта. Закон Ламберта: інтенсивність відбитого світла пропорційна до косинуса кута між напрямком на джерело світла і вектором, перпендикулярним до поверхні, чи нормаллю до цієї поверхні.

Освітленість виявляється найсильнішою, коли поверхня перпендикулярна до напрямку на світлове джерело. Для блискучих

поверхонь, що породжують дзеркальні відблиски (такими поверхнями є добре відполіроване дерево чи метал), кількість відбитого світла визначається кутами між нормаллю до поверхні і двома напрямками: на світлове джерело і на точку спостереження. Поверхня працює як дзеркало, тобто відбиває більшу частину світла, коли кути майже рівні (тобто коли точка зору і джерело світла розташовані симетрично щодо нормалі до поверхні). Сума складових розсіяного, дифузійного і дзеркально відбитого світла визначає інтенсивність свічення поверхні. Якщо ж береться до уваги світло, то розглядаються рівняння для кожного з трьох основних кольорів.

У результаті такої комбінації операцій з'являється неприроднє, з великою кількістю граней, зображення. Оскільки багатокутник визначається однією нормаллю до поверхні, сусідні багатокутники з різними нормалями мають різні значення інтенсивності. Це призводить до помітного спотворення неперервності на спільних ребрах.

Зафарбування Гуро дозволяє усереднити інтенсивність на вершинах багатокутників, а потім уздовж скануючих прямих, і цим самим згладити їх.

Зафарбування Фонга є досконалішим, ніж зафарбування Гуро, тому що використовує детальніші обчислення, набагато чутливіші до ефектів, залежних від напрямку дзеркальних відблисків.

Інші ефекти, що доводиться враховувати, - це тіні, прозорість і властивості поверхні, такі як фактура і зернистість.

Алгоритми побудови тіней від джерел світла нагадують алгоритми видалення схованих поверхонь, оскільки в них визначається, які поверхні можуть бути "видимі" з точок, де розташовані джерела світла. Поверхні, одночасно видимі з точки спостереження і з джерел світла, не знаходяться в тіні, в той час як ті з них, що видимі з точки спостереження, але не видимі зі світлового джерела - затінені. У випадку розподілених джерел світла потрібні складні обчислення, як для повних тіней, так і для напівтіней.

Ще складнішу проблему створює фактор прозорості. "Спрямоване" проходження, характерне для прозорих поверхонь, подібних до скла, визначається коефіцієнтом заломлення речовини. Дифузійне проходження світла через напівпрозорі матеріали, такі як замерзле скло, розсіює світло в усіх напрямках.

Алгоритм трасування променів дозволяє будувати найреалістичніші зображення з урахуванням як відбиття, так і заломлення світла. У таких алгоритмах відслідковуються індивідуальні світлові промені, щоб визначити, які з них закінчуються в точці спостереження і як вони там виявляються.

Для того, щоб не мати справи з нескінченним числом променів, що виходять із джерела світла, їх прослідковують у зворотному напрямку, з початком у кожному пікселі. Кожен промінь, що виходить із точки спостереження і проходить через піксел, проектується назад до перетину з поверхнею. Потім зворотнє трасування відбитого променя продовжується, щоб визначити, чи прийшов цей промінь від джерела світла, чи він є результатом відбиття від іншого об'єкта.

У випадку прозорої поверхні повинен бути відслідкований і другий,

заломлений промінь. Кожен промінь необхідно перевірити на перетин з кожним об'єктом (рис.1.4).

Рис.1.4. Зображення з трасуванням променів

Для формування фактури поверхні можуть використовуватися різні моделі, що вносять локальні нерегулярності. Для відображення двовимірного візерунка на поверхню можна скористатися візерунком, утвореним значеннями інтенсивності, щоб промоделювати інтенсивності, обчислені за допомогою алгоритмів зафарбування та затінення.

1.5. Приклад, який ілюструє синтез зображення

Об'єкт (крокодил) задається сіткою багатокутників і зображується спочатку у вигляді "каркасного" малюнка. Потім викидаються невидимі ребра.

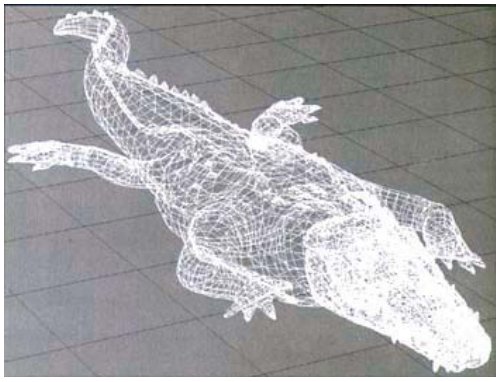


Рис.1.5.

На наступному етапі проводиться зафарбування (тут зафарбування кольором) кожного з багатокутників, що є функцією кута, утвореного багатокутником з джерелами світла, і властивостей його поверхні; у результаті виходить зображення з великою кількістю граней, яке має неприродний вигляд.

Порушення неперервності на спільних ребрах між сусідніми багатокутниками можна згладити за допомогою зафарбування Гуро, а дзеркальні світлові відблиски можна додати, якщо скористатися зафарбуванням Фонга (рис.1.6).

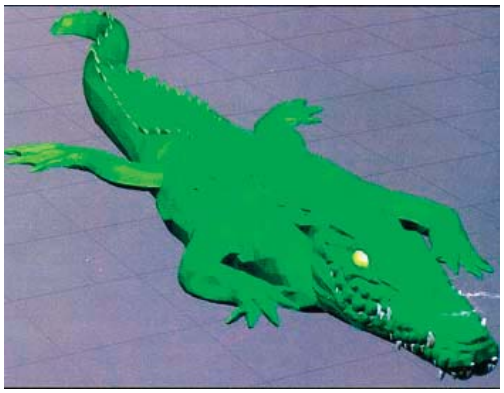


Рис.1.6. Зображення після застосування зафарбування

На останньому етапі методом вирівнювання згладжуються зазубрення країв при раструванні (рис.1.7).

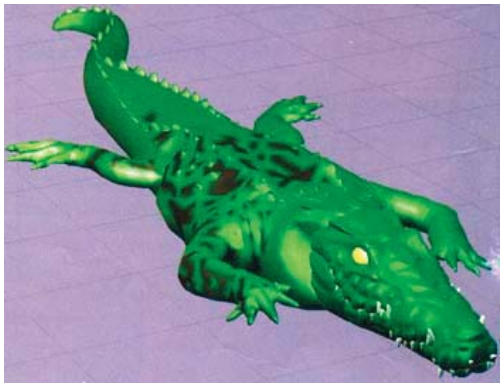


Рис.1.7. Зображення після застосування методу вирівнювання

Фінальне зображення - алгоритм трасування променів прораховує тіні і відбиття (рис.1.8).

Рис.1.8. Зображення після застосування алгоритму трасування променів

Лекція 2
Апаратні засоби комп’ютерної графіки

2.1. Дисплеї (монітори)

Однією з найважливіших складових частин персонального комп'ютера є його відеопідсистема, що складається з монітора та відеоадаптера (звичайно розташованого на системній платі).

Монітор призначений для відображення на екрані текстової і графічної інформації для її візуального сприйняття користувачем персонального комп'ютера. Зараз існує велика різноманітність типів моніторів. Їх можна охарактеризувати за наступними основними ознаками:

- Режим відображення:
  1. Растрові дисплеї;
  2. Векторні дисплеї.

У векторних дисплеях з регенерацією зображення на базі електронно-променевої трубки (ЕПТ) використовується люмінофор з дуже коротким часом післясвічення. Такі дисплеї часто називають дисплеями з довільним скануванням. Через те, що час післясвічення люмінофора малий, зображення на ЕПТ за секунду повинне багаторазово перемалюватися чи регенеруватися. Мінімальна швидкість регенерації повинна складати, принаймні, 30 (1/с), а краще 40-50 (1/с). Швидкість регенерації менша 30 приводить до мерехтіння зображення.

Крім ЕПТ, для векторного дисплея необхідні дисплейний буфер і дисплейний контролер. Дисплейний буфер - безперервна ділянка пам'яті, що містить усю інформацію, необхідну для виведення зображення на ЕПТ. Функція дисплейного контролера полягає в тому, щоб циклічно обробляти цю інформацію зі швидкістю регенерації. Складність малюнка обмежується двома факторами - розміром дисплейного буфера і швидкістю контролера.

На рис.2.1 зображені блок-схеми двох високопродуктивних векторних дисплеїв. В обох випадках передбачається, що такі геометричні перетворення, як поворот, перенесення, масштабування, перспективне проектування і відсікання, реалізовані апаратно в геометричному процесорі.

Рис.2.1. Блок-схеми векторних дисплеїв з регенерацією

У першому випадку геометричний процесор працює повільніше, ніж це необхідно при регенерації зображення. Геометричні дані, що посилаються ЦПУ графічному дисплею, обробляються до збереження в дисплейному буфері. Виходить, у ньому містяться тільки ті інструкції, які необхідні генератору для виводу зображень. Контролер зчитує інформацію з дисплейного буфера і посилає генератору. При досягненні кінця

дисплейного буфера контролер повертається на його початок, і цикл повторюється знову.

При використанні першої схеми виникає ідея подвійної буферизації та роздільної зміни зображення і його регенерації. Оскільки в цій конфігурації геометричний процесор не встигає згенерувати складне нове чи змінене зображення під час одного циклу регенерації, то дисплейний буфер поділяється на дві частини. У той час як змінене зображення обробляється і записується в одну половину буфера, дисплейний контролер регенерує ЕПТ з іншої половини буфера. Після завершення зміни зображення буфери міняються ролями, і цей процес повторюється. Отже, нове чи змінене зображення може генеруватися кожен другий, третій, четвертий і т.д. цикли регенерації. Використання подвійної буферизації запобігає одночасному виведенню частини старого і частини нового зміненого зображення протягом одного і більше циклів регенерації.

У другій схемі геометричний процесор працює швидше, ніж необхідно для регенерації досить складних зображень. У цьому випадку вихідна геометрична база даних, послана з ЦПУ, зберігається безпосередньо в дисплейному буфері, а вектори звичайно задаються в користувацьких координатах у вигляді чисел із плаваючою точкою. Дисплейний контролер за один цикл регенерації зчитує інформацію з дисплейного буфера, пропускає її через геометричний процесор і результат передає генератору векторів. При такому способі обробки геометричні перетворення повинні виконуватися "на льоту" протягом одного циклу регенерації.

Растровий пристрій можна розглядати як матрицю дискретних комірок (точок), кожна з яких може бути підсвічена. Отже, він є точково-малюючим пристроєм. Неможливо, за винятком спеціальних випадків, безпосередньо намалювати відрізок прямої з однієї адресуючої точки чи піксела в матриці в другу адресуючу точку. Відрізок можна тільки апроксимувати послідовностями точок (пікселів), які лежать близько до реальної траєкторії відрізка. Цю ідею ілюструє рис.2.2.

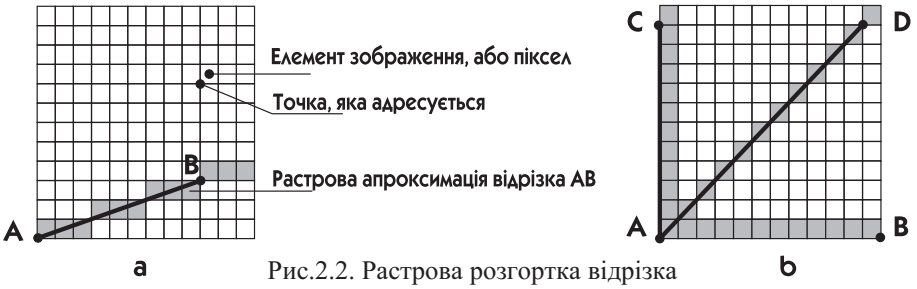


Рис.2.2. Растрова розгортка відрізка

Відрізок прямої з точок вийде тільки у випадку горизонтальних, вертикальних чи розташованих під кутом 45^0 відрізків. Всі інші відрізки будуть виглядати як послідовності сходинок. Це явище називається сходовим чи ефектом "зазубреності".

Найчастіше для графічних пристроїв з растровою ЕПТ використовується буфер кадру. Буфер кадру являє собою велику

безперервну ділянку пам'яті комп'ютера. Для кожної точки чи піксела в растрі виділяється як мінімум один біт пам'яті. Ця пам'ять називається бітовою площиною. Для квадратного растра розміром 512×512 потрібно 2^{18}, чи 262144 біти пам'яті в одній бітовій площині. Через те, що біт пам'яті має тільки два стани (двійковий 0 чи 1), маючи одну бітову площину, можна одержати лише чорно-біле зображення.

Бітова площина є цифровим пристроєм, тоді як растрова ЕПТ - аналоговий пристрій. Тому при зчитуванні інформації з буфера кадру та її виведенні на графічний пристрій з растровою ЕПТ повинне відбуватися перетворення з цифрового представлення в аналоговий сигнал. Таке перетворення виконує цифро-аналоговий перетворювач (ЦАП). На рис.2.3 наведена схема графічного пристрою з чорно-білою растровою ЕПТ, побудованого на основі буфера кадру з однією бітовою площиною.

Рис. 2.3. Чорно-білий буфер кадру (з одною бітовою площиною) для растрового дисплея

Кольори чи напівтони сірого кольору можуть бути введені в буфер кадру шляхом використання додаткових бітових площин. На рис.2.4 показана схема буфера кадру з N бітовими площинами.

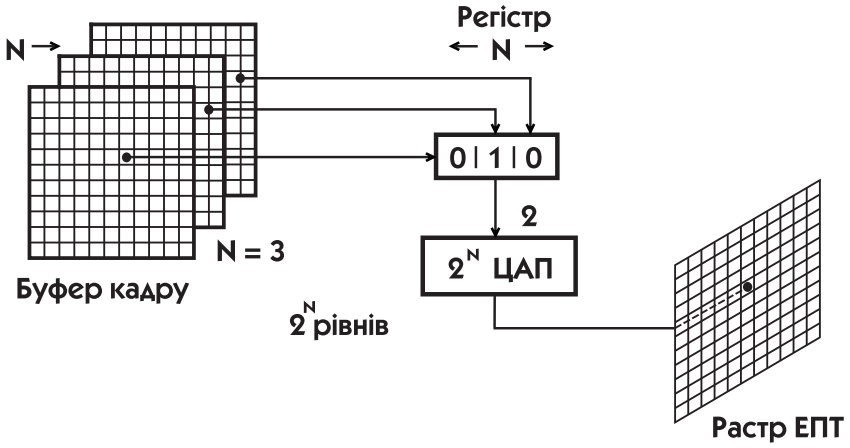


Рис. 2.4. Півтоновий чорно-білий буфер кадру з N площинами

Інтенсивність кожного піксела на ЕПТ керується вмістом відповідних пікселів у кожній з N бітових площин. У відповідну позицію регістра завантажується бінарна величина (0 чи 1) з кожної площини. Двійкове число, одержане в результаті, інтерпретується як рівень інтенсивності між 0

і 2^N-1. Буфер кадру з трьома бітовими площинами для растра 512×512 займає 786432 (3*512*512) бітів пам'яті.

Число доступних рівнів інтенсивності можна збільшити, скориставшись таблицею кольорів, як схематично показано на рис.2.5.

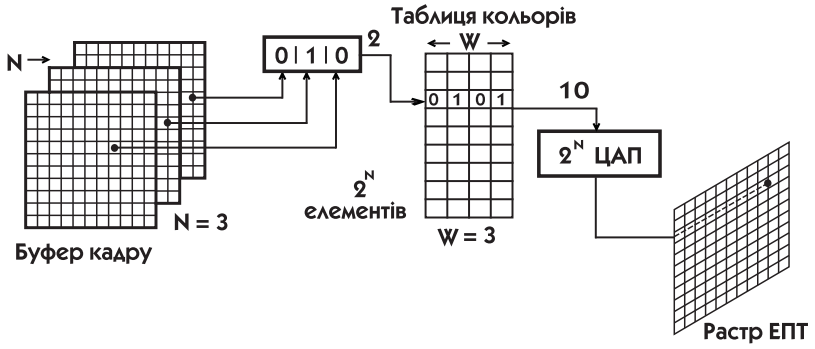


Рис. 2.5. Півтоновий чорно-білий буфер кадру з N бітовими площинами і W-розрядною таблицею кольорів

Після зчитування з буфера кадру бітових площин число, що вийшло - 2^N, використовується як індекс у таблиці кольорів, кожен елемент якої може містити W біт, причому W може бути більше N.

Оскільки існує три основних кольори, можна реалізувати простий кольоровий буфер кадру з трьома бітовими площинами, по одній для кожного з основних кольорів. Кожна бітова площина керує індивідуальною електронною гарматою для кожного з трьох основних кольорів. Три основних кольори, комбінуючись на ЕПТ, дають вісім кольорів. Схема простого кольорового растрового буфера показана на рис.2.6.

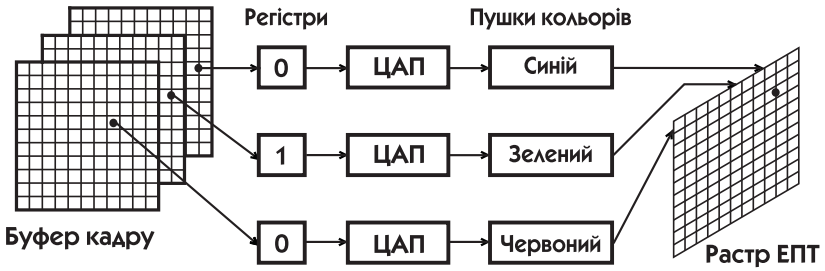


Рис. 2.6. Простий кольоровий буфер кадру

Щоб збільшити кількість кольорів для кожної з трьох колірних гармат, використовуються додаткові бітові площини.

- Тип екрану:
  1. дисплеї на основі ЕПТ;
  2. рідкокристалічні (РК);
  3. плазменні.

Для розуміння принципів роботи растрових дисплеїв і векторних дисплеїв з регенерацією розглянемо уявлення про конструкцію ЕПТ і

методи створення відеозображення. На рис.2.7 показана ЕПТ, яка використовується у відеомоніторах.

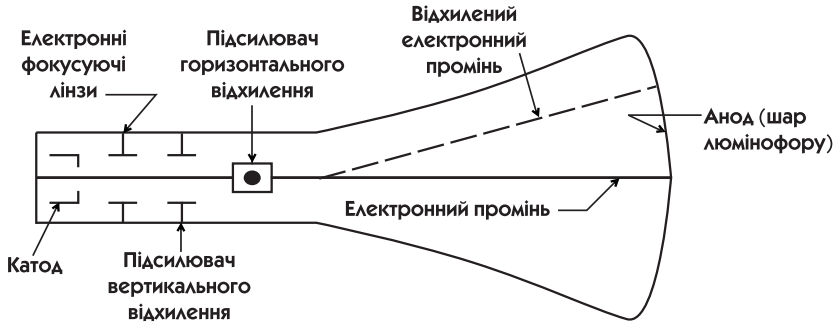


Рис. 2.7. Електронно-променева трубка

Катод (негативно заряджений) нагрівають доти, доки збуджені електрони не створять хмари, що розширюються, (електрони відштовхуються один від одного, тому що мають однаковий заряд). Ці електрони притягуються до сильно зарядженого позитивного анода. На внутрішню сторону розширеного кінця ЕПТ нанесений люмінофор. Хмара електронів за допомогою лінз фокусується у вузький, строго паралельний пучок, і промінь дає яскраву пляму в центрі ЕПТ. Промінь відхиляється чи позиціонується вліво або вправо від центра і (або) вище чи нижче від нього за допомогою підсилювачів горизонтального і вертикального відхилення. Саме в цей момент виявляється відмінність векторних і растрових дисплеїв. У векторному дисплеї електронний промінь може бути відхилений безпосередньо з будь-якої довільної позиції в будь-яку іншу довільну позицію на екрані ЕПТ (аноді). Оскільки люмінофорне покриття нанесене на екран ЕПТ суцільним шаром, у результаті виходить майже ідеальна пряма. На відміну від цього в растровому дисплеї промінь може відхилятися тільки в строго визначені позиції на екрані, що утворюють своєрідну мозаїку. Ця мозаїка складає відеозображення. Люмінофорне покриття на екрані растрової ЕПТ теж не безперервне, а являє собою безліч тісно розташованих дрібних точок, куди може позиціонуватись промінь, утворюючи мозаїку.

Екран рідкокристалічного дисплея (РКД) складається з двох скляних пластин, між якими знаходиться маса, що містить рідкі кристали, які змінюють свої оптичні властивості в залежності від прикладеного електричного заряду. Рідкі кристали самі не світяться, тому РКД мають потребу в підсвічуванні чи в зовнішньому освітленні. Основною перевагою РКД є їхні габарити (екран плоский). До недоліків можна віднести недостатню швидкодію при зміні зображення на екрані, що особливо помітно при переміщенні курсору “миші”, а також залежність різкості та яскравості зображення від кута зору.

РК-дисплеї мають безперечні переваги перед конкуруючими пристроями відображення:

1. Розміри. РК-дисплеї відрізняються малою глибиною і невеликою масою, тому їх зручніше переміщати і встановлювати, ніж ЕПТ-монітори, в яких розмір у глибину приблизно дорівнює ширині.
2. Енергоспоживання. РК-дисплей споживає меншу потужність, ніж ЕПТ-монітор з такими самими характеристиками.
3. Зручність для користувача. У ЕПТ електронні промені при розгорненні рухаються екраном, відновлюючи зображення. Хоча в більшості випадків можна встановити таку частоту регенерації (кількість відновлень екрану електронними променями в секунду), що зображення виглядає стабільним, деякі користувачі все-таки сприймають мерехтіння, здатне викликати швидку втому очей і головний біль. На екрані РК-дисплея кожен піксел або ввімкнений, або вимкнений, так що мерехтіння відсутнє. Для ЕПТ-моніторів характерне невелике електромагнітне випромінювання; у РК-моніторах воно відсутнє. Недолік - висока ціна.

Ще одне досягнення, завдяки якому незабаром може відбутися зниження цін, - вдосконалення технології панелей на супертвістованих нематичних кристалах (dual supertwist nematic, DSTN). DSTN-дисплеї завжди були дешевші, ніж РК-пристрої на тонкоплівкових транзисторах (thin-film transistors, TFT), але трохи поступалися їм за якістю: DSTN-дисплеї не забезпечують такої контрастності і чіткості, як матриці TFT, а їхня повільна реакція призводить до мерехтіння і появи паразитних (повторних) зображень на екрані, особливо при відображенні рухомих об'єктів. Однак фірма Sharp, найбільший постачальник DSTN-панелей, недавно провела презентацію панелі, у якій використовується розроблена нею технологія HCA (High-Contrast Addressing - висококонтрастна адресація).

HCA-панелі забезпечують таку саму контрастність зображення, як TFT-матриці, і майже не поступаються їм у швидкості реакції при відтворенні відео. Фірма Arithmos розробила процесор візуалізації для DSTN-панелей, що дозволяє ще більше поліпшити якість зображення. Таким чином, для користувачів з обмеженими коштами DSTN-дисплей може виявитися чудовим компромісним рішенням.

У РК-дисплеях кут огляду не тільки малий, але й асиметричний: звичайно він складає 45° по горизонталі і +15...-30 по вертикалі. Випромінюючі дисплеї, такі як електролюмінесцентні, плазменні і на базі ЕПТ, як правило, мають конус огляду від 80° до 90° по обох осях.

Представник Hitachi Тім Паттон вважає, що в традиційних РК-дисплеях спостерігається залежність контрастності та кольору зображення від кута зору. Ця проблема загострювалася в міру збільшення розмірів РК-дисплеїв і набуття ними здатності відтворювати більше кольорів.

Hitachi при створенні свого нового дисплея SuperTFT скористалася іншою технологією - IPS. Як відомо, у звичайних РК-дисплеях молекули рідкого кристала змінюють свою орієнтацію з горизонтальної на вертикальну під впливом електричного поля, а адресуючі електроди містяться на двох розташованих одна навпроти одної скляних підкладках. У

IPS(in-plane switching)-дисплеях, навпаки, відбувається чергування двох кутів у горизонтальній площині, причому обидва електроди знаходяться на одній з підкладок. У результаті кут огляду як по горизонтальній, так і по вертикальній осях досягає 70°.

Газоплазменні монітори складаються з двох пластин, між якими знаходиться газова суміш, що світиться під впливом електричних імпульсів. Такі монітори не мають недоліків, властивих РКД, однак їх не можна використовувати в переносних комп'ютерах з акумуляторним і батарейним живленням, тому що вони споживають великий струм.

В загальному, більшість моніторів мають такі характеристики:

- розмір по діагоналі (звичайно від 14" до 21") і розмір зерна (звичайно від 0.25 до 0.31 мм). Розмір по діагоналі (відстань від лівого нижнього до правого верхнього кута екрану) задається в дюймах. Найбільш поширені монітори з діагоналлю 15". Однак працювати за монітором з діагоналлю 17" набагато зручніше, а для роботи з графічними пакетами, видавничими системами і системами автоматизованого проектування необхідні монітори з діагоналлю не менше 19";

- тіньова маска (Dot Pitch) екрану. Одиницею виміру є відстань між отворами маски в мм. Чим менша ця відстань і чим більше отворів, тим вища якість зображення. Цей параметр часто ототожнюють із зерном екрану монітора, однак це справедливо не в усіх випадках;

- роздільна здатність. Виміряється в пікселах (точках), що містяться по горизонталі і вертикалі видимої частини екрана. Зараз найбільш поширені монітори з роздільною здатністю не менше 1024*768 пікселів;

- кінескоп. Найкращі наступні типи кінескопів: Black Trinitron, Black Matrix і Black Planar. Ці кінескопи дуже контрастні, дають відмінне зображення, однак їх люмінофор чутливий до світла, що може скоротити термін дії монітора. До того ж, при роботі з контрастним монітором швидше стомлюються очі;

- потужність споживання. У моніторів з діагоналлю 14" потужність споживання не повинна перевищувати 60 Вт, інакше підвищується ймовірність теплового перегріву монітора, що скорочує термін його дії. У більших моніторів потужність споживання відповідно вища;

- антиблікове покриття. Для дешевих моніторів використовують піскоструменеву обробку поверхні екрану. При цьому якість зображення погіршується. У дорогих моніторах на поверхню екрана наноситься спеціальна хімічна речовина, що має антиблікові властивості; - захисні властивості монітора;

- рівень випромінювання. В даний час поширені монітори з низьким рівнем випромінювання (LR-монітори). Вони відповідають нормам стандарту MPRI чи MPR II;

- кольоровість:

1. кольорові;
2. монохромні;

- частота кадрів (звичайно від 50 до 100 Гц). Сучасні аналогові монітори за частотою кадрів умовно можна розділити на такі основні типи:

1. з фіксованою частотою розгортки;
2. з декількома фіксованими частотами;
3. багаточастотні (мультичастотні).

Мультичастотні монітори мають здатність налаштовуватися на довільні значення частот синхронізації з деякого заданого діапазону, наприклад, 30-64 кГц для рядкової і 50-100 Гц для кадрової розгортки. Розробником моніторів такого типу є багато фірм, наприклад, фірма NEC. У назві таких моніторів присутнє слово Multisync. Ці монітори відносяться до найбільш розповсюдженого типу моніторів з електронно-променевою трубкою.

Відеодіапазон (звичайно від 65 до 200 МГц).

Відеосигнал:

1. цифровий;
2. аналоговий.

До цифрових моніторів належать пристрої відображення зорової інформації на основі електронно-променевої трубки, керованої цифровими схемами. До цифрових відносяться монохромні монітори, забезпечені відеоадаптерами стандартів MDA і Hercules, кольорові RGB-монітори, призначені для підключення до відеоадаптера стандарту EGA. Монохромні монітори здатні відображати на екрані тільки темні та світлі точки, іноді точки можуть відрізнятися інтенсивністю. Hercules-монітори мають роздільну здатність до 728*348 пікселів, невеликі габарити і вагу. Блок розгортки монітора одержує синхроімпульси від відповідного відеоадаптера. RGB-монітори здатні відображати 16 кольорів, однак роздільна здатність екрану в них менша, ніж у Hercules-моніторів.

Електронно-променева трубка моніторів даного типу керується аналоговими сигналами, що надходять від відеоадаптера. Принцип роботи електронно-променевої трубки монітора такий самий, як у телевізійної трубки. Аналогові монітори здатні підтримувати роздільну здатність стандарту VGA (640*480) пікселів і вище.

Інші характеристики (функції керування растром, яскравістю та контрастом, система енергозбереження, захист від випромінювання, вага, габарити).

2.2 Світлове перо (light pen)

Це світлочутливий пристрій, призначений для зняття координат точок екрану, введення даних в інформаційну систему.

Світлове перо за формою нагадує ручку. В наконечнику пера встановлюється фотоелемент, який реагує на світловий сигнал, переданий екраном у точці дотику пера, і момент цієї реакції повідомляється системі. Тут зіставляється час появи сигналу з синхросигналом розгортки зображення. В результаті визначається положення світлового пера на екрані. Світлове перо не вимагає створення спеціального екрана чи його покриття, як у сенсорного пристрою. Сказане дозволяє виділяти точку, що вказується користувачем, і завдяки цьому вводити інформацію в систему.

Таким чином, можна записати і потім здійснити розпізнавання рукописного тексту, намалювати малюнок.

2.3. Маніпулятор “миша”

У сучасних програмних продуктах, що мають графічну оболонку, “миша” є основним інструментом керування програмою (рис.2.8).

Рис.2.8. Комп’ютерна “миша”

За принципом дії “миші” поділяються на:

1. механічні;
2. оптико-механічні;
3. оптичні.

Багато комп'ютерних “мишок” використовують оптико-механічний принцип кодування переміщення. З поверхнею столу стикається важка, вкрита гумою кулька порівняно великого діаметру. Ролики, притиснуті до поверхні кульки, встановлені на перпендикулярних одна до одної осях з двома датчиками. Датчики, що являють собою оптопари (світлодіод-фотодіод), розташовуються по різні боки дисків із прорізами. Порядок, у якому висвітлюються фоточутливі елементи, визначає напрямок переміщення “миші”, а частота імпульсів, які приходять від них - швидкість. Добрий контакт із поверхнею забезпечує спеціальний килимок.

Більш точного позиціонування курсору дозволяє домогтися оптична “миша”. Для неї використовується спеціальний килимок, на поверхні якого нанесена дрібна сітка з перпендикулярних одна до одної темних і світлих смуг. Розташовані в нижній частині “миші” дві оптопари освітлюють килимок і за кількістю перетятих при русі ліній визначають величину і швидкість переміщення. Оптичні “миші” не мають рухомих частин і позбавлені такого властивого оптико-механічним “мишам” недоліку, як переміщення курсору “миші” ривками через забруднення кульки. Роздільна здатність застосовуваного в “миші” пристрою зчитування координат складає 400 dpi (dot per inch точок на дюйм) і вище, перевершуючи аналогічні значення для механічних пристроїв.

Для оптимального функціонування “миша” повинна пересуватися по рівній поверхні. Найкраще підходять спеціальні килимки (mouse pad). Покажчик “миші” пересувається екраном синхронно з рухом “миші” по килимку. Пристроєм введення “миші” є кнопки (клавіші). Більшість “мишей” мають дві та більшу кількість кнопок.

Однією з важливих характеристик “миші” є її роздільна здатність, що вимірюється в dpi. Роздільна здатність визначає мінімальне переміщення, яке здатний відчути контролер “миші”. Чим більша роздільна здатність, тим точніше позиціонується “миша”, тим з дрібнішими об'єктами можна працювати. Нормальна роздільна здатність “миші” лежить у діапазоні від 300 до 900 dpi. В удосконалених “мишах” використовують змінний балістичний ефект швидкості, який полягає в тому, що при невеликих

переміщеннях швидкість зсуву курсору невелика, а при значних переміщеннях - істотно збільшується. Це дозволяє ефективніше працювати в графічних пакетах де доводиться обробляти дрібні деталі.

За принципом передачі інформації "миші" поділяються на:

1. Послідовні (serial mouse), що підключаються до послідовного порту COM1 чи COM2;

2. Паралельні (bus mouse), що використовують системну шину і підключаються до спеціальної карти розширення.

Паралельні "миші" використовують в системах, в яких до комп'ютера потрібно підключити багато периферійних пристроїв, що займають послідовні порти, і де комп'ютер піддається конфліктам переривань периферійних пристроїв (bus mouse не використовує переривання).

Існує декілька стандартів послідовних "мишей". Найпоширенішим є стандарт MS-mouse. Альтернативними стандартами є PC-mouse, який використовується для трикнопкових "мишей" фірми Genius, і PS/2, який використовується рідко. MS-mouse і сумісні з нею PC-mouse для роботи вимагають встановлення відповідних драйверів. Більшість програмних забезпечень для персональних комп'ютерів орієнтована на MS-mouse. Стандарт PS/2 не вимагає підключення драйверів.

До основних тенденцій розвитку сучасних "мишей" можна віднести поступовий перехід на шину USB, а також пошуки в області ергономічних вдосконалень. До них можна віднести безпровідні (Cordless) "миші", що працюють у радіо- чи інфрачервоному діапазоні хвиль, а також "миші" з додатковими кнопками. Найбільш вдалими рішеннями є наявність між двома стандартними кнопками коліщати ("миша" Microsoft IntelliMouse) чи середньої кнопки, яка коливається ("миші" Genius NetMouse NetMouse Pro), що використовуються для прокручування документа під Windows (рис.2.9).

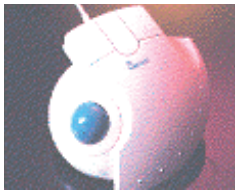


Рис.2.9. "Миша" Genius NetMouse Pro

2.4. Джойстики

Джойстик є координатним пристроєм вводу інформації і найчастіше застосовується в області комп'ютерних ігор та комп'ютерних тренажерів.

Джойстики бувають:

1. аналогові (використовуються в комп'ютерних тренажерах);

2. цифрові (в ігрових комп'ютерах).

Аналогові джойстики забезпечують більш точне керування, що дуже важливо для програмних додатків, у яких об'єкти повинні точно позиціонуватись. Для зручності роботи конструкція джойстика повинна бути досить міцною і стійкою. Джойстик підключають до зовнішнього роз'єму карти розширення, що має відповідний (ігровий) порт (рис.2.10).

Рис.2.10. Джойстик

Ігровий порт (чи адаптер) може бути розташований на платі асинхронного послідовного адаптера, на платі мультипорта чи на окремій платі, іноді на системній платі комп'ютера.

До одного ігрового порту може бути підключено два джойстики. Для перевірки наявності ігрового порту можна скористатися програмою System Information з пакета Norton Utilities чи програмою Microsoft Diagnostic.

Робота з джойстиком не вимагає підключення додаткового драйвера. Досить налагодити ігрову програму на його використання. Зауважимо, що далеко не всі ігри можуть працювати з джойстиком. Звичайно це емулятори польоту на літаках, вертольотах і космічних кораблях.

2.5. Трекбол (Trackball)

Трекбол (Trackball) - це пристрій вводу інформації у вигляді перевернено̂ї “миші” з кулькою. Принцип дії і спосіб передачі даних трекбола такий самий, як і “миші”. Найчастіше використовується оптико-механічний принцип реєстрації положення кульки. Підключення трекбола, як правило, здійснюється через послідовний порт (рис.2.11).

Рис.2.11. Трекбол

Основні відмінності від “миші”:

- стабільність положення за рахунок нерухомого корпусу;
- не потрібна площадка для руху, тому що позиція курсору розраховується за обертанням кульки.

Перший трекбол був розроблений компанією Logitech. Трекболи набули спочатку поширення в ноутбуках. Вбудовані трекболи можуть розташовуватися у довільному місці корпусу ноутбука, зовнішні кріпляться спеціальним затискачем, а до інтерфейсу підключаються кабелем. Недоліком їх у ноутбуках було поступове забруднення поверхні кулі та роликів і, отже, втрата точності. Згодом їх замінили тачпади і трекпойнти.

2.6. Дигитайзери

Дигитайзер призначений для професійних графічних робіт. За допомогою спеціального програмного забезпечення він дозволяє перетворювати рух руки оператора в формат векторної графіки. Спочатку дигитайзер був розроблений для додатків систем автоматизованого проектування, оскільки в цьому випадку необхідно визначати і задавати точне значення координат великої кількості точок. На відміну від миші дигитайзер здатний точно визначати і обробляти абсолютні координати.

Дигитайзер складається зі спеціального планшета, що є робочою поверхнею і, крім цього, виконує різноманітні функції управління відповідним програмним забезпеченням, і світлового пера або частіше кругового курсора, що є пристроями вводу інформації (рис.2.12).

Рис.2.12. Дигитайзер

Одним з різновидів дигитайзера є графічний або рисувальний планшет. Він являє собою панель, під якою розташовані електромагнітні гратки. Якщо провести по його поверхні спеціальним пером, то на екрані монітора з'явиться штрих. У планшеті реалізований принцип абсолютного позиціонування: зображення, намальоване в лівому нижньому куті планшета, з'явиться в лівому нижньому кутку екрану монітора. Зазвичай малювальні планшети мають розміри килимка для миші, але робоча поверхня трохи менша.

Існують планшети, чутливі до натиску, за допомогою яких, регулюючи натиск, можна отримувати на екрані лінії різної товщини.

Спеціальна пластмасова плівка, прикладена до планшета, дозволяє копіювати підкладені під неї зображення на паперових оригіналах. Планшети підключаються до послідовного порту персонального комп'ютера. Графічний планшет може мати різні формати: від А2 - для професійної діяльності і менше - для простіших робіт.

Сканування фотографій або інших плоских документів - справа порівняно проста: оригінал кладеться на скляну пластину планшетного сканера, закривається кришка і проводиться запуск апарату. Але сканування в трьох вимірюваннях, що визначають наш світ, набагато складніше і вимагає великої праці, тому досі ця задача для користувачів ПК була майже нерозв'язна. Дигитайзер являє собою настільний пристрій, здатний відображати об'єкти невеликих і середніх розмірів у вигляді точних тривимірних файлів.

Прикладом може служити прилад для перецифрування тривимірних об'єктів MicroScribe-3D компанії Immersion (рис.2.13).

Рис.2.13. Прилад для перецифрування тривимірних об’єктів

Компанія Immersion розробила унікальну механічну технологію перецифрування, компактну, доступну і легку в використанні. Кожне з'єднання використовує цифрові оптичні датчики, робота яких не залежить від будь-якого впливу, що відноситься до оточення. Результат - універсальна система, яка може працювати практично в будь-якому середовищі і сканувати об'єкти з будь-якого матеріалу.

Але крім цього, є також інші технології тривимірного сканування:

1. Ультразвукове сканування.
2. Магнітне сканування.
3. Лазерні сканери.

З усіх тривимірних технологій сканування ультразвукові системи найменш точні, найменш надійні і найбільш сприйнятливі до геометричних спотворень. Внаслідок того, що швидкість звуку залежить від атмосферних умов, ефективність ультразвукових систем може змінюватися разом з погодою. Крім того, вони сприйнятливі до роботи обладнання.

Магнітні тривимірні цифрові перетворювачі використовують магнітне поле. Вони несприйнятливі до атмосферних змін і дуже чутливі до спотворень від сусіднього металу або магнітних полів. Металеві стільці, плати, комп'ютери або інше обладнання, розміщені близько від магнітного цифрового перетворювача, спотворюють дані. Крім того, такі системи не можна використати для перецифрування об'єктів з металевими частинами.

Лазерні сканери в 10-100 раз дорожчі, ніж системи механічного перецифрування. Лазерні системи мають багато обмежень: об'єкти з відображаючими або яскравими поверхнями, великі об'єкти і об'єкти з увігнутими поверхнями, які затіняють прямий шлях лазерного променя.

Приклад роботи дигитайзера, наведений на рис.2.14.

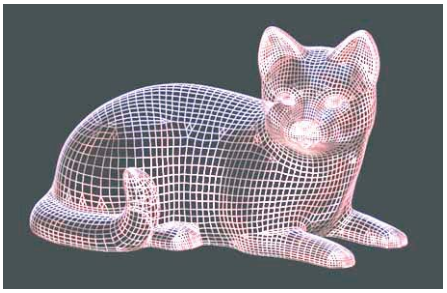


Рис.2.14. Приклад роботи дигитайзера

Лекція 3
Математичні основи комп'ютерної графіки. Представлення і перетворення точок і відрізків

3.1. Представлення і перетворення точок

Представлення точок здійснюється наступним чином:

- на площині $(x \quad y)$;
- у просторі $(x \quad y \quad z)$.

Перетворення точок.

Розглянемо результати матричного множення $(x \quad y)$, що визначає точку P, і матриці перетворення 2x2 загального виду:

$$(x \quad y)\cdot\begin{pmatrix} a & b \\ c & d \end{pmatrix} = (ax+cy \quad bx+dy) = (x^* \quad y^*) \tag{3.1}$$

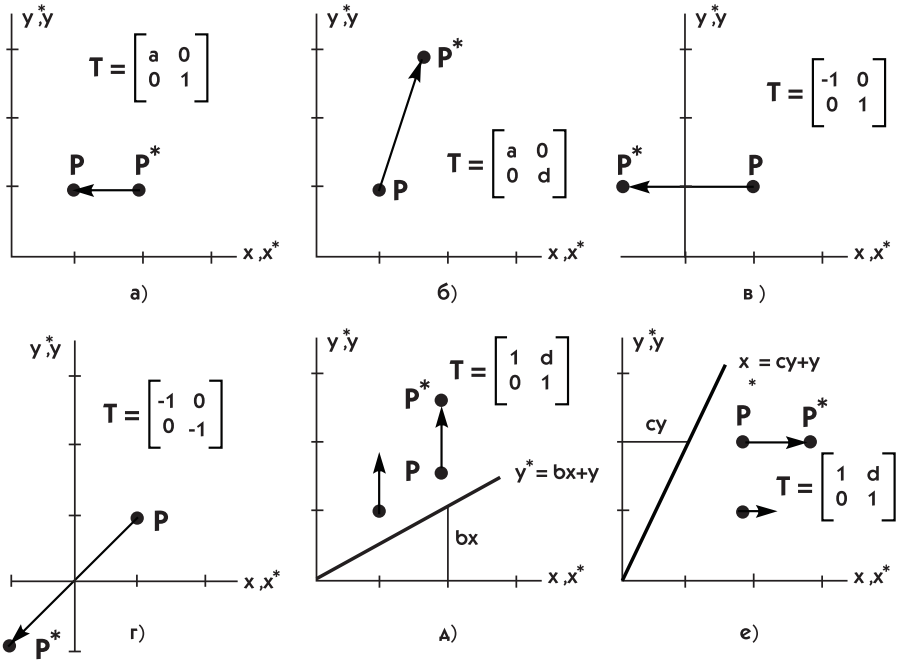


Рис.3.1. Перетворення точок.

Дослідимо декілька часткових випадків.

1) $a=d=1$ і $c=b=0$. Змін не відбувається

$$(x \quad y)\cdot\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = (x \quad y) = (x^* \quad y^*). \tag{3.2}$$

2) $d=1$, $b=c=0$. Зміна масштабу по осі x

$$(x \quad y)\cdot\begin{pmatrix} a & 0 \\ 0 & 1 \end{pmatrix} = (ax \quad y) = (x^* \quad y^*). \tag{3.3}$$

3) b=c=0. Зміна масштабу по осях x і y

$$\begin{pmatrix} x & y \end{pmatrix} \cdot \begin{pmatrix} a & 0 \\ 0 & d \end{pmatrix} = \begin{pmatrix} ax & dy \end{pmatrix} = \begin{pmatrix} x^* & y^* \end{pmatrix}. \qquad (3.4)$$

4) b=c=0, d=1, a=-1. Відображення координат відносно осі y

$$\begin{pmatrix} x & y \end{pmatrix} \cdot \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} -x & y \end{pmatrix} = \begin{pmatrix} x^* & y^* \end{pmatrix}. \qquad (3.5)$$

5) b=c=0, a=d=–1. Відображення відносно початку координат

$$\begin{pmatrix} x & y \end{pmatrix} \cdot \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} -x & -y \end{pmatrix} = \begin{pmatrix} x^* & y^* \end{pmatrix}. \qquad (3.6)$$

6) a=d=1,c=0. Зсув

$$\begin{pmatrix} x & y \end{pmatrix} \cdot \begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} x & bx+y \end{pmatrix} = \begin{pmatrix} x^* & y^* \end{pmatrix}. \qquad (3.7)$$

Для початку координат маємо інваріантно

$$\begin{pmatrix} 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 0 & 0 \end{pmatrix} = \begin{pmatrix} x^* & y^* \end{pmatrix}.$$

3.2. Перетворення прямих ліній

Пряма задана 2 векторами. Вектори положення точок А і В рівні значенням $\begin{pmatrix} 0 & 1 \end{pmatrix}$ і $\begin{pmatrix} 2 & 3 \end{pmatrix}$ відповідно.

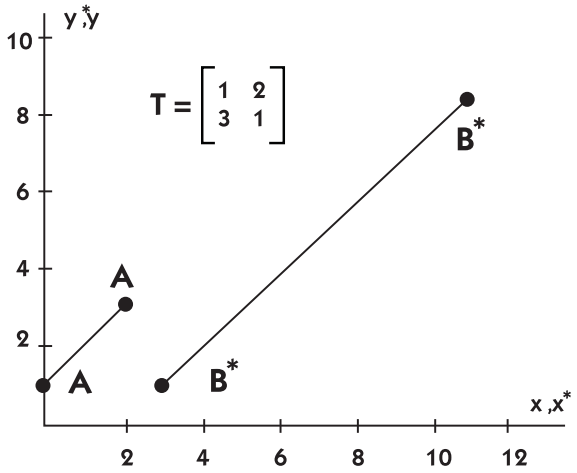


Рис.3.2. Перетворення прямих ліній.

Матриця перетворення прямих ліній має вигляд:

$$T = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}.$$

Одержимо:

$$AT = \begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 1 \end{pmatrix} = A^*, \qquad (3.8)$$

$$BT = \begin{pmatrix} 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 11 & 7 \end{pmatrix} = B^* . \qquad (3.9)$$

Альтернативне представлення лінії AB:

$$L = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}.$$

Після цього множення матриці L на T дасть таке представлення лінії A^*B^*:

$$LT = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 1 \\ 11 & 7 \end{pmatrix} = L^* . \qquad (3.10)$$

Операція перетворення збільшила довжину лінії і змінила її положення відносно осей координат.

3.3. Обертання

Розглянемо плоский трикутник ABC.

Здійснимо поворот на 90° проти годинникової стрілки. Покажемо це на рис.3.3.

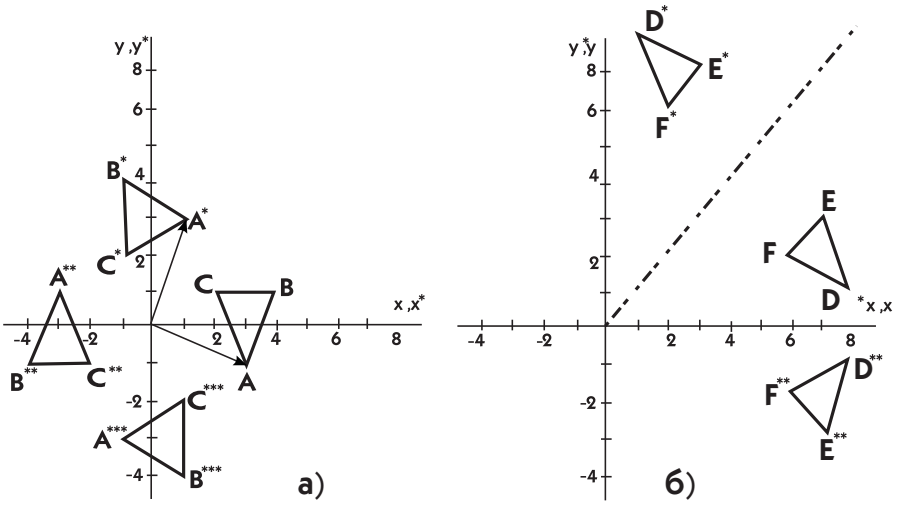


Рис.3.3. Обертання і відображення.

Одержимо

$$\begin{pmatrix} 3 & -1 \\ 4 & 1 \\ 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ -1 & 4 \\ -1 & 2 \end{pmatrix}. \qquad (3.11)$$

В результаті отримаємо трикутник A*B*C*.

Поворот на 180° задається матрицею

$$\begin{pmatrix} 3 & -1 \\ 4 & 1 \\ 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} -3 & 1 \\ -4 & -1 \\ -2 & -1 \end{pmatrix}, (\Delta A'B'C'),$$

поворот на 270° навколо початку координат - за допомогою такої матриці:

$$\begin{pmatrix} 3 & -1 \\ 4 & 1 \\ 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} -1 & -3 \\ 1 & -4 \\ 1 & -2 \end{pmatrix}, (\Delta A''B''C'').$$

3.4. Відображення

Відображення визначається поворотом на 180° навколо осі, що лежить у площині *xy*.

1) Обертання навколо прямої *y*=*x* задається матрицею:

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

Нові вирази визначаються співвідношенням:

$$\begin{pmatrix} 8 & 1 \\ 7 & 3 \\ 6 & 2 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 8 \\ 3 & 7 \\ 2 & 6 \end{pmatrix}. \qquad (3.12)$$

2) Обертання навколо осі *y*=0 задається матрицею:

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

Нові вершини визначаються співвідношенням:

$$\begin{pmatrix} 8 & 1 \\ 7 & 3 \\ 6 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 8 & -1 \\ 7 & -3 \\ 6 & -2 \end{pmatrix} \qquad (3.13)$$

3.5. Зміна масштабу

Зміна масштабу визначається значенням 2-х елементів головної діагоналі матриці.

