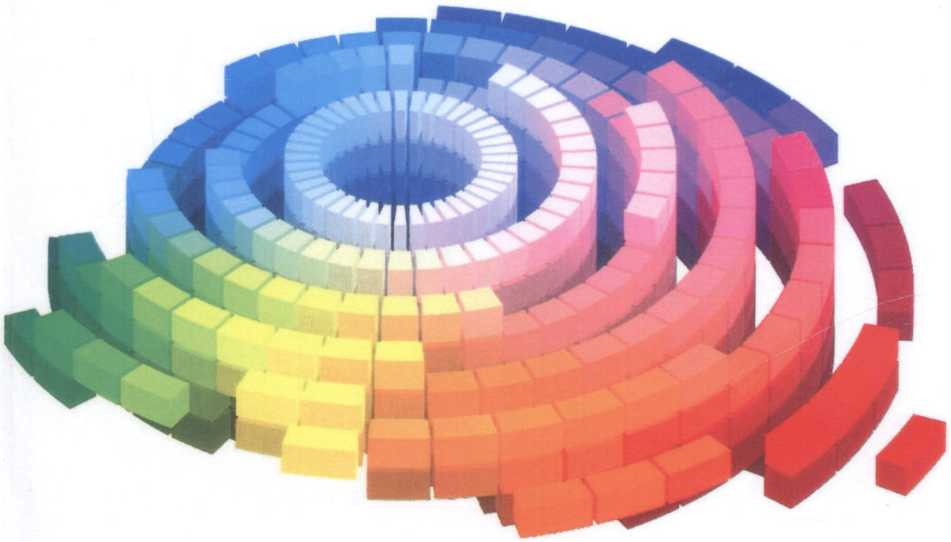
В. Ю. Коцюбинський, О. Ю. Софина, Л. М. Мельник

КОМП'ЮТЕРНА ГРАФІКА



Міністерство освіти і науки України
Вінницький національний технічний університет

В. Ю. Кощюбинський, О. Ю. Софина, Л. М. Мельник

Комп'ютерна графіка

Навчальний посібник

Вінниця
ВНТУ
2015

УДК 004.92 (075)
ББК 32.972.131.2я73
К75

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки (протокол № 4 від 28.11.2013 р.).

Рецензенти:
В. М. Лисогор, доктор технічних наук, професор
О. Д. Азаров, доктор технічних наук, професор
В. М. Дубовой, доктор технічних наук, доцент

Коцюбинський, В. Ю.

К75 Комп'ютерна графіка : навчальний посібник / В. Ю. Коцюбинський, О. Ю. Софина, Л. М. Мельник. – Вінниця : ВНТУ, 2015. – 152 с.

Навчальний посібник присвячений вивченню теоретичних основ комп'ютерної графіки, принципів створення і оброблення різноманітних графічних зображень з використанням комп'ютерної техніки та сучасного програмного забезпечення, розглянуто методи та алгоритми сучасної комп'ютерної графіки.

Посібник призначений для студентів напрямів підготовки «Системна інженерія» та «Комп'ютерна інженерія», для використання при вивченні курсів «Комп'ютерна графіка», «Інженерна та комп'ютерна графіка», а також буде корисний широкому колу фахівців та науковців у галузі інформаційної техніки.

УДК 004.92 (075)
ББК 32.972.131.2я73

ЗМІСТ

ВСТУП

У наш час комп'ютерна графіка є вже цілком сформованою галуззю науки. Існує різноманітне апаратне і програмне забезпечення для створення зображень – від простих креслень до реалістичних образів природних об'єктів. Комп'ютерна графіка використовується майже у всіх наукових і інженерних дисциплінах для наочності сприйняття і передавання інформації. Знання її основ у наш час необхідне будь-якому вченому або інженерові. Комп'ютерна графіка активно використовується в бізнесі, медицині, рекламі, індустрії розваг. Застосування під час ділових нарад демонстраційних слайдів, підготовлених методами комп'ютерної графіки, вважається нормою. У медицині стає звичайним одержання тривимірних зображень внутрішніх органів за даними комп'ютерних томографів. У наші дні телебачення та інші рекламні підприємства часто користуються послугами комп'ютерної графіки і комп'ютерної мультиплікації. Використання комп'ютерної графіки в індустрії розваг охоплює такі області як відеоігри й повнометражні художні фільми. На сьогоднішній день створена велика кількість програм, що дозволяють створювати і редагувати тривимірні сцени й об'єкти.

У даному навчальному посібнику розглянуті основні поняття комп'ютерної графіки, способи збереження та стискання графічних файлів, математичний апарат перетворень та проекцій зображень, растрове подання зображень та базові растрові алгоритми, особливості застосування, створення та математичні основи векторної графіки, розглянуто поняття кольору в комп'ютерній графіці та основні колірні моделі, а також розглянуто основи програмування графіки та наведено багато прикладів програм мовою С++.

При підготовці посібника особлива увага приділялася орієнтації читача на останні досягнення в галузі комп'ютерних технологій, пов'язаних з обробленням графічної інформації. Саме тому більшість практичних прикладів наводиться з використанням сучасних версій програмних продуктів відомих світових виробників.

Посібник призначений для студентів напрямів підготовки «Системна інженерія» та «Комп'ютерна інженерія», для використання при вивченні курсів «Комп'ютерна графіка», «Інженерна та комп'ютерна графіка», а також буде корисний широкому колу фахівців та науковців у галузі інформаційної техніки.

РОЗДІЛ 1 ОСНОВНІ ПОНЯТТЯ

1.1. Вступ в комп'ютерну графіку

Комп'ютерна графіка, яку також називають машинною графікою – область людської діяльності, у якій комп'ютери застосовуються для синтезу зображень реальних або уявних об'єктів і ландшафтів, для обробки зображень і для аналізу зображень як створених штучно, так і отриманих за допомогою відео- і фототехніки.

Виділяють три основні завдання комп'ютерної графіки.

1. **Візуалізація (Rendering).** Історично склалося так, що першим завданням комп'ютерної графіки стала візуалізація або рендерінг – створення зображення об'єктів за їхнім математичним описанням. Візуалізація активно застосовується в комп'ютерних іграх, навчальних та імітаційних програмах, програмах тривимірного проектування, у рекламній і кіноіндустріях. В залежності від способу відображення об'єкта, візуалізацію можна класифікувати так:
   - звичайне зображення – зображення (або декілька зображень) об'єкта в необхідному ракурсі;
   - панорамне або сферичне зображення – позволяє на моніторі інтерактивно обертати об'єкт або віртуальну камеру, але при цьому не змінюється позиція спостерігача в просторі;
   - анімація – зображення об'єкта в динаміці за розробленим для демонстрації сценарієм;
   - віртуальний огляд – дозволяє ознайомитися з об'єктом шляхом довільного переміщення і обертання віртуальної камери в просторі об'єкта і за його межами;
   - інтерактивна взаємодія – охоплює всі можливості віртуального огляду плюс можливості мультимедіа, а також взаємодія з віртуальним світом об'єкта.

2. **Обробка зображень (Image Processing).** Завданням обробки зображення є перетворення одного зображення в інше з метою поліпшення якості зображення, зміни сприйняття зображення або виділення і підкреслення певних деталей зображення. Хоча результати обробки зображень не настільки добре помітні, як результати візуалізації, часто користуються результатами обробки зображень, іноді не помічаючи цього. Наприклад, цифрові фотоапарати для одержання оптимального знімка використовують методи й алгоритми обробки зображень для автоматичного фокусування або автоматичного настроювання колірного балансу. При скануванні документів програмне забезпечення використовує методи обробки зображення для автоматичного поліпшення контрастності отриманого зображення. Методи обробки зображень використовуються для відновлення і ретушування старих фотографій, а також для усунення

дефектів зображень при перекладі старих кінострічок у сучасні цифрові формати.

3. **Розпізнавання зображень (Image Recognition).** Своєю метою алгоритми розпізнавання зображення ставлять побудову опису і знаходження параметрів об'єктів, поданих на вихідному зображенні.

Мета розпізнавання може формулюватися по-різному: виділення окремих елементів (наприклад, букв тексту на зображенні документа або умовних знаків на зображенні карти); класифікація зображень у цілому (наприклад, перевірка того, чи є це зображення певного літального апарата, або встановлення персони за відбитками пальців). Методи класифікації і виділення окремих елементів можуть бути тісно пов'язані між собою. Так, класифікація може бути зроблена на основі структурного аналізу окремих елементів об'єкта або для виділення окремих елементів можна використовувати методи класифікації. Завдання розпізнавання є зворотним щодо візуалізації.

У випадку, якщо користувач може керувати характеристиками об'єктів, то говорять про **інтерактивну комп'ютерну графіку**, тобто здатність комп'ютерної системи створювати графіку і вести діалог з людиною. У наш час майже будь-яку програму можна вважати системою інтерактивної комп'ютерної графіки.

Інтерактивна комп'ютерна графіка – це те ж використання комп'ютерів для підготовки і відтворення зображень, але при цьому користувач має можливість оперативно вносити зміни в зображення безпосередньо в процесі його відтворення, тобто, передбачається можливість роботи із графікою в режимі діалогу в реальному масштабі часу. Інтерактивна графіка являє собою важливий розділ комп'ютерної графіки, коли користувач має можливість динамічно керувати вмістом зображення, його формою, розміром і кольором на поверхні дисплея за допомогою інтерактивних пристроїв керування.

Різновиди комп'ютерної графіки

Поняття комп'ютерної графіки досить широке – від алгоритмів, що рисують на екрані вигадливі візерунки, до потужних пакетів 3D-графіки і програм, що імітують класичні інструменти художника. Для кожного розділу комп'ютерної графіки створене своє програмне забезпечення, що включає різноманітні спеціальні програми (графічні редактори). Іншими словами, комп'ютерна графіка – це не просто рисування за допомогою комп'ютера, а досить складний комплекс, що умовно можна розділити на такі напрямки.

1. Двовимірна графіка (2D) – основа всієї комп'ютерної графіки (у тому числі й 3D-графіки). Жоден дизайнер не може плідно працювати над своїми проектами без розуміння основних принципів двовимірної графіки. Такі програми як Adobe Photoshop (растрова програма), Adobe Illustrator, CorelDRAW, Macromedia FreeHand (векторні програми) працюють у двовимірній графіці.

2. Поліграфія – досить складний напрямок, що потребує від працюючих в цій області найбільшої широти знань. Навіть, на перший погляд, робота в поліграфії досить різноманітна: створення візиток, бланків, рекламних листівок, буклетів і плакатів; робота в періодичних виданнях (часто має свою специфіку). Для реалізації цих завдань призначені такі програми верстки, як Adobe PageMaker, Adobe InDesign і QuarkXPress. Програми верстки дозволяють з'єднувати текстову і графічну інформацію для створення інформаційних бюлетенів, журналів, брошур і рекламної продукції. Більшість програм верстки використовуються для компонування різних елементів на сторінці, а не для того, щоб з нуля створювати в них текстові або графічні файли. Тексти об'ємних документів, як правило, набираються в текстових редакторах типу MS Word, а потім імпортуються в програми верстки. Графіка часто створюється в програмах креслення (ділової графіки) і редагування зображень, а потім імпортується в програму верстання. Фахівець в області поліграфії повинен не тільки володіти програмами верстання і графічними редакторами, але й знати основи друку, розбиратися в процесах сканування, кольороподілу, кольорокалібрування моніторів і контролю якості.

3. Мультимедіа – це область комп'ютерної графіки, пов'язана зі створенням інтерактивних енциклопедій, довідкових систем, навчальних програм і інтерфейсів до них. На відміну від поліграфії, де файли повинні мати досить велику роздільну здатність, яка у результаті призводить до того, що розміри файлів можуть становити десятки і навіть сотню мегабайтів. У мультимедіа обмеженням слугує роздільна здатність екрана монітора і вимога мінімізації розмірів файлів. Для роботи в цій області поряд із графічними редакторами необхідно знати програми створення мультимедіа – наприклад, Macromedia Director або MS Power Point. Ці програми допускають зручний імпорт відео- і звукових файлів, у них передбачені засоби анімації діаграм.

4. World Wide Web (WWW). Зображення набули особливого значення, особливу значимість із розвитком глобальних комп'ютерних мережних технологій. У наш час – це одна з областей застосування комп'ютерної графіки, що найбільш бурхливо розвиваються. Вимоги до створення зображень для мережі Інтернет дуже суперечливі. З одного боку, обмеження щодо зменшення розмірів файлів для мінімізації часу їхньої передачі по мережі, з іншого боку – необхідність збереження якості переданого по мережі зображення;

5. 3D-графіка – це створення штучних предметів і персонажів, їхня анімація і поєднання з реальними предметами і інтер'єрами. На сьогоднішній день визначилося кілька перспективних напрямків її використання: індустрія комп'ютерних ігор, телевізійна реклама, напрямок, у якому архітектори й дизайнери використовують 3D-графіку для побудови макетів і тривимірних моделей архітектурних об'єктів.

6. Програми САПР і ділова графіка (або CAD) являють собою векторні програмні засоби, які знайшли широке застосування в різних сферах людської діяльності: архітектура (дизайн, проектування приміщень), медицина (автоматизоване проектування імплантів, особливо для кісток і суглобів, дозволяє мінімізувати необхідність внесення змін у ході операції, що скорочує час перебування на операційному столі), різні області інженерної конструкторської діяльності – від проектування мікросхем до створення літаків. Серед програм моделювання під Windows безумовним лідером є програма AutoCad. Особливістю комп'ютерних програм даного типу є їхня предметна спрямованість. Тому їхнє використання передбачає знання не тільки основ комп'ютерної графіки, але й самого предмета проектування.

7. Відеомонтаж можна умовно розділити на два види: спецефекти в кіно, підготовка телевізійних передач. Відеомонтаж відрізняється від інших напрямків комп'ютерної графіки тим, що маніпулює «живими» картинками і використовує свою технологію роботи. На сьогоднішній день однією з найбільш популярних програм, що використовують у цій області комп'ютерної графіки, є Adobe Premier.

Незважаючи на те, що для роботи з комп'ютерною графікою існує безліч класів програмного забезпечення, розрізняють усього три види комп'ютерної графіки. Це растрова графіка, векторна графіка і фрактальна графіка. Вони відрізняються принципами формування зображення при відображенні на екрані монітора або при друці на папері.

Растрову графіку застосовують при розробленні електронних (мультимедійних) і поліграфічних видань. Ілюстрації, виконані засобами растрової графіки, рідко створюють вручну за допомогою комп'ютерних програм. Частіше для цієї мети сканують ілюстрації, підготовлені художником на папері, або фотографії. Останнім часом для уведення растрових зображень у комп'ютер знайшли широке застосування цифрові фото- і відеокамери. Відповідно більшість графічних редакторів, призначених для роботи з растровими ілюстраціями, орієнтовані не стільки на створення зображень, скільки на їхню обробку.

Програмні засоби для роботи з *векторною графікою* навпаки, призначені, у першу чергу, для створення ілюстрацій і в меншому ступені для їхньої обробки. Такі засоби широко використовують у рекламних агентствах, дизайнерських бюро, редакціях і видавництвах. Засобами векторної графіки створюються роботи, основані на застосуванні шрифтів і найпростіших геометричних елементів. Проте існують приклади високомистецьких виробів, створених засобами векторної графіки, але вони скоріше виняток, ніж правило, оскільки художня підготовка ілюстрацій засобами векторної графіки надзвичайно складна.

Програмні засоби для роботи з фрактальною графікою призначені для автоматичної генерації зображень шляхом математичних розрахунків. Створення фрактальної художньої композиції полягає не в рисуванні або

оформленні, а в програмуванні. Фрактальну графіку рідко застосовують для створення друкованих або електронних документів, але її часто використовують у розважальних програмах.

