Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Я. М. Клятченко, О. В.Тарасенко-Клятченко

Комп'ютерна графіка

Конспект лекцій

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за спеціальністю 123 Комп'ютерна інженерія

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2024

УДК 004.92 (075.8)
К52

Автори: *Клятченко Ярослав Михайлович*, канд. техн. наук, доц.
Тарасенко-Клятченко Оксана Володимирівна, канд. техн. наук, доц.

Рецензент *Заболотня Т.М.*, канд. техн. наук, доц., кафедра програмного забезпечення комп'ютерних систем КПІ ім. Ігоря Сікорського

Відповідальний редактор *Тарасенко В.П.*, д-р техн. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 5 від 29.02.2024 р.)
за поданням вченої ради факультету/навчально-наукового інституту
(протокол № 7 від 29.01.2024 р.)

Клятченко Я. М.

К52 Комп'ютерна графіка [Електронний ресурс] : конспект лекцій : навч. посіб. для здобувачів ступеня бакалавра за спец. 123 Комп'ютерна інженерія / Я. М. Клятченко, О. В.Тарасенко-Клятченко ; КПІ ім. Ігоря Сікорського. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2024. – 128 с.

Навчальний посібник містить теоретичні відомості з освітнього компонента «Інженерна та комп'ютерна графіка. Частина 2. Комп'ютерна графіка», які необхідні для опанування матеріалу курсу, виконання лабораторних робіт та подальшої самостійної роботи.

Список рекомендованої літератури дозволить докладніше вивчити теоретичний матеріал з тем курсу. Навчальне видання призначене для студентів, які навчаються за спеціальністю 123 «Комп'ютерна інженерія» факультету прикладної математики КПІ ім. Ігоря Сікорського.

УДК 004.92 (075.8)

Реєстр. № НП 23/24-XXX. Обсяг X,X авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

Вступ

Дисципліна “Комп’ютерна графіка” (КГ) є однією із дисциплін обов’язкової частини програми підготовки студентів за спеціальністю 123 „Комп’ютерна інженерія”. Вивченню дисципліни передує вивчення дисциплін „Структури даних та алгоритми”, „Програмування”, „Об’єктно-орієнтоване програмування”, „Інженерна графіка”, „Лінійна алгебра та аналітична геометрія”.

Завдання дисципліни КГ передбачають розгляд і аналіз класичних алгоритмів та методів, розгляд сучасних графічних систем, вивчення принципів організації систем комп’ютерного синтезу зображень, математичного моделювання форм та вигляду об’єктів, методів їх візуалізації та анімації, програмного забезпечення растрової машинної графіки, а також принципів формування реалістичних зображень.

Як не можна не помітити, комп’ютерна графіка сьогодні займає значне місце у багатьох сферах життя людини, у великій кількості галузей промисловості вона відіграє ключову роль (наприклад, у поліграфії, у веб-дизайні, тощо).

З появою можливості попереднього перегляду моделей, проектів, креслень комп’ютерна графіка все більше актуалізується, фактично перетворившись певною мірою на зір розробника чи дизайнера. Без комп’ютерної графіки важко уявити освіту: вона присутня у всьому - від використання звичайних комп’ютерів (моніторів) і до графічних планшетів. Під час навчання спеціалістів різних галузей засоби комп’ютерної графіки також широко використовуються.

Сьогодні комп’ютерну графіку можна віднести до складових частин всіх сучасних інформаційних технологій. Графічний користувацький інтерфейс (GUI) фактично став стандартом програмного забезпечення сучасних

інформаційних систем і тому комп'ютерна графіка стала основою засобів візуалізації даних. Хоча комп'ютерна графіка – така галузь знань, де багато вже зроблено і накопичено великий обсяг знань, але вона все одно постійно розвивається. Але студентам, як майбутнім спеціалістам, потрібно володіти програмуванням та основами базових математичними концепцій, щоб забезпечити подальше розуміння та розвиток комп'ютерної графіки.

Мета курсу КГ - ознайомитись із базовими класичними алгоритмами КГ, висвітлити, розкрити різні можливості використання комп'ютерної графіки, розглянути всі сучасні напрями розвитку комп'ютерної графіки.

В результаті вивчення дисципліни формуються знання, необхідні для розробки засобів КГ і їх застосування на практиці. В процесі виконання лабораторних робіт у студентів формується система знань в даній області, розвиваються уміння і навички, необхідні для розробників і користувачів систем КГ.

1. Поняття комп'ютерної графіки

Розглянемо спрощене визначення. На перший погляд, для людини, далекої від галузі інформатики та обчислювальної техніки, комп'ютерна графіка представляється як галузь інформатики, що займається створенням, зберіганням і змінюванням (редагуванням) різних зображень (малюнків, креслень, анімації) за допомогою засобів обчислювальної комп'ютерної техніки.

Сфери же застосування КГ можна окреслити як візуалізація даних в різних сферах людської діяльності, наприклад:

- медицина (комп'ютерна томографія);
- візуалізація наукових і ділових даних;
- дизайн (для реклами, поліграфії, моделювання та ін.);
- графічний інтерфейс користувача;
- системи автоматизованого проектування;
- цифрова фотографія;
- цифрове телебачення, Інтернет, відео-конференції;
- комп'ютерні ігри, системи віртуальної реальності;
- спецефекти, цифрова кінематографія.

Далі наведемо спрощені класифікації за різними характеристиками.

- За спеціалізацією в різних галузях діяльності, комп'ютерну графіку поділяють на інженерну, наукову, веб-графіку, комп'ютерну поліграфію.
- За способами представлення кольорів виділяють монохромну та кольорову графіку.
- Комп'ютерну графіку поділяють також на статичну (фотографії, рисунки, схеми, діаграми, окремі елементи оформлення ВЕБ-сторінок) та динамічну (анімація, комп'ютерна мультиплікація).

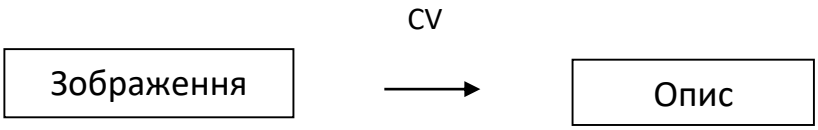
Комп’ютерна графіка - складна багатовекторна дисципліна, вивчення якої потрібно почати з розгляду її складових та способів взаємодії між ними, враховуючи той факт, що «продуктом» комп’ютерної графіки є зображення, яке, в свою чергу, можна використовувати для найрізноманітніших цілей.

Зображення є тим наріжним каменем, навколо якого об’єднуються такі дисципліни КГ, як COMPUTER VISION, IMAGE PROCESSING і COMPUTER GRAPHICS.

I. COMPUTER (MACHINE) VISION – комп’ютерний (машинний) зір (розпізнавання образів (зображень), аналіз сцен).

Припустимо, є деяка сцена (кімната, приміщення, обстановка тощо). Обчислювальна система (комп’ютер) повинна дати опис цієї сцени (чи є в ній предмети, яка освітленість та т.ін.). Computer Vision (CV) переводить зображення в опис і робить те, що можна назвати розпізнаванням образів (зображень).

CV (розпізнавання образів (зображень)) є сукупністю методів, що дозволяють отримати опис зображення, поданого на вхід, або віднести задане зображення до деякого класу (тут вже постають задачі класифікації та кластеризації).



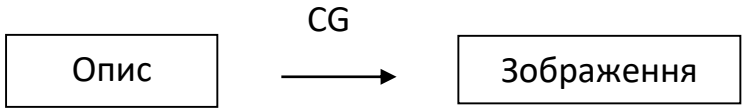
Інакше кажучи, CV займається аналізом образів сцени. Предметом CV є дослідження абстрактних моделей графічних об’єктів и взаємозв’язків між ними. До особливостей CV можна віднести те, що об’єкти можуть бути як синтезованими, так и виділеними на фотознімках.

Перший крок в аналізі сцени - виділення характерних особливостей, що формують графічний(і) об’єкт(и). Приклади: машинний зір роботів, аналіз

відеозображень, з виділенням та відстеженням об’єкта. Хоча CV та MV раніше вважалися тотожними, сьогодні вони виконують різні завдання, хоча суть одна і та сама. CV проводить більш глибокий аналіз сцени, а MV використовується на виробництві як зір роботів (конвеєр, сортування продукції, контроль і т.д.). Отже, в основі CV лежать комп’ютерна (образотворча) графіка + аналіз зображень + спеціалізовані засоби обробки зображень.

II. COMPUTER (MACHINE) GRAPHICS - комп’ютерна (машинна) графіка

Розглянемо задачі, що вирішують системи комп’ютерної графіки: нехай у нас є опис якого-небудь об’єкта, явища, тобто є віртуальний образ. Ми хочемо від опису перейти до зображення.

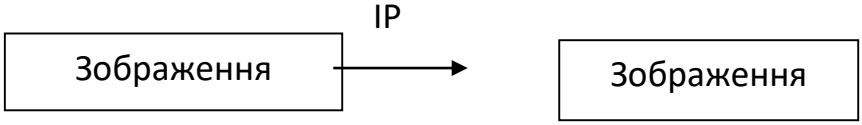


Комп’ютерна графіка має справу з синтезом зображення (наприклад, є опис предметів, джерел світла, а треба отримати цілу картину).

Об’єкти: синтезовані зображення.

Завдання CG: побудова моделі об’єкта и генерування зображення,

III. IMAGE PROCESSING – обробка зображень (обробка та аналіз зображень).



IP (Обробка зображень) розглядає задачі, у яких і вхідні, і вихідні дані є зображеннями, і виконується їхнє перетворення.

Об’єкти: дискретне, числове подання фотографій (зображень).

Завдання:

- підвищення якості зображення;
- отримання зображення із потрібними характеристиками;

- обробка зображення з метою знаходження (виділення) певних рис, фрагментів;
- препроцесинг – попередня підготовка даних (зображень) для передачі до систем аналізу (як складова систем розпізнавання образів (CV)).

Отже, в основі обробки та аналізу зображень знаходяться методи *представлення, обробки та аналізу* зображень плюс комп'ютерна графіка для представлення результатів.

2. Комп'ютерне подання зображення

Очевидно, що в комп'ютерній графіці фундаментальною сполучною ланкою є зображення. Причому, будемо розрізняти зображення з фізичної точки зору - це зображення оптичне, - картина, отримана в результаті проходження через оптичну систему променів, що поширюються від об'єкта і відтворюють його контури і деталі. У рамках дисципліни КГ надалі під терміном «зображення» підрозуміється набір ліній, точок, текстів тощо, що виводяться на графічний пристрій. Таким чином в цьому контексті комп'ютерна графіка має справу з розв'язком наступних завдань:

- представлення зображень;
- підготовка зображень для виводу;
- візуалізація попередньо підготовлених зображень;
- взаємодія із зображенням.

Отже, розглянувши особливості галузі комп'ютерної графіки, можна сформулювати наступне визначення: комп'ютерна графіка – галузь інформатики, у межах якої вивчають методи створення та перетворення графічних зображень об'єктів за допомогою обчислювальних засобів. Комп'ютерна графіка використовується для введення інформації, що початково має графічну форму або визначає спосіб її введення до обчислювальної машини, обробки, оптимізації характеристик, зберігання на носіях, захисту та виведення графічної інформації з комп'ютерного засобу.

Під графічною формою подання інформації розуміють: ескізи, креслення, візуальне подання каркасних та твердотільних 3D–моделей найрізноманітніших об'єктів, схеми, діаграми, графіки, рисунки, анімація.

В основу розв'язування задач сучасної комп'ютерної графіки покладено фундаментальні теоретичні положення аналітичної та диференційної геометрії, векторної алгебри, нарисної геометрії, чисельних методів,

математичної логіки, методів оптимізації, теорії прийняття рішень, методів розпізнавання образів.

3. Зоровий апарат людини. Світло і колір

Слід розрізняти те, як зображення сприймається людиною через органи зору і те, яким чином зображення вводиться в комп'ютер, а також формується (синтезується), і за допомогою яких засобів.

Видиме світло складається із спектрального розподілу електромагнітної енергії з довжинами хвиль в діапазоні ≈ 400 ÷ 700 нм. Колір - одна з властивостей матеріальних об'єктів, що сприймається як усвідомлене зорове відчуття. Той чи інший колір «присвоюється» людиною об'єкту в процесі зорового сприйняття цього об'єкта. У величезній більшості випадків колірне відчуття виникає в результаті впливу на око потоків так званого видимого випромінювання, що сприймається оком - випромінювання з довжинами хвиль від 380 до 760нм.

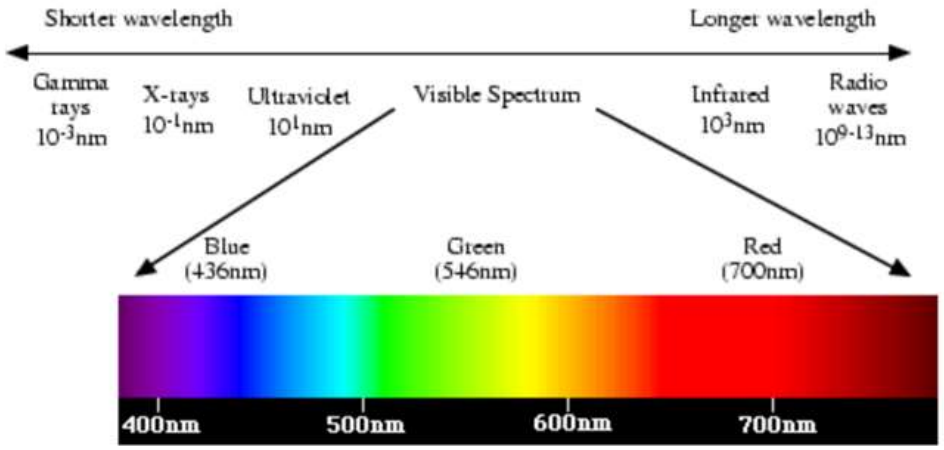


Рисунок 1 - Видиме випромінювання

Спектральний колір - колір випромінювань, довжини хвиль яких розташовані в діапазоні видимого світла в певних інтервалах навколо довжини якого-небудь монохроматичного випромінювання. Випромінювання з довжинами хвиль від 380 до 470 нм мають фіолетовий і синій колір, від 470 до 500 нм - синьо-зелений, від 500 до 560 нм - зелений, від 560 до 590 нм -

жовто-помаранчевий, від 590 до 760 нм - червоний (в інших ділянках цих інтервалів колір випромінювань відповідає різним відтінкам вказаних кольорів).

Система зору людини, включає два типи світлочутливих фоторецепторів: колбочки (короткі і товсті рецептори денного зору і розташовані в основному по периферії сітківки) та, так звані, палички (довгі і тонкі рецептори нічного зору).

Будучи трихроматом, людина має три типи світлочутливих рецепторів, або, іншими словами, зір людини трикомпонентний. За чутливістю колбочки поділяють на три типи: B, G і R, або як S, M і L. Піки їх чутливості припадають приблизно на 440 нм, 545 нм і 580 нм (для "усередненого" спостерігача).

Цим довжинах хвиль відповідає не синій, зелений і червоний кольори, а фіолетовий, синьо-зелений і жовто-зелений. Варто відзначити, що чутливість ока до синього кольору істотно нижче, ніж до зеленого і червоного(приблизно у 10 разів).

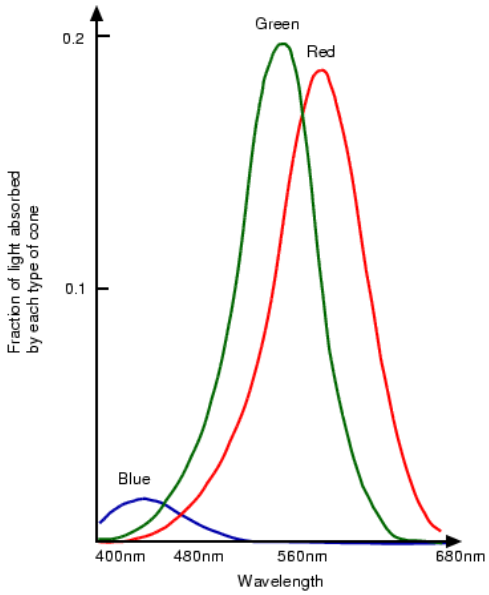


Рисунок 2 - Чутливість ока до різних кольорів

4. Колірні моделі

У КГ є поняття колірної моделі. Колірні моделі, що використовують у комп'ютерній графіці, - це засоби для опису кольорів в певному діапазоні. Серед цих колірних моделей можна умовно розрізняти моделі, що орієнтовані на людину (на зоровий апарат та сприйнятті людиною або перцепційні) та т.зв. апаратно-орієнтовані колірні моделі, що пристосовані до використання у системах кольорового телебачення, системах комп'ютерної графіки, поліграфії тощо. Апаратно-орієнтовні моделі поділяються на адитивні (RGB) та субрактивні (CMY, CMYK).

Колірні моделі орієнтована на людину:

Вирізняють такі три характеристики кольору:

-колірний тон (КТ, hue);

-насиченість (saturation);

-значення кольору (value) або яскравість (brightness).

Найбільш важливий атрибут кольору асоціюється в людській свідомості з обумовленістю забарвлення предмета певним типом пігменту, фарби, барвника. Насиченість характеризує ступінь, силу, рівень вираження КТ.

HSV (також HSB) — колірна модель, заснована на трьох характеристиках кольору: колірному тоні (hue), насиченості (saturation) і значенні кольору (value), який також називають яскравістю (brightness).

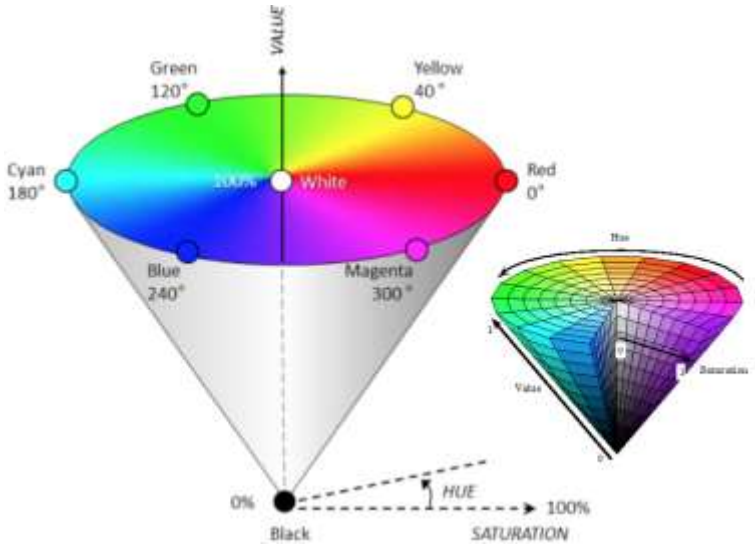


Рисунок 3 - Система координат та колірна модель HSV

• Hue — колірний тон, (наприклад, червоний, зелений або синьо-блакитний). Варіюється в межах 0-360°, але іноді приводиться до діапазону 0-100 або 0-1. У Windows весь колірний спектр ділиться на 240 відтінків, тобто тут «Hue» зводиться до діапазону 0-239 (відтінок 240 відсутній, оскільки він дублював би 0).

• Saturation — насиченість. Варіюється в межах 0-100 або 0-1. Чим більший цей параметр, тим «чистіший» колір, тому цей параметр іноді називають чистотою кольору. А чим ближчий цей параметр до нуля, тим ближчий колір до нейтрального сірого.

• Value — значення кольору, або Brightness — яскравість. Також задається в межах 0-100 або 0-1.

Ця модель є нелінійним перетворенням моделі RGB. Колір, представлений в HSV, залежить від пристрою, на який він буде виведений, так як HSV — перетворення моделі RGB, яка теж залежить від пристрою. Для отримання коду кольору, не залежного від пристрою, використовується модель Lab. Слід зазначити, що HSV (HSB) і HSL — дві різні колірні моделі. Тривимірні візуалізації простору HSV: циліндр або конус (див. рис.3).

HSL (скорочено від англ. hue, saturation, lightness) — колірна модель, в якій будь-який колір визначається трьома характеристиками: кольоровим тоном (англ. hue), наприклад, синім, червоним, жовтим тощо; насиченістю (англ. saturation), тобто частиною чистого кольору, без домішки чорної та білої фарб; «світлотою» (англ. lightness), тобто близькістю до білого кольору (див. рис.4).

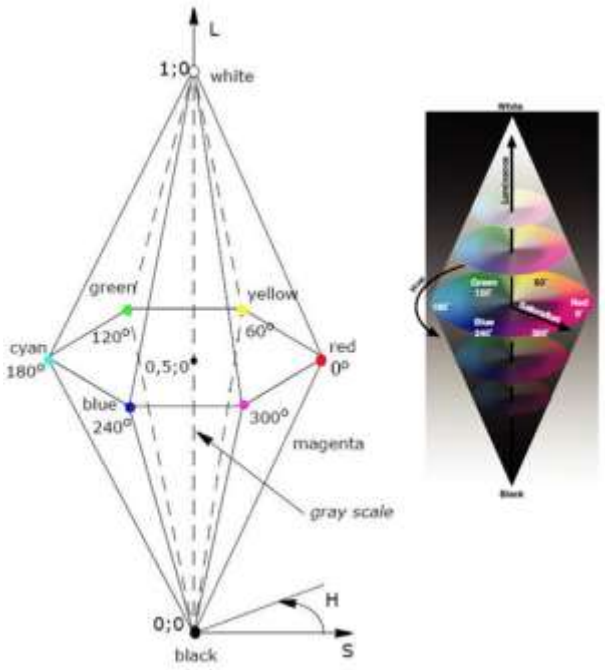


Рисунок 4 - Система координат та колірна модель HSL

Розглянуті інтуїтивні колірні моделі незручні для застосування в технічних засобах, що відтворюють кольорові зображення. У цих випадках широко застосовується інші моделі які ми розглянемо далі.

В комп'ютерній графіці розрізняють 2 види об'єктів: випромінюючі (монітори, світлодіодні матриці) та не випромінюючі (папір, світлофільтри).

Для випромінюючих об'єктів використовується адитивне формування відтінків, коли необхідний колір формується за рахунок змішування трьох основних відтінків кольорів.

У цьому випадку зручно використання моделі змішування RGB (Red, Green, Blue - червоний, зелений, синій). RGB (Red, Green, Blue - червоний,

зелений, синій) - апаратно-орієнтована модель, використовувана в дисплеях для адитивного формування відтінків самосвітних об’єктів (пікселів екрану). Система координат RGB – куб (див. рис.5) з початком відліку (0,0,0), відповідним чорному кольору (див. рис.6). Максимальне значення RGB - (1,1,1) відповідає білому кольору.

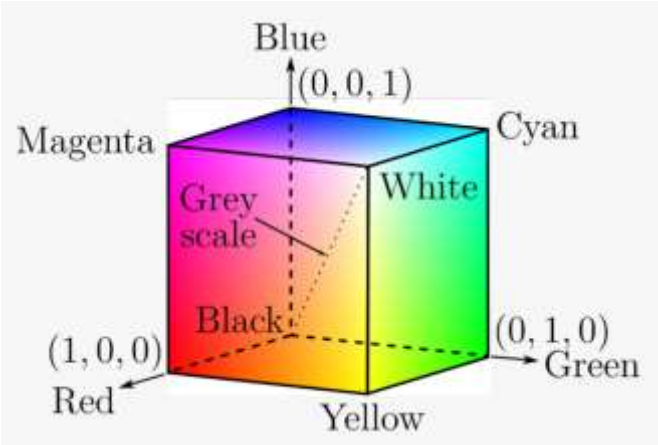


Рисунок 5 - Колірний куб моделі RGB

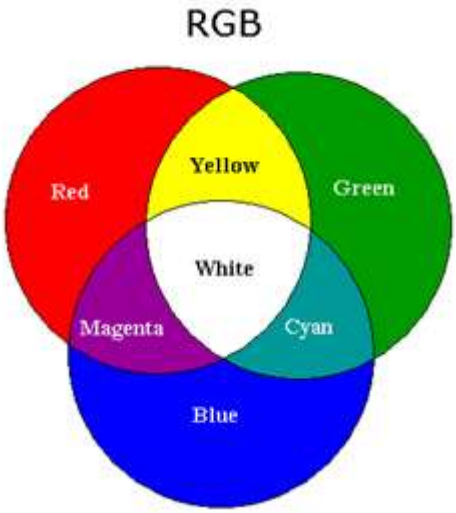


Рисунок 6 - Змішування кольорів для моделі RGB

Для невипромінюючих об’єктів використовується субтрактивне формування відтінків, засноване на відніманні з падаючого білого світла певних довжин хвиль (див. рис.7).

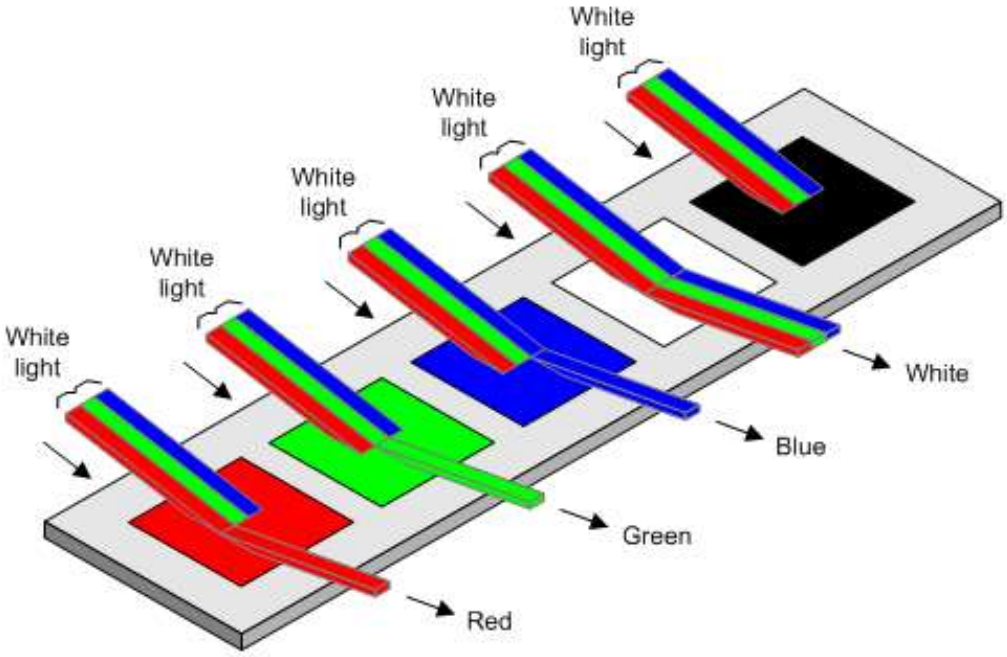


Рисунок 7 - Субтрактивне формування відтінків

У цьому випадку зручно використання моделі змішування CMY (Cyan, Magenta, Yellow - блакитний, пурпурний, жовтий). CMY (Cyan, Magenta, Yellow - блакитний, пурпурний, жовтий) - апаратно-орієнтована модель, використовувана в поліграфії для субтрактивного формування відтінків, заснованого на відніманні шаром фарби частини падаючого світлового потоку. Кольори моделі CMY є доповнюючою до кольорової моделі RGB, тобто доповнюють їх до білого. Таким чином система координат CMY - той же куб, що і для RGB, але з початком відліку в точці з RGB координатами (1,1,1), відповідної білому кольору (див. рис. 8, 9).