Якщо використовуємо матрицю $\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$, маємо збільшення в 2 рази.

Якщо значення діагональних елементів не рівні, то має місце спотворення.

Трикутник ABC перетворений за допомогою матриці $\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$.

Трикутник DEF перетворений за допомогою матриці $\begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}$.

Маємо спотворення.

Рівномірна і нерівномірна зміни масштабу трикутників ABC та DEF наведені на рис.3.4.

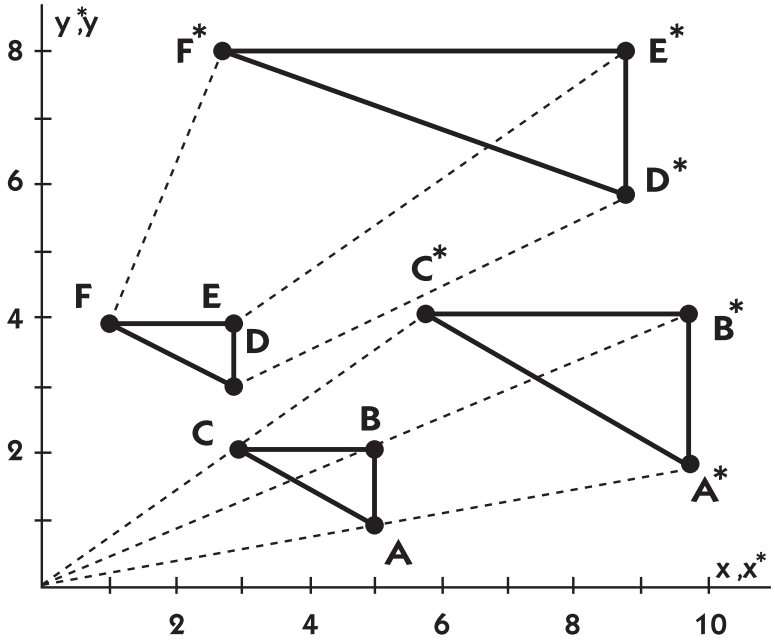


Рис.3.4. Рівномірна і нерівномірна зміна масштабів.

3.6. Двовимірний зсув і однорідні координати

Введемо третій компонент у вектори точок $(x \quad y)$ і $(x^* \quad y^*)$- $(x \quad y \quad 1)$ і $(x^* \quad y^* \quad 1)$.

Матриця перетворення матиме вигляд:

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

Таким чином,

$$(x \quad y \quad 1)\cdot\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ m & n \end{pmatrix} = (x+m \quad y+n) = (x^* \quad y^*). \qquad (3.14)$$

Константи m, n викликають зсув x^* і y^* відносно x і y.

Матриця перетворення 3×2 не квадратна - вона не має оберненої матриці.

Доповнимо матрицю перетворення до квадратної

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{pmatrix}. \qquad (3.15)$$

Третій компонент не змінюється.

Лекція 4
Однорідні координати. Тривимірні перетворення і стандартні проекції. Перспективні перетворення

4.1. Тривимірні перетворення

4.1.1. Вступ

Для кращого сприйняття форми об'єкта необхідно мати його зображення в тривимірному просторі. У багатьох випадках наочне представлення об'єкту можна одержати шляхом виконання операцій обертання і переносу, а також побудови проекцій.

Введемо однорідні координати. Точка в тривимірному просторі $(x \quad y \quad z)$ задається чотиримірним вектором $(x \quad y \quad z \quad 1)$ чи $(X \quad Y \quad Z \quad H)$. Перетворення з однорідних координат описується співвідношеннями

$$\begin{aligned} &(X \quad Y \quad Z \quad H) = (x \quad y \quad z \quad 1)\cdot T \\ &(x^* \quad y^* \quad z^* \quad 1) = (X/H \quad Y/H \quad Z/H \quad 1) \end{aligned} \tag{4.1}$$

де T - деяка матриця перетворення.

$$T = \begin{pmatrix} a & b & c & p \\ d & e & f & g \\ h & i & j & r \\ l & m & n & s \end{pmatrix}$$

Ця матриця може бути представлена у вигляді 4 окремих частин

$$\begin{pmatrix} 3\times3 & 3\times1 \\ 1\times3 & 1\times1 \end{pmatrix}$$

Матриця 3×3 здійснює лінійне перетворення у вигляді зміни масштабу, зсуву й обертання. Матриця-рядок 1×3 робить перенос, а матриця-стовпець 3×1 - перетворення в перспективі. Останній скалярний елемент виконує загальну зміну масштабу. Повне перетворення, отримане дією на вектор положення матрицею 4×4 і нормалізації перетвореного вектора, будемо називати білінійним перетворенням. Воно забезпечує виконання комплексу операцій зсуву, часткової зміни масштабу, обертання, відображення, переносу, а також зміни масштабу зображення в цілому.

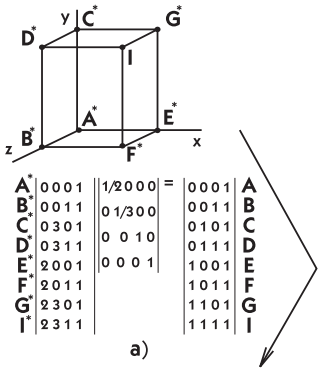
4.1.2. Тривимірна зміна масштабу

Діагональні елементи основної матриці перетворення 4×4 здійснюють часткову і повну зміну масштабу. Розглянемо перетворення

$$(x \quad y \quad z \quad 1)\cdot\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = (ax \quad ey \quad jz \quad 1) = (x^* \quad y^* \quad z^* \quad 1), \tag{4.2}$$

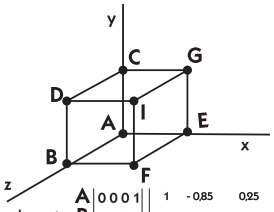
яке робить часткову зміну масштабу.

На рис.4.1(а) показане перетворення паралелепіпеда в одиничний куб шляхом зміни масштабу.

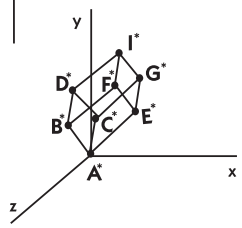


$$\begin{array}{c|c} A^* & 0\,0\,0\,1 \\ B^* & 0\,0\,1\,1 \\ C^* & 0\,3\,0\,1 \\ D^* & 0\,3\,1\,1 \\ E^* & 2\,0\,0\,1 \\ F^* & 2\,0\,1\,1 \\ G^* & 2\,3\,0\,1 \\ I^* & 2\,3\,1\,1 \end{array} \left\| \begin{array}{c} 1/2\,0\,0\,0 \\ 0\,1/3\,0\,0 \\ 0\;0\;1\,0 \\ 0\;0\;0\,1 \end{array} \right| = \begin{array}{c|c} 0\,0\,0\,1 & A \\ 0\,0\,1\,1 & B \\ 0\,1\,0\,1 & C \\ 0\,1\,1\,1 & D \\ 1\,0\,0\,1 & E \\ 1\,0\,1\,1 & F \\ 1\,1\,0\,1 & G \\ 1\,1\,1\,1 & I \end{array}$$

а)



$$\begin{array}{c|c} A & 0\,0\,0\,1 \\ B & 0\,0\,1\,1 \\ C & 0\,1\,0\,1 \\ D & 0\,1\,1\,1 \\ E & 1\,0\,0\,1 \\ F & 1\,0\,1\,1 \\ G & 1\,1\,0\,1 \\ I & 1\,1\,1\,1 \end{array} \left\| \begin{array}{cccc} 1 & -0{,}85 & 0{,}25 & 0 \\ -0{,}75 & 1 & 0{,}7 & 0 \\ 0{,}5 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right| = \begin{array}{c|c} 0\;\;0\;\;0\;\;1 & A^* \\ 0{,}5\;\;1\;\;1\;\;1 & B^* \\ -0{,}75\;\;1\;\;0{,}7\;\;1 & C^* \\ -0{,}25\;\;2\;\;1{,}7\;\;1 & D^* \\ 1\;\;-0{,}85\;\;0{,}25\;\;1 & E^* \\ 1{,}5\;\;0{,}15\;\;1{,}25\;\;1 & F^* \\ 0{,}25\;\;0{,}15\;\;0{,}95\;\;1 & G^* \\ 0{,}75\;\;1{,}15\;\;1{,}95\;\;1 & I^* \end{array}$$



$$\begin{array}{c|c} A & 0\,0\,0\,1 \\ B & 0\,0\,1\,1 \\ C & 0\,1\,0\,1 \\ D & 0\,1\,1\,1 \\ E & 1\,0\,0\,1 \\ F & 1\,0\,1\,1 \\ G & 1\,1\,0\,1 \\ I & 1\,1\,1\,1 \end{array} \left\| \begin{array}{c} 1\;0\;0\;0 \\ 0\;1\;0\;0 \\ 0\;0\;1\;0 \\ 0\;0\;0\;1/2 \end{array} \right| = \begin{array}{c|c} 0\,0\,0\,1 & A^* \\ 0\,0\,2\,1 & B^* \\ 0\,2\,0\,1 & C^* \\ 0\,2\,2\,1 & D^* \\ 2\,0\,0\,1 & E^* \\ 2\,0\,2\,1 & F^* \\ 2\,2\,0\,1 & G^* \\ 2\,2\,2\,1 & I^* \end{array}$$

б)

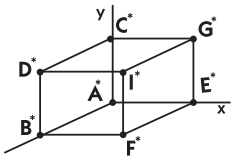


Рис.4.1. Тривимірні перетворення із зміною масштабів.

Загальна зміна масштабу отримується за рахунок використання четвертого діагонального елемента, тобто

$$\begin{pmatrix} x & y & z & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{pmatrix} = \begin{pmatrix} x & y & z & s \end{pmatrix} = \tag{4.3}$$

$$= \begin{pmatrix} x^* & y^* & z^* & 1 \end{pmatrix} = \begin{pmatrix} \frac{x}{s} & \frac{y}{s} & \frac{z}{s} & 1 \end{pmatrix}$$

Це перетворення ілюструє рис.4.1(б). Такий же результат можна отримати при рівних коефіцієнтах часткових змін масштабів. У цьому випадку матриця перетворення повинна бути рівна

$$\begin{pmatrix} \frac{1}{s} & 0 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & 0 \\ 0 & 0 & \frac{1}{s} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.4}$$

Вектори положення точок A і B рівні $\begin{pmatrix} 0 & 1 \end{pmatrix}$ і $\begin{pmatrix} 2 & 3 \end{pmatrix}$.

4.1.3. Тривимірний зсув

Недіагональні елементи верхньої лівої підматриці 3×3 від загальної матриці перетворення розміру 4×4 здійснюють зсуви в трьох вимірах, тобто

$$\begin{pmatrix} x & y & z & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ h & i & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \tag{4.5}$$

$$= \begin{pmatrix} x + dy + hz & bx + y + iz & cx + fy + z & 1 \end{pmatrix}.$$

Простий тривимірний зсув одиничного куба представлений на рис.4.1(в).

4.1.4. Тривимірні обертання

Раніше було показано, що матриця 3×3 забезпечувала комбінацію операцій зміни масштабу і зсуву. Однак, якщо визначник матриці перетворення 3×3 дорівнює +1, то має місце чисте обертання навколо початку координат.

Перед розглядом загального випадку тривимірного обертання навколо довільної осі дослідимо кілька окремих випадків. При обертанні навколо осі x розміри уздовж осі x не змінюються. Таким чином, матриця перетворень буде мати нулі в першому рядку і першому стовпці, за винятком одиниці на головній діагоналі.

Це приводить до матриці перетворення, що відповідає повороту на кут θ навколо осі x і задається співвідношенням

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (4.6)$$

Обертання вважається додатнім, тобто за годинниковою стрілкою, якщо дивитися з початку координат вздовж осі обертання. На рис.4.2(а) показаний поворот на -90° навколо осі *x*.

Для обертання на кут *Ф* навколо осі *y* - нулі ставлять у другому рядку і другому стовпці матриці перетворення, за винятком одиниці на головній діагоналі. Повна матриця задається виразом

$$T = \begin{pmatrix} \cos\Phi & 0 & -\sin\Phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\Phi & 0 & \cos\Phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (4.7)$$

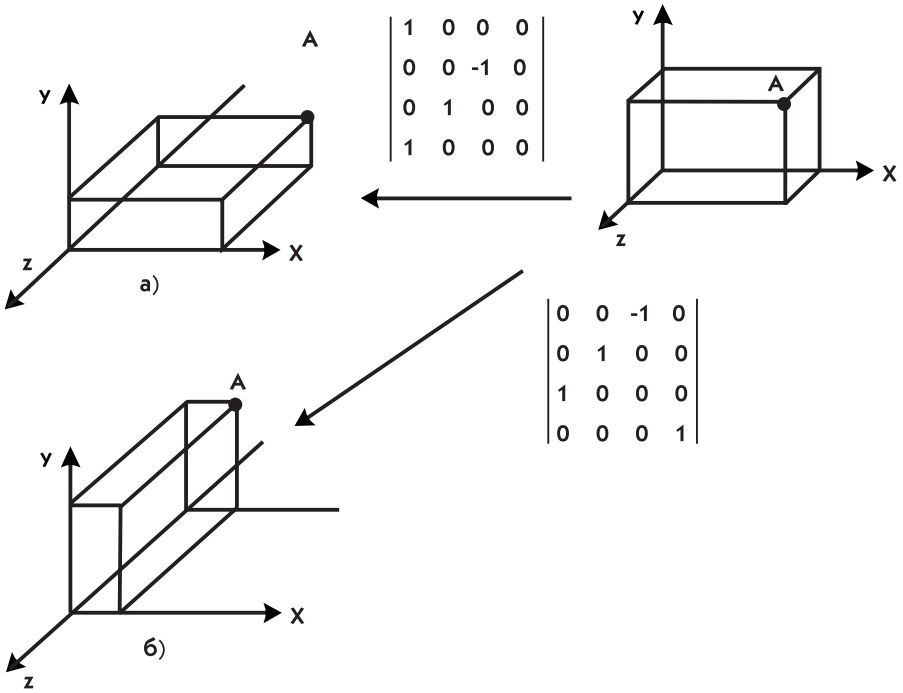


Рис.4.2. Тривимірні обертання.

На рис.4.2(б) показаний поворот на 90° навколо осі *y*. Аналогічно матриця перетворення для обертання на кут ϕ навколо осі *z* має вид

$$T = \begin{pmatrix} \cos\phi & \sin\phi & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (4.8)$$

Аналіз визначників для матриць (4.6)-(4.8) показує, що для будь-якої матриці обертання детермінант дорівнює +1.

Через те що обертання описуються множенням матриць, тривимірні обертання некомутативні, тобто порядок множення буде впливати на кінцевий результат.

Для того щоб показати це, розглянемо обертання навколо осі x, за яким слідує обертання на такий же кут навколо осі y. Використовуючи рівняння (4.6) і (4.7) при $\theta = \Phi$, одержимо

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\Phi & 0 & -\sin\Phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\Phi & 0 & \cos\Phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ \sin^2\theta & \cos\theta & \cos\theta\sin\theta & 0 \\ \cos\theta\sin\theta & -\sin\theta & \cos^2\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (4.9)$$

Інша послідовність дій, тобто обертання навколо осі y і наступне за ним обертання на такий же кут навколо осі x при $\theta = \Phi$ дає

$$T = \begin{pmatrix} \cos\Phi & 0 & -\sin\Phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\Phi & 0 & \cos\Phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos\theta & \sin^2\theta & -\sin\theta\cos\theta & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ \sin\theta & -\sin\theta\cos\theta & \cos^2\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (4.10)$$

На рис.4.3 для лівого верхнього зображення штриховими лініями показані результати двох послідовних обертань, описаних матрицею перетворення (4.9). Зображення, отримане обертаннями, виконаними в іншій послідовності, описаними рівняннями (4.10), показані суцільною лінією. З порівняння отриманих зображень видно, що при зміні порядку обертання виходять різні результати.

Часто буває необхідно обертати зображення навколо однієї з осей декартової системи координат.

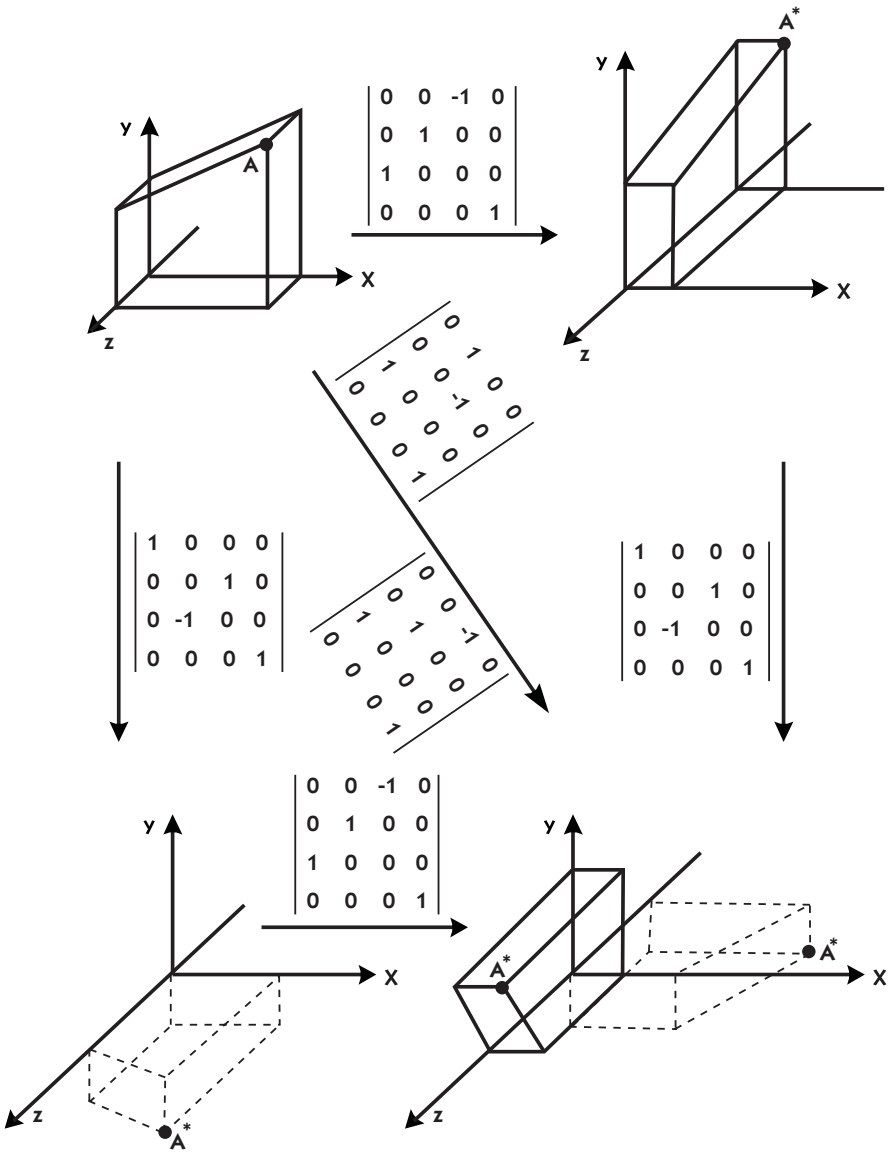


Рис.4.3. Некомутативність тривимірних обертань

4.1.5. Відображення в просторі

Іноді потрібно виконати дзеркальне відображення тривимірного зображення. У трьох вимірах найпростіше відображення здійснюється щодо площини. Для відображення без зміни масштабів необхідно, щоб визначник перетворення дорівнював -1,0. При відображенні щодо площини xy змінюється тільки знак координати *z*.

Матриця перетворення для відображення щодо площини *xy* має вигляд

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (4.11)$$

Відображення одиничного куба щодо площини xy показане на рис.4.4. Для відображення щодо площини *yz* матриця перетворення має вигляд

$$T = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (4.12)$$

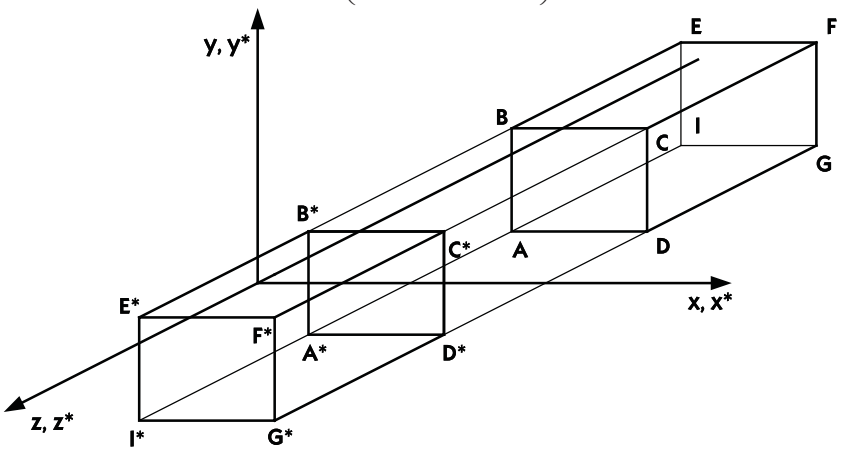


Рис.4.4. Просторове відображення щодо площини *xy*.

$$\begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \\ G \\ I \end{array} \begin{pmatrix} 1 & 0 & -1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 0 & -2 & 1 \\ 1 & 1 & -2 & 1 \\ 2 & 0 & -1 & 1 \\ 2 & 1 & -1 & 1 \\ 2 & 0 & -2 & 1 \\ 2 & 1 & -2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 \\ 1 & 1 & 2 & 1 \\ 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 1 \\ 2 & 0 & 2 & 1 \\ 2 & 1 & 2 & 1 \end{pmatrix} \begin{array}{c} A^* \\ B^* \\ C^* \\ D^* \\ E^* \\ F^* \\ G^* \\ I^* \end{array} \qquad (4.12)$$

Для відображення щодо площини xz матриця перетворення має наступний вигляд

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.13}$$

Відображення щодо інших площин можна одержати шляхом комбінації обертання і відображення.

4.1.6. Просторовий перенос

Тривимірний лінійний перенос зображення задається наступним виразом

$$\begin{pmatrix} X & Y & Z & H \end{pmatrix} = \begin{pmatrix} x + l & y + m & z + n & 1 \end{pmatrix} \tag{4.14}$$

Після перемножування одержимо

$$\begin{aligned} x^* &= \frac{X}{H} = x + l; \\ y^* &= \frac{Y}{H} = y + m; \\ z^* &= \frac{Z}{H} = z + n. \end{aligned} \tag{4.15}$$

4.1.7. Тривимірне обертання навколо довільної осі

Метод двовимірного плоского обертання навколо довільної осі був розглянутий раніше. Узагальненням цього методу є спосіб обертання навколо довільної осі в тривимірному просторі. Як і для плоского випадку, розглянута процедура полягає в переносі зображення і заданої осі обертання, що забезпечує обертання навколо осі, що проходить через початок координат.

Метод тривимірного обертання полягає в лінійному переносі, обертанні навколо початку координат і зворотному лінійному переносі у вихідне положення.

Якщо вісь, навколо якої виконується обертання, проходить через точку A=$\begin{pmatrix} l & m & n & 1 \end{pmatrix}$, матриця перетворення визначається наступним виразом:

$$\begin{pmatrix} X & Y & Z & H \end{pmatrix} = \begin{pmatrix} x & y & z & 1 \end{pmatrix} \times$$
$$\times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -l & -m & -n & 1 \end{pmatrix} R \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{pmatrix} \tag{4.16}$$

де елементи матриці обертання R розміру 4×4 визначаються в загальному випадку співвідношенням

$$R=\begin{pmatrix} n_1^2+\left(1-n_1^2\right)\cos\theta & n_1n_2\left(1-\cos\theta\right)+n_3\sin\theta & n_1n_3\left(1-\cos\theta\right)-n_2\sin\theta & 0 \\ n_1n_2\left(1-\cos\theta\right)-n_3\sin\theta & n_2^2+\left(1-n_2^2\right)\cos\theta & n_2n_3\left(1-\cos\theta\right)-n_1\sin\theta & 0 \\ n_1n_3\left(1-\cos\theta\right)+n_2\sin\theta & n_2n_3\left(1-\cos\theta\right)-n_1\sin\theta & n_2^2+\left(1-n_3^2\cos\theta\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.17)$$

4.1.8. Афінна і перспективна геометрія

Ідея опису точки вектором виникла з геометричних уявлень. Теореми геометрії розвивалися як для перспективної, так і для афінної геометрії. У них важливими є поняття паралельності і співвідношення між паралельними прямими.

Афінне перетворення є комбінацією лінійних перетворень, супроводжуваних переносом зображень. Для афінного перетворення останній стовпець в узагальненій матриці перетворення розміром 4×4 повинен бути рівним $\begin{pmatrix}0 & 0 & 0 & 1\end{pmatrix}$. Афінні перетворення формують зручну підсистему білінійних перетворень, тому що добуток двох афінних перетворень також є афінним. Це дозволяє представити узагальнену орієнтацію системи точок стосовно довільної координатної системи при збереженні одиничного значення однорідної координати *H*.

Перспективними зображеннями часто користуються художники й архітектори, тому що ці зображення дозволяють одержати картину, близьку до реальної, однак через складність побудови їх рідко використовують конструктори. У тих випадках, коли для машинного представлення зображення використовуються однорідні координати, однаково легко можуть бути отримані як афінні, так і перспективні перетворення.

У перспективній геометрії немає двох прямих, паралельних одна одній, і перспективна площина може бути розглянута як напівсферична поверхня, а перспективне перетворення - як перетворення з одного тривимірного простору в інший.

Перспективне перетворення має місце у випадку, коли останній стовпчик узагальненої матриці перетворення 4×4 не нульовий.

Часто перспективне перетворення асоціюється з побудовою проекції на площині, наприклад, із точки $z=c1$, іменованої центром проекції. Комбінація перспективного перетворення з проекційним утворить перспективну проекцію.

Перспективна проекція являє собою перетворення зображення з тривимірного простору в двовимірний.

Якщо центр проектування розташовується в нескінченності, то перспективна проекція називається аксонометричною. Іншими словами, вид проекції залежить від розташування цього центра.

Аксонометрична проекція утворюється за допомогою афінного перетворення, визначник якого дорівнює нулю. Існують кілька типів аксонометричних проекцій, що застосовуються в нарисній геометрії й інженерній графіці. Для одержання математичних залежностей, що описують аксонометричні проекції, використовується матриця перетворення розміром 4×4, необхідна для проведення афінного перетворення системи точок. Точки потім проектуються на площину з центра проектування в нескінченності.

Класифікація загальних аксонометричних проекцій приведена в табл.4.1.

Таблиця 4.1

Аксонометричні проекції	
Проекція	Опис
Ортогональна	Матриця перетворення здійснює тільки обертання; таким чином, координатні осі залишаються ортогональними під час проектування
Диметрична	Дві з трьох осей під час проектування однаково зменшені
Ізометрична	Усі три осі під час проектування однаково зменшені

4.2. Тривимірні проекції

4.2.1. Вступ

У загальному випадку проекції перетворюють точки, задані в системі координат розмірністю n, у точки системи координат розмірністю, меншою, ніж n.

У розглянутому випадку за допомогою проектування три виміри відображаються в два. Проекція тривимірного об'єкта (представленого у виді сукупності точок) будується за допомогою прямих проектуючих променів, що називаються проекторами і які виходять з центра проекції, проходять через кожну точку об'єкта і, перетинаючи картинну площину, утворюють проекцію.

На рис.4.5 і 4.6 показані дві різні проекції того самого відрізка, а також проектори, що проходять через його кінцеві точки. На щастя, проекція відрізка на площину сама є відрізком, так що достатньо спроектувати лише кінцеві точки.

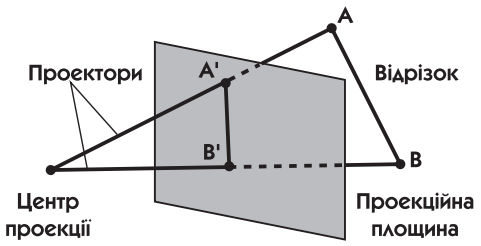


Рис.4.5. Відрізок AB і його центральна проекція A'B'

Визначений таким чином клас проекцій відомий під назвою плоских геометричних проекцій, оскільки проектування в цьому випадку здійснюється на площину, а не на викривлену поверхню і як проектори використовуються прямі а не криві лінії.

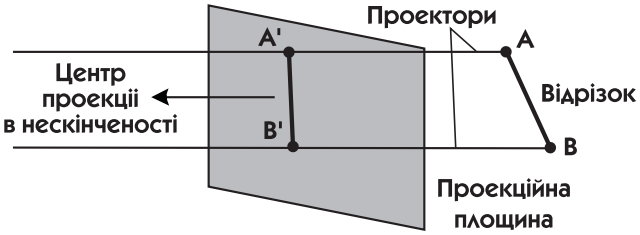


Рис.4.6. Відрізок AB і його центральна проекція A'B'
(проектори AA' і BB' паралельні)

Плоскі геометричні проекції, що надалі будемо називати просто проекціями, можна поділити на два основних класи: центральні і паралельні. Розходження між ними визначається співвідношенням між центром проекції і проекційною площиною. Якщо відстань між ними обмежена, то проекція буде центральною, якщо ж вона нескінченна, то проекція буде паралельною. Обидва ці випадки проілюстровані на рис.4.5 і 4.6. Паралельні проекції названі так тому, що центр проекції нескінченно віддалений і всі проектори паралельні. При описі центральної проекції ми явно задаємо її центр проекції, у той час як визначаючи паралельну проекцію, ми вказуємо напрям проектування.

Центральна проекція породжує візуальний ефект, аналогічний тому, до якого приводять фотографічні системи чи зорова система людини, і тому використовується у випадках, коли бажано досягти певного ступеня реальності. Цей ефект відомий як перспективне скорочення: розмір центральної проекції об'єкта змінюється обернено пропорційно до відстані від центра проекції до об'єкта. Це означає, що, хоча центральна проекція об'єктів є реалістичною, вона виявляється непридатною для представлення точної форми і розмірів об'єктів: із проекції не можна одержати інформацію про відстані; кути зберігаються тільки на тих гранях об'єкта, що паралельні проекційній площині; проекції паралельних прямих у загальному випадку не паралельні.

Паралельна проекція породжує менш реалістичне зображення, оскільки відсутнє перспективне скорочення, хоча при цьому можуть мати місце різні постійні скорочення вздовж кожної з осей. Проекція фіксує дійсні розміри (з точністю до скалярного множника), і паралельні прямі залишаються паралельними. Як і у випадку центральної проекції, кути зберігаються тільки на тих гранях об'єкта, що паралельні проекційній площині. Кресляреві легше зображувати паралельні проекції, ніж центральні, однак, що простіше у випадку машинної графіки – неясно.

4.2.2. Центральні проекції

Центральні проекції будь-якої сукупності паралельних прямих, що не паралельні до проекційної площини, будуть сходитися в точці сходу. Паралельні прямі в тривимірному просторі перетинаються лише в нескінченності, тому точку сходу можна уявити собі як проекцію точки, що знаходиться в нескінченності. Зазвичай існує нескінченне число точок сходу.

Якщо сукупність прямих паралельна до однієї з головних координатних осей, то їхня точка сходу називається головною точкою сходу. Є тільки три такі точки, що відповідають перетинам головних координатних осей із проекційною площиною. Наприклад, якщо проекційна площина перетинає одну тільки вісь *z* (і, таким чином, перпендикулярна до неї), то лише на цій осі буде лежати головна точка сходу, оскільки прямі, паралельні як *y*-, так і *x*-осям, паралельні також і до проекційної площини і тому не мають точки сходу.

Центральні проекції класифікуються в залежності від числа головних точок сходу, якими вони володіють, а отже, і від числа координатних осей, що перетинають проекційну площину. На рис.4.7 наведені дві різні одноточкові проекції куба.

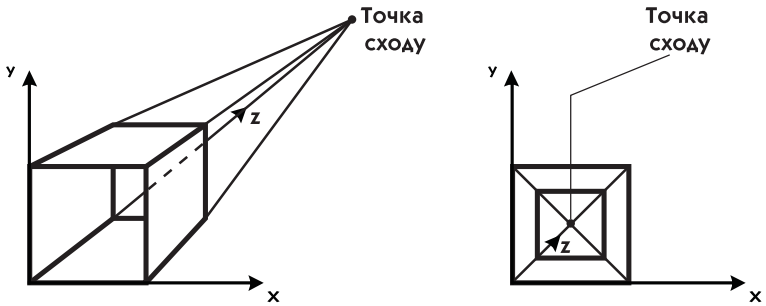


Рис.4.7. Одноточкові центральні проекції куба на площину, що перетинає вісь z, а також точка сходу прямих, перпендикулярних до проекційної площини

Це одноточкові проекції, оскільки прямі паралельні осям *x* і *y*, не сходяться, а сходяться тільки прямі, паралельні осі *z*. На рис.4.8 показаний процес побудови одноточкової центральної проекції, деякі з її проекторів, а також проекційна площина, що перетинає одну лише вісь *z*.

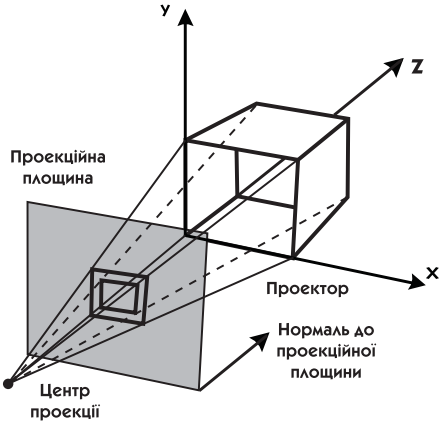


Рис.4.8. Побудова одноточкової центральної проекції куба на площину, що перетинає вісь *z* (нормаль до проекційної площини паралельна до осі *z*)

Побудова двохточкової центральної проекції ілюструється рис.4.9. Прямі, паралельні осі *y*, у цій проекції не сходяться.

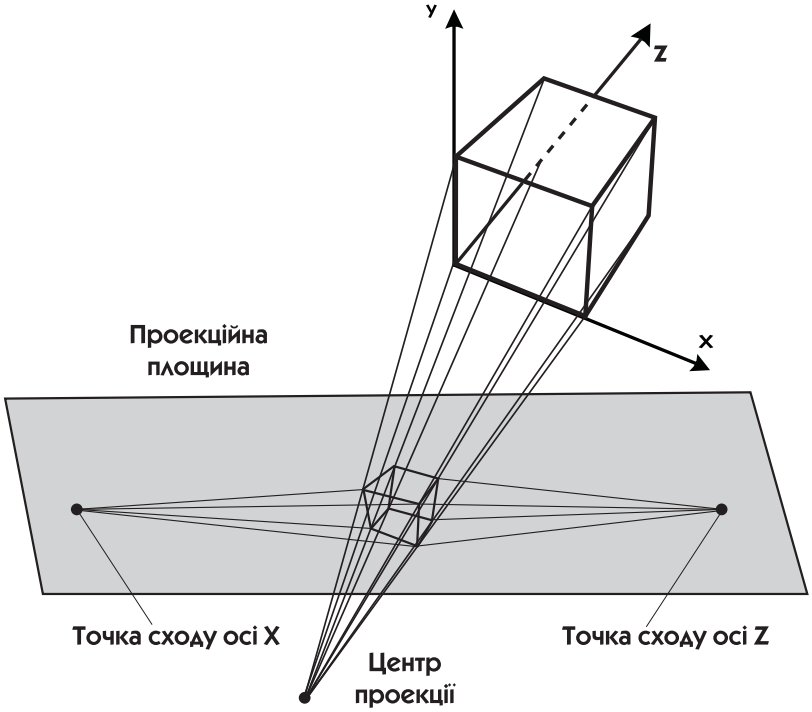


Рис.4.9. Побудова двохточкової центральної проекції куба на площину, що перетинає вісь *z* (нормаль до проекційної площини паралельна осі *z*)

Двохточкова центральна проекція широко застосовується в архітектурному, інженерному і промисловому проектуванні і у рекламних зображеннях, в яких вертикальні прямі проектуються як паралельні і, отже, не сходяться. Трьохточкові центральні проекції майже зовсім не використовуються, по-перше, тому, що їх важко конструювати, а, по-друге, через те, що вони додають мало нового з погляду реалістичності в порівнянні з двохточковою проекцією.

4.2.3. Рівнобіжні проекції

Рівнобіжні проекції поділяються на два типи в залежності від співвідношення між напрямком проектування і нормаллю до проекційної площини. В ортографічних паралельних проекціях ці напрямки збігаються, а в косокутних паралельних проекціях вони не збігаються. Тобто в ортографічних проекціях напрям проектування є нормаллю до проекційної площини. Найчастіше використовуваними видами ортографічних проекцій є вид спереду, вид зверху (план) і вид збоку, у яких картинна площина перпендикулярна головним координатним осям, що збігаються внаслідок цього з напрямком проектування.

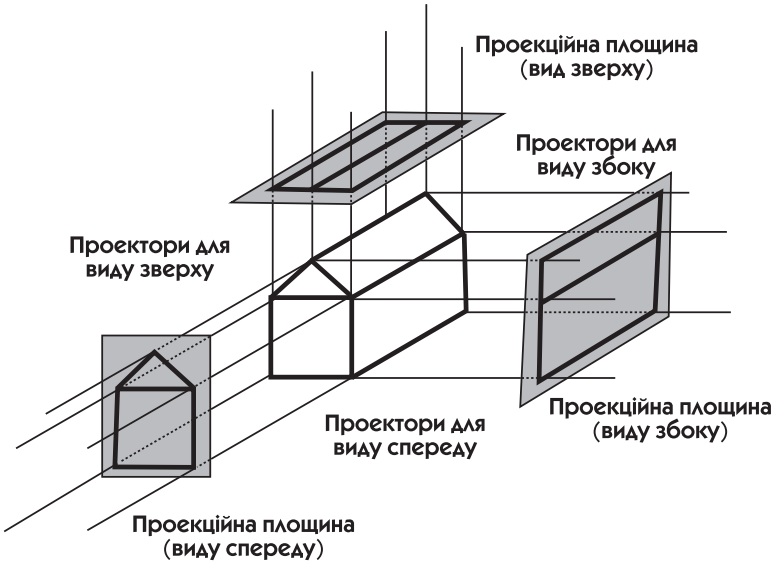


Рис.4.10. Побудова трьох ортографічних проекцій

На рис.4.10 показаний процес побудови кожної з цих трьох проекцій, що часто застосовуються в інженерній графіці для опису машинобудівних деталей, агрегатів і споруд, тому що по них можна вимірювати відстані і кути. Оскільки кожна проекція відображає лише одну сторону об'єкта, часто зовсім непросто уявити собі просторову структуру проектованого об'єкта, навіть якщо розглядати відразу кілька проекцій того самого об'єкта.

У випадку аксонометричних ортографічних проекцій використовуються проекційні площини не перпендикулярні головним координатним осям, тому на них зображуються відразу кілька сторін об'єкта, так само як і при центральному проектуванні, однак в аксонометрії укорочування постійне, тоді як у випадку центральної проекції воно зв'язане з відстанню від центра проекції.

При аксонометричному проектуванні зберігається паралельність прямих, а кути змінюються; відстані ж можна виміряти вздовж кожної з головних координатних осей (у загальному випадку з різними масштабними коефіцієнтами).

Широко використовується такий вид аксонометричної проекції, як ізометрична проекція. У цьому випадку нормаль до проекційної площини (а, отже, і напрямок проектування) утворює рівні кути з кожною з головних координатних осей. Якщо нормаль до проекційної площини має координати (a, b, c), то вимагатимемо, щоб $|a|=|b|=|c|$ чи $\pm a=\pm b=\pm c$. Є рівно вісім напрямків (по одному в кожному з октантів), що задовольняють цій умові, однак існують лише чотири різні ізометричні проекції (якщо не розглядати видалення схованих ліній), оскільки вектори (a, a, a) і (-a, -a, -a) визначають нормалі до одній і тієї ж проекційної площини. Одиничними нормалями цих напрямків є вектори (a, a, a), (-a, a, a), і (a, -a, a).

На рис.4.11 показаний процес побудови ізометричної проекції з напрямком (1, -1, 1).

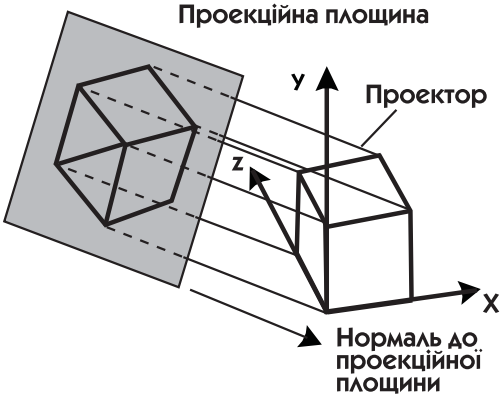


Рис.4.11. Побудова ізометричної проекції одиничного куба

Ізометрична проекція має наступну властивість: усі три головні координатні осі однаково зменшуються. Тому можна проводити виміри вздовж напрямків осей з тим самим масштабом (звідси назва*: ізо*, що означає "дорівнює", і *метрія* - "вимір"). Крім того, головні координатні осі проектуються так, що їхні проекції утворюють між собою рівні кути (рис.4.12).

Косокутні проекції (другий тип паралельних проекцій) поєднують у собі властивості ортографічних проекцій (видів спереду, зверху і збоку) із властивостями аксонометрії. У цьому випадку проекційна площина перпендикулярна до головної координатної осі, тому сторона об'єкта, паралельна до цієї площини, проектується так, що можна вимірювати кути і відстані. Проектування інших сторін об'єкта також допускає проведення лінійних вимірів (але не кутових) уздовж головних осей. Завдяки цим властивостям, а також простоті побудови, косокутні проекції широко (хоч і не занадто) використовуються в цій книзі. На рис.4.13 показаний процес побудови косокутної проекції. Відзначимо, що нормаль до проекційної площини і напрям проектування не збігаються.

Двома важливими видами косокутних проекцій є проекції кавальє (cavalier) (горизонтальна косокутна ізометрія) і кабіне (cabinet) (фронтальна косокутна диметрія).

У проекції кавальє напрям проектування складає з площиною кут 45°. В результаті проекція відрізка, перпендикулярного до проекційної площини, має ту ж довжину, що і сам відрізок, тобто скорочування відсутнє. На рис.4.14 наведено декілька проекцій кавальє одиничного куба на площину *xy*. Тут усередину ідуть лінії, що є проекціями тих ребер куба, які перпендикулярні до площини *xy*; вони розташовані під кутом α до горизонталі. Цей кут звичайно складає 30° чи 45°.

Проекція кабіне, показана на рис.4.15, має напрямок проектування, що складає з проекційною площиною кут arccos (1/2). При цьому відрізки, перпендикулярні до проекційної площини, після проектування складають 1/2 їх дійсної довжини.

Проекції кабіне є більш реалістичними, ніж проекції кавальє, оскільки скорочування з коефіцієнтом 1/2 більше погоджується з нашим візуальним досвідом.

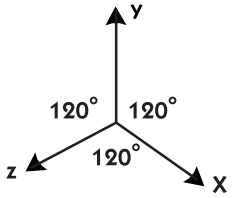


Рис.4.12. Ізометрична проекція одиничних векторів

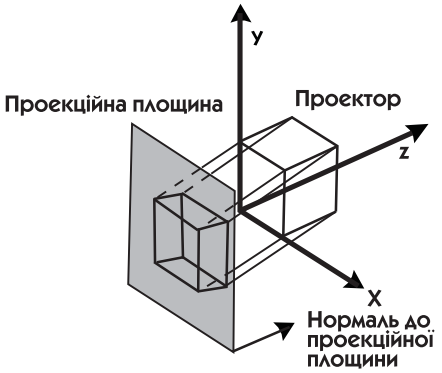


Рис.4.13. Побудова косокутної поекції

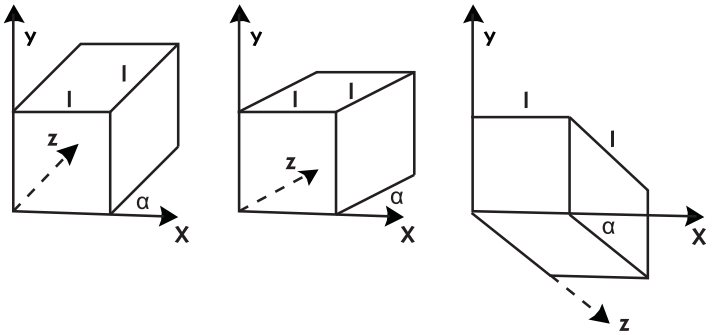


Рис.4.14 Проекція кавальє одиничного куба на площину z = 0

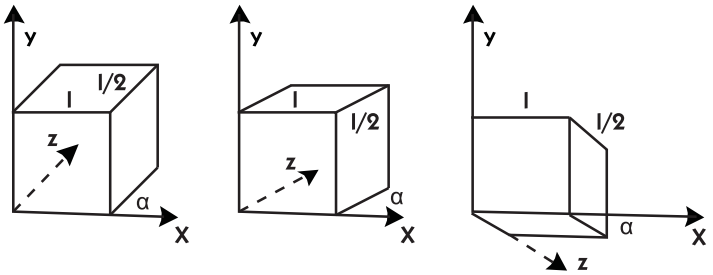


Рис.4.15 Проекція кавальє одиничного куба на площину z = 0

4.2.4. Паралельні проекції

Метою розділу є ознайомлення з елементарним математичним апаратом плоских геометричних проекцій. Для простоти будемо вважати, що при центральному проектуванні картинна площина перпендикулярна осі *z* і збігається з площиною *z*=*d*, а при паралельному збігається з площиною *z*=0.

Проекції розглядаються в системі координат спостерігача, що є лівосторонньою. Система координат, в якій вісь *x* спрямована вправо, вісь *y* - вгору, а вісь *z* - усередину екрана, природньо погоджується з екраном дисплея.

На рис.4.16 наведені три зображення лівосторонньої системи координат, у яких точка P проектується на проекційну площину, розташовану на відстані *d* від початку координат.

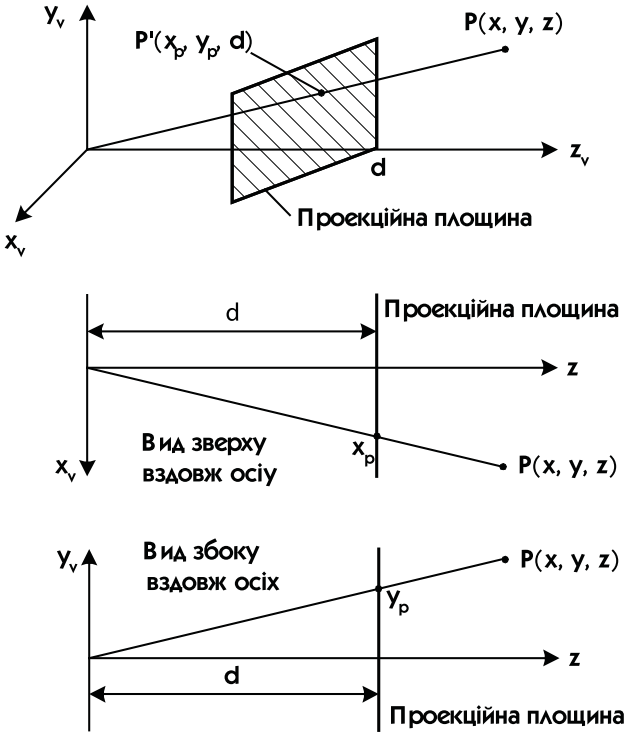


Рис.4.16. Центральна проекція

Кожну з проекцій можна описати матрицею розміром 4×4. Цей спосіб виявляється зручним, оскільки з'являється можливість об'єднати матрицю проектування з матрицею перетворення, представивши в результаті дві операції (перетворення і проектування) у виді однієї матриці.

У цьому розділі ми одержимо матриці розміром 4×4 для декількох проекцій і насамперед для центральної.

Для обчислення координат *Xp* і *Yp* центральної проекції точки (*x*, *y*, *z*) напишемо співвідношення, які отримані з подібності трикутників (рис.4.16):

$$\frac{X_P}{d} = \frac{x}{z}, \quad \frac{Y_P}{d} = \frac{y}{z}. \tag{4.18}$$

Перемножуючи обидві сторони кожного співвідношення на d, одержимо

$$X_P = \frac{d \cdot x}{z} = \frac{x}{z/d}, \quad Y_P = \frac{d \cdot y}{z} = \frac{y}{z/d}. \tag{4.19}$$

Відстань *d* є в даному випадку масштабним множником, що застосовується до координат *Xp* і *Yp*.

Фактором, що приводить до того, що на центральній проекції більш віддалені об'єкти виглядають дрібніше, ніж ближчі, є ділення на *z*. Відзначимо, що допустимі всі значення *z*, крім *z*=0. Точки можуть розташовуватися як за центром проекції на від'ємній частині осі *z*, так і між центром проекції і проекційною площиною.