Детальніше про основні особливості растрових та векторних зображень розглянуто у інших розділах навчального посібника.

1.2 Апаратна підтримка комп'ютерної графіки

Комп'ютерна графіка (computer graphics) зобов'язана своєю появою можливостям ЕОМ, а точніше її периферійним пристроям, здійснювати виведення інформації в задані програмою ділянки або точки поля виведення. Цю можливість стали використовувати вже на ранньому етапі застосування обчислювальної техніки для візуалізації одержуваної інформації. Спочатку це були прості графіки, побудовані зірочками, точками або буквою «Х». Необхідність у графічній інтерпретації оброблюваної інформації привела до розробки й створення векторних графічних пристроїв у вигляді дисплеїв і графопобудовників, що вміють довільно рухати електронний промінь по екрані або перо по планшеті і дозволяють формувати зображення, яке складається з відрізків прямих, дуг кіл тощо. Це периферійне устаткування коштувало дуже дорого й потребувало нетривіального програмування. Широке використання комп'ютерної графіки почалося з винаходом растрових дисплеїв, у яких промінь сканує екран по рядках і зображення створюється варіацією інтенсивності променя.

Місце зберігання зображення називається *бітовою картою*, вона безупинно сканується спеціалізованим дисплейним процесором, який, переглядаючи вміст бітової карти, формує електричний сигнал, що керує променем дисплея. Дисплейний процесор разом з бітовою картою зазвичай розташовують на окремій платі, яка називається *платою графічного адаптера*. Структура бітової карти досить проста: кожному рядку екрана відповідає неперервна ділянка карти, а кожному пікселю – байт (декілька байтів, або частина байта).

Бітова карта знаходиться під керуванням *центрального процесора* (ЦП) і є частиною *оперативної пам'яті* (ОЗП). Тому ЦП може звичайними командами звертання до пам'яті читати й змінювати вміст бітової карти. Таким чином, над бітовою картою одночасно працюють два процесори – центральний і дисплейний.

Оскільки обсяг бітової карти завжди обмежений (16–256 Кбайт), кількість пікселів екрана, як і можливих кольорів кожного пікселя, також обмежена.

Найпоширенішою формою створення кольорового зображення є використання трьох основних кольорів, змішуючи які, одержують інші кольори спектра. У більшості випадків як основні кольори використовують

червоний, зелений і синій. Простий кольоровий буфер кадру можна реалізувати на основі трьох бітових площин, по одній для кожного з основних кольорів. Кожна бітова площина керує індивідуальною електронною гарматою для кожного із трьох основних кольорів. Три основних кольори, комбінуючись на електро-променевій трубці (ЕПТ), дають вісім кольорів. Схема простого кольорового растрового буфера кадру показана на рис. 1.1.

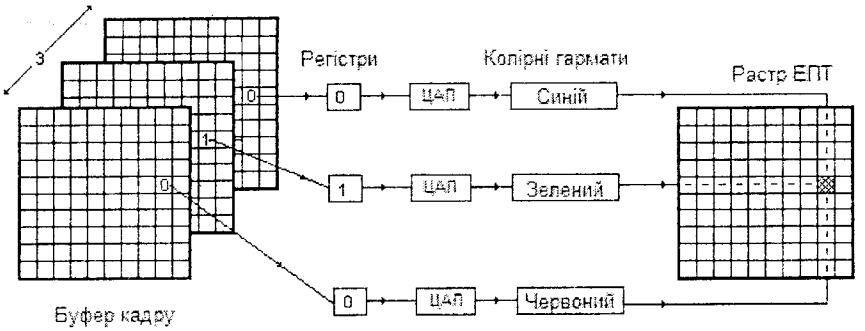


Рисунок 1.1 – Простий кольоровий буфер кадру

Для кожної із трьох колірних гармат можуть використовуватися додаткові бітові площини. Наприклад, кольоровий буфер кадру з восьми бітовими площинами на кожний колір, тобто отримуємо буфер кадру з двадцяти чотирма бітовими площинами. Кожна група бітових площин керує 8-розрядним ЦАП і може генерувати 256 (2^8) відтінків або інтенсивностей червоного, зеленого або синього кольорів. Їх можна скомбінувати в 16 777 216 [$(2^8)^3 = 2^{24}$] можливих кольорів. Це називається повнокольоровим буфером кадру.

Повнокольоровий буфер кадру може бути збільшений ще більше шляхом використання груп бітових площин як індексів в таблицях кольорів. При N бітах на колір і W-розрядних елементах таблиць кольорів одночасно може бути показано $(2^3)^N$ колірних відтінків з палітри $(2^3)^W$ можливих кольорів. Наприклад, при буфері кадру з 24 бітовими площинами (N = 8) і трьома 10-розрядними таблицями кольорів (W = 10) може бути отримано 16 777 216 (2^{24}) колірних відтінків з палітри 1 073 741 824 (2^{30}) кольорів, тобто близько 17 млн відтінків з палітри приблизно в 1 мільярд кольорів.

1.3. Еволюція комп'ютерних відеосистем

Комп'ютерні відеосистеми можна розглянути на прикладі персональних комп'ютерів класу IBM PC. Перший персональний комп'ютер фірми IBM з'явився у 1981 році. Поява саме цього комп'ютера

привела до значного поширення персональних комп'ютерів. Деякі особливості архітектури IBM PC збережені і досі (рис. 1.2).

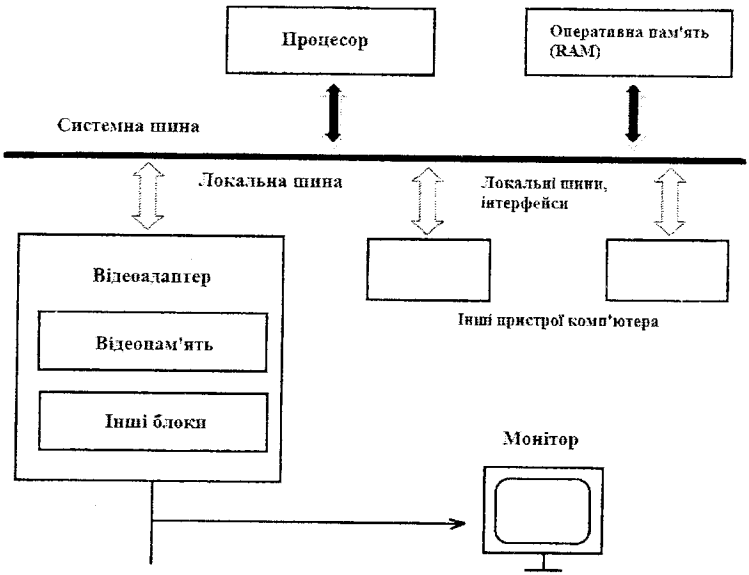


Рисунок 1.2 – Загальна (спрощена) структура персонального комп'ютера

Однією з таких особливостей, що вигідно відрізняє його від інших персональних комп'ютерів, є відкритість архітектури. Це означає гнучкість можливості під'єднання різноманітних пристроїв, простоту модернізації комп'ютера. Важливим чинником стала ціна – для IBM PC вона була меншою, ніж, наприклад, для комп'ютерів Apple, які були кращі за іншими показниками. Крім того, із початку користувачам комп'ютерів сімейства IBM PC були доступні різноманітні засоби програмування – мільйони користувачів отримали можливість самостійно розробляти програмне забезпечення. Все це привело до масового поширення таких комп'ютерів і використання їх у різних областях. Важливою межею архітектури персонального комп'ютера з позицій графіки є те, що контролер відеосистеми (відеоадаптер) розташований поряд з процесором і оперативною пам'яттю і під'єднаний до системної шини через швидку локальну шину. Це дає можливість швидко вести обмін даними між оперативною пам'яттю і відеопам'яттю (для виведення графічних зображень, особливо в режимі анімації, потрібна висока швидкість передачі даних). На відміну від цього, у великих комп'ютерах (мейнфреймах) дані до дисплеїв передавалися через інтерфейс каналу введення-

виведення, який працює набагато повільніше за системну шину. Великі комп'ютери, як правило, працювали з багатьма дисплеями, розташованими на значній відстані.

Перший комп'ютер IBM PC був оснащений відеоадаптером **MDA (Monochrome Display Adapter)**. Відеосистема була призначена для роботи в текстовому режимі – відображувалися 25 рядків по 80 символів в кожному рядку. Через рік невелика фірма Hercules випустила відеоадаптер **Hercules** Graphic Card. Він підтримував також і графічний чорно-білий режим 720×350.

Наступним кроком був відеоадаптер **CGA (Color Graphic Adapter).** Це перша кольорова модель для IBM PC. Адаптер CGA дозволяв працювати в кольоровому, текстовому або в графічному режимах. Далі ми розглядатимемо лише графічні режими відеоадаптерів. Графічних режимів було два: чорно-білий 640×200 і кольоровий 320×200. У кольоровому режимі можна було відображувати одночасно лише чотири кольори (2 біти на піксель).

У 1984 році з'явився адаптер EGA **(Enhanced Graphic Adapter).** Це було значне досягнення для персональних комп'ютерів даного типу. З'явився графічний 16-кольоровий відеорежим 640×350 пікселів. Кольори можна було вибирати з палітри 64 кольорів. В цей час почали поширюватися комп'ютерні ігри з більш-менш якісною графікою і графічні програми для роботи. Проте недоліком відеорежиму 640×350 була різна роздільна здатність по горизонталі і вертикалі – «неквадратні пікселі».

У 1987 році з'явилися відеоадаптери **MCGA (Multi-color Graphic Array)** і VGA (Video Graphic Array). Вони забезпечували вже 256-кольорові відеорежими.

Популярнішим став відеоадаптер VGA. Адаптер VGA мав 256-кольоровий графічний відеорежим з розмірами растра 320×200. Кольори можна було вибирати з палітри в 256 тисяч кольорів. Це дало можливість повністю задовольнити потреби відображення чорно-білих (у 256 градаціях сірого) фотографій. Кольорові фотографії відображалися досить якісно, проте 256 кольорів замало, тому в комп'ютерних іграх і графічних пакетах активно використовувався дизеринг (в обробці цифрових сигналів являє собою підмішування у первісний сингал псевдовипадкового шуму зі спеціально підібраним спектром). Метою дизерингу є розупорядкування шуму квантування, запобігання появі небажаних спотворень, гірших для сприйняття. Дизеринг застосовується при обробці аудіо- (зокрема як завершальний етап підготовки аудіокомпакт-дисків), відео- та графічної інформації. Крім того, режим 320×200 теж має різну роздільну здатність по горизонталі і вертикалі. Для моніторів, які використовувалися в персональних комп'ютерах типу IBM PC, необхідно, щоб кількість пікселів по горизонталі і вертикалі була в пропорції 4:3. Тобто, не 320×200, а 320×240. Відеоадаптер VGA також має

16-колірний відеорежим 640×480. Це відповідає «квадратним пікселям». Збільшення роздільної здатності в порівнянні з EGA дуже велике, що дало новий поштовх для розвитку графічних програм на персональних комп'ютерах. Подальший розвиток відеоадаптерів для комп'ютерів типу IBM PC пов'язаний зі збільшенням роздільної здатності і кількості кольорів. Можна відзначити відеосистему IBM 8514, яка була призначена для роботи з пакетами САПР (системи автоматизованого проектування). Почали з'являтися відеоадаптери різних фірм, які забезпечували спочатку відеорежими 800×600, а потім і 1024×768 при 16-ти кольорах, а також відеорежими 640×480. 800×600 і більше – для 256 кольорів. Ці відеоадаптери стали називати **SUPERVGA**. Пізніше з'явився відеоадаптер **IBM XGA**.

Першою глибини кольору в 24-біти досягла фірма Truevision з відеоадаптером **Targa 24**, що дозволило отримати на персональних комп'ютерах IBM PC відеорежим True Color. Таке досягнення можна вважати початком професійної графіки на персональних комп'ютерах цього типу. Там, де раніше використовували графічні робочі станції або персональні комп'ютери Apple Macintosh, віднині поступово переходили на дешевші комп'ютери IBM PC.

Зараз на персональних комп'ютерах використовується багато типів відеоадаптерів. Всі відеосистеми растрового типу. Вони дозволяють встановлювати глибину кольору до 32 бітів на піксель при розмірах растра 1600×1200 і більше. Існують стандарти на відеорежими, регламентовані VESA (Video Electronic Standards Association).

Параметри відображення обумовлюються не лише моделлю відеоадаптера, але і об'ємом встановленої відеопам'яті. Відеопам'ять персонального комп'ютера (VRAM – Video RAM) зберігає растрове зображення, яке демонструється на екрані монітора. Зображення на моніторі повністю відповідає поточному вмісту відеопам'яті. Відеопам'ять постійно сканується з частотою кадрів монітора. Запис нових даних у відеопам'ять миттєво змінює зображення на моніторі. Необхідний об'єм відеопам'яті розраховується як площа растра екрана в пікселях, помножена на кількість бітів (або байтів) на піксель. Наприклад, для 24-бітового відеорежиму 1024×768 потрібно відеопамяті: 24×1024×768 = 18,874,368 бітів = 2,25 Мбайт.

У відеоадаптерах перших зразків кількість відеопам'яті обчислювалася кілобайтами, наприклад, адаптер CGA мав 16 Кбайт. У сучасних відеоадаптерах рахунок йде на мегабайти. Зазвичай об'єм відеопам'яті кратний степені два – 1, 2, 4, 8 Мбайт (в наш час – від 32 Мбайт і більше). Спостерігається тенденція збільшення об'ємів відеопам'яті. Основним чинником тут вже не є глибина кольору. Відеопам'ять зараз використовується не лише як кадровий буфер – вона може зберігати текстури, z-буфер і т. д.

Адреси, за якими процесор звертається до відеопам'яті, знаходяться у

спільному адресному просторі. Наприклад, для багатьох відеорежимів VGA-адреса першого байта відеопам'яті дорівнює A0000. Для деяких відеорежимів старих зразків використовується інша адреса, наприклад, B8000 для CGA 320×200. Сучасні відеоадаптери зазвичай підтримують відеорежими, які використовувалися раніше. Це робиться для забезпечення можливості функціонування старих програм. Кожен відеорежим має власний номер (код) згідно зі стандартом VESA.

Окрім фізичної організації пам'яті комп'ютера – у вигляді одновимірного вектора байтів в загальному адресному просторі, необхідно враховувати логічну організацію відеопам'яті. Наприклад, у відеорежимі VGA 16 кольорів 640×480 використовують чотири масиви байтів пам'яті. Кожен масив названий бітовою площиною, для будь-якого пікселя використовують однакові біти даних різних площин. Кожна бітова площина має 80 байтів в одному рядку. Площини мають однакову адресу в пам'яті, тому для доступу до окремої площини необхідно встановлювати індекс площини у відповідному регістрі відеоадаптера.

Подібний спосіб організації відеопам'яті використовується в багатьох інших відеорежимах, він дозволяє, наприклад, швидко копіювати масиви пікселів. Для зберігання декількох кадрів зображення в деяких відеорежимах передбачаються окремі сторінки відеопам'яті з однаковою логічною організацією. Тоді можна змінювати стартову адресу відеопам'яті – це призводить до зсуву зображення на екрані. У всіх графічних відеорежимах стартова адреса відеопам'яті відповідає лівому верхньому пікселю на екрані. Тому координатна система з центром координат (0, 0) у лівому верхньому куті растра часто використовується як основа (або встановлюється за замовчуванням) у багатьох графічних інтерфейсах програмування, наприклад GDI API Windows. Обмін даними із системної шини для відеосистеми забезпечують процесор, відеоадаптер і контролер локальної шини.