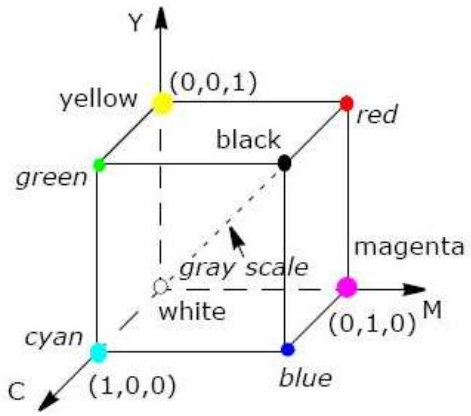


Рисунок 8 - Колірний куб модели CMY

Рисунок 9 - Змішування кольорів моделі CMY

Перетворення між просторами RGB і CMY визначаються наступним чином:

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

5. Формування зображення

В загальному плані зображення можна охарактеризувати за допомогою функції інтенсивності (яскравості) каналу: де *I = g(x, y)* – безперервна функція інтенсивності

$$I = g(x, y), \{x \in [x_0, x_1], y \in [y_0, y_1]\}$$

або використовується дискретне представлення

$$I = \mathrm{g}_{\mathrm{ij}}, \{i = \overline{1, n},\ j \in \overline{1, m}\}$$

Якщо брати значення цієї функції в певних точках, то буде отримано матрицю (кожний елемент матриці g_{ij} є значенням функції інтенсивності у точці *(i,j)*). За допомогою такої матриці зображення представляється в обчислювальній машині.

Елемент відображуваної матриці - *picture element (pixel)* - найменший елемент двовимірного цифрового зображення, що його формує. Піксель являє собою неподільний об'єкт прямокутної або іншої форми, що характеризується певним кольором. Растрове комп'ютерне зображення складається з пікселів, розташованих по рядках і стовпцях матриці екрану.

Ще одне визначення. Під терміном *піксель (pixel)* розуміється найменший елемент поверхні візуалізації, якому може бути незалежним чином задані колір, інтенсивність і інші характеристики зображення.

Існують зовсім прості зображення, які ми будемо називати бінарними, у яких значеннями яскравості можуть бути тільки два значення: 0 і 1. Такі зображення називають монохромними. Ці зображення містять тільки 2 кольори (наприклад, зелений і чорний), але зазвичай це чорно-білі зображення. Тобто в цьому випадку в матриці достатньо 1 розряду, щоб закодувати зображення в даній точці: 0 - відсутність світла (чорний); 255 - білий.

Якщо зображення базується на 3-х кольорах (Red (червоний), Green (зелений), Blue (блакитний)), то для кожного пікселя потрібно 3 байта (див. рис. 10, 11).

Як вже зазначалось, піксель - це світлова пляма на екрані монітора, яка може приймати різні відтінки. Будь-яке зображення незалежно від його складності - це всього лише сукупність пікселів і тут головне, щоб ці пікселі з потрібними відтінками поміщалися в потрібне місце екрану, тим самим створюючи цілісне зображення.

Таким чином, можна сформулювати ще одне із завдань комп'ютерної графіки як науки: це розробка методів для перетворень того, що потрібно побачити на екрані, в масиви значень кольорів пікселів на екрані монітора.

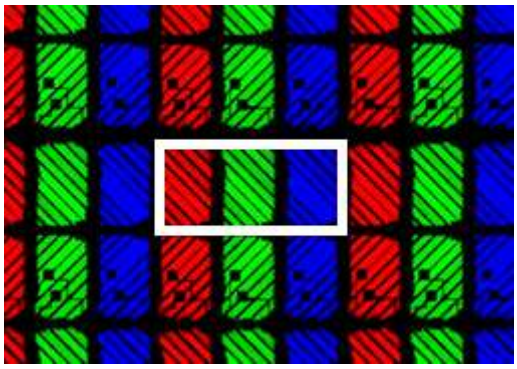
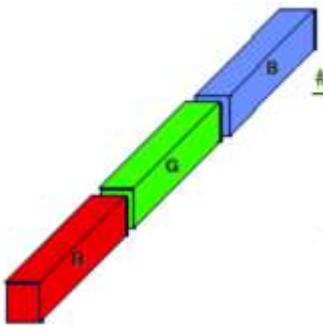


Рисунок 10 - Кольорові пікселі зображення



# Bits/color	# Shades per color	# Bits/pixel	Total colors:
4	2^4 = 16	12	2^{12} = 4K
6	2^6 = 64	18	2^{18} = 256K
8	2^8 = 256	24	2^{24} = 16.7M

Рисунок 11 - Глибина кольору

Одним з найважливіших параметрів монітора є роздільна здатність екрану, що визначається кількістю пікселів у кожному рядку і в кожному стовпці. Це може бути, наприклад, роздільна здатність: 1024 на 768 або вище. Змінювати колір кожного пікселя можна незалежно, але кількість відтінків, які одночасно можуть бути присутніми на екрані, залежить від графічної плати (відеоадаптера).

6. Растрова графіка

Растр (від англ. raster) - спосіб представлення зображення у вигляді двовимірного масиву точок, впорядкованих по рядках та стовпчиках. Або іншими словами, растрове зображення - це двовимірний масив даних (пікселів). Більшість графічних редакторів орієнтовані не стільки на створення зображень, скільки на їхню обробку.

Найбільш часто використовуються растр, елементами якого є квадрати; такий растр називається квадратним (прямокутним).

Як растровий елемент можна використовувати фігури іншої форми, але вони повинні відповідати наступним вимогам:

- всі фігури повинні бути однакові;
- повинні повністю покривати площину без «наїжджання» та пробілів.

Так, наприклад, як растровий елемент можна використати правильний трикутник, правильний шестикутник (гексаедр). Можна будувати растри, використовуючи неправильні багатокутники, але практичного розповсюдження такі растри не знайшли. Якщо розглянути приклади побудови відрізку в прямокутному і гексагональному растрі, то при квадратному поданні растру побудова прямої може здійснюватися двома способами:

1) лінія, що утворена із растрів, які мають вісім напрямків для з'єднання, може знаходитися в одному з восьми можливих (див. рис. 12,а) положень. Недолік - занадто тонка лінія при куті 45°.

2) для чотирьохзв'язної лінії сусідні пікселі лінії можуть знаходиться в одному з чотирьох можливих (див. рис. 12,б) положень. Недолік такого способу - надто товста лінія при куті 45°.

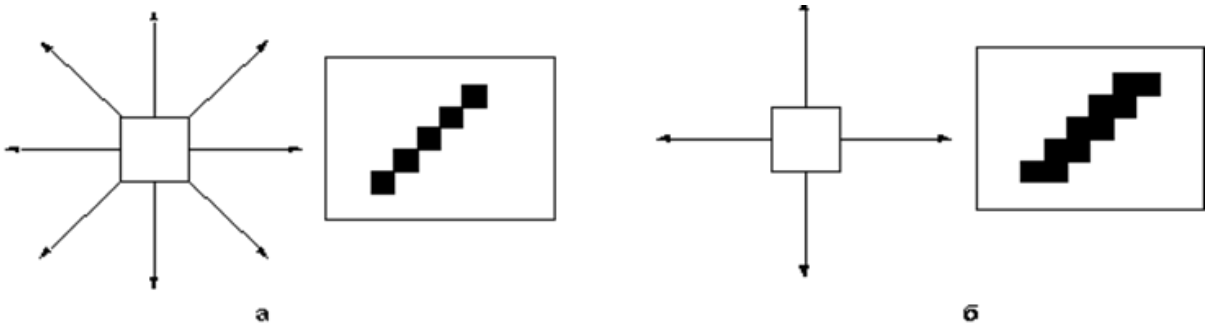


Рисунок 12 - Побудова лінії у прямокутному растрі

В гексагональному растрі лінії будуються із растру, що має шість напрямків зв’язаності (див. рис. 13); такі лінії більш стабільні по ширині, тобто дисперсія (коливання) ширини лінії менше, ніж в квадратному растрі.

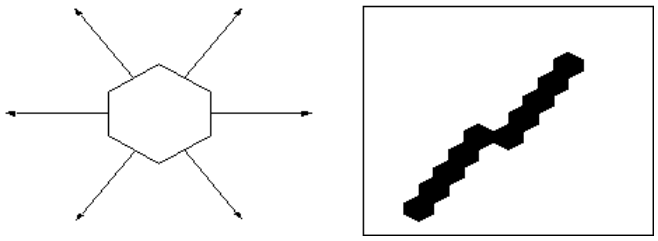


Рисунок 13 - Побудова лінії у прямокутному растрі

7. Параметри растрових зображень

До параметрів растрових зображень можна віднести наступні характеристики растрового зображення :

- кількість пікселів (розмір зображення в екранних пікселях) — зазвичай вказують розмір в пікселях по ширині та висоті (наприклад, 1024 × 768, 1920 × 1080);

- об’єм пам’яті в бітах, що використовуються для одного пікселя або глибина кольору ((англ. Color Depth, Bits per Pixel) або кількість використовуваних кольорів) - характеризує максимальне число кольорів, які використані у зображенні. В свою чергу, існує декілька типів зображень із різною глибиною кольору: чорно-білі, у відтінках сірого, повноколірні;

- колірна схема (для кольорових зображень) — RGB, CMY та інші, що відрізняються за глибиною кольорів і способом математичного опису кольорів;

- роздільна здатність, роздільність (resolution) - це ступінь деталізації зображення, кількість пікселів (точок), що відводяться на одиницю площі (довжини). Тому, як правило, термін роздільна здатність зображення вживають тільки стосовно до якого-небудь пристрою вводу або виводу зображення. Оскільки зображення можна розглядати стосовно до різних пристроїв, то слід розрізняти роздільну здатність оригіналу (цифрового зображення) та роздільну здатність екранного зображення.

Роздільна здатність оригіналу використовується при введенні зображення в комп’ютер і вимірюється в точках на дюйм (dots per inch - dpi). Встановлення роздільної здатності оригіналу залежить від вимог, що пред’являються до якості зображення, та впливають на розмір файлу. В загальному випадку діє правило: чим вище вимоги до якості, тим вище повинна бути роздільна здатність введення оригіналу. Наприклад, якщо

розглянути звичайну фотографію на твердому носії, то не можна сказати нічого про її роздільну здатність. Але після оцифровки через сканер необхідно буде визначити роздільну здатність оригіналу, тобто вказати кількість точок, зчитуваних сканером з одного квадратного дюйма.

Щодо роздільної здатності екранного зображення, то розмір растрового елемента (можна розглядати піксель) залежить від встановленої роздільної здатності екрана (з діапазону стандартних значень), а також роздільної здатності оригіналу та масштабу відображення. Для визначення роздільної здатності екранного зображення, крім dpi, використовують ppi (pixel per inch).

8. Переваги растрової графіки

Основною перевагою растрових зображень є можливість передавати величезну кількість відтінків кольорів та плавних переходів між ними. Тому растрове зображення має переваги при роботі з фотореалістичними об’єктами, наприклад сценами природи або фотографіями людей. Лише якісні растрові зображення здатні передати всю гаму процесів і явищ, що сприймаються зором людини. Формати файлів, призначені для збереження растрових зображень, є стандартними, тому немає вирішального значення, в якому графічному редакторі створено те чи інше зображення. Простота, і як наслідок, технічна реалізованість введення і виведення образотворчої інформації.

Підсумовуючи зазначене можна виділити такі *переваги*:

- растрова графіка дозволяє створити практично будь-яке зображення, незалежно від складності;

- простота автоматизованого вводу (оцифрування) зображень, фотографій, слайдів, малюнків за допомогою сканерів, відеокамер, цифрових фотоапаратів;

- фотореалістичність.

Недоліки растрової графіки:

- файли растрових зображень як правило мають дуже великий об’єм;

- при спробі повернути на невеликий кут зображення, наприклад, з чіткими тонкими вертикальними лініями, чіткі лінії перетворюються на чіткі "сходинки" (aliasing).

9. Векторна графіка

Векторна графіка характеризується розкладанням зображення в ряд, так званих, графічних примітивів - точок, прямих, ламаних, дуг, полігонів (див. рис.14). Оскільки на зображенні легко можна виділити множини простих об’єктів - відрізків прямих, ламаних, еліпсів, замкнутих кривих, то зберігаються не всі точки зображення, а координати вузлів примітивів та їх властивості (колір, зв’язок з іншими вузлами і т. і.).

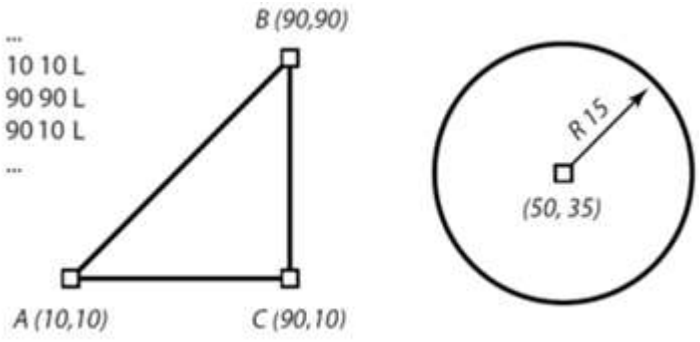


Рисунок 14 - Примітиви векторного зображення

10. Класифікація алгоритмів комп'ютерної графіки

Алгоритми комп'ютерної графіки можна розділити на два рівні: *нижній і верхній*.

Група алгоритмів нижнього рівня призначена для реалізації графічних примітивів: відрізків, кола, зафарбування областей і т.п. Такі алгоритми представлено в графічних бібліотеках мов високого рівня або реалізовано апаратно в графічних прискорювачах.

Серед алгоритмів нижнього рівня можна виділити наступні групи:

I група – алгоритми, що використовують прості математичні методи і відрізняються простотою реалізації. Як правило, такі алгоритми не є найкращими за об'ємом виконуваних обчислень або необхідних ресурсів пам'яті. Мають достатньо низьку ефективність, що стосується швидкодії.

II група – алгоритми, що використовують більш складні математичні методи в порівнянні з алгоритмами I групи (але часто і евристичні), і відрізняються більшою ефективністю.

III група – алгоритми, що оптимізовані на апаратну реалізацію (допускають розпаралелювання, рекурсивні, реалізовані в найпростіших командах). У цю групу можуть потрапити і алгоритми, представлені в перших двох групах.

IV група – алгоритми спецпризначення (наприклад, для усунення ефекту сходинок).

До алгоритмів верхнього рівня відносяться в першу чергу алгоритми видалення невидимих ліній і поверхонь. Задача видалення невидимих ліній і поверхонь є центральною в машинній графіці. Від ефективності алгоритмів, що дозволяють вирішити цю задачу, залежать якість і швидкість побудови 3-D зображення. З цією задачею також межує задача побудови реалістичних зображень – обробка явищ, що пов'язані із розташуванням, кількістю і

характером джерел світла, обробкою даних для створення відображення поверхні тіла (прозорість, заломлення та відбиття світла) тощо.

При цьому алгоритми верхнього рівня забезпечуються примітивами, що реалізуються алгоритмами нижнього рівня.

Для різних областей застосування комп'ютерної графіки в першу чергу можуть використовуватися різні властивості алгоритмів КГ. Для наукової графіки велике значення має універсальність алгоритму, його застосовність для широкого класу задач. Для цих випадків швидкодія може відходити на другий план. Для систем моделювання, що відтворюють рух об'єктів, швидкодія стає головним критерієм, оскільки потрібно генерувати зображення в реальному часі.

Оскільки зображення складається із набору точок і подається матрицею дискретних елементів, що мають конкретні фізичні розміри, то такі особливості растрової графіки обумовлюють проблеми, серед яких, наприклад, неможливість провести ідеальну (гладку) лінію з однієї точки екрану в іншу; можна виконати лише апроксимацію цієї лінії – наближене відтворення одних об'єктів за допомогою інших, простіших, з її відображенням на дискретній площині (растровій площині або растрі). Ця растрова площина являє собою квадратну сітку з кроком 1.

Визначення. Відображення будь-якого об'єкта на дискретну площину називається розкладанням його в растр або растровим представленням.

Сформувати растрове зображення можна по-різному.

1-й спосіб. Для того щоб створити зображення на растровому моніторі, можна скопіювати готовий растр у відеопам'ять. Цей растр може бути отриманий, наприклад, за допомогою сканера або цифрового фотоапарата.

2-й спосіб. Можна створювати зображення об'єкта шляхом послідовного відтворення окремих простих елементів.

Прості елементи, що складають складний об’єкт, називаються графічними примітивами. Найпростішими і в той же час найбільш універсальними примітивами растрової графіки є пікселі.

11.Растрові алгоритми генерування графічних примітивів

Більшість графічних бібліотек містять досить багато простих растрових алгоритмів. На рис. 15 наведено систему растрових алгоритмів.

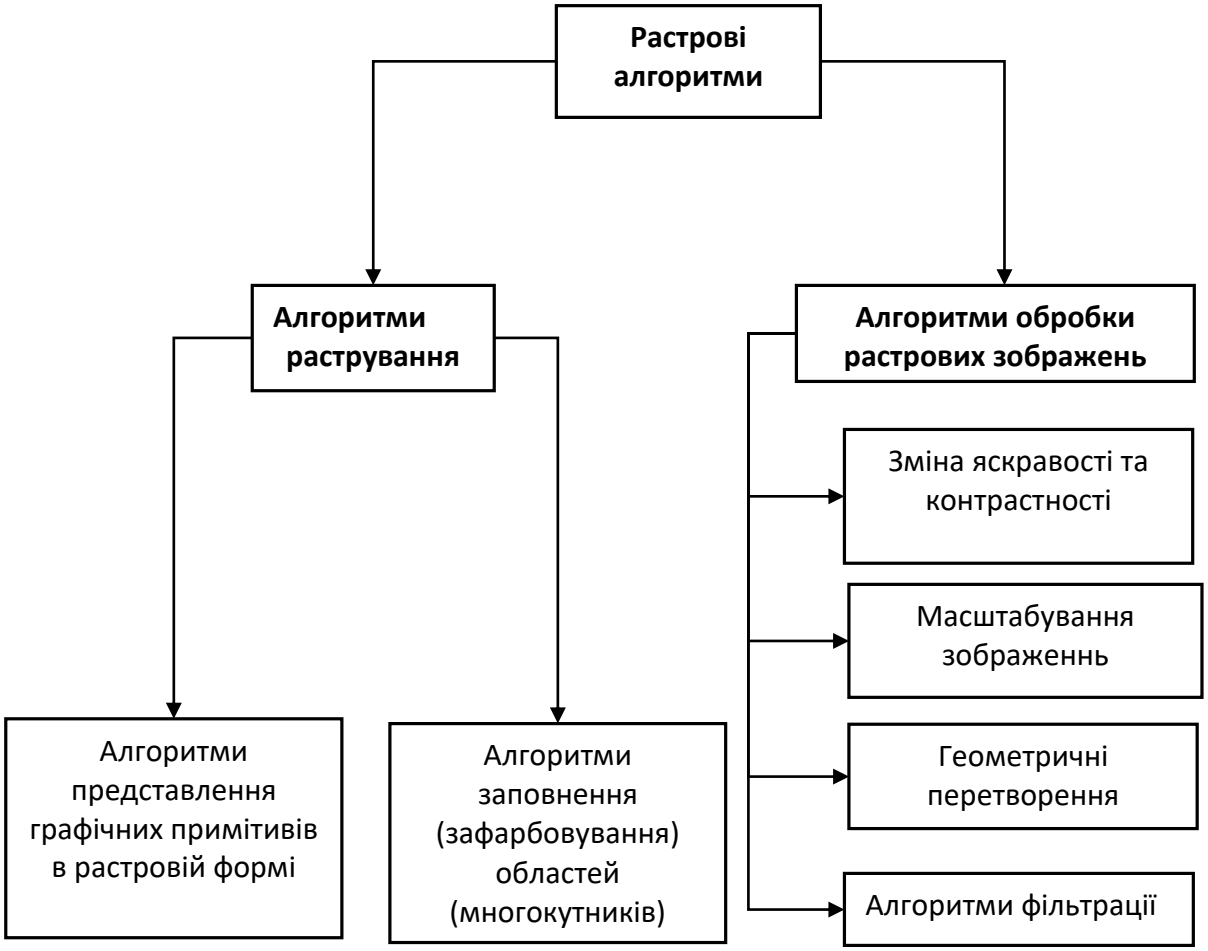


Рисунок 15 – Класифікація растрових алгоритмів

12.Алгоритми генерування відрізка прямої лінії

Перш ніж перейти до безпосереднього розгляду переведення математичного опису об'єкта (лінії, кривої та ін.) в растрову форму, потрібно розгялнути поняття зв'язності.

Зв'язність - можливість з'єднання двох пікселів растровою лінією, тобто послідовним набором пікселів. Два елементи (пікселі) з *координатами* (x_1, y_1) *і* (x_2, y_2) можна вважати сусідніми (рис. 16 та рис. 17), якщо:

- їхні x-координати або y-координати відрізняються на одиницю – 4-зв'язність ($|x_1 - x_2| + |y_1 - y_2| \le 1$);

- їхні x-координати і y-координати відрізняються не більше ніж на одиницю – 8-зв'язність ($|x_1 - x_2| \le 1$, $|y_1 - y_2| \le 1$).

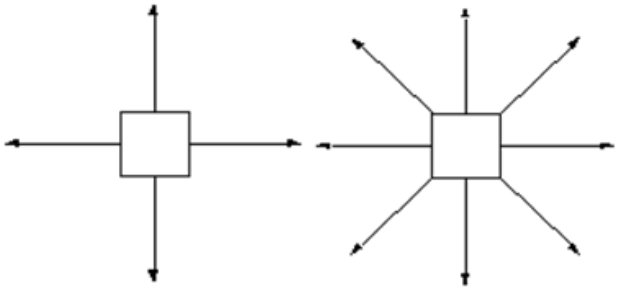


Рисунок 16 – 4-х зв'язність та 8-ми зв'язність

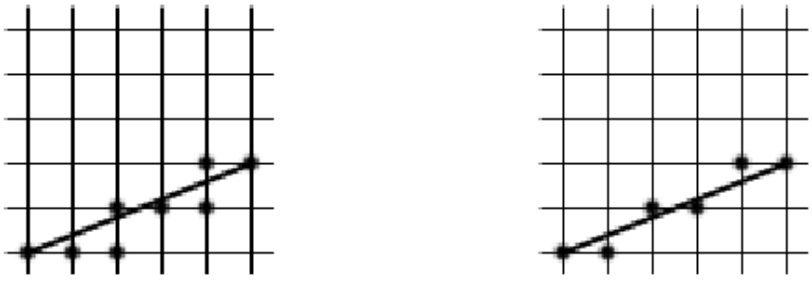


Рисунок 17 – Приклад побудови лінії для 4-зв'язних та 8-зв'язних пікселів

Загальні вимоги до зображення відрізка:

- кінці відрізка повинні знаходитися в заданих точках;
- відрізки повинні виглядати прямими (тобто рівними);
- яскравість уздовж відрізка повинна бути постійною і не залежати від довжини і нахилу.

Через те, що зображення складається з елементів (пікселів) кінцевого розміру, жодна з перерахованих вище умов не може бути точно виконана на растровому моніторі через існування таких проблем та особливостей:

- кінці відрізків зазвичай розташовуються в пікселях, близьких до бажаного положення. У деяких випадках координати кінців відрізка можуть точно збігатися з координатами пікселів;
- відрізок апроксимується набором пікселів;
- лише в окремих випадках для вертикальних, горизонтальних і відрізків під кутом 45^{o} лінії будуть виглядати прямими;
- відрізок виглядає прямим без сходинок тільки в разі вертикальних і горизонтальних ліній;
- інтенсивність варіюється по відношенню до різних відрізків, наприклад, відстань між центром пікселя вертикального сегмента і сегментом під кутом 45^{o} різна;
- при будь-якому іншому розташуванні вибрати потрібні пікселі важче, що показано на рис. 18.

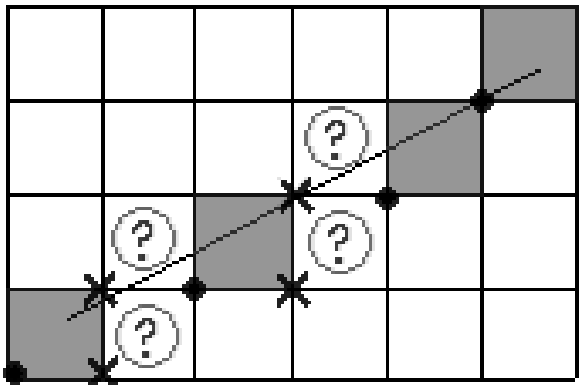


Рисунок 18 – Проблема вибору потрібних позицій пікселів при растуванні відрізка

Щоб подолати ці недоліки, можна збільшити роздільну здатність і об’єктивно поліпшити апроксимацію відрізка. І навпаки, можна зменшити роздільну здатність, щоб суб’єктивно покращити цю апроксимацію. Інші методи суб’єктивного поліпшення якості апроксимації засновані на програмних методах, таких як "розмиття" чітких меж зображення.