Ці перетворення для центральної проекції можна представити у вигляді матриці розміром 4x4:

$$M_{\text{центр}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.20}$$

Перемножуючи точку $P = (x \quad y \quad z \quad 1)$ з матрицею $M_{\text{центр}}$. отримаємо загальний вираз для точки в однорідних координатах $(X \quad Y \quad Z \quad W)$:

$$(X \quad Y \quad Z \quad W) = P \cdot M_{\text{центр}} = (x \quad y \quad z \quad 1) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.21}$$

Після множення маємо

$$(X \quad Y \quad Z \quad W) = \left(x \quad y \quad z \quad \frac{z}{d} \right). \tag{4.22}$$

Тепер, поділивши на *W* (що дорівнює *z/d*) для зворотнього переходу до трьох вимірів, отримаємо

$$\left(\frac{X}{W} \quad \frac{Y}{W} \quad \frac{Z}{W} \quad 1 \right) = (x_p \quad y_p \quad z_p \quad 1) = \left(\frac{x}{z/d} \quad \frac{y}{z/d} \quad d \quad 1 \right). \tag{4.23}$$

Результат є коректним, оскільки містить перетворену *z* - координату з 1, що відповідає положенню проекційної площини вздовж осі z.

При іншому представленні центрального проектування, що застосовується в деяких роботах, проекційна площина сполучається з площиною z=0, а центр проекції розташовується в точці z=- *d* (рис.4.17).

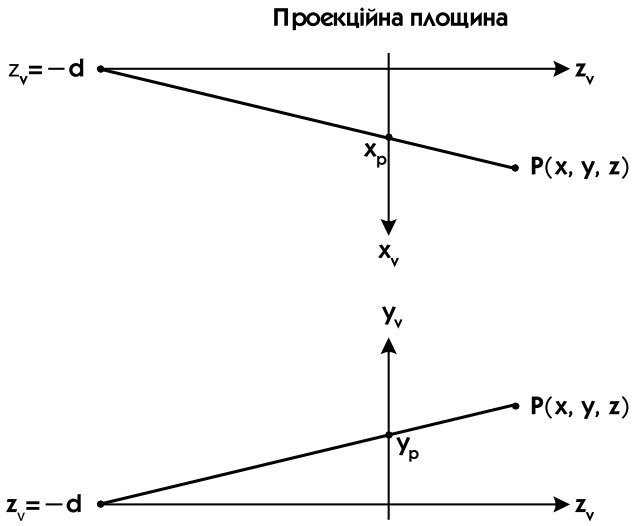


Рис.4.17. Інша схема побудови центральної проекції

З подібності трикутників випливає

$$\frac{X_P}{d} = \frac{x}{z+d}, \quad \frac{Y_P}{d} = \frac{y}{z+d}. \tag{4.24}$$

Звідси одержуємо

$$X_P = \frac{d \cdot x}{z+d} = \frac{x}{(z/d)+1}, \quad Y_P = \frac{d \cdot y}{z+d} = \frac{y}{(z/d)+1}. \tag{4.25}$$

Матриця записується у виді

$$M'_{\text{центр}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.26}$$

Цю матрицю можна одержати з матриці $M_{\text{центр}}$ шляхом переносу центра проекції в початок координат, застосування $M_{\text{центр}}$ і зворотнього переносу:

$$M'_{\text{центр}} = T\begin{pmatrix} 0 & 0 & d \end{pmatrix} \cdot M_{\text{центр}} \cdot T\begin{pmatrix} 0 & 0 & -d \end{pmatrix}. \tag{4.27}$$

Ортографічне проектування на площину z=0 очевидне. Напрямок проектування збігається з нормаллю до площини проекції, тобто в нашому випадку з віссю z.

Таким чином, точка P має координати:

$$X_P = x, \quad Y_P = y, \; Z_P = 0. \tag{4.28}$$

Ця проекція описується матрицею

$$M_{\text{орт}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (4.29)$$

Розглянемо тепер косокутну проекцію, матриця якої може бути записана виходячи зі значень *a* і *l* (рис.4.18).

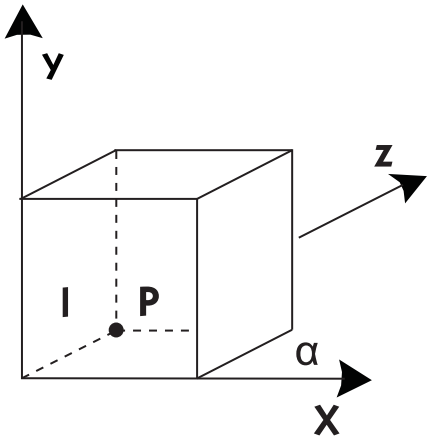


Рис.4.18. Косокутна рівнобіжна проекція одиничного куба.
Точка P' є проекцією точки P (0, 0, 1)

На рис.4.18 зображений одиничний куб, спроектований на xy-площину. З малюнка видно, що проекцією точки P (0, 0, 1), що знаходиться на задній стороні одиничного куба, є точка P'($l$cos$a$, $l$sin$a$, 0), що належить площині xy. По визначенню це означає, що напрям проектування збігається з відрізком PP', що проходить через ці дві точки (рис.4.19). Цей напрям є P'-P= ($l$cos$a$, $l$sin$a$, -1). Напрям проектування складає кут β із площиною xy.

Тепер розглянемо довільну точку x, y, z і визначимо її косокутну проекцію (Xp,Yp) на площину xy. На рис.4.20 показані два зображення точки і проектор, що рівнобіжний проектору, приведеному на рис.4.19. Рівняння для x- і y-координат проектора як функцій z мають вид $x=nz+a$, $y=mz+b$. Вирішуючи два рівняння относительно Xp і Yp, відзначених на рис.4.20, одержуємо

$$X_P = x + z \cdot l \cos a, \quad Y_P = y + z \cdot l \sin a. \qquad (4.30)$$

Матриця розміром 4×4, що виконує ці дії і, отже, описує косокутну проекцію, має вигляд

$$M_{\text{кос}} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ l\cos a & l\sin a & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (4.31)$$

Застосування матриці $M_{кос}$ приведе до зсуву і наступного проектування об'єкта: площини з постійною координатою $z=z1$ переносяться в напрямку x на $z1*l\cos a$ в напрямку y на $z1*l\sin a$ і потім проектуються на площину $z=0$. Зсув зберігає паралельність прямих, а також кути і відстані в площинах, паралельних осі z.

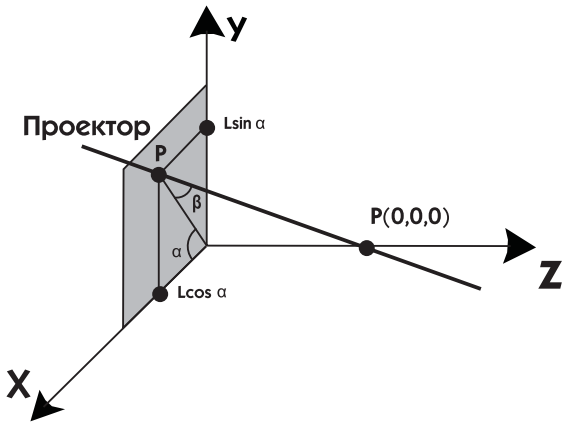


Рис.4.19. Косокутна паралельна проекція P'($l\cos a$, $l\sin a$, 0) є проекцією точки P (0, 0, 1).

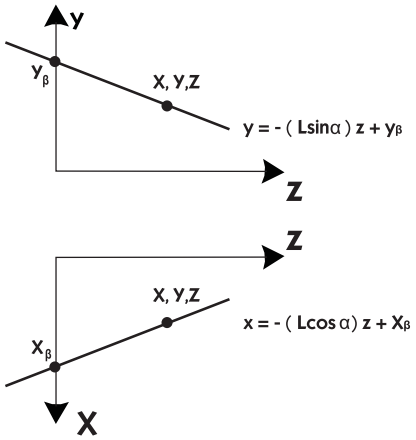


Рис.4.20. Косокутна паралельна проекція (Xp, Yp, 0) точки (x, y, z)

Для проекції кавальє l=1, тому кут β, показаний на рис.4.19, складає 45°.

Для проекції кабінт l=1/2, а β=artg (2) =63,4°. У випадку ортографічної проекції l=0 і β = 90°, тому $M_{орт}$ є окремим випадком $M_{кос}$.

Лекція 5

Растрова графіка. Представлення точок і зображень. Алгоритми креслення відрізків і кіл

5.1. Алгоритм креслення відрізків

Оскільки екран растрового дисплея з електронно-променевою трубкою (ЕПТ) можна розглядати як матрицю дискретних елементів (пікселів), кожний з яких може бути підсвічений, не можна безпосередньо провести відрізок з однієї точки в іншу. Процес визначення пікселів, якнайкраще апроксимуючих заданий відрізок, називається розкладанням у растр. У поєднанні з процесом порядкової візуалізації зображення він відомий як перетворення растрової розгортки. Для горизонтальних, вертикальних і нахилених під кутом 45° відрізків вибір растрових елементів очевидний. При іншій орієнтації вибрати потрібні піксели складніше (рис.5.1).

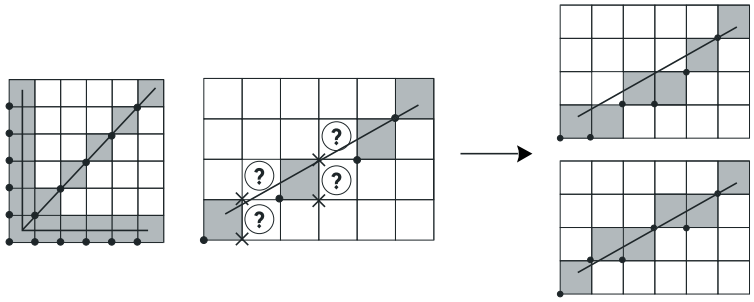


Рис.5.1. Розкладання в растр відрізків прямих

Загальні вимоги до алгоритмів креслення відрізків наступні: відрізки повинні виглядати прямими, починатися і закінчуватися в заданих точках, яскравість уздовж відрізка повинна бути постійною і не залежати від довжини і нахилу, малювати потрібно швидко. Постійна уздовж всього відрізка яскравість досягається лише при проведенні горизонтальних, вертикальних і нахилених під кутом 45° прямих. Для всіх інших орієнтація розкладання в растр приведе до нерівномірності яскравості, як це показано на рис.5.1. У більшості алгоритмів креслення відрізків для спрощення обчислень використовується покроковий алгоритм. Наведемо приклад:

Простий покроковий алгоритм

позиція=початок
крок=збільшення
1 **if** позиція - кінець<точність **then** 4
    if позиція>кінець **then** 2
    if позиція<кінець **then** 3
2 позиція=позиція - крок
    go to 1
3 позиція=позиція+крок
    go to 1
4 **finish**

Простий алгоритм розкладання відрізка в растр, описаний у наступному розділі, ілюструє застосування покрокових методів.

5.2. Цифровий диференціальний аналізатор

Один з методів розкладання відрізка в растр полягає в розв'язуванні диференціального рівняння, що описує цей процес. Для прямої лінії маємо

$$dy/dx=const \text{ чи } Dy/Dx=(y_2-y_1)/(x_2-x_1)$$

Розв'язок представляється у вигляді

$$y_{i+1}=y_i+Dy,\ y_{i+1}=y_i+Dx(y_2-y_1)/(x_2-x_1) \qquad (5.1)$$

де x_1, y_1 і x_2, y_2 - кінці відрізка, що розкладається, і y_i - початкове значення для чергового кроку вздовж відрізка. Фактично рівняння (5.1) являє собою рекурентне співвідношення для послідовних значень у вздовж потрібного відрізка.

Цей метод, що використовується для розкладання в растр відрізків, називається цифровим диференціальним аналізатором (ЦДА).

У простому ЦДА або Dx, або Dy (більше із збільшень) вибирається як одиниця растру. Нижче наводиться простий алгоритм, що працює у всіх квадрантах:

Процедура розкладання в растр відрізка за методом цифрового диференціального аналізатора (ЦДА)

передбачається, що кінці відрізка (x_1, y_1) і (x_2, y_2) не збігаються

Integer - функція перетворення дійсного числа в ціле.

Sign - функція, що повертає -1, 0, 1 для від'ємного, нульового і додатнього аргументу відповідно

апроксимуємо довжину відрізка

if abs(x2-x1)>=abs(y2-y1) **then** Довжина=abs(x2-x1)
else Довжина=abs(y2-y1)
end if

назначаємо більше із збільшень Dx чи Dy рівним одиниці растра

Dx=(x2-x1)/Довжина
Dy=(y2-y1)/Довжина

заокруглюємо величини, а не відкидаємо дробову частину

використання знакової функції робить алгоритм придатним для всіх квадрантів

x=x1+0.5***Sign**(Dx)
y=y1+0.5***Sign**(Dy)

початок основного циклу

```
i=1
while (i<=Довжина)
        Plot (Integer(x), Integer(y))
        x=x+Dx
        y=y+Dy
        i=i+1
end while
finish
```

5.3. Приклад, що ілюструє роботу алгоритму цифрового диференціального аналізатора в першому квадранті

Розглянемо відрізок із точки (0,0) у точку (5,5). Використовуємо ЦДА для розкладання цього відрізка в растр.

Результати роботи алгоритму такі:
початкові установки
x1=0
y1=0
x2=5
y2=5
Довжина=5
Dx=1
Dy=1
x=0,5
y=0,5
результати роботи покрокового циклу

i	Plot	x	y
		0,5	0,5
1	(0,0)		
		1,5	1,5
2	(1,1)		
		2,5	2,5
3	(2,2)		
		3,5	3,5
4	(3,3)		
		4,5	4,5
5	(4,4)		
		5,5	5,5

Отримане растрове представлення відрізка зображене на рис.5.2.

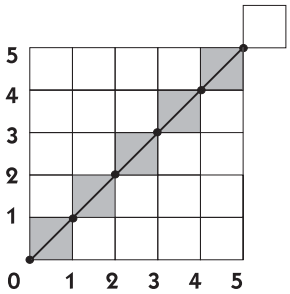


Рис.5.2. Результати роботи простого ЦДА в першому квадранті

Зауважимо, що кінцеві точки визначені точно, обрані піксели рівномірно розподілені вздовж відрізка. Однак якщо початковим значенням змінної і взяти нуль замість одиниці, то активізується піксел з координатами (5,5), що небажано. Якщо адреса піксела задана цілими координатами лівого нижнього кута, то активація цього піксела дасть явно невірну кінцеву точку відрізка (рис.5.2). До того ж при побудові серії послідовних відрізків піксел (5,5) буде активований двічі: наприкінці даного відрізка і на початку наступного. Такий піксел може виглядати як більш яскравий чи мати інший колір. Наведемо приклад, що ілюструє роботу алгоритму в третьому квадранті.

5.4. Приклад, що ілюструє роботу алгоритму цифрового диференціального аналізатора в третьому квадранті

Розглянемо відрізок із точки (0,0) у точку (-8,-4) в третьому квадранті. Використовуємо ЦДА для розкладання цього відрізка в растр. Результати роботи алгоритму такі:

початкові установки

x1=0

y1=0

x2=-8

y2=-4

Довжина=8

Dx=-1

Dy=-0,5

x=-0,5

y=-0,5

результати роботи покрокового циклу

i	Plot	x	y
		-0,5	-0,5
1	(-1,-1)		
		-1,5	-1,0
2	(-2,-1)		
		-2,5	-1,5
3	(-3,-2)		
		-3,5	-2,0
4	(-4,-2)		
		-4,5	-2,5
5	(-5,-3)		
		-5,5	-3,0
6	(-6,-3)		
		-6,5	-3,5
7	(-7,-4)		
		-7,5	-4,0
8	(-8,-4)		
		-8,5	-4,5

Результати, представлені на рис.5.3, виглядають цілком прийнятними.

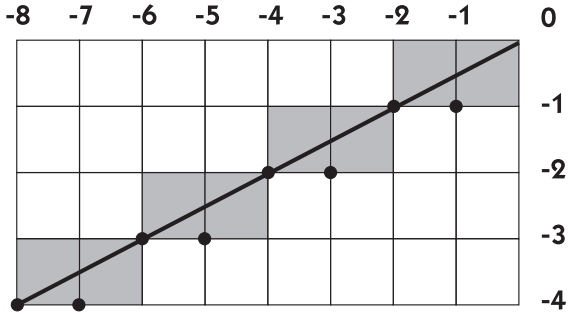


Рис.5.3. Результати роботи простого ЦДА в третьому квадранті

Проте аналіз відрізків, проведених із точки (0,0) у точку (-8,4) і (8,-4), показує, що розкладений у растр відрізок лежить по одну сторону від реального і що на одному з кінців відрізка з'являється зайва точка, тобто результат роботи алгоритму залежить від орієнтації. Отже точність у кінцевих точках зменшується.

У наступному розділі описаний більш прийнятний алгоритм.

5.5. Алгоритм Брезенхема

Хоча алгоритм Брезенхема був спочатку розроблений для цифрових графопобудовувачів, однак він в тій самій мірі підходить для растрових пристроїв з ЕПТ. Алгоритм вибирає оптимальні растрові координати для представлення відрізка. У процесі роботи одна з координат - або x, або y (в залежності від кутового коефіцієнта) - змінюється на одиницю. Зміна іншої координати (на 0 чи 1) відбувається залежно від відстані між дійсним положенням відрізка і найближчих координат сітки. Таку відстань ми назвемо похибкою.

Алгоритм побудований так, що потрібно перевірити лише знак цієї похибки. На рис.5.4 це ілюструється для відрізка в першому октанті, тобто для відрізка з кутовим коефіцієнтом, що лежить у діапазоні від 0 до 1. З малюнка видно що, якщо кутовий коефіцієнт відрізка з точки (0,0) більший, ніж 1/2, то перетин з прямою x=1 буде розташований ближче до прямої y=1, ніж до прямої y=0. Отже, точка растра (1,1) краще апроксимує хід відрізка, ніж точка (1,0). Якщо кутовий коефіцієнт менше 1/2, то навпаки. для кутового коефіцієнта, що дорівнює 1/2, немає кращого вибору. У даному випадку алгоритм вибирає точку (1,1).

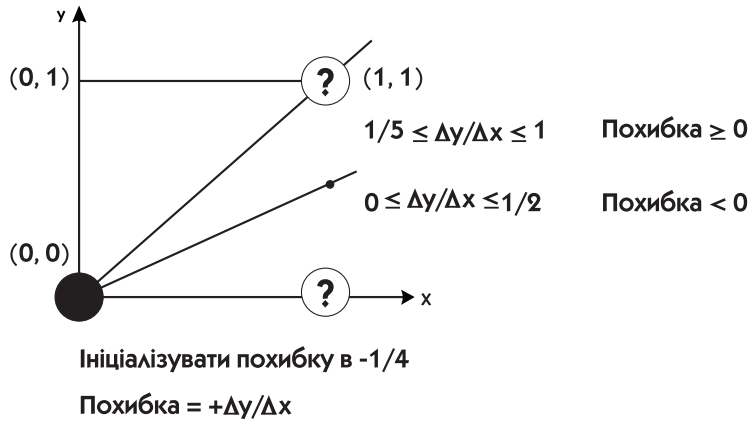


Рис.5.4. Основна ідея алгоритму Брезенхема

Не усі відрізки проходять через точки растра. Подібна ситуація показана на рис.5.5, де відрізок з тангенсом кута нахилу 3/8 спочатку проходить через точку растра (0,0) і послідовно перетинає три піксели. Також ілюструється обчислення похибки при представленні відрізка дискретними пікселами.

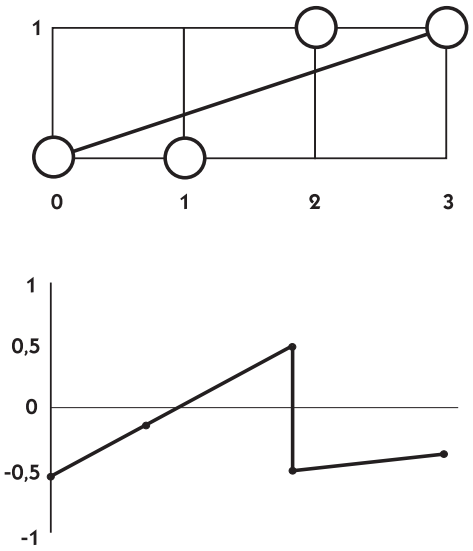


Рис.5.5. Графік похибки в алгоритмі Брезенхема

Через те що бажано перевіряти тільки знак похибки, спочатку вона встановлюється рівною -1/2. Таким чином, якщо кутовий коефіцієнт відрізка дорівнює чи більший 1/2, величина похибки в наступній точці растра з координатами (1,0) може бути обчислена як

$$e=e+m$$

де m - кутовий коефіцієнт. У нашому випадку при початковому значенні похибки -1/2

$$e=-1/2+3/8=-1/8$$

Через те що e від'ємне, відрізок пройде нижче середини піксела. Отже, піксел на тому ж самому горизонтальному рівні краще апроксимує положення відрізка, тому y не збільшується. Аналогічно обчислюємо похибку

$$e=-1/8+3/8=1/4$$

у наступній точці растра (2,0). Тепер e додатне, значить відрізок пройде вище середньої точки. Растровий елемент (2,1) з наступною по величині координатою y краще апроксимує положення відрізка. Отже, y збільшується на 1. Перше ніж розглядати наступний піксел, необхідно відкоректувати похибку відніманням від неї 1. Маємо

$$e=1/4-1=-3/4$$

Зауважимо, що перетин вертикальної прямої x=2 із заданим відрізком лежить на 1/4 нижче прямої y=1. Якщо ж перенести відрізок на 1/2 вниз, ми одержимо саме величину -3/4. Продовження обчислень для наступного піксела дає

$$e=-3/4+3/8=-3/8$$

Через те що e від'ємне, y не збільшується. З усього сказаного випливає, що похибка - це інтервал, що відтинається по осі y розглянутим відрізком у кожному растровому елементі (відносно -1/2).

Блок-схема алгоритму на рис.5.6. Приклад наведений нижче.

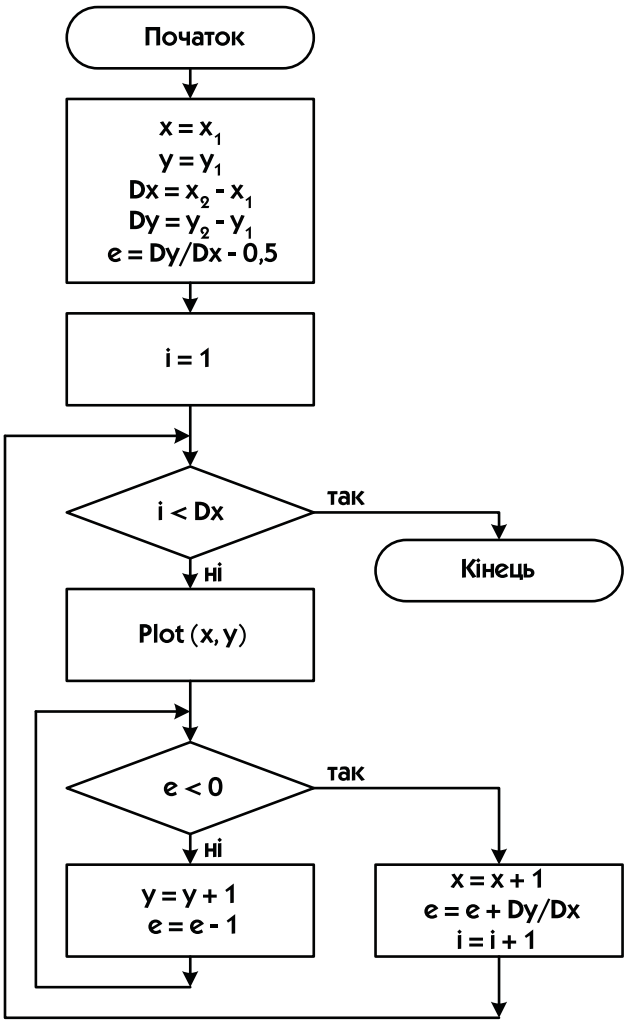


Рис.5.6. Блок-схема алгоритму Брезенхема

Наведемо алгоритм Брезенхема для першого октанта, тобто для випадку $0 \le Dy \le Dx$.

Алгоритм Брезенхема розкладання в растр відрізка для першого октанта

припускається, що кінці відрізка (x_1, y_1) і (x_2, y_2) не збігаються

Integer - функція перетворення в ціле

x, y, Dx, Dy - цілі

e - дійсне

ініціалізація змінних

x=x1
y=y1
Dx=x2-x1
Dy=y2-y1
Ініціалізація з виправленням на половину піксела
e=Dy/Dx-1/2
початок основного циклу

```
for i=1 to Dx
    plot (x,y)
    while (e=>0)
        y=y+1
        e=e-1
    end while
    x=x+1
    e=e+Dy/Dx
next i
finish
```

5.6. Приклад роботи алгоритму Брезенхема в першому октанті

Розглянемо відрізок, проведений із точки (0,0) у точку (5,5). Розклад відрізка в растр за алгоритмом Брезенхема приводить до такого результату:
початкові установки
x=0
y=0
Dx=5
Dy=5
e=1-1/2=1/2
результати роботи покрокового циклу

i	Plot	e	x	y
		1/2	0	0
1	(0,0)			
		-1/2	0	1
		1/2	1	1
2	(1,1)			
		-1/2	1	2
		1/2	2	2
3	(2,2)			
		-1/2	2	3
		1/2	3	3
4	(3,3)			
		-1/2	3	4
		1/2	4	4
5	(4,4)			
		-1/2	5	5
		1/2	5	5

Результат показаний на рис.5.7 і збігається з очікуваним. Зауважимо, що точка растра з координатами (5,5) не активована. Цю точку можна

активувати шляхом зміни циклу for-next на 0 to Dx. Активацію точки (0,0) можна усунути, якщо поставити оператор Plot безпосередньо перед рядком next i.

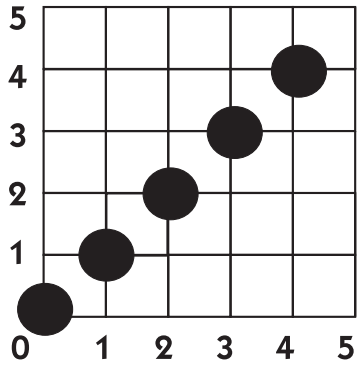


Рис.5.7. Результат роботи алгоритму Брезенхема в першому октанті.

У наступному розділі описаний загальний алгоритм Брезенхема.

5.7. Узагальнений алгоритм Брезенхема

Щоб реалізація алгоритму Брезенхема була повною необхідно розглянути відрізки у всіх октантах. Модифікацію легко зробити, враховуючи в алгоритмі номер квадранта, в якому лежить відрізок і його кутовий коефіцієнт. Коли абсолютна величина кутового коефіцієнта більше 1, *y* постійно змінюється на одиницю, а критерій похибки Брезенхема використовується для прийняття рішення про зміну величини *x*. Вибір координати, що постійно змінюється (на +1 чи -1), залежить від квадранта (рис.5.8).

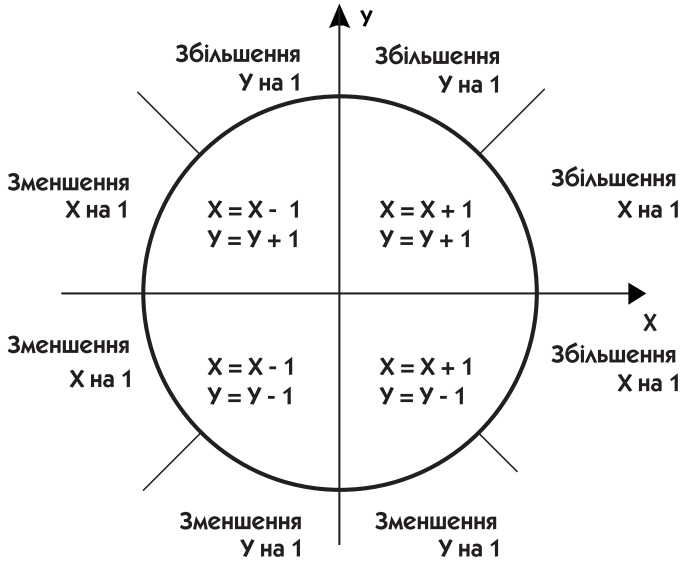


Рис.5.8. Розгляд випадків для узагальненого алгоритму Брезенхема

Узагальнений цілочисельний алгоритм Брезенхема для квадрантів може бути оформлений у наступному вигляді:

припускається, що кінці відрізка (x_1, y_1) і (x_2, y_2) не збігаються

усі змінні вважаються цілими

Sign - функція, що повертає -1, 0, 1 для від'ємного, нульового і додатнього аргумента відповідно

```
ініціалізація змінних
x=x1
y=y1
Dx=abs(x2-x1)
Dy=abs(y2-y1)
s1=Sign(x2-x1)
s2=Sign(y2-y1)
обмін значень Dx і Dy в залежності від кутового коефіцієнта нахилу
відрізка
if Dy<Dx then
   Temp=Dx
   Dx=Dy
   Dy=Temp
   Обмін=0
else Обмін=1
end if
ініціалізація e з виправленням на половину піксела
e=2*Dy-Dx
основний цикл
for i=1 to Dx
      Plot(x,y)
      while(e=>0)
            if Обмін=1 then x=x+s1
            else y=y+s2
            end if
            e=e-2*Dx
      end while
      if Обмін=1 then y=y+s2
      else x=x+s1
      end if
      e=e+2*Dy
next i
finish
```

5.8. Приклад, що ілюструє роботу узагальненого алгоритму Брезенхема

Для ілюстрації розглянемо відрізок із точки (0, 0) у точку (-8, -4).

```
початкові установки
x=0
y=0
Dx=8
```

Dy=4
s1=-1
s2=-1
Обмін=0
e=0
результати роботи покрокового циклу

i	Plot	e	x	y
		0	0	0
1	(0,0)			
		-16	0	-1
		-8	-1	-1
2	(-1,-1)			
		0	-2	-1
3	(-2,-1)			
		-16	-2	-2
		-8	-3	-2
4	(-3,2)			
		0	-4	-2
5	(-4,2)			
		-16	-4	-3
		-8	-5	-3
6	(-5,-3)			
		0	-6	-3
7	(-6,-3)			
		-16	-6	-4
		-8	-7	-4
8	(-7,-4)			
		0	-8	-4

На рис.5.9 показаний результат. Порівнюючи з рис.5.3 бачимо, що результати роботи двох алгоритмів відрізняються.

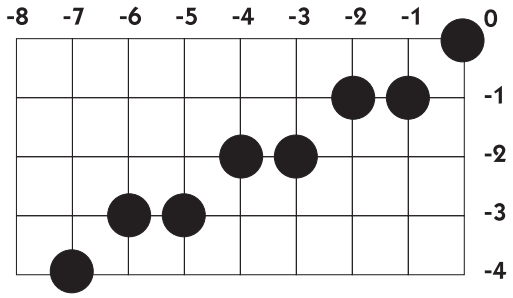


Рис.5.9. Результат роботи алгоритму Брезенхема в третьому квадранті

5.9. Алгоритм Брезенхема для генерації кола

У растр потрібно розкладати не тільки лінійні, але й інші, більш складні функції. Розкладанню конічних перетинів, тобто кіл, еліпсів, парабол, гіпербол, було присвячено значне число робіт. Найбільша увага, зрозуміло, була приділена колу. Один з найбільш ефективних і простих для

розуміння алгоритмів генерації кола належить Брезенхему. Для початку зауважимо, що необхідно згенерувати тільки одну восьму частину кола. Інші його частини можуть бути отримані послідовними відображеннями, як це показано на рис. 5.10. Якщо згенерований перший октант (від 0 до 45° проти годинникової стрілки), то другий октант можна отримати дзеркальним відображенням відносно прямої y=x, що дає в сукупності перший квадрант. Перший квадрант відбивається відносно прямої x=0 для отримання відповідної частини кола в другому квадранті. Верхнє півколо відбивається відносно прямої y=0 для завершення побудови. На рис.5.10 наведені двовимірні матриці відповідних перетворень.

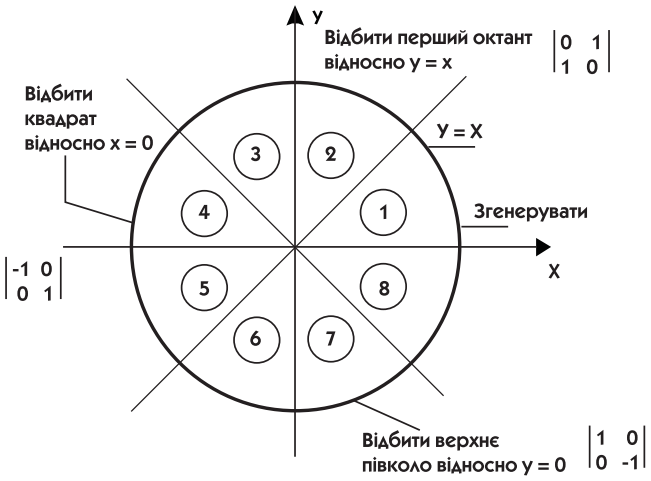


Рис.5.10. Генерація повного кола з дуги в першому октанті

Для пояснення алгоритму розглянемо першу чверть кола з центром у початку координат. Зауважимо, що якщо робота алгоритму починається в точці *x=0, y=R,* то при генерації кола за годинниковою стрілкою в першому квадранті *y* є монотонно спадною функцією аргумента *x* (рис.5.11). Аналогічно, якщо вихідною точкою є *y*=0, *x=R,* то при генерації кола проти годинникової стрілки *x* буде монотонно спадною функцією аргументу *y.*

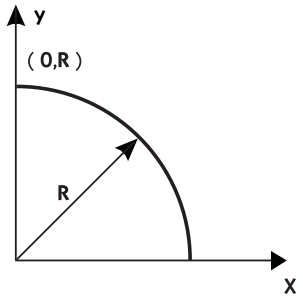


Рис. 5.11. Коло у першому квадранті

У нашому випадку вибирається генерація за годинниковою стрілкою з початком у точці $x=0$, $y=R$. Передбачається, що центр кола і початкова точка знаходяться точно в точках растра.

Для будь-якої заданої точки на колі при генерації за годинниковою стрілкою існує тільки три можливості вибрати наступний піксел, що щонайкраще наближає коло: горизонтально вправо, по діагоналі вниз і вправо, вертикально вниз. На рис.5.12 ці напрямки позначені відповідно m_H, m_D, m_V.

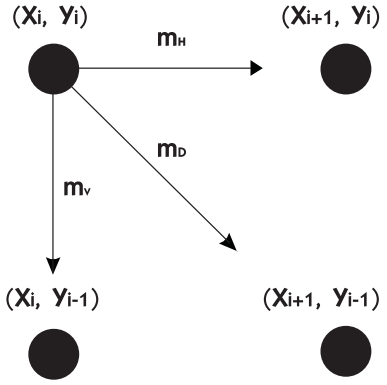


Рис. 5.12. Вибір пікселів у першому квадранті

Алгоритм вибирає піксел, для якого мінімальний квадрат відстані між одним з цих положень і колом, тобто мінімум з

$$m_H=|x_{i+1}^2+y_i^2-R^2|$$
$$m_D=|x_{i+1}^2+y_{i-1}^2-R^2|$$
$$m_V=|x_i^2+y_{i-1}^2-R^2|$$

Обчислення можна спростити, якщо зауважити, що в околі точки (xi, yi) можливі тільки п'ять типів перетинів кола і сітки растра, наведених на рис.5.13.

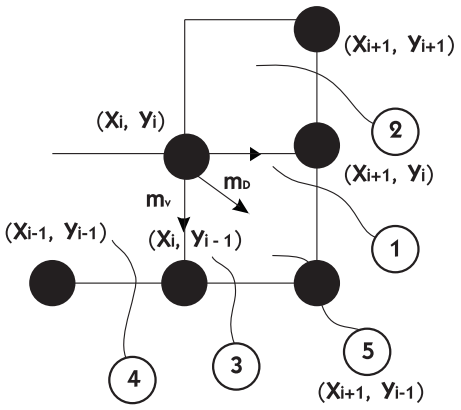


Рис. 5.13. Перетини кола і сітки растра

Різниця між квадратами відстаней від центра кола до діагонального піксела (x_{i+1}, y_{i-1}) і від центра до точки на колі R^2 дорівнює

$$D_i=x_{i+1}^2+y_{i-1}^2-R^2$$

Як і в алгоритмі Брезенхема для відрізка, для вибору відповідного піксела бажано використовувати тільки знак похибки, а не її величину.

При $D_i<0$ діагональна точка (x_{i+1}, y_{i-1}) знаходиться всередині реального кола, тобто це випадки 1 або 2 на рис.5.13. Ясно, що в цій ситуації варто вибрати або піксел (x_{i+1}, y_i), тобто m_H, або піксел (x_{i+1}, y_{i-1}), тобто m_D. Для цього спочатку розглянемо випадок 1 і перевіримо різницю квадратів відстаней від кола до пікселів у горизонтальному і діагональному напрямках:

$$d=|x_{i+1}^2+y_i^2-R^2|-|x_{i+1}^2+y_{i-1}^2-R^2|.$$

При $d<0$ відстань від кола до діагонального піксела більша, ніж до горизонтального. Навпаки, якщо $d>0$, відстань до горизонтального піксела більша. Таким чином,

при $d<=0$ вибираємо m_H (x_{i+1}, y_i);

при $d>0$ вибираємо m_D (x_{i+1}, y_{i-1}).

При $d=0$, коли відстань від кола до обох пікселів однакова, вибираємо горизонтальний крок. Кількість обчислень, необхідних для оцінки величини d, можна скоротити, якщо зауважити, що у випадку 1

$$x_{i+1}^2+y_i^2-R^2>=0,$$
$$x_{i+1}^2+y_{i-1}^2-R^2<0,$$

тому що діагональний піксел (x_{i+1}, y_{i-1}) завжди лежить всередині кола, а горизонтальний (x_i+1, y_i) - поза ним. Таким чином, можна обчислити d:

$$d=x_{i+1}^2+y_i^2-R^2+x_{i+1}^2+y_{i-1}^2-R^2.$$

Доповнення до повного квадрата члена y_i^2 за допомогою додавання і віднімання - $2y_i+1$ дає

$$d=2[x_{i+1}^2+y_{i-1}^2-R^2]+2y_{i-1}.$$

У квадратних дужках стоїть за визначенням D_i і його підстановка

$$d=2(D_i+y_i)-1$$

значно спрощує вираз.

Розглянемо випадок 2 на рис.5.13 і зауважимо, що тут повинен бути обраним горизонтальний піксел (x_{i+1}, y_i), тому що у є монотонно спадною функцією. Перевірка компонентів d показує, що

$$x_{i+1}^2+y_i^2-R^2<0,$$
$$x_{i+1}^2+y_{i-1}^2-R^2<0,$$

оскільки у випадку 2 горизонтальний (x_{i+1}, y_i) і діагональний (x_{i+1}, y_{i-1}) піксели лежать всередині кола. Отже, $d<0$, і при використанні того ж самого критерію, що й у випадку 1, вибирається піксел (x_{i+1}, y_i).

Якщо $D_i>0$, то діагональна точка (x_{i+1}, y_{i-1}) знаходиться поза колом, тобто це випадки 3 і 4 на рис.5.13. У даній ситуації зрозуміло, що потрібно вибрати піксел (x_{i+1}, y_{i-1}), або (x_i, y_{i-1}). Аналогічно до попереднього випадку критерій вибору можна отримати, розглядаючи спочатку випадок 3 і перевіряючи різницю між квадратами відстаней від кола до діагонального m_D і вертикального m_V пікселів, тобто

$$d'=|x_{i+1}^2+y_{i-1}^2-R^2|-|x_i^2+y_{i-1}^2-R^2|.$$

При $d'<0$ відстань від кола до вертикального піксела (x_i, y_{i-1}) більша і варто вибрати діагональний крок до піксела (x_{i+1}, y_{i-1}). Навпаки, у випадку $d'>0$ відстань від кола до діагонального піксела більша і варто вибрати вертикальний рух до піксела (x_i, y_{i-1}). Таким чином,

при $d'<=0$ вибираємо m_D (x_{i+1}, y_{i-1}),

при $d'>0$ вибираємо m_V (x_i, y_{i-1}).

Тут для випадку $d'=0$, тобто коли відстані рівні, обраний діагональний крок. Перевірка компонентів d' показує, що

$$x_i^2+y_{i-1}^2-R^2>=0$$
$$x_{i+1}^2+y_{i-1}^2-R^2<0$$

оскільки для випадку 3 діагональний піксел (x_{i+1}, y_{i-1}) знаходиться поза колом, в той час як вертикальний піксел (x_i, y_{i-1}) лежить всередині кола. Це дозволяє записати d' у вигляді

$$d'=x_{i+1}^2+y_{i-1}^2-R^2+x_i^2+y_{i-1}^2-R^2$$

Доповнення до повного квадрата члена x_i^2 за допомогою додавання і віднімання $2x_{i+1}$ дає

$$d'=2[x_{i+1}^2+y_{i-1}^2-R^2]-2x_{i-1}$$

Використання визначення D_i приводить вираз до вигляду

$$d'=2(D_i-x_i)-1$$

Тепер, розглядаючи випадок 4, знову зауважимо, що варто вибрати вертикальний піксел (x_i, y_{i-1}), тому що у є монотонно спадною функцією від *x*. Перевірка компонентів d' для випадку 4 показує, що

$$x_{i+1}^2+y_{i-1}^2-R^2>0,$$
$$x_i^2+y_{i-1}^2-R^2>0,$$

оскільки обидва піксели знаходяться поза колом. Отже, $d'>0$ і при використанні критерію, розробленого для випадку 3, відбувається вірний вибір m_V.

Залишилося перевірити тільки випадок 5 на рис.5.13, який зустрічається, коли діагональний піксел (x_{i+1}, y_{i-1}) лежить на колі, тобто $D_i=0$. Перевірка компонентів d показує, що

$$x_{i+1}^2+y_i^2-R^2>0$$
$$x_{i+1}^2+y_{i-1}^2-R^2=0$$

Отже, $d>0$ і вибирається діагональний піксел (x_{i+1}, y_{i-1}).

Аналогічним чином оцінюємо компоненти d':

$$(x_{i+1})^2+(y_{i-1})^2-R^2=0$$
$$(x_{i+1})^2+(y_{i-1})^2-R^2<0$$

і $d'<0$, що є умовою вибору правильного діагонального кроку до (x_{i+1}, y_{i-1}). Таким чином, випадок $D_i=0$ підлягає тому ж критерію, що і випадок $D_i<0$ чи $D_i>0$. Підіб'ємо підсумок отриманих результатів:

$D_i<0$

$d<=0$ вибираємо піксел (x_{i+1}, y_i)-m_H

$d>0$ вибираємо піксел (x_{i+1}, y_{i-1})-m_D

$D_i>0$

$d'<=0$ вибираємо піксел (x_{i+1}, y_{i-1})- m_D

$d'>0$ вибираємо піксел (x_i, y_{i-1})-m_V

$D_i=0$ вибираємо піксел (x_{i+1}, y_{i-1})-m_D

Легко розробити прості рекурентні співвідношення для реалізації покрокового алгоритму. Спочатку розглянемо горизонтальний крок m_H до піксела (x_i+1, y_i). Позначимо це нове положення піксела як (i+1). Тоді координати нового піксела і значення D_i рівні

$x_{i+1}=x_i+1$

$y_{i+1}=y_i$

$D_{i+1}=(x_{i+1}+1)^2+(y_{i+1}-1)^2-R^2=D_i+2x_{i+1}+1$

Аналогічно координати нового піксела і значення D_i для кроку m_D до піксела (x_i+1, y_i-1) такі:

$x_{i+1}=x_i+1$

$y_{i+1}=y_i-1$

$D_{i+1}=D_i+2x_{i+1}-2y_{i+1}+2$

Те ж саме для кроку m_V до (x_i, y_i-1)

$x_{i+1}=x_i$

$y_{i+1}=y_i-1$

$D_{i+1}=D_i-2y_{i+1}+1$

Покроковий алгоритм Брезенхема на псевдокоді для генерації кола в першому квадранті:

усі змінні - цілі
ініціалізація змінних
  x_i=0
  y_i=R
  D_i=2(1-R)
  Границя=0
1 **Plot** (x_i, y_i)
  if y_i<= Границя **then** 4
Виділення випадку 1 чи 2, 4 чи 5, чи 3
  if D_i<0 **then** 2
  if D_i>0 **then** 3
  if D_i=0 **then** 20
визначення випадку 1 чи 2
2 d=2D_i+2y_i-1
  if d<=0 **then** 10
  if d>0 **then** 20
визначення випадку 4 чи 5
3 d=2D_i+2x_i-1
  if d<=0 **then** 20
  if d>0 **then** 30
виконання кроків
крок до m
10 x_i=x_i+1
  D_i=D_i+2x_i+1
  go to 1
крок m
20 x_i=x_i+1
  y_i=y_i-1

$D_i=D_i+2x_i-2y_i+2$
go to 1
30 $y_i=y_i-1$
$D_i=D_i-2y_i+1$
go to 1
4 **finish**

Змінна границі встановлюється в нуль для закінчення роботи алгоритму на горизонтальній осі, в результаті генерується коло у першому квадранті. Якщо необхідний лише один з октантів, то другий октант можна отримати за допомогою установки Границя=Integer(R/sqrt(2)), а перший - за допомогою відображення другого октанта відносно прямої y=x (рис.5.10).

Блок-схема алгоритму показана на рис.5.14.

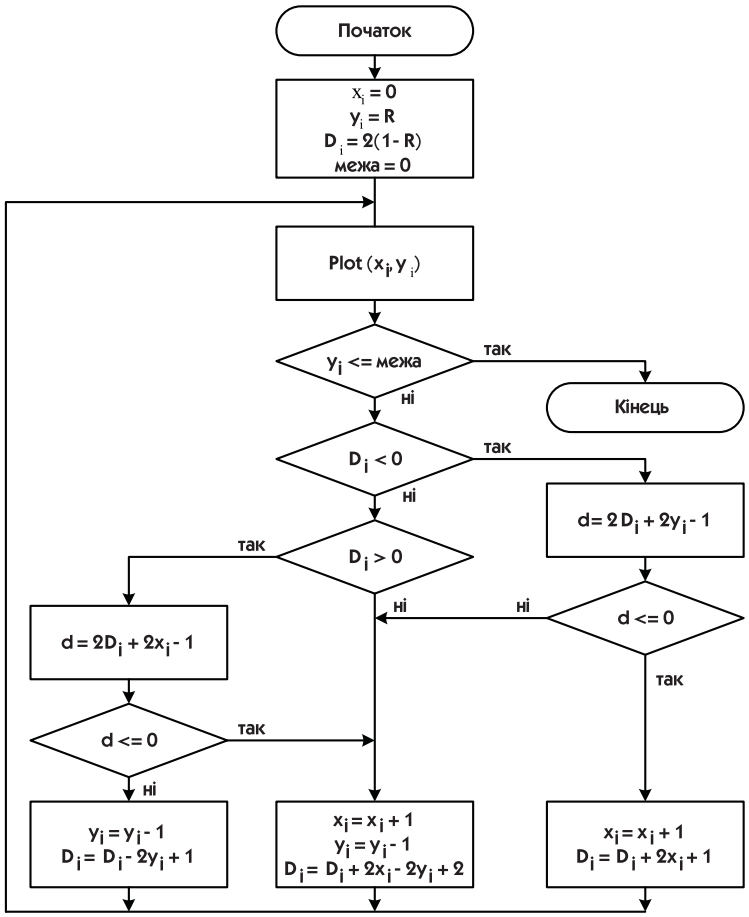


Рис.5.14. Блок-схема покрокового алгоритму Брезенхема для генерації кола в першому квадранті

5.10. Приклад, що ілюструє роботу алгоритму Брезенхема для кола

Розглянемо коло радіусом 8 з центром в початку координат. Генерується тільки перший квадрант.

початкові установки
x=0
y=8
Di=2(1-8)=-14
Межа=0
Покрокове виконання основного циклу

1 **Plot** (0,8)
yi>Межа (продовжувати)
Di<0 **go to** 2
2 d=2(-14)+2(8)-1=-13<0 go to 10
10 x=0+1=1
Di=-14+2+1=-11
go to 1
1 **Plot** (1,8)
yi>Межа (продовжувати)
Di<0 **go to** 2
2 d=2(-11)+2(8)-1=-7<0 go to 10
10 x=1+1=1
Di=-11+2(2)+1=-6
go to 1
1 **Plot** (2,8)
... Продовжувати.

Результати показані на рис.5.15 разом з реальним колом.

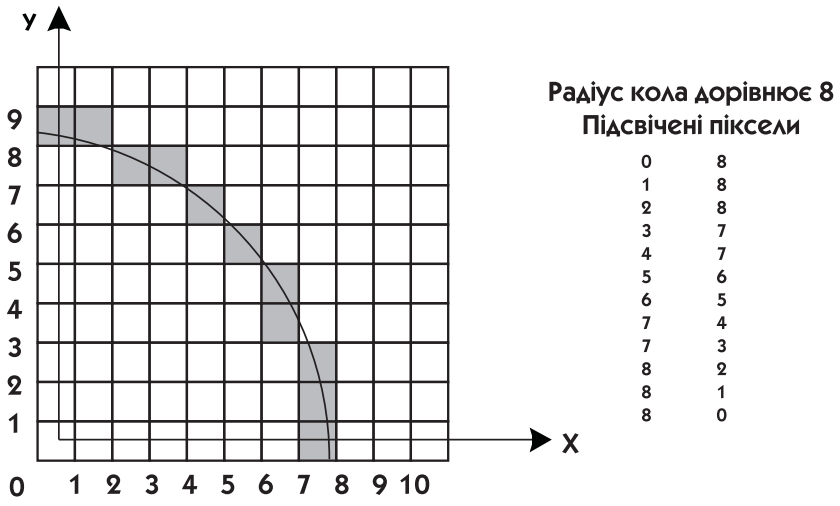


Рис.5.15. Результати роботи алгоритму Брезенхема генерації кола

Результати всіх послідовних проходів алгоритму зведені в таблицю. Список пікселів обраних алгоритмів складається з (0,8), (1,8), (2,8), (3,7), (4,7), (5,6), (6,5), (7,4), (7,3), (8,2), (8,1), (8,0).