До недавнього часу для підключення відеоадаптерів використовувалася локальна шина PCI (Peripheral Component Interconnect local bus). Шина PCI призначена не тільки для графіки, вона є стандартом приєднання різних пристроїв, наприклад, модемів, мережевих контролерів, контролерів інтерфейсу Ця шина – 32-бітова, працює на частоті 33 Мгц, швидкість обміну до 132 Мбайт/с.

Сучасні відеоадаптери являють собою складні графічні пристрої. На платі відеоадаптера (зараз його часто називають відеокартою) розташовується потужний спеціалізований графічний процесор (GPU – Graphic Processor Unit), який за складністю наближається до центрального процесора.

Крім візуалізації кадрового буфера графічний процесор відеоадаптера виконує як відносно прості растрові операції (копіювання масивів пікселів, маніпуляції з кольорами пікселів), так і більш складні. Там, де раніше використовувався виключно центральний процесор, тепер все частіше

застосовується графічний процесор відеоадаптера, наприклад, для виконання операцій графічного виведення ліній, полігонів. Перші графічні процесори відеоадаптерів виконували переважно операції рисування плоских елементів.

Сучасні графічні процесори виконують уже багато базових операцій 3D-графіки, наприклад, підтримку Z-буфера, накладення текстур тощо. Відеоадаптер виконує ці операції апаратно, що дозволяє значно прискорити їх у порівнянні з програмною реалізацією даних операцій центральним процесором. Так з'явився термін графічні акселератори. Швидкодія таких відеоадаптерів часто вимірюється кількістю графічних елементів, які створюються за одну секунду. Сучасні графічні акселератори здатні створювати мільйони трикутників за секунду. Цим «інтелектуальність» відеоадаптерів не обмежується. Також з'явилися моделі, які, крім відносно простих незмінних базових операцій, здатні самі виконувати невеликі програми, що можуть встановити користувачі. Ці програми називаються «рейдерами» (shaders). Такі можливості графічних акселераторів активно використовуються розробниками комп'ютерних ігор.

На рис. 1.3 подано загальну структуру сучасного відеоадаптера.

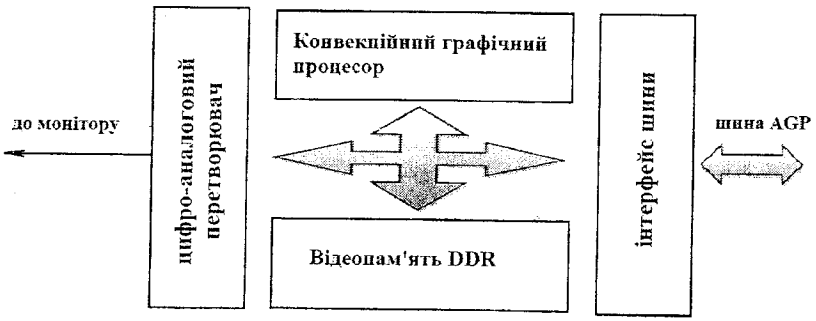


Рисунок 1.3 – Загальна структура відеоадаптера

Номенклатура відеоадаптерів для персональних комп'ютерів широка. Декілька прикладів: відеоадаптери Matrox (якісна двовимірна графіка), ATI Radeon, NVidia (професійні та ігрові 3D-акселератори).

Використання програмістами графічних можливостей відеосистеми може здійснюватися по-різному. По-перше, найпростіші операції, такі, як визначення графічного відеорежиму, виведення пікселя на екран і деякі інші, підтримуються BIOS. По-друге, можна використовувати функції операційної системи. Різні операційні системи можуть надавати різні можливості. Наприклад, в MS-DOS графічних функцій майже не було, проте програмісту був дозволений вільний доступ до всіх апаратних ресурсів комп'ютера. В швидкодіючих графічних програмах часто не

використувався безпосередній доступ до відеопам'яті. На відміну від цього, операційна система Windows забороняє прикладним програмам безпосередній доступ до апаратних ресурсів, проте можна застосовувати декілька сотень графічних функцій операційної системи – інтерфейс GDI API. По-третє, можна використовувати спеціалізовані графічні інтерфейси, які підтримують апаратні можливості сучасних графічних процесорів.

Один з найвідоміших графічних інтерфейсів – OpenGL. Цей інтерфейс у вигляді бібліотеки графічних функцій був розроблений Silicon Graphics і підтримується багатьма операційними системами (в тому числі Windows) і виробниками графічних акселераторів. Інтерфейс OpenGL для графічного відображення використовує взаємодію типу клієнт-сервер.

Іншим відомим графічним інтерфейсом є DirectX. Цей інтерфейс розроблений Microsoft і призначений тільки для ОС Windows.

Контрольні питання та завдання

1. Дайте означення комп'ютерної графіки.
2. Що розуміють під графічною інформацією?
3. Що є кінцевим продуктом комп'ютерної графіки?
4. Назвіть один з основних напрямів підвищення продуктивності і поліпшення якості графічних розробок.
5. Які основні задачі розглядаються у комп'ютерній графіці?
6. Які основні напрями виділяють при обробці інформації, пов'язаної із зображенням?
7. У чому полягає основне завдання розпізнавання образів?
8. Які завдання розглядає обробка зображень? Наведіть приклади обробки зображень.
9. Дайте означення графічного редактора. Наведіть приклади графічних редакторів.
10. У яких галузях використовується комп'ютерна графіка?
11. Які функції виконує бітова карта? Опишіть її структуру.
12. Опишіть схему простого кольорового растрового буфера кадру.
13. Як працює повнокольоровий буфер кадру?
14. Опишіть існуючі відеоадаптери та дайте їх порівняльну характеристику.
15. Які операції виконують графічні процесори відеоадаптерів?
16. Опишіть загальну структуру відеоадаптера.
17. Які графічні інтерфейси ви знаєте?

РОЗДІЛ 2 КООРДИНАТИ ТА ПЕРЕТВОРЕННЯ НА ПЛОЩИНІ

2.1 Координатний метод

Координатний метод був введений в XVII столітті французькими математиками *Р. Декартом і П. Ферма.* На цьому методі основується *аналітична геометрія,* яку можна вважати фундаментом комп'ютерної графіки. В сучасній комп'ютерній графіці координатний метод широко використовується. Цьому є декілька причин:

- кожна точка на екрані (або на папері при друці на принтері) задається координатами – наприклад піксельними.
- координати використовуються для опису об'єктів, котрі будуть відображатися як просторові. Наприклад, об'єкти мікросвіту, об'єкти на поверхні Землі, об'єкти космічного простору і т. п. Навіть тоді, коли відображається щось, що не має прив'язки до положення в просторі (наприклад, випадкові кольорові плями в якомусь відеоефекті), також використовуються координати для врахування взаєморозміщення окремих елементів.
- при виконанні багатьох проміжних дій для відображення використовують різні системи координат і перетворення з однієї системи в іншу.

2.1.1 Перетворення координат

Спочатку потрібно розглянути загальні питання перетворення координат. Нехай задана n-вимірна система координат в базисі (k_1, k_2, ..., k_n), яка описує положення точки в просторі з допомогою числових значень k_i. В комп'ютерній графіці частіше використовується двовимірна ($n = 2$) і тривимірна ($n = 3$) системи координат.

Якщо задати іншу, N-вимірну, систему координат в базисі (m_1, m_2, ..., m) і поставити задачу визначення координат в новій системі, знаючи координати в старій, то розв'язок (якщо він існує) можна записати в такому вигляді:

$$\begin{cases} m_1 = f_1(k_1, k_2, \dots, k_n) \\ m_2 = f_2(k_1, k_2, \dots, k_n) \\ \dots \\ m_N = f_N(k_1, k_2, \dots, k_n), \end{cases} \tag{2.1}$$

де f_i – функція перерахунку і-ї координати, аргументами є координати в системі k_i.

Можна поставити й обернену задачу: за відомими координатам m_1, m_2, ..., m) визначити координати (k_1, k_2, ..., k_n). Розв'язок оберненої задачі може бути записано так:

$$\begin{cases} k_1 = F_1(m_1, m_2, \ldots, m_N) \\ k_2 = F_2(m_1, m_2, \ldots, m_N) \\ \ldots \\ k_n = F_n(m_1, m_2, \ldots, m_N) \end{cases}, \qquad (2.2)$$

де F_i – функції оберненого перетворення.

У випадку якщо розмірності систем координат не збігаються ($n \neq N$), здійснити однозначне перетворення координат частіше всього не вдається. Наприклад, за двовимірними екранними координатами неможливо без додаткових умов однозначно визначити тривимірні координати об'єктів, що відображаються.

Якщо розмірності систем збігаються ($n = N$), то також можливі випадки, коли неможливо однозначно розв'язати пряму чи обернену задачі.

Перетворення координат класифікують:

• за системами координат – наприклад, перетворення з полярної системи в прямокутну;

• за виглядом функції перетворення.

За видом функції перетворення розрізняють лінійні і нелінійні перетворення. Якщо при всіх $u = 1, 2, \ldots, N$ функції f_i – лінійні відносно аргументів $(k_1, k_2, \ldots, k_n)$, тобто

$$f_i = a_{i1}k_1 + a_{i2}k_2 + \cdots + a_{in}k_n + a_{in+1},$$

де a_{ij} – константи, то такі перетворення називаються лінійними, а при $n = N$ – афінними.

Якщо хоча б для одного i функція f_i – нелінійна відносно $(k_1, k_2, \ldots, k_n)$, тоді перетворення координат в цілому є нелінійним. Наприклад, перетворення

$$\begin{cases} X = 3x + 5y, \\ Y = 4xy + 10y \end{cases}$$

нелінійне, оскільки присутній xy в виразі для Y.

Лінійні перетворення наочно записуються в матричній формі:

$$\begin{bmatrix} m_1 \\ m_2 \\ \ldots \\ \ldots \\ m_N \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} & a_{1n+1} \\ a_{21} & a_{22} & \ldots & a_{2n} & a_{2n+1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{N1} & a_{N2} & \ldots & a_{Nn} & a_{Nn+1} \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ \ldots \\ \ldots \\ k_N \end{bmatrix}. \qquad (2.3)$$

Тут матрицю коефіцієнтів (a_{ij}) множимо на матрицю-стовпець (k_i), і в результаті отримуємо матрицю-стовпець(m_i).

Для двох матриць – матриці A розмірами $(m \times n)$ та B – $(n \times p)$

$$A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \qquad B=\begin{bmatrix} b_{11} & b_{12} & \dots & a_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{np} \end{bmatrix}$$

матричним утворенням є матриця $C=AB$ розмірами $(m \times p)$:

$$C=\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mp} \end{bmatrix},$$

для якої елементи c_{ij} розраховують за формулою:

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

Правило обчислення елементів матриці C можна легко запам'ятати за назвою «рядок на стовпець». І дійсно, для обчислення будь-якого елемента c_{ij} необхідно помножити елементи i-го рядка матриці A на елементи j-го стовпця матриці B.

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & c_{ij} & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ a_{i1} & a_{i2} & \bullet & a_{in} \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} \begin{bmatrix} \bullet & \bullet & b_{1j} & \bullet \\ \bullet & \bullet & b_{2j} & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & b_{nj} & \bullet \end{bmatrix}. \qquad (2.4)$$

Утворення матриць визначається тільки для випадку, коли кількість стовпців матриці A рівна кількості рядків матриці B.

2.1.2 Афінні перетворення на площині

Нехай задано деяку двовимірну систему координат (x, y). Афінне перетворення на площині описується формулами:

$$\begin{cases} X = Ax + By + C, \\ Y = Dx + Ey + F, \end{cases} \qquad (2.5)$$

де A, B, ..., F – константи. Значення (X, Y) можна розглядати як координати в новій системі координат.

Обернене перетворення (X, Y) в (x, y) також є афінним:

$$\begin{cases} x = A'X + B'Y + C', \\ y = D'X + E'Y + F'. \end{cases} \qquad (2.6)$$

Афінне перетворення зручно записувати в матричному вигляді. Константи *A*, *B*, ..., *F* утворюють матрицю перетворення, яка, при множенні на матрицю-стовпець координат *(x, y)*, дає матрицю-стовпець *(X, Y)*. Однак, щоб врахувати константи *C* і *F*, необхідно перейти до так званих однорідних координат – додамо іще один рядок в матрицях координат:

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \qquad (2.7)$$

Матричний запис дає можливість наочно описувати декілька перетворень, що йдуть одне за одним. Наприклад, якщо необхідно спершу виконати перетворення $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = [A] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$, а потім – інше перетворення $\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = [B] \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$, то це можна описати як $\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = [B] \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = [B][A] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$.

Однак замість двох перетворень можна виконати тільки одне $\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = [C] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$, де матриця *C* рівна виразу *BA*.

Властивості афінних перетворень.

1. За властивостями координат афінне перетворення є взаємно однозначним відображенням площини на площину:
   - кожна точка має відображення і при тому лише одне;
   - різні точки мають різні відображення.
2. Оскільки афінне відображення зберігає координати точок, то воно зберігає рівняння фігур. Звідси випливає, що пряма переходить в пряму.
3. Перетворення, обернене до афінного, є знову афінним перетворенням.
4. Точки, які не лежать на одній прямій, переходять в точки, які не лежать на одній прямій, аналогічно – прямі, що перетинаються, переходять в прямі, що перетинаються, а паралельні – в паралельні.
5. При афінних перетвореннях зберігаються відношення довжин відрізків, які лежать на одній або паралельних прямих.
6. Відношення площ багатокутників також зберігаються.
7. Необов'язково зберігаються відношення довжин відрізків не паралельних прямих, кутів.

Відомі такі окремі випадки афінного перетворення.

1. Паралельний зсув координат (рис. 2.1).

$$\begin{cases} X = x - dx, \\ Y = y - dy. \end{cases} \tag{2.8}$$

В матричній формі:

$$\begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{bmatrix}$$

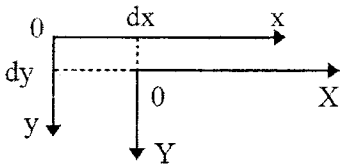


Рисунок 2.1 – Паралельний зсув координат

Обернене перетворення:

$$\begin{cases} x = X + dx, \\ y = Y + dy. \end{cases} \quad \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

2. Розтягнення-стиснення відносно осей координат (масштабування) (рис. 2.2).

$$\begin{cases} X = x/k_x, \\ Y = y/k_y. \end{cases} \tag{2.9}$$

В матричній формі:

$$\begin{bmatrix} 1/k_x & 0 & 0 \\ 0 & 1/k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

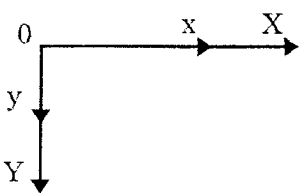


Рисунок 2.2 – Розтягнення-стиснення осей координат

Обернене перетворення:

$$\begin{cases} x = Xk_x, \\ y = Yk_y. \end{cases} \quad \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Коефіцієнти k_x і k_y можуть бути від'ємними. Наприклад, k_x = -1 відповідає дзеркальному відображенню відносно осі y.

3. Поворот (рис. 2.3)

$$\begin{cases} X = x\cos\alpha + y\sin\alpha, \\ Y = -x\sin\alpha + y\cos\alpha. \end{cases} \tag{2.10}$$

В матричній формі:

$$\begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

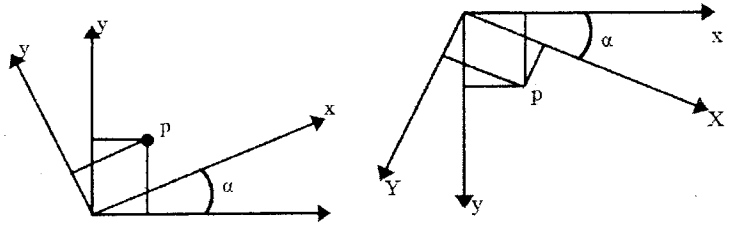


Рисунок 2.3 – Поворот

Обернене перетворення відповідає повороту системи (X, Y) на кут ($-\alpha$).

$$\begin{cases} x = X\cos\alpha - Y\sin\alpha, \\ y = X\sin\alpha + Y\cos\alpha, \end{cases} \quad \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

4. Відображення відносно осей координат (рис. 2.4.).