13.Задача растрування відрізку

Найпростіший алгоритм полягає в наступному. Розглянемо задачу побудови растрового зображення відрізка, що з'єднує точки $A(x_1, y_1)$ і $B(x_2, y_2)$. Для спрощення будемо вважати, що $0 \le y_2 - y_1 \le x_2 - x_1$. Тоді відрізок описується рівнянням:

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1), x \in [x_1, x_2], \text{ або } y = kx + b,$$

де

$$k = \frac{y_2 - y_1}{x_2 - x_1},$$

$$b = y_1 - kx_1$$

Отримуємо найпростіший (наївний) алгоритм растрового представлення відрізку:

```
void line(int x1, int y1, int x2, int y2, int color)
{
 double k = ((double)(y2 – y1)) / (x2 – x1);
double b = y1 – k * x1;
for (int x = x1; x <= x2; x++)
putpixel(x, (int)(k * x + b), color);
```

Але обчислення значень функції $y = kx + b$ можна уникнути, використовуючи в циклі такі співвідношення між елементами послідовності, в якій наступний елемент виражається через кілька попередніх, тобто при зміні x на 1 значення y змінюється на k. Аналогічно отримуємо алгоритм:

```
void line(int x1, int y1, int x2, int y2, int color)
{
 double k = ((double)(y2 – y1)) / (x2 – x1);
double y = y1;
for (int x = x1; x <= x2; x++, y += k)
```

putpixel(x, (int)y, color);

Наведений простий покроковий алгоритм побудови відрізків має деякі недоліки:

- операція виконується з дійсними змінними, краще використовувати цілочисельну арифметику;
- операція округлення виконується на кожному кроці, що також знижує продуктивність;
- якщо додається приріст для обчислення координат, то помилки обчислення можуть накопичуватися.

Параметричний алгоритм (рис. 19) є найпростішим способом створення лінії. Однак, у цього методу є істотні недоліки, а саме:

- необхідність виконання операцій над дійсними числами;

- використання операції ділення, що значно ускладнює апаратну організацію та збільшує час роботи алгоритму.

Розглянемо задачу, в якій необхідно провести лінію з точки (x_1, y_1) в точку (x_2, y_2) з урахуванням того, що задані інтенсивності (яскравості) в початковій (V_1) і кінцевих (V_2) точках лінії. Тобто необхідно виконати лінійну інтерполяцію за інтенсивністю (або яскравістю). Інтерполяція зображень працює в двох вимірах і, так би мовити, «намагається» досягти найкращого наближення кольору та яскравості пікселів на основі значень цих характеристик у сусідніх пікселів.

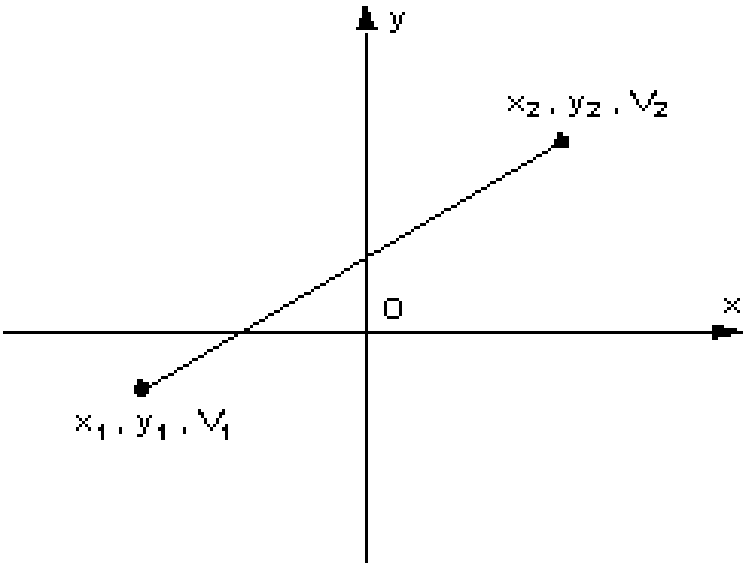


Рисунок 19 – Приклад для параметричного алгоритму

Тоді будь-яку точку на цій лінії можна представити у вигляді :

$$x_{i+1} =]x_i + t \cdot (x_2 - x_1)[$$
$$y_{i+1} =]y_i + t \cdot (y_2 - y_1)[$$
$$V_{i+1} =]V_i + t \cdot (V_2 - V_1)[$$

де *][* – знак округлення до цілого, а $t \in [0;1]$.

Оптимальний крок приросту по t вибирається як

$$t = \frac{1}{\max\{|x_2 - x_1|, |y_2 - y_1|\}} = \frac{1}{N-1};$$

де *N* – довжина лінії в пікселях.

Можна проводити обчислення через приріст координат по *x, y, V* :

$$\Delta x = \frac{x_2 - x_1}{N-1}; \Delta y = \frac{y_2 - y_1}{N-1}; \Delta V = \frac{V_2 - V_1}{N-1}$$

Особливістю є те, що значення приросту обчислюються один раз на початку і не входять в основний цикл побудови лінії на екрані. За рахунок цього підвищується швидкодія. Переваги алгоритму:

- простота програмної реалізації;
- простота реалізації лінійної інтерполяції по інтенсивності (яскравості) (у разі задання кольору точок кінців відрізку, лінійна інтерполяція кожної колірної компоненти проводиться окремо).

14.Інкрементні алгоритми

Найбільшого поширення в комп'ютерній графіці отримали інкрементні алгоритми генерування відрізків. У цих алгоритмах відрізок малюється послідовно піксель за пікселем. Координати наступного пікселя визначаються виходячи з координат поточного і деякої додаткової інформації. Розглядаються алгоритми:

- алгоритм ЦДА - цифрового диференціального аналізатора (DDA - Digital Differential Analyzer) для генерування векторів;

- алгоритм Брезенхема для генерування векторів.

Розглянемо генератор векторів, що працює на принципі ЦДА - цифрового диференціального аналізатора. В основу роботи цього алгоритму покладено той факт, що похідна функції, що представляє пряму лінію на площині, є величиною постійною, тобто

$$\frac{dy}{dx} = const$$

Один з методів розкладання відрізка в растр полягає у розв'язанні диференціального рівняння, що описує цей процес. Для прямої, що проведена через 2 відомі точки (x_1, y_1) та (x_2, y_2) маємо:

$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{y - y_1}{x - x_1}$$

Розрізняють 2 варіанта алгоритма ЦДА: звичайний та несиметричний.

У звичайному ЦДА спочатку задається кількість кроків N, для апроксимації відрізка (де більший із приростів вибирається як одиниця растру):

$$N = max\,(|y_2 - y_1|, |x_2 - x_1|)$$

Потім в циклі за N кроків обчислюються координати чергових вузлів (звичайний ЦДА в першому квадранті). Розв'язок виглядає як:

$$\begin{cases} x_{i+1} = x_i + \dfrac{P_x}{N} \\ y_{i+1} = y_i + \dfrac{P_y}{N} \end{cases}$$

де координати (x_1, y_1) та (x_2, y_2) - кінці відрізку, а y_i - початкове значення для чергового кроку вздовж відрізка.

Отримані поточні значення *x* і *y* перетворюються в цілочисельні значення координати чергового пікселя, який або округлюється, або відкидається дробова частина. Недоліками даного варіанту алгоритму є:

- виконується обробка дійсних чисел (числа з плаваючою точкою);

- точки можуть прописуватися двічі, що збільшує час побудови лінії;

- виконується округлення;

- оскільки обидві координати обчислюються незалежно, то немає пріоритетних напрямків для наступної точки, і побудовані відрізки виглядають не дуже красиво.

Існує модифікація цього алгоритму – цілочисельний варіант.

В несиметричному ЦДА використовується різний крок по осях X та Y. Визначається, що більше: P_x *чи* P_y.

Нехай $P_x > P_y$, тоді якщо *x* збільшується на одиницю P_x разів, то *y* має збільшуватись теж P_x разів, але на величину $\frac{Py}{Px}$.

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = y_i + \dfrac{P_y}{P_x} \end{cases}$$

В алгоритмі Брезенхема запропоновано підхід, що дозволяє реалізувати алгоритми растеризації, які відносяться до інкрементних алгоритмів. Основна ідея полягає в тому, що цей підхід не передбачає виконання операції ділення,

як в алгоритмі несиметричного ЦДА, проте забезпечує мінімізацію відхилення згенерованого образу від істинного відрізка, як в алгоритмі звичайного ЦДА.

Основною метою для розробки таких алгоритмів є побудова циклів обчислення координат на основі тільки цілочисельних операцій додавання/віднімання без використання множення і ділення. Інкрементні алгоритми виконуються як послідовне обчислення координат сусідніх пікселів шляхом додавання/збільшень координат та використання тільки цілочисельних змінних.

Алгоритм вибирає (обчислює) оптимальні растрові координати (рис. 20) для подання відрізка. У процесі роботи одна з координат - *x* або *y* (залежно від кутового коефіцієнта $\frac{Py}{Px}$) - змінюється на одиницю, інша - або залишається у своєму старому значенні, або, так само, змінюється на одиницю. Вибір між цими двома варіантами залежить від того, який із них забезпечує меншу похибку (*E*).

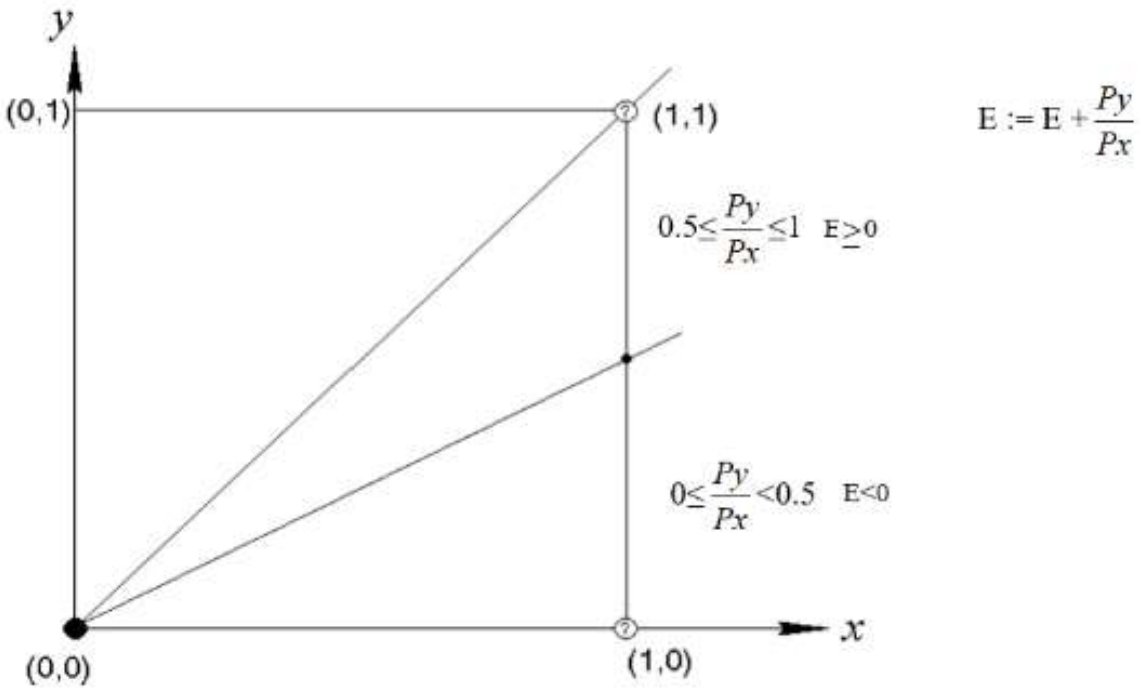


Рисунок 20 - Кутовий коефіцієнт в алгоритмі Брезенхема

Для прийняття рішення, куди виводити наступний піксель, обчислюється величина відхилення *E* - похибки точної позиції від середини між двома можливими позиціями растрової точки в напрямку найменшої відносної координати. Знак *E* використовується як критерій для вибору найближчої растрової точки. На першому кроці похибка ініціалізується в -0.5. Далі похибка обчислюється як $E := E + \frac{Py}{Px}$.

Якщо *E<0*, то точне *Y*-значення округлюється до останнього меншого цілочисельного значення *Y*, тобто *Y*-координата не змінюється в порівнянні з попередньою точкою. В іншому випадку *Y* збільшується на 1. Якщо похибка *(E)* перевищує 0,5, то *Y*-координата зменшується на 1 та віднімається від похибки 1.

Даний варіант алгоритму вимагає використання арифметики із дійсними числами, проте можна обійтися тільки цілочисельною. Для цього Брезенхем помножив всі змінні алгоритму на $2P_x$, *де* $P_x = x_2 - x_1$. Алгоритм не змінився за винятком кількох замін. Множення на 2 реалізовано зсувом вліво.

```
void   line ( int x1, int y1, int x2, int y2, int color )
{
int    Px = x2 – x1;
int    Py = y2 – y1;
int    d1 = Py << 1;
int    d2 = ( Py - Px ) << 1;
int    E  = d1 - Px;
int    y = y1;

putpixel ( x1, y1, color );
```

```
for ( int x = x1; x <= x2; x++ )
{
if (E  > 0 )
 {
  E  += d2;
  y ++;
 }
else
E += d1;
putpixel ( x, y, color );
}                                                                    }
```

Наглядний приклад роботи алгоритму Брезенхема:

Для спрощення обчислення *E* будемо вважати, що розглянутий вектор починається в точці (0,0) і проходить через точку (4, 1.5) (рис. 21), тобто має додатній нахил, менший 1.

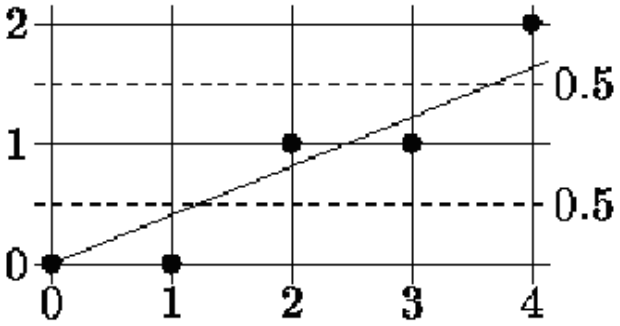


Рисунок 21 - Приклад роботи алгоритму

Із рис. 21 видно, що відхилення (похибка) *E* для першого кроку:

$$E_1 = P_y/P_x - 1/2 < 0,$$

тому для занесення пікселя вибирається точка (1,0).

Відхилення *E* для другого кроку обчислюється додаванням прирощення *Y*-координати для наступної *X*-позиції (рис. 21):

$$E_2 = E_1 + Py/Px > 0,$$

тому для виведення пікселя вибирається точка (2,1). Оскільки відхилення рахується від *Y*-координати, яка тепер збільшилася на 1, то з накопиченого відхилення *E* для обчислення наступних відхилень треба відняти 1:

$$E_2 = E2 - 1.$$

Відхилення *E* для третього кроку роботи алгоритму буде дорівнювати:

$$E_3 = E_2 + P_y/P_x < 0,$$

І тому для виведення пікселя на екран вибирається точка із координатою (3,1).

Існують модифікації алгоритму Брезенхема для 4-зв’язної та 8- зв’язної ліній.

15.Алгоритм Ву (Wu Xiaolin)

Щойно розглянуті алгоритми Брезенхема розкладання векторів у растр мають загальний недолік: вони генерують відрізки з нерівними різкими краями. Для подолання цього недоліку Wu Xiaolin (Ву Сяолінь) створив алгоритм, який генерує згладжений відрізок. Алгоритм Ву заснований на використанні напівтонів. Алгоритми, що було розглянуто вище, відтворювали відрізки одним тоном (кольором). Алгоритм Ву зафарбовує різні ділянки відрізка в різний тон і за рахунок цього "згладжує" нерівності.

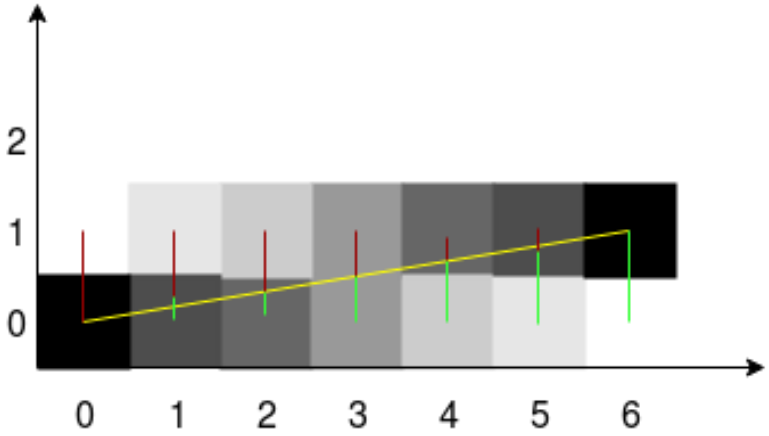


Рисунок 22 - Приклад роботи алгоритму

На практиці при відтворенні руху відрізку, який згенеровано алгоритмом Брезенхема, він переміщується різко, стрибками. Відрізок за алгоритмом Ву буде переміщуватись плавно. Алгоритм Ву використовує механізми згладжування при растеризації лінії. При цьому ступінчасті виступи на лінії стають менш помітні. Цей ефект досягається за рахунок наступних дій:

- алгоритм оперує не з цілими координатами кінців відрізка, а з дійсними числами, що представляють координати;
- на першому кроці для точки, що знаходиться на лінії, обчислюються координати двох найближчих точок растру;

- далі, на наступному кроці, між цими двома точками розподіляється (рис. 22) прозорість кольору пікселя, яка пропорційна „відстані" пікселів до лінії та з умовою, щоб сумарна яскравість дорівнювала 1. При такому розподілі людське око сприймає послідовність пікселів із взаємодоповнюючими значеннями прозорості як безперервну згладжену лінію.

Значення параметра, рівне 1, відповідає повністю промальованому непрозорому пікселю (наприклад, білому, якщо ми малюємо білим по чорному). Значення параметра, рівне 0, відповідає цілком прозорим пікселям, тобто нічого не малюється. Проміжні значення відповідають різним відтінкам сірого, або ж, у випадку відтворення кольорових пікселів, визначаються по черзі для кожної з трьох компонент R, G і B.

16.Алгоритми растрування кола

Існує декілька дуже простих, але неефективних способів перетворення кіл в растрову форму. Наприклад розглянемо для простоти коло з центром на початку координат. Його рівняння записується як

$$x^2 + y^2 = R^2.$$

Розв'язуючи це рівняння для y, отримаємо

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

Щоб показати четверту частина кола, будемо змінювати *x* з одиничним кроком від 0 до *R* і на кожному кроці обчислювати *y*.

Такий тривіальний спосіб має недоліки:

- занадто складні обчислення – операція добування квадратного кореня;

- пікселі нерівномірно розподіляються по дузі кола (нерівномірність товщини дуги) (див. рис.23).

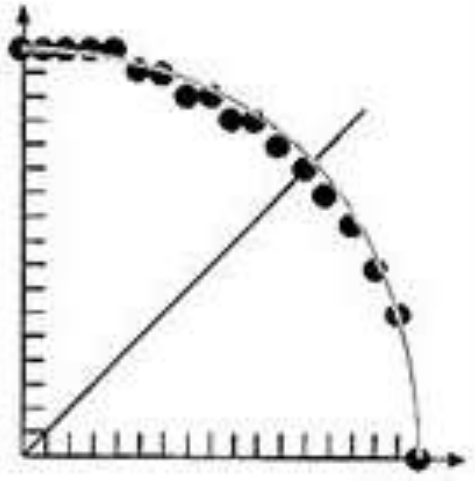


Рисунок 23 - Нерівномірність формування дуги кола

Другого недоліку можна позбутися, якщо використовувати представлення кола в полярних координатах:

$$x = R \cos\theta$$
$$x = R \sin\theta$$

Тоді, пробігаючи по всіх можливих значеннях θ із заданим кроком, для кожного з них можна визначити координати відповідного йому пікселя, і зафарбувати його. На цей раз, оскільки щоразу повертається радіус на один і той же кут, то пікселі будуть рівномірно розподілені по колу. Однак, як і в попередньому алгоритмі, обчислення *cos i sin* не є бажаним.

Завдання можна значно спростити, якщо використовувати те, що коло є центрально-симетричною фігурою, а значить, якщо побудувати, наприклад, 1/8 її частину, а потім за допомогою перетворення симетрії, то можна добудувати коло повністю (рис. 24).

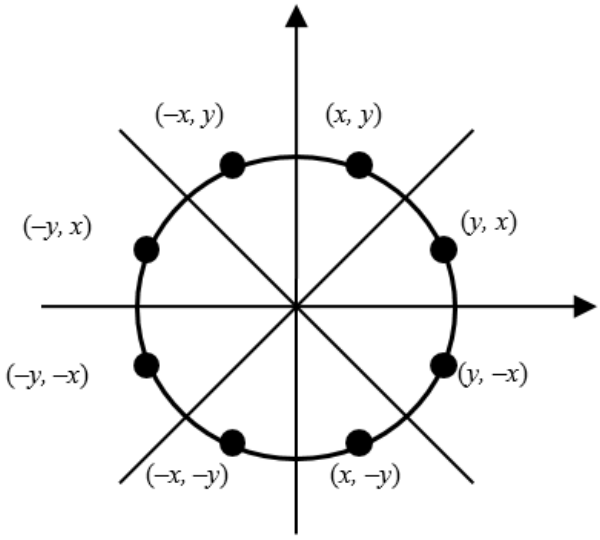


Рисунок 24 - Знаходження точок, що симетричні поточній

Якщо точка *(x, y)* лежить на окружності, то легко обчислити сім точок, що належать окружності, симетричних цій точці.

Для спрощеного розгляду алгоритма Брезенхема для кола розглянемо першу чверть кола з центром на початку координат (рис. 24). Зауважимо, що можливі варіанти роботи алгоритму: при генеруванні кола за годинниковою стрілкою в першому квадраті в точці *x = 0, y = R,* то *y* є монотонно спадаючою функцією аргументу *x*. Аналогічно, якщо вихідною точкою є *y = 0, x = R*, то при

генеруванні окружності проти годинникової стрілки *x* буде монотонно спадаючою функцією аргументу *y*. Розглянемо растрування генерування за годинниковою стрілкою з початком в точці *x = 0, y = R*. Передбачається, що центр кола і початкова точка перебувають точно в точках растра.

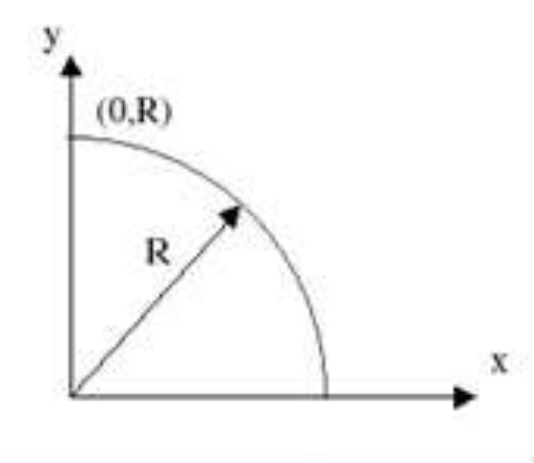


Рисунок 24 - Коло в першому квадранті

Сама ідея алгоритму наступна. Для будь-якої заданої точки кола існує тільки три можливості вибрати наступний піксель, що найкращим чином наближує цю послідовність до кола: горизонтально вправо, по діагоналі вниз і вправо, вертикально вниз. На рис.25 ці напрямки позначені відповідно m_H, m_D, m_V.

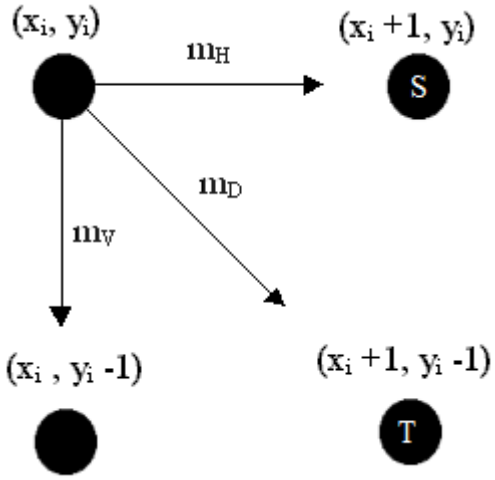


Рисунок 25 - Можливі варіанти вибору наступного пікселя

Алгоритм вибирає той піксель, для якого квадрат відстані між ним та окружністю є мінімальним, тобто, мінімум з

$$m_H = |(x_i + 1)^2 + (y_i)^2 - R^2|$$
$$m_D = |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$
$$m_V = |(x_i)^2 + (y_i - 1)^2 - R^2|$$

Розглянувши варіанти можливих позицій наступних точок згідно цього алгоритма, в цій ситуації слід вибрати або піксель $S(x_i + 1, y_i)$, тобто m_H, або піксель $T(x_i + 1, y_i - 1)$, тобто m_D . Різниця помилок (похибок) або квадратів відстаней від кола до пікселів в горизонтальному (S) і діагональному (T) напрямках тут визначається як

$$\Delta = |(x_i + 1)^2 + (y_i)^2 - R^2| - |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$

При $\Delta < 0$ відстань від окружності до діагонального пікселя більша, ніж до горизонтального. Навпаки, якщо $\Delta > 0$, відстань до горизонтального пікселя більша. Таким чином, при виборі наступної точки за (x_i, y_i), варто керуватися таким критерієм:

при $\Delta <= 0$ слід обрати точку $S(x_i + 1, y_i)$

при $\Delta > 0$ слід обрати точку $T(x_i + 1, y_i - 1)$

Опускаючи деякі перетворення, скорочення та спрощення, можна записати похибку для першого кроку:

$$\Delta_1 = 3 - 2R.$$

Обчислення похибки на наступному кроці:

- Якщо попередньою точкою була точка S, то похибка

  $\Delta_{i+1} = \Delta_i + 4x_i + 6;$

- Якщо попередньою точкою була точка T, то похибка

  $\Delta_{i+1} = \Delta_i + 4(x_i - y_i) + 10.$

Перевагою даної варіації алгоритму є його цілочисельність і простота апаратної реалізації. До недоліків слід віднести необхідність задання координат центру кола та її радіусу цілими числами.