Plot	Di	d	d’	x	y
	-14			0	8
(0,8)					
	-11	-13		1	8
(1,8)					
	-6	-7		2	8
(2,8)					
	-12	3		3	7
(3,7)					
	-3	11		4	7
(4,7)					
	-3	7		5	6
(5,6)					
	1	5		6	5
(6,5)					
	9		-11	7	4
(7,4)					
	4		3	7	3
(7,3)					
	18		-7	8	2
(8,2)					
	17		19	8	1
(8,1)					
	18		17	8	0
(8,0)					

Алгоритм легко узагальнюється для інших квадрантів чи дуг кіл.
Алгоритм завершено.

Лекція 6

Растрова розгортка суцільних областей. Алгоритми заповнення контурів. Алгоритми зафарбування багатокутників

6.1. Растрова розгортка суцільних областей

Однією з унікальних характеристик растрового графічного пристрою є можливість представлення суцільних областей. Генерацію суцільних областей з простих описів ребер чи вершин будемо називати растровою розгорткою суцільних областей, заповненням багатокутників чи заповненням контурів. Для цього можна використовувати кілька методів, що звичайно поділяються на 2 категорії: растрова розгортка і затравочне заповнення.

У методах растрової розгортки намагаються визначити в порядку сканування рядків, чи лежить точка всередині багатокутника чи контура. Ці алгоритми звичайно йдуть від "верху" багатокутника до "низу".

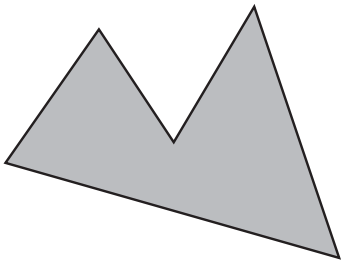


Рис.6.1. Штрихування чи зафарбовування контура.

У методах затравочного заповнення передбачається, що відомо деяку точку (запал) всередині замкненого контура. В алгоритмах шукають точки, сусідні з затравочною і розташовані всередині контура. Якщо сусідня точка розташована не всередині контура, це означає, що виявлена границя контура. Якщо ж точка розташована всередині контуру, то вона стає затравочною точкою і пошук продовжується рекурсивно. Подібні алгоритми застосовні тільки до растрових пристроїв.

6.2. Заповнення багатокутників

Багато замкнених контурів є простими багатокутниками. Якщо контур складається з кривих ліній, то його можна апроксимувати придатним багатокутником чи багатокутниками. Найпростіший метод заповнення багатокутника складається з перевірки на приналежність до внутрішньої частини багатокутника кожного піксела в растрі. Цей метод занадто марнотратний. Витрати можна зменшити шляхом обчислення для багатокутника прямокутної оболонки - найменшого багатокутника, що містить усередині себе контур. Як показано на рис.6.2 перевіряються тільки внутрішні точки цієї оболонки.

Використання прямокутної оболонки для багатокутника зображеного на рис.6.2(а), набагато скорочує число пікселів, що перевіряються. У той же час для багатокутника на рис.6.2(б) істотно скорочується число перевірок.

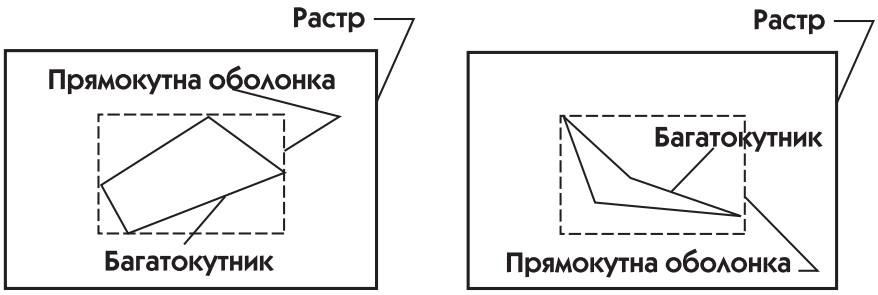


Рис.6.2. Прямокутна оболонка багатокутника

6.3. Растрова розгортка багатокутників

Можна розробити більш ефективний метод, ніж тест на приналежність до внутрішньої частини, якщо скористатися тим фактом, що сусідні піксели, ймовірно, мають однакові характеристики (крім пікселів граничних ребер). Ця властивість називається просторовою когерентністю. Для растрових графічних пристроїв сусідні піксели на скануючому рядку, ймовірно, мають однакові характеристики. Це когерентність растрових рядків. Характеристики пікселів на даному рядку змінюються тільки там, де ребро багатокутника перетинає рядок. Ці перетини поділяють скануючий рядок на області.

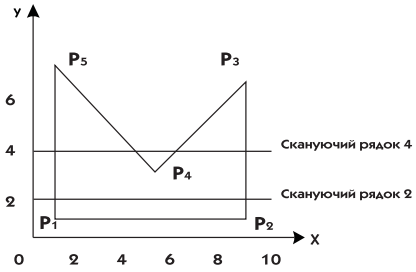


Рис.6.3. Растрова розгортка суцільної області

Для простого багатокутника на рис.6.3 рядок 2 перетинає багатокутник при x=1 і x=8. Одержуємо три області:

$x<1$ поза багатокутником
$1 \le x \le 8$ всередині багатокутника
$x>8$ поза багатокутником

Рядок 4 поділяється на п'ять областей:

$x<1$ поза багатокутником
$1 \le x \le 4$ всередині багатокутника
$4<x<6$ поза багатокутником
$6 \le x \le 8$ всередині багатокутника
$x>8$ поза багатокутником

Зовсім необов’язково, щоб точки перетину для рядка 4 відразу визначалися у фіксованому порядку (зліва направо). Наприклад, якщо

багатокутник задається списком вершин $P_1P_2P_3P_4P_5$, а список ребер - послідовними парами вершин P_1P_2, P_2P_3, P_3P_4, P_4P_5, P_5P_1 то для рядка 4 будуть знайдені наступні точки перетину з ребрами багатокутника: 8, 6, 4, 1. Ці точки треба відсортувати в порядку зростання по x, тобто одержати 1, 4, 6, 8.

При визначенні інтенсивності, кольору і відтінку пікселів на скануючому рядку розглядаються пари відсортованих точок перетинань. Для кожного інтервалу, що задається парою перетинань, використовується інтенсивність чи колір заповнюваного багатокутника. Для інтервалів між парами перетинань і крайніх (від початку рядка до першої точки перетину і від останньої точки перетину до кінця рядка) використовується фонова інтенсивність чи колір. На рис.6.3 для рядка 4 у фоновий колір встановлені піксели: від 0 до 1, від 4 до 6, від 8 до 10, тоді як піксели від 1 до 4 і від 5 до 8 зафарбовані в колір багатокутника.

Точне визначення тих пікселів, що повинні активуватися вимагає деякої обережності. Розглянемо простий прямокутник, що зображений на рис.6.4. Прямокутник має координати (1,1) (5,1), (5,4), (1,4). Скануючі рядки з 1 по 4 мають перетини з ребрами багатокутника при x=1 і 5. Нагадаємо, що піксел адресується координатами свого лівого нижнього кута, отже, для кожного з цих скануючих рядків будуть активовані піксели з x-координатами 1, 2, 3, 4 і 5. На рис.6.4(а) показаний результат. Зауважимо, що площа, що покривається активованими пікселами, дорівнює 20, у той час як дійсна площа прямокутника дорівнює 12.

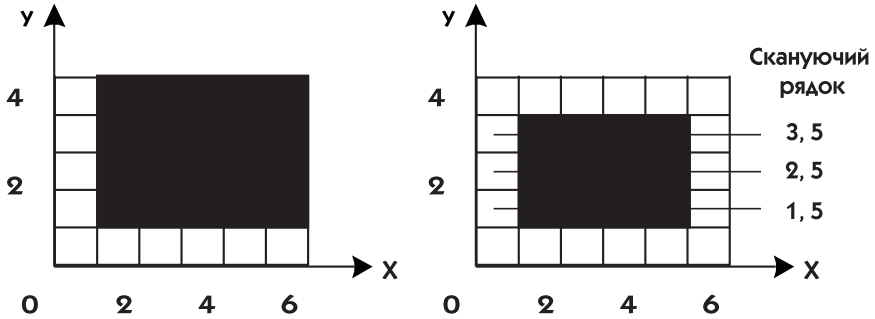


Рис. 6.4. Система координат рядків сканування

Модифікація системи координат скануючого рядка і тесту активації усуває цю проблему, як це показано на рис.6.4(б). Вважається, що скануючі рядки проходять через центри рядків пікселів, тобто через середини інтервалів, як це показано на рис.6.4(б). Тест активації модифікується в такий спосіб: перевіряється, чи лежить всередині інтервалу центр піксела, розташованого праворуч від перетину. Однак піксели все ще адресуються координатами лівого нижнього кута. Як показано на рис.6.4(б), результат даного методу коректний.

Горизонтальні ребра не можуть перетинати скануючий рядок і, таким чином, ігноруються. Це зовсім не означає, що їх немає на рисунку. Ці ребра формуються верхнім і нижнім рядками пікселів, як показано на рис.6.4.

Рис.6.4 ілюструє коректність верхнього і нижнього ребер багатокутника, отриманих у результаті модифікації системи координат скануючих рядків.

Додаткові труднощі виникають при перетині скануючого рядка і багатокутника точно по вершині, як це показано на рис.6.5.

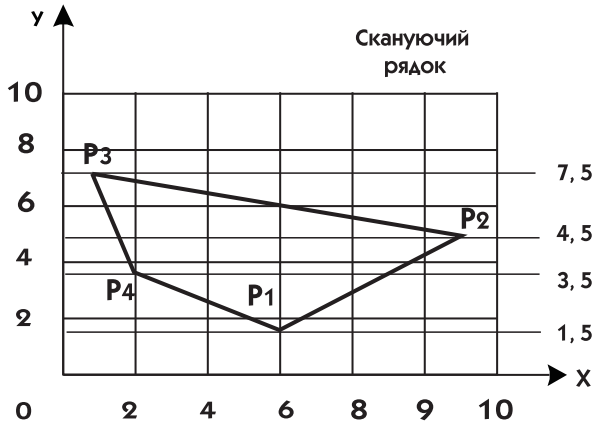


Рис.6.5. Особливості перетинів з рядками сканування

При використанні згоди про середину інтервалу між скануючими рядками одержуємо, що рядок y=3,5 перетне багатокутник у 2, 2 і 8, тобто отримається непарна кількість перетинань. Отже, розбивка пікселів на пари дасть невірний результат, тобто піксели (0, 3), (1, 3) і від (3, 3) до (7, 3) будуть фоновими, а піксели (2, 3), (8, 3), (9, 3) зафарбуються в колір багатокутника. Тут виникає ідея враховувати тільки одну точку перетинання з вершиною. Тоді для рядка y=3,5 одержимо правильний результат. Однак результат застосування методу до рядка y=1,5, що має два перетини в (5, 1), показує, що метод невірний. Для цього рядка саме розбивка на пари дасть вірний результат, тобто зафарбований буде тільки піксел (5, 1), Якщо ж враховувати у вершині тільки одне перетинання, то піксели від (0, 1) до (4, 1) будуть фоновими, а піксели від (5, 1) до (9, 1) будуть зафарбовані в колір багатокутника.

Правильний результат можна одержати, враховуючи точку перетинання у вершині два рази, якщо вона є точкою локального мінімуму чи максимуму і враховуючи її один раз в іншому випадку. Визначити локальний максимум чи мінімум багатокутника в розглянутій вершині можна за допомогою перевірки кінцевих точок двох ребер, з'єднаних у вершині. Якщо в обох кінців координати y більші, ніж у вершини, виходить, вершина є точкою локального мінімуму. Якщо менші, значить, вершина - точка локального максимуму. Якщо одна більша, а інша менша, то вершина не є ні точкою локального мінімуму, ні точкою локального максимуму. На рис.6.5 точка P_1 - локальний мінімум, P_3 - локальний максимум, а P_2, P_4 - ні те і ні інше. Отже, у точках P_1 і P_3 враховуються два перетинання із скануючими рядками, а в P_2 і P_4 - одне.

6.4. Простий алгоритм з упорядкованим списком ребер

Використовуючи описані вище методи, можна розробити ефективні алгоритми растрової розгортки суцільних областей, які називаються алгоритмами з упорядкованим списком ребер. Вони залежать від сортування в порядку сканування точок перетинань ребер багатокутника із скануючими рядками. Ефективність цих алгоритмів залежить від ефективності сортування. Наведемо дуже простий алгоритм.

Простий алгоритм з упорядкованим списком ребер

Підготувати дані:

Визначити для кожного ребра багатокутника точки перетинань із скануючими рядками, проведеними через середини інтервалів, для цього можна використовувати алгоритм Брезенхема чи ЦДА. Горизонтальні ребра ігноруються. Занести кожне перетинання (x, y+1/2) у список. Відсортувати список по рядках і по зростанню x у рядку; тобто (x_1, y_1) передує (x_2, y_2), якщо $y_1>y_2$ чи $y_1=y_2$ і $x_1x_2\leq x_2$.

Перетворити ці дані в растрову форму:

Виділити з відсортованого списку пари елементів (x_1, y_1) і (x_2, y_2). Структура списку гарантує, що $y=y_1=y_2$ і $x_1\leq x_2$. Активувати на скануючому рядку у піксели для цілих значень x, таких, що $x_1<x+1/2\leq x_2$.

6.5. Приклад, що ілюструє роботу простого алгоритму з упорядкованим списком ребер

Розглянемо багатокутник, зображений на рис.6.6. Його вершини: P_1, (1,1), P_2(8,1), P_3(8,6), P_4(5,3) і P_5(1,7). Перетинання із серединами скануючих рядків наступні:

скан. рядок 1.5 (8, 1.5), (1, 1.5)
скан. рядок 2.5 (8.2.5), (1, 2.5)
скан. рядок 3.5 (8, 3.5), (5.5, 3.5), (4.5, 3.5), (1, 3.5)
скан. рядок 4.5 (8, 4.5), (6.5, 4.5), (3.5, 4.5). (1, 4.5)
скан. рядок 5.5 (8, 5.5), (7.5, 5.5), (2.5, 5.5), (1, 5.5)
скан. рядок 6.5 (1.5, 6.5), (1, 6.5)
скан. рядок 7.5 немає

Весь список сортується в порядку сканування спочатку зверху вниз і потім - зліва направо: (1, 6.5), (1.5, 6.5), (1, 5.5), (2.5, 5.5),(7.5, 5.5), (8, 5.5), (1, 4.5), (3.5, 4,5). (6.5, 4.5). (8, 4.5). (1, 3.5). (4.5,3.5), (5.5, 3.5), (8, 3.5), (1, 2.5), (8, 2.5),(1, 1.5), (8, 1.5).

Виділяючи із списку пари перетинань і застосовуючи алгоритм, описаний вище, одержимо список пікселів, що повинні бути активовані:

(1,6)
(l, 5), (2, 5), (7, 5)
(1, 4), (2, 4), (3, 4), (6, 4), (7, 4)
(1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3)
(1,2), (2, 2), (3, 2) (4, 2), (5, 2), (6, 2), (7, 2) (1, 1), (2, 1), (3,1), (4,1), (5, 1), (6,1), (7, 1)

Результат представлений на рис.6.6.

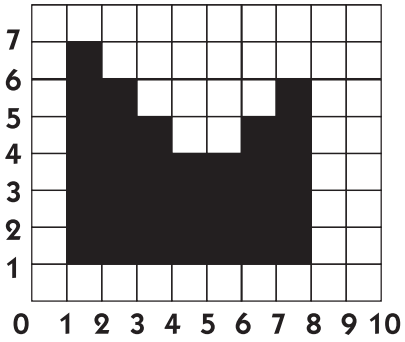


Рис.6.6. Результат растрової розгортки суцільної області, зображеної на рис.6.3

Зауважимо, що обидва вертикальних і нижнє ребро зображені правильно.

6.6. Більш ефективний алгоритм з упорядкованим списком ребер

В алгоритмі, описаному в попередньому розділі, генерується великий список, який необхідно цілком відсортувати. Алгоритм можна покращити, якщо підвищити ефективність сортування. Останнє досягається поділом сортування по рядках у напрямку y і сортування в рядку в напрямку x за допомогою групового сортування по y. Зокрема, алгоритм тепер можна представити в наступному вигляді:

Більш ефективний алгоритм з упорядкованим списком ребер

Підготувати дані:

Визначити для кожного ребра багатокутника точки перетинань із скануючими рядками, проведеними через середини інтервалів, тобто через y+1/2. Для цього можна використовувати алгоритм Брезенхема чи ЦДА. Горизонтальні ребра не враховувати. Помістити координату y точки перетинання в групу, що відповідає y.

Для кожної y-групи відсортувати список координат x точок перетинань у порядку зростання; тобто x_1, передує x_2, якщо: $x_1<x_2$.

Перетворити ці дані в растрову форму:

Для кожного скануючого рядка виділити із списку координат x точок перетинань пари точок перетинань. Активувати на скануючому рядку у піксели для цілих значень x, таких, що $x_1 \le x+1/2 \le x_2$.

В алгоритмі спочатку за допомогою групового сортування по y відбувається сортування в порядку сканування рядків, а потім сортування в рядку. Таким чином, розгортка починається до завершення всього процесу сортування.

У такому алгоритмі частково легше додавати чи видаляти інформацію з дисплейного списку. Необхідно тільки додати чи видалити інформацію з відповідних y-груп; отже, пересортувати доведеться тільки порушені зміною рядки.

Нижче даний алгоритм ілюструється на прикладі.

6.7. Приклад, що ілюструє роботу більш ефективного алгоритму з упорядкованим списком ребер

Розглянемо знову як і в прикладі 6.5 багатокутник з рис.6.6 Спочатку створюються y-групи для всіх скануючих рядків, як це показано на рис.6.7. Багатокутник розглядається проти годинникової стрілки, починаючи з вершини *P,* і перетинання, що виходять, невідсортовані по x, заносяться у відповідні групи, як це показано на рис.6.7(a). Перетинання були обчислені відповідно до домовленості про середину інтервалу між скануючими рядками. Як ілюстрацію на рис.6.7(b) показано відсортовані перетинання. На практиці можна використовувати невеликий буфер скануючого рядка, що містить відсортовані координати-y точок перетинання, як це показано на рис.6.7(c). Такий буфер дозволяє більш ефективно додавати чи видаляти перетинання в список, їх просто можна додавати до кінця списку кожної y-групи, тому що сортування відкладається до моменту переміщення рядка в буфер. Отже, не потрібно тримати цілком відсортований список y-груп.

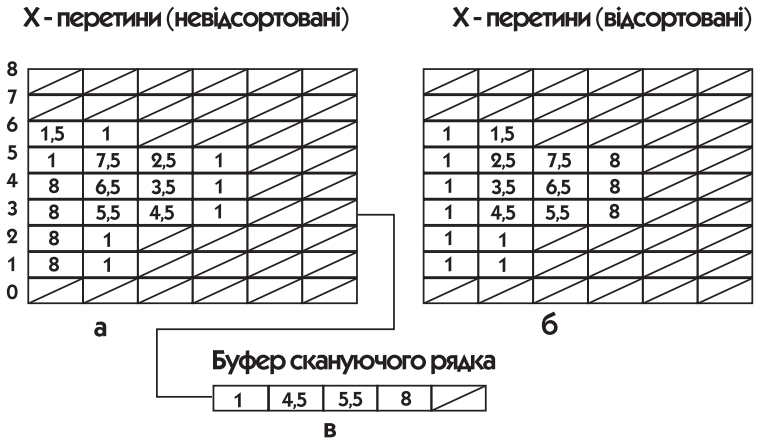


Рис.6.7. Y-групи скануючих рядків для багатокутника, зобр. на рис.6.3

В результаті виділення пар перетинань з відсортованого по y списку і застосування алгоритму для кожного скануючого рядка одержимо список активованих пікселів. Результат показаний на рис.6.6 і збігається з результатом прикладу 6.1.

6.8. Алгоритм з упорядкованим списком ребер, що використовує список активних ребер

Хоча в попередньому алгоритмі задача сортування спрощується, потрібно або обмежене число перетинань з даним скануючим рядком, або резервування великої кількості пам'яті, значна частина якої не буде використовуватися. Цей недолік можна усунути завдяки використанню зв'язного списку, тобто шляхом введення додаткової структури даних. Попереднє обчислення перетинань кожного скануючого рядка з кожним ребром багатокутника вимагає великих обчислювальних затрат і значних обсягів пам'яті. Увівши список активних ребер (САР), як це передбачалося

для растрової розгортки в реальному часі, можна скоротити потреби в пам'яті й обчислювати перетинання із скануючими рядками в покроковому режимі. Алгоритм, утворений в результаті всіх вдосконалень, виглядає так:

Алгоритм з упорядкованим списком ребер, що використовує список активних ребер.

Підготувати дані:

Використовуючи скануючі рядки, проведені через середини відрізків, тобто через y+1/2, визначити для кожного ребра багатокутника найвищий скануючий рядок, перетятий ребром.

Занести ребро багатокутника в y-групу, відповідно до цього скануючого рядка.

Зберегти в зв'язному списку значення: початкове значення координат x точок перетинання, Δy - число скануючих рядків, перетятих ребром багатокутника, і Δx - крок збільшення по x при переході від одного скануючого рядка до іншого.

Перетворити ці дані в растрову форму:

Для кожного скануючого рядка перевірити відповідну y-групу на наявність нових ребер. Нові ребра додати в САР.

Відсортувати координати x точок перетинання з САР у порядку зростання; тобто x_1 передує x_2 ,якщо $x_1<x_2$.

Виділити пари точок перетинань з відсортованого по x списку. Активувати на скануючому рядку y піксели для цілих значень x, таких, що $x_1 \le x+1/2 \le x_2$. Для кожного ребра з САР зменшити Δy на 1. Якщо $\Delta y<0$, то вилучити дане ребро з САР. Обчислити нове значення координат x точок перетинання $x_{нов}= x_{стар}+ \Delta x$.

Перейти до наступного скануючого рядка.

В алгоритмі передбачається, що всі дані попередньо перетворено до вигляду, прийнятого для багатокутників.

6.9. Приклад, що ілюструє роботу алгоритму упорядкованого списку ребер разом зі списком активних ребер

Знову розглянемо простий багатокутник з рис.6.3. Перевірка списку ребер багатокутника показує, що скануючий рядок 5 - найвищий для ребер P_2P_3 і P_3P_4 , а рядок 6 - для ребер P_4P_5 і P_5P_1. На рис.6.8(а) схематично показана структура зв'язного списку, що містить дані для дев'яти y-груп, що відповідають дев'яти (з 9) скануючим рядкам рис.6.3. Зауважимо, що більшість y-груп порожні.

На рис.6.8(б) показаний підхід, що застосовується на практиці. Тут список y-груп - це одномірний масив, по одному елементу на кожен скануючий рядок. У цьому елементі міститься тільки покажчик на масив даних, використовуваний для зв'язного списку. Масив показаний на тому ж рисунку.

Зв'язний список реалізований як масив розмірності n x 4. Для кожного індексу масиву n чотири елементи містять: значення координати x точки перетинання ребра багатокутника з найвищим рядком, перетятим цим ребром; крок збільшення по y при переході від одного скануючого рядка до

іншого; число скануючих рядків, перетятих ребром багатокутника; покажчик на розташування даних для наступного ребра, яке починається на цьому ж скануючому рядку.

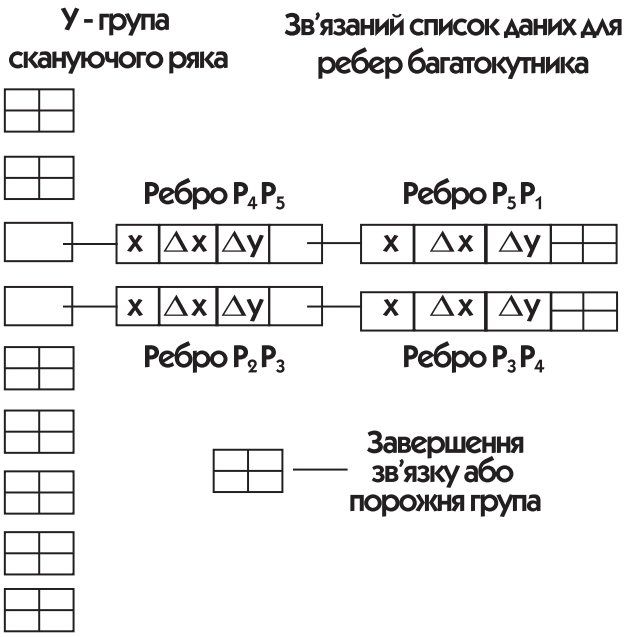


Рис.6.8. Схема зв'язного списку для багатокутника, зобр. на рис.6.3

Це продемонстровано на рис.6.8(б). Y-група для рядка 5 містить покажчик 1, що відповідає першій адресі в зв'язному списку даних. Перші три стовпчики даних у списку містять інформацію про ребро P_2P_3. Число в четвертому стовпчику - це покажчик на наступний елемент даних.

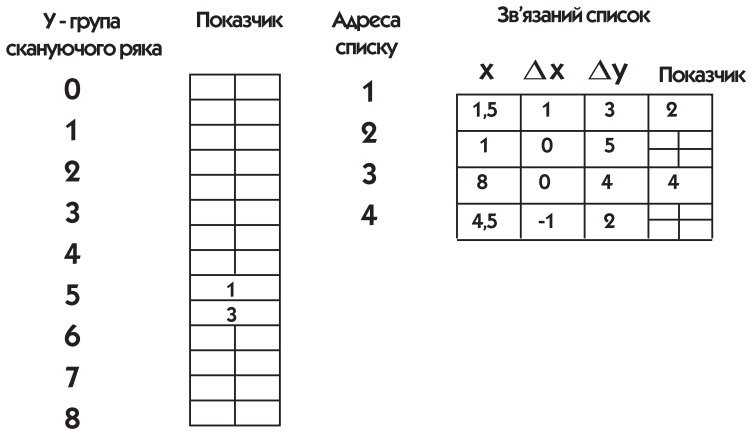


Рис.6.8. Схема зв'язного списку для багатокутника, зобр. на рис.6.3

САР реалізований у вигляді стекового масиву розмірності n x 3 і для всіх дев'яти скануючих рядків показаний на рис.6.8(с). Скануючі рядки (y-групи) послідовно перевіряються зверху вниз, починаючи з рядка 8. Оскільки y-групи для рядків 8 і 7 порожні, то САР також порожній. Рядок 6 додає в САР два елементи і рядок 5 - ще два. На рядку 2 значення Δy для ребер P_3P_4 і P_4P_5 стає меншим за нуль. Ці ребра вилучаються із САР. Аналогічно ребра P_2P_3 і P_5P_1, вилучаються на рядку 0. Для рядка 0 y-група порожня, САР порожній і більше y-груп немає. Алгоритм завершений.

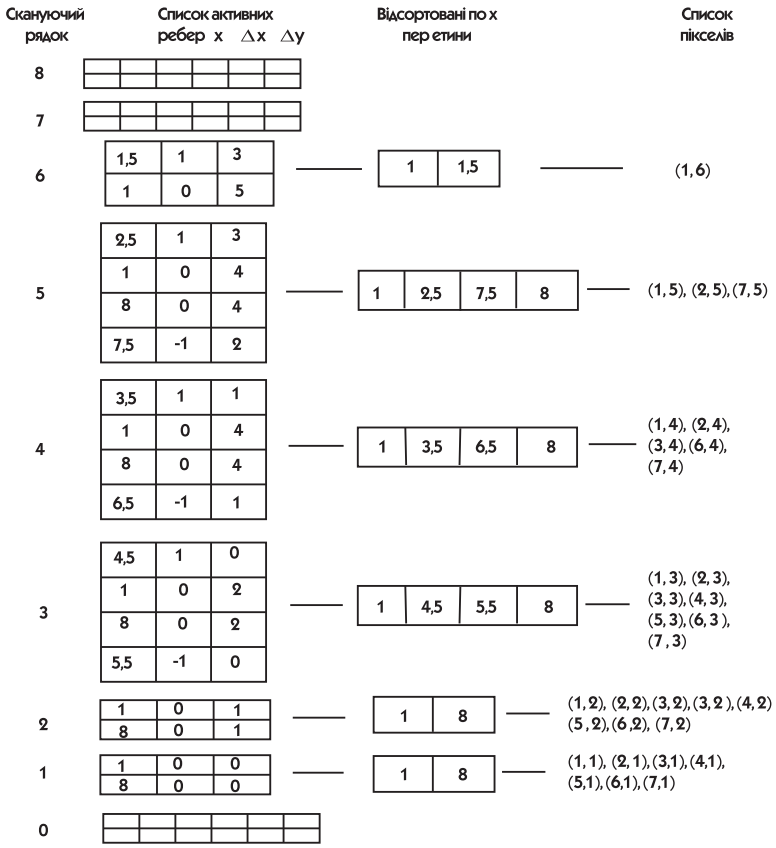


Рис.6.8. Схема зв'язного списку для багатокутника, зобр. на рис.6.3

Для кожного скануючого рядка вилучаються координати x точок перетинання активних ребер для цього рядка, сортуються по зростанню x і записуються в інтервальний буфер, реалізований у вигляді масиву 1xn. З цього буфера перетинання поділяються на пари, і потім, використовуючи описаний алгоритм, визначається список активних пікселів. Об'єднаний для всіх скануючих рядків список активних пікселів збігається із списком з попереднього прикладу. Результат знову представлений на рис.6.6.

Лекція 7
Алгоритми заповнення

7.1. Алгоритми заповнення з запалом

В алгоритмах заповнення з запалом передбачається, що відомо хоча б один піксел з області багатокутника. Алгоритм намагається знайти і зафарбувати всі інші піксели, що належать внутрішній області.

Області можуть бути або внутрішньо-, або гранично-визначеними. Якщо область відноситься до внутрішньо-визначених, то всі піксели, що належать внутрішній частині, мають той самий колір чи інтенсивність, а всі піксели, зовнішні стосовно області, мають інший колір (рис.7.1).

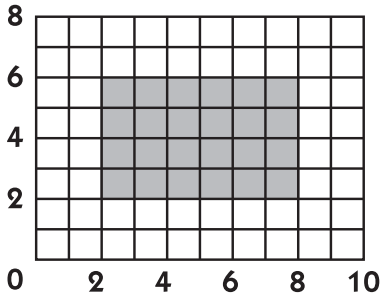


Рис.7.1 Внутрішньо-визначена область

Якщо область відноситься до гранично-визначених, то всі піксели на границі області мають виділене значення чи колір (рис.7.2).

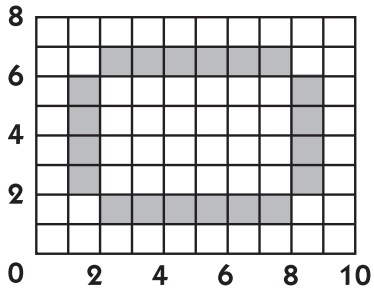


Рис.7.2 Гранично-визначена область

Жоден з пікселів із внутрішньої частини такої області не може мати це виділене значення. Проте, піксели, зовнішні відносно границі, можуть мати граничне значення.

Алгоритми, що заповнюють внутрішньо-визначені області, називаються внутрішньозаповнюючими, а алгоритми для гранично-визначених областей - граничнозаповнюючими. Далі будуть розглядатися граничнозаповнюючі алгоритми. Однак внутрішньозаповнюючі алгоритми можна отримати аналогічно.

Внутрішньо- чи гранично-визначені області можуть бути 4- чи 8-зв'язними. Якщо область 4-зв'язна, то будь-якого піксела можна досягти за допомогою комбінації рухів тільки в 4 напрямках: ліворуч, праворуч, вгору, вниз. Для 8-зв'язної області піксела можна досягти за допомогою комбінації

рухів у двох горизонтальних, двох вертикальних і 4 діагональних напрямках. Алгоритм заповнення 8-зв'язної області заповнить і 4-зв'язну область, однак зворотнє твердження невірне.

На рис.7.3 показані прості приклади 4- і 8-зв'язних внутрішньо-визначених областей.

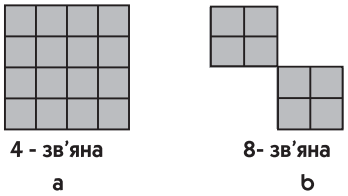


Рис.7.3. 4- і 8-зв'язні внутрішньо-визначені області

Хоча кожна з підобластей 8-зв'язної області на рис.7.3(б) є 4-зв'язною, для переходу з однієї підобласті в іншу потрібен 8-зв'язний алгоритм. Однак у ситуації, коли треба заповнити різними кольорами дві окремі 4-зв'язні підобласті, використання 8-зв'язного алгоритму викликає неправильне заповнення обох областей тим самим кольором.

На рис.7.4 показана 8-зв'язна область з рис.7.3, перевизначена у виді гранично-визначеної області.

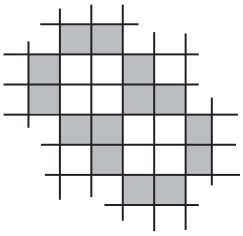


Рис.7.4. 4- і 8-зв'язні гранично-визначені області

На рис.7.4 ілюструється той факт, що для 8-зв'язної області, у якої дві підобласті дотикаються кутами, границя 4-зв'язна, а границя 4-зв'язної області 8-зв'язна. Далі мова йтиме про алгоритми для 4-зв'язних областей, але їх можна легко переробити для 8-зв'язних областей, якщо заповнення проводити не в 4-х, а 8-ми напрямках.

7.2. Простий алгоритм заповнення з запалом

Використовуючи стек, можна розробити простий алгоритм заповнення гранично-визначеної області. *Стек* - це просто масив чи інша структура даних, в яку можна послідовно заносити значення і з якої їх можна послідовно виймати. Коли нові значення додаються чи заносяться в стек, всі інші значення опускаються вниз на один рівень. Коли значення виймаються чи видаляються із стека, інші значення спливають чи піднімаються нагору на один рівень. Такий стек називається стеком прямої дії чи стеком з порядком обслуговування "першим прийшов, останнім вийшов" (FILO).

Простий алгоритм заповнення з запалом можна представити в наступному вигляді:

Простий алгоритм заповнення з запалом і стеком

- *Помістити затравочний піксел у стек*
- *Поки стек не порожній*
- *Витягти піксел зі стека*
- *Привласнити пікселу необхідне значення*
- *Для кожного із сусідніх до біжучого 4-зв'язних пікселів перевірити: чи є він граничним пікселом, чи не привласнене пікселу необхідне значення. Проігнорувати піксел у кожному з цих двох випадків. В іншому випадку помістити піксел у стек.*

Алгоритм можна модифікувати для 8-зв'язних областей, якщо переглядати 8-связні піксели, а не тільки 4-зв'язні. Наведемо більш формальний виклад алгоритму, у якому передбачається існування затравочного піксела і гранично-визначеної області.

Простий алгоритм заповнення

Запал (x, y) видає затравочний піксел
Push - *процедура, що поміщає піксел у стек*
Pop - *процедура, що витягає піксел зі стека*
Піксел (x, y) = Запал (x, y)
Ініціалізуємо стек
Push Піксел (x, y)
while (стек не порожній)
    Вибираємо піксел зі стека
    Pop Піксел (x, y)
    if Піксел(x, y)<>Нове_значення **then**
    Піксел(x, y)=Нове_значення
    end if
    Перевіримо, чи треба поміщати сусідні піксели в стек
    if (Піксел(x+1,y)<>Нове_значення and Піксел(x+1,y)<>Гран_значення)
        then Push Піксел(x+1, y)
    if (Піксел(x,y+1)<>Нове_значення and Піксел(x,y+1)<>Гран_значення)
    then Push Піксел(x, y+1)
    if (Піксел(x-1, y)<>Нове_значення and Піксел(x-1, y)<>Гран_значення)
      then Push Піксел(x-1, y)
    if (Піксел(x, y-1)<>Нове_значення and Піксел(x, y-1)<>Гран_значення)
      then Push Піксел(x, y-1)
    end if
end while

В алгоритмі перевіряються і поміщаються в стек 4-зв'язні піксели, починаючи з правого від біжучого піксела. Напрямок обходу пікселів - проти годинникової стрілки.

7.3. Приклад, який ілюструє роботу простого алгоритму заповнення з запалом

Як приклад застосування алгоритму розглянемо гранично-визначену полігональну область, задану вершинами (1, 0), (7, 0), (8, 1), (8, 4), (6, 6), (1, 6), (0,5) і (0, 1). Область зображена на рис.7.5.

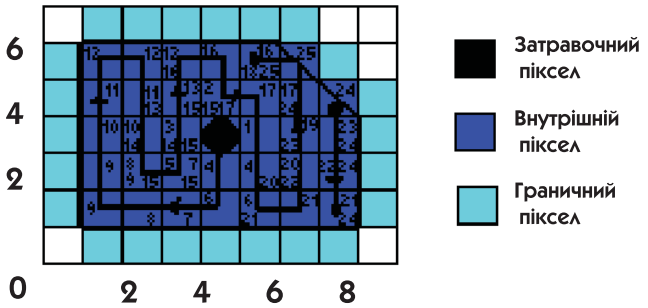


Рис.7.5 Затравочне заповнення за допомогою простого стекового алгоритму

Затравочний піксел - (4, 3). Область заповнюється піксел за пікселем у порядку, зазначеному на рис.7.5 лінією зі стрілками. Числа, зображені на пікселі, позначають позицію піксела в стеку, яку він займає при роботі алгоритму. Для деяких пікселів таких чисел декілька. Це означає, що піксел поміщали в стек більше 1 разу. Коли алгоритм доходить до піксела (5, 5), стек нараховує 25 рівнів глибини і містить піксели (7, 4), (7, 3), (7, 2), (7, 1), (6, 2), (6, 3), (5, 5), (6, 4), (5, 5), (4, 4),(3, 3), (3, 4), (3, 5), (2, 4), (2, 3), (2, 2), (2, 2), (3, 2), (5, 1), (3, 2), (5, 2), (3, 3), (4, 4), (5, 3).

Оскільки всі піксели, які оточують (5, 5), містять або граничні, або нові значення кольору, жоден з них у стек не поміщається. Отже зі стека виймається піксел (7, 4) і алгоритм продовжує заповнювати колонку (7, 4), (7, 3), (7, 2), (7, 1). При досягненні піксела (7, 1) перевірка знову показує, що навколишні піксели або уже заповнено, або є граничними пікселами. Тому що в цей момент багатокутник цілком заповнений, то вибирання пікселів зі стека до повного його спустошення не викликає появи додаткових пікселів, які варто запам'ятати. Коли стек стає порожнім, алгоритм завершує роботу.

Багатокутник із прикладу 7.3 є однозв'язною областю, але алгоритм буде правильно заповнювати й області, що містять діри.

7.4. Приклад, який ілюструє роботу алгоритму заповнення з запалом для багатокутника з дірою

Як приклад застосування алгоритму розглянемо гранично-визначену область, що містить діру. Вона зображена на рис.7.6.

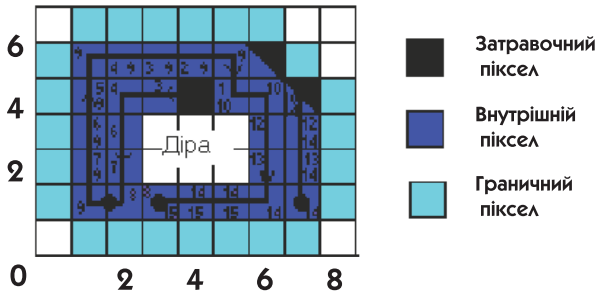


Рис.7.6 Затравочне заповнення області з дірою за допомогою простого стекового алгоритму

Як і в попередньому прикладі, вершини багатокутника задані пікселами (1, 0), (7, 0), (8, 1), (8, 4), (6, 6), (1, 6), (0,5) і (0, 1). Внутрішня діра визначається пікселами (3, 2), (5, 2), (5, 3), (3, 3). Затравочний піксел - (4, 4). Через діру багатокутник заповнюється не в тому порядку, що у прикладі 7.3. Новий порядок заповнення зазначений на рис.7.6 лінією зі стрілками. Як і в попередньому прикладі, числа в квадратику піксела показують позицію в стеку, яку займає піксел. Коли обробка доходить до піксела (3, 1), усі навколишні його 4-зв'язні піксели або вже заповнені, або є граничними. Тому жоден з пікселів не поміститься в стек. Глибина стеку в цей момент дорівнює 15. У стеку знаходяться піксели (7, 1), (7, 2), (7, 3), (6, 5), (7, 4), (6, 5), (3, 1), (1, 2), (1, 3), (1, 4), (2, 5), (3, 5), (4, 5), (5, 4).

Після вибору із стека піксела (7, 1) заповнюється стовпчик (7, 1), (7, 2), (7, 3), (7, 4), при цьому жоден піксел у стек не додається. Для піксела (7, 4) знову всі 4-зв’язні навколишні піксели або вже заповнені, або є граничними. Звертаючись до стеку, алгоритм витягає піксел (6, 5), його заповнення завершує заповнення всього багатокутника. Подальша обробка відбувається без будь-якого заповнення, і коли стек стає порожнім, алгоритм завершує роботу.

7.5. Порядковий алгоритм заповнення з запалом

Даний алгоритм застосовується до гранично-визначених областей. Гранично-визначена 4-зв'язна область може бути як опуклою, так і не опуклою, а також може містити діри.

В зовнішній області, пов'язаній з нашою гранично-визначеною областю, не повинно бути пікселів з кольором, яким заповнюється область чи багатокутник.

Схематично роботу алгоритму заповнення з запалом можна розбити на чотири етапи.

Порядковий алгоритм заповнення з запалом

- Затравочний піксел на інтервалі витягається зі стека, що містить затравочні піксели.
- Інтервал із затравочним пікселем заповнюється вліво і вправо від запалу уздовж скануючого рядка доти, поки не буде знайдена границя. У змінних $X_{лів}$ і $X_{прав}$ запам'ятовуються крайній лівий і крайній правий піксели інтервалу.
- У діапазоні $X_{лів}<x<X_{прав}$ перевіряються рядки, розташовані безпосередньо над і під біжучим рядком. Визначається, чи є на них ще не заповнені піксели. Якщо такі піксели є (тобто не всі піксели граничні, чи уже заповнені), то в зазначеному діапазоні крайній правий зазначений піксел у кожнім інтервалі відзначається як затравочний і додається в стек.
- При ініціалізації порядкового алгоритму з запалом в стек додається єдиний затравочний піксел, робота завершується при спустошенні стека.

Як показано на рис.7.7 і в прикладі нижче, алгоритм справляється з дірами і зубцями на границі.

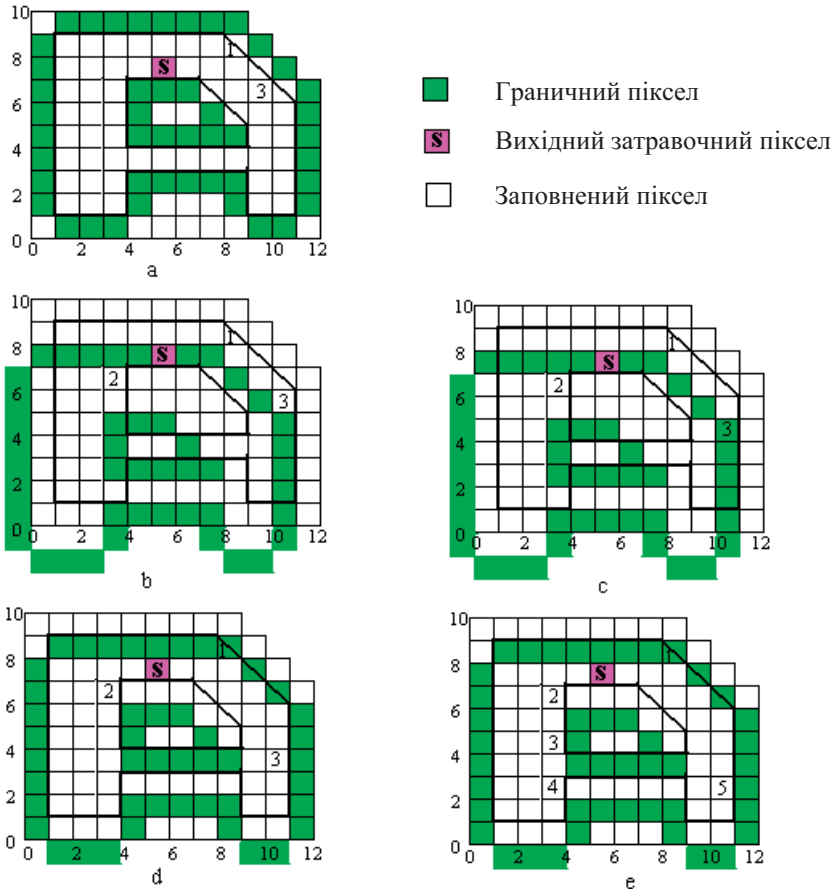


Рис.7.7 Порядковий алгоритм заповнення з запалом для багатокутника

Порядковий алгоритм заповнення з запалом

Запал (x, y) видає затравочний піксел
Pop - *процедура, що витягає піксел зі стека*
Push - *процедура, що поміщає піксел у стек*
Ініціалізуємо стек
Push Запал(x, y)
while (стек не порожній)
    Вибираємо піксел зі стека і привласнюємо йому нове значення
    Pop Піксел(x, y)
    Піксел(x, y)=Нове_значення
    Зберігаємо x-координату затравочного піксела
    ТимЧас_x=x
    Заповнюємо інтервал праворуч від запалу
    x=x+1
    while Піксел(x, y)<>Гран_значення

```
        Піксел(x, y)=Нове_значення
        x=x+1
end while
```
зберігаємо крайній праворуч піксел
$X_{прав}$=x-1
Відновлюємо x-координату запалу
x=ТимЧас_x
заповнюємо інтервал ліворуч від запалу
x=x-1
while Піксел(x, y)<>Гран_значення
        Піксел(x, y)=Нове_значення
        x=x-1
end while
зберігаємо крайній ліворуч піксел
$X_{лів}$=x+1
Відновлюємо x-координату запалу
x=ТимЧас_x
перевіримо, що рядок вище не є ані границею багатокутника, ані вже цілком заповнений;
якщо це не так, то знайти запал, починаючи з лівого краю підінтервала скануючого рядка.
x=$X_{лів}$
y=y+1
while x<=$X_{прав}$
        шукаємо запал на рядку вище
        Прапор=0
        while (Піксел(x, y)<>Гран_значення and
                Піксел(x, y)<>Нове_значення and x<$X_{прав}$
                if Прапор=0 **then** Прапор=1
                x=x+1
        end while
        поміщаємо в стек крайній правий піксел
        if Прапор=1 **then**
          if (x=$X_{прав}$ and Піксел(x, y)<>Гран_значення and
            Піксел(x, y)<>Нове_значення) **then** Push Піксел(x, y)
          else Push Піксел(x-1, y)
          end if
          Прапор=0
        end if
        Продовжимо перевірку, якщо інтервал був перерваний
        $X_{вход}$=x
        while ((Піксел(x, y)=Гран_значення or
              Піксел(x, y)=Нове_значення)and x<$X_{прав}$)
                x=x+1
        end while
        впевнимося, що координата піксела збільшена

if $x=X_{вход}$ **then** $x=x+1$

end while

перевіримо, що рядок нижче не є ані границею багатокутника, ані вже цілком заповнений

ця частина алгоритму цілком аналогічна перевірці для рядка вище, за винятком того, що замість y=y+1 треба поставити y=y-1

end while

finish

Тут функція Pop вибирає координати x, y піксела із стека, а функція Push поміщає їх у стек.

7.6. Приклад, що ілюструє роботу порядкового алгоритму заповнення з запалом

Розглянемо роботу алгоритму для гранично-визначеної області на рис.7.7. При ініціалізації в стек додається затравочний піксел, позначений як Запал (5, 7) на рис.7.7(a). Спочатку як запал інтервалу зі стека вибирається цей піксел. Інтервал заповнюється праворуч і ліворуч від запалу. Знайдені при цьому кінці інтервалу $X_{прав}=9$ і $X_{лів}=1$. Потім перевіряється рядок, розташований вище біжучого і виявляється, що він граничний і незаповнений. Крайнім правим пікселем у діапазоні $1<x<9$ виявляється піксел (8, 8), позначений цифрою 1 на рис.7.7a. Цей піксел поміститься в стек. Потім аналогічним чином обробляється рядок нижче біжучого. У діапазоні $X_{лів}<x<X_{прав}$ є два підінтервали на цьому рядку. Для лівого підінтервала як запал виступає піксел (3, 6), позначений цифрою 2 на рис.7.7(b), він теж додасться в стек. Запал для правого підінтервала - піксел (9, 6), він додається в стек. Зауважимо, що піксел (9, 6) - це не крайній правий піксел на інтервалі, однак це крайній піксел у діапазоні $X_{лів}<x<X_{прав}$, тобто в діапазоні $1<x<9$. На цьому завершується перший прохід алгоритму.