– відносно осі X:

$$\begin{cases} X = -X, \\ Y = Y. \end{cases} \tag{2.11}$$

В матричній формі:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

– відносно осі Y:

$$\begin{cases} X = X, \\ Y = -Y. \end{cases} \tag{2.12}$$

В матричній формі:

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

Властивості афінного перетворення на площині.

- будь-яке афінне перетворення може бути подане як послідовність операцій з числа вказаних найпростіших: зсув, розтягнення-стиснення, поворот.
- зберігаються: прямизна ліній, паралельність прямих, відношення довжин відрізків, що лежать на одній прямій, і співвідношення площ фігур.

Приклад 2.1. Нехай потрібно знайти точку P^*, симетричну точці P відносно деякого центра $C(x_c, y_c)$.

У прикладі потрібно виконати перетворення центральної симетрії (відносно точки), а часткове перетворення відображення відносно осі описує осьову симетрію (відносно прямої). Тоді центральну симетрію можна подати як результат двох послідовних перетворень осьової симетрії: спочатку відносно горизонтальної осі, потім – відносно вертикальної. При цьому обидві осі повинні проходити через центр симетрії. Оскільки матриця відображення працює лише відносно осей, що проходять через початок координат, перед симетрією потрібно передбачити зсув центра в початок системи координат. Аналогічно, після перетворень симетрії потрібно передбачити повернення точки на своє старе місце (ще один зсув). В результаті для виконання задачі потрібно виконати такий ряд перетворень:

$$\begin{vmatrix} x_p' & y_p' & 1 \end{vmatrix} = \begin{vmatrix} x_p & y_p & 1 \end{vmatrix} \times \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 0 \end{vmatrix} \times \begin{vmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \ldots$$

$$\ldots \times \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_c & y_c & 0 \end{vmatrix}.$$

Приклад 2.2. Нехай потрібно описати поворот точки на кут q навколо довільного центра $C(x_c, y_c)$. і подальше двократне масштабування результату відносно початку координат.

Очевидно, що початкові перетворення в цьому випадку збігаються з перетвореннями з прикладу 2.1, коли центр потрібно перенести в початок координат, потім здійснити всі необхідні перетворення і повернути точку у початкове положення. В результаті отримуємо такий вираз для розрахунку скінченної матриці перетворення:

$$\left|x_p' \;\; y_p' \;\; 1\right| = \left|x_p \;\; y_p \;\; 1\right| \times M1 \times M2 \times M3 \times M4,$$

де $M1$ – матриця перенесення точки в початок координат на відстань $(-x_c, -y_c)$;

$M2$ – матриця масштабування;

$M3$ – матриця повороту;

$M4$ – матриця перенесення точки на відстань (x_c, y_c).

Тривимірне афінне перетворення

Тривимірне афінне перетворення можна записати у вигляді формули:

$$\begin{cases} X = Ax + By + Cz + d, \\ Y = Ex + Fy + Gz + H,, \\ Z = Kx + Ly + Mz + N, \end{cases}$$

де $A, B, ..., N$ – константи.

Запис в матричній формі:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & C & D \\ E & F & G & H \\ K & L & M & N \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \qquad (2.13)$$

Для тривимірного простору будь-яке афінне перетворення також може бути подано послідовністю найпростіших операцій. Розглянемо їх.

1. Зсув осей координат відповідно на dx, dy, dz:

$$\begin{cases} X = x - dx, \\ Y = y - dy, \\ Z = z - dz, \end{cases} \qquad (2.14)$$

Запис в матричній формі:

$$\begin{bmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Розтягнення-стиснення на k_x, k_y, k_z:

$$\begin{cases} X = x/k_x, \\ Y = y/k_y, \\ Z = z/k_z. \end{cases} \quad (2.15)$$

Запис в матричній формі:

$$\begin{bmatrix} 1/k_x & 0 & 0 & 0 \\ 0 & 1/k_y & 0 & 0 \\ 0 & 0 & 1/k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

3. Повороти. В тривимірному просторі існує більше різновидів поворотів, в порівнянні з двовимірним простором. Існують декілька окремих випадків повороту.

Поворот навколо осі x на кут φ (рис. 2.4).

$$\begin{cases} X = x, \\ Y = y\cos\varphi + z\sin\varphi, \\ Z = -y\sin\varphi + z\cos\varphi. \end{cases} \quad (2.16)$$

Запис в матричній формі:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

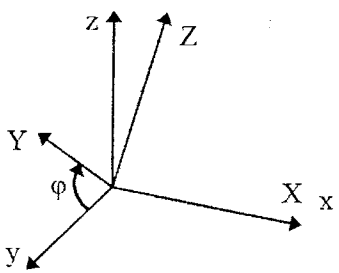


Рисунок 2.4 – Поворот навколо осі x

Поворот навколо осі y на кут ψ (рис. 2.5)

$$\begin{cases} X = x\cos\psi + z\sin\psi, \\ Y = y, \\ Z = -x\sin\psi + z\cos\psi. \end{cases} \quad (2.17)$$

Запис в матричній формі:

$$\begin{bmatrix} \cos\psi & 0 & \sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

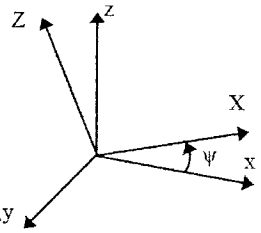


Рисунок 2.5 – Поворот навколо осі y

Поворот навколо осі z на кут γ (рис. 2.6*)*

$$\begin{cases} X = x\cos\gamma + y\sin\gamma, \\ Y = -x\sin\gamma + y\cos\gamma, \\ Z = z. \end{cases} \qquad (2.18)$$

Запис в матричній формі:

$$\begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

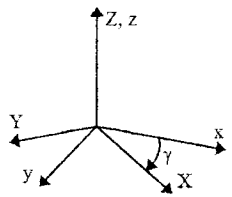


Рисунок 2.6 – Поворот навколо осі z

2.1.3 Зв'язок перетворень об'єкта з перетвореннями координат

Перетворення об'єктів і систем координат тісно пов'язані між собою. Рух об'єктів можна розглядати як рух в оберненому напрямку відповідної системи координат.

Така відносність для об'єктів відображення і систем координат дає

розробнику комп'ютерних систем додаткові можливості для моделювання і візуалізації просторових об'єктів. З кожним об'єктом можна пов'язувати як власну локальну систему координат, так і єдину для декількох об'єктів. Це можна використовувати, наприклад, при моделюванні рухомих об'єктів.

Звичайно, одного і того ж ефекту можна досягти, використовуючи різні підходи. В окремих випадках зручніше використовувати перетворення координат, а в інших – перетворення об'єктів. Не останню роль відіграє складність обґрунтування будь-якого способу, його зрозумілість.

Можна розглянути приклад комбінованого підходу. Нехай потрібно отримати функцію розрахунку координат $(X, Y) = F(x, y)$ для повороту навколо точки з координатами (x_0, y_0) (рис. 2.7).

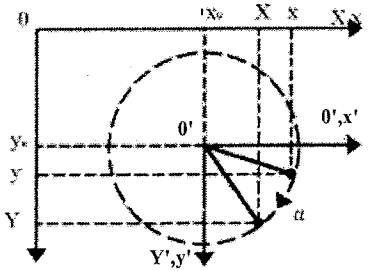


Рисунок 2.7 – Поворот навколо точки (x_0, y_0) на кут α

Вище було розглянуто поворот відносно центра координат (0, 0). Для розв'язання даної задачі потрібно ввести нову систему координат (x', y') з центром в точці (x_0, y_0):

$$\begin{cases} x' = x - x_0, \\ y' = y - y_0. \end{cases} \tag{2.19}$$

Для такої системи поворот точок відбувається навколо її центра:

$$\begin{cases} X' = x' \cos\alpha - y' \sin\alpha, \\ Y' = x' \sin\alpha + y' \cos\alpha, \end{cases}$$

Потрібно перетворити координати (X', Y') в (X, Y) із зсувом центра координат в точку (0, 0):

$$\begin{cases} X = X' + x_0, \\ Y = Y' + y_0. \end{cases}$$

Якщо об'єднати формули перетворень, то можна отримати:

$$\begin{cases} X = (x - x_0)\cos\alpha - (y - y_0)\sin\alpha + x_0, \\ Y = (x - x_0)\sin\alpha + (y - y_0)\cos\alpha + y_0. \end{cases} \quad (2.20)$$

Розв'язання цієї задачі можна було б здійснити і в матричній формі:

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} \text{Зсув системи} \\ \text{координат на} \\ -x_0, -y_0 \end{bmatrix} \begin{bmatrix} \text{Поворот} \\ \text{на кут} \\ \alpha \end{bmatrix} \begin{bmatrix} \text{Зсув системи} \\ \text{координат на} \\ x_0, y_0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \cos\alpha & -\sin\alpha & -x_0\cos\alpha + y_0\sin\alpha + x_0 \\ \sin\alpha & \cos\alpha & -x_0\sin\alpha - y_0\cos\alpha + y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Далі подано другий приклад. Тепер задача – вивести формулу параметричного опису поверхні тора. На рис. 2.8 поданий тор.

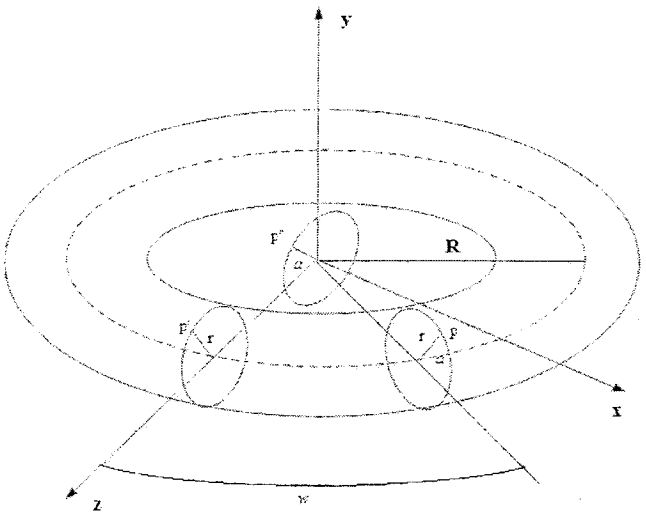


Рисунок 2.8 – Тор

Для довільної точки *p*, що лежить на поверхні тора, необхідно виразити координати (*x, y, z*) через константи, що описують розмір фігури, а також через деякі параметри. Для поверхні в тривимірному просторі необхідно використовувати два параметри. Як такі були вибрані кутові величини: *a* (широта) і ω (довгота).

Безпосереднє визначення координат точки *P* є складним, тому ці координати можна знайти за допомогою декількох перетворень. Якщо коло лежить в

площині zOy, то центр цього кола збігається з центром координат. Координати точки P'' з широтою φ визначаються як

$$x'' = 0,$$
$$y'' = r \cos a,$$
$$z'' = r \sin a,$$

де r – малий радіус тора.

Тепер потрібно перенести це коло на відстань R (великий радіус тора) по осі y в тій же площині zOy. В результаті можна отримати точку p' з координатами:

$$\begin{aligned} x' &= x'' = 0, \\ y' &= R + y'' = R + r \cos a, \\ z' &= z'' = r \sin a. \end{aligned} \tag{2.21}$$

Коло, якому належить точка p', є геометричним місцем точок тора з нульовою довготою ω. Якщо точку p' повернути на кут ω, то можна отримати шукану точку p поверхні тора з координатами:

$$\begin{aligned} x &= x' \cos \omega + y' \sin \omega, \\ y &= -x' \sin \omega + y' \cos \omega, \\ z &= z'. \end{aligned} \tag{2.22}$$

Якщо підставити значення (x', y', z'), то можна отримати шукані формули:

$$\begin{aligned} x &= (R + r \cos a) \sin \omega, \\ y &= (R + r \cos a) \cos \omega, \\ z &= r \sin a. \end{aligned} \tag{2.23}$$

2.2 Проекції

В наш час найбільш розповсюджені пристрої відображення, що синтезують зображення на площині – на екрані дисплею чи на папері. Пристрої, які створюють істинно об'ємні зображення, поки зустрічаються доволі рідко.

При використанні будь-яких графічних пристроїв звичайно застосовують проекції. Проекція задає спосіб відображення об'єктів на графічному пристрої. Розглянемо тільки проекції на площину.

При відображенні просторових об'єктів на екрані чи листі паперу з допомогою принтера необхідно знати координати об'єктів.

Існують дві системи координат.

Перша – *світові координати*, які описують істинне положення об'єктів в просторі із заданою точністю. Друга – система координат пристрою відображення, в якому здійснюється виведення зображення об'єктів в заданій проекції. Система координат графічного пристрою називається *екранними координатами* (хоча цей пристрій не обов'язково повинен бути подібний до монітора комп'ютера).

Нехай світові координати будуть тривимірними прямокутними координатами. Де має розміщуватись центр координат, і якими будуть одиниці вимірювання вздовж кожної осі, не дуже важливо. Важливо те, що для відображення будемо знати будь-які числові значення координат об'єктів, що відображаються.

Для отримання зображення у певній проекції необхідно вирахувати координати проекцій. Для синтезу зображення на площині екрана або папері використовуємо двовимірну систему координат. Основна задача – задати перетворення координат зі світових в екранні.

Проекціюванням називається процес побудови зображення предмета на площині – на папері, екрані, класній дошці тощо. При цьому утворюється зображення, яке називається проекцією. Прикладом проекції є креслення предметів, наочні зображення, фотографічні знімки, кінокадри та ін.

Отже, проекція – це зображення предмета (відкинуте) на площину за допомогою проекціювальних променів. Спроекціювати предмет – це означає зобразити його на площині.

Проекції поділяють на центральні і паралельні. Якщо проекціювальні промені, за допомогою яких будується зображення предмета, розходяться з однієї точки, проекціювання називається центральним. Точка, з якої виходять промені, називається центром проектування. Отримане при цьому зображення предмета називається центральною проекцією.

Центральні проекції часто називають перспективою. Прикладами центральної проекції є фотознімки; кінокадри; тіні, відкинуті від предметів променями електричної лампочки тощо.

Ідею центрального проекціювання демонструє рис. 2.9. Точка S, з якої виходять проекціювальні промені, називається *центром проекції*. Площина Π_1, на яку проекціюється предмет, називається *площиною проекції*. Проекційований трикутник ABC називається *оригіналом*.

Щоб спроекціювати трикутник, треба провести з центра проекцій S проекціювальні промені через вершини трикутника ABC до перетину з площиною проекцій Π_1. Точки перетину A_1, B_1, C_1, називаються *центральними проекціями вершин* A, B, C, а спроекційований трикутник $A_1B_1C_1$ – *центральною проекцією трикутника* ABC.

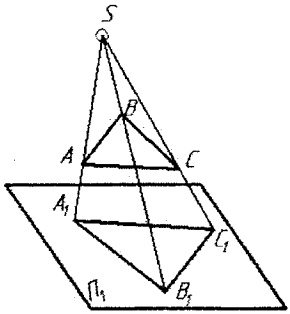


Рисунок 2.9 – Приклад центрального проекціювання

Якщо у вибраній об'єктній систем координат $OXYZ$ відомі координати точок A і S, а також рівняння поверхні зображення $\varphi(X,Y,Z)=0$, то координати точки зображення A' визначаються в результаті розв'язання системи рівнянь:

$$\begin{cases}(X_A-X_{A'})/(X_A-X_S)=(Y_A-Y_{A'})/(Y_A-Y_S)=(Z_A-Z_{A'})/(Z_A-Z_S)\\ \varphi(X,Y,Z)=0\end{cases} \quad (2.24)$$

Недоліком центрального проекціювання є те, що не можна зафіксувати дійсну величину нашого предмета – тому центральні проекції застосовують в архітектурно-будівельній справі, у рисуванні тощо.