Якщо згенеровано другий октант (від 45 ° до 90° проти годинникової стрілки), то перший октант можна отримати дзеркальним відображенням відносно прямої *y = x*, що дає в сукупності перший квадрант. Перший квадрант відображається відносно прямої *x = 0* для отримання відповідної частини кола у другому квадранті. Верхнє півколо відображується відносно прямої *y = 0*.

Алгоритм растрування еліпсу шляхом стиснення кола є комбінацією алгоритмів Брезенхема для кола і для відрізка (рис.26).

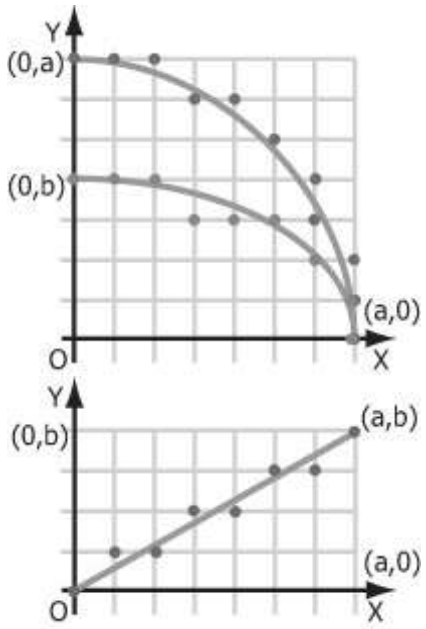


Рисунок 26 - Растрування еліпсу

Растрування за цим алгоритмом бере початок з точки *(a, 0)* на колі та з точки *(0, 0)* на відрізку. Будемо будувати еліпс точно так само, як коло, але зміщати поточну точку по *y* тільки в тому випадку, коли такий зсув відбувається в поточному кроці для відрізка (паралельно будуємо і відрізок з точки *(0,0)* до

(a,b)). Тобто побудова відрізку саме і є реалізацією дискретної апроксимації еліпсу шляхом стиснення кола по осі *y* в *a/b* раз.

17.Методи покращення зображення

При растуванні простих примітивів (особливо для зображень, які мають невисоку роздільну здатність) виникає проблема сходинкового ефекту (ступінчатість, aliasing). Цей ефект помітний для ліній, що проведено під кутом – при великому кроці решітки растра пікселі утворюють так звані сходинки.

Особливо сходинковий ефект проявляє себе при:

- при візуалізації дрібних деталей;
- при відтворенні (промальовуванні) ребер і границь;
- при анімації дрібних деталей.

Для отримання згладжених растрових зображень можна використовувати алгоритми для генерування згладжених зображень окремих простих об’єктів – відрізків, примітивів, як, наприклад, алгоритм Ву, або модифікувати існуючі алгоритми для усунення цього ефекту. Але тут є недолік: різні алгоритми можуть дати різні варіанти растрового подання (зображення) одного і того ж об’єкту. Інший шлях - це здійснювати обробку вже згенерованих зображень.

Один із методів усунення ступінчастості полягає у збільшенні так званої ефективної роздільної здатності. Для цього використовуються спеціальні алгоритми для отримання кольору пікселя за допомогою емуляції кольорів кількох пікселів навколо. Ці методи обчислюють значення кольору ніби всередині екранного пікселя в декількох точках. Ці точки називаються семплами (sample). Семпли являють собою ті самі додаткові пікселі зображення, які збільшують ефективну роздільну здатність. Значення цих семплів використовуються для обчислення кінцевого кольору пікселя.

Так от, для даного випадку метод усунення ступінчастості реалізується збільшенням частоти вибірки (збільшенням розміру растра, роздільної здатності) - так враховуються більш дрібні деталі. Створюється проміжний

буфер кадру, в якому по кожному із напрямків кількість пікселів збільшено в 2 та більше разів (рис.27). Іншими словами, обчислюється растр із збільшенням роздільної здатності. Побудова сцени проводиться в проміжному буфері з більшою роздільною здатністю. Після цього зображення переноситься в основний буфер із усередненням пікселей, тобто відбувається перенос растра в екранне зображення із зменшенням роздільної здатності. Підпікселі в цьому випадку розподіляються рівномірно і їхні атрибути враховуються однаково – це так зване рівномірне усереднення.

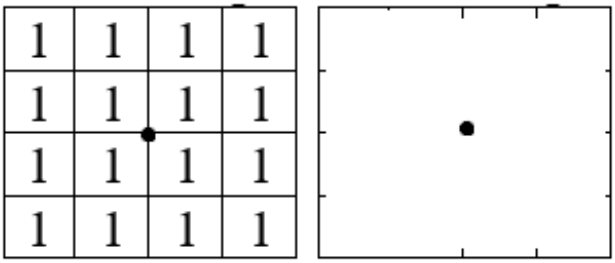


Рисунок 27 - Зміна роздільної здатності

Деякою мірою можна отримати кращі результати, якщо розглядати більшу кількість підпікселей і враховувати їх вплив за допомогою вагових коефіцієнтів при визначенні атрибутів „моніторних" пікселів. Такий спосіб представляє собою локальну фільтрацію (рис.28).

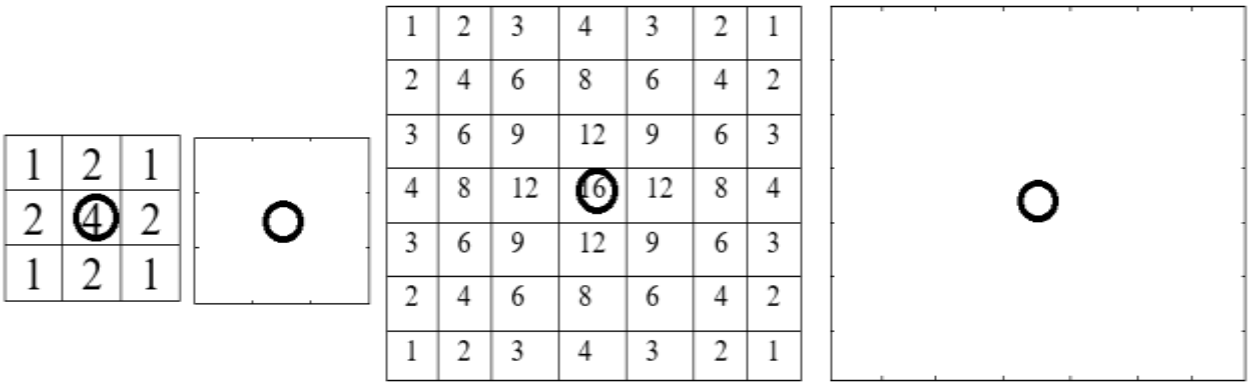


Рисунок 28 - Введення вагових коефіцієнтів

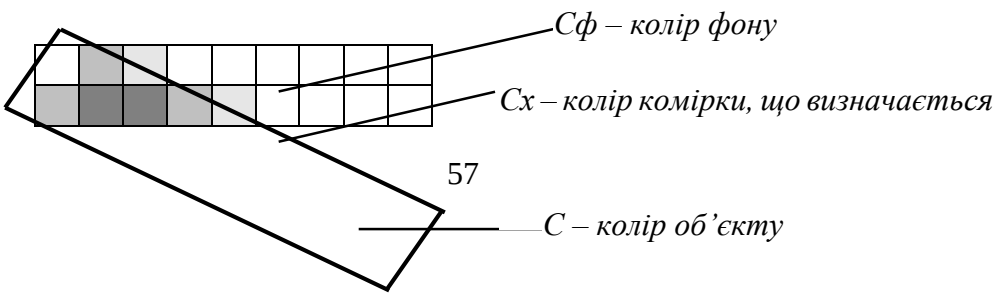
Зрозуміло, що якщо зображення чорно-біле, то усунути ефект ступінчастості растра практично неможливо. Але при наявності відтінків напівтонів можна задати інтенсивність кольору пікселя в залежності від площі його перетину з областю об'єкта.

Наближення напівтонами – це метод, задачею якого є використання мінімальної кількості рівнів інтенсивності для отримання більшої кількості напівтонів. Метод напівтонів базується на властивості зорової системи людини інтегрувати (усереднювати або згладжувати) дискретну інформацію. Ідея полягає в тому, що якщо достатньо близько розташувати точки з різними кольорами, то вони будуть сприйматися як точка з деяким усередненим кольором.

Для того, щоб растрове зображення виглядало більш згладженим, колір кутових пікселей або сходинок треба замінити на деякий проміжний відтінок, між кольором об'єкту і кольором фону (як в алгоритмі Ву). Алгоритм передбачає обчислення колірного відтінку пропорційно до частини площини комірки растру, який покритий ідеальним контуром об'єкту. Якщо площу всієї комірки позначити через *S*, а частину площі комірки, яка покрита ідеальним контуром (об'єктом) - S_x, то необхідний колір дорівнює (рис.29):

$$Cx = \frac{C \cdot Sx + C\phi \cdot (S - Sx)}{S}$$

де $C_ф$ – колір фону, C_x – колір пікселя (визначається при обчисленні), який визначається частиною S_x площі комірки, що накривається об'єктом, *C* - колір об'єкту, *S* - площа всієї комірки, S_x - частина площі комірки, що покривається ідеальним контуром (об'єктом).



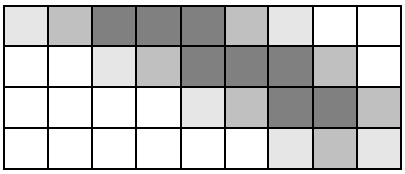


Рисунок 29 - Визначення проміжних відтінків

Отримати згладжені растрові зображення можна за допомогою локальної фільтрації. Вона здійснюється шляхом обчислення яскравості пікселей, розміщених в деякій області поточного пікселя, який розглядається.

Як приклад можна навести обробку зображення, яка полягає в тому, що по растру пересувається деяке вікно (рис.30).

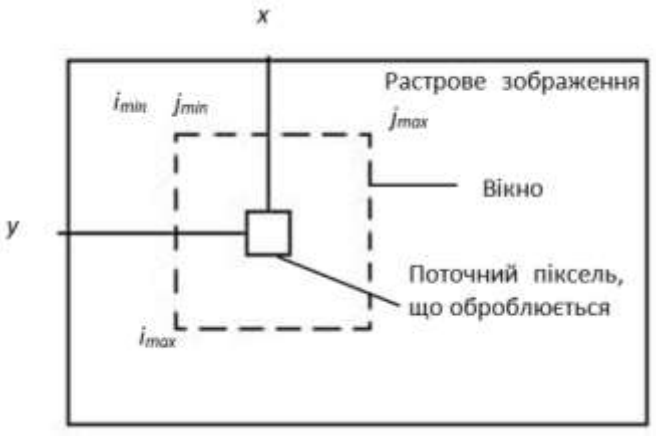


Рисунок 30 - Обробка поточного пікселя

Це вікно вихоплює пікселі, що використовуються для обчислення кольору деякого поточного пікселя. Якщо область симетрична, то даний піксель знаходиться в центрі вікна. Для обробки всього растру необхідно провести обчислення для кожного пікселя. Базову операцію такого фільтру можна представити як:

$$F_{x,y} = \frac{1}{K} \sum_{i=i_{\min}}^{i_{\max}} \sum_{j=j_{\min}}^{j_{\max}} P_{x+j,y+i} \cdot M_{i-i_{\min},j-j_{\max}}$$

де *P* – значення кольору поточного пікселя, *F* – нове значення кольору пікселя, *K* – нормуючий коефіцієнт, *M* – двовимірний масив коефіцієнтів, який визначає властивості фільтру (цей масив коефіцієнтів, що розташовуються відповідно пікселям вікна називають маска, матриця згортки, де розмірність задана непарними числами).

Розміри вікна фільтру: $(j_{max} – j_{min}+1)$ – по горизонталі та $(i_{max}–i_{min}+1)$ – по вертикалі. При i_{min}, j_{min}=-1 і i_{max}, j_{max} = 1 маємо фільтр з вікном *3x3*, який часто використовують на практиці (рис.31).

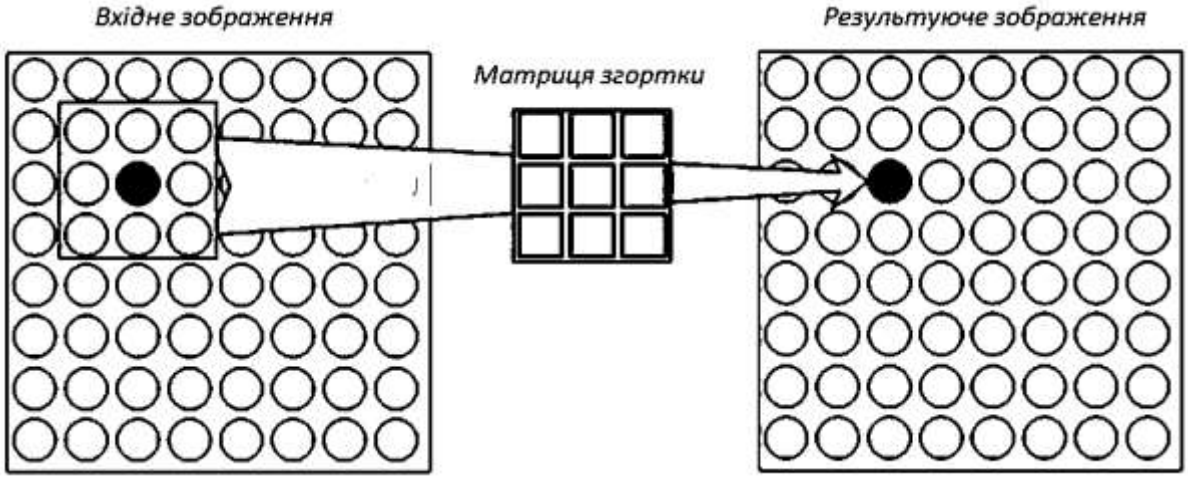


Рисунок 31 - Застосування фільтру з вікном *3x3*

Якщо в ході визначення значень кольорів пікселів ці нові значення заносяться у початковий (вхідний) растр та застосовуються при обчисленні кольорів решти пікселей, то таку фільтрацію називають рекурсивною.

При нерекурсивній фільтрації в обчисленнях використовуються тільки попередні (старі) значення кольорів пікселів. Нерекурсивність можна забезпечити, якщо нові значення записувати в окремий масив.

Рекурсивний фільтр може дати більший ефект порівняно з нерекурсивним фільтром – зміна кольору одного пікселя може призвести до змін пікселів всього зображення. Поведінка нерекурсивного фільтра в цьому

сенсі більш передбачувана. На рис. 32 представлено коефіцієнти цифрового згладжуючого фільтру, що використовує вікно (маску) 3x3:

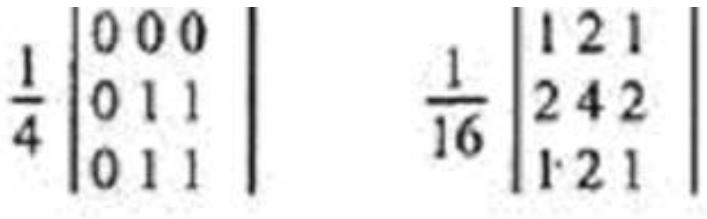


Рисунок 32 - Приклади згладжуючих фільтрів

Значення нормуючого коефіцієнта *K* дорівнює сумі елементів маски. Цим забезпечується збереження масштабу, яскравості перетвореного растра. Перший фільтр (рис 32, ліворуч) можна задати також маскою 2x2, якщо відкинути нульові коефіцієнти. При згладжуванні кольорових зображень можна використовувати модель RGB і виконувати фільтрацію по кожній компоненті.

18.Цифрова фільтрація зображень

Розглянемо цифрові фільтри в контексті різноманітної обробки растрових зображень. Цифрові фільтри дозволяють працювати з різними ефектами зображення, наприклад: згладжування (розмивання), зміна яскравості, різкості, контрасту, накладати такий ефект як деформація, усувати (зменшувати) шум і т. д.

Яскравість - це відношення сили світла, що випромінюється поверхнею, до площі її проекції на площину, яка перпендикулярна осі спостереження, оскільки зображення на екрані складається із таких світлових плям (точок), кожна із яких має свій розмір (площу) та свою яскравість.

Різкість – міра виразності границі між двома ділянками зображення або суб'єктивна характеристика зображення, що визначається ступенем розрізненості деталей на зображенні.

Контра́ст — міра виявлення (розпізнавання) об'єкта на якому-небудь тлі (фоні); контраст зображення - це відношення яскравості найсвітлішої та найтемнішої частин зображення.

Деформація зображень – ефект спотворення об'єктів - від простого «стиснення» до додавання зображенню фактури (наприклад, вигляд як у водної поверхні або диму). Формуються контрольні точки на зображенні, і потім вони зміщуються.

Шум (цифровий шум на зображенні) - дефект зображення, який полягає у виникненні хаотично розкиданих пікселів випадкових кольору і яскравості по всьому зображенню, тобто ці пікселі не відповідають дійсному світлу в тих місцях сцени. Основні причини виникнення шуму:

- чутливість сенсора цифрової фотокамери;
- витримка (витримка затвора фотокамери) ;
- щільність розміщення пікселів (розмір комірок матриці);

- технологія виробництва сенсорів;

- технічні параметри матриці (дефекти кристалічної решітки кремнію, «биті» пікселі матриці і т.д.).

У кольорових зображеннях шум має різну інтенсивність у кожному колірному каналі, що і надає йому певний колір.

Цифровий фільтр являє собою алгоритм обробки зображення. Велика група цифрових фільтрів має один і той же алгоритм, але ефект, що накладається фільтром на зображення, залежить від коефіцієнтів, використовуваних в алгоритмі (маска *M*).

Під фільтрацією зображень розуміється операція, що має своїм результатом зображення того ж розміру, що початкове, але отримане з початкового зображення за деякими правилами. Зазвичай інтенсивність (колір) кожного пікселя результуючого зображення обумовлена інтенсивностями (кольорами) пікселів, розташованих в деякій його околиці у початковому зображенні.

Згідно із запропонованим визначенням, операція, що полягає в послідовному застосуванні двох або більше фільтрів, теж є фільтрацією. Таким чином, можна говорити про складені фільтри, яким відповідають комбінації простих. Фільтрація зображень є однією з найбільш фундаментальних операцій комп'ютерного зору, розпізнавання образів і комп'ютерної обробки зображень. Фактично з фільтрації початкового зображення починається робота переважної більшості методів обробки зображень. Фільтри можна розділити на лінійні та нелінійні.

Лінійні фільтри являють собою групу фільтрів, що мають дуже простий математичний опис, але вони дозволяють добитися найрізноманітніших ефектів. До лінійних фільтрів відносять згладжуючі фільтри, фільтри для підвищення контрастності або різкості зображення, різницеві фільтри,

наприклад, фільтри Прюіта (Prewitt) і Собеля (Sobel) (рис.33). Ці різницеві фільтри є лінійними фільтрами, що задаються дискретними апроксимаціями диференціальних операторів. Найпростішим диференціальним оператором є взяття похідної за *x*-координатою $\frac{\partial}{\partial x}$.

$$M_{Prewit} = \frac{1}{3}\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \qquad M_{Sobel} = \frac{1}{4}\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \qquad M_{\nabla} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Рисунок 33 - Маски фільтрів Прюіта, Собеля та Лапласа

На відміну від згладжуючи фільтрів і фільтрів, що підвищують контрастність і не змінюють середню інтенсивність зображення, в результаті застосування різницевих операторів отримують зображення з середнім значенням інтенсивності пікселя близьким до нуля. Вертикальним границям вихідного зображення відповідають пікселі з великими за модулем значеннями на кінцевому зображенні. Тому різницеві фільтри називають також фільтрами, що знаходять границі; дані фільтри відіграють найважливішу роль для задач пошуку границь на зображенні. В результаті застосування даного фільтра Лапласа знаходяться границі будь-якої орієнтації. Крім того, різницеві фільтри дуже чутливі до шумів зображення, тому перед обробкою такого зображення необхідно попередньо здійснити фільтрацію шумів.

До нелінійних фільтрів відносять порогову фільтрацію (фільтр обнуляє всі значення пікселів, менше встановленого значення (порогу)), фільтри мінімум і максимум (результатом такої фільтрації є мінімальне і максимальне значення пікселів деякої околиці).

Приклади інших фільтрів:

• Box filter (усереднення, просте розмиття) - крім заглушення шуму спотворює різкі границі і розмиває дрібні деталі зображення. На рис. 34

наведено три найбільш використовуваних Box-фільтрів, які ще називаються шумозаглушуючими масками. Як видно, ці маски нормалізовані для отримання одиничного коефіцієнта передачі, щоб процедура заглушення шумів не викликала зміщення середньої яскравості зображення, яке обробляється.

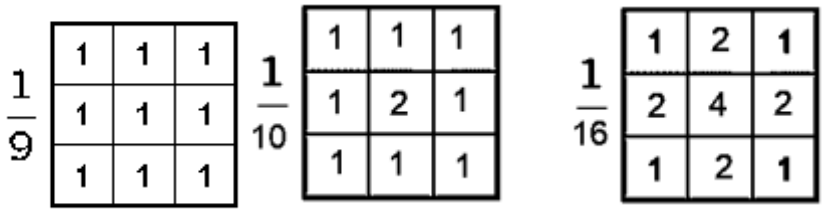


Рисунок 34 - Приклади масок для заглушення шумів

• Gaussian filter (розмиття по Гауссу) - менше розмиває дрібні деталі, краще прибирає шум (фільтр нижніх частот, прибирає високі частоти).

• Median filter – на різкі границі не впливає, прибирає дрібні деталі, зображення стає менш природним.

• Адаптивні фільтри - менше спотворюють деталі, залежать від великої кількості параметрів. Іноді зображення стає менш природним.

• «Advanced (просунуті)» фільтри - краще збереження деталей, менше розмиття. Часто складні в реалізації і дуже повільні.

19.Растрування кривих

Розглянемо задачу побудови кривої по контрольних точках (рис.35). Сплайновою кривою у комп'ютерній графіці називають складену криву, що сформована поліноміальними ділянками, і які задовольняють заданим умовам безперервності на границях ділянок. Але існує проблема, яка полягає в тому, що для візуалізації кривих ліній за допомогою комп'ютера потрібно знати їхній математичний опис. Розглянемо наступні варіанти подання кривих.

Рисунок 35 - Проблема побудови кривої по заданих точках

Перший спосіб полягає в заданні кривої за допомогою функцій змінних *x, y*:

$$y = f(x) \text{ або } F(x, y) = 0$$

Цей спосіб представлення накладає багато обмежень, особливо якщо деяким значенням координати *x* будуть відповідати декілька значень *y*, якщо крива утворює петлю. Крім того, в деяких точках кривої значення тангенса кута нахилу може прямувати в нескінченність.

Ці проблеми зникають у випадку параметричного способу представлення кривих, коли координати *x, y* описуються як функції від деякого параметра *t:*

$$x = x(t)$$

$$y = y(t)$$

При такому способі подання легко описуються замкнуті й многозначні функції, а замість тангенсів кутів нахилу використовуються дотичні вектори, які ніколи не бувають нескінченними. Як функція від параметра *t* найбільш часто використовується поліном третього порядку (кубічний), що задає координати точок у двовимірному просторі:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

Діапазон зміни параметра *t* може бути будь-яким, але найбільш часто його обмежують $0 \le t \le 1$. Усі наступні варіанти та вирази розглядаються для цього діапазону.

Значення дотичного вектора до кривої визначається як перша похідна функції *x (t)* по параметру *t (t*- показує відстань від P_o до P_1):

$$dx/dt = 3a_x t^2 + 2b_x t + c_x .$$

Крива лінія описується як послідовність окремих сегментів параметричних кубічних кривих, і у точках з'єднання сегментів повинні дотримуватися вимоги безперервності самої кривої (без розривів) та безперервності дотичних векторів (без зміни нахилу). В загальному випадку для інтерполяції *N* точок потрібен многочлен степеня *N-1*. Але поліноми меншого степеня, ніж три, не можуть забезпечувати виконання деяких умов, тому що немає контролю за ходом кривої. При використанні многочленів більш високих степенів, ніж три, збільшується складність розрахунків, виникають небажані осциляції (коливання) кривих (рис.36).

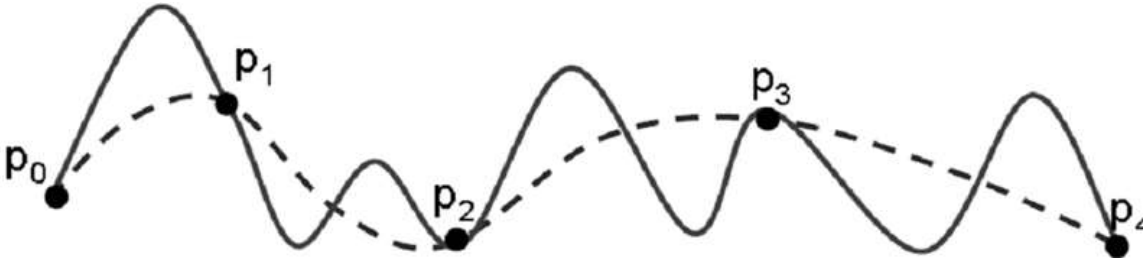


Рисунок 36 - Осциляція (коливання) кривої

Вимоги до побудови кривих:

- простота математичного опису;
- локальний контроль;
- безперервність;
- ефективні алгоритми обчислення.

В рамках форми параметричного подання кривих Ерміта ця крива задається не тільки через координати кінцевої і початкової точок (P_1 *і* P_2) але і через значення дотичних в цих точках (тут позначено як Q_1 *і* Q_2). При цьому вважають, що для початкової *точки* $t = 0$, а для кінцевої $t = 1$.

Підсумкове рівняння на одиничному інтервалі від 0 до 1 для t :

$$P(t) = P_1(2t^3 - 3t^2 + 1) + P_2(-2t^3 + 3t^2) + Q_1(t^3 - 2t^2 + t) + Q_2(t^3 - t^2).$$

Замість *P* підставляються координати опорної точки. Ці рівняння векторні (для кожної із координат). Недоліком форми Ерміта є необхідність явного задання значення дотичних векторів в кінцевих точках кривої, що не завжди зручно при реалізації інтерактивних режимів апроксимації. Форма Ерміта зручна при апроксимації вже наявних кривих, коли приблизно відомі довжина або напрямок дотичних векторів.