Далі зі стека вибирається верхній піксел. Тут заповнюються інтервали, розташовані в правій частині багатокутника на послідовних скануючих рядках (рис.7.7(b), (c), (d)). Для рядка 3 запалом служить піксел (10, 3) (рис.7.7(d)). У результаті заповнення інтервалу праворуч і ліворуч від запалу одержуємо нові межі діапазону: $X_{лів}=1$ і $X_{прав}=10$. Обробляючи рядок вище, одержуємо для лівого підінтервала затравочний піксел (3, 4), що вміщується в стек. Правий підінтервал до цього часу уже заповнений. Обробка рядка нижче дає запал (3, 2) для лівого і (10, 2) для правого підінтервалів. Ці піксели також помістяться в стек. Саме зараз досягається максимальна глибина стека для багатокутника, який обробляється.

Тепер залишається тільки один цікавий момент. Після заповнення 4-зв'язних полігональних підобластей із затравочними пікселами 5, 4 і 3 на рис.7.7(e) зі стека вибирається піксел, позначений цифрою 2. Тут ми виявляємо, що всі піксели на цьому рядку і на сусідніх рядках (вище і нижче) уже заповнені. Таким чином, жоден піксел у стек не поміститься. Зі стека вибирається піксел 1 і рядок обробляється, при цьому знову додаткових пікселів не з'являється. Тепер стек порожній, багатокутник заповнений і робота алгоритму довершена.

Лекція 8
Основні методи усунення ступінчастості
8.1. Основи методів усунення ступінчастості

Щоб ефективно боротися із ступінчастістю (сходовим ефектом), що призводить до спотворень у зображенні, необхідно розуміти причини її виникнення. Основна причина появи сходового ефекту полягає в тому, що відрізки, ребра багатокутника, колірні границі і т.д. мають неперервну природу, тоді як растровий пристрій – дискретну. Для представлення відрізка, ребра багатокутника і т.д. на растровому пристрої необхідно побудувати їх у дискретних координатах, що може призвести до досить незвичних результатів.

Для запобігання спотворенню варто проводити вибірку сигналу з частотою, що принаймні вдвічі перевищує найбільшу частоту сигналу. Недостатня вибірка призводить до того, що високоперіодичні зображення візуалізуються невірно. Наприклад, огорожа паркану чи піднімальні жалюзі можуть виглядати як кілька широких смуг, а не як багато окремих, вужчих смужок.

Міркування, наведені вище, і матеріал попередніх розділів ілюструють два з трьох загальних проявів спотворень такого роду в машинно-згенерованих зображеннях: ступінчастість ребер і некоректна візуалізація тонких деталей чи фактури.

Третій прояв зв'язаний з дуже дрібними об'єктами. Якщо об'єкт менший за розмір піксела чи не покриває внутрішню точку, яка служить для оцінки атрибутів піксела, то він не буде враховуватися в результуючому зображенні. З іншого боку, якщо малий обсяг покриває цю точку, то він може занадто сильно вплинути на атрибути піксела. Якщо для визначення атрибутів використовується середина піксела, то в даній ситуації весь піксел буде мати атрибути маленького об'єкта. Також є об'єкти, що будуть зігноровані чи загублені. Помітимо, що можуть бути зігноровані також і довгі, тонкі об'єкти. Особливо помітні такі ефекти в анімаційних послідовностях кінокадрів. Наприклад, маленький трикутник на трьох кадрах анімаційної послідовності. Якщо атрибути піксела визначаються в його центрі, то на першому кадрі об'єкта не бачимо, на другому - бачимо, а на третьому - знову не бачимо. В сукупності це буде виглядати як мерехтіння об'єкта.

В основному існує два методи усунення спотворень зображення такого роду. Перший зв'язаний зі збільшенням частоти вибірки, що досягається за допомогою збільшення роздільчої здатності растра. Таким чином, враховуються дрібніші деталі.

Однак існує певне обмеження на здатність растрових графічних пристроїв з ЕПТ виводити дуже дрібні растри. На даний час межа складає близько 2000 пікселів у рядку.

Таке обмеження припускає, що растр треба обчислювати з більш високою роздільчою здатністю, а зображати з більш низькою, використовуючи усереднення деякого типу для одержання атрибутів піксела з більш низькою роздільчою здатністю.

На рис.8.1 показане усереднення двох типів. Рівномірне усереднення навколишніх пікселів для зменшення роздільчої здатності в 2 і 4 рази демонструється на рис.8.1(a).

Кожен дисплейний піксел поділяється на підпіксели в процесі формування растра вижчої роздільчої здатності. Для одержання атрибутів дисплейного піксела визначаються атрибути в центрі кожного підпіксела, що потім усереднюються. До деякої міри можна одержати кращі результати, якщо розглядати більше підпікселів і враховувати їх вплив за допомогою ваг при визначенні атрибутів.

1	2	I
2	2	2
1	2	1

a

1	2	3	4	3	2	1
2	4	6	8	6	4	2
3	6	9	12	9	6	3
4	8	12	12	12	8	4
3	6	9	12	9	6	3
2	4	6	8	6	4	2
1	2	3	4	3	2	1

b

Рис.8.1. Усереднення характеристик піксела: (a) рівномірне, (b) зважене (числа позначають відносні ваги)

На рис.8.1(b) представлені зважені середні, запропоновані Кроу, для зменшення роздільчої здатності в 2 і 4 рази.

Для цих зважених середніх при зменшенні роздільчої здатності в 2 рази розглядаються 9 підпікселів, а при зменшенні в 4 рази - 49.

Другий метод усунення спотворень зображення полягає в тому, щоб трактувати піксел не як точку, а як скінчену область.

8.2. Простий метод усунення сходового ефекту

В алгоритмах розкладання відрізка в растр і заповнення багатокутника, що обговорювалися вище, інтенсивність чи колір піксела визначалися інтенсивністю чи кольором єдиної точки всередині області піксела. У цих методах передбачалося, що піксел є скоріше математичною точкою, ніж скінченою областю. Наприклад, згадавши алгоритм Брезенхема, ми побачимо, що інтенсивність пікселів визначалася місцем розташування однієї точки перетинання відрізка і границі піксела. У методах растрової розгортки суцільної області, що теж обговорювалися вище, визначення місця розташування піксела (всередині чи поза багатокутником) ґрунтувалося на положенні центра піксела. Якщо центр знаходився всередині, то активувався весь піксел. Якщо центр знаходився поза, то ігнорувалася вся область піксела.

Цей метод визначення місця розташування піксела необхідний для простих дворівневих зображень, тобто чорного чи білого кольору чи кольору фону багатокутника.

В результаті виходить характерне східчасте чи зазубрене ребро багатокутника чи відрізок. Як обговорювалося в попередньому розділі, основною причиною сходового ефекту є те, що дискретність відрізка чи ребра недостатня для того, щоб відповідати дискретним пікселам екрана дисплея.

При наявності декількох інтенсивностей, тобто півтонів сірого чи відтінків кольору, зовнішній вигляд ребра чи відрізка може бути поліпшений розмиванням країв. Простий евристичний метод полягає в тому, щоб встановлювати інтенсивність піксела на ребрі пропорційною до площі частини піксела, що знаходиться всередині багатокутника.

Рис.8.2. Усунення ступінчастості ребра багатокутника:
(a) без усунення ступінчастості
(в) інтенсивність пропорційна площі частини піксела, що знаходиться всередині багатокутника
(c) ребро, отримане за допомогою модифікованого алгоритму Брезенхема

Ця проста форма усунення ступінчастості ілюструється рис.8.2, на якому зображене одне ребро багатокутника. Внутрішня частина багатокутника розташована праворуч. На рис.8.2(a) представлене ребро, розкладене в растр за допомогою стандартного алгоритму Брезенхема з використанням тільки двох рівнів інтенсивності. Зображення має характерний зазубрений чи східчастий вигляд. На рис.8.2(b) для вибору одного з восьми (від 0 до 7) рівнів інтенсивності використовується площа частини піксела, що знаходиться всередині багатокутника. Зауважимо, що оскільки ця площа для деяких пікселів менше однієї восьмої, то деякі піксели, цілком чорні на рис.8.2(a), зафарбовані в білий колір на рис.8.2(b).

В результаті простої модифікації алгоритму Брезенхема можна одержати апроксимацію площі частини піксела, що знаходиться всередині багатокутника. Цю апроксимацію можна використовувати для модуляції інтенсивності.

Якщо до похибки у вихідному алгоритмі Брезенхема додати величину $w=1-w$, тобто ввести перетворення $e=e+w$, то $0<e<1$. Тепер похибка e - це міра площі тієї частини піксела, що знаходиться всередині багатокутника, тобто $Y_i+m/2$. У зв'язку з цими модифікаціями початкове значення похибки рівне 1/2, тому для першого піксела алгоритм, що наводився на рис.5.5, завжди буде видавати значення інтенсивності, рівне половині максимальної.

Модифікований алгоритм виглядає таким чином:

Модифікований алгоритм Брезенхема з усуненням ступінчастості для першого квадранта

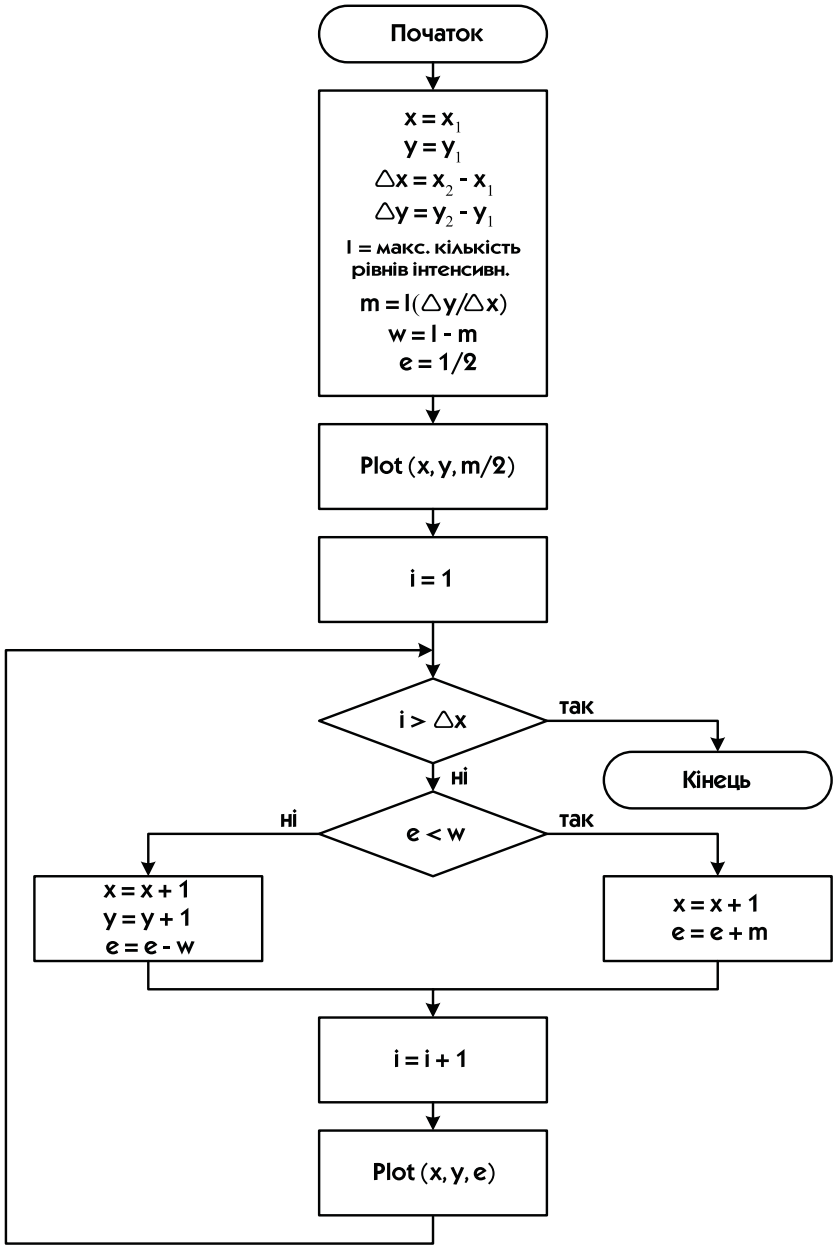


Рис.8.3. Алгоритм Брезенхема з усуненням ступінчастості

Більш реалістичне значення для першого піксела дає переміщення оператора активізації піксела на інше місце. Більш того, можна одержати безпосереднє значення інтенсивності, а не десятковий дріб від її максимуму за допомогою множення на максимальне число доступних рівнів інтенсивності 7 наступних величин: тангенса кута нахилу, вагового коефіцієнта і похибки.

Інтенсивність для першого піксела припускає, що відрізок починається з адреси піксела. На рис.8.3 наводиться блок-схема алгоритму. Результати для відрізка з тангенсом кута нахилу w=5/8 і вісьмома рівнями інтенсивності подані на рис.8.2(с).

Поширити роботу алгоритму на інші октанти можна тим же способом, що і для основного алгоритму Брезенхема.

8.3. Згортка й усунення ступінчастості

Поширення простих методів усунення ступінчастості (згладжування), що обговорювалися в попередньому розділі, вимагає використання математичного методу, який називається згорткою. Для згладжування береться згортка сигналу, тобто зображення, з ядром згортки, а результат використовується для визначення атрибутів піксела.

Уявити собі фізичний зміст згортки з її математичного визначення дуже важко. Однак нам у цьому допоможе простий графічний аналіз (рис.8.4).

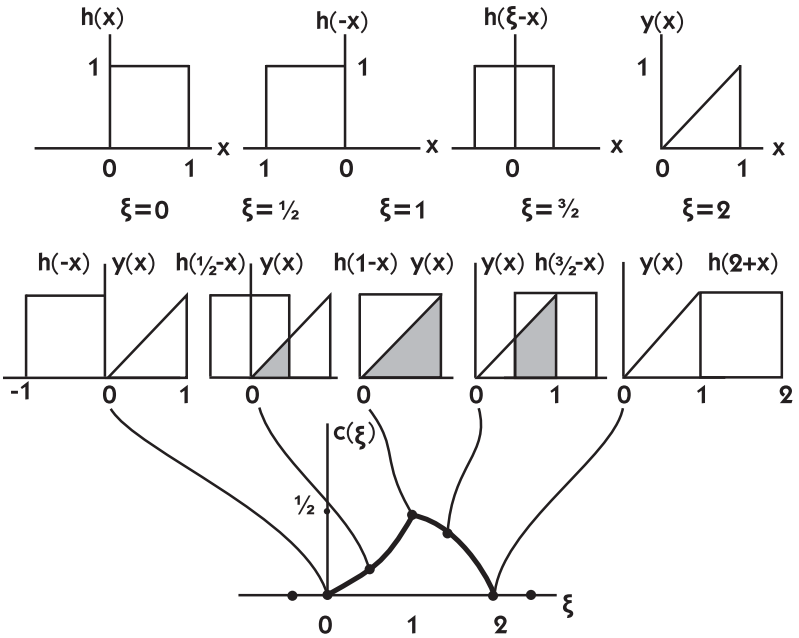


Рис. 8.4. Відображення ядра згортки

Визначення згортки еквівалентно відображенню ядра згортки, його зсуву, перемноженню двох функцій і визначенню площі під об'єднаними кривими. Оскільки ядро і функція, що підлягає згортці, не дорівнюють

нулю лише на кінцевих інтервалах, то межі інтегрування в згортці теж кінцеві. З рис.8.4 зрозуміло, що нижня межа дорівнює максимуму початків інтервалів, на яких функції не дорівнюють нулю, а верхня межа дорівнює мінімуму кінців цих інтервалів.

Щоб зрозуміти, який зв'язок існує між згорткою і усуненням ступінчастості (згладжуванням) згадаємо евристичний метод модуляції інтенсивності, у якому для визначення інтенсивності використовується площа піксела.

Операція згортки часто називається фільтрацією, а ядро згортки - функцією фільтра. У простому методі з площами, що обговорювався вище, зображення попередньо фільтрується. Постфільтрація вирівнює атрибути пікселів обчисленої роздільчої здатності до виводу зображення на екран.

Метод обчислення зображення з роздільчою здатністю, більшою, ніж роздільча здатність дисплея, і подальше усереднення атрибутів декількох пікселів для одержання пікселів з меншою роздільчою здатністю можна розглядати як постфільтруючу операцію.

Хоча за допомогою простого прямокутного фільтра чи ядра згортки одержують цілком прийнятні результати, трикутний і гауссовський фільтри приводять до ще більш якісних результатів. Використовуються також двовимірні фільтри. В даний час досліджені просте прямокутне, пірамідальне, кінцеве і двовимірне гауссове ядра згортки чи функції фільтра.

Прості фільтри у виді згортки не завжди ефективні для маленьких багатокутників із площею, меншою від площі пікселу чи для довгих, тонких багатокутників. Там згладжування можна реалізувати за допомогою відсікання (рис.8.5).

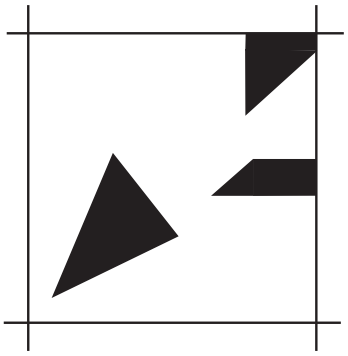


Рис.8.5. Усунення ступінчастості за допомогою відсікання

Сторони області піксела утворюють відтинаюче вікно. Кожен індивідуальний багатокутник відтинається по сторонах цього вікна. Для модулювання інтенсивності піксела використовується відношення площі отриманого в результаті відсікання багатокутника до площі піксела. Якщо всередині піксела знаходиться декілька багатокутників, то використовується середнє (рівномірне або зважене) їх атрибутів для модулювання атрибутів піксела.

Приклад такого відсікання наведений на рис.8.6.

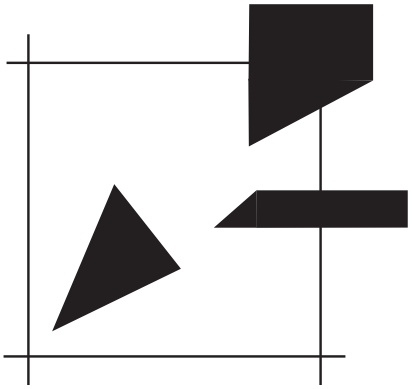


Рис.8.6. Усунення ступінчастості за допомогою зваженого відсікання

8.4. Апроксимація півтонами

Згладжування чи усунення ступінчастості - це метод покращення візуальної роздільчої здатності з використанням декількох рівнів інтенсивності. З іншого боку, апроксимація півтонами - це метод, у якому використовується мінімальне число рівнів інтенсивності, звичайно чорний і білий, для покращення візуальної роздільчої здатності, тобто одержання декількох півтонів сірого чи рівнів інтенсивності.

Метод півтонів відомий досить давно. Спочатку він використовувався при виготовленні шовкових картин і інших текстильних виробів. У 1880 р. Стефаном Хагевом був винайдений сучасний напівтоновий друк. Цим методом можна одержати велику кількість фотографічних півтонів сірого, використовуючи чисто дворівневий термін: чорну фарбу на білому папері. Напівтоновий друк - це решітчастий чи клітинний процес. Розмір клітинки змінюється в залежності від дрібнозернистості ґратки і тривалості експозиції. Для газетних фотографій через низьку якість паперу застосовуються ґратки від 50 до 90 точок на дюйм. Папір більш високої якості, призначений для книг і журналів, дозволяє використовувати гратки з кількістю точок від 100 до 300 на дюйм. Успіх методу півтонів залежить від властивості зорової системи людини бути інтегратором, тобто поєднувати чи згладжувати дискретну інформацію.

Візуальну роздільчу здатність машинно-згенерованих зображень можна покращити за допомогою методу конфігурування. На противагу напівтоновому друку, у якому використовуються змінні розміри клітинок, у даному методі зазвичай розміри клітинок фіксовані. Для зображення з фіксованою роздільчою здатністю декілька пікселів поєднуються в конфігурації. Тут погіршення просторової роздільчої здатності компенсується покращенням візуальної. На рис.8.7(а) показана одна з можливих груп конфігурацій для дворівневого чорно-білого дисплея. Для кожної клітинки використовуються чотири піксели. При такій організації отримуються п'ять можливих рівнів чи тонів сірого (0-4).

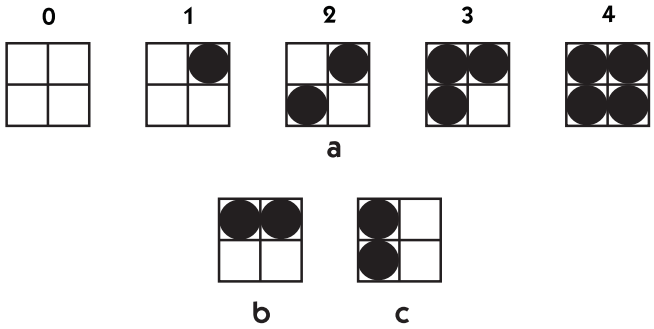


Рис.8.7. Дворівневі конфігурації 2×2

У загальному випадку для двохрівневого дисплея число можливих інтенсивностей на одиницю більше від числа пікселів у клітці. При виборі конфігурацій варто бути обережним, тому що інакше можуть виникнути небажані дрібномасштабні структури. Наприклад, не слід застосовувати жодну з конфігурацій, зображених на рис.8.7(b), тому що це призведе до того, що для великої області з постійною інтенсивністю на зображенні з'являться небажані горизонтальні чи вертикальні лінії. Число доступних рівнів інтенсивності можна збільшити за допомогою збільшення розміру клітинки. Конфігурації для клітинки 3×3 пікселів показані на рис.8.8.

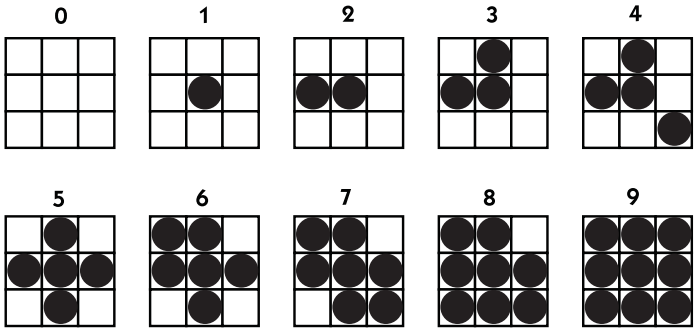


Рис.8.8. Дворівневі конфігурації 3×3

Вони дають десять рівнів інтенсивності (з 0 по 9). Клітинки конфігурацій не обов'язково повинні бути квадратними; на рис.8.9 зображена клітинка 3×2 пікселів, що дає сім (0-6) рівнів інтенсивності.

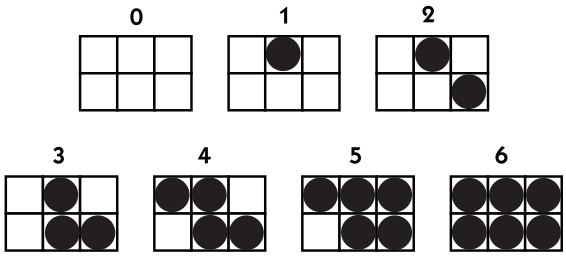


Рис.8.9. Дворівневі конфігурації 3×2

Якщо точки можуть бути різного розміру, то можна одержати додаткову кількість рівнів інтенсивності. На рис.8.10 представлені конфігурації для клітки 2×2 піксели з двома розмірами точок.

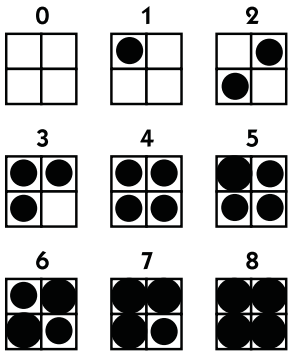


Рис.8.10. Конфігурації 2×2 з точками різних розмірів

У результаті виходить 9 рівнів інтенсивності. Подібна клітинка розміром 3×3 із точками двох розмірів дозволяє задавати 27 рівнів інтенсивності. Якщо на піксел припадає більше одного біта, то також можна одержати додаткові рівні інтенсивності. Для конфігурації 2×2 з 2 бітами на піксел отримається 13 рівнів інтенсивності. Більша кількість біт на піксел чи більший розмір клітинки дадуть відповідне збільшення числа рівнів інтенсивності.

Використання конфігурацій веде до втрати просторової роздільчої здатності, що допустимо у випадку, коли роздільча здатність зображення менша від роздільчої здатності дисплея. Розроблені також методи покращення візуальної роздільчої здатності при збереженні просторової.

Найпростіший з них полягає в застосуванні граничного значення для кожного піксела. Якщо інтенсивність зображення перевищує деяку граничну величину, то піксел вважається білим, в іншому випадку він чорний. Білий відповідає максимальній інтенсивності для дисплея, а чорний - мінімальній. Граничну величину звичайно встановлюють приблизно рівною половині максимальної інтенсивності. У точках вихідної фотографії, що відповідають кожному пікселу, інтенсивність була розбита на дискретні значення в діапазоні від 0 до 255, тобто використовувалося 8 біт. При простому граничному методі спостерігається втрата великої кількості дрібних деталей. Особливо це помітно для волосся і рис обличчя. Дрібні деталі губляться через відносно великі похибки виведеної інтенсивності для кожного піксела.

У методі, розробленому Флойдом і Стейнбергом, ця похибка розподіляється на навколишні піксели. Розподіл похибки відбувається завжди вниз і вправо. Отже, при генерації зображення в порядку сканування повертатися назад не потрібно.

Для порога рівного середньому між мінімальною і максимальною інтенсивностями, Т=(Білий+Чорний)/2, алгоритм описується в наступний спосіб:

Алгоритм розподілу похибки Флойда - Стейнберга

- Xmin, Xmax, Ymin, Ymax - *межі растра* T=(Чорний+Білий)/2 for y=Ymax to Ymin step -1 *для кожного піксела на рядку (зліва направо)* for x=Xmin to Xmax
- *визначаємо виведене значення піксела для граничної величини* T і *обчислюємо похибку* if |*I(x, y)*|<T then
- *Піксел (x, y)=Чорний Похибка=I(x, y) - Чорний else*
- *Піксел (x, y)=Білий Похибка=I(x, y) - Білий end If*

*зображуємо піксел Display Піксел (x, y) розподіляємо похибку на сусідні піксели: I(x+1, y)=I(x+1, y)+3*Похибка/8 (вправо), I(x, y-1)=I(x, y-1) +3*Похибка/8 (вниз), I=I(x+1, y-1)+Похибка/4 (вниз по діагоналі).*

Розподіл похибки на сусідні піксели покращує вигляд деталей зображення, тому що інформація, закладена в зображенні, не губиться.

Існує інший метод покращення візуальної роздільчої здатності для дворівневих дисплеїв без зменшення просторової роздільчої здатності - метод зрушення. У зображення вводиться випадкова похибка, що додається до інтенсивності кожного піксела до її порівняння з обраною граничною величиною. Додавання цілком довільної похибки не призводить до оптимального результату. Тим не менше існує оптимальна адитивна матриця похибки, яка мінімізує ефекти появи фактури на зображенні. Матриця похибки додається до зображення таким самим способом, як розташовані клітинки на шахівниці. Даний метод називається впорядкованим зрушенням. Мінімальна матриця впорядкованого зрушення має розмір 2×2. Оптимальну 2×2-матрицю першим запропонував Лім [9]. Матриці 4×4, 8×8 і більших розмірів одержують за допомогою рекурентних співвідношень.

Як показують два цих приклади, з матриці зрушення можна утворити n^2 інтенсивностей. Із збільшенням *n* зображення не втрачає просторової роздільчої здатності. Наведемо алгоритм впорядкованого зрушення.

Алгоритм упорядкованого зрушення

Xmin, Xmax, Ymin, Ymax - *межі растра*

Mod - *функція, що повертає залишок від цілочисельного ділення першого аргументу на другий*

for y=Ymax to Ymin step -1 *для кожного піксела на рядку (зліва направо)*
    for x = Xmin to Xmax
        визначаємо позицію в матриці зрушення
        i=(x **Mod** n)+1
        j=(y **Mod** n)+1
        визначаємо виведене значення піксела
        if I(x, y)<I(i, j) **then** Піксел (x, y)=Чорний
        else Піксел (x, y)=Білий
        end if
        зображуємо піксел Display Піксел (x, y)
    next x
next y
finish

Лекція 9

Відсічення: двовимірні, тривимірні, відсічення багатокутників

9.1. Відсічення

Відсічення, тобто процес видалення деякої частини бази даних, відіграє важливу роль у задачах машинної графіки. Відсічення використовується і для усунення ступінчастості, крім свого більш звичного застосування для отримання необхідної інформації, що необхідна для візуалізації конкретної сцени чи виду, як частини більш ширшої обстановки. Окрім того, відсічення застосовується в алгоритмах видалення невидимих ліній і поверхонь, при побудові тіней, а також при формуванні фактури. Алгоритми і поняття, розглянуті тут, можна застосувати для створення більш досконалих алгоритмів, що відтинають багатогранники іншими багатогранниками. Такі алгоритми можна використовувати для реалізації булевих операцій, які потрібні в простих системах геометричного моделювання, наприклад при обчисленні перетинань і об'єднанні простих тіл, обмежених площинами чи поверхнями другого порядку. Отримувані при цьому наближені рішення використовуються у багатьох додатках.

Алгоритми відсічення бувають:

1. Двовимірними;
2. Тривимірними.

Ці алгоритми застосовуються як до регулярних, так і до нерегулярних областей і об'ємів. Під регулярними розуміються канонічні області й об'єми. Зокрема, до них відносяться прямокутники і паралелепіпеди зі сторонами, паралельними до осей координат.

Ці алгоритми можна реалізувати:

1. Апаратно;
2. Програмно.

Алгоритми відсічення, реалізовані програмно, найчастіше виявляються недостатньо швидкодіючими для додатків, орієнтованих на процеси, що протікають у реальному часі. Тому трьох- і двовимірні алгоритми відсічення реалізуються апаратними чи мікропрограмними засобами. У подібних реалізаціях звичайно обмежуються дво- чи тривимірними відсіченнями типових форм. Однак, з появою нових інтегральних схем (НПІС) відкриваються можливості для більш загальної реалізації, що дозволяють працювати в реальному часі як з регулярними, так і з нерегулярними областями і тілами.

9.2. Двовимірне відсічення

На рис.9.1 показана плоска сцена і відтинаюче вікно регулярної форми. Вікно задається лівим (Л), правим (П), верхнім (В) і нижнім (Н) двовимірними ребрами. Регулярним відтинаючим вікном є прямокутник, сторони якого паралельні осям координат об'єктного простору чи осям координат екрана. Метою алгоритму відсічення є визначення тих точок, чи частин відрізків, що лежать всередині відтинаючого вікна. Ці точки, чи частини відрізків залишаються для візуалізації. А все інше відкидається.

Оскільки в звичайних сценах чи картинках необхідно відтинати велике число відрізків чи точок, то ефективність алгоритмів відсічення стає

особливо важливою. У багатьох випадках переважна більшість точок чи відрізків лежить цілком всередині чи поза відтинаючим вікном. Тому важливо вміти швидко відбирати відрізки, подібні *ab*, чи точки, подібні *p*, і відкидати відрізки, подібні *ij*, чи точки, подібні *q* на рис.9.1.

Точки, що лежать всередині відтинаючого вікна задовольняють умові:

$$x_л \le x \le x_п \text{ і } y_н \le y \le y_в.$$

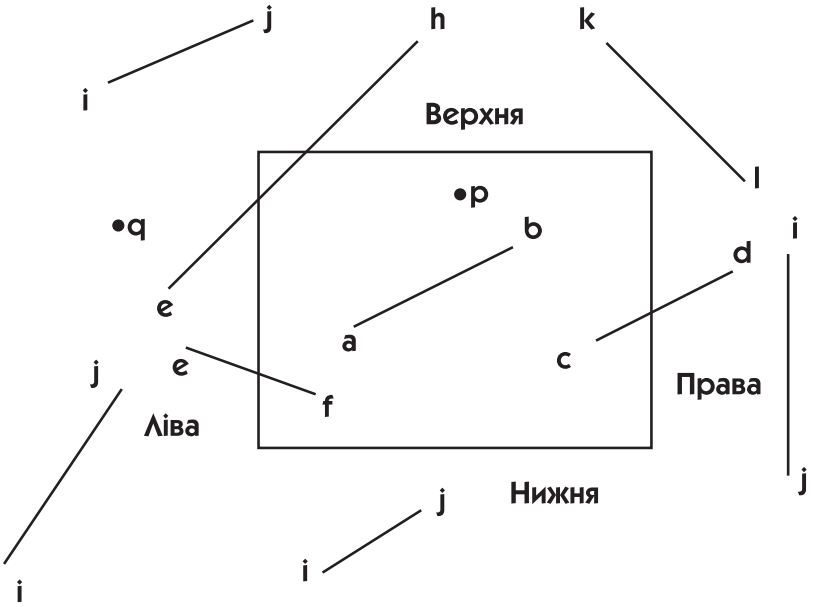


Рис.9.1 Двомірне відтинаюче вікно

Знак рівності тут показує, що точки, які лежать на границі вікна, вважаються такими, що знаходяться всередині вікна.

Відрізок лежить всередині вікна , отже, є видимим, якщо обидві його кінцеві точки лежать всередині вікна, наприклад відрізок *ab* на рис.9.1. Однак якщо обидва кінці відрізка лежать поза вікном, то цей відрізок не обов'язково лежить цілком поза вікном, наприклад відрізок *gh* на рис.9.1. Якщо ж обидва кінці відрізка лежать праворуч, ліворуч, вище чи нижче вікна, то цей відрізок цілком лежить поза вікном, а отже, є невидимим. Перевірка останньої умови відкине всі відрізки, позначені *ij* на рис.9.1. Але вона не відкине ні відрізка *gh,* що видимий частково, ні відрізка *kl,* що цілком невидимий.

Нехай *a* і *b* - кінці відрізка, тоді можна запропонувати простий алгоритм визначення видимості, що визначає всі цілком видимі і більшість невидимих відрізків.

Простий алгоритм визначення видимості:

a, b - *кінцеві точки відрізка в координатному просторі* (x, y)

для кожного відрізка перевірити повну видимість відрізка, якщо одна з координат якого-небудь кінця відрізка знаходиться поза вікном, то відрізок не є цілком видимим

if $x_a<x_л$ or $x_a>x_п$ **then** 1
if $x_b<x_л$ or $x_b>x_п$ **then** 1
if $y_a<y_н$ or $y_a>y_в$ **then** 1
if $y_b<y_н$ or $y_b>y_в$ **then** 1
відрізок цілком видимий
візуалізувати відрізок
go to 3
перевірити повну невидимість відрізка
якщо обидва кінці відрізка лежать ліворуч, праворуч, зверху чи знизу від вікна, то факт невидимості відрізка тривіальний
1 **if** $x_a<x_л$ and $x_b<x_л$ **then** 2
if $x_a>x_п$ and $x_b>x_п$ **then** 2
if $y_a>y_в$ and $y_b>y_в$ **then** 2
if $y_a<y_н$ and $y_b<y_н$ **then** 2
відрізок частково перетинає продовження діагоналі, залишаючись невидимим
визначити перетинання відрізка з вікном
2 *відрізок невидимий*
3 *перехід до наступного відрізка*

Тут через $x_л$, $x_п$, $y_в$, $y_н$ позначені координати *x* і *y* лівого, правого, верхнього і нижнього країв вікна відповідно. Порядок проведення порівнянь при визначенні видимості чи невидимості несуттєвий. Для деяких відрізків може знадобитися проведення всіх чотирьох порівнянь, перш ніж визначиться їхня повна видимість чи невидимість. Для інших відрізків може знадобитися тільки одне порівняння. Несуттєво також, що виявляється раніше - цілком видимі чи цілком невидимі відрізки. Однак, оскільки визначення перетинання відрізка з вікном вимагає великого обсягу обчислень, його варто проводити в останню чергу.

Наведені вище тести повної видимості чи невидимості відрізків можна формалізувати, використовуючи метод Д. Коена й А. Сазерленда. У цьому методі для визначення тієї з дев'яти областей, якій належить кінець ребра, відводиться чотирьохрозрядний (бітовий) код. Коди цих областей показані на рис.9.2.

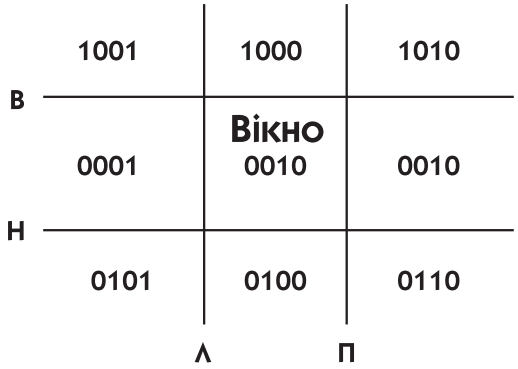


Рис.9.2. Коди областей, яким належать кінцеві точки

Крайній правий біт коду вважається першим. У відповідний біт заноситься 1 при виконанні наступних умов:

для першого біта - якщо точка лівіше вікна

для другого біта - якщо точка правіше вікна

для третього біта - якщо точка нижче вікна

для четвертого біта - якщо точка вище вікна

У противному випадку в біт заноситься нуль. Звідси, очевидно, випливає що, якщо коди обох кінців ребра дорівнюють нулю, то обидві ці точки лежать всередині вікна, і відрізок видимий. Коди кінцевих точок можна використовувати також і для тривіального відкидання цілком невидимих відрізків.

Розглянемо таблицю істинності еквівалентну логічному оператору "і":

Істина і Хибність	Хибність	Хибність =0	1 і 0 R 0
Хибність і Істина	Хибність		0 і 1 R 0
Хибність і Хибність	Хибність	Істина=1	0 і 0 R 0
Істина і Істина	Істина		1 і 1 R 1

Якщо побітний логічний добуток кодів кінцевих точок відрізка не дорівнює нулю, то відрізок цілком невидимий і його можна відкинути. Приклади, наведені у табл.9.1, ілюструють ці твердження.

Таблиця 9.1. Коди кінців відрізків

Відрізок (рис.9.1)	Коди кінців (рис.9.2)	Результати логічного множення	Примітка
ab	0000 0000	0000	Цілком видимий
ij	0010 0110	0010	Цілком невидимий
ij	1001 1000	1000	-”-
ij	0101 0001	0001	-”-
ij	0100 0100	0100	-”-
cd	0000 0010	0000	Частково видимий
ef	0001 0000	0000	-”-
gh	0001 1000	0000	-”-
kl	1000 0010	0000	Цілком невидимий

З табл.9.1 видно що, якщо результат логічного множення не дорівнює нулю, то фактично відрізок буде цілком невидимий. Однак якщо логічний добуток дорівнює нулю, то відрізок може виявитися цілком чи частково видимим, або навіть цілком невидимим. Тому для визначення повної видимості необхідно перевіряти значення кодів обох кінців відрізка окремо.

Перевірку значень кодів кінців відрізка можна легко реалізувати, якщо скористатися підпрограмами, що оперують бітами. Однак в алгоритмах, що розглядаються нижче, такі підпрограми не використовуються.

Якщо спочатку знайдені цілком видимі і, очевидно, невидимі відрізки, то підпрограмі, що обчислює перетинання відрізків, передаються тільки відрізки, що, можливо, частково видимі, тобто ті, для яких результат логічного множення кодів їхніх кінцевих точок дорівнює нулю. Звичайно, ця підпрограма визначає і передані їй цілком невидимі відрізки.

Перетинання двох відрізків можна шукати як параметричним, так і непараметричним способом. Очевидно, що рівняння нескінченної прямої, що проходить через точки $P_1(x_1, y_1)$ і $P_2(x_2, y_2)$, має вигляд
$y=m(x-x_1)+y_1$, чи $y=m(x-x_2)+y_2$, де $m=(y_2-y_1)/(x_2-x_1)$ - це нахил даної прямої.

Точки перетинання цієї прямої зі сторонами вікна мають наступні координати:

з лівою: $x_л$, $y=m(x_л-x_1)+y_1$

з правою: $x_п$, $y=m(x_п-x_1)+y_1$

з верхньою: $y_в$, $x=x_1+(1/m)(y_в-y_1)$

з нижньою: $y_н$, $x=x_1+(1/m)(y_н-y_1)$

У прикладі 9.3 показано, як цей простий метод дозволяє відкинути некоректні перетинання шляхом простого порівняння координат точок перетинання з координатами сторін вікна.

9.3. Приклад простого двовимірного відсічення

Розглянемо відтинаюче вікно і відрізки, зображені на рис.9.3.

Нахил відрізка від $P_1(-3/2, 1/6)$ до $P_2(1/2, 3/2)$ дорівнює $m=(y_2-y_1)/(x_2-x_1)=(3/2-1/6)/[1/2-(-3/2)]=2/3$.

Його перетинання зі сторонами вікна такі:

з лівою: .x=-1; y=(2/3)[-1-(-3/2)]+1/6=1/2

з правою: x=1; y=(2/3)[1-(-3/2)]+1/6=11/6

(останнє число більше, ніж $y_в$ і тому відкидається)

з верхньою: y=1; x=-3/2+(3/2)[1-1/6]=-1/4

з нижньою: y=-1; x=-3/2+(3/2)[-1-1/6]=-13/4

(останнє число менше, ніж $x_л$, і тому відкидається).

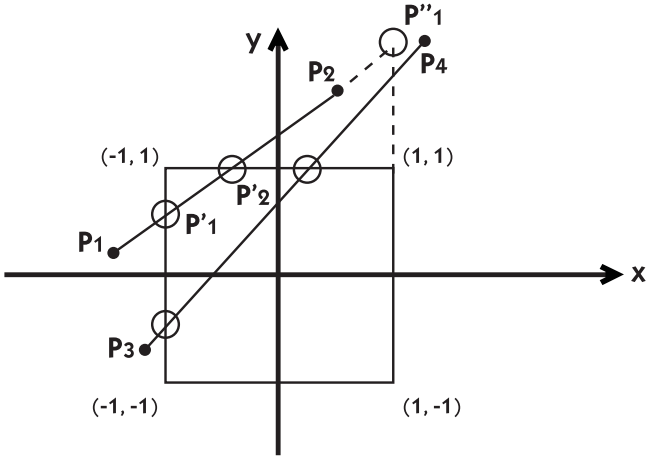


Рис.9.3. Двовимірне параметричне відсічення

Аналогічно, відрізок від $P_3(-3/2, -1)$ до $P_4(3/2, 2)$ має $m=(y_2-y_1)/(x_2-x_1)=(2-(-1))/[3/2-(-3/2)]=1$.

Його перетинання:

з лівою: $x=-1$; $y=(1)[-1-(-3/2)]+(-1)=-1/2$

з правою: $x=1$; $y=(1)[1-(-3/2)]+(-1)=3/2$

(останнє число більше, ніж $y_в$, і тому відкидається)

з верхньою: $y=1$; $x=-3/2+(1)[1-(-1)]=1/2$

з нижньою: $y=-1$; $x=-3/2+(1)[-1-(-1)]=-3/2$

(останнє число менше, ніж $x_л$, і тому відкидається).

Щоб розробити схему ефективного алгоритму відсічення, необхідно спочатку розглянути декілька часткових випадків. Нагадаємо, що, як вже вказувалося, якщо нахил нескінченний, то відрізок паралельний лівій і правій сторонам вікна і треба шукати його перетинання тільки з верхньою і нижньою сторонами.

Аналогічно, якщо нахил дорівнює нулю, то відрізок паралельний верхній і нижній сторонам вікна, і шукати його перетинання треба тільки з лівою і правою сторонами. Нарешті, якщо код одного з кінців відрізка дорівнює нулю, то цей кінець лежить всередині вікна, і тому відрізок може перетнути тільки одну сторону вікна.

Простий алгоритм двовимірного відсічення

P_1, P_2 - *кінцеві точки відрізка*

P_1', P_2'- *кінцеві точки видимої частини відрізка*

$x_л$, $x_п$, $y_в$, $y_н$ - *координати лівої, правої, верхньої і нижньої сторін вікна*

Прапор - *ознака видимості, рівний:* 0, *видимість*; - 1, *невидимість*

обчислення кодів кінцевих точок

занесення цих кодів у два масиви Т1код і Т2код, *розмірністю* 1 х 4 *кожен*

для першого кінця: P_1

if $x_1<x_л$ **then** Т1код(4)=1

else Т1код(4)=0

if $x_1>x_п$ **then** Т1код(3)=1

else Т1код(3)=0

if $y_1<y_н$ **then** Т1код(2)=1

else Т1код(2)=0

if $y_1>y_в$ **then** Т1код(1)=1 **else** Т1код(1)=0

для другого кінця: P_2

if $x_2<x_л$ **then** Т2код(4)=1

else Т2код(4)=0

if $x_2>x_п$ **then** Т2код(3)=1

else Т2код(3)=0

if $y_2<y_н$ **then** Т2код(2)=1

else Т2код(2)=0

if $y_2>y_в$ **then** Т2код(1)=1

else Т2код(1)=0

ініціалізація ознаки видимості і невидимості кінцевих точок

ініціалізація m дуже великим числом, що імітує нескінченний нахил

Прапор=0

```
  P1'=P1
  P2'=P2
  m=Велике число
  перевірка повної видимості відрізка
  Сума1=0
  Сума2=0
  for i=1 to 4
      Сума1=Сума1+Т1код(i)
      Сума2=Сума2+Т2код(i)
  next i
  if Сума1=0 and Сума2=0 then 7
  відрізок не є цілком видимим
  перевірка випадку тривіальної невидимості
  обчислення логічного добутку (Добуток) кодів кінцевих точок відрізка
  Добуток=0
  for i=1 to 4
      Добуток=Добуток+Ціла частина((Т1код(i)+Т2код(i))/2)
      if Добуток<>0 then
        Прапор=-1
        go to 7
      end if
  next i
  відрізок може бути частково видимим
  перевірка попадання першої точки всередину вікна
  if Сума1=0 then
    Номер=1
    P1'=P1
    P=P2
    go to 2
  end if
  перевірка попадання другої точки всередину вікна
  if Сума2=0 then
    Номер=2
    P1'=P2
    P=P1
    go to 2
  end if
  всередині вікна немає кінців відрізка
  ініціалізація номера кінця відрізка
  Номер=0
1 if Номер<>0 then P'номер=P
  Номер=Номер+1
  if Номер>2 then 7
  P=Pномер
 перевірка перетинання з лівим краєм
 перевірка вертикальності відрізка
```

2 **if** $(x_2-x_1)=0$ **then** 4
  $m=(y_2-y_1)/(x_2-x_1)$
  if $x_л<P_x$ **then** 3
  $y=m*(x_л-P_x)+P_y$
  if $y>y_в$ **then** 3
  if $y<y_н$ **then** 3
  виявлено коректне перетинання
  $P_y=y$
  $P_x=x_л$
  go to 1
  перевірка перетинання з правим краєм
3 **if** $x_п>P_x$ **then** 4
  $y=m*(x_п-P_x)+P_y$
  if $y<y_в$ **then** 4
  if $y<y_н$ **then** 4
  виявлено коректне перетинання
  $P_y=y$
  $P_x=x_п$
  go to 1
  перевірка перетинання з верхнім краєм
  перевірка горизонтальності відрізка
4 **if** $m=0$ **then** 1
  if $y_в>P_y$ **then** 5
  $x=(1/m)*(y_в-P_y)+P_x$
  if $x<x_л$ **then** 5
  if $x>x_п$ **then** 5
  виявлено коректне перетинання
  $P_x=x$
  $P_y=y_в$
  go to 1
  перевірка перетинання з нижнім краєм
5 **if** $y_н>P_y$ **then** помилка
  $x=(1/m)*(y_н-P_y)+P_x$
  if $x<x_л$ **then** 6
  if $x>x_п$ **then** 6
  виявлено коректне перетинання
  $P_x=x$
  $P_y=y_н$
  go to 1
  відрізок невидимий
6 Прапор=-1
  завершення роботи і виклик процедури побудови
7 **if** Прапор=-1 **then** 8
  Draw $P_1'P_2'$
  перейти до обробки наступного відрізка
8 **finish**

9.4. Алгоритм відсічення Сазерленда - Коена, заснований на розподілі відрізка

Алгоритм, описаний у попередньому розділі, аналогічний тому, що запропонували Коен і Сазерленд. В алгоритмі двовимірного відсічення відрізок відтинався по черзі кожною зі сторін вікна, а для отриманих точок перетинання перевірялася їхня приналежність внутрішній області вікна, тобто коректність перетинання. Ця процедура застосовувалася спочатку до відрізка P_1P_2 і отримувався відрізок P'_1P_2, а потім до відрізка P'_1P_2 і отримувався результуючий відрізок $P'_1P'_2$.