Якщо проекційні промені паралельні один одному, то процес проекціювання називається *паралельним*, а зображення – *паралельною проекцією*. Прикладом паралельної проекції є сонячні тіні.

При паралельному проекціюванні всі промені падають на площину проекцій під однаковим кутом. Якщо це будь-який гострий кут, то проекціювання називається косокутним. В косокутній проекції, як і в центральній, форма і величина предмета спотворюється. В тому разі, коли проекціювальні промені перпендикулярні до площини проекцій, тобто утворюють з нею кут 90°, проекціювання називають прямокутним (ортогональним). Отримане зображення називається прямокутною проекцією предмета.

У кресленні користуються методом паралельного проекціювання. Умовна точка S знаходиться в нескінченному просторі і тому проекціювальні промені також умовно вважаються паралельними і перпендикулярними до площини Π_1, а отримана проекція трикутника $A_1B_1C_1$ є паралельною проекцією трикутника ABC (рис. 2.10).

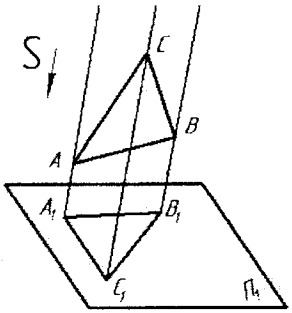


Рисунок 2.10 – Приклад паралельного проекціювання

Координати точок зображення при паралельній проекції визначають шляхом одночасного розв'язання рівняння прямої, яка проходить через предметну точку A паралельно одиничному вектору, і рівняння поверхні проекції:

$$\begin{cases}(X_A - X_{A'})/V_X = (Y_A - Y_{A'})/V_Y = (Z_A - Z_{A'})/V_Z \\ F(X_{A'}, Y_{A'}, Z_{A'}) = 0\end{cases} \qquad (2.25)$$

де $X_A Y_A Z_A$ координати точки предмета;

$X_{A'}, Y_{A'}, Z_{A'}$ – координати точки зображення;

$F(X_{A'}, Y_{A'}, Z_{A'})$ – рівняння поверхні проекції.

Якщо проекціювання відбувається на площину і проекціювальні промені перпендикулярні до неї, то проекція називається ортогональною або перпендикулярною. Цей вид проекції широко використовується в технічному кресленні. Якщо осі X та Y лежать в площині проекції, α^Z перпендикулярна до неї, то при поданні предмета координатним базисом площини проекції, координати точок зображення можна визначити за координатами точок предмета за допомогою простого співвідношення:

$$[X_{A'}\ \ Y_{A'}\ \ Z_{A'}] = [X_A\ \ Y_A\ Z_0] \qquad (2.26)$$

де $Z_0 = const$ – координати площини проекції по осі Z.

Паралельну проекцію можна також вважати різновидом центральної, для якої точка, в якій сходяться промені проекціювання, знаходиться в нескінченності.

Аксонометрична проекція – це різновид паралельної проекції. Для неї всі промені проекціювання розміщуються під прямим кутом до площини проекціювання.

На рис. 2.11 наведений приклад зображення каркаса куба і *xzy* осей координат в аксонометричній проекції.

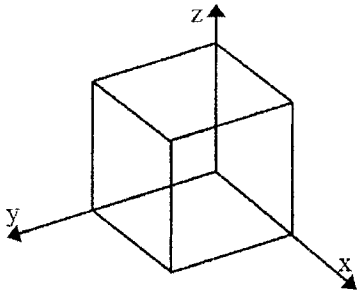


Рисунок 2.11 – Приклад зображення каркаса куба і *xzy* осей координат в аксонометричній проекції.

Нехай (X, Y, Z) – нова система координат, повернута відносно системи *(xzy)* на кути α і β. Потрібно розмістити площину проекціювання паралельно площині XOY на відстані $Z_{пл}$ і позначити координати в площині проекціювання як $X_{пр}$ та $Y_{пр}$ (рис. 2.12).

Тепер необхідно знайти співвідношення між координатами (x, y, z) і координатами $(X_{пр}, Y_{пр})$ для будь-якої точки в тривимірному просторі.

Спочатку можна вказати на окремий тривіальний випадок, при якому координати проекції збігаються зі світовими, тобто $X_{пр} = x$, $Y_{пр} = y$. Це буде вигляд зверху – для нього кути повороту $\alpha = 0$, $\beta = 0$.

Тепер можна розглянути перетворення системи координат (x, y, z) в систему (X, Y, Z) для довільних кутів (α, β). Нехай це перетворення задається двома кроками.

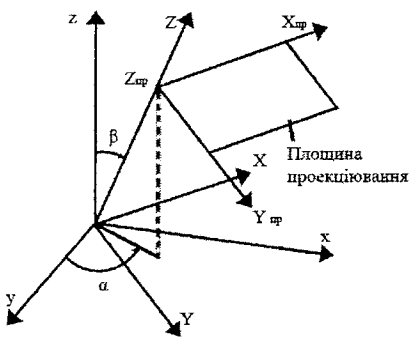


Рисунок 2.12 – Площина проекціювання паралельна площині XOY на відстані $Z_{пл}$.

1-й крок. Поворот системи координат відносно осі z на кут α отримуємо систему координат (x', y', z'). Такий поворот осей описується матрицею

$$A=\begin{bmatrix}\cos\alpha & -\sin\alpha & 0 & 0\\ \sin\alpha & \cos\alpha & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}. \tag{2.27}$$

2-й крок. Поворот системи координат (x', y', z') відносно осі x' на кут β – можна отримати координати (X, Y, Z). Матриця повороту:

$$B=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\beta & -\sin\beta & 0\\ 0 & \sin\beta & \cos\beta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}. \tag{2.28}$$

Перетворення координат можна виразити множенням $B \times A$:

$$B \times A = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\beta & -\sin\beta & 0\\ 0 & \sin\beta & \cos\beta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\alpha & -\sin\alpha & 0 & 0\\ \sin\alpha & \cos\alpha & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} =$$

$$= \begin{bmatrix}\cos\alpha & -\sin\alpha & 0 & 0\\ \sin\alpha\cos\beta & \cos\alpha\cos\beta & -\sin\beta & 0\\ \sin\alpha\sin\beta & \cos\alpha\sin\beta & \cos\beta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

Перетворення координат можна подати у вигляді формул:

$$\begin{aligned} X &= x\cos\alpha + y\sin\beta,\\ Y &= x\sin\alpha\cos\beta + y\cos\alpha\cos\beta - z\sin\beta,\\ Z &= x\sin\alpha\sin\beta + y\cos\alpha\sin\beta + z\cos\beta. \end{aligned} \tag{2.29}$$

Оскільки площина проекціювання розміщена паралельно площинні (XOY), а промені проекціювання перпендикулярні до цієї площини, то

$$\begin{aligned} X_{\text{пр}} &= X,\\ Y_{\text{пр}} &= Y,\\ Z_{\text{пр}} &= Z - Z_n \end{aligned}$$

Можна назвати систему координат (X, Y, Z) видовою системою координат. Вона визначає ракурс показування. Тривимірна система координат проекцій $(X_{\text{пр}}, Y_{\text{пр}}, Z_{\text{пр}})$ – зсунута відносно системи (X, Y, Z) на

$Z_{пр}$. Для візуалізації зображення достатньо знати двовимірні координати $X_{пр}$ та $Y_{пр}$.

Контрольні питання та завдання

1. Назвіть причини використання координатного методу в комп'ютерній графіці.
2. В чому полягає перетворення координат?
3. Як можна класифікувати перетворення координат? Охарактеризуйте кожен тип.
4. Опишіть матричний запис афінного перетворення.
5. Охарактеризуйте властивості афінних перетворень.
6. Опишіть основні випадки афінних перетворень на площині (матричний, математичний та графічний записи)
7. Запишіть скінченну матрицю афінного перетворення для точки з координатами (x, y), яка повертається на кут α відносно центра C з координатами (a, b).
8. Запишіть скінченну матрицю афінного перетворення для точки з координатами (x, y), яка симестрично відображається відносно центра C з координатами (a, b).
9. Запишіть скінченну матрицю афінного перетворення для точки з координатами (x, y), яка повертається на кут α відносно центра C з координатами (a, b) і подальшим двократним маштабуванням.
10. Опишіть основні випадки афінних перетворень в просторі (матричний, математичний та графічний записи)
11. Охарактеризуйте зв'язок перетворень об'єкта з перетворенням координат.
12. Що таке світові та екранні координати?
13. Що таке проектування та проекції?
14. Які види проекцій ви знаєте?
15. Охарактеризуйте основну ідею центрального проектування.
16. Охарактеризуйте основну ідею паралельного проектування.
17. Що таке ортогональна проекція? Як її отримати?
18. Що таке аксонометрична проекція? Як її отримати?

РОЗДІЛ 3 ФОРМАТИ ГРАФІЧНИХ ФАЙЛІВ. АЛГОРИТМИ СТИСНЕННЯ ЗОБРАЖЕНЬ

3.1 Формати графічних файлів

Формат – спосіб організації інформації у файлі. Графічні файли служать для зберігання зображень між сеансами роботи із графічними програмами і перенесення зображень між програмами й комп'ютерами. Графічна інформація у файлах кодується трохи інакше, ніж у пам'яті комп'ютера. Більше того, способів кодування, що називаються форматами, існує безліч. Співіснування великої кількості форматів графічних файлів обумовлено специфічними сферами їхнього застосування.

Всі графічні дані в комп'ютері можна розділити на дві великі групи: растрову й векторну. Векторні файли являють собою математичний опис об'єктів відносно точки початку координат, тобто набір геометричних примітивів. Більшість векторних форматів можуть містити впроваджені у файл растрові об'єкти або посилання на растровий файл (технологія OPI). Складність при передачі даних з одного векторного формату в растровий полягає у використанні програмами різних алгоритмів, різної математики при побудові векторних і описі растрових об'єктів.

OPI (Open Prepress Interface) – технологія, розроблена фірмою Aldus, що дозволяє імпортувати не оригінальні файли, а їхні образи, створюючи в програмі лише копію з низькою роздільною здатністю (ескіз) і посилання на оригінал. У процесі друку на принтері, ескізи підмінюються на оригінальні файли. Застосування OPI, замість простого впровадження, (embedding) дає можливість заощаджувати ресурси комп'ютера (насамперед пам'ять), помітно підвищуючи його продуктивність. OPI є основною в роботі з імпортованими графічними файлами в таких програмах, як FreeHand і QuarkXPress, і широко застосовується в інших продуктах.

Растровий файл влаштований іншим чином. Він являє собою прямокутну матрицю (bitmap), розділену на маленькі квадрати – пікселі (pixel – picture element). Растрові файли можна розділити на два типи: призначені для виведення на екран і для друку.

Роздільна здатність файлів таких растрових форматів, як GIF, JPEG, BMP залежить від відеосистеми комп'ютера. Растрові формати, призначені винятково для виведення на екран, мають тільки екранну роздільну здатність, тобто один піксель у файлі відповідає одному екранному пікселю. На друк вони виводяться так само з екранною роздільною здатністю.

Растрові файли, призначені для додрукової підготовки видань, мають, подібно до більшості векторних форматів, параметр Print Size – друкований розмір. З ним пов'язане поняття друкованої роздільної

здатності, що являє собою співвідношення кількості пікселів на один квадратний дюйм сторінки (ppi – pixels per inch або dpi – dots per inch). Друкована роздільна здатність може бути від 130 dpi (для газети) до 300 dpi (високоякісний друк), більше майже ніколи не потрібно.

Растрові формати так само відрізняються один від одного здатністю нести додаткову інформацію: різні колірні моделі, вектори, альфа-канали або канали spot-кольорів, шари різних типів, інтерліньяж (через рядкове завантаження), анімація, можливості стиснення та інше.

Співіснування великої кількості форматів графічних файлів обумовлено специфічними сферами їхнього застосування. Тому можна виділити такі **параметри графічних форматів:**

- *поширеність.* Багато додатків мають власні формати файлів. Вони підтримують особливі можливості конкретних програм, але можуть виявитися несумісними з іншими додатками. Програми ілюстрування і видавничі системи можуть не вміти імпортувати такі формати або робити це некоректно. Питання поширеності стосується не тільки власних форматів програм. Деякі формати розроблялися спеціально під апаратне забезпечення (наприклад, формати Scitex, Targa, Amiga IFF). Якщо ви не маєте у своєму розпорядженні цю апаратуру, не використовуйте подібних форматів. Зберігаючи зображення в малопоширених форматах, ви створюєте потенційні проблеми при перенесенні їх на інші комп'ютери.
- *відповідність сфері застосування.* Більшість графічних форматів орієнтовано на конкретні області застосування. У випадку помилки при виборі формату зображення може виявитися непридатним для використання. Наприклад, зберігши зображення у форматі JPEG з більшим коефіцієнтом стиснення, ви зробите його непридатним для друку через втрату якості. При цьому повторне відкриття й збереження в іншому форматі не виправить допущену помилку.
- *підтримувані типи точкових зображень і колірні моделі.* Вибирайте формат файлів, що підтримує задані сферою застосування типи зображень. Наприклад, формат BMP не підтримує зображень у моделі CMYK, що потребується в поліграфії, і, отже, не може використовуватися в цій сфері. Проте варто враховувати можливість наступного перетворення типів і колірних моделей, необхідних у вибраній сфері застосування.
- *можливість зберігання додаткових колірних каналів.*
- *можливість зберігання масок.* Найчастіше маски потрібні тільки в процесі редагування. Якщо ви не завершили редагування зображення або плануєте повернутися до нього через якийсь час, зберігайте зображення разом з усіма створеними масками. Зберігання масок у вигляді альфа-каналів підтримується далеко не всіма форматами.
- *можливість зберігання обтравочних контурів.* Обтравочні контури створюються й використовуються для маскування фрагментів

зображення в програмах ілюстрування і видавничих систем. Якщо ви підготовлюєте зображення для верстання, то краще вибирати формати, що підтримують обтравочні контури. Зрозуміло, необхідно попередньо переконатися, що імпорт обтравочних контурів у видавничу систему з вибраного формату можливий і здійснюється коректно.

- *можливість стиснення графічної інформації.* Для зменшення розмірів графічних файлів багато форматів припускають стиснення даних. Вибір одного з таких форматів заощадить місце на вашому жорсткому диску і тих носіях, які ви, можливо, використовуєте для передачі файлів замовникам або підрядникам.
- *спосіб стиснення.* Формати файлів, що підтримують стиснення, використовують для цього різні алгоритми. Всі алгоритми стиснення поділяються на ті, що не приводять до втрат якості, і ті, що знижують якість зображень. Останні дозволяють досягти на порядок більше високих коефіцієнтів стиснення. Вибирайте формат, алгоритм стиснення в якому повністю відповідає сфері застосування зображень. Якщо ви плануєте використовувати їх тільки для екранного перегляду, то можете пожертвувати якістю зображення. Підготовка зображень для типографського друку не допускає зниження якості.
- *можливість зберігання каліброваної інформації.* Для точного відтворення кольорів у поліграфії використовуються системи керування кольором. У рамках наскрізного керування кольором колірні профілі вбудовуються у файли зображень. Якщо ваш виробничий процес використовує керування кольором, то при збереженні файлів варто вибирати формати, що підтримують впровадження колірних профілів.
- *можливість зберігання параметрів растрування.* Якщо ви готуєте зображення для поліграфічного тиражування й використовуєте особливі параметри растрування, то вибирайте формати файлів, що підтримують зберігання цієї інформації.