Для забезпечення умов безперервності при стикуванні сегментів кривої необхідно, щоб кінцева точка першого сегмента збігалася з початковою точкою другого сегмента, а дотичні вектора до сегментів в цих точках мали однаковий напрямок (довжина векторів може бути різною).

Крива Безьє розроблена математиком П’єром Безьє. Криві і поверхні Безьє були використані в 60-х роках компанією «Рено» для комп’ютерного проектування форми кузовів автомобілів. В даний час вони широко

використовуються в комп'ютерній графіці. Криві Безьє описуються в параметричній формі.

Крива Безьє – параметрична крива вигляду

$$\mathbf{B}(t) = \sum_{i=0}^{n} \mathbf{b}_{i,n}(t)\mathbf{P}_i, \qquad t \in [0,1]$$

де $\boldsymbol{P}_i$ – опорні вершини, а

$$\mathbf{b}_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$ - поліноми Берштайна.

$\boldsymbol{b}_{i,n}(t)$ - поліном Бернштейна це *i*-та функція базиса Бернштайна порядку *n*. Тут *i* - порядковий номер опорної вершини, *n* – порядок, який визначає функції базису Бернштайна.

Значення *t* виступає як параметр, якому відповідають координати окремої точки кривої. Параметрична форма опису може бути більш зручною для деяких кривих, ніж задання у вигляді функції.

Розглянемо криві Безьє, класифікуючи їх за значеннями m (кількості точок - 1):

m = 1 (по двох точках)

Крива вироджується у відрізок прямої лінії (рис. 37), який визначається кінцевими точками: P_0 і P_1, $P(t)=(1-t)P_0 + t\,P_1$.

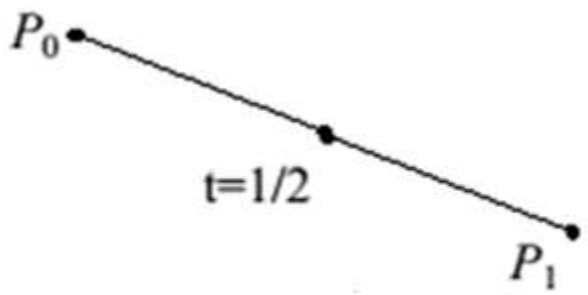


Рисунок 37 - Крива Безьє по двох точках

m=2 (по трьох точках, рис.38)

Особливості квадратичної кривої :

- крива проходить через точки P_0 та P_2 ;
- дотичні в точках P_0 та P_2 проходять(перетинають) через P_1.

$P(t)=(1-t)^2P_0+2t(1-t)P_1+t^2P_2$.

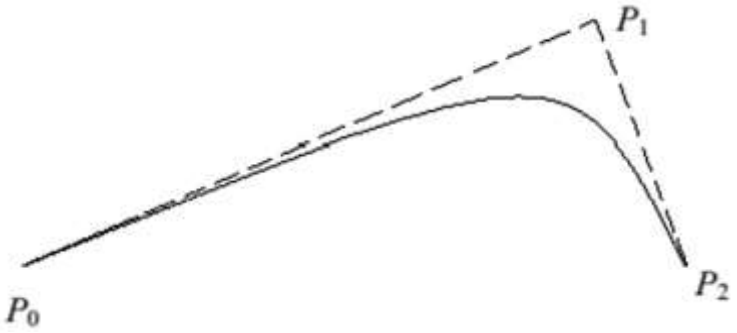


Рисунок 38 - Крива Безьє по трьох точках

У форми Безьє для задання кубічної кривої використовуються чотири опорні точки: P^0, P_1, P_2 та P_3; $m=3$ (по чотирьох точках, кубічна). Дана форма задання кубічної кривої дуже близька до ермітової форми Ерміта, але дотичні вектори в кінцевих точках Q_1 і Q_2 задаються відрізками P_0P_1 і P_2P_3:

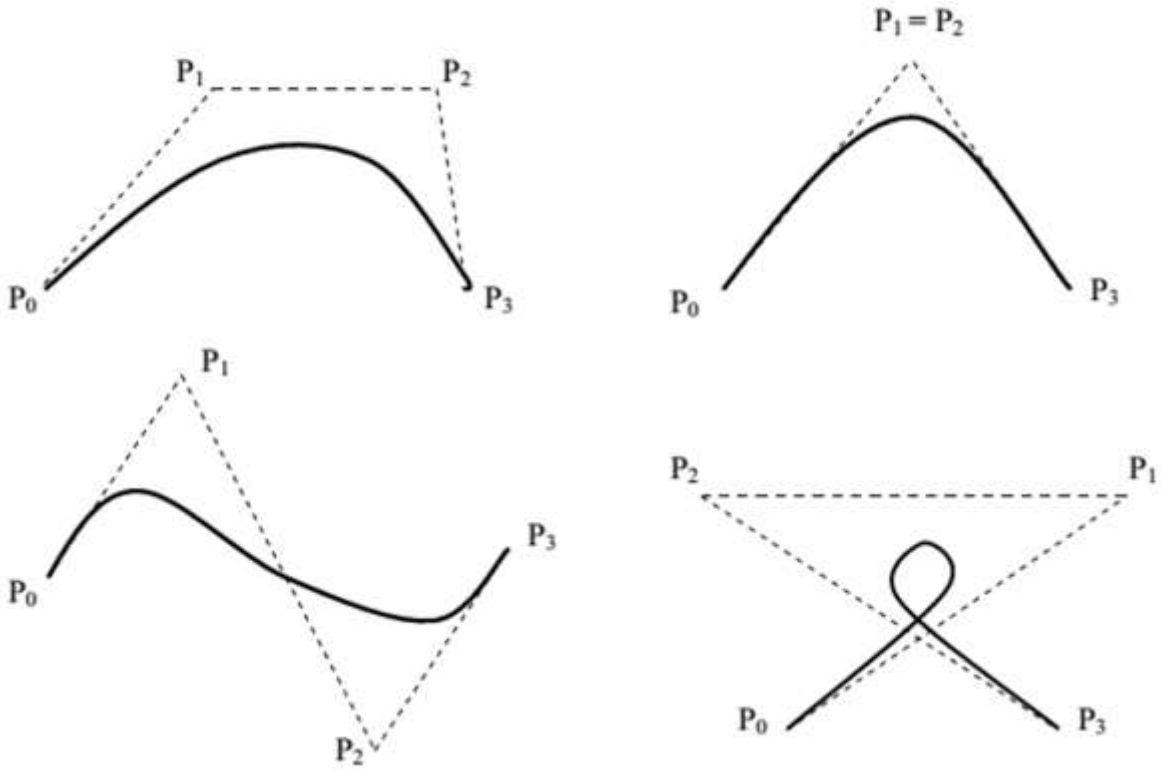


Рисунок 39 - Криві Безьє по чотирьох точках

Точки P_0, P_3 – *початкові/кінцеві точки (опорні),* P_1, P_2 - *точки, що задають кривизну.* Маємо підсумкове рівняння для координати:

$$P(t) = (1 - t)^3P_0 + 3t(t - 1)^2P_1 + 3t^2(1 - t)P_2 + t^3P_3.$$

Особливості кубічної кривої :

- проходить через P_0 в напрямку P_1;
- приходить в P_3 з напряму P_2 ;
- зазвичай не проходить ні через P_1, ні через P_2 - вони тільки для контролю кривизни.

Розглянувши приклади, можна побачити, що форма Безьє найбільш зручна при апроксимації кривих, які задані набором точок. У комп'ютерній графіці частіше використовується форма Безьє, ніж форма Ерміта.

Чотири опорні точки у формі Безьє визначають опуклий чотирикутник, всередині якого знаходиться сама крива. Опуклий многокутник є корисним

при виконанні багатьох операцій, наприклад, при відсіканні кривої по видимому об’єму. При виконанні цієї операції замість того, щоб відразу проводити перевірку відсікання кривої, спочатку перевіряється опукла її оболонка і тільки у разі перетину опуклою оболонкою видимого об’єму виникає необхідність у перевірці самої кривої.

Один із алгоритмів відтворення кривої Безьє (див. рис. 40) дозволяє обчислити координати *(x, y)* точки кривої Безьє за значенням параметра t:

1. Кожна сторона контуру багатокутника, що проходить по точках-орієнтирах, ділиться пропорційно значенню *t* (як на прикладі *t=0,5*) .

2. Точки поділу з’єднуються відрізками прямих і утворюють новий багатокутник. Кількість вузлів нового контуру на одиницю менше, ніж кількість вузлів попереднього контуру.

3. Сторони нового контуру знову діляться пропорційно значенню *t*. І так далі. Це продовжується до тих пір, поки не буде отримана єдина точка поділу. Ця точка і буде точкою кривої Безьє.

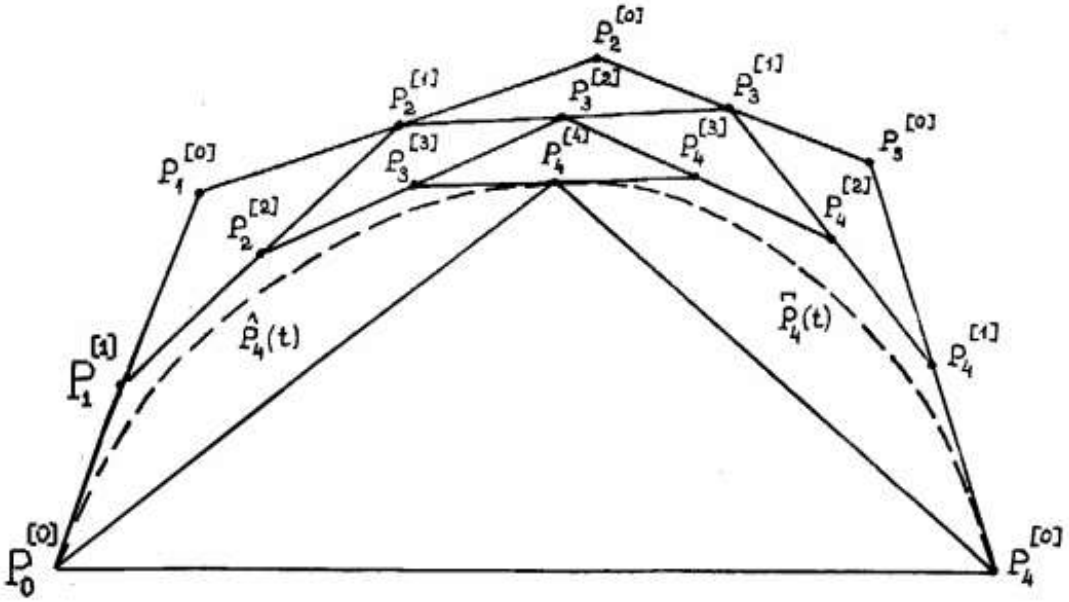


Рисунок 40 - Приклад відтворення кривої Безьє

Крива Безьє має такі властивості та особливості :

1. Крива є гладкою.

2. Крива Безьє починається в початковій вершині P_0 багатокутника і закінчується в його кінцевій точці P_n.

3. У початковій і кінцевій точках крива Безьє має дотичні, що збігаються з напрямком векторів (ребер багатокутника) P_0P_1 *і* $P_{n-1}P_n$, відповідно.

4. Крива завжди лежить всередині опуклої оболонки, натягнутої на вершини багатокутника.

5. Функціональні коефіцієнти $C_m^i \; t^i \; (1-t)^{m-i}$ при вершинах *Pi* є універсальними многочленами Бернштейна, вони невід'ємні, і їх сума дорівнює одиниці. Тому крива Безьє цілком лежить в опуклій оболонці, що породжується множиною точок {Pi}.

6. Крива Безьє завжди може бути представлена деяким степеневим поліномом, причому степінь полінома визначає найменший порядок кривої Безьє.

7. Крива Безьє проходить поблизу вершин багатокутника, причому, чим більше точок контролю кривизни на даному сегменті лінії, тим ближче до них проходить крива Безьє.

8. При побудові згладженої кривої по впорядкованому набору точок-орієнтирів, ламана P_0P_1 *і* $P_{n-1}P_n$ називається контрольною ламаною.

Недоліки кривих Безьє:

1. Степінь функціональних коефіцієнтів пов'язана з кількістю точок-орієнтирів.

2. При додаванні хоча б однієї точки в заданий набір, необхідно провести повний перерахунок коефіцієнтів в параметричному рівнянні кривої.

3. Зміна хоча б однієї точки призводить до помітної зміни виду всієї кривої.

20.Геометричні перетворення на площині

Для створення деяких сцен, для редагування моделей, для синтезу зображення у процесі перетворення, для отримання проекції на плоский екран потрібно застосовувати перетворення.

Перетворення, як і комп'ютерну геометрію, поділяють на двовимірні (перетворення на площині) і тривимірні (просторові). Спочатку розглянемо перетворення на площині. Точки на площині задаються за допомогою двох координат. Координати точок можна розглядати як елементи матриці *[x, y]*, тобто у вигляді вектора-рядка або вектора-стовпця.

Нехай на площині в прямокутній системі координат задана точка *N (x, y)*, яка в результаті послідовних перетворень переводиться в точку *N** з координатами *(x*,y*)*.

Будемо розрізняти два класи перетворень:

- нелінійні перетворення (довільне перетворення точок моделі);

- лінійні перетворення (трансформує вихідну лінійну комбінацію векторів в деяку лінійну комбінацію), коли координати кінцевої точки можна описати рівняннями виду:

$$x^* = a_x x + b_x y + c_x,$$

$$y^* = a_y x + b_y y + c_y,$$

де a_x, b_x, c_x, a_y, b_y і c_y – деякі коефіцієнти.

У лінійних перетвореннях особливу роль грають кілька важливих окремих випадків: поворот, масштабування, відображення і перенос (зсув), що мають очевидні геометричні трансформації.

Поворот навколо початку координат на кут φ описується наступними рівняннями (рис. 41):

$$x^* = x\cos\varphi - y\sin\varphi,$$

$$y^* = x\sin\varphi + y\cos\varphi.$$

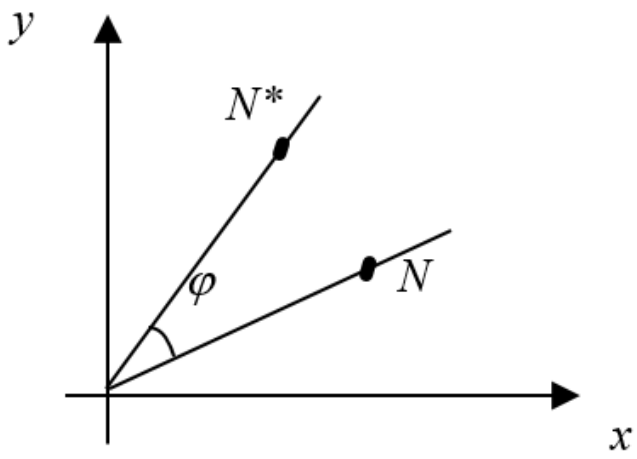


Рисунок 41 - Приклад перетворення повороту навколо початку координат

Але часто це не те, що потрібно. Якщо потрібно виконати поворот щодо заданої точки (x_0 , y_0), то в цих рівняннях можна замінити x на ($x - x_0$), а y на ($y - y_0$) , x^* – на ($x^* - x_0$), а y^* – на ($y^* - y_0$) :

$$x^* = x_0 + (x - x_0)\cos\varphi - (y - y_0)\sin\varphi,$$

$$y^* = y_0 + (x - x_0)\sin\varphi + (y - y_0)\cos\varphi.$$

Масштабування (розтягнення-стиснення) уздовж координатних осей і відносно початку координат:

$$x^* = M_x x,$$

$$y^* = M_y y.$$

$$M_x > 0,\ M_y > 0.$$

Розтягнення уздовж осі абсцис (ординат) має місце при $M_x > 1$ ($M_y > 1$), стиснення при $M_x < 1$ ($M_y < 1$).

Відображення відносно координатних осей описується наступними рівняннями:

- відносно осі абсцис:

$$x^* = x,\ y^* = -y;$$

- відносно осі ординат:

$$x^* = -x,\ y^* = y.$$

Перенос (зсув) точки вздовж осі абсцис на величину Δx, та на величину Δy вздовж осі ординат відповідно:

$$x^* = x + \Delta x,\ y^* = y + \Delta y.$$

У комп'ютерній графіці частіше використовується матричний запис цих перетворень. Матриці, що відповідають випадкам повороту, масштабування і відображення, будуються легко, але з записом рівнянь зсуву (переносу) в матричній формі є проблема: це неможливо.

У ситуаціях, коли поєднуються кілька перетворень, було б зручно мати єдине представлення перетворень в зручній формі - за допомогою множення матриць, тобто мати матричний добуток для кожного елементарного перетворення. Але якщо перетворення включає операцію переносу, то за допомогою матриці *2x2* цю операцію неможливо уявити: перенос (зсув), на відміну від масштабування і повороту, реалізується за допомогою додавання. Однак при використанні для представлення точок на площині *однорідних координат*, це стає реальним.

Двовимірні однорідні координати точки мають такий вигляд:

$$[X\ Y\ W]$$

де W – будь-який множник, що не дорівнює 0.

Двомірні декартові координати *(x, y)* точки отримуються із однорідних шляхом ділення останніх на множник W:

$$x = X/W;\ y = Y/W;\ W \neq 0$$

Однорідні координати можна представити як промасштабовані з коефіцієнтом *W* значення двовимірних координат, що розташовані в площині з *Z = W*. В силу довільності значення *W* в однорідних координатах не існує єдиного представлення точки, заданої в декартових координатах.

Таким чином, третя компонента вектора дозволяє використовувати матричний запис для всіх лінійних перетворень (якщо використовувати матриці *2x2*, то неможливо уявити перенос (зсув)), а також дозволяє представити так званий перспективний поділ.

Зауважимо, що якщо помножити дві матриці, що задають повороти на кути α і β, то вийде матриця повороту на кут α + β. Це легко перевірити, перемноживши відповідні матриці і використавши формули для косинуса і синуса суми.

За допомогою трійок однорідних координат і матриць третього порядку можна описати будь-яке перетворення на площині рівняннями виду:

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = [P] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

де $\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix}$ и $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ – вектори-стовпці координат кінцевої та початкової точок відповідно, *[P]* – матриця відповідного перетворення, яка має для двовимірного випадку розмір *3x3*.

Поворот:

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

Масштабування:

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} Mx & 0 & 0 \\ 0 & My & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \qquad M_x > 0,\ M_y > 0.$$

Відображення:

відносно осі абсцис

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

відносно осі ординат

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Перенос(зсув):

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Будь-яке геометричне перетворення на площині або в просторі можна представити як послідовне виконання розглянутих вище приватних випадків. При такому підході підсумкову матрицю перетворення обчислюють як добуток матриць окремих випадків, з яких складається підсумкове перетворення. Наприклад, поворот можна представити як комбінацію масштабування і зсуву. Тим не менш, для зручності поворот також вважається елементарним перетворенням. Поворот навколо довільного вектора представляється як комбінація поворотів навколо координатних осей.

Слід пам’ятати, що перемножування матриць є некомутативною операцією, тобто важливий порядок її виконання. Правильний порядок перемноження визначається положенням матриці деякого перетворення щодо матриці координатного вектора. Матриця, найближча до матриці координатного вектора, задає перше перетворення, а найбільш віддалена від нього матриця задає останнє перетворення.

Нехай точка М з координатами *(x, y)* послідовно піддається перетворенням виду $P_1, P_2, \ldots, P_n$. Причому, $[P_1]$ – матриця першого

перетворення (розміром *3x3*), а $[P_n]$ – останнього. Кінцеві координати точки (*x*, y**) можуть бути обчислені за допомогою рівняння:

$$\begin{bmatrix} x^* \\ y^* \\ \\ 1 \end{bmatrix} = [P_n] \ldots [P_2]\,[P_1] \begin{bmatrix} x \\ y \\ \\ 1 \end{bmatrix},$$

21.Геометричні перетворення у просторі

Відтворення просторових об’єктів відбувається набагато складніше, ніж плоских: система координат об’єкту стає тривимірною, геометричні примітиви утворюють в ній просторову композицію, яку потрібно вміти коректно описати. Самі примітиви теж стають просторовими, для їх опису може вводитись, наприклад, тривимірна система координат примітиву та система координат спостерігача. В такій системі координат т.зв. сценарний простір не відповідає розміру екрана. При цьому система координат екрану залишається двовимірною (плоскою), екран лежить у площині *XY* системи координат спостерігача. Отже, виникає необхідність переходу від просторового подання об’єкта до плоского, тобто необхідність проекціювання. Крім того, щоб просторові об’єкти виглядали об’ємними і правильно передавалися просторові взаємозв’язки між ними, потрібно виконати ще цілий ряд операцій, відсутніх в двовимірній графіці. Це, наприклад, накладення тіней від джерел освітлення, видалення на зображенні тих ділянок об’єктів, які не повинен бачити спостерігач тощо.

При розгляді тривимірних перетворень приймається ліва система координат (рис.42, зліва), яка є більш зручною для тривимірної комп’ютерної графіки. Так, наприклад, якщо поверхня екрана поєднана з площиною XY, точка спостереження знаходиться на від’ємній частині осі *z*, а точки із більшим значенням *z*-координати знаходяться далі від спостерігача.

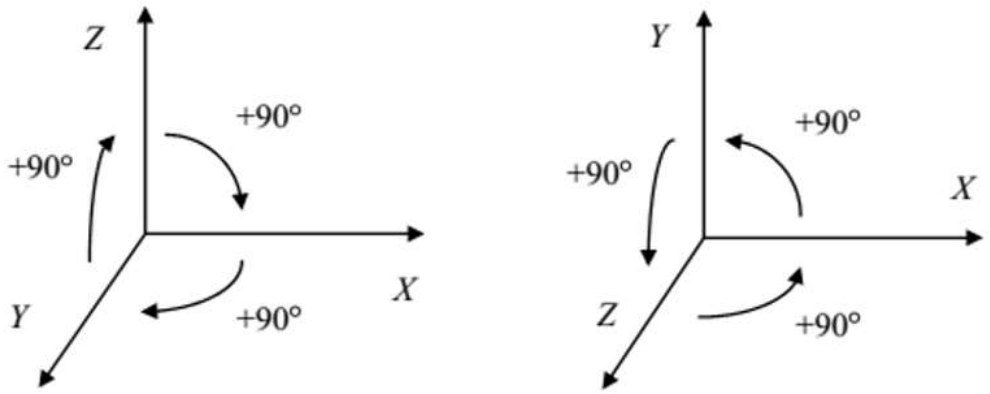


Рисунок 42 - Системи координат

Загальне рівняння тривимірного геометричного перетворення:

$$x^* = a_x x + b_x y + c_x z + d_x,$$

$$y^* = a_y x + b_y y + c_y z + d_y,$$

$$z^* = a_z x + b_z y + c_z z + d_z,$$

де a_x, b_x, c_x, d_x, a_y, b_y, c_y, d_y, a_z, b_z, c_z и d_z – деякі коефіцієнти.

Робота з однорідними тривимірними координатами і матрицями перетворення (формування і композиція) аналогічна двовимірному варіанту, тому тут будуть розглянуті тільки матриці перетворень зсуву, масштабування, відображення і повороту.

Особливістю повороту в тривимірному просторі є те, що один поворот в просторі слід розкладати на 3 повороти: навколо осі абсцис, навколо осі ординат, навколо осі аплікат. Друга особливість - це напрям кута повороту. Існує домовленість, що додатнім напрямом кута повороту для обох систем координат вважається той напрямок, при якому, якщо дивитися з додатного кінця піввісі в центр координат, здійснюється найкоротший перехід від однієї додатної осі до іншої: для *X (обертання навколо осі X)*: *Y* → Z; для Y: Z → X; для Z: X → Y. У лівій системі координат додатними будуть повороти за годинниковою стрілкою.

Поворот навколо осі абсцис (осі x) на кут φ:

$$x^* = x,$$

$$y^* = y\cos\varphi - z\sin\varphi,$$

$$z^* = z\cos\varphi + y\sin\varphi.$$

У матричній формі з використанням однорідних координат:

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Поворот навколо осі ординат (осі *y*) на кут φ:

$$x^* = x\cos\varphi + z\sin\varphi,$$

$$y^* = y,$$

$$z^* = z\cos\varphi - x\sin\varphi.$$

В матричній формі:

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Поворот навколо осі аплікат (осі *z*) на кут φ:

$$x^* = x\cos\varphi - y\sin\varphi,$$

$$y^* = y\cos\varphi + x\sin\varphi,$$

$$z^* = z.$$

В матричній формі:

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

У випадку повороту у зворотному (від'ємному) напрямку значення кутів беруться зі знаком мінус та отримуються за правилами: *cos(-α) = cosα, sin(-α) = -sinα .*

Тривимірний *перенос* є простим розширенням двовимірного випадку:

$$\begin{aligned} x^* &= x + \Delta x, \\ y^* &= y + \Delta y, \\ z^* &= z + \Delta z. \end{aligned} \qquad \text{або} \qquad \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Іноді необхідно виконати перенос на величину, що не є константою, а що залежить від початкових координат точки. Це перетворення описується наступними формулами:

$$\begin{aligned} x^* &= x + ay + bz, \\ y^* &= y + cx + dz, \\ z^* &= z + ex + fy. \end{aligned} \qquad \text{або} \qquad \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & b & 0 \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Масштабування описується наступними рівняннями:

$$\begin{aligned} x^* &= M_x x, \\ y^* &= M_y y, \qquad M_x>0,\ M_y>0,\ M_z>0. \\ z^* &= M_z z. \end{aligned}$$

Або:

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} Mx & 0 & 0 & 0 \\ 0 & My & 0 & 0 \\ 0 & 0 & Mz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

На відміну від двовимірного випадку, у просторі розглядаються випадки *відображення* відносно координатних площин.