В алгоритмі Сазерленда - Коена відрізок теж розбивається сторонами вікна. Відмінність полягає в тому, що тут не проводиться перевірка попадання точки перетинання всередину вікна, замість цього кожна з пари утворених частин відрізка зберігається чи відкидається в результаті аналізу кодів їх кінцевих точок. Розгляд відрізка P_1P_2 на рис.9.3 показує складність реалізації цієї ідеї. Якщо P_1P_2 розбивається лівою стороною вікна, то виходить два нових відрізки $P_1P'_1$ і P'_1P_2. Коди кінцевих точок кожного з цих відрізків такі, що обидва вони можуть бути частково видимі. Отже, жоден з них не можна відкинути як невидимий чи залишити як видимий. Ключем до алгоритму Сазерленда - Коена є інформація про те, що одна з кінцевих точок відрізка лежить поза вікном. Тому той відрізок, що міститься між цією точкою і точкою перетинання, можна відкинути як невидимий. Фактично це означає заміну вихідної кінцевої точки на точку перетинання.

Алгоритм Сазерленда - Коена формулюється в такий спосіб:

Для кожної сторони вікна виконати:

для кожного відрізка P_1P_2 визначити, чи є він цілком видимим чи може бути відкикутий як тривіально невидимий;

якщо P_1 поза вікном, то продовжити виконання, інакше поміняти P_1 і P_2 місцями;

замінити P_1 на точку перетинання P_1P_2 зі стороною вікна.

Алгоритм двовимірного відсічення Сазерленда - Коена

Вікно - *масив 1 × 4, що містить координати ($x_л$, $x_п$, $y_н$, $y_в$) сторін вікна.*

P_1, P_2 - *кінцеві точки відрізка з координатами (P_{1x}, P_{1y}) і (P_{2x}, P_{2y}) відповідно.*

Т1код, Т2код - *масиви 1 × 4, що містять коди точок P_1 і P_2 .*

Прапор - *індикатор координатної орієнтації відрізка, рівний: -1 - при вертикальності, 0 - при горизонтальності.*

ініціалізація Прапор

Прапор=1

перевірка вертикальності і горизонтальності відрізка

if P_{2x}-P_{1x}=0 **then** Прапор=-1

else *обчислення нахилу* Нахил=(P_{2y}-P_{1y})/(P_{2x}-P_{1x})

if Нахил=0 **then** Прапор=0

end if

для кожної сторони вікна

for i=1 to 4

call Коен(P_1, P_2, Вікно; Видимість)
if Видимість=так **then** 2
if Видимість=ні **then** 3
перевірка перетинання відрізка зі стороною вікна
if Т1код(5 -i)=Т2код(5-i) **then** 1
перевірка перебування P_1 поза вікном; якщо P_1 всередині вікна, то поміняти P_1 і P_2 місцями
if Т1код(5-i)=0 **then**
$P_{аб}$=P_1
P_1=P_2
P_2=$P_{аб}$
end if
пошук перетинань відрізка зі сторонами вікна
вибір відповідної підпрограми обчислення перетинання
контроль вертикальності відрізка
if Прапор<>-1 and i<=2 **then**
P_{1y}=Нахил*($Вікно_i$-P_{1x})+P_{1y}
P_{1x}=$Вікно_i$
else if Прапор<>0 **then**
if Прапор<>-1 **then** P_1x=(1/Нахил)*($Вікно_i$-P_{1y})+P_{1x}
end if
P_{1y}=$Вікно_i$
end if
end if
next i {1 крок}
побудувати видимий відрізок
Draw P_1 P_2 {2 крок}
finish {3 крок}
підпрограма визначення видимості відрізка
subroutine Коен(P_1, P_2, Вікно; Видимість)
P_1, P_2 - *кінцеві точки відрізка з координатами (P_{1x}, P_{1y}) і (P_{2x}, P_{2y}) відповідно.*
Вікно - *масив 1 × 4, що містить координати ($x_л$, $x_п$, $y_н$, $y_в$) сторін вікна.*
Видимість - *ознака видимості відрізка рівна:*
"ні", "так", "частково" - якщо відрізок відповідно :
цілком невидимий, видимий чи частково видимий
обчислення кодів кінцевих точок відрізка
call Кінець(P_1, Вікно; Т1код, Сума1)
call Кінець(P_2, Вікно; Т2код, Сума2)
припустимо, що відрізок частково видимий
Видимість=частково
перевірка повної видимості відрізка
if Сума1=0 and Сума2=0 **then** Видимість=так
else *перевірка тривіальної невидимості відрізка*
call Логічне(Т1код, Т2код, Добуток)
if Добуток<>0 **then** Видимість=ні

end if
відрізок може виявитися частково видимим
return
підпрограма обчислення кодів кінцевої точки відрізка
subroutine Кінець(P, Вікно; Ткод, Сума)
P_x, P_y - *координати точки P*
Вікно - *масив 1×4, що містить координати ($x_л$, $x_п$, $y_н$, $y_в$) сторін вікна.*
Ткод - *масив 1×4, що містить коди кінцевої точки.*
Сума - *сума всіх елементів масиву Ткод*
обчислення кодів кінцевої точки
if $P_x<x_л$ **then** Ткод(4)=1
else Ткод(4)=0
if $P_x>x_п$ **then** Ткод(3)=1
else Ткод(3)=0
if $P_y<y_н$ **then** Ткод(2)=1
else Ткод(2)=0
if $P_y>y_в$ **then** Ткод(1)=1
else Ткод(1)=0
обчислення суми
Сума=0
for i=1 to 4
    Сума=Сума+Ткод(i)
next i
return
підпрограма обчислення логічного добутку
subroutine Логічне(Т1код, Т2код; Добуток)
Т1код, Т2код - *масиви 1×4, що містять коди першої і другої кінцевих точок відповідно.*
Добуток - *сума побітних логічних добутків.*
Добуток=0
for i=1 to 4
    Добуток=Добуток+Ціла частина((Т1код(i)+Т2код(i))/2)
next i
return

Цей алгоритм ілюструє наступний приклад.

9.5. Приклад роботи алгоритму відсічення Сазерленда - Коена

Розглянемо відсічення відрізка P_1P_2 вікном, показаним на рис.9.3. Коди кінцевих точок P_1(- 3/2, 1/6) і P_2(1/2, 3/2) рівні (0001) і (1000) відповідно. Цей відрізок не є ні цілком видимим, ні тривіально невидимим.

Відрізок перетинає ліву сторону вікна. P_1 - поза вікном.

Перетинання з лівої сторони (x=-1) вікна відбувається в точці P'_1(-1, 1/2). Заміна P_1 на P'_1 дає новий відрізок від P_1(-1, 1/2) до P_2(1/2, 3/2).

Коди кінцевих точок P_1 і P_2 тепер стали (0000) і (1000). Відрізок не є ні цілком видимим, ні тривіально невидимим.

Відрізок не перетинається з правою стороною вікна. Перейти до нижньої сторони.

Коди кінцевих точок P_1 і P_2 залишаються як і раніше рівними (0000) і (1000). Відрізок не є ні цілком видимим, ні тривіально невидимим.

Відрізок не перетинається з нижньою стороною вікна. Перейти до верхньої сторони.

Коди кінцевих точок P_1 і P_2 залишаються рівними (0000) і (1000). Відрізок не є ні цілком видимим, ні тривіально невидимим.

Відрізок не перетинається з верхньою стороною вікна. P_1 - не зовні вікна. Помінявши P_1 і P_2 місцями, одержуємо новий відрізок від $P_1(1/2, 3/2)$ до $P_2(-1, 1/2)$.

Точка перетинання з верхньою стороною вікна (y=-1) дорівнює $P'_1(-1/4, 1)$.

Замінивши P_1 на P'_1, одержуємо новий відрізок від $P_1(-1/4, 1)$ до $P_2(-1, 1/2)$. Тепер коди кінцевих точок P_1 і P_2 рівні (0000) і (0000). Відрізок цілком видимий.

Процедура довершена. Побудувати відрізок.

9.6. Внутрішнє і зовнішнє відсічення

У попередніх розділах наголос був зроблений на відсіченні відрізка внутрішньою областю вікна. Однак існує можливість відсічення відрізка і зовнішньою його областю. Для цього треба визначити частини відрізка, що лежать поза вікном, і накреслити їх. Наприклад, видимі частини відрізка P_1P_2 показаного на рис.9.4(b), розташовані в інтервалах $0 \le t < 1/6$ і $5/6 < t \le 1$, чи від точки (-1, 1) до точки (0, 3/2), а також від точки (7/3, 8/3) до точки (3, 3). Результати як внутрішнього, так і зовнішнього відсічень цього відрізка показані на рис.9.4.

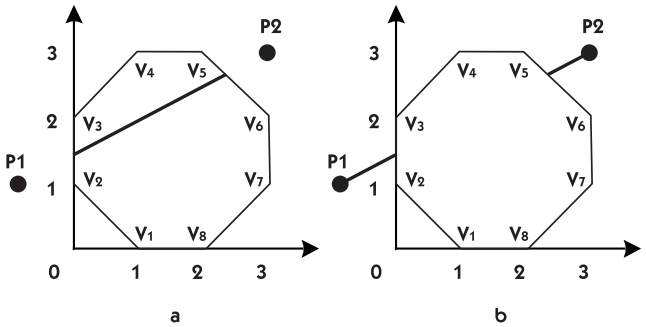


Рис.9.4. Внутрішнє і зовнішнє відсічення

Зовнішнє відсічення відіграє важливу роль у дисплеях, що допускають роботу з декількома вікнами, як це показано на рис.9.5.

На цьому рисунку пріоритет вікон 1, 2, 3 вище пріоритету вікна всього екрана, а пріоритет вікон 1 і 3 вище пріоритету вікна 2. Тому зображення у вікні екрана відтинається його власною областю і зовнішніми областями вікон 1, 2, 3. Зображення в 2 відтинається його власною областю і зовнішніми областями 1 і 3. Зображення у вікнах 1 і 3 потрібно відітнути тільки відповідними внутрішніми областями.

Зовнішнім відсіченням можна скористатися і при роботі з увігнутим полігональним вікном.

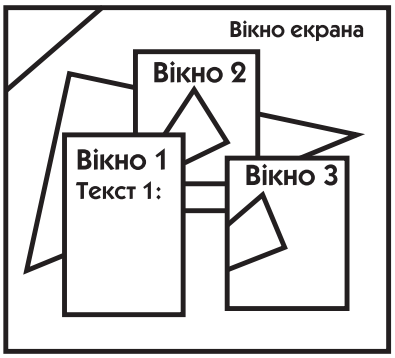


Рис.9.5 Відсічення декількома вікнами

На рис.9.6 показаний увігнутий багатокутник, заданий вершинами V_1 V_2 V_3 V_4 V_5 V_6 V_1. Цей увігнутий багатокутник можна перетворити в опуклий шляхом з'єднання вершин V_3 і V_5, що і показано на рис.9.6 штриховим відрізком.

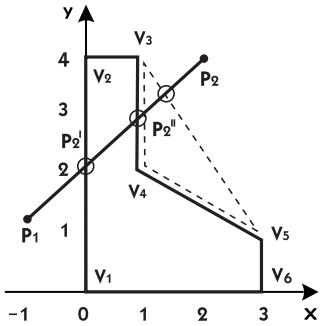


Рис9.6 Відсічення неопуклим вікном

Відрізок P_1 P_2 внутрішньо відтинається отриманим багатокутником за допомогою алгоритму Кіруса - Бека. А потім отриманий при цьому відрізок $P'_1P'_2$ зовні відтинається багатокутником V_3 V_4 V_5 V_3, що і дає шуканий результат - $P'_1P''_2$. Для роботи з алгоритмом Кіруса - Бека треба, насамперед, переконатися, що вікно є опуклим, а потім обчислити внутрішні нормалі до кожної його сторони.

Визначення факту опуклості чи неопуклості двовимірного полігонального вікна можна встановити шляхом обчислення векторних добутків його суміжних сторін.

Висновки, які можна зробити з аналізу знаків цих добутків, такі:

Усі знаки дорівнюють нулю - багатокутник вироджується у відрізок.

Є як додатні, так і від’ємні знаки - багатокутник неопуклий.

Усі знаки від’ємні - багатокутник опуклий, а внутрішні нормалі орієнтовані вліво від його контуру.

Усі знаки недодатні - багатокутник опуклий, а внутрішні нормалі орієнтовані вправо від його контуру.

Рис.9.7 ілюструє ці правила.

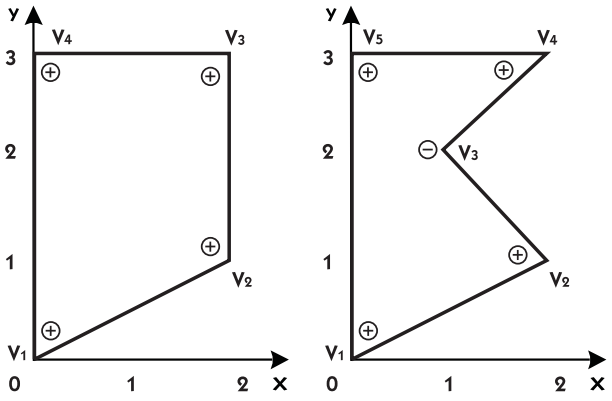


Рис 9.7 Визначення факту опуклості багатокутника
(a - опуклий, b - неопуклий.)

Інший підхід полягає в тому, що одна з вершин багатокутника може бути обрана базою, і можуть обчислюватися векторні добутки для пар векторів, що починаються в цій базі і закінчуються в послідовних вершинах багатокутника. Результати цього методу інтерпретуються точно так само.

Векторні добутки будуть перпендикулярні до площини багатокутника. Векторні добутки двох плоских векторів V_1, і V_2 дорівнюють $(Vx_1Vy_2-Vy_1Vx_2)k$, де k - одиничний вектор, перпендикулярний до площини, яка містить вектори-співмножники.

Нормаль до сторони багатокутника можна обчислити, якщо згадати, що скалярний добуток пари перпендикулярних векторів дорівнює нулю.

Якщо n_x і n_y - невідомі компоненти нормалі до відомого вектора (V_x, V_y) сторони багатокутника, то:

$$n*V=(n_xi+n_yj)*(Ve_xi+Ve_yj)=n_xVe_x+n_yVe_y=0,\ n_xVe_x=-n_yVe_y$$

Оскільки нас цікавить тільки напрямок нормалі, то припустимо, що $n_y=1$ без втрати загальності. Отже, нормаль дорівнює $n=-Ve_y/Ve_xi+j$. Якщо вектор сторони багатокутника утворений як різниця векторів пари суміжних його вершин V_{i-1} і V_i і якщо скалярний добуток нормалі і вектора від V_{i-1} до V_{i+1} позитивний, то n - внутрішня нормаль. У протилежному випадку n - зовнішня нормаль. В останньому випадку внутрішню нормаль можна одержати, помноживши n на -1. Покажемо простий приклад, який ілюструє цей метод.

9.7. Приклад, що ілюструє роботу алгоритму Кіруса - Бека

На рис.9.7(a) показаний простий опуклий багатокутник, а на рис.9.7(b) - неопуклий багатокутник. У табл.9.2 і 9.3 наведені результати всіх обчислень. Розглянемо, наприклад, векторний добуток сторін, суміжних вершині V_2 і внутрішню нормаль до сторони V_1 V_2 для багатокутника з рис.9.7(a).

Сторони, суміжні вершині V_2 рівні: $V_1V_2=2i+j$, $V_2V_3=2j$. Їх векторний добуток дорівнює $V_1V_2 \times V_2V_3=4k$, де k - одиничний вектор, перпендикулярний до площини багатокутника. Цей векторний добуток додатній.

У табл.9.2 показано, що векторні добутки додатні для всіх вершин багатокутника. Тому даний багатокутник опуклий.

Таблиця 9.2

Вершина	Вектори	Векторний добуток
V1	V4V1×V1V2	[0 -3]×[2 1]=+6
V2	V1V2×V2V3	[2 1]×[0 2]=+4
V3	V2V3×V3V4	[0 2]×[-2 0]=+4
V4	V3V4×V4V1	[-2 0]×[0 -3]=+6

Таблиця 9.3

Вершина	Вектори	Векторний добуток
V1	V5V1×V1V2	[0 -3]×[2 1]=+6
V2	V1V2×V2V3	[2 1]×[-1 1]=+3
V3	V2V3×V3V4	[-1 1]×[1 1]=-2
V4	V3V4×V4V5	[1 1]×[-2 0]=+2
V5	V4V5×V5V1	[-2 0]×[0 -3]=+6

У табл.9.3. показано, що для багатокутника з рис.9.7(b) векторний добуток сторін, суміжних вершині V_3, від'ємний, у той час як для всіх інших вершин він додатній. Отже, цей багатокутник неопуклий.

Нормаль до вектора сторони V_1V_2 дорівнює n=-1/2i+j, чи n=-i+2j. Вектор V_1V_3 дорівнює 2i+3j. Отже, $n*V_1V_3$=(-i+2j)*(2i+3j)=4>0, тому отримана внутрішня нормаль.

Розглянемо розбиття неопуклих багатокутників.

Просте узагальнення методу поворотів і переносів вікна, що використовується для визначення факту його опуклості чи неопуклості, дозволяє розбивати чи розділяти простий неопуклий багатокутник на декілька опуклих багатокутників.

Якщо вершини багатокутника перелічуються проти годинникової стрілки, то ця процедура матиме вигляд:

для кожної i-ї вершини багатокутника треба її перенести в такий спосіб, щоб вона збіглася з початком координат;

повернути багатокутник щодо координат за годинниковою стрілкою так, щоб (i+1)-а вершина виявилася на додатній півосі x;

проаналізувати знак ординати (i+2)-ої вершини. Якщо він не від'ємний, то багатокутник опуклий у (i+1)-й вершині. Якщо ж цей знак від'ємний, то багатокутник неопуклий; розбити його;

багатокутник розрізається вздовж додатньої півосі x, тобто шукаються такі його сторони, що перетинаються з віссю x. Утворяться два нових багатокутники: один складається з вершин, що лежать вище осі x і найближчої до початку координат точки перетинання з $x>x_{i+1}$, а другий - з вершин, що лежать нижче осі x і уже згаданої точки перетинання.

Алгоритм рекурсивно застосовується до отриманих багатокутників доти, поки усі вони не стануть опуклими.

Цей алгоритм не дає оптимальної розбивки в сенсі мінімального числа опуклих компонентів.

Крім цього, алгоритм не зможе коректно розбити багатокутник, сторони якого перетинаються між собою.

9.8. Тривимірне відсічення

Перш ніж почати узагальнення викладених методів для випадку трьох вимірів, необхідно обговорити питання про форму відтинаючої області. Двома найбільш розповсюдженими формами тривимірних відсікачів є: прямокутний паралелепіпед, тобто порожній брусок, що використовується при паралельному чи аксонометричному проектуванні, а також зрізана піраміда, яка часто називається пірамідою видимості, що використовується при центральному проектуванні. Ці форми показані на рис.9.8, у кожної з них шість граней: ліва, права, верхня, нижня, ближня і дальня. Існує, крім того, необхідність відтинати і по нестандартних областях.

Як і при двовимірному відсіченні, відрізки, що цілком видимі чи тривіально невидимі, можна ідентифікувати використовуючи узагальнення кодів кінцевих точок Коена-Сазерленда. У тривимірному випадку використовується 6-бітовий код. Знову найправіший біт коду вважається першим. У біти коду заносяться одиниці за допомогою узагальнення двовимірної процедури. Конкретно одиниця заноситься:

у перший біт - якщо кінець ребра лівіше об'єму,
в другий біт - якщо кінець ребра правіше об'єму,
у третій біт - якщо кінець ребра нижче об'єму,
у четвертий біт - якщо кінець ребра вище об'єму,
у п'ятий біт - якщо кінець ребра ближче об'єму,
у шостий біт - якщо кінець ребра далі об'єму.

У протилежному випадку у відповідні біти заносяться нулі. Отже, якщо коди обох кінців відрізка *дорівнюють нулю,* то обидва кінці видимі і відрізок теж буде цілком видимим. Аналогічно, якщо побітний логічний добуток кодів кінців відрізка *не дорівнює нулю,* то відрізок цілком невидимий. Якщо ж цей логічний добуток дорівнює нулю, то відрізок може виявитися як частково видимим, так і цілком невидимим. У цьому випадку необхідно визначати перетинання відрізка з гранями відтинаючого об'єму.

Пошук кодів точки щодо відтинаючого прямокутного паралелепіпеда є прямим узагальненням відповідного двовимірного алгоритму. Однак випадок, коли відсікачем служить зрізана піраміда, показана на рис.9.8(б), заслуговує додаткового обговорення. Один з методів полягає в перетворенні відсікача в канонічну форму, де $x_{прав}=1$, $x_{лів}=-1$, $y_{верх}=1$, $y_{низ}=-1$ при $z_{даль}=1$. Якщо $z_{ближ}=a$, де $0<a\leq 1$, а центр проекції збігається з початком лівої системи координат, то перевірка кодів кінцевих точок помітно спрощується.

У більш природньому методі, що менше спотворює форму відсікача, відрізок, що з'єднує центр проекції з центром зрізаної піраміди, з'єднується з віссю z правої системи координат, як це показано на рис.9.8(б).

Вид зрізаної піраміди зверху показаний на рис.9.9а. Рівняння прямої на площині xz, що містить проекцію правої грані відсікача, має вид:

$$x=(z-z_{цп})*x_п/(z_д-z_{цп})=za_1+a_2,$$

де $a_1=x_п/(z_д-z_{цп})$ і $a_2=-a_1z_{цп}$

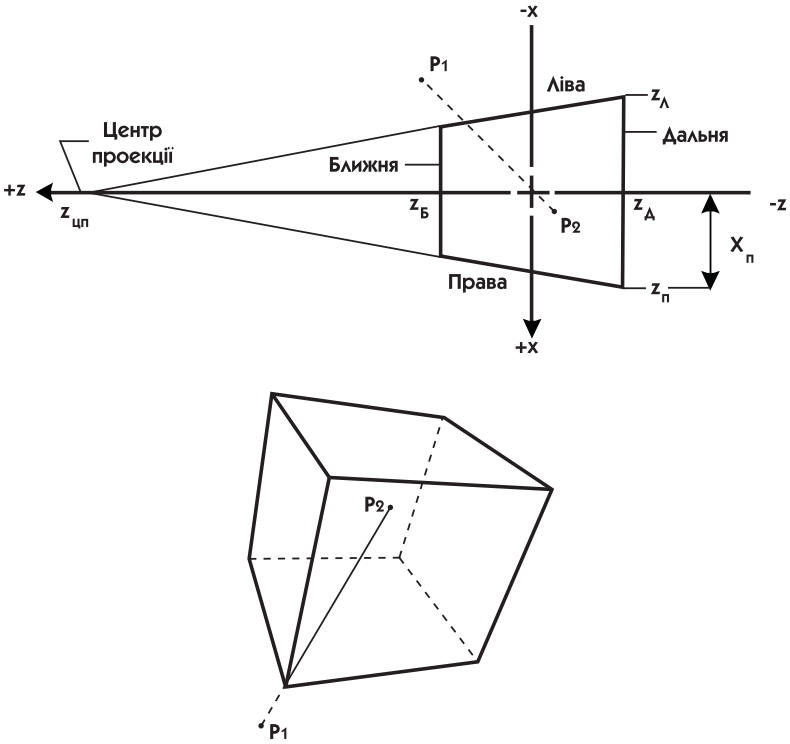


Рис.9.9. Зрізана піраміда

Рівняння цієї прямої можна використовувати для визначення місця розташування точки: праворуч, на чи ліворуч від прямої, тобто поза відсікачем, на площині, що містить його праву грань, чи всередені відсікача. Підстановка координат x і z точки P в пробну функцію правої грані дає наступний результат:

м>0, якщо P праворуч від площини

$\__п = x - za_1 - a_2$ н=0, якщо P на площині

про<0, якщо P ліворуч від площини

Пробні функції для лівої, верхньої і нижньої граней мають вид:

м>0, якщо P праворуч від площини

$\__л = zb_1 - b_2$ H=0, якщо P на площині

про<0, якщо P ліворуч від площини

де $b_1 = x_л/(z_д - z_{цп})$ і $b_2 = -b_1 z_{цп}$

м>0, якщо P вище площини

$\__у = y - zg_1 - g_2$ н=0, якщо P на площині

про<0, якщо P нижче площини

де $g_1 = y_в/(z_д - z_{цп})$ і $g_2 = -g_1 z_{цп}$

м>0, якщо P нижче площини

$\__н = y - zd_1 - d_2$ H=0, якщо P на площині

про<0, якщо P вище площини

де $d_1 = y_н/(z_д - z_{цп})$ і $d_2 = -d_1 z_{цп}$

Нарешті, пробні функції для ближньої і дальної граней мають вид:

м>0, якщо Р ближче площини
_б=z-zб н=0, якщо Р на площині
про<0, якщо Р далі площини

м>0, якщо Р далі площини
_д=z-zд н=0, якщо Р на площині
про<0, якщо Р ближче площини

Чим ближче $z_{цп}$ до нескінченності, тим більше форма відсікача наближається до прямокутного паралелепіпеда. Пробні функції при цьому теж наближаються до відповідних пробних функцій прямокутного паралелепіпеда.

Як вказували Ліанг і Панський, останній метод може дати некоректні значення кодів, якщо кінці відрізка лежать за центром проекції. Це відбувається тому, що площини, які містять ліву, праву, верхню і нижню грані зрізаної піраміди, перетинаються в точці центра проекції. Тому існують точки, розташовані одночасно лівіше лівої і правіше правої граней. Ліанг і Панський запропонували спосіб усунення цієї невизначеності. Для цього необхідно згорнути значення перших чотирьох бітів коду при $z>z_{цп}$.

9.9. Відсічення багатокутників

Багатокутник можна розглядати як набір відрізків. У додатках, зв'язаних з кресленням штрихових зображень, не надто істотно, якщо багатокутник розбитий на відрізки до його відсічення. Якщо замкнутий багатокутник відтинається, як набір відрізків, то вихідна фігура може перетворитися в один чи більш відкритих багатокутників чи просто є сукупністю розрізнених відрізків, як показано на рис.9.10.

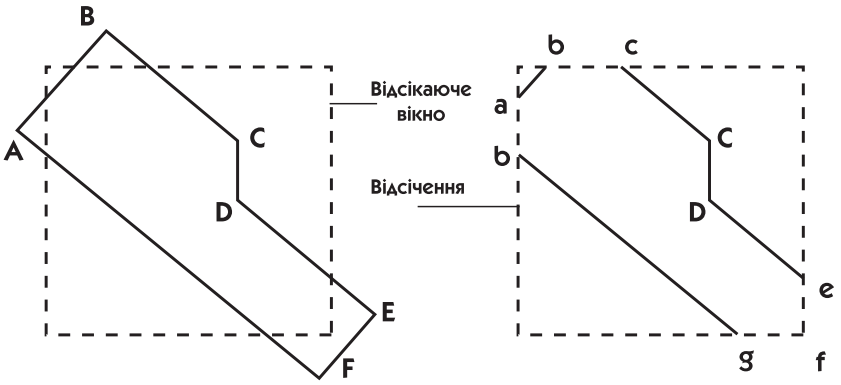


Рис 9.10 Відсічення багатокутника: відкритий багатокутник

Однак, якщо багатокутники розглядаються як суцільні області, то необхідно, щоб замкнутість зберігалася й у результаті. Для прикладу, на рис.9.10 це означає, що відрізки *bc, ef, fg* і *ha* повинні бути додані до опису результуючого багатокутника.

Додавання відрізків *ef* і *fg* не є особливо складним. Немає проблем і тоді, коли результат відсічення представляє декілька незв'язаних між собою багатокутників менших розмірів, як це показано на рис.9.11.

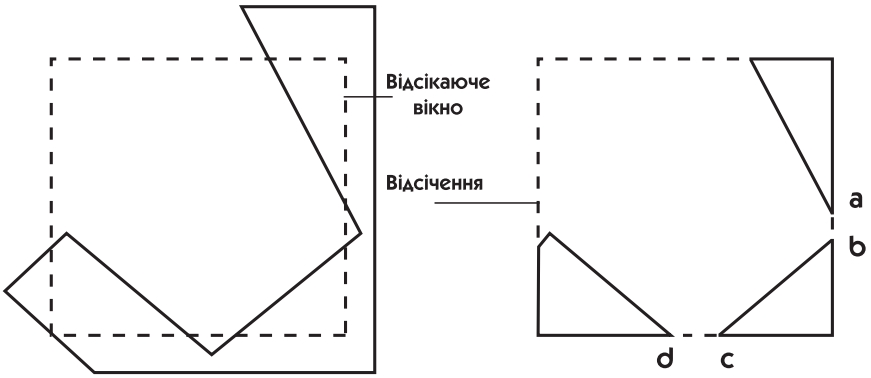


Рис 9.11 Відсічення багатокутника: не зв'язані між собою багатокутники

Наприклад, іноді відрізки *ab* і *cd,* показані на рис.9.11, включаються в опис результату. Якщо, наприклад, вихідний багатокутник заданий червоним на синьому фоні, то відрізки *ab* і *cd* теж будуть виглядати червоними на синьому фоні. Це суперечить очікуваному результату.

Розглянемо послідовне відсічення багатокутника – алгоритм Сазерленда-Ходжена. Основна ідея алгоритму Сазерленда-Ходжмена полягає в тому, що відітнути багатокутник щодо однієї прямої чи площини легко. У цьому алгоритмі вихідний і кожний із проміжних багатокутників відтинається послідовно щодо однієї прямої. Робота алгоритму для прямокутного вікна показана на рис.9.12.

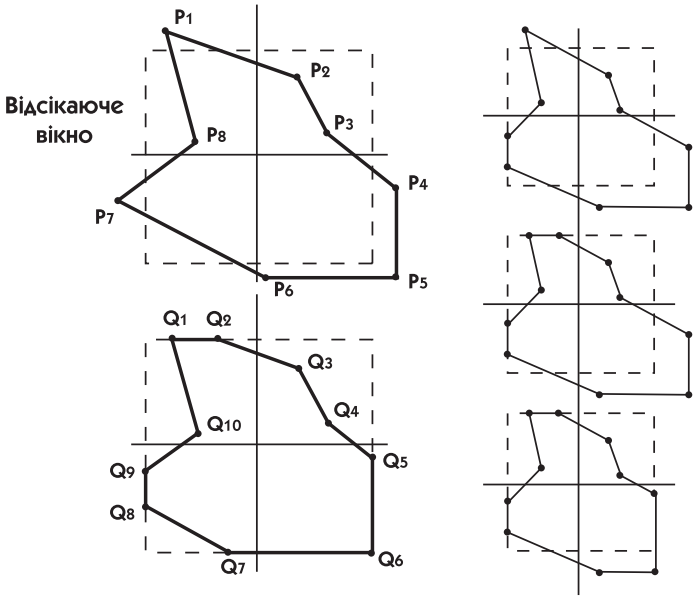


Рис.9.12 Послідовне відсічення багатокутника

Вихідний багатокутник задається списком вершин P_1, ..., P_n, що породжує список його ребер P_1 P_2, P_2 P_3, : ,P_{n-1} P_n, P_n P_1.

На рис.9.12 показано, що багатокутник спочатку відтинається лівою стороною вікна, в результаті чого одержується проміжна фігура. Потім алгоритм знову відтинає цю фігуру верхньою стороною вікна. Одержується друга проміжна фігура. Далі процес відсічення продовжується зі сторонами вікна, що залишилися. Етапи відсічення показані на рис.9.12. Додавання кутової точки Q_8 в остаточний результат відсічення тепер стало тривіальним. Цей алгоритм здатний відтинати будь-який багатокутник, опуклий чи неопуклий, плоский чи неплоский, щодо будь-якого вікна, що є опуклим багатокутником. Порядок відсічення багатокутника різними сторонами вікна непринциповий.

Результатом роботи алгоритму є список вершин багатокутника, у якого всі вершини лежать по видиму сторону від чергової відтинаючої площини. Оскільки кожна вершина багатокутника відтинається незалежно від інших, то досить розглянути тільки можливі ситуації розташування одного відрізка щодо однієї відтинаючої площини. Будемо розглядати кожну точку P зі списку вершин багатокутника, за винятком першої, як кінцеву точку ребра, початковою точкою S якого є вершина, що передує P у цьому списку. Тоді можливі тільки чотири ситуації взаємного розташування ребра і відтинаючої площини.

Результатом кожного зіставлення ребра багатокутника з відтинаючою площиною буде занесення в список вершин результуючого усіченого багатокутника, нуля, однієї чи двох вершин.

Якщо розглянуте ребро цілком видиме, то результатом буде вершина P. Заносити в результат початкову вершину S у цьому випадку не треба, тому що якщо вершини розглядаються по черзі, то S уже була кінцевою точкою попереднього ребра і тому вже потрапила в результат. Якщо ж ребро цілком невидиме, то результат не змінюється.

Якщо ребро видиме не цілком, то воно може входити чи виходити з області видимості відтинаючої площини. Якщо ребро виходить з області видимості, то треба визначити і занести в результат точку перетинання ребра і відсікаючої площини. Якщо ж ребро входить в область видимості, то варто виконати аналогічні дії. Оскільки в останньому випадку кінцева вершина P ребра видима, то вона також повинна потрапити в результат.

Для першої вершини багатокутника необхідно визначити тільки факт її видимості. Якщо вершина видима, то вона попадає в результат і стає початковою точкою S. Якщо ж вершина невидима, вона теж стає початковою точкою, але в результат не попадає.

Особливо варто розглянути останнє ребро - Pn_1. Це реалізується шляхом запам'ятовування першої вершини багатокутника в F. Тоді останнім ребром стає Pn і його можна обробляти точно так само, як і будь-яке інше ребро.

Визначення видимості точки еквівалентне визначенню тієї сторони границі відтинаючої площини, з якої лежить ця точка. Якщо ребра відтинаючого багатокутника обходяться за годинниковою стрілкою, то його внутрішня частина лежить праворуч від границі. Якщо порядок обходу протилежний, вона лежить ліворуч.

Методи визначення положення (видимості) точки щодо орієнтованого відрізка чи площини:

1. Перший метод зводиться до визначення знаку скалярного добутку вектора нормалі на вектор, що починається в довільній точці на прямій чи на площині і закінчується в пробній точці.
2. Другий метод полягає в підстановці координат пробної точки в рівняння орієнтованої прямої чи площини.
   Цей метод є варіантом алгоритму Сазерленда і Ходжмена.
3. Третій метод зводиться до перевірки знака координати z у векторного добутку двох векторів, що лежать в одній площині. Нехай дві точки P_1 і P_2 лежать на відтинаючій площині, а P_3 - це пробна точка. Ці три точки задають деяку площину, на якій лежать два вектори P_1P_2 і P_1P_3. Якщо цю площину вважати площиною xy, то у векторному добутку векторів $P_1P_3 \times P_1P_2$ ненульовою буде компонента z, рівна $(x_3-x_1)(y_2-y_1)-(y_3-y_1)(x_2-x_1)$. Якщо знак цієї компоненти z буде додатнім, нульовим чи від'ємним, то P_3 буде лежати відповідно праворуч, на чи ліворуч від прямої P_1P_2.

Усі ці методи реалізуються особливо просто для випадку прямокутних відтинаючих вікон, сторони яких паралельні координатним осям.

9.10. Приклад, який ілюструє визначення положення точки щодо площини

Розглянемо відтинаючу площину x=w=-1, що перпендикулярна осі x. Потрібно визначити положення двох точок $P_3(-2, 1)$ і $P'_3(2, 1)$ щодо даної відтинаючої площини. Використовуючи метод векторного добутку з $P_1(-1, 0)$ і $P_2(-1, 2)$, одержуємо для випадку P_3:

$$(x_3-x_1)(y_2-y_1)=[-2-(-1)]*(2-0)=-2<0,$$

це означає, що P_3 лежить ліворуч від P_1P_2. Для P'_3 маємо:

$$(x'_3-x_1)(y_2-y_1)=[2-(-1)]*2=6>0,$$

це означає, що P'_3 лежить праворуч від P_1P_2.

Метод підстановки координат особливо простий. Тут пробна функція дорівнює x-w. Для точки P_3:

$$x_3-w=-2-(-1)=-1<0,$$

для P'_3: $$x'_3-w=2-(-1)=3>0.$$

Це доводить, що P_3 і P'_3 лежать ліворуч і праворуч від P_1P_2 відповідно.

Узявши як внутрішню нормаль n=[1 0], а в ролі точки на відтинаючій площині f(-1, 0), одержуємо скалярний добуток двох векторів:

для P_3: $n*[P_3-f]=[1\ 0]*[-1\ 1]=-1<0$;

для P'_3: $n*[P'_3-f]=[1\ 0]*[3\ 1]=3>0$.

Це доводить, що P_3 розташована ліворуч, P'_3 - праворуч від відтинаючої площини.

Сазерленд і Ходжмен показали, як можна уникнути породження і запам'ятовування вершин проміжних багатокутників. Для цього замість відсічення кожного ребра (вершини) багатокутника однією площиною, що обмежує вікно, треба відтинати кожне таке ребро (вершину) послідовно всіма границями вікна. Після відсічення чергового ребра (вершини) багатокутника однією з границь вікна, алгоритм рекурсивно звертається до

самого себе, щоб відсікти результат попереднього звертання за наступною границею вікна. Ця властивість робить даний алгоритм зручнішим для апаратної реалізації.

Ліанг і Панський запропонували новий алгоритм відсічення багатокутників. Цей алгоритм у тій формі, у якій він був описаний в оригіналі, оптимально пристосований для відсічення прямокутними вікнами, однак його можна узагальнити і на випадок довільних опуклих вікон. Даний алгоритм грунтується на ідеях, запозичених з алгоритму відсічення двох- і тривимірних відрізків, що належить тим же авторам. Експерименти показали, що для прямокутних вікон цей оптимізований алгоритм працює удвічі швидше, ніж алгоритм Сазерленда-Ходжмена.

9.11. Алгоритм Сазерленда-Ходжмена для відсічення багатокутника

P - *масив вершин вихідного багатокутника*
Q - *масив вершин результуючого багатокутника*
W - *масив вершин відтинаючого вікна. Перша вершина повторюється наприкінці масиву*
N_P - *число вершин вихідного багатокутника*
N_Q - *число вершин результуючого багатокутника*
N_W - *число вершин вікна плюс одиниця*
вершини всіх багатокутників перераховуються за годинниковою стрілкою
для кожної сторони вікна виконати:

for i=1 to N_W-1
встановити лічильник вершин результату й обнулити результат
N_Q=0
Q=0
відітнути кожне ребро багатокутників по даній стороні вікна
for j=1 to N_P
особливо обробити першу вершину багатокутника
if j<>1 **then** 1
запам'ятати першу вершину
S=P_j
go to 2
перевірити факт перетинання ребром багатокутника сторони вікна
call Факт-січ(S,P_j,W_i,W_{i+1};Ознака)
if Ознака=немає **then** 2
якщо ребро перетинає сторону вікна, обчислити точку перетинання
1 **call** Перетинання(S,P_j,W_i,W_{i+1};Тсічення)
занести точку перетинання в результат
call Вихід(Тсічення,N_Q,Q)
змінити початкову точку ребра багатокутника
2 S=P_j
перевірити видимість кінцевої точки (тепер це S) ребра багатокутника

call Видимість(S,W_i,W_{i+1};Свидимість)
if Свидимість<0 **then** 3
якщо точка видима, то занести її в результат
call Вихід(S,N_Q,Q)
3 **next** j
обробити замикаюче ребро багатокутника, якщо результат порожній, то перейти до наступної сторони вікна
if N_Q=0 **then** 4
перевірити факт перетинання останнім ребром багатокутника сторони вікна
call Факт-січ(S,F,W_i,W_{i+1};Ознака)
if Ознака=немає **then** 4
факт перетинання встановлений;
обчислити точку перетинання
call Перетинання(S,F,W_i,W_{i+1};Тсічення)
вивести точку перетинання в результат
call Вихід(Тсічення,N_Q,Q)
тепер багатокутник відсічений стороною W_iW_{i+1} вікна робота алгоритму відновлюється з результатом відсічення
P=Q
4 N_P=N_Q
5 **next** i
finish
підпрограма визначення факту перетинання ребра багатокутника стороною вікна
subroutine Факт-січ(Початок, Кінець, W_1, W_2; Ознака)
визначити видимість початкової точки ребра багатокутника
call Видимість(Початок, W_1, W_2, Твидимість)
Твидимість1=Твидимість
визначити видимість кінцевої точки ребра багатокутника
call Видимість(Кінець, W_1, W_2, Твидимість)
Твидимість2=Твидимість
вважається, що ребро багатокутника, що починається чи закінчується на стороні вікна, не перетинається з нею.
Ця точка повинна бути занесена в результат раніше
if Твидимість1<0 and Твидимість2 >0 or Твидимість1>0 and Твидимість2<0 **then** Ознака=так
else Ознака=ні
end if
return
підпрогама визначення видимості точки
subroutine Видимість(Точка, P_1, P_2; Твидимість)
видимість Точка *варто визначити щодо сторони* P_1P_2
<0, *якщо* Точка *невидима*
Твидимість =0, *якщо* Точка *лежить на стороні* P_1P_2
>0, *якщо* Точка *видима*

у цій підпрограмі використовується обчислення векторного добутку
Sign - *функція, що приймає значення* -1, 0, 1 *у залежності від того, чи буде знак її аргументу від'ємний, дорівнювати нулю чи додатній.*
Раб1=(Точка x-P1x)*(P2y-P1y)
Раб2=(Точка y-P1y)*(P2x-P1x)
Раб3=Раб1-Раб2
Твидимість=sign(Раб3)
Return
підпрограма обчислення точки перетину двох відрізків
Subroutine Перетинання(P_1,P_2,W_1,W_2;Тсічення)
підпрограма використовує параметричний опис відрізків
відрізки P_1P_2 і W_1W_2 *вважаються двовимірними*
матричне рівняння з невідомими значеннями параметрів отримується шляхом прирівнювання компонент x і y двох параметричних описів відрізків
Коеф - *матриця* 2×2*, що містить значення коефіцієнтів рівняння відрізка*
Параметр - *матриця* 2×1, що *містить значення параметрів опису відрізка*
Параметр(1,1) - *значення параметра опису P*
Прав - *матриця* 2×1*, що містить значення правих частин рівнянь*
Зворот - *функція, що звертає матрицю*
Множ - *функція множення матриць*
сформувати матрицю Коеф
Коеф(1,1)=P_{2x}-P_{1x}
Коеф(1,2)=W_{1x}-W_{2x}
Коеф(2,1)=P_{2y}-P_{1y}
Коеф(2,2)=W_{1y}-W_{2y}
сформувати матрицю правих частин
Прав(1,1)=W_{1x}-P_{1x}
Прав(2,1)=W_{1y}-P_{1y}
звернути матрицю коефіцієнтів
нема необхідності перевіряти тут невирожденість цієї матриці, оскільки факт перетинання уже встановлений
Коеф=Зворот(Коеф)
обчислити значення параметрів у точці перетинання
Параметр=(Коеф) Множ (Прав)
обчислити координати точки перетинання
Тсічення=P_1+(P_2-P_1)*Параметр(1,1)
Return
підпрограма формування результуючого багатокутника
Subroutine Вихід(вершина, N_Q, Q)
вершина містить точку, що заноситься в результат
збільшити число вершин результату і додати точку в Q
N_Q=N_Q+1
Q(N_Q)=Вершина
Return

Лекція 10

10.1. Видалення невидимих ліній і поверхонь

Задача видалення невидимих ліній і поверхонь є однією з найскладніших у машинній графіці. Алгоритми видалення невидимих ліній і поверхонь служать для визначення ліній ребер чи поверхонь об'ємів, що видимі чи невидимі для спостерігача, який знаходиться в заданій точці простору. Необхідність видалення невидимих ліній, ребер, чи поверхонь об'ємів проілюстрована на рис.10.1.

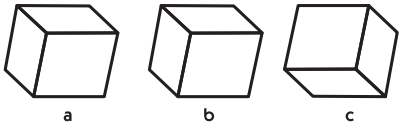


Рис.10.1. Необхідність видалення невидимих ліній

На рис.10.1(a) наведене типове каркасне креслення куба. Каркасне креслення представляє тривимірний об'єкт у вигляді штрихового зображення його ребер. Рис.10.1(a) можна інтерпретувати подвійно: як вигляд куба зверху зліва чи знизу справа. Для цього досить примружитися і перефокусувати очі. Видалення тих ліній чи поверхонь, що невидимі з відповідної точки спостереження, дозволяє позбутися неоднозначності. Результати показані на рис.10.1(b) і 10.1(c).

Складність задачі видалення невидимих ліній і поверхонь призвела до появи великого числа різних способів її вирішення. Більшість з них орієнтовані на спеціалізовані додатки. Найкращого розв'язку загальної задачі видалення невидимих ліній і поверхонь не існує. Для моделювання процесів у реальному часі, наприклад, для авіатренажерів, вимагаються швидкі алгоритми, що можуть породжувати результати з частотою відеогенерації (30 кадр/с). Для машинної мультиплікації, наприклад, вимагаються алгоритми, що можуть генерувати складні реалістичні зображення, у яких представлені тіні, прозорість і фактура, що враховують ефекти відображення і заломлення кольору в дрібних відтінках. Подібні алгоритми працюють повільно, і часто на обчислення необхідно кілька хвилин чи навіть годин. Строго кажучи, облік ефектів прозорості, фактури, відображення і т.п. не входить до завдання видалення невидимих ліній чи поверхонь. Природньо вважати їх частиною процесу візуалізації зображення. Процес візуалізації є інтерпретацією або представленням зображення чи сцени в реалістичній манері. Однак багато з цих ефектів вбудовані в алгоритми видалення невидимих поверхонь і тому будуть розглянуті в даному розділі. Існує тісний взаємозв'язок між швидкістю роботи алгоритму і детальністю його результату. Жоден з алгоритмів не може досягти гарних оцінок для цих двох показників одночасно. В міру створення більш швидких алгоритмів можна будувати більш детальні зображення. Реальні задачі, однак, завжди вимагатимуть врахування ще більшої кількості деталей.

Всі алгоритми видалення невидимих ліній (поверхонь) містять у собі сортування [1]. Порядок, у якому виконується сортування координат об'єктів, взагалі кажучи, не впливає на ефективність цих алгоритмів.

Головне сортування ведеться за геометричною відстанню від точки тіла, поверхні, чи ребра до точки спостереження. Головна ідея, покладена в основу сортування за відстанню, полягає в тому, що чим дальше розташовано об’єкт від точки спостереження, тим більша імовірність, що він буде цілком чи частково закритий одним з об’єктів, ближчих до точки спостереження. Після визначення відстаней чи пріоритетів по глибині залишається провести сортування по горизонталі і по вертикалі, щоб з’ясувати, чи буде розглянутий об’єкт дійсно закритий об’єктом, розташованим ближче до точки спостереження. Ефективність будь-якого алгоритму видалення невидимих ліній чи поверхонь в значній мірі залежить від ефективності процесу сортування. Для підвищення ефективності сортування використовується також когерентність сцени, тобто тенденція незмінюваності характеристик сцени в малому. У растровій графіці використання когерентності для покращення результатів сортування в алгоритмах видалення невидимих поверхонь приводить до алгоритмів, що дуже нагадують алгоритми растрової розгортки, які обговорювалися раніше в розділі 2.

Алгоритми видалення невидимих ліній чи поверхонь можна класифікувати за способом вибору системи координат чи простору, у якому вони працюють [1]. Алгоритми, що працюють в об’єктному просторі, мають справу з фізичною системою координат, у якій описані ці об’єкти. При цьому одержуються дуже точні результати, обмежені, взагалі кажучи, лише точністю обчислень. Отримані зображення можна легко збільшувати в багато разів. Алгоритми, що працюють в об'єктному просторі, особливо корисні в тих додатках, де необхідна висока точність. Алгоритми, що працюють у просторі зображення, мають справу з системою координат того екрану, на якому об’єкти візуалізуються. При цьому точність обчислень обмежена роздільчою здатністю екрану. Звичайна роздільча здатність екрану буває досить низькою, типовий приклад - 512×512 точок. Результати, отримані в просторі зображення, а потім збільшені в багато раз, не будуть відповідати вихідній сцені. Наприклад, можуть не збігтися кінці відрізків. Алгоритми, що формують список пріоритетів, працюють позмінно в обох згаданих системах координат.

Обсяг обчислень для будь-якого алгоритму, що працює в об’єктному просторі, і порівнює кожен об’єкт сцени із всіма іншими об’єктами цієї сцени, росте теоретично як квадрат числа об’єктів n^2. Аналогічно, обсяг обчислень будь-якого алгоритму, що працює в просторі зображення і порівнює кожен об’єкт сцени з позиціями пікселів у системі координат екрану, росте теоретично, як nN. Тут n позначає кількість об’єктів (тіл, площин чи ребер) у сцені, а N - число пікселів. Теоретично трудомісткість алгоритмів, що працюють в об'єктному просторі, менше трудомісткості алгоритмів, що працюють у просторі зображення, при $n<N$. Оскільки N звичайно дорівнює $(512)^2$, то теоретично більшість алгоритмів варто реалізовувати в об’єктному просторі. Однак, на практиці алгоритми, які працюють у просторі зображення, більш ефективні тому, що для них легше скористатися перевагою когерентності при растровій реалізації.

У наступних розділах дається докладний виклад деяких алгоритмів, що працюють як в об'єктному просторі, так і в просторі зображення. Кожен з них ілюструє одну чи декілька основних ідей теорії алгоритмів видалення невидимих ліній і поверхонь.

10.2. Алгоритм плаваючого горизонту

Алгоритм плаваючого горизонту найчастіше використовується для видалення невидимих ліній тривимірного представлення функцій, що описують поверхню у вигляді:

$$F(x, y, z)=0.$$

Подібні функції виникають в багатьох додатках у математиці, техніці, природничих науках та інших дисциплінах.