3.2 Растрові формати файлів

До растрових форматів файлів належать GIF, BMP, WBMP, PCX, PCD, PSD, PNG, TIF/TIFF, JPEG, SCT, SCT/PICT, ICO та інші. Розглянемо найпоширеніші з них.

GIF (Graphics Interchange Format)

В 1987 році фахівці компанії CompuServe створили новий формат для зберігання зображень у режимі індексованих кольорів. Формат GIF був створений найбільшою мережною службою CompuServe (нині підрозділ AOL, America OnLine) спеціально для передачі растрових зображень у глобальних мережах. В 1989 році формат був модифікований, і його нова

версія одержала назву gif89a, до нього було додано підтримку прозорості та анімації. GIF орієнтований, у першу чергу, на зберігання зображень у режимі індексованих кольорів (не більше 256), також підтримує компресію без втрат LZW, що дозволяє непогано стискати файли, у яких багато однорідних заливок (логотипи, написи, схеми).

GIF дозволяє записувати зображення «через рядок» (Interlaced), завдяки чому, маючи тільки частину файлу, можна побачити зображення повністю, але з меншою роздільною здатністю. Це досягається за рахунок запису, а потім дозавантаження, спочатку 1, 5, 10 і т. д. рядків пікселів і розтягування даних між ними, другим проходом слідують 2, 6, 11 рядки, роздільна здатність зображення в Internet-браузері збільшується. Таким чином, задовго до закінчення завантаження файлу користувач може зрозуміти, що усередині й вирішити, чи варто чекати, доки файл повністю завантажиться. Через рядковий запис незначно збільшує розмір файлу, але це, як правило, виправдується.

В GIF форматі можна призначити один або більше кольорів прозорими, вони стануть невидимими в Internet- раузерах і деяких інших програмах. Прозорість забезпечується за рахунок додаткового Alpha-каналу, що зберігається разом з файлом. Крім того файл GIF може містити не одну, а декілька растрових картинок, які браузери можуть довантажувати одну за одною із зазначеною у файлі частотою. Так досягається ілюзія руху – GIF-анімація.

Основне обмеження формату GIF полягає в тому, що кольорове зображення може бути записане тільки в режимі 256 кольорів.

Особливості:

- GIF використовує 8-бітовий колір і ефективно стискає суцільні колірні ділянки, при цьому зберігаючи деталі зображення.
- кількість кольорів в зображенні може бути від 2 до 256, але це можуть бути будь-які кольори з 24-бітової палітри.
- файл у форматі GIF може містити прозорі ділянки. Якщо використовується відмінний від білого кольору фон, він буде просвічуватися крізь прозорі ділянки в зображенні.
- GIF підтримує покадрову зміну зображень, що робить формат популярним для створення банерів і простої анімації.
- використовує вільний від втрат метод стиснення.

Область застосування:

текст, логотипи, ілюстрації з чіткими краями, анімовані рисунки, зображення з прозорими ділянками, банери.

JPEG(Joint Photographic Experts Group)

Формат JPEG уперше реалізував новий принцип стиснення із втратами інформації. Він оснований на видаленні із зображення тієї частини інформації, що слабко сприймається людським оком. Позбавлене надлишкової інформації зображення займає набагато менше місця, ніж

вихідне. Ступінь стиснення, а отже і кількість інформації, що видаляється, плавно регулюється. Найширше JPEG використовується при створенні зображень для електронного поширення або в Інтернеті. Компактність файлів JPEG робить цей формат незамінним у тих випадках, коли розмір файлів критичний, наприклад, при їхній передачі по каналах зв'язку. У поліграфії використовувати його не рекомендується, хоча формат допускає зберігання колірних профілів і контурів обтравки. JPEG підтримує півтонові й повнокольорові зображення в моделях RGB і CMYK. Не підтримуються додаткові колірні альфа-канали. Використовуйте формат JPEG тільки для зберігання фотографічних зображень. На рисунках із чіткими границями й більшими областями з однорідним кольором заливки сильно проявляються дефекти стиснення. Особливо характерний прояв бруду навколо темних ліній на світлому тлі й видимих квадратних областях. Останній дефект пов'язаний з тим, що алгоритм стиснення обробляє зображення квадратними блоками зі стороною 8 пікселів.

Існує три підформати JPEG: звичайний, optimized (файли трохи менші, але не підтримуються старими програмами) і Progressive (черезрядкове відображення, аналог interlaced в GIF). Деякі додатки дозволяють зберігати зображення в JPEG у режимі CMYK і навіть включати у файл обтравочні контури. Однак, використовувати JPEG для поліграфічних потреб категорично не рекомендується через взаємодію регулярної структури блоків 8×8 пікселів, що виходять у результаті компресії, з не менш регулярною структурою друкарського растра, що в результаті приводить до утворення муару. З довгострокового користування цим безумовно корисним форматом можна витягти дві речі. По-перше, не варто зберігати в ньому все, що потрапило, а тільки великі фотографії з більшою кількістю плавних колірних переходів. А, по-друге, у жодному разі не варто зберігати одне і те саме зображення в jpg більше одного разу: занадто помітними виявляються деструктивні зміни картинки від повторного використання компресії.

Різновид progressive JPEG дозволяє зберігати зображення з виведенням за зазначену кількість кроків (від 3 до 5 в Photoshop) – спочатку з низькою роздільною здатністю (поганою якістю), на наступних етапах первинне зображення перерисовується усе більш якісною картинкою. Анімація або прозорий колір форматом не підтримуються.

Особливості:

- в зображенні може бути понад 16 мільйонів кольорів, що цілком достатньо для збереження фотографічної якості зображення;
- основною характеристикою формату є якість, яка визначає скінченний розмір файлу. Слід пам'ятати, що формат застосовує стиснення зі втратами. Чим вище стиснення, тим менша якість і навпаки;
- підтримка технології progressive JPEG. Тоді спочатку у вікні перегляду з'являється версія рисунка з низькою роздільною

здатністю, яка поступово набуває початкового вигляду при повному завантаженні самого зображення.

Область застосування:

використовується переважно для фотографій. Не є доцільним для зображень з прозорими ділянками, дрібними деталями чи текстом.

PNG (Portable Network Graphics)

PNG – Інтернет-формат, який долає обмеження GIF. Використовує стиснення без втрат Deflate, подібне до LZW. Стиснуті індексовані файли PNG, зазвичай, є меншими за аналогічні GIF.

Глибина кольору може бути будь-якою, до 48 біт. Використовується двовимірний interlacing (не лише рядків, але і стовпців), що, подібно до GIF, дещо збільшує розмір файлу. На відміну від GIF, де застосовується 100% прозорість, PNG підтримує також напівпрозорі пікселі (в діапазоні прозорості від 0 до 99%) за рахунок альфа-каналу з 256 градаціями сірого.

У файл формату PNG записується інформація про гамму. Гамма є певним числом, що характеризує залежність яскравості світіння екрана монітора від напруги на електродах кінескопа. Це число обчислюється з файлу і дозволяє вводити поправку яскравості при відображенні. Воно потрібне для однакового відображення інформації незалежно від апаратної платформи комп'ютера. PNG підтримується у всіх сучасних браузерах.

Існує два різновиди: PNG8 і PNG24, цифри означають максимально можливу для формату глибину кольору.

PNG-8 використовує 8-бітову палітру (256 кольорів) в зображенні. При цьому можна вибирати, скільки кольорів буде використано у файлі – від 2 до 256. На відміну від GIF, не відображає анімацію.

PNG-24 – формат, аналогічний до PNG-8, але використовує 24-бітову палітру кольору. Подібно до формату JPEG, зберігає яскравість і відтінки кольорів у фотографіях. Подібно до форматів GIF і PNG-8, зберігає деталі зображення, як, наприклад, в лінійних рисунках, логотипах, або ілюстраціях. Використовує понад 16 млн кольорів, тому застосовується для повнокольорових зображень. Підтримує багаторівневу прозорість, це дозволяє створювати плавний перехід від прозорої області зображення до колірної, так званий градієнт. Використаний алгоритм стиснення зберігає всі кольори і пікселі в зображенні незмінними. Якщо порівнювати з іншими форматами, то в PNG-24 скінченний об'єм графічного файлу виходить найбільшим.

BMP (Windows Device Independent Bitmap)

Растровий формат BMP, створений Microsoft, орієнтований на застосування в операційній системі Windows. Він підтримується всіма графічними редакторами, що працюють під керуванням цієї операційної системи. Застосовується для зберігання растрових зображень, призначених для використання в Windows і, по суті, більше ні на що не придатний. Здатний зберігати як індексовані (до 256 кольорів), так і RGB-колір

(більше 16 млн відтінків). Можливе застосування стиснення за принципом RLE, але робити це не рекомендується, тому що дуже багато програм таких файлів (вони можуть мати розширення .rle) не розуміють.

Використання BMP не для потреб Windows є розповсюдженою помилкою новачків. Важливо зрозуміти, що використовувати BMP не бажано ні у web, ні для друку, ні для простого перенесення й зберігання інформації.

TIF, TIFF (Tagged Image File Format)

Формат TIFF (Tagged Image File Format) створений об'єднаними силами компаній Aldus, Microsoft і Next спеціально для зберігання сканованих зображень. Виняткова гнучкість формату зробила його дійсно універсальним. TIFF – один із найдревніших форматів у світі мікрокомп'ютерів, на сьогоднішній день він є найгнучкішим, універсальним і активно розвивається. У ньому можна зберігати графіку в будь-якому режимі: від бітових і індексованих кольорів до Lab, CMYK і RGB (крім дуплексів і багатоканальних документів). Хоча з моменту його створення пройшло вже багато часу, TIFF дотепер є основним форматом, що використовується для зберігання сканованих зображень і розміщення їх у видавничих системах і програмах ілюстрування. Версії формату існують на всіх комп'ютерних платформах, що робить його винятково зручним для перенесення растрових зображень між ними.

TIFF підтримує монохромні, індексовані, півтонові і повнокольорові зображення в моделях RGB і CMYK з 8- і 16-бітовими каналами. Він дозволяє зберігати обтравочні контури, калібровану інформацію, параметри друку. Допускається використання будь-якої кількості додаткових альфа-каналів. Додаткові колірні канали не підтримуються. Великою перевагою формату залишається підтримка практично будь-якого алгоритму стиснення. Найпоширенішим є стиснення без втрат інформації за алгоритмом LZW, що забезпечує дуже високий ступінь компресії.

PSD (PhotoShop Document)

Формат PSD – це власний формат програми Adobe Photoshop. Єдиний формат, що підтримує всі можливості програми. Кращий для зберігання проміжних результатів редагування зображень, тому що зберігає їхню пошарову структуру. Всі останні версії продуктів фірми Adobe Systems підтримують цей формат і дозволяють імпортувати файли Photoshop безпосередньо. До недоліків формату PSD можна віднести недостатню сумісність із іншими розповсюдженими додатками й відсутність можливості стиснення.

Підтримуються всі колірні моделі й будь-яка глибина кольору від білочорного до true color, стиснення без втрат. Починаючи з версії 3.0, Adobe додала підтримку шарів і контурів, тому формат версії 2.5 і раніше виділяється в окремий підформат. Для сумісності з ним у більш пізніх версіях Photoshop є можливість вмістити режим додавання у файл одного

базового шару, у якому злиті всі шари. Такі файли вільно читаються більшістю популярних переглядачів, імпортуються в інші графічні редактори й програми для 3D-моделювання.

3.3 Векторні формати файлів

До векторних форматів файлів належать WMF, EMF, EPS, WPG, AutoCAD, DXF, DWG, CDR, AI, PCT, FLA/SWF та інші, розглянемо найпоширеніші з них.

WMF (Windows Metafile)

Векторний формат WMF використовує графічну мову Windows і є її рідним форматом. Служить для передачі векторів через буфер обміну (Clipboard). Читається практично всіма програмами Windows, які певним чином пов'язані з векторною графікою. Однак, незважаючи на простоту і універсальність, користуватися форматом WMF варто лише в крайніх випадках, оскільки WMF спотворює колір, не може зберігати ряд параметрів, які можуть бути присвоєні об'єктам у різних векторних редакторах, не може містити растрові об'єкти, не читається дуже багатьма програмами на Macintosh.

EPS (Encapsulated PostScript)

Завдяки своїй надійності, сумісності з багатьма програмами і платформами, а також багатьма параметрами, що можна налаштувати, формат EPS є вибором більшості професіоналів в області поліграфії. Він призначений лише для перенесення готових зображень у програми верстання, підтримує колірні моделі CMYK, RGB, дуплекси й містить готові команди для пристрою виведення. Дані зберігаються трьома способами: ASCII (повільний, але найбільш сумісний), Binary (швидкий і компактний), JPEG (швидкий, але із втратами якості й поганою сумісністю). При збереженні в eps можна вказати формат і глибину кольору ескізу, що для прискорення роботи буде виводитися на екран у програмах верстання замість великого оригіналу. Ескізом (image header, preview) є копія з низькою роздільною здатністю у форматі PICT, TIFF, JPEG або WMF, що зберігається разом з файлом EPS і дозволяє побачити, що усередині, оскільки відкрити файл для редагування можуть тільки Photoshop і Illustrator. Спочатку EPS розроблявся як векторний формат, пізніше з'явився його растровий різновид – Photoshop EPS.

Дані в EPS записуються відповідно до стандарту DSC (Document Structuring Conventions) і ряду розширень, що дозволяють використовувати EPS для запису графічної інформації. Формат EPS найчастіше використовують при друці на PostScript пристроях (принтерах), при передачі PostScript зображень між програмами, при збереженні зображень для їх використання програмами верстання, а також при передачі зображень у друкарні. Але цим не обмежується область застосування

Encapsulated PostScript, він є одним з найбільш популярних форматів, внаслідок його універсальності й надійності. EPS має безліч переваг над іншими форматами, з яких можна виділити такі:

- EPS здатний містити растрові, векторні зображення і їхні комбінації, шрифти, контури обтравки, криві калібрувань, текст;
- EPS підтримує всі необхідні для друку колірні моделі;
- підтримує Duotone;
- файл формату EPS може бути створений і відкритий більшою кількістю програм, що працюють із графікою;
- можливість зберігання Preview;
- можливість зберігання інформації про розміри зображення;
- формат забезпечує дуже високу якість рисунків.

Однак у форматі також є деякі недоліки. У зв'язку з тим, що EPS має велику кількість версій, виникають проблеми з підтримкою програмами різних версій EPS, не всі додатки здатні працювати зі свіжими версіями EPS, та й не всі програми вміють створювати коректний EPS- файл, з яким надалі можна буде працювати в інших програмах. На даний момент існує три версії мови PostScript, яка є основою EPS, що також не додає стабільності. Файли EPS досить великі, тому що крім оригінального зображення, файл зберігає растрову копію – ескіз.

AI (Adobe Illustrator Document)

AI (Adobe Illustrator Document) – векторний формат програми Adobe Illustrator, використовується для створення рисунків, двовимірних об'єктів, графічних елементів (креслення, декоративні написи тощо) як простих, так і дуже складних.

Формат AI вийшов практично відразу після виходу PostScript Level 1, що ліг у його основу (його називають інтерфейсом для PS, у багатьох програмах формат Adobe Illustrator Document визначається як Generic EPS). Тому все, що створює Adobe Illustrator, сумісне з PostScript. Аж до 9-ї версії ядро формату ґрунтувалося на форматі EPS, після чого основою став формат PDF, після чого стала можлива підтримка багатосторінковості.