Відображення відносно площини $z = 0$:

$$\begin{aligned} x^* &= x, \\ y^* &= y, \\ z^* &= -z. \end{aligned} \qquad \text{або} \qquad \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Відображення відносно площини $y = 0$:

$$\begin{aligned} x^* &= x, \\ y^* &= -y, \\ z^* &= z. \end{aligned} \qquad \text{або} \qquad \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Відображення відносно площини $x = 0$:

$$\begin{aligned} x^* &= -x, \\ y^* &= y, \\ z^* &= z. \end{aligned} \qquad \text{або} \qquad \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

22.Заливка замкнутих контурів на зображенні

Заливка (заповнення, зафарбовування) контурів зображень – задача, що часто використовується на практиці, суть якої - заповнити деяку область зображення, що обмежена контуром або встановленим (заданим) кольором. Наприклад, в деяких графічних редакторах існують графічні примітиви у вигляді контурів – відкритих або замкнутих. Замкнуті контури мають окрему характеристику - заливку. Задача заливки використовується багатьма процедурами та складними алгоритмами КГ, в т.ч. і для підвищення реалістичності сцен. Пікселі контура (границі) – це границя, за яку не можна виходити в ході послідовного перефарбовування пікселів, що знаходяться в замкнутому контурі.

В КГ розглядаються прості та багатомісні задачі заливки. Існує кілька типів заливок:

- заливка основним кольором, тобто заповнення внутрішньої області (контура) вибраним кольором;

- текстурна заливка - це така заливка, в якій весь внутрішній простір об'єкта заповнюється єдиним малюнком, що відображає текстуру таких поверхонь, як метал, камінь, дерево, бетон і т.д., за допомогою спеціально створених кольорових растрових зображень. Набір текстурних заливок - це великий набір математичних моделей, у кожної з яких є особливі параметри. Ці моделі розташовані у декількох бібліотеках, в яких зберігаються кілька десятків описів текстур, кожна з яких має ім'я, яке описує її зовнішній вигляд. Основним параметром кожної з моделей, за якими будується зображення, є номер текстури;

- підвид текстурної заливки - заливка зображенням-картою (візерунком); візерунки є заливкою у вигляді набору заздалегідь підготовлених однотипних прямокутних комірок, що мають оригінальний

рисунок. В результаті використання такої заливки внутрішній простір об’єкта заповнюється сукупністю цих комірок, та утворює безперервний малюнок. Повноколірний візерунок (векторна заливка) - це повноколірний візерунок з використанням і ліній, і заливок. За допомогою візерункових заливок можна створювати мозаїчні фони або підвищувати реалістичність створюваних зображень (при використанні растрових заливок). Все заливки складаються з невеликих квадратних плиток, які складаються в правильну мозаїку без спотворень. Дуже важливо, що даний тип заливки дозволяє в інтерактивному режимі змінювати елементарні частини, що дозволяє створювати новий, абсолютно не схожий на вихідний, візерунок;

- градієнтна заливка - заповнення двома кольорами з плавним переходом між ними. Градієнтні заливки на відміну від однорідних відображають складні колірні переходи між різними кольорами. Додаючи в градієнт необхідне число проміжних кольорів, можна формувати багатоколірні градієнтні заливки. В цьому випадку градієнтну заливку можна трактувати як своєрідний візерунок, утворений поступовим переходом між двома або кількома кольорами або відтінками одного кольору.

Існує два види вирішення задач заливки, що обумовлені способом задання області:

- перша задача, пов’язана із заповненням області, яка представлена внутрішньою частиною багатокутника, заданого через координати його вершин; можна сказати, що область задана математичним описом (може бути коло, еліпс і т.д.);
- друга задача - для заливки так званої довільної замкнутої області (контура).

Кожна із цих областей може бути задана одним із двох способів:

- гранично визначені області, що задаються своєю замкнутою (!) границею (контуром), яка має таку властивість, що коди кольору пікселів границі відрізняються від кодів пікселів внутрішньої частини області, що перефарбовується. Для кодів пікселів внутрішньої частини області накладаються дві умови: вони повинні відрізнятися від пікселів границі та від нового кольору пікселів (кольору перефарбовування). Також якщо всередині гранично-визначеної області може бути ще одна границя, яка зафарбована тим же кольором, що і зовнішня границя, то відповідна частина області не повинна перефарбовуватися;
- внутрішньо визначені, що відтворені одним певним кольором пікселів, що відмінний від кольору фону. При заливці (зафарбуванні) цей код замінюється на новий код зафарбовування.

Початковими умовами для виконання задачі заливки контура можуть бути наступні: деяким чином задається область, що зафарбовується, код пікселя, яким буде виконуватися заливка, та стартова точка в області, починаючи з якої почнеться заливка.

Також важливим фактором є зв'язність. Область, що зафарбовується, та її границя розглядаються як деяка зв'язна множина пікселів. За способами доступу до сусідніх пікселів будемо розглядати відомі області із 4- та 8-зв'язаністю цих пікселів (рис.16). В 4-зв'язаних областях доступ до сусідніх пікселів здійснюється за чотирма напрямками: горизонтально вліво і вправо, і вертикально вгору і вниз. У 8-зв'язаних областях до цих напрямків додаються ще 4 діагональних. Використовуючи зв'язність можна, рухаючись від точки заливки, досягти і зафарбувати всі пікселі області.

Важливо відзначити, що для 4-зв'язної області границя є 8-зв'язною та можна застосовувати 4- та 8- зв'язний алгоритми, і навпаки у 8-зв'язної області границя 4-зв'язна (застосовується 8-зв'язний алгоритм). Якщо застосувати 8-

зв’язний алгоритм (в 8-зв’язній області) із 8-зв’язною границею, то це приведе до "просочування" через границю і заливції пікселів в прилеглій області (рис.43, праворуч)

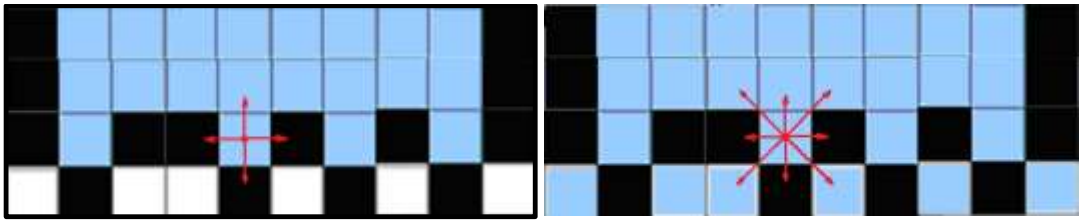


Рисунок 43 - Заливка для різної зв’язності

Розглянемо універсальний алгоритм. За допомогою цього алгоритму можна зафарбовувати будь-які замкнуті області (контури). Початковими (вхідними) даними для цього алгоритму (процедури) є точка, що належить цій області (т. зв. затравочний піксель), колір границі області та новий колір області. Суть методу полягає в наступному (варіант для гранично визначеної області). Береться поточна точка (на першому кроці цією точкою є початкова точка) і перевіряється: якщо точка не належить контуру і не зафарбована в новий колір (тобто має старий колір), то зафарбовується в новий колір. Далі логіка проста - якщо даний піксель належить області зафарбовування, то, швидше за все, його найближчі сусіди також належать даній області (на рисунку показано 4-зв’язність). Для кожного сусіда викликається ця ж процедура (яка або зафарбує точку, або ні) до тих пір, поки не дійдемо до границі області. Якщо точка вже зафарбована або є границею, то нічого робити не треба.

Незважаючи на простоту алгоритма, така рекурсивна реалізація має недоліки у вигляді повільної роботи та необхідності використання певних (великих) об’ємів пам’яті для зберігання вкладених викликів процедури через велику глибину рекурсії.

Для внутрішньо визначеної 4-зв’язної області аналогічно береться поточна точка (на першому кроці цією точкою є початкова точка) і перевіряється: якщо точка має колір, що задає область (старий колір), то вона перефарбовується в новий колір, і далі відбувається виклик цієї ж процедури для 4-х сусідів. Процес повторюється до виходу за область.

Розглянемо інший, ітеративний алгоритм:

1. Додається елемент в стек;

Поки стек не порожній :

2. Витягується елемент зі стека;

3. Присвоюється елементу необхідне значення (зафарбовується елемент);

4. Кожен сусідній елемент додається в стек, якщо він

4.1. Не є границею;

4.2. Не перефарбований раніше (тобто його колір відрізняється від кольору границі або кольору зафарбовування);

5. Перехід до кроку 2.

Такий ітеративний алгоритм економніший, оскільки в стек треба поміщати тільки координати (елемента, пікселя). Використовуючи показані підходи, можна реалізувати як 4- , так і 8- зв’язні алгоритми.

Розглянемо алгоритм, який є ефективною модифікацією попереднього алгоритму, що за одну ітерацію зафарбовує цілий рядок пікселів. Ідея цього алгоритму полягає в тому, що для заданої точки *(x, y)* з області визначається і заповнюється максимальний відрізок $[x_1, x_2]$, що лежить всередині області і містить цю точку. Після цього перевіряються відрізки області, що лежать вище і нижче знайденого відрізка. Якщо такі пікселі знаходяться, то рекурсивно викликається функція для їх обробки.

Якщо розглянути алгоритм детальніше, то можна виділити наступні етапи:

Нехай *(x, y)* – координати початкового пікселя,

- Line_Fill (x,y);

- знайти крайню ліву точку x_1 інтервалу;

- знайти крайню праву точку x_2 інтервалу;

- заповнити інтервал від x_1 до x_2;

- для всіх *x* від x_1 до x_2 перевірити, якщо точка *(x, y +1)* не належить границі і не зафарбована, то Line_Fill (x, y +1);

- для всіх *x* від x_1 до x_2 виконати якщо точка *(x, y – 1)* не належить границі і не зафарбована, то Line_Fill (x, y – 1) .

Цей алгоритм є рекурсивним, але оскільки функція Line_Fill викликається при переході на сусідні лінії (лінії в цілому), а не для кожного пікселя, то це зменшує кількість вкладених викликів, а отже, зменшуються витрати стекової пам'яті.

Алгоритм ефективно працює і для областей із т. зв. «дірками» (рис.44).

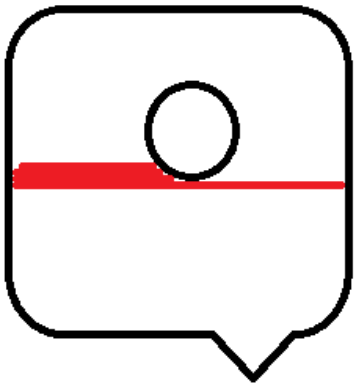


Рисунок 44 - Заливка рядками

Найпростіший спосіб - це заповнення прямокутника. Відомі його протилежні вершини. Тут заливка полягає у растеризаціях горизонтальних або вертикальних відрізків, що заповнять прямокутник.

При заливці багатокутника не завжди обов'язково задавати початкову точку: задано багатокутник і необхідно його заповнити.

Найпростіший спосіб заповнення багатокутника (полігона), заданого координатами вершин, полягає у визначенні того, чи належить поточний піксель внутрішній частині багатокутника: пробігаємо по всіх точках, якщо належить, то піксель зафарбовується новим кольором. Визначити приналежність поточного пікселя багатокутнику можна простим методом - наприклад, визначенням сумарного кута із вершиною в пікселі, при обході вершини багатокутника по контуру. Якщо піксель знаходиться всередині, то сумарний кут буде дорівнювати 360^o, якщо піксель лежить поза багатокутником , то сумарний кут дорівнює 0^o (рис.45)

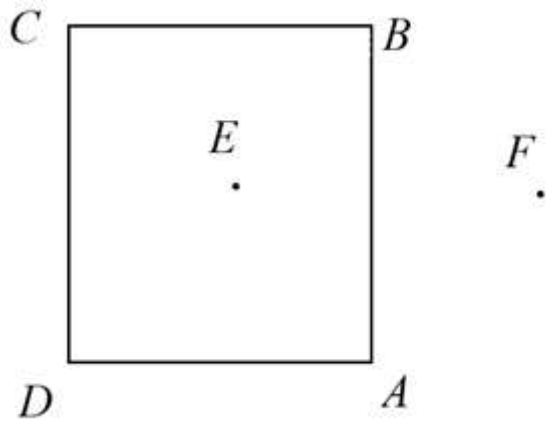


Рисунок 44 - Заливка багатокутника

Обчислення приналежності проводиться для всіх пікселів екрану і, оскільки більшість пікселів швидше за все буде знаходитись ззовні багатокутника, то даний спосіб занадто витратний з точки зору використання обчислювальних ресурсів.

Інший алгоритм підходить тільки для тих випадків, коли область заливки може бути задана у вигляді багатокутника. Будемо вважати, що багатокутник має координати вершин $P_1 (x_1, y_1)$, $P_2 ()$, ..., $P_N ()$, P_1 і потрібно відобразити його на екрані разом з усіма внутрішніми точками. Для зручності будемо припускати, що кожне ребро багатокутника задається координатами його

кінців x_1, y_1 і x_2, y_2 (при цьому $y_2 \geq y_1$). Також приймемо, що координата x зростає зліва направо, а координата y згори вниз.

Переважна більшість алгоритмів растеризації багатокутників засновані на наступному припущенні: будь-яка січна пряма перетинає границю (ребро) багатокутника парну кількість разів. Це твердження невірне тільки в двох випадках (рис.45):

- коли січна пряма містить ребро (проходить через ребро);

- коли січна пряма містить вершину (проходить через вершину), а суміжні ребра лежать по одну сторону від січної прямої – вище або нижче.

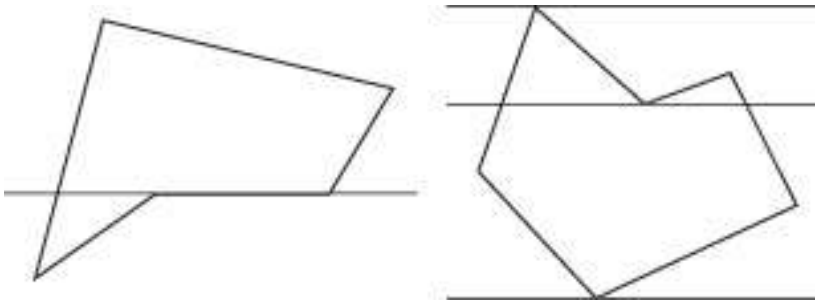


Рисунок 45 - Особливі випадки розташування многокутника і січної

Ці два випадки досить легко виявити, тому при розгляді алгоритмів растеризації багатокутників будемо вважати, що наведене вище припущення завжди вірно.

Алгоритм, заснований на роботі зі списком реберних точок, умовно можна представити як суперпозицію наступних трьох етапів:

1. На першому етапі растеризуються всі негоризонтальні ребра многокутника (растеризуються усі границі). Іншими словами, алгоритм представляє собою цикл вздовж осі y. В циклі знаходяться точки перетину ребер многокутника із горизонтальними січними. Кожне ребро розглядається як відрізок (рис.46).

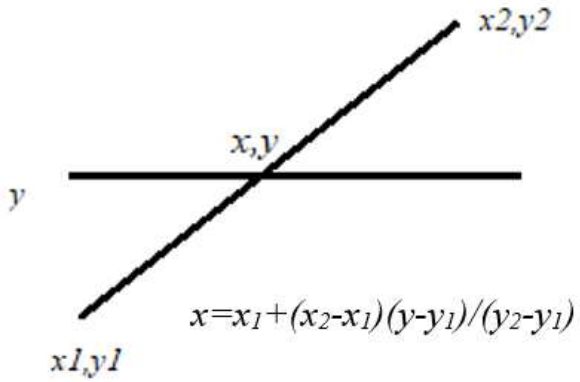


Рисунок 46 - Знаходження *x* - координати точки перетину

Знаходиться перетин горизонталей (з відомою *y*-координатою) з кожним ребром (відрізком, у якого відомі координати кінців). Потім складається список, де для кожної *y*-координати ставляться у відповідність пари *x*-координат (рис.47).

y_{min}	$x_2; x_1$
y_{min+1}	$x_2'; x_1'$
y_{min+2}	$x_2''; x_3''; x_4''; x_1''$
y_{max}	

y_{min}	$x_1; x_2$
y_{min+1}	$x_1'; x_2'$
y_{min+2}	$x_1''; x_2''; x_3''; x_4''$
y_{max}	

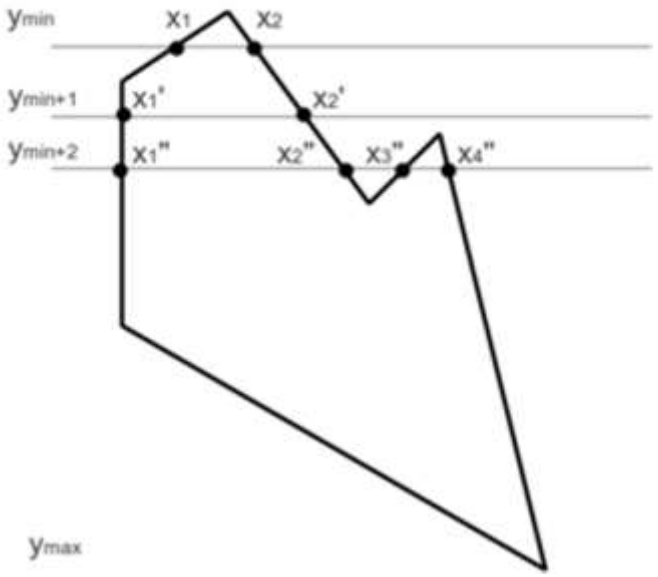


Рисунок 47 - Формування пар *x*-координат точок перетину

Для наведеного многокутника ми отримаємо відповідні значення при обході многокутника за годинниковою стрілкою.

2. На другому етапі для кожного значення *y* списки *x* координат впорядковуються за зростанням. Пари відсортованих точок перетину задають інтервали заливки.

3. На третьому етапі для кожного *y* заповнюються необхідним кольором всі отримані відрізки. Перевага цього алгоритму в тому, що кожен піксель обробляється один раз. Зрозуміло, що один з найважливіших етапів у цьому алгоритмі порядкового сканування – сортування.

Існує модифікація даного алгоритму, оптимізована по витраті пам’яті. У ній на кожному кроці обробляються тільки ті ребра, які перетинаються з поточною лінією розгортки.

23.Алгоритми відсікання відрізків

Існує наступна задача. При генеруванні зображень виникають ситуації, коли можуть створюватися фігури довільної форми і розмірів, але поле відтворення монітору не може відобразити згенеровані зображення повністю. Також іноді виникають ситуації, коли необхідно задати область зображення на екрані і виводити зображення тільки всередині цієї області. Для вирішення цих завдань застосовують алгоритми відсікання (clipping), оскільки якщо обчислювати частину зображення, яка не виводиться на монітор, то це призводить до «гальмування». Також потрібно усунути невидимі частини, це більш актуальна задача, ніж проблема «гальмування».

У простих графічних системах достатньо двовимірного відсікання, в тривимірних пакетах використовується трьохвимірне відсікання.

Тепер проведемо декомпозицію цієї складеної задачі (відсікання). Графічні об’єкти розглядаються такими, що складаються із відрізків, що відсікаються. Ці відрізки діляться на: повністю видимі, повністю невидимі і такі, що перетинають поле видимості (вікно) – частково видимі (рис.48).

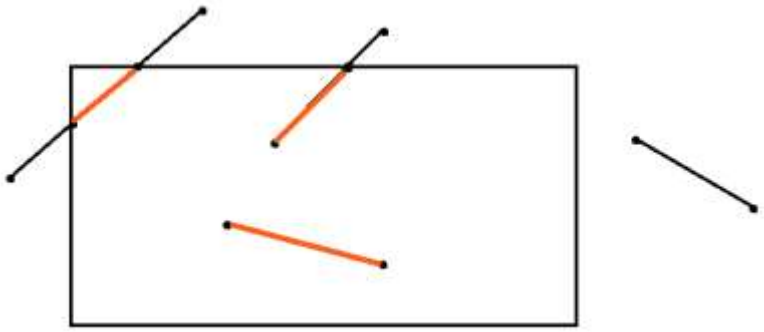


Рисунок 48 - Приклади взаємного положення відрізків відносно вікна відсікання

За способом визначення того, потрібне відсікання чи ні, існують наступні основні типи алгоритмів відсікання:

- алгоритми, що використовують кодування кінців відрізка або всього відрізка;
- алгоритми, які використовують параметричне представлення відрізків та вікна відсікання.

Розглянемо приклад (рис.49), де розташовано трикутник (*PRQ*) та прямокутник(*ABCD*), що задає область виводу. Необхідно визначити вершини многокутника, який потрапляє у вікно(область) виводу.

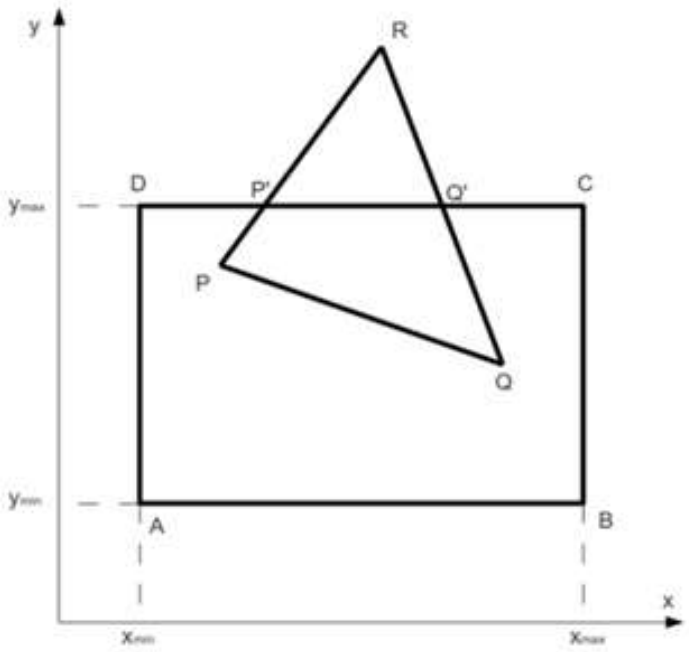


Рисунок 49 - Приклад розташування об’єкта та вікна виводу

Як видно зі схеми, для відтворення трикутника *PRQ* повинні виконатися тільки команди відтворення відрізків *PQ*, *PP’* та *Q’ Q*. При цьому координати точок *P’* та *Q’* заздалегідь невідомі. Реально можливі різноманітні варіанти розташування точок відрізка і області виводу.

Для вирішення цих завдань створено алгоритм відсікання, що запропонований Сазерлендом і Коеном. Даний алгоритм ефективно (за швидкодією) працює у випадках, коли більшість примітивів міститься всередині великого вікна або ззовні відносно маленького вікна, тобто

алгоритм Коена-Сазерленда дозволяє швидко виявити відрізки, які треба залишити або повністю видалити (відкинути). Особлива ситуація виникає при необхідності проведення обчислень тоді, коли відрізок не потрапляє в один з цих випадків. Суть алгоритму полягає в тому, що кінцям відрізка присвоюється 4-бітний код: *b0, b1, b2, b3*. Цей 4-бітний код містить інформацію про розташування точки відносно області виводу. Область виводу окреслена вертикальними та горизонтальними прямими. Границя вікна вважається такою, що належить вікну.

Значення розрядів коду (рис. 50, зліва):

b0 -й = 1 - точка над верхнім краєм вікна;

b1 -й = 1 - точка під нижнім краєм вікна;

b2 -й = 1 - точка праворуч від правого краю вікна;

b3 -й = 1 - точка зліва від лівого краю вікна.

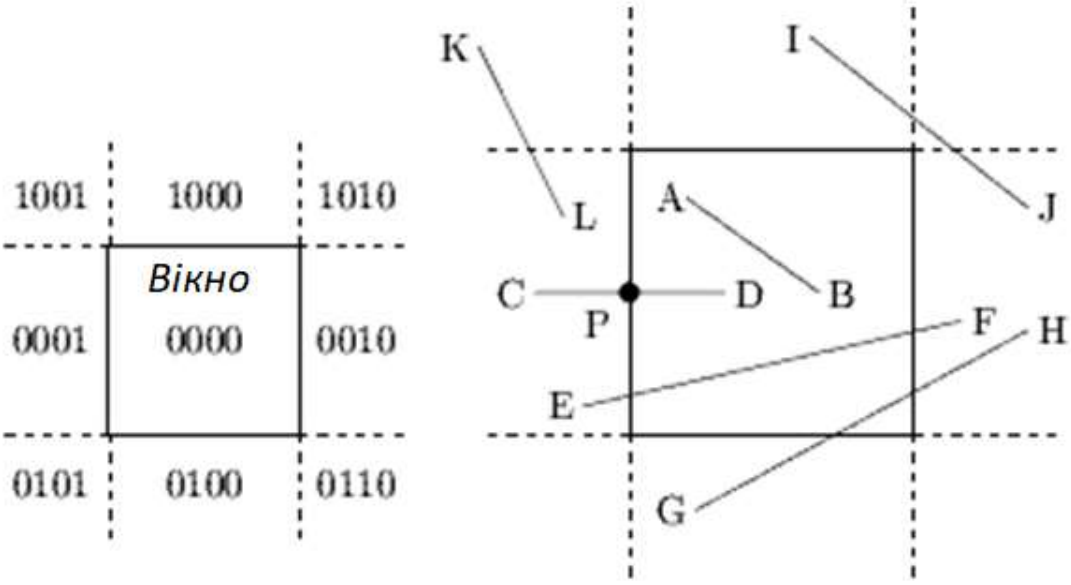


Рисунок 50 - Приклад розташування об’єкта та вікна виводу

Можливі наступні варіанти розташування відрізків відносно вікна відсікання (рис.50, справа).

Порядок визначення розташування відрізків, як, наприклад, цілком у вікні або цілком поза вікном можна описати так:

- перевіряються коди обох кінців відрізка. Якщо вони дорівнюють 0, то відрізок знаходиться повністю всередині вікна, відсікання не потрібне, відрізок приймається як тривіально видимий (відрізок *AB* на рис. 50, справа);

- якщо результат операції логічного І кодів обох кінців відрізка ненульовий (в однакових позиціях кодів є одиниці), то відрізок лежить цілком поза вікном, відрізок відкидається як невидимий (відрізок *KL* на рис. 50, справа).

- але виникає і третій випадок. Якщо в результаті виконання операції логічного І кодів обох кінців відрізка виходить нуль, то відрізок може бути частково видимим (відрізки *CD, EF, GH*) або цілком невидимим (відрізок *IJ*);

Далі для таких відрізків (відрізки *IJ, CD, EF, GH*) потрібно визначити координати перетинів зі сторонами вікна і для кожної отриманої частини відрізка визначити, видимий він чи ні. Так, для відрізків *IJ , CD* необхідно буде обчислення одного перетину з границями вікна (точка *P*), для інших (*EF і GH*) – двох.