Запропоновано багато алгоритмів, що використовують цей підхід (див. [2]-[6]). Оскільки в додатках в основному цікавляться описом поверхні, цей алгоритм звичайно працює в просторі зображення. Головна ідея даного методу полягає в зведенні тривимірної задачі до двовимірної шляхом перетинання вихідної поверхні послідовністю паралельних січних площин, які мають однакові значення координат *x, y* чи *z*. На рис.10.2 наведений приклад, де зазначені паралельні площини визначаються постійними значеннями *z*.

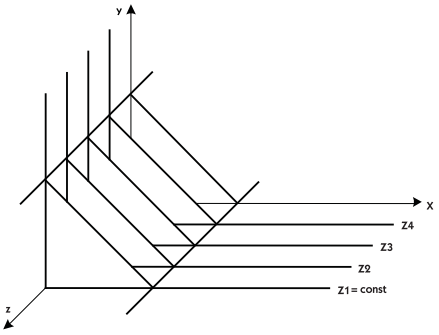


Рис.10.2. Паралельні січні площини

Функція $F(x,y,z)=0$ зводиться до послідовності кривих, що лежать у кожній з цих паралельних площин, наприклад до послідовності

$$y=f(x, z) \text{ чи } x=g(y, z),$$

де *z* постійне на кожній із заданих паралельних площин.

Отже, поверхня тепер складається з послідовності кривих, що лежать у кожній з цих площин, як показано на рис.10.3.

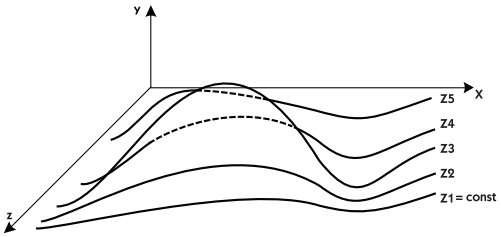


Рис.10.3. Криві в паралельних січних площинах

Тут передбачається, що отримані криві є однозначними функціями незалежних змінних. Якщо спроектувати отримані криві на площину z=0, як показано на рис.10.4, то відразу стає зрозумілою ідея алгоритму видалення невидимих ділянок вихідної поверхні.

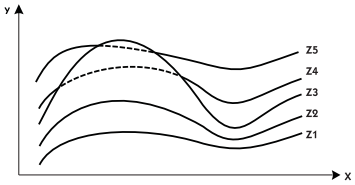


Рис.10.4. Проекції кривих на площину Z=0

Алгоритм спочатку впорядковує площини $z=const$ за зростанням відстані до них від точки спостереження. Потім для кожної площини, починаючи з найближчої до точки спостереження, будується крива, що лежить на ній, тобто для кожного значення координати x у просторі зображення визначається відповідне значення y. Алгоритм видалення невидимої лінії полягає в наступному:

Якщо на біжучій площині при деякому заданому значенні x відповідне значення y на кривій більше від значень y для всіх попередніх кривих для цього значення x, то біжуча крива видима в цій точці; інакше вона невидима.

Невидимі ділянки показані пунктиром на рис.10.4. Реалізація даного алгоритму досить проста. Для збереження максимальних значень y при кожному значенні x використовується масив, довжина якого дорівнює числу видимих точок (роздільчій здатності) по осі x у просторі зображення. Значення, що зберігаються в цьому масиві, являють собою біжучі значення "горизонту". Тому в міру рисування кожної наступної кривої цей горизонт "спливає". Фактично цей алгоритм видалення невидимих ліній працює щораз з однією лінією.

Алгоритм працює дуже добре доти, поки яка-небудь біжуча крива не виявиться нижче найпершої з кривих, як показано на рис.10.5(а).

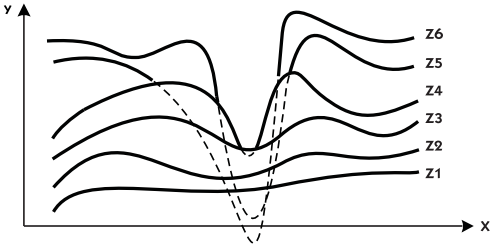


Рис.10.5. Обробка нижньої сторони поверхні

Подібні криві, зрозуміло, видимі і представляють собою нижню сторону вихідної поверхні, однак алгоритм буде вважати їх невидимими. Нижня сторона поверхні стає видимою, якщо модифікувати цей алгоритм, включивши в нього нижній горизонт, що опускається вниз по ходу роботи алгоритму. Це реалізується за допомогою другого масиву, довжина якого

дорівнює числу видимих точок вздовж осі *x* у просторі зображення. Цей масив містить найменші значення *y* для кожного значення *x*. Алгоритм тепер стає таким:

Якщо на біжучій площині при деякому заданому значенні *x* відповідне значення *y* на кривій більше максимуму чи менше мінімуму по *y* для всіх попередніх кривих при цьому *x*, то біжуча крива видима. У протилежному випадку вона невидима.

У викладеному алгоритмі передбачається, що значення функції, тобто *y*, відоме для кожного значення *x* у просторі зображення. Однак якщо для кожного значення *x* не можна вказати (обчислити) відповідне йому значення *y*, то неможливо підтримувати масиви верхнього і нижнього плаваючих горизонтів. Тоді використовується лінійна інтерполяція значень *y* між відомими значеннями для того, щоб заповнити масиви верхнього і нижнього плаваючих горизонтів, як показано на рис.10.6.

Рис.10.6. Лінійна інтерполяція між заданими точками

Якщо видимість кривої міняється, то метод з такою простою інтерполяцією не дасть коректного результату. Цей ефект проілюстрований рис.10.7(а).

Припускаючи, що операція по заповненню масивів проводиться після перевірки видимості, одержуємо, що при переході біжучої кривої від видимого до невидимого стану (сегмент AB на рис.10.7(а)), точка (X_{N+K}, Y_{N+K}) є невидимою. Тоді ділянка кривої між точками (X_N, Y_N) і (X_{N+K}, Y_{N+K}) не зображується й операція по заповненню масивів не проводиться. Утвориться зазор між біжучою і попередньою кривими. Якщо на ділянці біжучої кривої відбувається перехід від невидимого стану до видимого (сегмент CD на рис.10.7(а)), то точка (X_{M+K}, Y_{M+K}) є видимою, а ділянка кривої між точками (X_M, Y_M) і (X_{M+K}, Y_{M+K}) зображується і операція по заповненню масивів проводиться. Тому зображується і невидимий шматок сегмента CD. Крім того, масиви плаваючих горизонтів не будуть містити точних значень *y*. А це може викликати додаткові небажані ефекти для наступних кривих.

Отже, необхідно вирішувати задачу про пошук точок перетинання сегментів біжучої і попередньої кривих.

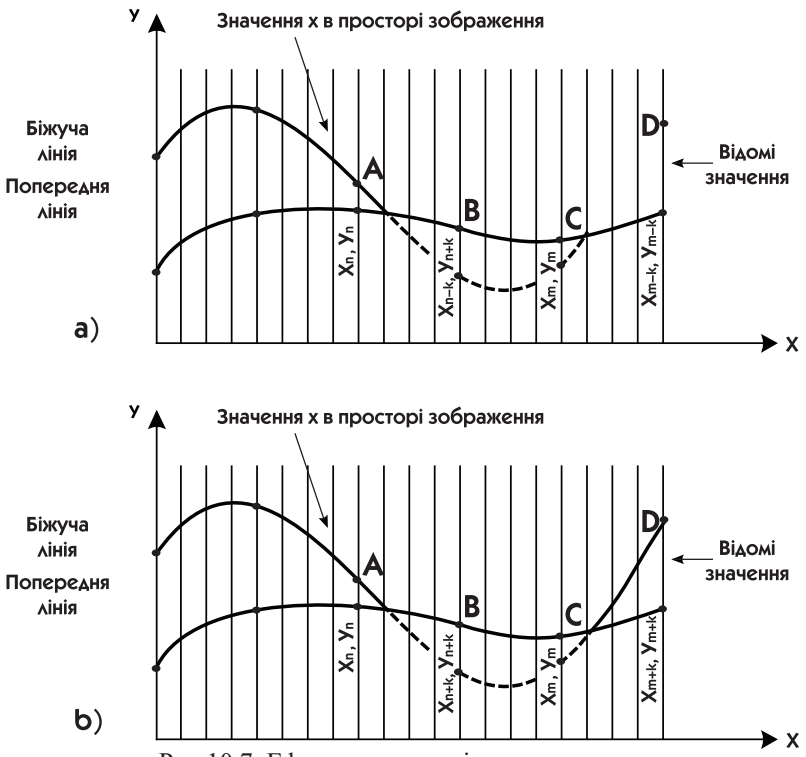


Рис.10.7. Ефект кривих, які перетинаються

Існує декілька методів одержання точок перетину кривих. На растрових дисплеях значення координати x можна збільшувати на 1, починаючи з x_n чи x_m (рис.10.7(a)). Значення y, що відповідає біжучому значенню координати x у просторі зображення, одержується шляхом додавання до значення y, що відповідає попередньому значенню координати x, вертикального збільшення Dy вздовж заданої кривої. Потім визначається видимість нової точки з координатами (x+1, y+Dy). Якщо ця точка видима, то активується зв'язаний з нею піксел. Якщо невидима, то піксел не активується, а x збільшується на 1. Цей процес продовжується доти, поки не зустрінеться x_{n+k} чи x_{m+k}. Для растрових дисплеїв перетинання, визначені за даним методом, є достатньо точними.

Близький і навіть більш елегантний метод визначення перетинань заснований на двійковому пошуку [6]. Точне значення точки перетинання двох прямолінійних відрізків, що інтерполюють біжучу і попередню криві, між точками (x_n, y_n) і (x_{n+k}, y_{n+k}) (рис.10.7) задається формулами:

$$x=x_n-Dx(y_{np}-y_{nc})/(Dy_p-Dy_c);$$

$$y=m(x-x_n)-y_n,$$

де $Dx=x_{n+k}-x_n$;

$Dy_p=(y_{n+k})_p-(y_n)_p$;

$Dy_c=(y_{n+k})_c-(y_n)_c$;

$m=(y_{n+k}-y_n)/Dx$,

а індекси c і p відповідають біжучій (current) і попередній (previous) кривим.

Отриманий результат показаний на рис.10.7(b). Тепер алгоритм викладемо більш формально.

Якщо на біжучій площині при деякому заданому значенні x відповідне значення y на кривій більше максимуму чи менше мінімуму по y для всіх попередніх кривих при цьому x, то біжуча крива видима. У протилежному випадку вона невидима. Якщо на ділянці від попереднього (x_n) до біжучого (x_{n+k}) значення x видимість кривої змінюється, то обчислюється точка перетинання (x_i). Якщо на ділянці від x_n до x_{n+k} сегмент кривої цілком видимий, то він зображується цілком; якщо він став невидимим, то зображується фрагмент від x_n до x_i; якщо ж він став видимим, то зображується фрагмент від x_i до x_{n+k}. Заповнити масиви верхнього і нижнього плаваючих горизонтів.

Викладений алгоритм приводить до деяких дефектів, коли крива, що лежить в одній з дальших від точки спостереження площин, з'являється ліворуч чи праворуч з-під багатьох кривих, що лежать у ближчих площинах до зазначеної точки спостереження. Цей ефект продемонстрований на рис.10.8, де вже оброблені площини n-1 і n розташовані ближче до точки спостереження.

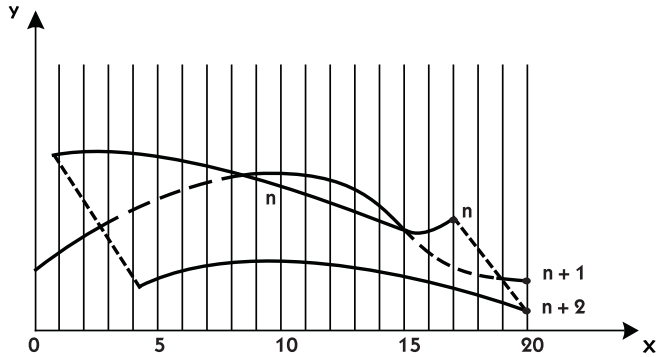


Рис.10.8. Ефект зазубреного ребра

На рисунку показано, що одржується при обробці площини n+1. Після обробки кривих n-1 і n верхній горизонт для значень x=0 і 1, дорівнює початковому значенню y; для значень x від 2 до 17 він дорівнює ординатам кривої n; а для значень 18, 19, 20 - ординатам кривої n-1. Нижній горизонт для значень x=0 і 1 дорівнює початковому значенню y; для значень x=2, 3, 4 - ординатам кривої n; а для значень x від 5 до 20 - ординатам кривої n-1. При обробці біжучої кривої (n+1) алгоритм визначає її видимою при x=4. Це показано суцільною лінією на рис.10.8. Аналогічний ефект виникає і праворуч при x=18. Такий ефект приводить до появи зазубрених бічних ребер.

Проблема з зазубреністю бічних ребер вирішується включенням у масиви верхнього і нижнього горизонтів ординат, що позначаються штриховими лініями на рис.10.8. Це можна виконати ефективно, створивши хибні бічні ребра. Наведемо алгоритм, що реалізує цю ідею для обох ребер.

Обробка лівого бічного ребра:

Якщо P_n є першою точкою на першій кривій, то збережемо P_n у якості P_{n-1} і закінчимо заповнення. У протилежному випадку створимо ребро, що з'єднує P_{n-1} і P_n.

Занесемо в масиви верхнього і нижнього горизонтів ординати цього ребра і збережемо P_n у якості P_{n-1}.

Обробка правого бічного ребра:

Якщо P_n є останньою точкою на першій кривій, то збережемо P_n у якості P_{n-1} і закінчимо заповнення. У протилежному випадку створимо ребро, що з'єднує P_n і P_{n-1}.

Занесемо в масиви верхнього і нижнього горизонтів ординати цього ребра і збережемо P_n у якості P_{n-1}.

Тепер повний алгоритм виглядає так:

Для кожної площини z=const.

Обробити ліве бічне ребро.

Для кожної точки, що лежить на кривій з біжучої площини:

Якщо при деякому заданому значенні x відповідне значення y на кривій більше максимуму чи менше мінімуму по y для всіх попередніх кривих при цьому x, то крива видима (у цій точці). У протилежному випадку вона невидима.

Якщо на сегменті від попереднього (x_n) до біжучого (x_{n+k}) значення x видимість кривої змінюється, то обчислюється перетинання (x_i).

Якщо на ділянці від x_n до x_{n+k} сегмент кривої цілком видимий, то він зображується цілком; якщо він став невидимим, то зображується його шматок від x_n до x_i; якщо ж він став видимим, то зображується його шматок від x_i до x_{n+k}.

Заповнити масиви верхнього і нижнього плаваючих горизонтів.

Обробити праве бічне ребро.

Якщо функція містить дуже гострі ділянки (піки), то наведений алгоритм може дати некоректні результати. Цей ефект показаний на рис.10.9.

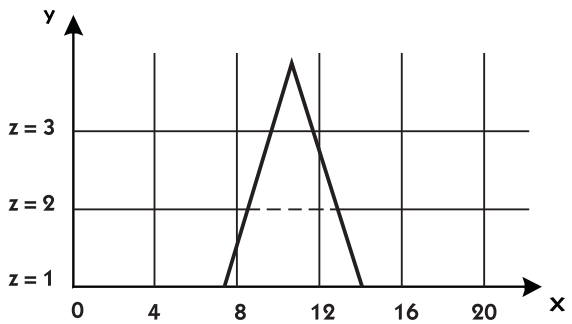


Рис.10.9. Область з дуже гострим кутом

В даній ситуації найнижча лінія (z=1) містить пік. При x=8 наступна лінія (z=2) стає видимою. При x=12 ця лінія (z=2) стає невидимою, визначається точка перетинання і лінія (z=2) зображується від x=8 до цієї точки. На ділянці від x=12 до x=16 ця лінія (z=2) знову стає видимою,

визначається нова точка перетинання і крива зображується від цього перетинання до x=16. Наступна лінія (z=3) при x=8 видима; однак вона є видимою і при x=12. Отже, ця лінія зображується на ділянці від x=8 до x=12, незважаючи на те що вона закрита піком. Цей ефект викликаний обчисленням значень функції й оцінкою її видимості на ділянках, менших за роздільчу здатність екрана, тобто тим, що функція задана занадто малою кількістю точок. Якщо зустрічаються вузькі ділянки, то функцію варто обчислювати для більшого числа точок. Якщо в прикладі на рис.10.9 функцію обчислювати в точках з абсцисами 0, 2, 4, ..., 18, 20, замість точок 0, 4, ..., 16, 20, то лінія z = 3 буде зображена правильно. На рис.10.10 показаний типовий результат роботи алгоритму плаваючого горизонту.

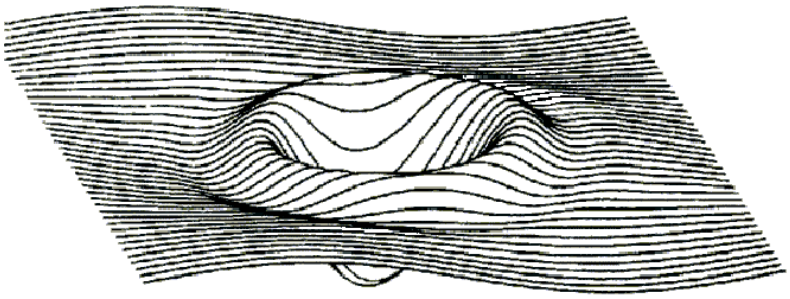


Рис.10.10.Функція $y = \frac{\sin x \cos z}{5} - \frac{3}{2}\cos\left(\frac{7a}{4}\right)\exp(-a), \quad a = (x-\pi)^2 + (z-\pi)^2$,

зображена на інтервалі $(0,2\pi)$ за алгоритмом плаваючого горизонту

Запис цього алгоритму на псевдокоді приводиться нижче.

Алгоритм плаваючого горизонту

$Г_{екран}$ - *роздільча здатність екрана в горизонтальному напрямку*
$В_{екран}$ - *роздільча здатність екрана у вертикальному напрямку*
Верх - *масив, що містить ординати верхнього горизонту*
Низ - *масив, що містить ординати нижнього горизонту*
Y - *біжуче значення функції y=f(x, z) при z=const*
Tflag - *прапор видимості для біжучої точки*
Пflag - *прапор видимості для попередньої точки, рівний* 0=невидима, 1=видима і вище верхнього горизонту, -1=видима і нижче нижнього горизонту
Draw - *графічна команда, що креслить видиму лінію між точками, заданими їх координатами*
Xmin, Xmax - *мінімальна і максимальна абсциси функції*
Xкрок - *збільшення вздовж осі x*
Zmin, Zmax - *мінімальна і максимальна аплікати функції*
Zкрок - *крок між площинами z=const*
Dimension Верх($Г_{екран}$), Низ($Г_{екран}$)
ініціалізація змінних
$X_{ліве}$=-1; $Y_{ліве}$=-1; $X_{праве}$=-1; $Y_{праве}$=-1

ініціалізація масивів горизонтів
Верх=0 Низ=$B_{екран}$
Обчислення функції на кожній площині z=const, починаючи з найближчої до спостерігача площини Zmax
for z=Zmax to Zmin step -Zкрок
  ініціалізація попередніх значень по x і y:
  Хпоперед і Ypoперед
  Хпоперед=Xmin
  Ypoперед=f(Xmin, z)
  якщо використовується перетворення за видом, то його потрібно застосувати до Хпоперед, Ypoперед, z у даній точці обробки лівого бічного ребра
  call Горребра(Хпоперед, Ypoперед, $X_{ліве}$, $Y_{ліве}$; Верх, Низ)
  call Видимість(Хпоперед, Ypoперед, Верх, Низ; Пflag)
  для кожної точки на кривій, що лежить у площині z=const
  for x=Xmin to Xmax step Xкрок
    y=f(x, z)
    якщо використовується перетворення за видом, то його потрібно застосувати до даної точки перевірки видимості біжучої точки і заповнення відповідного масиву горизонту
    call Видимість(x, y, Верх, Низ; Tflag)
    if Tflag=Пflag then
      if Tflag=1 чи Tflag=-1 **then**
        Draw(Хпоперед, Ypoперед, x, y)
        call Горизонт(Хпоперед, Ypoперед, x, y; Верх, Низ)
      else
      end if
    якщо видимість змінилася, то обчислюється перетинання і заповнюється масив горизонту
    else if Tflag=0 **then**
        if Пflag=1 **then** call Перетинання(Хпоперед, Ypoперед, x, y, Верх; Xi, Yi)
        else call Перетинання(Хпоперед, Ypoперед, x, y, Низ; Xi, Yi)
      end if
      Draw(Хпоперед, Ypoперед, Xi, Yi)
      call Горизонт(Хпоперед, Ypoперед, Xi, Yi; Верх, Низ)
      else if Tflag=1 **then**
          if Пflag=0 **then**
            call Перетинання(Хпоперед, Ypoперед, x, y, Верх; Xi, Yi)
            Draw(Xi, Yi, x, y)
            call Горизонт(Xi, Yi, x, y; Верх, Низ)
          else call Перетинання(Хпоперед, Ypoперед, x, y, Низ; Xi, Yi)
            Draw(Хпоперед, Ypoперед, Xi, Yi)
            call Горизонт(Хпоперед, Ypoперед, Xi, Yi; Верх, Низ)

```
                call Перетинання(Хпоперед, Ypoперед, x, y, Верх; Xi,
                        Yi)
                Draw(Xi, Yi, x, y)
                call Горизонт(Xi, Yi, x, y; Верх, Низ)
            end if
        else if Пflag=0 then
                call Перетинання(Хпоперед, Ypoперед, x, y, Верх;
                        Xi, Yi)
                Draw(Xi, Yi, x, y)
                call Горизонт(Xi, Yi, x, y; Верх, Низ)
            else call Перетинання(Хпоперед, Ypoперед, x, y. Верх;
                        Xi, Yi)
                Draw(Хпоперед, Ypoперед, Xi, Yi)
                call Горизонт(Хпоперед, Ypoперед, Xi, Yi;Верх, Низ)
                call Перетинання(Хпоперед, Ypoперед, x, y, Низ; Xi,
                Yi)
                Draw(Xi, Yi, x, y)
                call Горизонт(Xi, Yi, x, y; Верх, Низ)
            end if
        end if
    end if
  end if
  Знову ініціалізувати Пflag, Хпоперед, Ypoперед
  Пflag=Tflag
  Хпоперед=x
  Ypoперед=y
next x
Обробка правого кінця ребра
call Обрребра(x, y, Хправ, Управ; Верх, Низ)
next z
finish
підпрограма обробки бічного ребра
subroutine Обрребра(x, y, Хребра, Yребра; Верх, Низ)
  якщо Хребра=-1, то зустрічається перша крива і ребро не створюється
  if Хребра=-1 then 1
  call Горизонт(Хребра, Yребра, x, y; Верх, Низ)
1: Хребра=x Yребра=y
return
підпрограма визначення видимості точки
subroutine Видимість(x, y. Верх, Низ; Тфлаг)
видимість точки визначається відносно верхнього і нижнього горизонтів.
Якщо точка лежить на самому горизонті, то вона вважається видимою.
Тфлаг=0, якщо точка невидима
Тфлаг =1, якщо вона видима і вище верхнього горизонту
Тфлаг =-1, якщо вона видима і нижче нижньогогоризонту x вважається
цілим
```

```
if y<Верх(x) and y>Низ(x) then Тфлаг=0
if y>=Верх(x) then Тфлаг=1
if y<=Низ(x) then Тфлаг=-1
return
підпрограма заповнення масивів плаваючих горизонтів
subroutine Горизонт(XI, YI, X2, Y2; Верх, Низ)
Ця підпрограма використовує лінійну інтерполяцію для заповнення масивів
горизонтів між X1 і X2
Max(a, b) визначає більше з a і b
Min(a, b) визначає менше з a і b
перевірка вертикальності нахилу
if (X2-X1)=0 then
   Верх(X2)=Max(Верх(X2), Y2)
   Низ(X2)=Min(Низ(X2), Y2)
else Нахил=(Y2-Y1)/(X2-X1)
      for x=X1 to X2 step 1
            y=Нахил*(x-X1)+Y1
            Верх(x)=Max(Верх(x), y)
            Низ(x)=Min(Низ(x), y)
      next x
end if
return
підпрограма обчислення перетинання біжучої кривої з горизонтом
subroutine Перетинання(X1, Y1, X2, Y2), Масив(Xi, Yi)
Ця процедура обчислює перетин двох відрізків прямих. Масив містить
інформацію про відповідний горизонт
sign - функція, що приймає значення -1, 0, 1,
         якщо її аргумент <0, =0, >0
перевірка нескінченності нахилу
if (X2-X1)=0 then
   Xi=X2
   Yi=Масив(X2)
else обчислення перетинання
      обхід починається із найлівішої точки, перетинання вважається
      виявленим, коли змінюється знак різниці значень y
      Нахил=(Y2-Y1)/(X2-X1)
      Ysign=Sign(Y1+Нахил-Масив(X1+1))
      Csign=Ysign
      Yi=Y1+Нахил
      Xi=X1+1
      while (Csign=Ysign)
            Yi=Yi+Нахил
            Xi=Xi+1
            Csign=Sign(Yi-Масив(Xi))
      end while
      вибирається найближче ціле число
```

```
    if |Yi-Нахил-Масив(Xi-1)|<=|Yi-Масив (Xi)| then
      Yi=Yi-Нахил
      Xi=Xi-1
    end if
end if
return
```

10.4. Приклад роботи алгоритму плаваючого горизонту

Розглянемо геометричні функції, описані в табл.10.1. Ці функції задані на площинах z=0, 3, 6. У кожній площині задані дві лінії. Перша з цих ліній - пряма, а друга описує пилоподібну хвилю, точки якої лежать як вище, так і нижче площини, в якій лежить пряма.

Таблиця 10.1

№ Кривої	№ точки	x	y	z	Коментар
1	1	0	0	0	Пилоподібна хвиля
	2	1	4	0	
	3	6	-4	0	
	4	8	0	0	
2	5	0	0	0	Пряма лінія
	6	8	0	0	
3	7	0	0	3	Пилоподібна хвиля
	8	2	4	3	
	9	6	-4	3	
	10	8	0	3	
4	11	0	0	3	Пряма лінія
	12	0	0	3	
5	13	0	0	3	Пилоподібна хвиля
	14	2	4	6	
	15	6	-4	6	
	16	8	0	6	
6	17	0	0	6	Пряма лінія
	18	8	0	6	

Алгоритм плаваючого горизонту легко справляється з випадком пари ліній, що лежать в одній площині z=const. Однак порядок обробки цих ліній впливає на кінцевий результат. Тут першою буде розглядатися пряма.

Перед візуалізацією до поверхні, описаної в табл.10.1, необхідно застосувати перетворення за видом. Поверхня спочатку повертається на 30° навколо осі y, а потім на 15° навколо осі x. Результат проектується на площину z=0 з центра проекції, що знаходиться в нескінченності на осі +z.

Матриця розміром 4x4 результуючого перетворення однорідних координат має вигляд:

$$\begin{pmatrix} 0{,}866 & 0{,}129 & 0 & 0 \\ 0 & 0{,}966 & 0 & 0 \\ 0{,}5 & -0{,}224 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Виконання цього перетворення приводить до результатів, показаних у табл.10.2.

Таблиця 10.2.

№ кривої	№ точки	x	y
1	1	0	0
	2	17	41
	3	52	-31
	4	69	10
2	5	0	0
	6	69	10
3	7	15	-7
	8	32	35
	9	67	-38
	10	84	36
4	11	15	-7
	12	84	36
5	13	30	-13
	14	47	28
	15	82	-44
	16	99	-3
6	17	30	-13
	18	99	-3

Результати відображені на цілочисельній сітці в діапазонах $0 \le x \le 100$ і $0 \le y \le 50$, тобто в координатах простору зображення. Упорядковуючи криві за пріоритетом глибини вздовж осі z і пам'ятаючи, що прямі обробляються (кожна на своїй площині z=const) в першу чергу, бачимо, що порядок обробки кривих зворотний тому, що зазначений у алгоритмі плаваючого горизонту. Порядок обробки такий: 6, 5, 4, 3, 2, 1. Початкові значення верхнього і нижнього горизонтів рівні відповідно - 50 і 50 як показано в табл.10.3, що містить деякі стани екрана, утвореного горизонтами.

Таблиця 10.3

Початкові значення горизонтів	X	0	10	20	30	40	50	60	70	80	90	100
	У	-50	-50	-50	-50	-50	-50	-50	-50	-50	-50	-50
	У	50	50	50	50	50	50	50	50	50	50	50
Рис.10.11(a), крива 6	Н	-50	-50	-50	-13	-12	-10	-9	-7	-6	-4	-50
	У	50	50	50	-13	-12	-10	-9	-7	-6	-4	50
Рис.10.11(b), крива 5	У	-50	-50	-6	-13	10	22	1	-7	-6	-4	-50
	Н	50	50	50	-13	-12	-10	-9	-19	-40	-25	50
Рис.10.11(c), крива 4	У	-50	-50	-6	-4	10	22	1	1	3	1	-50
	Н	50	50	-9	-13	-12	-10	-9	-19	-40	-25	50
Рис.10.11(d), крива 3	У	-50	-50	5	29	19	22	1	1	3	1	-50
	Н	50	50	-9	-13	-12	-10	-23	-30	-40	-25	50
Рис.10.11(e), крива 2	У	0	1	5	29	19	22	9	10	5	1	-50
	Н	0	-4	-9	-13	-12	-10	-23	-30	-40	-25	50
Рис.10.11(f), крива 1	У	0	24	36	29	19	22	9	10	5	1	-50
	Н	0	-4	-9	-13	-12	-28	-23	-30	-40	-25	50

Крім того, у табл.10.3 і на рис.10.11 від (а) до (f) показані ті значення координат (заокруглені до найближчого цілого числа), що алгоритм одержав для кожної лінії. Штриховими лініями позначені фіктивні ліве і праве бічні ребра.

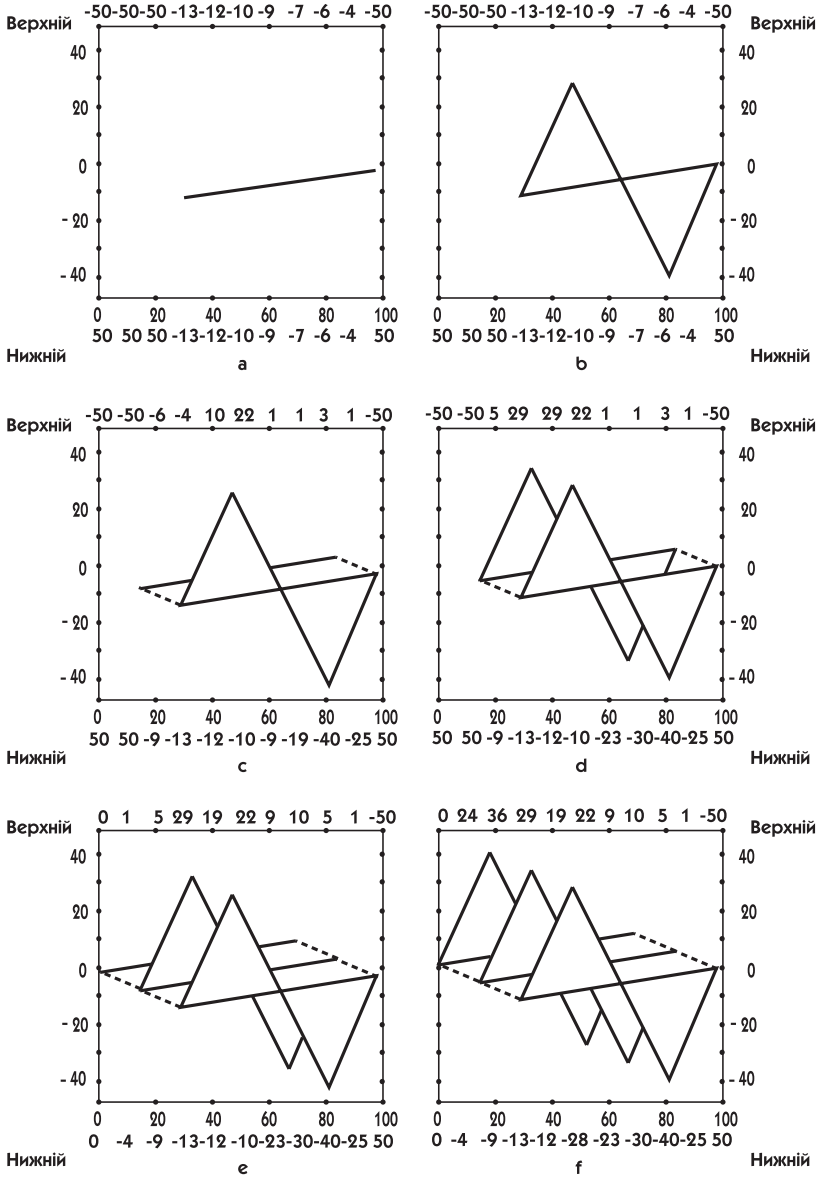


Рис.10.11. Результати прикладу 10.3.

У наведених вище алгоритмі і прикладі функція y=F(x, z) розглядалася тільки при z=const. Часто буває зручно викреслювати криві, вважаючи

постійними як z, так і x. При цьому виникає ефект перехресного штрихування. На перший погляд здається, що перехресне штрихування можна одержати шляхом накладання двох результатів, утворених площинами z=const і x=const. На рис.10.12 показано, що це не так [3].

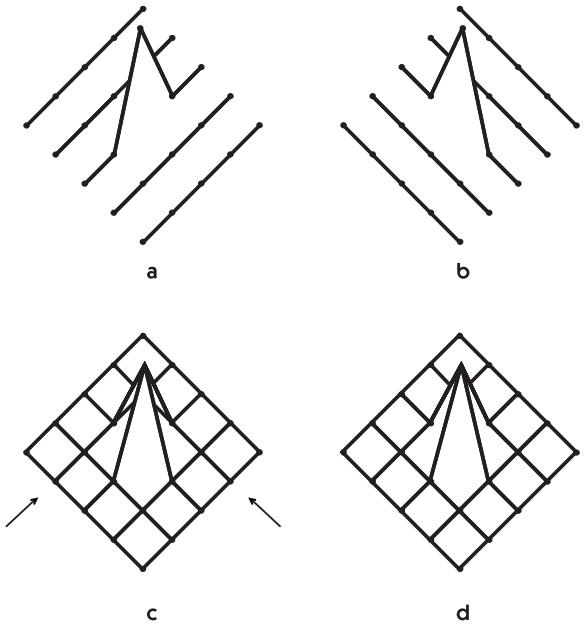


Рис.10.12. Перехресна штриховка:
лінії з постійним значенням z (a); лінії з постійним значенням x (b);
накладання a і b (c); правильний результат (d).

Зверніть увагу на рис.10.12(c), де стрілки вказують на некоректні місця. Правильний результат, показаний на рис.10.12(d), отриманий обробкою тих кривих з числа лежачих у площинах z=const чи x=const, що найближчі до горизонталей при звичайному порядку їхнього проходження. Однак після обробки кожної кривої, найближчої до горизонталі необхідно обробляти ділянки кривих, що лежать в ортогональних їй площинах, які знаходяться між зазначеною кривою і кривою, що слідує за нею. Зрозуміло, при обробці обох послідовностей кривих потрібно використовувати ті самі масиви верхнього і нижнього плаваючих горизонтів. Зокрема, якщо криві для функції y=F(x, z) при z=const найбільш близькі до горизонтальних, то після обробки кривої при z1 потрібно обробити ділянки кривих

10.4. Алгоритм, що використовує z-буфер

Це один з найпростіших алгоритмів видалення невидимих поверхонь. Вперше він був запропонований Кетмулом [8]. Працює цей алгоритм у просторі зображення. Ідея z-буфера є простим узагальненням ідеї про буфер кадру. Буфер кадру використовується для запам'ятовування атрибутів (інтенсивності) кожного піксела в просторі зображення, z-буфер - це окремий буфер глибини, що використовується для запам'ятовування

координати z чи глибини кожного видимого піксела в просторі зображення. В процесі роботи глибина чи значення z кожного нового піксела, який потрібно занести в буфер кадру, порівнюється з глибиною того піксела, що вже занесений у z-буфер. Якщо це порівняння показує, що новий піксел розташовано перед пікселом, що знаходиться в буфері кадру, то новий піксел заноситься в цей буфер і, крім того, проводиться коректування z-буфера новим значенням z. Якщо ж порівняння дає протилежний результат, то ніяких дій не проводиться. По суті, алгоритм є пошуком по x і y найбільшого значення функції z(x, y).

Головна перевага алгоритму - його простота. Крім того, цей алгоритм вирішує задачу про видалення невидимих поверхонь і робить тривіальною візуалізацію перетинань складних поверхонь. Сцени можуть бути будь-якої складності. Оскільки габарити простору зображення фіксовані, оцінка обчислювальної трудомісткості алгоритму не більш ніж лінійна. Оскільки елементи сцени чи картинки можна заносити в буфер кадру чи в z-буфер у довільному порядку, їх не потрібно попередньо сортувати за пріоритетом глибини. Тому заощаджується обчислювальний час, що затрачається на сортування за глибиною.

Основний недолік алгоритму - великий обсяг необхідної пам'яті. Якщо сцена піддається перетворенню за видом і відтинається до фіксованого діапазону координат z значень, то можна використовувати z-буфер з фіксованою точністю. Інформацію про глибину потрібно обробляти з більшою точністю, ніж координатну інформацію на площині (x, y); в основному буває досить 20 біт. Буфер кадру розміром 512×512×24 біт у комбінації з z-буфером розміром 512×512×20 біт вимагає майже 1.5 мегабайт пам'яті. Однак зниження цін на пам'ять робить економічно виправданим створення спеціалізованих запам'ятовуючих пристроїв для z-буфера і зв'язаної з ним апаратури.

Альтернативою створенню спеціальної пам'яті для z-буфера є використання для цієї мети оперативної чи загальної пам'яті. Зменшення необхідної пам'яті досягається розбивкою простору зображення на 4, 16 чи більше квадратів чи смуг. У граничному варіанті можна використовувати г-буфер розміром в один рядок розгортки. Для останнього випадку існує цікавий алгоритм порядкового сканування. Оскільки кожен елемент сцени обробляється багато разів, то сегментування z-буфера, взагалі кажучи, приводить до збільшення часу, необхідного для обробки сцени. Однак сортування на площині, що дозволяє не обробляти всі багатокутники в кожному з квадратів чи смуг, може значно скоротити цей ріст.

Інший недолік алгоритму z-буфера полягає в трудомісткості і високій вартості усунення сходового ефекту, а також реалізації ефектів прозорості і просвічування. Оскільки алгоритм заносить піксели в буфер кадру в довільному порядку, то нелегко одержати інформацію, необхідну для методів усунення сходового ефекту, що ґрунтуються на попередній фільтрації. При реалізації ефектів прозорості і просвічування, піксели можуть заноситися в буфер кадру в некоректному порядку, що веде до локальних помилок.

Хоча реалізація методів усунення сходового ефекту, які грунтуються на префільтрації, в принципі можлива [7], практично це зробити важко. Однак відносно легко реалізуються методи постфільтрації (усереднення підпікселів). Нагадаємо, що в методах усунення сходового ефекту, які ґрунтуються на постфільтрації, сцена обчислюється в такому просторі зображення, роздільча здатність якого вище, ніж роздільча здатність екрана. Тому можливі два підходи до усунення сходового ефекту на основі постфільтрації. У першому використовується буфер кадру, заданий у просторі зображення, роздільча здатність якого вища, ніж в екрана, і z-буфер, роздільча здатність якого збігається з роздільчою здатністю екрана. Глибина зображення обчислюється тільки в центрі тієї групи підпікселів, що усереднюється. Якщо для імітації відстані від спостерігача використовується масштабування інтенсивності, то цей метод може виявитися неадекватним.

В другому методі обидва буфери, задані в просторі зображення, мають підвищену роздільчу здатність. При візуалізації зображення як піксельна інформація, так і глибина усереднюються. В цьому методі вимагаються дуже великі обсяги пам'яті. Наприклад, зображення розміром 512×512×24 біт, що використовує z-буфер розміром 20 біт на піксел, роздільча здатність якого підвищена в 2 рази по осях x і y, на якому усунена ступінчастість методом рівномірного усереднення, вимагає майже 6 мегабайт пам'яті. Більш формальний опис алгоритму z-буфера такий:

Заповнити буфер кадру фоновим значенням інтенсивності чи кольору.

Заповнити z-буфер мінімальним значенням z.

Перетворити кожен багатокутник у растрову форму в довільному порядку.

Для кожного Піксел(x, y) у багатокутнику обчислити його глибину z(x,y).

Порівняти глибину z(x, y) зі значенням Zбуфер(x, y), що зберігається в z-буфері в цій же позиції.

Якщо z(x, y)>Zбуфер(x, y), то записати атрибут цього багатокутника (інтенсивність, колір і т.п.) у буфер кадру і замінити Zбуфер(x, y) на z(x, y).

У протилежному випадку ніяких дій не робити.

Як попередній крок там, де це доцільно, застосовується видалення нелицьових граней (див. алгоритм плаваючого горизонту).

Якщо відоме рівняння площини, яка містить кожен багатокутник, то обчислення глибини кожного піксела на скануючому рядку можна провести покроковим способом. Нагадаємо, що рівняння площини має вигляд

$$ax+by+cz+d=0$$

$$z=-(ax+by+d)/c, \quad c\neq 0$$

Для скануючого рядка y=const. Тому глибина піксела на цьому рядку, в якого $x_1=x+Dx$, дорівнює

$$z_1-z=-(ax_1+d)/c+(ax+d)/c=a(x-x_1)/c$$

чи

$$z_1=z-(a/c)Dx.$$

Хоча Dx=1, тому $z_1=z-(a/c)$.

10.5. Приклад, який ілюструє роботу алгоритму, що використовує z-буфер

Розглянемо прямокутник, координати кутових точок якого рівні P_1(10, 5, 10), P_2(10, 25, 10), P_3(25, 25, 10), P_4(25, 5, 10) і трикутник з вершинами P_5(15, 15, 15), P_6(25, 25, 5), P_7(30, 10, 5), що його протикає (рис.10.13(a)).

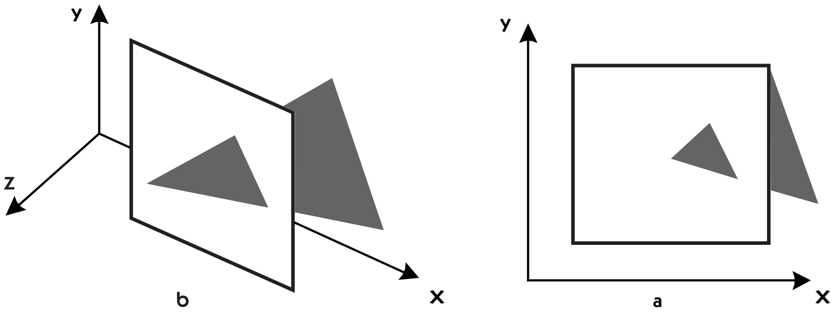


Рис.10.13 Протикаючий трикутник

Ці багатокутники потрібно зобразити на екрані з роздільчою здатністю 32×32, використовуючи простий буфер кадру з двома бітовими площинами. В буфері фон позначено через 0, прямокутник - через 1, а трикутник - через 2. Спочатку в буфері кадру і в z-буфері містяться нулі. Після растрового розгорнення прямокутника вміст буфера кадру буде мати вигляд:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1010101010101010101010101010100 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 101010101010101010101010

Під z-буфер надається 4 бітові площини розміром 32×32 біт. Таким чином, вміст z-буфера виявиться в діапазоні від 0 до 16. Точка спостереження розташована в нескінченності на додатній півосі z, як показано на рис.10.13(b).

10.6. Алгоритм, що використовує список пріоритетів

При реалізації всіх алгоритмів видалення невидимих ліній і поверхонь, що обговорювалися, встановлювалися пріоритети, тобто глибини об'єктів сцени чи їхні відстані від точки спостереження. Алгоритми, що використовують список пріоритетів, намагаються одержати перевагу за допомогою попереднього сортування за глибиною чи пріоритетом. Ціль такого сортування полягає в тому, щоб одержати остаточний список елементів сцени, впорядкованих за пріоритетом глибини, що базується на відстані від точки спостереження. Якщо такий список остаточний, то ніякі два елементи не будуть взаємно перекривати один одного. Тоді можна записати всі елементи в буфер кадру по черзі, починаючи з елемента, найбільш віддаленого від точки спостереження. Більш близькі до спостерігача елементи будуть затирати інформацію про більш далекі елементи в буфері кадру. Тому задача про видалення невидимих поверхонь розв’язується тривіально.

Ефекти прозорості можна включити до складу алгоритму шляхом не повного, а часткового коректування вмісту буфера кадру з врахуванням атрибутів прозорих елементів (див. [9] і розд.5.8).

Для простих елементів сцени, наприклад для багатокутників, цей метод іноді називають алгоритмом художника, оскільки він аналогічний способу, яким художник створює картину. Спочатку художник малює фон, потім предмети, що лежать на середній відстані, і, нарешті, передній план. Тим самим художник вирішує задачу про видалення невидимих поверхонь, чи задачу видимості, шляхом побудови картини в порядку зворотньому до пріоритету.

Для простої сцени, на зразок тієї, що показана на рис.10.14(a), остаточний список пріоритетів можна одержати безпосередньо. Наприклад, ці багатокутники можна впорядкувати по їх максимальному чи мінімальному значенню координати z. Однак для сцени, показаної на рис.10.14(b), остаточний список пріоритетів за глибиною неможливо одержати простим сортуванням по z.

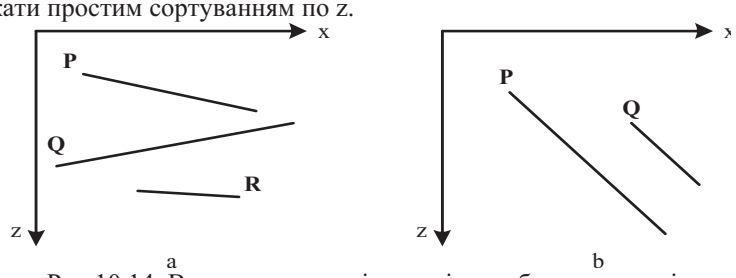


Рис.10.14. Встановлення пріоритетів для багатокутників

Якщо P і Q з рис.10.14(b) впорядковані за мінімальним значенням координати $z(z_{min})$, виявиться, що P в списку пріоритетів по глибині буде стояти перед Q. Якщо їх записати в буфер кадру в такому порядку, то вийде, що Q частково екранує P. Однак фактично P частково екранує Q. Правильний порядок у списку пріоритетів буде тоді, коли P і Q поміняються місцями.

Інші труднощі показані на рис.10.15. Тут багатокутники циклічно перекривають один одного. На рис.10.15(a) P знаходиться перед Q, який лежить перед R, що у свою чергу знаходиться перед P. На рис.10.15(b) P екранує Q, а Q екранує P. Аналогічне циклічне екранування виникає при протиканні багатокутників; наприклад, на рис.10.13 показано, як трикутник протикає прямокутник. Там прямокутник екранується трикутником і навпаки. В обох прикладах остаточний список пріоритетів неможливо встановити відразу. Вихід з положення полягає в циклічному розрізуванні багатокутників по лініях, утворених перетинаннями їхніх площин доти, поки не буде отриманий остаточний список пріоритетів. Такі лінії показані пунктиром на рис.10.15.

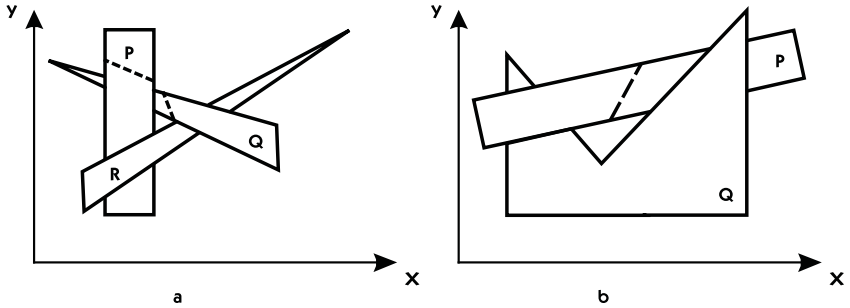


Рис.10.15. Багатокутники, які циклічно перекриваються

Ньюел М., Ньюел Р. і Санча [9] запропонували спеціальний метод сортування для розв'зування конфліктів, які виникають при створенні списку пріоритетів по глибині. Цей метод включений до складу алгоритму Ньюела - Ньюела - Санча, що викладається нижче. В алгоритмі динамічно обчислюється новий список пріоритетів перед обробкою кожного кадру сцени. Не накладається ніяких обмежень на складність сцени і на тип багатокутників, використовуваних для опису елементів сцени. Первісний алгоритм Ньюела - Ньюела - Санча був призначений для обробки тривимірних тіл. Це розширення не обмежене рамками багатогранників. Воно може, крім того, обробляти тіла змішаних типів у рамках однієї сцени.

Алгоритм Ньюела - Ньюела - Санча для випадку багатокутників.

Сформувати попередній список пріоритетів по глибині, використовуючи як ключ сортування значення z_{min} для кожного багатокутника. Першим у списку буде багатокутник з мінімальним значенням z_{min}. Цей багатокутник лежить далі усіх від точки спостереження, розташованої в нескінченності на позитивній півосі z. Позначимо його через P, а наступний у списку багатокутник - через Q.