Формат AI відкривається Photoshop, його підтримують майже всі програми Macintosh і Windows, які пов'язані з векторною графікою і графікою взагалі. Однак при відкритті в растровому редакторі документ растеризується, а також не всі ефекти Adobe Illustrator можуть бути відображені в інших програмах або роздруковані на будь-якому принтері PostScript. Зі складними файлами AI можуть виникати проблеми при їхньому відображенні, тому радять, по можливості, спрощувати зображення, так, наприклад, практично все зображення можна реконструювати простими операціями.

CDR (CorelDraw Document)

CDR – рідний формат програми CorelDraw (для ОС Mac і Windows),

здатний зберігати інформацію про векторні форми, растрові й векторні прошарки, кольори. Підтримує текст, вбудовування шрифтів, розмір зображення до 45 квадратних метрів (45×45), зберігання більше однієї сторінки. Найчастіше його використовують як проміжний, хоча він і підтримується меншою кількістю програм, ніж AI. Може бути імпортований багатьма програмами, велике робоче поле є важливим параметром у роботі із плакатами, вивісками, зовнішньою рекламою, що робить цей формат популярним у типографії, хоча частіше рекомендують використовувати універсальний формат EPS (Encapsulated PostScript). CDR, створений в CorelDraw 10 або більш ранньої версії редактора, відкривається Adobe Illustrator'ом, а також відкривається в інших програмах, зроблених в Corel, однак для кращої сумісності радять зберігати файли в CDR дев'ятої версії. Так само, як і формат AI, кожна нова версія CDR не підтримується попередніми версіями програми, а кожна нова версія програми підтримує всі попередні версії формату. На цьому їхні подібності не закінчуються: при відображенні формату CDR в інших програмах (наприклад в Adobe Illustrator) виникають проблеми, так векторна графіка залишається без спотворень, а у всіх інших випадках відбувається часткова втрата даних.

3.4 Комплексні формати графічних файлів

Adobe PostScript

PostScript – мова опису сторінок (мова керування лазерними принтерами) фірми Adobe. Формат був створений в 80-х роках для реалізації принципу WYSIWYG (What You See is What You Get). Файли цього формату являють собою програму з командами на виконання для вивідного пристрою. Вони мають розширення .ps або, рідше, .prn і створюються за допомогою функції Print to File графічних програм при використанні драйвера PostScript-принтера. Такі файли містять у собі сам документ (тільки те, що розташовувалося на сторінках), всі пов'язані файли (як растрові, так і векторні), використані шрифти, а так само іншу інформацію: плати кольороподілу, додаткові плати, форму растрової точки для кожної плати та інші дані для вивідного пристрою.

Дані в PostScript-файлі, як правило, записуються у двійковому кодуванні (Binary). Бінарний код займає вдвічі менше місця, ніж ASCII. PostScript дав можливість для високоякісного друку (у тому числі й векторного), універсальна мова керування для принтерів, що дозволяє на одній сторінці з'єднувати векторну, растрову графіку й текст. Також PostScript можуть містити шрифти, захищені авторськими правами, підтримується багатьма програмами для верстання та різними графічними редакторами. Крім того PostScript лежить в основі таких форматів, як EPS і PDF.

CGM (Computer Graphics Metafile)

CGM – формат, створений технічним комітетом при міжнародних організаціях зі стандартизації ISO і ANSI. Відповідний йому метафайловий формат (зараз існує кілька різновидів цього формату, несумісних один з одним) відображення векторних зображень на Web був прийнятий наприкінці 1998 р. консорціумом W3C.

CGM – розповсюджений стандарт для двовимірної графіки, він працює як з векторною графікою, так і з растровою, тому може використовуватися як альтернатива форматам GIF і JPEG, а також є форматом для публікації даних систем автоматизованого проектування в компактній, ефективній і інтерактивній формі. Він є загальноприйнятим у промислових і урядових організаціях, а також використовується в картографії, технічній ілюстрації, комп'ютерних видавничих системах. CGM підтримує гіперзв'язки графічних об'єктів усередині ілюстрації, що робить його добре сумісним з мовою HTML. Використовуючи CGM і HTML, автори можуть легко створювати високоякісні інтерактивні компактні документи, до яких можна легко одержати доступ через WWW. Багато векторних редакторів, настільних видавничих систем, систем автоматизованого проектування й інших програм здатні імпортувати CGM.

CGM містить безліч графічних примітивів і атрибутів, дозволяє створювати компактні файли й підтримує обмін дуже складними й високомистецькими зображеннями, не дивлячись на це, він менш складний, ніж PostScript. Формат використовує різні види стиснення (наприклад RLE, CCITT Group 3 і Group 4), його колірна палітра не обмежена. Один файл CGM може містити декілька зображень.

PDF (Portable Document Format)

PDF запропонований фірмою Adobe як незалежний від платформи формат для створення електронної документації, презентацій, передачі графіки через мережі. Використовується як внутрішній графічний формат в Mac OS X.

PDF-файли створюються шляхом конвертації з PostScript-файлів або функцією експорту ряду програм. Для конвертації використовується програма Adobe Acrobat Distiller, це кращий спосіб створення PDF. Створення PDF методом експорту із програм дає, як правило, гірший результат – файли виходять більш важкими, часто мають проблеми із вбудовуванням шрифтів.

В 2007 році Міжнародною організацією зі стандартів (ISO) формат PDF версії 1.7 був прийнятий як міжнародний стандарт ISO 32000-1. Популярність цього формату не випадкова, PDF має безліч переваг:

- крос-платформність;
- відкритість специфікацій формату;
- компактність;
- підтримка тексту, гіперпосилань, зображень і мультимедіа;

- підтримка RGB і CMYK;
- аналогічність із ідеями WYSIWYG;
- використання ряду можливостей мови PostScript;
- можливість конвертування в PDF з багатьох інших форматів.

Крос-платформність робить формат універсальним для передачі інформації між різними операційними системами; відкритість специфікацій дозволяє створювати безліч додатків, здатних працювати з PDF; компактність виражається в тому, що всі дані можуть стискатися, причому для кожного типу інформації використовується найбільш підходяще для цього тип стиснення: JPEG, RLE, CCITT, ZI; PDF може містити в собі текст, гіперпосилання, шрифти, заповнювані поля, растрові й векторні зображення, мультимедіа-вставки (відео й звук), попередній перегляд (preview); підтримка CMYK дозволяє формату використовуватися в поліграфії; PDF підтримує механізм електронних підписів для захисту дійсності документів, також виготовлювач документа може заборонити зміну файлу, його друк, а також установити пароль; PDF являє собою електронну роздруківку документа, тобто версія документа на екрані не буде відрізнятися від роздрукованої версії. Гнучкість даного формату крім перерахованих позитивних моментів, також має й негативні. PDF уступає спеціалізованим форматам; складний у редагуванні, він не зберігає логічну структуру документів, тому що спрямований на візуалізацію документів; також PDF є патентованим форматом компанії Adobe.

PICT (Macintosh QuickDraw Picture Format)

Метафайловий формат PICT (позначається як pict, .pic, .pct) був розроблений в 1984 році компанією Apple Inc. як внутрішній формат графіки для Macintosh, його використовує буфер обміну при передачі графічних даних між програмами. Він кодується командами QuickDraw. PICT здатний містити растрову й векторну інформацію, текст, а також звук, що працює тільки на Macintosh. Однак PICT, як векторний формат, практично не використовується, тому що були проблеми з товщиною лінії й інші відхилення при друці, його використовує сама Mac OS, а також ним іноді користуються при створенні певних типів презентацій тільки на Mac. Растровий PICT придатний під час редагування зображення, але не більше. PICT підтримує 1, 4, 8, 16, 24 і навіть 32- бітові кольори (однак для CMYK останні 8 біт використовуються для альфа-каналу), графічні примітиви, Bitmap, Grayscale, Indexed colors, RGB I CMYK, причому в RGB можна зберегти один альфа-канал, в інших режимах – декілька; використовується стиснення без втрат Run-Lengt Encoding (RLE), ефективне при наявності більших областей одного кольору, а також існує можливість стиснення з використанням JPEG. Всі додатки, створені під Mac, підтримують його, на PC кількість програм, здатних працювати з даним форматом мало, а коректно працюючих програм ще менше. PICT – формат досить складний,

тому не має широкої підтримки на інших платформах. Програми верстання також можуть використовувати цей формат, але краще використовувати TIFF або EPS, тому що деякі програми не здатні ефективно працювати із зображеннями в .pct (InDesign, QuarkXPress і PageMaker).

3.5 Конвертація файлів з одного формату в інший

Зазвичай графічні програми використовують свої власні формати для збереження зображень у зовнішній пам'яті. Власний файловий формат – приватний і найбільш ефективний формат для зберігання файлів окремого графічного додатка. Наприклад, «рідний» формат CorelDRAW – CDR, Adobe PhotoShop – PSD, Paint – BMP. При збереженні зображення у файлі завжди потрібно вказувати тип формату. Крім того, для кожного «чужого» графічного формату відкриваються додаткові діалогові вікна, за допомогою яких користувач встановлює параметри формату (кількість використовуваних кольорів, необхідність стиснення – для BMP і TIFF, коефіцієнт стиснення – для JPEG та ін.)

Тому через ряд причин виникає необхідність конвертації графічних файлів з одного формату в інший:

– програма, з якою працює користувач, не сприймає формат його файлу;

– дані, які треба передати іншому користувачеві, повинні бути подані в спеціальному форматі.

Конвертація файлів з растрового формату у векторний

Існують два способи конвертації файлів з растрового формату у векторний:

1. Конвертація растрового файлу в растровий об'єкт векторного зображення;
2. Трасування растрового зображення для створення векторного об'єкта.

Перший спосіб використовується в програмі CorelDRAW, яка успішно імпортує файли різних растрових форматів. До прикладу, якщо растрова картинка містить 16 мільйонів кольорів, CorelDRAW покаже зображення, наближене за якістю до телевізійного. Однак імпортований растровий об'єкт може стати досить великим навіть у тому випадку, якщо вихідний файл невеликий. У файлах растрових форматів інформація зберігається досить ефективно, оскільки часто використовуються методи стиснення. Векторні формати такої здатності не мають. Тому растровий об'єкт, що зберігається у векторному файлі, може значно перебільшувати за розмірами вихідний растровий файл.

Особливість другого способу перетворення растрового зображення у векторне полягає в тому, що програма трасування растрових зображень (наприклад CorelTRACE) шукає групи пікселів з однаковим кольором, а

потім створює відповідні їм векторні об'єкти. Після трасування векторизовані рисунки можна редагувати як завгодно.

Конвертація файлів одного векторного формату в інший

Векторні формати містять описи ліній, дуг, зафарбованих полів, тексту і т. д. У різних векторних форматах ці об'єкти описуються по-різному. Коли програма намагається перетворити один векторний формат в інший, вона діє так:

- зчитує опис об'єктів однією векторною мовою;
- намагається перевести їх на мову нового формату.

Якщо програма-перекладач вважає опис об'єкта таким, для якого в новому форматі немає точної відповідності, цей об'єкт може бути або описаний схожими командами нової мови, або не описаний взагалі. Таким чином, деякі частини рисунка можуть спотворитися або зникнути. Все залежить від складності вихідного зображення.

Конвертація файлів з векторного формату в растровий

Конвертація зображень з векторного формату в растровий (цей процес часто називають раструванням векторного зображення) зустрічається дуже часто. Перш ніж розмістити нарисовану (векторну) картинку на фотографії, її необхідно експортувати в растровий формат. При експорті векторних файлів в растровий формат може бути втрачена інформація, пов'язана з кольором вихідного зображення. Це пояснюється тим, що в ряді растрових форматів кількість кольорів обмежена (наприклад, формат GIF використовує не більше 256 кольорів).

Конвертація файлів одного растрового формату в інший

Цей вид конвертації зазвичай найпростіший і полягає в читанні інформації з вихідного файлу і запису її в новому файлі, де дані про розмір зображення, бітову глибину і колір кожного відеопікселя зберігаються іншим способом. Якщо старий формат використовує більше кольорів, ніж новий, то можлива втрата інформації. Перетворення файлу з 24-бітовим кольором (16777216 кольорів) у файл з 8-бітовим кольором (256 кольорів) потребує зміни кольору майже для кожного пікселя. У найпростішому випадку це робиться так: для кожного пікселя вихідного файлу шукається найбільш близький до нього колір з нового обмеженого набору кольорів. При такому способі можливі небажані ефекти, коли частина рисунка, що містить велику кількість елементів, виявляється зафарбованою одним кольором або коли плавні переходи кольору стають різкими.

Для конвертації файлів з одного формату в інший використовуються спеціальні програми – перетворювачі (конвертори) форматів. Однак більшість графічних програм (CorelDRAW, Adobe Illustrator, Adobe PhotoShop та ін) можуть читати й створювати файли різних форматів, тобто також виконувати функції конверторів форматів.

3.6 Алгоритми стиснення зображень

Стиснення – це більш ефективне подання інформації, що базується на трьох узагальнених властивостях графічних даних: надмірності, передбачуваності й необов'язковості.

Зображення (як і відео) займають набагато більше місця в пам'яті комп'ютера, ніж текст. Як приклад можна розглянути те, скільки тисяч сторінок тексту ми зможемо помістити на CD-ROM, і як мало там поміститься якісних фотографій, що не піддавалися стисненню. Ця особливість зображень визначає актуальність алгоритмів архівації графіки.

Другою особливістю зображень є те, що людський зір при аналізі зображення оперує контурами, загальним переходом кольорів і є порівняно невідчутним до малих змін у зображенні. Таким чином, ми можемо створити ефективні алгоритми стиснення зображень, у яких стиснуте зображення не буде збігатися з оригіналом, однак людина цього не помітить. Дана особливість людського зору дозволила створити спеціальні алгоритми стиснення, орієнтовані тільки на зображення. Ці алгоритми мають дуже високі характеристики.

Можна легко помітити, що зображення, на відміну від тексту, характеризується надмірністю в 2-х вимірах. Тобто, як правило, сусідні точки як по горизонталі, так і по вертикалі в зображенні близькі за кольором. Крім того, ми можемо використовувати подібність між колірними площинами R, G і B, що дає можливість створити ще більш ефективні алгоритми.

Стиснення даних – це один з типів кодування, що застосовується для зменшення обсягу даних. Інші типи кодування містять шифрування (криптографію) і передачу даних (наприклад, абетка Морзе). Програма-компресор здійснює стиснення даних, а програма-декомпресор – їхнє відновлення. Декомпресор може виконувати свої функції тільки на базі алгоритму стиснення, що був використаний для перетворення оригінальних даних у стислу форму.

3.6.1 Алгоритми стиснення без втрат

Алгоритм RLE. Даний алгоритм надзвичайно простий у реалізації. Групове кодування – від англійського Run Length Encoding (RLE) – один із найстаріших і найпростіших алгоритмів архівації графіки. Зображення в ньому витягується в ланцюжок байтів по рядках растра. Саме стиснення в RLE відбувається за рахунок того, що у вихідному зображенні зустрічаються ланцюжки однакових байтів. Заміна їх на пари <лічильник повторень, значення> зменшує надмірність даних.

Алгоритм декомпресії при цьому виглядає так:

```
Initialization(...);
do {
    byte = ImageFile.ReadNextByte();
```

```
    if(является счетчиком(byte)) {
            counter = Low6bits(byte)+1;
            value = ImageFile.ReadNextByte();
            for(i=1 to counter)
                    DecompressedFile.WriteByte(value)
    }
    else {
    DecompressedFile.WriteByte(byte)
} while(ImageFile.EOF());
```

У даному алгоритмі ознакою лічильника (counter) служать одиниці у двох верхніх бітах зчитаного файла. Відповідно 6 біт, що залишилися, витрачаються на лічильник, що може приймати значення від 1 до 64. Рядок з 64 повторюваних байтів перетворюється у два байти, тобто стискається в 32 рази. На рис. 3.1 наведено растрове подання зображення з використанням групового стиснення.