При обчисленні координат точок перетину використовується горизонтальність або вертикальність сторін вікна. На рис. 51 показані конкретні ситуації та формули для обчислення координат точки перетину для вказаних випадків (x_0 та y_0 - координати границь вікна). Аналогічним способом обчислюються координати для решти границь вікна.

$$y_Q = y_s + \frac{x_0 - x_s}{x_p - x_s}(y_p - y_s)$$

$$x_Q = x_s + \frac{y_0 - y_s}{y_p - y_s}(x_p - x_s)$$

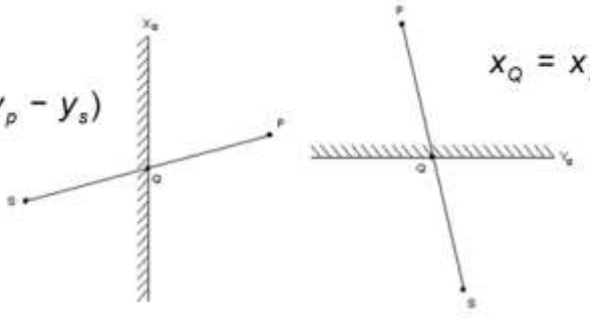


Рисунок 51 - Обчислення координат точок перетину

Розгляд точки перетину представляє окремий випадок, оскільки границя вікна вважається такою, що належить вікну, то з одного боку зовнішній відрізок – відрізок із точкою перетину *(CP)* вимагає подальшого дослідження, тому що *code_C* & *code_P* = 0, а з іншого боку відрізок *PD* можна вважати видимим і залишити його.

Заключний етап алгоритму зводиться до того, що Коеном і Сазерлендом запропоновано замінити (перенести) кінцеву точку з ненульовим кодом кінця відрізка на точку, що лежить на стороні вікна (перенос точок P_1 -> *R* ->*S* ; P_2 -> *U* -> *T*) (рис. 52, зліва). Відрізок *ST* залишається як видимий. В іншому випадку перенос точок має відбутися на продовженні сторони вікна (вертикальній або горизонтальній границі вікна) (рис. 52, справа). Відбувається перенос P_1->*Q* ; P_2 -> *R* -> *S*. Далі перевіряються коди *Q* та *S.* В однакових позиціях кодів є одиниці. Це означає, що відрізок відкидається, оскільки знаходиться поза вікном.

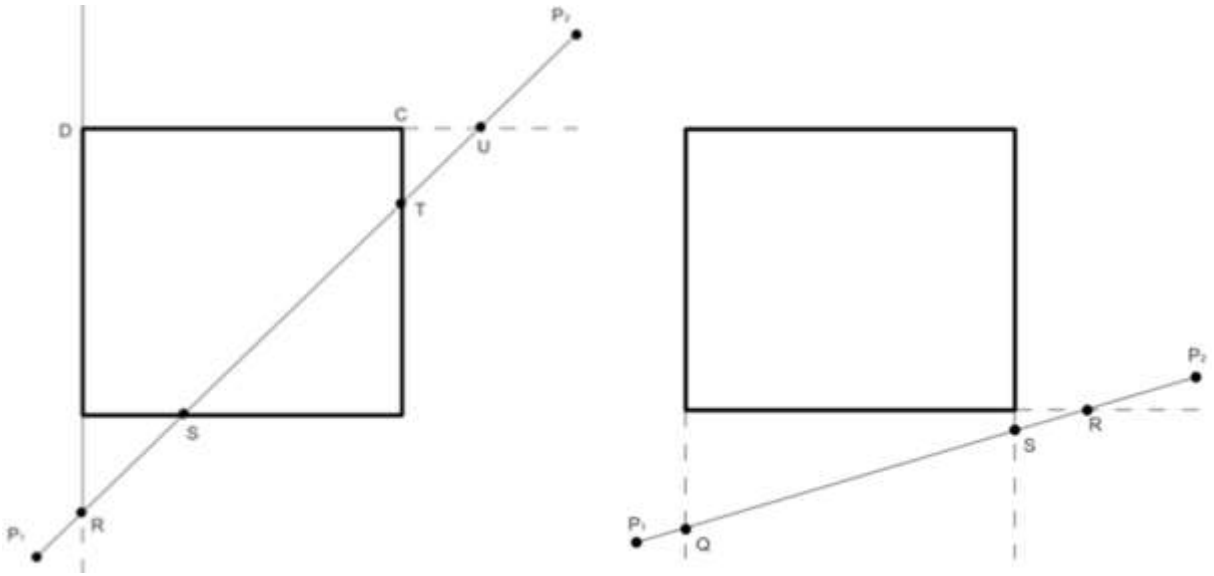


Рисунок 52 - Перенос точок

В багатьох випадках потрібне відсікання довільним многокутником, наприклад в алгоритмах усунення невидимих частин сцени. Алгоритм Кіруса-Бека реалізує відсікання довільним опуклим многокутником-вікном і базується на методі визначення орієнтації лінії, яка містить відрізок, що

відсікається, по відношенню до сторони многокутника, а також на методі визначення місцезнаходження точки відрізка відносно вікна – всередині, на границі або зовні вікна. Для цього в алгоритмі Кируся-Бека використовується вектор внутрішньої нормалі до ребра вікна. Розглянемо опуклу двовимірну область P, тобто опуклий многокутник. Внутрішньою нормаллю *n* до сторони вікна називається нормаль, яка направлена в бік області, що складає вікно відсікання. Як видно з рис. 53, внутрішня нормаль *n* у точці *a*, що лежить на границі, задовольняє умові

$$(n \cdot (b - a)) \geq 0,$$

де *b* – будь-яка інша точка границі опуклої області *P*.

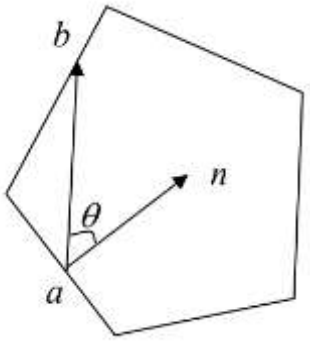


Рисунок 53 - Визначення місцезнаходження точки відносно вікна відсікання в алгоритмі Кіруса-Бека

Скалярний добуток $(n \cdot (b - a))$ для внутрішньої нормалі *n* невід'ємний тому, що $\theta \in [0, 2\pi]$, тобто $\cos\theta \geq 0$. Внутрішні нормалі можна обчислювати іншим способом, що полягає в обході полігона за годинниковою стрілкою і виборі нормалі, які направлені вправо від кожного ребра.

Нехай n_i – внутрішня нормаль до *i*-ї граничної лінії многокутного вікна, а $V = b - a$ – вектор, що визначає орієнтацію відрізка *ab*, який відсікається вікном *P*. Тоді орієнтація відрізка *ab* відносно *i*-ї сторони вікна визначається знаком скалярного добутку $r_i = (n_i \cdot V)$:

− при $r_i < 0$ відрізок *ab* направлений з внутрішньої сторони на зовнішню сторону *i*-ї граничної лінії вікна (рис. 54, а).

− при $r_i = 0$ точки *a, b* або збігаються (відрізок вироджується в точку) або відрізок *ab* паралельний *i*-й стороні вікна (рис. 54, б).

− при $r_i > 0$ відрізок *ab* направлений із зовнішньої сторони на внутрішню сторону *i*-ї граничної лінії вікна (рис. 54, в).

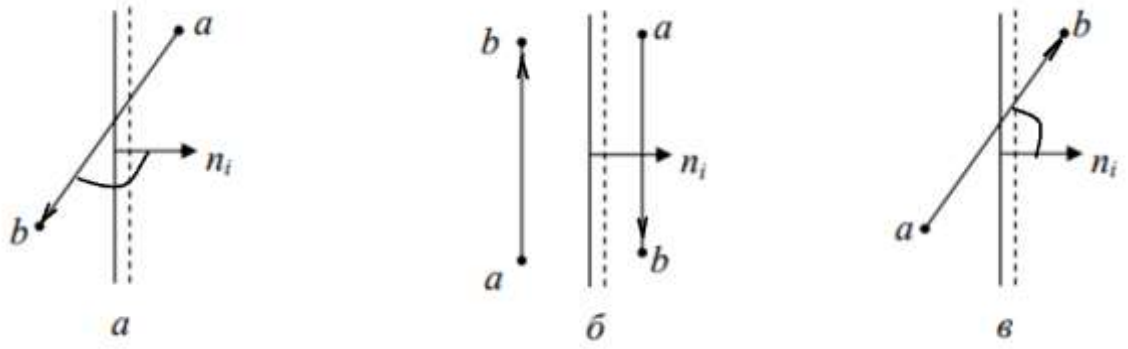


Рисунок 54 - Орієнтація відрізка *ab* відносно сторони вікна: а – зсередини назовні; б – паралельно границі; в – ззовні всередину

Наступна задача - це визначення розміщення точки *V(t).* Для визначення розміщення точки *V(t)* відрізка *ab* відносно *i*-ї сторони вікна розглянемо скалярний добуток внутрішньої нормалі n_i на вектор

$$Qi = V(t) - Pi,$$

що починається в початковій точці *Pi* ребра вікна і закінчується в деякій точці *V(t)* на відрізку *ab*, а саме

$$q_i = (n_i \cdot Q_i),\ i = 1, 2, \ldots$$

Аналогічно попередньому маємо (рис. 55):

− якщо $q_i < 0$, то точка *V(t)* лежить з зовнішньої сторони границі;

− якщо $q_i = 0$, то точка *V(t)* лежить на лінії границі вікна;

− якщо $q_i > 0$, то точка *V(t)* лежить із внутрішньої сторони *i*-ї границі вікна.

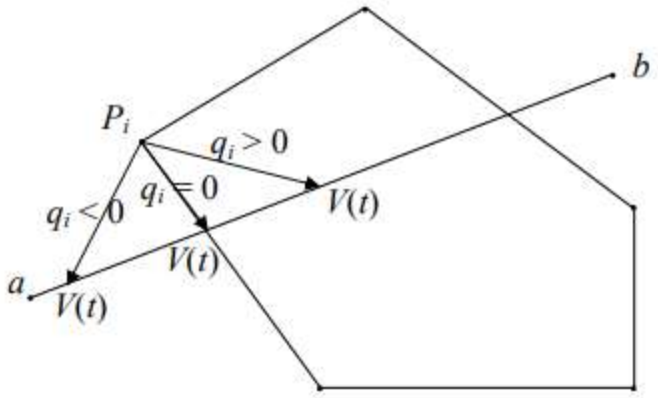


Рисунок 54 - Розміщення точки відносно границі вікна

Згадаємо векторно-параметричне рівняння відрізка *ab* $V(t)=a+(b-a)t$, $t \in [0,1]$.

Оскільки многокутник опуклий, то може бути дві точки перетину відрізка *ab* із вікном *P,* і необхідно знайти два значення параметра *t*, що відповідають початковій і кінцевій точкам видимої частини відрізка. З умови належності точки *V(t)* границі вікна, тобто з умови $q_i = 0$ маємо

$$(n_i \cdot (a + (b - a)t - P_i))) = 0,$$

або

$$(n_i \cdot (a - P_i)) + n_i \cdot (b - a)t) = 0$$

звідки

$$t = -(n_i \cdot (a - P_i))/(n_i \cdot (b - a)) = -(n_i \cdot Q_i / n_i \cdot V) = -q_i / r_i .$$

Значення параметрів t_0, t_1 точок перетину відрізка з вікном ініціалізуємо значеннями 0, 1, що відповідають початку і кінцю відрізка *ab*. В циклі по *i* для кожної сторони вікна відсікання обчислюємо значення скалярних добутків $q_i = n_i \cdot (a - P_i)$ та $r_i = n_i \cdot (b - a)$.

Якщо $r_i = 0$, то відрізок *ab* або вироджується в точку, або паралельний *i-й* стороні вікна. При цьому проаналізуємо значення $q_i = n_i \cdot (a - P_i)$. Якщо $q_i < 0$, то відрізок знаходиться ззовні вікна і відсікання закінчено, інакше переходимо до наступної сторони вікна.

Якщо $r_i \neq 0$, то обчислюємо значення t для точки перетину відрізка ab з i-ю границею вікна. Оскільки точки відрізка ab відповідають значенням $t \in [0, 1]$, то значення, що $t \notin [0, 1]$ відкидаємо.

При $r_i < 0$ лінія ab направлена з внутрішньої сторони i-граничної лінії на зовнішню, і знайдені значення t визначатимуть кінцеву точку видимої частини відрізка ab. В цьому випадку визначається $min\ t$ серед всіх одержаних t.

$$t_1 = \min\left(\left\{-\frac{q_i}{r_i}\,\middle|\, r_i < 0, i = 1,2,\ldots\right\}, 1\right).$$

Якщо поточне значення t_1 стане меншим за t_0, то відрізок ab відкидається, оскільки не виконується умова $t_0 \leq t_1$. При $r_i > 0$ лінія ab направлена із зовнішньої сторони граничної лінії на внутрішню, тому знайдені значення t визначатимуть початкову точку видимої частини відрізка ab. В цьому випадку визначається $max\ t$ серед всіх одержаних t. Воно дає значення t_0 для початкової точки видимої частини відрізка.

$$t_0 = \max\left(\left\{-\frac{q_i}{r_i}\,\middle|\, r_i > 0, i = 1,2,\ldots\right\}, 0\right).$$

Якщо поточне значення t_0 стане більшим за t_1, то відрізок відсікається.

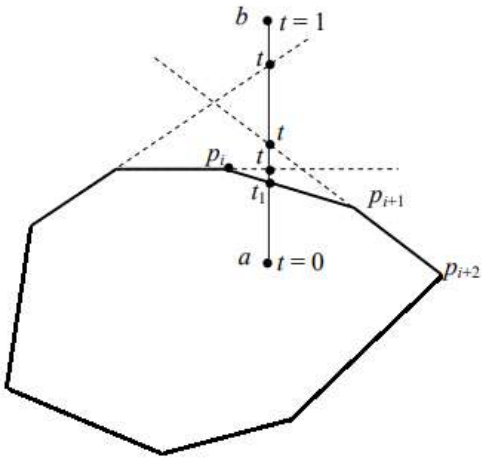


Рисунок 55 - Визначення кінцевої точки *V(t1)*

На заключному етапі алгоритму значення t_0, t_1 використовуються для знаходження координат точок перетину відрізка *ab* з вікном *P*, тобто для побудови видимої частини відрізка. При цьому, якщо $t_0 = 0$, то початкова точка залишається та сама, й обчислення $V(t_0)$ не потрібні. Аналогічно при $t_1 = 1$ кінцева точка – *b*. Обчислення $V(t_1)$ теж не потрібні.

Для роботи з алгоритмом Кірус-Бека слід переконатися, що многокутник опуклий. Факт опуклості многокутника можна визначити за знаком векторного добутку його суміжних сторін.

Аналізуються знаки такого добутку:

- всі добутки дорівнюють нулю - многокутник виродився у відрізок;

- є як додатні, так і від'ємні знаки - многокутник неопуклий;

- всі знаки невід'ємні - многокутника опуклий, його внутрішні нормалі орієнтовані вліво від напрямку обходу його контуру;

- всі знаки від'ємні - многокутник опуклий, його внутрішні нормалі орієнтовані вправо від напрямку обходу його контуру.

В алгоритмі Ліанга-Барскі знайдено спосіб відбраковувати параметр *t* на стадії його обчислення, і таким чином малювати або відкидати частини відрізка негайно після обчислення чергового значення *t*.

24.Проблема візуалізації тривимірних об’єктів.

Задачі видалення невидимих ліній та поверхонь

У сфері комп’ютерної тривимірної графіки головним напрямком діяльності завжди була задача підвищення реалістичності зображення. Це проявляється у підвищенні складності і деталізації тривимірних сцен, у застосуванні різних психофізичних прийомів, що створюють ефект присутності, а також за допомогою застосування спецефектів, таких, наприклад, як туман - системи, що складається із частинок або тіні.

Задачі візуалізації тривимірних сцен виникають в системах автоматизованого проєктування, програмних додатках для моделювання фізичних процесів, засобах комп’ютерної анімації та віртуальної реальності. З іншого боку, при відображенні тривимірної сцени на екрані деякі з об’єктів сцени можуть затулити інші об’єкти. Ці затулені частини об’єктів є невидимими, і не повинні відтворюватись на екрані, або повинні відтворюватись інакше, ніж видимі частини, наприклад, пунктиром. Якщо цього не робити, то зображення буде виглядати неправильно.

Тривимірний об’єкт може бути зображений різними способами. Способи візуалізації можна досить умовно розділити на кілька рівнів за зростанням складності:

- каркасний – відтворюються всі вершини і ребра об’єкта – такий спосіб добре висвітлює внутрішню структуру об’єкта, але дає найнереалістичніше зображення;
- з видаленням невидимих вершин і ребер;
- видалення невидимих граней об’єкта + різні види зафарбовування видимих граней (однотонна, з урахуванням освітлення, з імітацією плавних переходів між гранями, з визначенням затінених ділянок);

- видалення невидимих граней об’єкта + різні види деталізації поверхонь (нанесення текстур, моделювання мікрорельєфу поверхні).

Додаткові способи підвищення реалістичності зображень полягають у застосуванні перспективних перетворень, тобто використання при візуалізації перспективної проекції, при якій більш віддалені від спостерігача об’єкти будуть мати на екрані менший розмір, та моделюванні напівпрозорого середовища, в яке «занурені» об’єкти сцени.

Алгоритми видалення невидимих ліній і поверхонь в першу чергу визначають лінії ребер, поверхонь чи об’ємів, які видимі або невидимі для спостерігача, що знаходиться в заданій точці простору. Необхідність коректного видалення невидимих ліній, ребер, поверхонь чи об’ємів проілюстрована на рис.56. Те, що зображено ліворуч, може бути інтерпретовано двома випадками – посередині та праворуч. Каркасне креслення представляє тривимірний об’єкт у вигляді зображення його ребер (враховуючи правила зображення видимих та невидимих ліній). Так, рис.56 можна інтерпретувати по-різному: як вид куба згори зліва або знизу справа. Для цього достатньо примружитися і перефокусувати погляд. Видалення тих ліній або поверхонь, які невидимі з відповідною точки зору, дозволяють позбутися неоднозначності.

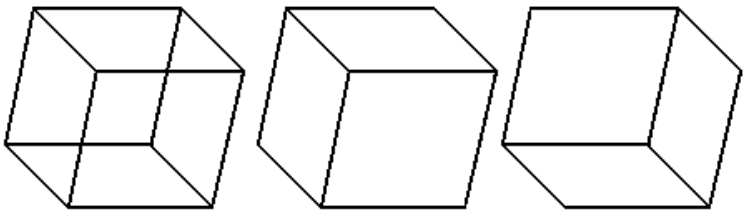


Рисунок 56 - Неоднозначність сприйняття моделі об’єкта

У загальному випадку необхідно вирішити такі задачі як:

- видалення всіх нелицьових поверхонь (граней);
- видалення всіх видимих поверхонь, що закриваються іншими об'єктами.

Якщо грані є границею тіла (або декількох тіл), то для кожної з них можна визначити вектор зовнішньої нормалі. Нормалі до граней (рис.57) A_1 *і* A_2 дивляться в бік спостерігача (спостерігач знаходиться в додатному напівпросторі по відношенню до площини, що проходить через відповідну грань). Такі границі називаються лицьовими (front-faced). Для граней A_3 *і* A_4 нормалі спрямовані від спостерігача, їх називають нелицьовими (back-faced).

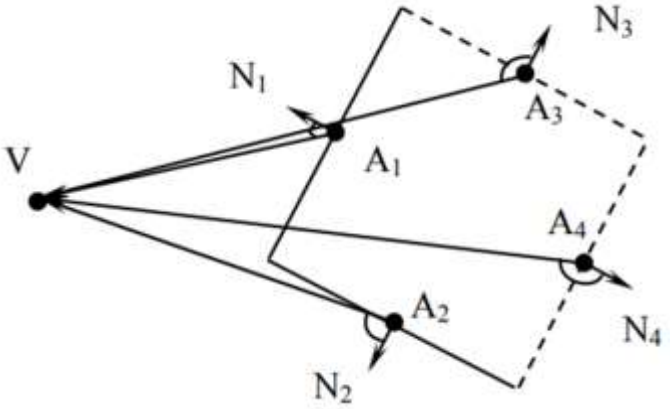


Рисунок 56 - Лицьові та нелицьові грані

Визначити, чи є грань нелицьовою, можна за знаком скалярного добутку нормалі *N* і вектора з будь-якої точки на границі A на спостерігача *V*. Вираз (скалярний добуток) $N \cdot (V - A) < 0$ означає, що косинус кута між векторами менше *0*, і кут більше 90^o, а значить грань нелицьова. І навпаки, для лицьових граней скалярний добуток буде додатнім.

Властивості нелицьових граней:

1. у випадку, коли ребро є границею тіла (або декількох тіл), то жодна з нелицьових граней не може бути видимою, навіть частково - кожна з них завжди буде закриватися від спостерігача лицьовими гранями;

2. при визначенні видимості все нелицьові грані можна завжди відкидати, що скорочує число розглянутих граней приблизно вдвічі;

3. коли вся сцена складається з одного опуклого об’єкта, то всі лицьові грані і тільки вони будуть видимими, причому повністю.

Складність задачі видалення невидимих ліній і поверхонь призвела до появи великої кількості різних способів її вирішення. Методи видалення невидимих об’єктів (частин сцени) можна класифікувати за наступними критеріями:

1. за видом частин, що видаляються: видалення невидимих ліній, ребер, поверхонь, об’ємів;

2. за порядком обробки елементів сцени: видалення в довільному порядку та в порядку, що визначається процесом візуалізації;

3. за системою координат :

- алгоритми, що оперують у просторі об’єктів, коли кожна з N граней об’єкта порівнюється з іншими $N-1$ гранями (об’єм обчислень N має квадратичне зростання N^2). Такі алгоритми, мають справу з фізичною системою координат, у якій описані ці об’єкти. При цьому виходять досить точні результати, обмежені лише точністю обчислень. Отримані зображення можна вільно збільшувати в багато разів. Алгоритми, що працюють у об’єктному просторі, особливо корисні в тих застосунках, де необхідна висока точність. Такі алгоритми ще називають 3D-алгоритмами.

- алгоритми, що оперують у просторі зображення (екрана), коли для кожного пікселя зображення визначається, яка з N граней об’єкта є видимою (при роздільній здатності екрана $M \times M$ обсяг обчислень оцінюється як $M^2 \times N$). Алгоритми ж, що працюють у просторі зображення, оперують із системою координат того екрана, на якому відбувається візуалізація об’єктів. При цьому

точність обчислень обмежена роздільною здатністю екрана. Такі алгоритми ще називають 2D-алгоритмами.

- алгоритми, що формують список пріоритетів - оперують по черзі в обох згаданих системах координат.

25.Моделі відбивання і заломлення світла

До основних прийомів для створення реалістичних сцен відносять врахування ефектів відбивання та заломлення променів світла, визначення затіненості об’єкта, нанесення текстур та різні способи зафарбовування, імітація мікрорельєфа та текстур, тощо.

Після видалення прихованих поверхонь для підвищення реалістичності зображення всі видимі грані об’єктів необхідно зафарбувати. Існують різні моделі зафарбовування поверхонь, які враховують розташування джерел світла, характеристики матеріалу поверхні, взаємне розташування граней.

У комп’ютерній графіці розглядаються об’єкти, які по відношенню до світла мають наступні властивості:

• випромінюють світло;

• відбивають і поглинають світло;

• пропускають світло крізь себе.

Кожну з цих властивостей можна описати деяким набором характеристик.

Випромінювання джерел світла можна охарактеризувати:

- інтенсивністю;
- спектральним складом (кольором);
- геометричними розмірами джерела (точковий, протяжний);
- напрямом випромінювання (на всі боки, уздовж вузького променя, конусом).

Відбивання світла може бути декількох видів - дзеркальне, не ідеальне дзеркальне, дифузне, зворотне.

Дзеркальне відбивання (рис. 57, 1) - падаючий на поверхню промінь (*l*) і відбитий промінь (*s*) лежать в одній площині, причому кут падіння світлового променя дорівнює куту відбиття. Колір відбитого променя збігається з

кольором падаючого променя, тобто власного кольору у дзеркальної поверхні немає. Спостерігач буде бачити відбитий промінь, тільки якщо кут γ між напрямком відбитого променя (*s*) і напрямом на спостерігача (*v*) дорівнює нулю.

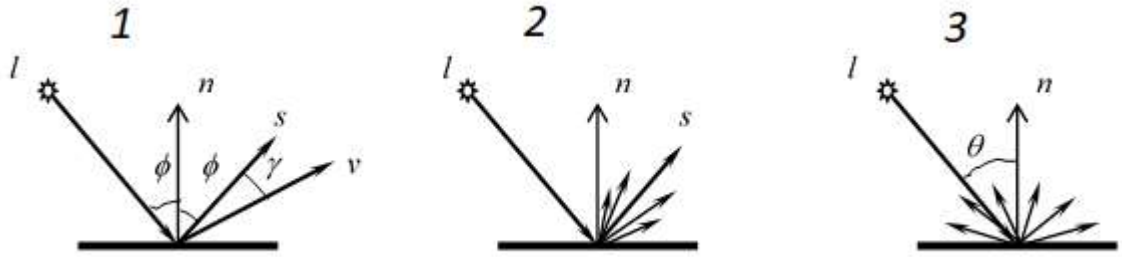


Рисунок 57 - Відбивання світла

Не ідеальне дзеркальне відбивання відбувається при падінні світла на злегка шорстку поверхню (рис. 57, 2)**.** У цьому випадку виникає кілька відбитих променів, що розсіюються по різних напрямах. Зона розсіювання залежить від якості полірування і, як правило, симетрична щодо направлення ідеально відбитого променя, а її форма може бути описана деяким законом розподілу. Однією з найпростіших моделей розподілу, найбільш часто використовуваною в комп'ютерній графіці, є модель Фонга:

$$I_s = I\, k_s \cos^P \alpha,$$

де I_s – інтенсивність відбитого променя, I – інтенсивність падаючого променя, k_s – коефіцієнт пропорціональності, p=1…200 – показник якості полірування поверхні, α – кут відхилення від напрямку ідеально відбитого променя.