Для кожного багатокутника P зі списку треба перевірити його відношення з Q.

Якщо найближча вершина $P(P_{zmax})$ буде далі від точки спостереження, ніж найвіддаленіша вершина $Q(Q_{zmin})$, тобто $Q_{zmin} \geq P_{zmin}$ ніяка частина P не може екранувати Q. Занести P в буфер кадру (рис.10.14a).

Якщо $Q_{zmax} < P_{zmax}$, то P потенційно екранує не тільки Q, але також і будь-який інший багатокутник типу Q зі списку, для якого $Q_{zmax} < P_{zmax}$. Тим самим утвориться множина {Q}. Однак P може фактично і не екранувати жоден з цих багатокутників. Якщо так, то P можна заносити в буфер кадру. Для відповіді на це питання використовується серія тестів, які подані по зростанню їхньої обчислювальної складності. Ці тести нижче формулюються у виді питань. Якщо відповідь на будь-яке питання буде позитивною, то P не може екранувати {Q}. Тому P відразу ж заноситься в буфер кадру. Ось ці тести:

Чи вірно, що прямокутні охоплюючі оболонки P і Q не перекриваються по x?

Чи вірно, що прямокутні оболонки P і Q не перекриваються по y?

Чи вірно, що P цілком лежить з того боку площини, яка містить Q і розташований далі від точки спостереження (рис. 10.16(a))?

Чи вірно, що Q цілком лежить по ту сторону площини, що містить P, і що ближче до точки спостереження (рис.10.16(b))?

Чи вірно, що проекції P і Q не перекриваються?

Кожний з цих тестів застосовується до кожного елемента {Q}.

Якщо жоден з них не дає позитивної відповіді і не заносить P в буфер кадру, то P може закривати Q.

Поміняти P і Q місцями, позначивши позицію Q у списку.

Повторити тести з новим списком. Це дає позитивний результат для сцени з рис.10.14(b).

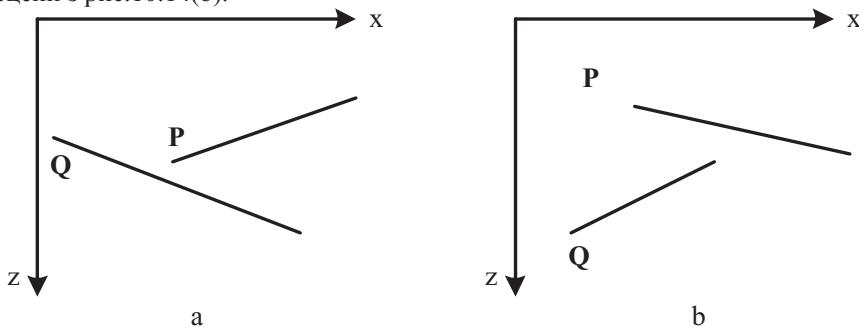


Рис.10.16. Тести для багатокутників, які перекриваються

Якщо зроблена спроба знову переставити Q, виходить, виявлена ситуація циклічного екранування (рис.10.15). У цьому випадку P розрізається площиною, що містить Q, вихідний багатокутник P видаляється зі списку, а його частини заносяться в список. Потім тести повторюються для нового списку. Цей крок запобігає зацикленню алгоритму.

Узяті разом, перші два з наведених вище питань утворюють звичайний габаритний тест для прямокутних оболонок. Оскільки багато сцен не є квадратними, то прямокутні охоплюючі оболонки будуть з більшою ймовірністю перекриватися в одному з двох можливих напрямків. Коли багатокутники переважно горизонтальні чи вертикальні, то використання одного з цих двох тестів виявляється більш ефективним. В алгоритмі в тій формі, у якій він записаний вище, передбачається, що ширина сцени більша від її висоти, тобто багатокутники переважно є горизонтальними. Якщо висота сцени більша від її ширини, то тести варто поміняти місцями. Якщо ж сцена є квадратною чи її структура ізоморфна, то порядок застосування цих тестів не має значення.

Третій і четвертий тести можна реалізувати з використанням деяких з раніше обговорюваних тестів видимості. Оскільки рівняння несучої площини чи нормаль до неї часто відомі для кожного багатокутника, то зручно застосувати простий тест підстановки. Якщо досліджується відношення багатокутника Q до багатокутника P, то координати вершин Q підставляються в рівняння площини, що містить P. Якщо всі знаки результатів підстановки збігаються, то Q цілком лежить по одну сторону від P. Тут, так само як і в інших алгоритмах видалення невидимих поверхонь, що обговорювалися раніше, якщо це потрібно, виконується попереднє видалення нелицьових граней. Більш повно ці поняття проілюструє приклад для багатокутників, що конфліктують у просторі.

10.7. Приклад роботи алгоритму, що використовує список пріоритетів

Перевіримо відношення між багатокутниками, які конфліктують у просторі. Розглянемо три трикутники P, Q1, і Q2 (рис.10.17).

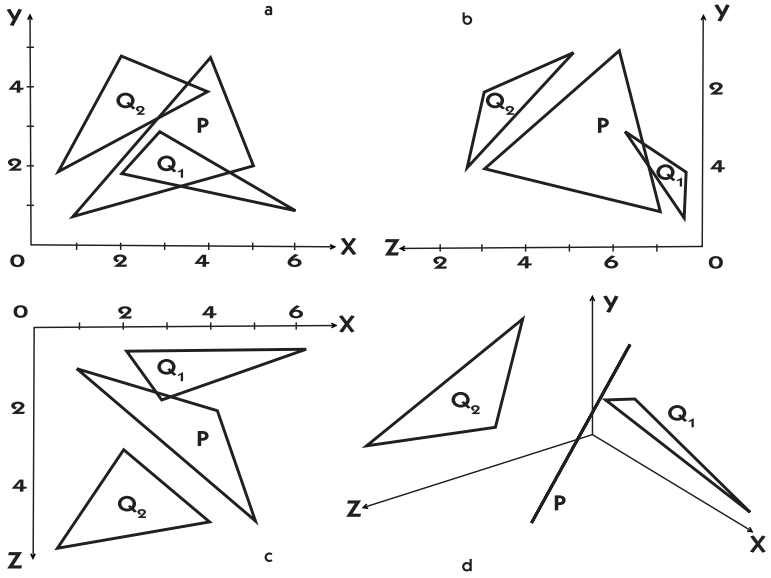


Рис.10.17. Багатокутники, які конфліктують в просторі

Вершини цих трикутників такі:

P: (1, 1, 1), (4, 5, 2), (5, 2, 5)

Q1: (2, 2, 0.5), (3, 3, 1.75), (6, 1, 0.5)

Q2: (0.5, 2, 5.5), (2, 5, 3), (4, 4, 5)

Потрібно визначити, чи вірно, що Q1 і Q2 цілком лежать по одну сторону від P. Три ортогональні проекції, показані на рис.10.17, не дають ясної відповіді. Рівняння площини трикутника P має вигляд 15x-8y-13z+6=0. Пробна функція T.F. дорівнює T.F.=15x-8y-13z+6.

Підстановка вершин Q1 у пробну функцію дає:

T.F.1=15(2)-8(2)-13(0.5)+6=13.5>0

T.F.2=15(3)-8(3)-13(1.75)+6=4.25>0

T.F.3=15(6)-8(1)-13(0.5)+6=81.5>0

Оскільки знаки всіх пробних функцій додатні, то трикутник Q1 цілком лежить по одну сторону від P.

Підставляючи вершини Q2 у пробну функцію, маємо:

T.F.4=15(0.5)-8(2)-13(5.5)+6=-74<0

T.F.5=15(2)-8(5)-13(3)+6=-43<0

T.F.6=15(4)-8(4)-13(5)+6=-31<0

Знову знаки всіх пробних функцій збіглися; тому трикутник Q2 цілком лежить по одну сторону від P.

На рис.10.17(d) добре видно, що Q1 лежить з того боку від площини трикутника P, який далі від точки спостереження, розташованої в нескінченності на позитивній півосі z. Отже, Q частково екранується P. На рис.10.17(d) також добре видно, що Q2 лежить з того боку від площини трикутника P, який ближче до зазначеної точки спостереження. Тому Q2 частково екранує P.

З приведеного прикладу випливає, що:

Якщо знаки пробної функції для всіх вершин деякого багатокутника збігаються і додатні чи дорівнюють нулю, то цей багатокутник знаходиться з дальшої (невидимої) сторони від площини P.

Якщо знаки пробної функції для всіх вершин деякого багатокутника збігаються і від'ємні чи дорівнюють нулю, то цей багатокутник розташований з ближньої (видимої) сторони від площини P.

Якщо значення пробної функції для кожної вершини багатокутника дорівнюють нулю, то цей багатокутник лежить на площині P.

Останній тест із приведеної серії особливо складний з обчислювальної точки зору, оскільки вимагає точного визначення того факту, що проекції P і Q не перетинаються. Метод рішення цієї задачі вже обговорювався раніше в рамках опису алгоритму Варнока.

Якщо має місце циклічне екранування, то для розбивки багатокутників уздовж лінії перетинання вміщуючих ці багатокутники площин можна скористатися алгоритмом відсікання багатокутників Сазерленда - Ходжмена. Тут площина, що містить Q, використовується як січна. Кожне ребро P відтинається площиною Q, при цьому формуються два нових багатокутники. Для пошуку перетинання кожного ребра P с площиною Q можна скористатися алгоритмом відсікання Кіруса - Віка.

В алгоритмі Ньюела - Ньюела - Санча робиться спроба вирішити задачу видалення невидимих поверхонь динамічно шляхом обробки всіх багатокутників сцени для кожного шуканого кадру. Якщо ж сцена складна, а частота кадрів велика, як у систем моделювання в реальному часі, то обчислювальної потужності звичайних універсальних ЕОМ може не вистачити (див. [11]). Однак у багатьох випадках моделювання в реальному часі, наприклад при імітації посадки літака, сцена статична, а міняється тільки точка спостереження. Шумейкер і ін. [12] вдало запропонували попередньо обчислювати в автономному режимі деякі більш загальні пріоритетні характеристики, такі, як список пріоритетів для моделювання згаданих вище статичних сцен.

В алгоритмі Шумейкера в сцені припустимі тільки опуклі багатокутники. Такі багатокутники групуються в кластери, що є лінійно роздільними. Кластери вважаються лінійно роздільними, якщо існує така поділяюча площина, що проходить між ними, не перетинаючи їх. Кілька двовимірних кластерів показано на рис.10.18(а). Поділяючі площини позначені буквами α і β. Вони розбивають сцену на чотири області A, B, C, D. Точка спостереження може розташовуватися в кожній з цих областей. На рис.10.18(b) показана деревовидна структура, що встановлює пріоритети кластерів у сцені.

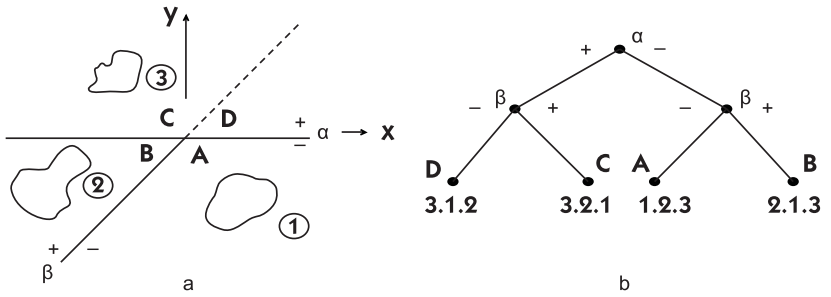


Рис.10.18. Кластерні пріоритети

Ці пріоритети можна обчислити заздалегідь для будь-якої точки спостереження в площині. Підстановка координат точки спостереження в рівняння поділяючих площин породжує відповідний вузол дерева кластерних пріоритетів. Потім для кожного кластера в зворотньому порядку пріоритетів вирішується задача видалення невидимих поверхонь.

Оскільки подібно алгоритмам Варнока і z-буфера в алгоритмах, що будують список пріоритетів, багатокутники обробляються в довільному порядку, застосування методів усунення сходового ефекту до результуючого зображення утруднено. Однак для цієї мети тут застосуємо метод постфільтрації, використовуваний також в алгоритмах Варнока і z-буфера.

Алгоритми, що будують список пріоритетів, що використовують z-буфер і відносяться до типу Варнока, можуть також застосовуватися і для видалення невидимих ліній. При їхньому використанні в цій ролі ребра кожного багатокутника заносяться в буфер кадру з одним атрибутом, при

чому внутрішня область кожного багатокутника заноситься в буфер кадру з атрибутом фону. При такому підході багатокутники, що знаходяться ближче до точки спостереження, "затуляють" ребра багатокутників, які розташовані далі від неї.

10.8. Приклад, який ілюструє роботу алгоритму, що використовує кластерні пріоритети

Припустимо, що поділяючі площини α і β, показані на рис.10.18, перетинаються в початку координат. Далі припустимо, що α - це площина y=0, а β - площина y=x; обидві площини перпендикулярні до площини креслення. Рівняння цих площин і відповідні їм пробні функції такі:

α: y=0 $(T.F.)_1$=y

β: y-x=0 $(T.F.)_2$=y-x

Точка спостереження, що лежить на прямій 2y-x=0. наприклад (20;10), дає:

$(T.F.)_1=10>0$

$(T.F.)_2=10-20=-10<0$

Виходить, ця точка спостереження лежить в області D. З рис.10.18(b) видно, що список кластерних пріоритетів такий: 3, 1, 2.

Кластери використовуються для розбивки сцени. Найпростішим кластером є єдиний багатокутник. Кластери можуть являти собою складні полігональні чи неполігональні поверхні і тіла, для кожного типу яких існують свої методи видалення невидимих ліній, як показано Ньюелом [10].

У рамках кластерів деяких типів пріоритети окремих багатокутників не залежать від положення точки спостереження [12, 13]. Ця обставина є однією з основних переваг алгоритму Шумейкера. Вона дозволяє заздалегідь обчислювати повний список пріоритетів. На рис.10.19(a) показаний двовимірний кластер, для якого можна заздалегідь обчислити індивідуальні пріоритети багатокутників. Пріоритет кожного багатокутника встановлюється в залежності від можливості екранування даним багатокутником будь-якого іншого багатокутника для довільної точки спостереження. Чим більше число багатокутників може екранувати даний багатокутник, тим вище його пріоритет. При встановленні пріоритетів багатокутників у рамках кластера для заданої точки спостереження спочатку видаляються нелицьові багатокутники. Багатокутники, що залишилися, розташовуються потім у пріоритетному порядку, як показано на рис.10.19(b) і (c).

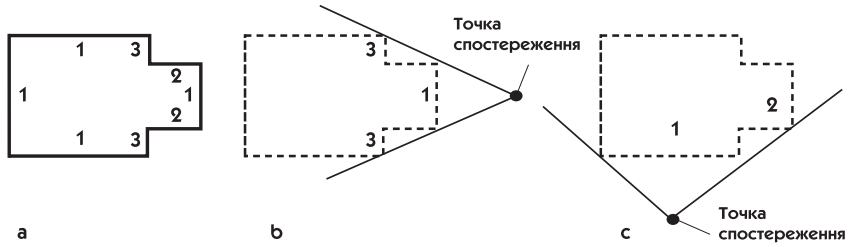


Рис.10.19. Пріоритети в рамках кластера

Алгоритми, що використовують список пріоритетів, працюють як у просторі об'єкта, так і в просторі зображення. Зокрема, формування списку пріоритетів ведеться в просторі об'єкта, а результат заноситься в буфер кадру в термінах простору зображення. Використання буфера кадру є критичним для даного алгоритму.

10.9. Алгоритм, що використовує трасування променів

Оцінка ефективності всіх алгоритмів видалення невидимих поверхонь, що обговорювалися в попередніх розділах, залежить від визначених характеристик когерентності тієї сцени, для якої ведеться пошук її видимих ділянок.

На відміну від них трасування променів є методом грубої сили (методом грубої сили прийнято називати метод, який не враховує специфіку оброблюваного об'єкта). Головна ідея, що лежить в основі цього методу, полягає в тім, що спостерігач бачить будь-який об'єкт за допомогою світла, яке випускається деяким джерелом, що падає на цей об'єкт і потім якимось шляхом доходить до спостерігача.

Світло може досягти спостерігача, відбившись від поверхні, чи переломившись, пройшовши через неї. Якщо простежити за променями світла, випущеними джерелом, то можна переконатися, що дуже багато з них не дійдуть до спостерігача. Отже, цей процес був би обчислювально неефективний.

Аппель [15] першим запропонував відслідковувати (трасувати) промені в зворотньому напрямку, тобто від спостерігача до об'єкта, як показано на рис.10.20.

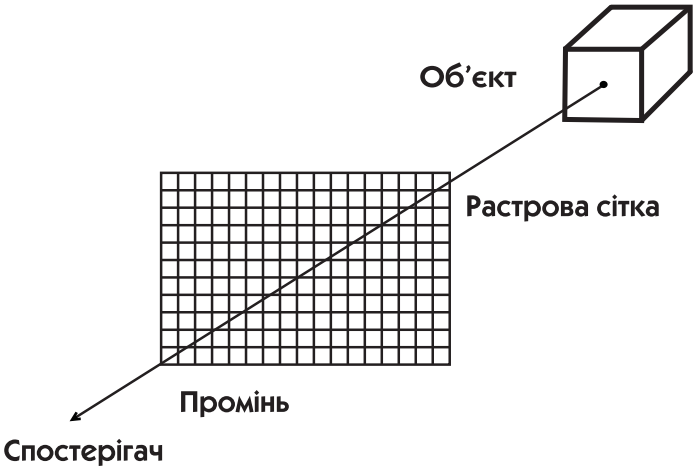


Рис.10.20. Просте трасування променя

Це метод був з успіхом реалізований у рамках дисплейної системи візуалізації твердих тіл MAGI[16]. У самій системі MAGI трасування припинялося, як тільки промінь перетинав поверхню видимого непрозорого об'єкта; тобто промінь використовувався для обробки схованих чи видимих поверхонь.

В результаті Кей [17,18] і Уіттед [16] реалізували алгоритми трасування променів з використанням загальних моделей висвітлення. Ці алгоритми враховують ефекти відображення одного об'єкта від поверхні іншого, заломлення, прозорості і затемнення. Спрацьовувало також усунення ступінчастості. Дане ж обговорення обмежується застосуванням методу трасування променів для визначення видимих чи схованих поверхонь.

Рис.10.20 служить ілюстрацією алгоритму трасування променів. У цьому алгоритмі передбачається, що сцена вже перетворена в просторі зображення. Перспективне перетворення не використовується. Вважається, що точка зору чи спостерігач знаходиться в нескінченності на додатній півосі z. Тому всі світлові промені паралельні осі z. Кожен промінь, що виходить від спостерігача, проходить через центр піксела на растрі до сцени. Траєкторія кожного променя відслідковується, щоб визначити, які саме об'єкти сцени, якщо такі існують, перетинаються з даним променем. Необхідно перевірити перетинання кожного об'єкта сцени з кожним променем. Якщо промінь перетинає об'єкт, то визначаються всі можливі точки перетинання променя й об'єкта. Можна одержати велику кількість перетинань, якщо розглядати багато об'єктів. Ці перетинання упорядковуються по глибині. Перетинання з максимальним значенням z представляє видиму поверхню для даного піксела. Атрибути цього об'єкта використовуються для визначення характеристик піксела.

Якщо точка зору знаходиться не в нескінченності, алгоритм трасування променів лише незначно ускладнюється. Тут передбачається, що спостерігач як і раніше знаходиться на додатній півосі z. Картинна площина, тобто растр, перпендикулярна до осі z, як показано на рис.10.21.

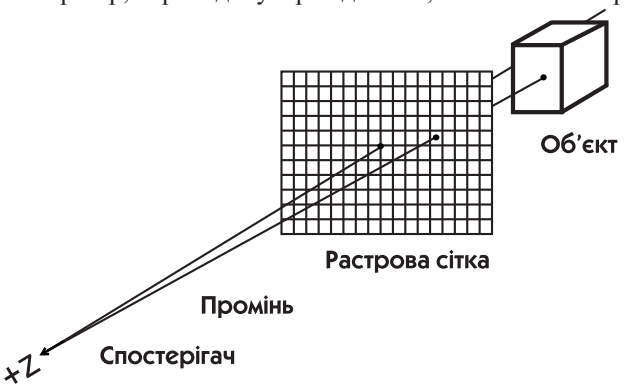


Рис.10.21. Трасування променя з врахуванням перспективи

Завдання полягає в тім, щоб побудувати одноточкову центральну проекцію на картинну площину[1-1].

Найважливішим елементом алгоритму визначення видимих поверхонь шляхом трасування променів є процедура визначення перетинань. До складу сцени можна включати будь-який об'єкт, для якого можна створити процедуру побудови перетинань. Об'єкти сцени можуть складатися з

набору плоских багатокутників, багатогранників чи тіл, обмежених квадратичними або біполіноміальними параметричними поверхнями.

Оскільки 75-96% часу, затрачуваного алгоритмом трасування променів, іде на визначення перетинань, то ефективність процедури пошуку перетинань значно впливає на продуктивність всього алгоритму. Обчислювальна вартість визначення перетинань довільної просторової прямої (чи променя) з одним виділеним об'єктом може виявлятися високою (див. наприклад [21]). Щоб позбутися від непотрібного пошуку перетинань, проводиться перевірка перетинання променя з об'ємною оболонкою розглянутого об'єкта. І, якщо промінь не перетинає оболонки, то не потрібно більше шукати перетинань цього об'єкта з променем. Як оболонку можна використовувати прямокутний паралелепіпед чи сферу.

Хоча, як показано на рис.10.22, використання сфери як оболонки може виявитися неефективним, факт перетинання тривимірного променя зі сферою визначається дуже просто.

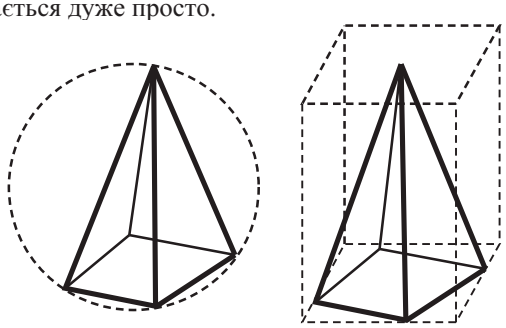


Рис.10.22. Сферична і прямокутна оболонки

Зокрема, якщо відстань від центра сферичної оболонки до променя перевершує радіус цієї сфери, то промінь не перетинає оболонки. Отже, він не може перетнутися і з об'єктом.

Тому тест зі сферичною оболонкою зводиться до визначення відстані від точки до тривимірної прямої, тобто променя. Будемо використовувати параметричне представлення прямої, що проходить через точки $P_1(x_1, y_1, z_1)$ і $P_2(x_2, y_2, z_2)$, тобто:

$P(t)=P_1+(P_2-P_1)t$ з компонентами

$x=x_1+(x_2-x_1)t=x_1+at$

$y=y_1+(y_2-y_1)t=y_1+bt$

$z=z_1+(z_2-z_1)t=z_1+ct$

Тоді мінімальна відстань d від цієї прямої до точки $P_0(x_0, y_0, z_0)$ дорівнює:

$$d^2=(x-x_0)^2+(y-y_0)^2+(z-z_0)^2,$$

де параметр t, що визначає найближчу точку P(t) дорівнює:

$$t=(a(x_1-x_0)+b(y_1-y_0)+c(z_1-z_0))/(a^2+b^2+c^2)$$

Якщо $d^2>R^2$, де R - радіус сферичної оболонки, то промінь не може перетнутися з об'єктом.

Виконання габаритного тесту з прямокутною оболонкою в тривимірному просторі вимагає більшого обсягу обчислень. При цьому

варто перевірити перетинання променя щонайменше з трьома нескінченними площинами, які обмежують прямокутну оболонку. Оскільки точки перетинання можуть виявитися поза гранями цього паралелепіпеда, то для кожної з них слід, крім того, зробити перевірку на охоплення чи влучення всередину. Отже, для трьох вимірів тест із прямокутною оболонкою виявляється повільнішим, ніж тест зі сферичною оболонкою.

Однією простою процедурою можна звести тест із прямокутною оболонкою до порівняння знаків, спрощуючи тим самим обчислення перетинань з об'єктом, а також порівняння по глибині серед точок перетинання. У цій процедурі використовуються переноси і повороти навколо координатних осей [1-1] для того, щоб домогтися збігу променя з віссю z.

Аналогічним перетворенням піддається і прямокутна оболонка об'єкта. Промінь перетинає оболонку, якщо в новій перенесеній і поверненій системі координат знаки x_{min} і x_{max}, а також y_{min} і y_{max} протилежні, як показано на рис.10.23.

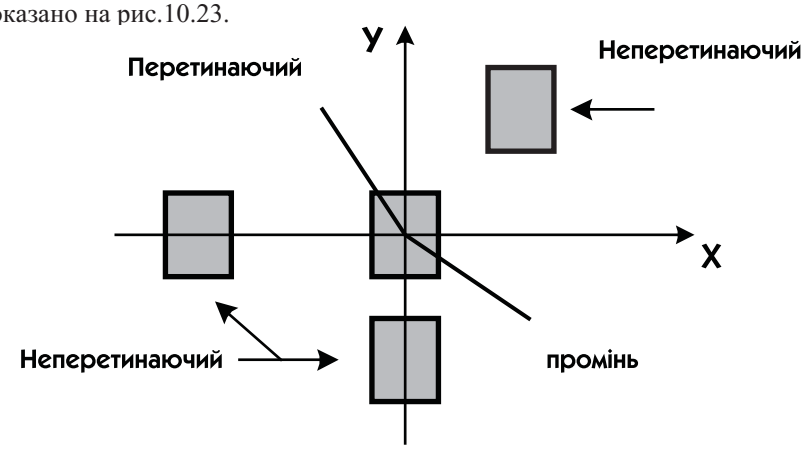


Рис.10.23. Перетинання прямокутної оболонки в перетвореній системі координат

Продемонструємо спрощення обчислень точок перетинання на прикладі поверхні другого порядку загального виду. У довільній системі координат поверхня другого порядку є геометричне місце точок, координати яких задовольняють рівнянню:

$$Q(x,y,z)=a_1x^2+a_2y^2+a_3z^2+b_1yz+b_2xz+b_3xy+c_1x+c_2y+c_3z+d=0.$$

Після застосування перетворення, що є комбінацією переносу і повороту, яке використовується для сполучення променя з віссю z, перетинання цього променя з поверхнею, якщо воно має місце, відбудеться при x=0, y=0. Тому в загальному випадку точки перетинання розв'язками рівняння:

$$a'_3z^2+c'_3z^2+d=0,$$

тобто $Z=(-c'_3\mathrm{sqrt}(c'^2_3-4a'_3d'))/(2a'_3)$,

де штрих зверху позначає коефіцієнти загального рівняння поверхні другого порядку після перетворення.

Якщо c'_3-$4a'_3d'$<0, то розв'язки виражаються комплексними числами і промінь не перетинає поверхні. Якщо нескінченна поверхня другого порядку (наприклад конус чи циліндр) обмежена площинами, то ці площини теж варто перетворити і перевірити на перетинання. Якщо знайдено перетинання з нескінченною обмежуючою площиною, то необхідно, крім того, провести перевірку на влучення усередину. Однак, у перетвореній системі координат цю перевірку можна провести на двомірній проекції фігури, утвореній перетинанням обмежуючої площини і квадратної поверхні. Для одержання точки перетинання у вихідній системі координат необхідно застосувати зворотнє перетворення.

Обчислення перетинання для елементів біполіноміальних параметричних поверхонь більш складні. Уіттед [19] запропонував простий метод розбивки для елемента бікубічної поверхні. Обчислення виконуються з елементом поверхні в його вихідному положенні. Якщо промінь перетинає сферичну оболонку елемента поверхні, то цей елемент розбивається за допомогою алгоритму розбивки Кетмула. Потім промінь перевіряється на перетинання зі сферичними оболонками піделементів. Якщо перетинань не виявлено, то промінь не перетинається і із самим елементом. Якщо ж промінь перетинається зі сферичною оболонкою якого-небудь піделемента, то останній розбивається далі. Процес завершується, якщо жодна зі сферичних оболонок не пересічена чи якщо досягнуто заздалегідь визначений їх мінімальний розмір. Ці сферичні оболонки мінімального розміру і є шуканими перетинаннями променя й елемента поверхні.

При реалізації перетворення, що з'єднує промінь з віссю z, метод розбивки можна використовувати скоріше стосовно до прямокутних оболонок, ніж до сферичних. Це скорочує число розбивок і збільшує ефективність алгоритму. Для параметричних поверхонь, що володіють властивістю опуклої оболонки, наприклад для поверхонь Безье і B-сплайнів [1-1], число розбивок можна скоротити додатково за рахунок ускладнення алгоритму, якщо для піделементів скористатися їхніми опуклими оболонками замість прямокутних.

Кадзі [21] розробив метод для біполіноміальних параметричних поверхонь, який не вимагає їхнього підрозділу. Цей метод заснований на поняттях, запозичених з алгебраїчної геометрії. Розв'язки алгебраїчних рівнянь вищих ступенів, що отримуються при цьому, знаходяться чисельно. Метод, подібний цьому, можна реалізувати в перетвореній системі координат. Нагадаємо, що біполіноміальна параметрична поверхня визначається рівнянням Q(u,w)=0 з компонентами x=f(u,w); y=g(u,w); z=g(u,w).

У перетвореній системі координат виконується умова x=y=0. Виходить, f(u,w)=0, g(u,w)=0.

Спільний розв'язок цієї пари рівнянь дає значення u і w для точок перетинання. Підстановка цих значень у рівняння z=h(u,w) дає компоненту z для точок перетинання. Невдача спроби знайти дійсний розв'язок означає, що промінь не перетинає поверхню. Ступінь системи рівнянь для u і w

дорівнює добутку ступенів біполіноміальних поверхонь. Бікубічна поверхня, наприклад, має шостий ступінь. Отже, у загальному випадку будуть потрібні чисельні методи розв'язку. Там, де це припустимо, для початкового наближення u і w можна використовувати перетинання променя з опуклою оболонкою. Для одержання перетинань у вихідній системі координат, як і раніше, варто застосувати зворотнє перетворення.

Якщо трасуючий промінь перетинає об'єкти сцени в декількох точках, то необхідно визначити видиме перетинання. Для алгоритмів визначення видимості простих непрозорих поверхонь, що обговорюються в даному розділі, перетинанням з видимою поверхнею буде точка з максимальним значенням координати z. Для складніших алгоритмів, що враховують відображення і заломлення, ці перетинання варто упорядкувати уздовж променя за відстанню від його початку. У перетвореній системі координат цього можна досягти простим сортуванням по z.

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

- Створити список об'єктів, що містить щонайменше наступну інформацію:
  повний опис об'єкта: тип, поверхня, характеристики і т.д.;
  опис сферичної оболонки: центр і радіус.
- Прапор прямокутної оболонки. Якщо цей прапор піднятий, то буде виконаний габаритний тест із прямокутною оболонкою, якщо ж опущений, то тест виконуватися не буде. Габаритний тест необхідний не для всіх об'єктів, наприклад, для сфери він не потрібний.
- Опис прямокутної оболонки: x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}
- Для кожного трасуючого променя:
  Виконати для кожного об'єкта тривимірний тест зі сферичною оболонкою у вихідній системі координат.
  Якщо промінь перетинає цю сферу, то занести об'єкт у список активних об'єктів.
  Якщо список активних об'єктів порожній, то зобразити даний піксел з фоновим значенням інтенсивності і продовжити роботу. У протилежному випадку, перенести і повернути промінь так, щоб він сполучився з віссю z.
  Запам'ятати це комбіноване перетворення.
- Для кожного об'єкта зі списку активних об'єктів:
  Якщо прапор прямокутної оболонки піднятий, перетворити, використовуючи комбіноване перетворення, цю оболонку в систему координат, у якій знаходиться промінь, і виконати відповідний тест. Якщо перетинання з променем немає, то перейти до наступного об'єкту.
  У протилежному випадку перетворити, використовуючи комбіноване перетворення, об'єкт у систему координат, в якій знаходиться промінь, і визначити його перетинання з променем, якщо вони існують.
  Занести всі перетинання в список перетинань.

- Якщо список перетинань порожній, то зобразити даний піксел з фоновим значенням інтенсивності.
  У протилежному випадку визначити z_{max} для списку перетинань.
- Обчислити перетворення, зворотнє комбінованому перетворенню.
- Використовуючи це зворотнє перетворення, визначити точку перетинання у вихідній системі координат.
- Зобразити даний піксел, використовуючи атрибути пересіченого об'єкта і відповідну модель освітленості.

Помітимо, що алгоритм визначення видимості простих непрозорих поверхонь, не вимагає обчислювати перетворення, зворотнє комбінованому, чи визначати точку перетинання у вихідній системі координат, якщо в моделі освітлення не виникає необхідність включення в алгоритм властивостей поверхні чи об'єкта, її орієнтації в точці перетинання. Ці кроки включені в даний алгоритм для повноти і зручності при реалізації алгоритму трасування променів з врахуванням загальної моделі освітленості. Більш повно приведені міркування проілюструємо на прикладі.

10.10. Приклад, який ілюструє роботу алгоритму, що використовує трасування променів

Візьмемо прямокутник і трикутник, що його протикає. Для простоти припустимо, що спостерігач розташований у нескінченності на додатній півосі z. Отже. усі промені паралельні осі z. Вісь x проходить через точку (0, 0) на растрі. Нагадаємо, що кути прямокутника розташовані в точках P_1(10, 5, 10), P_2(10, 25, 10). P_3(25, 25, 10), P_4(25, 5, 10). Центр його сферичної оболонки розташований у точці (17.5, 15, 10), а її радіус дорівнює 12.5. Значення x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max} прямокутної оболонки цього прямокутника рівні відповідно 10, 25, 5, 25, 10, 10.

Вершини трикутника розташовані в точках P_5(15, 15, 15), P_6(25, 25, 5), P_7(30, 10, 5). Центр його сферичної оболонки розташований у точці (22.885, 15.962, 8.846), а її радіус дорівнює 10.048. Прямокутна оболонка цього трикутника задається числами 15, 30, 10, 25, 5, 15.

Таким чином, у списку об'єктів міститься два елементи, і обидва прапори прямокутних оболонок підняті.

Розглянемо промінь, який проходить через центр піксела, що відповідає точці (20, 15). Оскільки спостерігач знаходиться в нескінченності, цей промінь паралельний осі z.

Спочатку візьмемо прямокутник. Оскільки промінь паралельний осі z, відстань від центра сферичної оболонки до променя не залежить від координати z. Конкретно, з огляду на те, що центр піксела знаходиться в точці (20.5, 15.5), одержуємо:

$$d^2=(20.5-17.5)^2+(15.5-15)^2=9.25$$

Тому що (d^2=9.25)<(R^2=156.25), то наш промінь перетинає сферичну оболонку прямокутника. Прямокутник заноситься в список активних об'єктів. Аналогічно для трикутника маємо:

$$d^2=(20.5-22.885)^2+(15.5-15.962)^2=5.90$$

Знову це число менше, ніж квадрат радіуса сферичної оболонки, тобто (d^2=5,90)<(R^2=100.96). Тому промінь перетинає і сферичну оболонку трикутника. Цей трикутник також заноситься в список активних об'єктів.

Оскільки список активних об'єктів не порожній, промінь піддається перетворенню, що сполучає його з віссю z. У нашому випадку промінь переноситься на (-20.5, -15.5, 0) у напрямках x, y, z, відповідно. Аналогічно перенесення прямокутної оболонки для прямокутника дає значення -10.5, 4.5, -10.5, 9.5, 10, 10. Оскільки знаки x_{min} і x_{max}, а також y_{min} і y_{max} протилежні, то промінь перетинає цю прямокутну оболонку. Точку перетинання променя з прямокутником знаходимо, використовуючи рівняння площини, що містить цей прямокутник. В обох системах координат до і після перетворення рівняння цієї площини має вигляд:

$$z-10=0.$$

Тому перетинання її з променем відбувається при z=10. Точка перетинання лежить всередині прямокутника. Ця точка і заноситься в список точок перетинання. Перенос прямокутної оболонки для трикутника дає значення -5.5, 9.5,-5.5, 9.5, 5, 15. Знову знаки x_{min} і x_{max}, а також y_{min} і y_{max} протилежні; тому промінь перетинає цю оболонку і для трикутника. У вихідній системі координат рівняння площини трикутника таке:

$$3x+y+4z-120=0.$$

У перетвореній системі координат воно має вигляд (див. розд.4.2);

3x+y+4z-43=0 і перетинання відбувається при:

$$z=(43-3x-y)/4=43/4=10.75$$

Ця точка лежить всередині трикутника і заноситься в список точок перетинання.

Список точок перетинання не порожній. Максимальним є значення z_{max}=10.75, і трикутник є видимим. Зворотнє перетворення до вихідної системи координат дає точку перетинання (20.5, 15.5, 10.75). Піксел, що відповідає точці (20, 15), зображується з атрибутами трикутника.

Дві модифікації цього простого алгоритму помітно підвищують його ефективність. Перша модифікація ґрунтується на понятті кластерних груп просторово зв'язаних об'єктів. Наприклад, припустимо, що сцена складається зі столу, на якому стоять ваза з фруктами і тарілка з цукерками. У вазі лежать апельсин, яблуко, банан і груша. Тарілка містить кілька цукерок різних форм і кольорів. Вводяться сферичні оболонки для груп кластерів чи зв'язаних об'єктів, наприклад для вази і всіх плодів у ній, для тарілки і всіх цукерок у ній, а також для столу і всіх предметів на ньому. Сферичні оболонки, що охоплюють більш ніж один об'єкт, називаються сферичними кластерами. Якщо це необхідно, то можна ввести і прямокутні кластери. Вводиться, крім цього, найбільший сферичний кластер, іменований сферою сцени, що охоплює всі об'єкти в цій сцені. Потім сферичні оболонки обробляються в ієрархічному порядку. Якщо промінь не перетинає сферу сцени, то він не може перетнути і жодного з її об'єктів. Отже, піксел, що відповідає цьому променю, буде зображений з фоновим значенням інтенсивності. Якщо ж промінь перетинає сферу сцени, то на перетинання з променем перевіряються сферичні кластери і сферичні

оболонки об'єктів, що не містяться в жодному зі сферичних кластерів, але приналежних кластеру сцени. Якщо промінь не перетинає сферичний кластер, то сам цей кластер і всі об'єкти чи кластери, що містяться в ньому, виключаються з подальшого розгляду. Якщо ж промінь перетинає кластер, то ця процедура рекурсивно повторюється доти, поки не будуть розглянуті всі об'єкти. Якщо промінь перетинає сферичну оболонку деякого об'єкта в якій-небудь точці, то цей об'єкт заноситься в список активних об'єктів. Ця процедура значно скорочує кількість обчислень точок перетинання променя зі сферичними оболонками і тим самим підвищує ефективність всього алгоритму.

Друга модифікація використовує упорядкування по пріоритету, щоб скоротити число об'єктів, для яких обчислюються перетинання з променем. Замість того, щоб негайно робити обчислення перетинання об'єкта променем, як це робиться у викладеному вище простому алгоритмі, об'єкт поміщується в список пересічених об'єктів. Після розгляду всіх об'єктів сцени перетворений список пересічених об'єктів упорядковується по пріоритету глибини (див. алгоритм, що використовує список пріоритетів). Для визначення пріоритетного порядку можна використовувати центри сферичних оболонок чи найбільші (найменші) значення z прямокутних оболонок. Перетинання променя з об'єктами зі списку пересічених об'єктів визначаються в порядку їх пріоритетів. На жаль, як раніше вказувалося в описі алгоритму, що використовує список пріоритетів, точка перетинання променя з першим з об'єктів в упорядкованому по пріоритетах списку пересічених об'єктів не обов'язково буде видимою. Необхідно визначити точки перетинання променя з усіма потенційно видимими об'єктами з множини *{Q}* (деталі див. в алгоритмі, що використовує список пріоритетів) і занести їх в список перетинань. Потім модифікований алгоритм упорядковує цей список перетинань так, як це робилося й у простому алгоритмі. На щастя, множина *{Q}* потенційно видимих об'єктів звичайно значно менша числа об'єктів у списку пересічених променем. Отже, ефективність алгоритму зростає. Обидві ці модифікації застосовні також і до загального алгоритму трасування променів, що враховує відображення, заломлення і прозорість.

Викладений вище простий алгоритм не використовує тієї обставини, що деякі грані багатогранника є нелицьовими і їх можна відразу видалити, не враховується тут і можлива когерентність сцени. Наприклад, несуттєвий порядок обробки пікселів. Разом з тим розгляд цих пікселів у порядку сканування рядка розгортки дозволило би скористатися в алгоритмі когерентністю скануючих рядків.

Інший підхід може полягати в підрозділі сцени в дусі алгоритму Варнока, причому врахування когерентності областей привело б до зменшення числа об'єктів, розглянутих для кожного променя і, отже, до підвищення ефективності алгоритму. Хоча використання подібних прийомів підвищує ефективність алгоритму визначення видимості непрозорих поверхонь, їх неможливо застосувати в загальному алгоритмі трасування променів, що враховує відбивання, заломлення і прозорість.

Наприклад, якщо в алгоритмі враховане відбивання, то об'єкт, що цілком закритий іншим об'єктом, може виявитися видимим, як відбивання від третього об'єкта. Оскільки метод трасування променів є методом грубої сили, алгоритми визначення видимості непрозорих поверхонь, що обговорювалися в попередніх розділах, є більш ефективними, і саме ними потрібно користатися [Експериментальні оцінки алгоритмів, описаних у попередніх розділах отримані на програмах, написаних на одній мові і запущених на одній обчислювальній системі, для однієї сцени, зв'язані між собою наступним відношенням: Трасування променів: Варнок: Уоткінс: Сканування з z-буфером: z-буфер - 9.2:6.2:2.1:1.9:1.].

Рот [20] вказав, що алгоритм трасування променів можна використовувати також і для створення каркасних креслень суцільних тіл. При цьому передбачається, що промені породжуються в тому порядку, в якому відбувається сканування екрана, тобто зверху вниз і зліва направо. Процедура, що виходить, така:

Якщо видима поверхня для Піксел(x, y) відповідає фону чи відрізняється від видимої поверхні для Піксел($x-1$, y) чи для Піксел(x, $y-1$), то зобразити цей піксел. У протилежному випадку піксел не зображувати.

Алгоритм трасування променів можна використовувати, крім того, для визначення фізичних властивостей суцільного тіла. Повний розгляд цього питання не входить до завдання даної книги. Однак для ілюстрації цього підходу наведемо один приклад. Зокрема, обсяг будь-якого суцільного тіла можна визначити, апроксимуючи його сумою маленьких прямокутних паралелепіпедів. Це можна проробити, породивши безліч паралельних променів, розташованих на визначених відстанях один від другого. Точки перетинання кожного променя з заданим обсягом обчислюються й упорядковуються вздовж напрямку цього променя. Якщо піддати промінь переносу, що сполучає його з віссю z, як це було описано вище, то обсяг кожного прямокутного паралелепіпеда буде дорівнювати:

$$V=l_x l_y[(z_1-z_2)+(z_3-z_4)+...+(z_{n-1}-z_n)]$$

де l_x і l_y - відстань між променями по горизонталі і вертикалі відповідно. Кожен доданок $(z_{i-1}-z_i)$ являє собою ділянку променя, що лежить всередині заданого тіла. Обсяг тіла, отже, дорівнює сумі обсягів усіх таких прямокутних паралелепіпедів. Точність результатів залежить від числа використаних променів. Точність можна підвищити, помірковано збільшивши обсяг обчислень і рекурсивно зменшивши розмір "піксела", у тому випадку, якщо обсяги суміжних прямокутних паралелепіпедів розрізняються більш ніж на заздалегідь задану величину. При такому підході точніше визначаються обсяги тих елементів тіла, де мають місце швидкі зміни, наприклад в околах ребер тіл, обмежених криволінійними поверхнями.

Через внутрішньо властиву алгоритму трасування променів паралельність обчислень (тут всі промені проробляються однаково і незалежно один від одного) його можна реалізувати апаратно на основі надпотужних інтегральних схем (НПІС) з використанням методів паралельної обробки.

Література

1. Аммерал Л. Машинная графика на персональных компьютерах. М.: "Сол Систем", 1992.
2. Аммерал Л. Программирование графики на Турбо Си. М.: "Сол Систем", 1992.
3. Болтянский В.Г., Ефремович В.А. Наглядная топология. М.: Наука, 1983.
4. Ватолин Д.С. Алгоритмы сжатия изображений. Методическое пособие. М.: Издательский отдел факультета вычмслительной математики и кибернетики МГУ им. М.В.Ломоносова, 1999, -76с.
5. Вельтмандер П.В. Машинная графика. Учебное пособие в 3-х книгах. "Новосибирский государственный университет", 1997.
6. Демидов В. Как мы видим то, что видим. М.: "Знание", 1979.
7. Котов Ю. В. Как рисует машина. М.: "Наука", 1988.
8. Паленичка Р.М., Грицик В.В. Англо-російсько-український тлумачний словник з комп'ютерної графіки та обробки зображень. К.: Наукова думка, 1994.
9. Роджерс Д. Алгоритмические основы машинной графики. М.: "Мир", 1989.
10. Фокс А., Пратт М. Вычислительная геометрия применение в проектировании и на производстве. М.: "Мир", 1982.
11. Butland J. Surface Drawing Made Simple. CAD Jurnal vol 11 pp19-22 1979.
12. Butland, J., "Surface Drawing Made Simple" CAD Jurnal vol 11 pp19-22 1979.
13. Catmull, Edwin, "A Subdivision Algorithm for Computer Display of Curved Surfaces" Ph.D. Thesis University of Utah, Dec. 1974 Also UTEC-CSs-74-133 and NTIS A004 968.
14. Catmull, Edwin, "Computer Display of Curved Surfaces" Proc. IEEE conf. comput. grafics pattern recognition data struct. May 1975 p11.
15. Catmull, Edwin. Computer Display of Curved Surfaces. Proc. IEEE conf. comput. grafics pattern recognition data struct. May 1975 p11.
16. Gottlib M. Hidden Line Subroutines for Three-Dimensional Plotting. Byte vol 3 No 5 pp49-58 1978.
17. Gottlib, M., "Hidden Line Subroutines for Three-Dimensional Plotting" Byte vol 3 No 5 pp49-58 1978.
18. Newell M.E., Newell R.G., Sancha T.L. A New Approach to the Shaded Picture Problem. Proc. ACM natl. conf. 1972 pp443-450.
19. Newell M.E., The Utilization of Procedure Models in Digital Image Systhesis. Ph.D. Thesis University of Utah, 1974 Also UTEC-CSs-76-218 and NTIS AD/A039 008/LL.
20. Newell, M.E., "The Utilization of Procedure Models in Digital Image Systhesis" Ph.D. Thesis University of Utah, 1974 Also UTEC-CSs-76-218 and NTIS AD/A039 008/LL.
21. Newell, M.E., Newell R.G., and Sancha, T.L., "A New Approach to the Shaded Picture Problem" Proc. ACM natl. conf. 1972 pp443-450.

22. Schachter, Bruce J. Computer Image Generation, John Wiley, New York, 1982.
23. Schachter, Bruce J., Computer Image Generation, John Wiley, New York, 1982.
24. Schumacher R.A., Bruce J., Gilliand M., Sharp W., Study for Applying Computer-generated Images to Visual Simulation. U.S. Air Force human resources lab. tech. rep., AFHRL-TR-69-14 sept. 1969 NTIS AD 700 375.
25. Schumacher, R.A., Bruce J., Gilliand, M., Sharp, W., "Study for Applying Computer-generated Images to Visual Simulation" U.S. Air Force human resources lab. tech. rep., AFHRL-TR-69-14 sept. 1969 NTIS AD 700 375.
26. Sutherland, Ivan E., Sproull, Robert F., and Schumacher, R.A., "A Characterization of Ten Hidden-Surface Algorithms" Computing surveys vol 6 pp1-55 1974.
27. Sutherland, Ivan E., Sproull, Robert F., Schumacher R.A. A Characterization of Ten Hidden-Surface Algorithms. Computing surveys vol 6 pp1-55 1974.
28. Watkins, Steven L. Algorithm 483, Masked Three-Dimensional Plot Program with Rotations. CACM vol 17 pp520-523 1974.
29. Watkins, Steven, L., "Algorithm 483, Masked Three-Dimensional Plot Program with Rotations" CACM vol 17 pp520-523 1974.
30. Williams, Hugh, "Algorithm 420, Hidden-Line Plotting Program" CACM vol 15 pp100-103 1972.
31. Williams, Hugh. Algorithm 420, Hidden-Line Plotting Program. CACM vol 15 pp100-103 1972.
32. Wright T.J. A Two-Space Solution to the Hidden Line Problem for Plotting Function of Two Variables. IEEE Trans. comput., vol C-22 pp 28-33 1973.
33. Wright, T.J, "A Two-Space Solution to the Hidden Line Problem for Plotting Function of Two Variables", IEEE Trans. comput., vol C-22 pp 28-33 1973.

НАВЧАЛЬНЕ ВИДАННЯ

ТЕХНІЧНІ, МАТЕМАТИЧНІ ТА АЛГОРИТМІЧНІ ОСНОВИ КОМП'ЮТЕРНОЇ ГРАФІКИ

Укладач: Різник Олег Яремович

Комп'ютерна верстка: Різник Олег Яремович

Редактор: Різник Олег Яремович