Сильно матовим поверхням властиво *дифузне (розсіяне)* відбивання, при якому падаючий промінь рівномірно розсіюється в усі сторони (рис. 57, 3)**.** Матова поверхня має свій колір - спостережуваний колір матової поверхні визначається комбінацією власного кольору поверхні і кольору падаючого

променя. Інтенсивність дифузно відбитого світла не залежить від положення спостерігача.

Дифузне відбивання описується законом Ламберта:

$$I_d = I\, k_d \cos\theta,$$

де I_d – інтенсивність дифузно-відбитого променя, I – інтенсивність падаючого променя, k_d – коефіцієнт, який враховує властивості матеріалу поверхні, θ – кут між нормаллю (n) до поверхні і напрямом падаючого променя (l).

Поверхні можуть мати властивості направленого і дифузного пропускання. Направлене пропускання світла відбувається крізь прозорі речовини (наприклад, скло). Через них звичайно добре видно предмети, навіть незважаючи на те, що промені світла, як правило, заломлюються, тобто відхиляються від початкового напрямку. Дифузне пропускання світла відбувається крізь матеріали, що просвічуються (наприклад, замерзле скло), в яких поверхневі або внутрішні неоднорідності призводять до безладного перемішування світлових променів. Тому коли предмет розглядається через прозору речовину, його контури є розмитими.

При проходженні світлових променів через напівпрозоре середовище відбувається їх заломлення. Падаючий і заломлені промені лежать в одній площині, причому:

$$\eta_1 \sin\alpha_1 = \eta_2 \sin\alpha_2,$$

де η_1 и η_2 – показники переломлення двох середовищ, α_1 и α_2 – кути падіння і переломлення світлового променя (рис. 58).

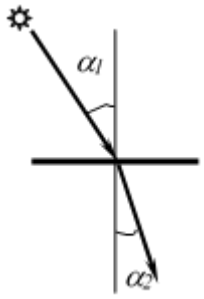


Рисунок 58 - Заломлення світла

Показник заломлення залежить не тільки від матеріалу середовища, а й від довжини світлової хвилі: чим менше довжина хвилі (фіолетові кольори), тим сильніше відхиляється промінь. Існує і дифузне заломлення, коли заломлений промінь розсіюється рівномірно в усі сторони, але в комп'ютерній графіці воно використовується рідко через складність обчислень. При розгляді напівпрозорих матеріалів необхідно враховувати і відбите, і заломлене світло.

26.Моделі зафарбовування

Необхідність в описі просторових об’єктів виникає при вирішенні багатьох задач комп’ютерної графіки. У загальному випадку реально існуючий об’єкт не може, звичайно, в точності відповідати своєму опису. Для цього треба нескінченне число трійок координат (x, y, z) - по одній для кожної точки поверхні об’єкту.

Об’єкти сцени, вузли, конструкції мають різні геометричні характеристики. Геометрична модель - сукупність відомостей, які однозначно визначають форму геометричного об’єкта. Геометричні моделі можуть бути представлені сукупністю рівнянь ліній і поверхонь, алгебро-логічними співвідношеннями, графами, списками, таблицями, описами спеціальними графічними мовами.

Об’єкт задається множиною обмежуючих граней і нормалями до кожної із цих граней, що спрямовані від об’єкта. Кожна грань задається циклом обмежуючих її ребер; кожне ребро - парою обмежуючих його точок (вершин); кожна точка - трійкою координат в тривимірному просторі.

У багатьох випадках для апроксимації складних поверхонь використовуються многогранники, але форма граней може бути різною. Тому програміст повинен сам подбати про те, щоб багатогранник був описаний правильно. У цьому випадку оптимальним виходом з положення є використання трикутників. У сучасній графіці це, мабуть, найпоширеніший підхід.

Сукупність граней, представлених плоскими многокутниками і обмежених прямолінійними ребрами, утворює полігональну сітку. Грані можуть мати будь-яку форму, але в переважній більшості випадків використовуються опуклі многокутники з мінімальною кількістю вершин, оскільки так обчислення виконується простіше.

Основним недоліком полігональної сітки є приблизність подання форми об’єкта при описі викривлених поверхонь. Для поліпшення кусочно-лінійної апроксимації таких об’єктів збільшують кількість граней, що призводить до додаткових витрат пам’яті та збільшенню обсягу обчислень (рис.59).

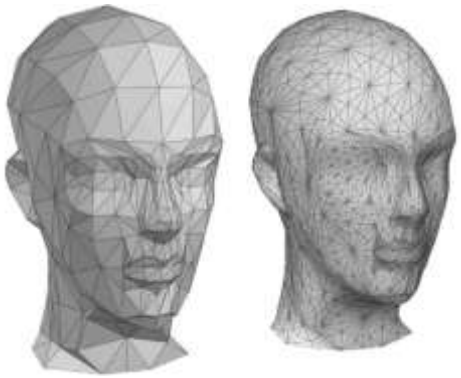


Рисунок 59 - Полігональне представлення викривлених поверхонь

Існує три основні способи зафарбовування об’єктів, заданих полігональними сітками. Розглянемо найпростіший із них. Якщо припустити, що джерело світла знаходиться в нескінченності, то промені світла, що падають на поверхню, паралельні між собою. Також, якщо до цього додати умову, що спостерігач перебуває в нескінченно віддаленій точці, то ефектом ослаблення світла зі збільшенням відстані від джерела можна знехтувати. Крім того, вектори, які спрямовані від різних точок поверхні до спостерігача, також будуть паралельні. При виконанні всіх цих умов виходить, що грань у всіх точках має однакову інтенсивність освітлення, тому вона зафарбовується одним кольором. Таке зафарбовування називається однотонним або плоским (flat).

Якщо ми апроксимуємо деяку гладку поверхню многогранником, то при плоскому зафарбовуванні неминуче стають помітними ребра, оскільки сусідні грані з різними напрямками нормалей мають різний колір. Різкість колірних переходів додатково збільшується через виникнення так званих смуг Маха,

коли світла грань поблизу її границі з більш темною гранню суб'єктивно сприймається оком людини ще світліше, ніж вона є насправді. На темній грані поблизу її границі з більш світлою гранню, навпаки, створюється ілюзія більш темної смуги.

Для його усунення при використанні цього способу зафарбовування можна лише збільшити число граней багатогранника, що призводить до збільшення обчислювальної складності алгоритму. Інший спосіб прибирання цього ефекта для згладжування смуг Маха – застосувати зафарбовування методом Гуро.

Цей метод полягає в тому, що використовуються не нормалі до граней (многокутників), а нормалі поверхні, яка апроксимується, що побудовані в вершинах многогранника (того, що апроксимує об'єкт). Після цього обчислюються інтенсивності в вершинах, а потім у всіх внутрішніх точках багатокутника виконується білінійна інтерполяція інтенсивності.

Процес зафарбовування по методу Гуро здійснюється в чотири етапи:

1. Обчислюються нормалі по всіх полігонах.

2. Визначаються нормалі у вершинах шляхом усереднення нормалей по всіх полігональним гранях, яким належить вершина (рис. 60).

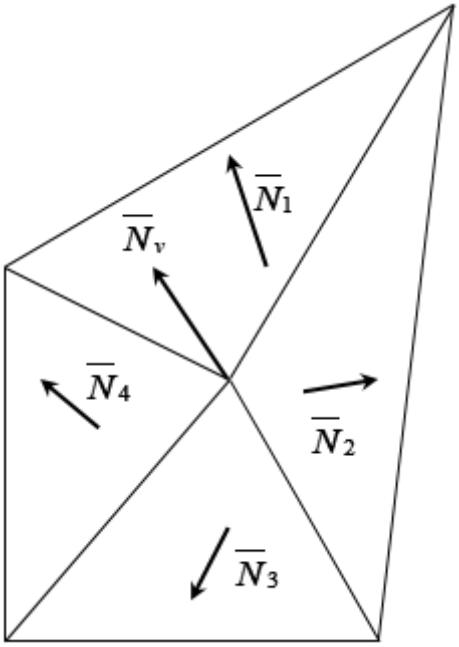


Рисунок 60 - Нормалі до вершин

$$\overline{N}_v = (\overline{N}_1 + \overline{N}_2 + \overline{N}_3 + \overline{N}_4 +)/4$$

3. Використовуючи нормалі у вершинах і застосовуючи довільний метод зафарбовування, обчислюються значення інтенсивності у вершинах.

4. Кожен многокутник зафарбовується шляхом лінійної інтерполяції значень інтенсивностей у вершинах спочатку уздовж кожного ребра, а потім і між ребрами уздовж кожної лінії (рядка) сканування (рис. 61).

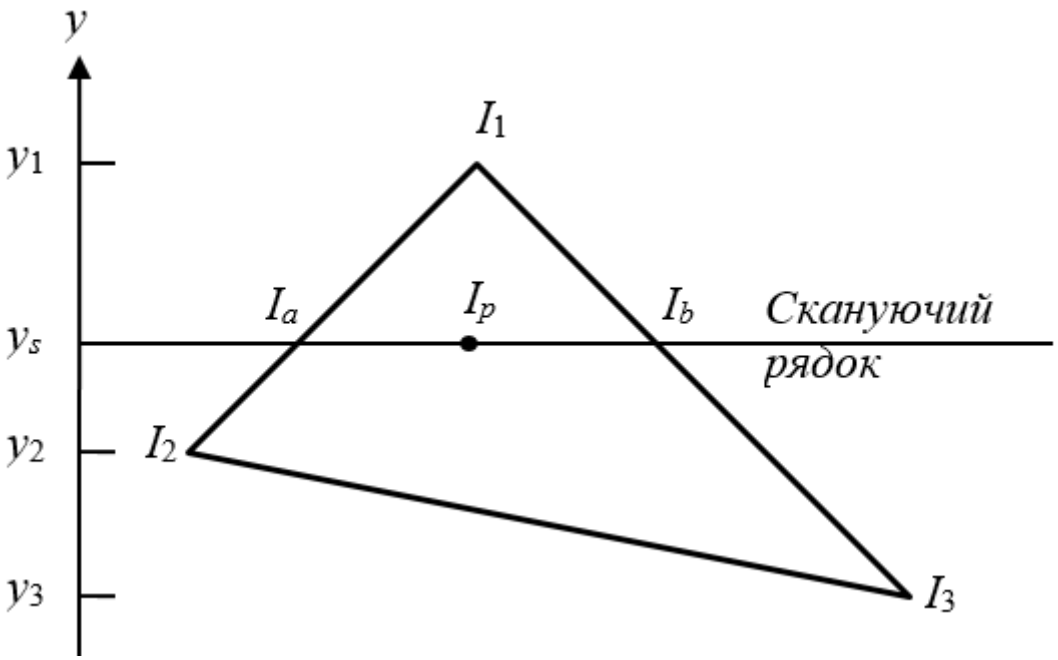


Рисунок 60 - Інтерполяція інтенсивностей по ребрах та по скануючому рядку

$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2};$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3};$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}.$$

Для кольорових об’єктів окремо інтерполюється кожна з компонент кольору. Зафарбовування методом Гуро дозволяє візуально згладити переходи між гранями, проте в деяких випадках не вдається уникнути появи смуг Маха.

До недоліків методу Гуро слід віднести те, що він добре працює тільки з дифузною моделлю відбиття світла. Форма відблисків на поверхні і їх розташування не можуть бути адекватно відтворені при інтерполяції на многокутниках. Також є проблема побудови нормалей до поверхні. В

алгоритмі Гуро нормаль у вершині многогранника обчислюється шляхом усереднення нормалей до граней, що примикають до цієї вершини. Така побудова сильно залежить від характеру розбиття поверхні на полігони.

За методом Фонга обчислюються усереднені значення нормалей у вершинах, однак при зафарбовуванні поверхні грані інтерполюються не значення інтенсивності, а значення нормалей в кожній точці, і лише потім обчислюється інтенсивність. Значення нормалі вздовж рядка інтерполюється між значеннями нормалей на ребрах для даного рядка. Значення нормалей на ребрах визначається як результат інтерполяції між вершинами. Значення ж нормалей у вершинах є результатом усереднення, як і у вище розглянутому методі Гуро. Значення нормалі для кожного з пікселів рядка використовується для обчислень з тієї чи іншої моделі освітлення. Метод Фонга дає кращий вигляд кривизни поверхні, що апроксимується, тому дзеркальні властивості поверхні відтворюються набагато краще. Цей метод дозволяє усунути ряд недоліків методу Гуро, але не всі. Зокрема, ефект смуг Маха в окремих випадках в методі Фонга буває навіть сильнішим, хоча в переважній більшості випадків апроксимація Фонга дає кращі результати.

Якщо порівняти все ці три метода зафарбовування (рис. 61) при генеруванні зображення моделі, то можна відмітити, що однотонне зафарбовування дає зображення ребристої поверхні з дуже контрастними переходами від однієї грані до іншої. Модель Гуро дає більш гладке зображення, але в районі відблисків чітко спостерігаються лінії ребер, хоча і згладжені. Зафарбовування за Фонгом є найбільш гладким, дзеркальні відблиски мають досить реалістичну форму.

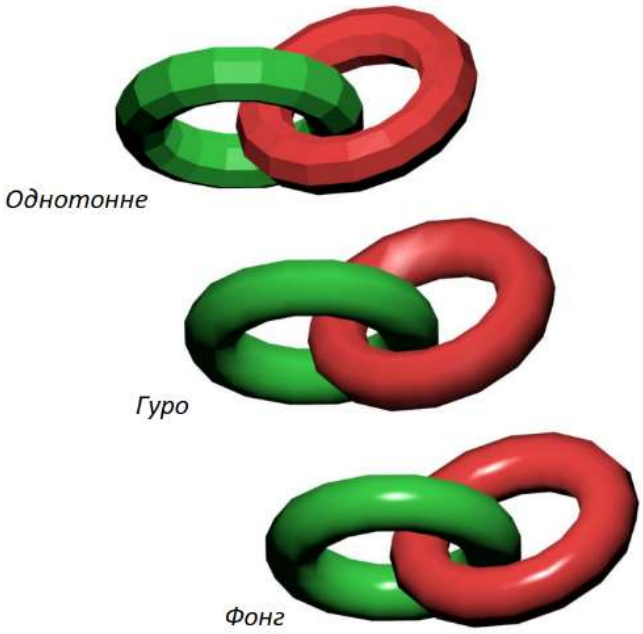


Рисунок 61 - Три варіанти зафарбовування

27. Деталізації поверхонь за допомогою текстур

Текстура являє собою масив растрових візерунків. Візерунки накладаються на поверхню тривимірного об’єкта на етапі тонування. Візерунок (текстура) задається функцією інтенсивності, яка використовується або для модифікації кольору основної поверхні, що визначається з урахуванням освітленості, або для призначення для основної поверхні кольору тільки текстури.

Текстура не змінює координат точок поверхні і не робить її більш рельєфною, а просто «розфарбовує» поверхню.

Види текстур:

1. Прозорі текстури.
2. Напівпрозорі текстури.
3. Циклічні текстури .
4. Динамічні текстури.
5. Текстури з мультироздільною здатністю.

Основні проблеми виникають при накладенні плоскої текстури на тривимірні криві поверхні об’єктів (наприклад, на сферу). Текстура може накладатися на поверхню об’єкта різними способами. Кожен з них пов’язаний з окремим математичним алгоритмом, визначальним щодо того, яким чином текстура покриває поверхню об’єкта. У найпростішому випадку (плоске накладення) текстура в більшій чи меншій мірі розтягується, щоб заповнити всю поверхню. При більш складних алгоритмах - циліндричному, сферичному і хромовому (із відблисками) покритті - об’єкт фактично «загортається» у текстуру (рис.62).

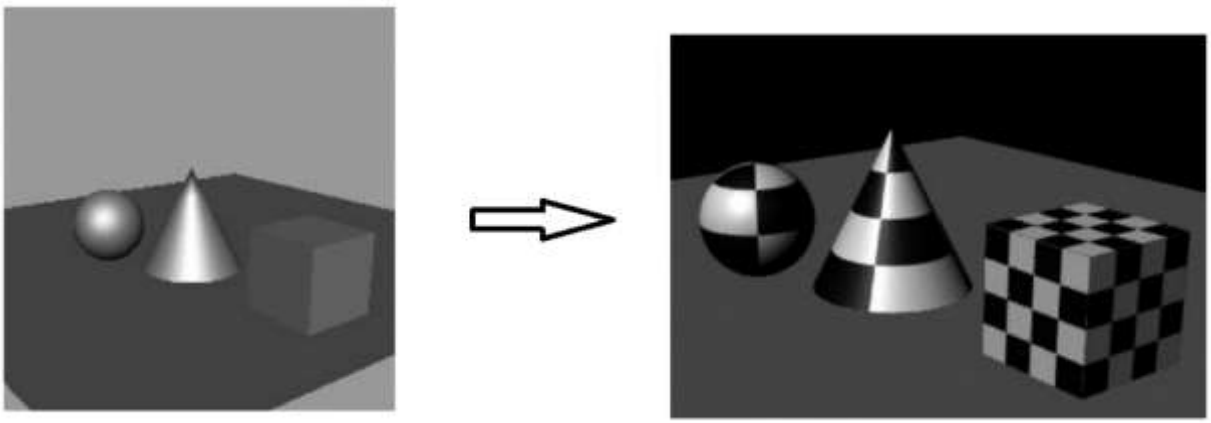


Рисунок 62 - Накладання текстур

Текстури мають такий же недолік, що і всі растрові зображення - для їх зберігання може знадобитися великий об’єм пам’яті. Для економії пам’яті зазвичай застосовують блочне текстурування: текстура являє собою не всю грань цілком, а лише окремий фрагмент, який циклічно повторюється в грані (циклічні текстури). В якості текстур можуть використовуватися не тільки прості візерунки, але й складні малюнки. Також, текстури можуть бути і анімованими. Для імітації мікрорельєфу поверхні часто використовується технологія внесення збурень в нормаль до поверхні об’єкта (безпосередньо перед тонуванням поверхні). Результат застосування цього методу буде досить реалістичним.

28. Алгоритми тріангуляції

Генерування об’ємних зображень за допомогою полігональної сітки представляє собою складну обчислювальну задачу, у зв’язку з чим на практиці виконують її декомпозицію. Складні зображення формують фрагментарно, для чого їх розбивають на складові частини.

Процес розбиття поверхні об’єктів на полігони називається теселяцією – замощення, технологія, за допомогою якої можливо збільшити кількість многокутників в полігоні. При цьому кожен многокутник моделі розбивається на задане число зв’язаних многокутників, які вибудовуються відповідно до загального напрямку поверхні моделі. Таким чином можна спочатку створити просту модель, а потім швидко і просто підвищити її деталізацію. Один із варіантів виконання такої процедури полягає в тому, що графічні акселератори, які мають апаратні засоби для зафарбовування тривимірних об’єктів, видалення невидимих частин, накладення текстур та інші графічні функції, використовують програмно реалізовану теселяцію об’єктів. Тобто відбувається програмна розбивка області та передача отриманої полігональної області в акселератор для подальшої її обробки.

На практиці найбільш часто проводиться розбиття зображень на симплекси (трикутники). Це відбувається за такими причинами:

- трикутник є найпростішим полігоном, вершини якого однозначно задають грань;
- будь-яку область можна гарантовано розбити на трикутники;
- обчислювальна складність алгоритмів розбиття на трикутники значно нижча, ніж при використанні інших полігонів;
- реалізація процедур рендеринга найбільш проста для області, обмеженої трикутником;

- для трикутника легко визначити три його найближчих сусіда, що мають з ним спільні ребра.

Процес розбиття полігональної області зі складною конфігурацією в набір трикутників називається тріангуляцією.

Задачу тріангуляції можна розглядати в двох напрямках: тріангуляція полігональних областей і тріангуляція набору точок. Остання має місце при описі поверхні набором точок та інтенсивностями їхніх кольорів. Поточечний опис поверхонь застосовують у тих випадках, коли поверхня дуже складна і не є гладкою, а детальне представлення багатьох геометричних особливостей цього об'єкту важливо для практики. До поверхонь такого роду можна віднести ділянки ґрунту, форми малих небесних тіл, мікрооб'єкти, зняті за допомогою електронного мікроскопа та інші освіти зі складною формою.

Важливість алгоритмів тріангуляції визначається тим, що вони використовуються в багатьох процедурах комп'ютерної графіки, таких як зафарбовування, видалення невидимих частин, відсікання, формування поверхонь.

Найпростіше рішення задачі тріангуляції полягає в розщепленні полігону уздовж деякої хорди на два полігони і надалі рекурсивному їхньому розбитті до ситуації, коли полігон, що підлягає тріангуляції є трикутником. Але даний спосіб застосовується лише для тріангуляції опуклих полігонів (рис.63).

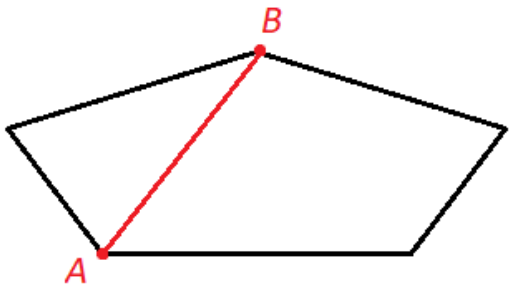


Рисунок 63 - Розділення полігону уздовж хорди на два полігони

Один з підходів до тріангуляції полягає в знаходженні внутрішньої точки області, обмеженою полігоном, і з'єднанні з нею всіх вершин (рис.64).

З огляду на, що тріангуляція є підготовчим етапом рендеринга, то до вибору внутрішньої точки можуть пред'являтися певні вимоги. Так, наприклад, при багатопроцесорній обробці доцільно внутрішню точку вибрати таким чином, щоб площа складових трикутників була приблизно однаковою. У цьому випадку буде забезпечена однакова ступінь обчислювального завантаження апаратури.

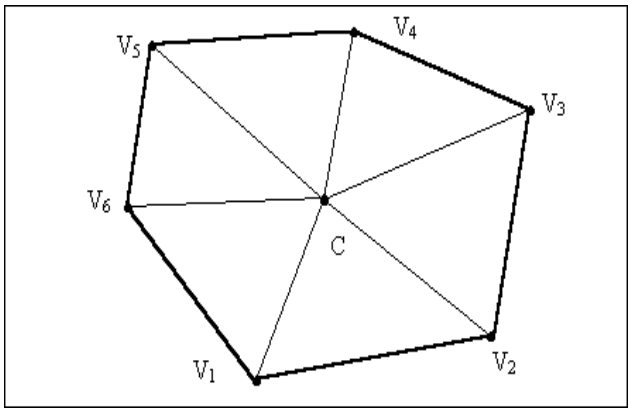


Рисунок 64 - Тріангуляція з використанням внутрішньої точки

Тріангуляція набору точок буде тріангуляцією Делоне, якщо описана окружність для кожного трикутника буде вільна від «сторонніх» точок (точок, що належать іншому трикутнику), тобто всередині її не буде більше ні однієї точки з набору.

Тріангуляція точок на площині за методом Делоне добре збалансована в тому сенсі, що формуються трикутники, які прямують до рівносторонності (рис.65, 1). Тут показано дві окружності, які не містять в собі інших точок. Тріангульоване зображення (рис.65, 2) не може бути віднесено до тріангуляції

Делоне, оскільки всередину області, обмеженої колом, потрапила одна точка іншого трикутника.

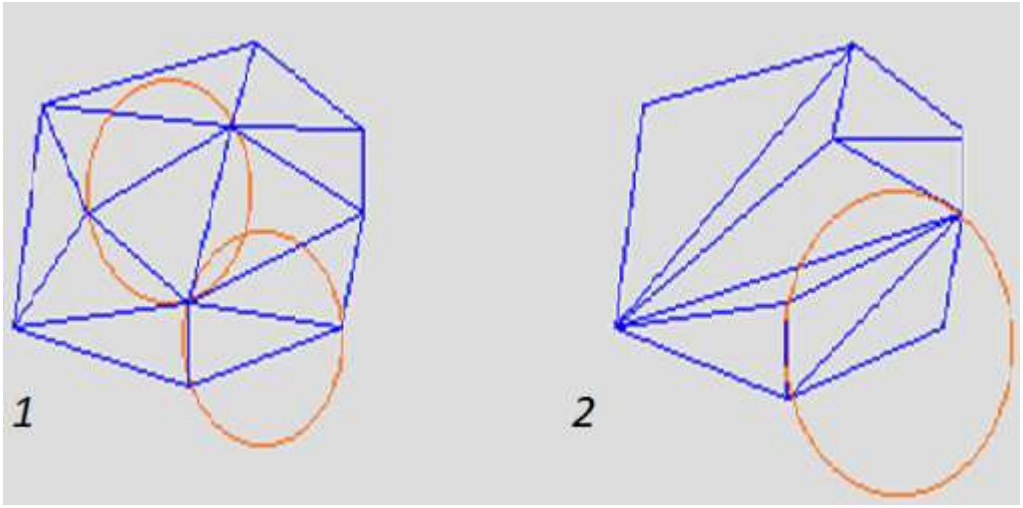


Рисунок 65 - Варіанти тріангуляцій

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Маценко В.Г. Комп’ютерна графіка: Навчальний посібник. – Чернівці: Рута, 2009 – 343 с.

2. Василюк А. С. Комп’ютерна графіка: навчальний посібник /Василюк А. С., Мельникова Н. І.- Львів : Видавництво Львівської політехніки, 2016. - 308 с.

3. Блінова Т.О., Порєв В.М. Комп’ютерна графіка. – К.: Юніор, 2004. – 456 с.

4. Горобець С.М. Основи комп’ютерної графіки: Навч. посібн. – К.:Центр навчальної літератури, 2006. – 232 с.

5. Новожилова М.В., Мироненко В.В. Комп’ютерна графіка. Частина1: Навчально-методичний посібник. – Х.: ХНУБА, 2015.– 60 с.

6. Роджерс Д., Адамс Дж. Математичні основи машинної графіки: Пер. с англ. – М.: Мир, 2001. – 604 с