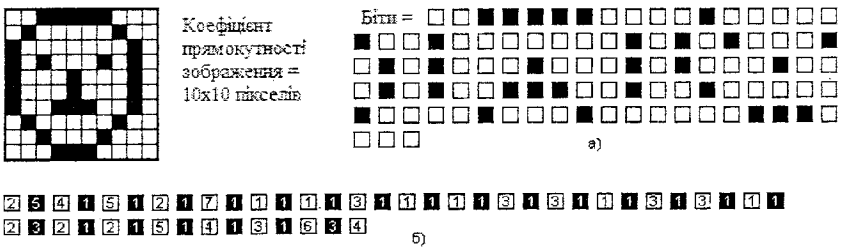


Рисунок 3.1 – Растрове подання зображення з використанням групового стиснення (RLE): а) повне подання; б) стиснуте подання

Коефіцієнти компресії: 32, 2, 0,5 (кращий, середній, гірший коефіцієнти).

Клас зображень: орієнтовано алгоритм на зображення з невеликою кількістю кольорів – на ділову й наукову графіку.

Характерні особливості: до переваг алгоритму можна віднести тільки те, що він не потребує додаткової пам'яті при архівації й розархівації, а також забезпечує високу швидкодію. Цікава особливість групового кодування полягає в тому, що ступінь архівації для деяких зображень може бути істотно підвищений всього лише за рахунок зміни порядку кольорів у палітрі зображення.

Алгоритм LZW. Свою назву алгоритм отримав за першими буквами прізвищ його розробників – Lempel, Ziv і Welch.

Існує досить велике сімейство LZ-подібних алгоритмів, що відрізняються, наприклад, методом пошуку повторюваних ланцюжків.

Розглянутий нижче варіант алгоритму буде використовувати дерево для подання й зберігання ланцюжків. Очевидно, що це досить сильне

обмеження на вид ланцюжків, і далеко не всі однакові підланцюжки в даному зображенні будуть використані при стисненні. Однак у пропонованому алгоритмі вигідно стискати навіть ланцюжки, що складаються з 2 байт.

Процес стиснення виглядає так: послідовно зчитуються символи вхідного потоку й перевіряється, чи є в створеній таблиці рядків шуканий рядок. Якщо рядок є, то зчитується наступний символ, а якщо рядка немає, то в потік заноситься код для попереднього знайденого рядка, рядок заноситься у таблицю й починається пошук знову.

Функція InitTable() очищує таблицю й поміщає в неї всі рядки одиничної довжини.

```
InitTable();
CompressedFile.WriteCode(ClearCode);
CurStr=пустая строка;

while(не ImageFile.EOF()){    // Доки не кінець файлу
    C=ImageFile.ReadNextByte();
    if(CurStr+C есть в таблице)
            CurStr=CurStr+C; // Приклеїти символ до рядка
    else {
            code=CodeForString(CurStr);//code-не байт!
            CompressedFile.WriteCode(code);
            AddStringToTable (CurStr+C);
            CurStr=C;      // Рядок з одного символу
    }
}
code=CodeForString(CurStr);
CompressedFile.WriteCode(code);
CompressedFile.WriteCode(CodeEndOfInformation);
```

Функція InitTable() ініціалізує таблицю рядків так, щоб вона містила всі можливі рядки, що складаються з одного символу. Наприклад, якщо стискаються байтові дані, то таких рядків у таблиці буде 256 («0», «1», ... , «255»). Для коду чищення (ClearCode) і коду кінця інформації (CodeEndOfinformation) зарезервовані значення 256 і 257. У розглянутому варіанті алгоритму використовується 12-бітовий код і, відповідно, підкоди, для рядків залишаються значення від 258 до 4095.

Рядки, що додаються, записуються в таблицю послідовно, при цьому індекс рядка в таблиці стає її кодом. Функція ReadNextByte() читає символ з файлу. Функція WriteCode() записує код у вихідний файл. Функція AddStringToTable () додає новий рядок у таблицю, приписуючи їй код. Крім того, у даній функції відбувається обробка ситуації переповнення таблиці. У цьому випадку в потік записується код попереднього знайденого рядка й код чищення, після цього таблиця очищається функцією InitTable(). Функція CodeForString() знаходить рядок у таблиці й видає код цього рядка.

Приклад 3.1

Нехай потрібно стиснути послідовність 45, 55, 55, 151, 55, 55, 55. Тоді, відповідно до викладеного вище алгоритму, у вихідний потік спочатку потрібно помістити код чищення <256>, потім додати до початково пустого рядка «45» і перевірити, чи є рядок «45» у таблиці. Оскільки при ініціалізації в таблицю було занесено всі рядки з одного символу, то рядок «45» є в таблиці. Далі потрібно зчитати наступний символ 55 із вхідного потоку й перевірити, чи є рядок «45, 55» у таблиці. Такого рядка в таблиці поки немає. Тому в таблицю заноситься рядок «45, 55» (з першим вільним кодом 258) і в потік записується код <45>. Можна коротко подати стискання так:

«45» – є в таблиці;

«45, 55» – немає. Додаємо в таблицю <258> «45, 55». У потік: <45>;

«55, 55» – немає. У таблицю: <259> «55, 55». У потік: <55>;

«55, 151» – немає. У таблицю: <260> «55, 151». У потік: <55>;

«151, 55» – немає. У таблицю: <261> «151, 55». У потік: <151>;

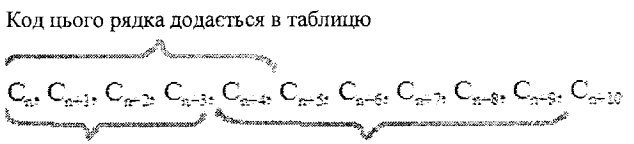
«55, 55» – є в таблиці;

«55, 55, 55» – немає. У таблицю: «55, 55, 55» <262>. У потік: <259>;

Послідовність кодів для даного прикладу, що попадають у вихідний потік: <256>, <45>, <55>, <55>, <151>, <259>.

Особливість LZW полягає в тому, що для декомпресії не треба зберігати таблицю рядків у файл для розпаковування. Алгоритм побудований таким чином, що можна відновити таблицю рядків, користуючись тільки потоком кодів.

Для кожного коду треба додавати в таблицю рядок, що складається із уже наявного там рядка й символу, з якого починається наступний рядок у потоці.



Алгоритм декомпресії, що здійснює цю операцію, виглядає так:

```
code=File.ReadCode();
while(code != CodeEndOfInformation){
    if(code = ClearCode) {
        InitTable();
        code=File.ReadCode();
            if(code = CodeEndOfInformation)
                {закончить работу};
        ImageFile.WriteString(StrFromTable(code));
        old_code=code;
    else {
```

```
            if(InTable(code)) {
                ImageFile.WriteString(FromTable(code));
                AddStringToTable(StrFromTable(old_code)+
              FirstChar(StrFromTable(code)));
                old_code=code;
            }
            else {
                OutString= StrFromTable(old_code)+
                  FirstChar(StrFromTable(old_code));
                ImageFile.WriteString(OutString);
                AddStringToTable(OutString);
                old_code=code;
            }
        }
    }
```

Функція ReadCode() читає черговий код з файла, що підлягає декомпресії. Функція InitTable() виконує ті ж дії, що й при компресії, тобто очищує таблицю й заносить у неї всі рядки з одного символу. Функція FirstChar() видає перший символ рядка. Функція StrFromTable() видає рядок з таблиці за кодом. Функція AddStringToTable() додає новий рядок у таблицю (присвоюючи їй перший вільний код). Функція WriteString() записує рядок у файл.

Коефіцієнти компресії: приблизно 1000, 4, 5/7 (кращий, середній, гірший коефіцієнти). Стиснення в 1000 разів досягається тільки на одноколірних зображеннях розміром, кратним приблизно 7 Мб (6.918...).

Клас зображень: орієнтований LZW на 8-бітові зображення. Стискає за рахунок однакових підланцюжків у потоці.

Характерні риси: ситуація, коли алгоритм збільшує зображення, зустрічається вкрай рідко. LZW універсальний – саме його варіанти використовуються у звичайних архіваторах.

Алгоритм Хаффмана

Один із класичних алгоритмів, відомих з 60-х років. Використовує лише частоту появи однакових байт у зображенні. Порівнює символи вхідного потоку, які зустрічаються найчастіше, з ланцюжком бітів меншої довжини. І, навпаки, ті символи, що зустрічається рідко – з ланцюжком більшої довжини. Для збирання статистики потребує двох проходів для зображення.

Для початку потрібно розглянути кілька визначень.

Визначення. Нехай задано алфавіт $\Psi=\{a_1, \ldots, a_r\}$, що складається з скінченного числа букв. Скінченна послідовність символів з Ψ

$$A=a_{i_1}a_{i_2}\ldots a_{i_n}$$

називається словом в алфавіті Ψ, а число n – довжиною слова A. Довжина

слова позначається як $l(A)$.

Нехай задано алфавіт Ω, $\Omega=\{b_1, ..., b_q\}$. Як B потрібно позначити слово в алфавіті Ω і через $S(\Omega)$ – множину всіх непустих слів в алфавіті Ω.

Нехай $S = S(\Psi)$ – множина всіх непустих слів в алфавіті Ψ, і S' – деяка підмножина множини S. Нехай також задано відображення F, яке кожному слову A, $A \in S(\Psi)$ ставить у відповідність слово $B=F(A)$, $B \in S(\Omega)$.

Слово B називається *кодом повідомлення* A, а перехід від слова A до його коду – *кодуванням*.

Відповідність між буквами алфавіту Ψ і деякими словами алфавіту Ω буде описуватись:

$$a_1 - B_1,$$
$$a_2 - B_2,$$
$$\dots$$
$$a_r - B_r.$$

Цю відповідність називають схемою і позначають через Σ. Вона визначає кодування таким чином: кожному слову $A=a_{i_1}a_{i_2}...a_{i_n}$ з $S'(\Omega) = S(\Omega)$ ставиться у відповідність слово $B=B_{i_1}B_{i_2}...B_{i_n}$, що називається *кодом слова* A. Слова B_1 ... B_r називаються *елементарними кодами*. Даний вид кодування називають алфавітним кодуванням.

Визначення. Нехай слово B має вигляд:

$$B = B'B''.$$

Тоді слово B' називається *початком або префіксом слова* B, а B'' – *кінцем слова* B. При цьому пусте слово Λ і саме слово B вважаються початком і кінцем слова B.

Визначення. Схема Σ має властивість префікса, якщо для будь-яких i та j $(1 \le i, j \le r, i \ne j)$ слово B_i не є префіксом слова B_j.

Теорема 1. Якщо схема Σ має властивість префікса, то алфавітне кодування буде взаємно однозначним.

Нехай задано алфавіт $\Psi=\{a_1, ..., a_r\}$ $(r>1)$ і набір ймовірностей $p_1, \dots,$ $p_r \left(\sum_{i=1}^{r} p_i = 1\right)$ появи символів $a_1, ..., a_r$. Далі задано алфавіт Ω, $\Omega=\{b_1, ..., b_q\}$ $(q>1)$. Тоді можна побудувати цілий ряд схем Σ алфавітного кодування:

$$a_1 - B_1,$$
$$\dots$$
$$a_r - B_r,$$

що мають властивість взаємної однозначності.

Для кожної схеми можна ввести середню довжину $l_{ср}$, визначену як математичне сподівання довжини елементарного коду:

$$l_{cp} = \sum_{i=1}^{r} p_i l_i, \qquad l_i = l(B_i) \text{ – довжини слів.}$$

Довжина $l_{\text{ср}}$ показує, у скільки разів збільшується середня довжина слова при кодуванні зі схемою Σ.

Можна показати, що $l_{\text{ср}}$ досягає величини свого мінімуму l_* на деякій схемі Σ і визначена як

$$l_* = \min_{\Sigma} l_{cp}^{\Sigma}.$$

Визначення. Коди, визначені схемою Σ з $l_{\text{ср}} = l_*$, називаються кодами з мінімальною надмірністю або *кодами Хаффмана.*

Коди з мінімальною надмірністю дають у середньому мінімальне збільшення довжин слів при відповідному кодуванні.

У даному випадку алфавіт $\Psi=\{a_1, ..., a_r\}$ задає символи вхідного потоку, а алфавіт $\Omega=\{0,1\}$, тобто складається всього з нуля й одиниці.

Алгоритм побудови схеми Σ можна подати таким способом:

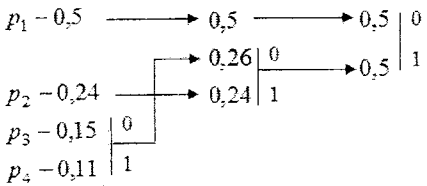
1. Упорядкувати всі букви вхідного алфавіту в порядку спадання імовірності. Вважати всі відповідні слова B_i з алфавіту $\Omega=\{0,1\}$ порожніми.

2. Об'єднати два символи a_{ir-1} і a_{ir} з найменшими ймовірностями p_{ir-1} і p_{ir} у псевдосимвол $a'\{a_{ir-1}, a_{ir}\}$ з імовірністю $p_{ir-1}+p_{ir}$. Дописати 0 у початок слова B_{ir-1} ($B_{ir-1}=0B_{ir-1}$), і 1 у початок слова B_{ir} ($B_{ir}=1B_{ir}$).

3. Видалити зі списку впорядкованих символів a_{n-1} і a_{ir}, занести туди псевдосимвол $a'\{a_{ir-1}, a_{ir}\}$. Виконати крок 2, додаючи при необхідності одиницю або нуль для всіх слів B_i, що відповідають псевдосимволам, доти, поки в списку не залишиться один псевдосимвол.

Приклад 3.2

Нехай є 4 букви в алфавіті $\Psi=\{a_1, ..., a_4\}$ ($r = 4$), $p_1 = 0{,}5$, $p_2 = 0{,}24$, $p_3 = 0{,}15$, $p_4 = 0{,}11$ $\left(\sum_{i=1}^{4} p_i = 1\right)$. Тоді процес побудови схеми можна подати так:



Виконуючи дії, що відповідають 2-му кроку, можна отримати псевдосимвол з імовірністю 0,26 (і приписуємо 0 і 1 відповідним словам). Повторюючи ці дії для зміненого списку, одержуємо псевдосимвол з імовірністю 0,5. І, нарешті на останньому етапі одержимо сумарну ймовірність 1.

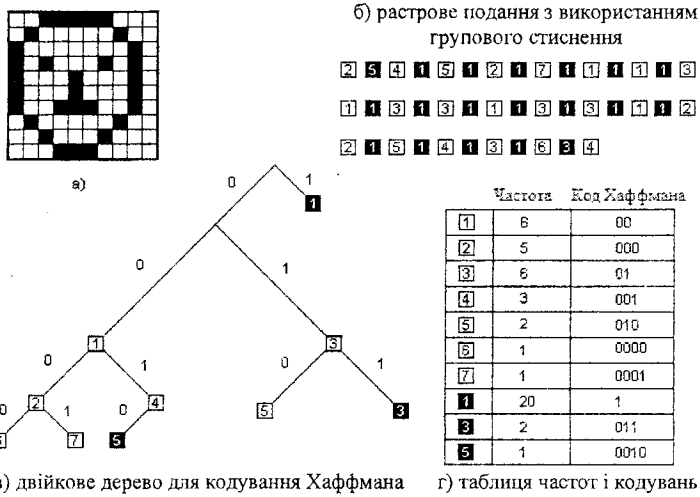
Для того, щоб відновити кодуючі слова, потрібно пройти по стрілках від початкових символів до кінця одержаного бінарного дерева. Так, для символу з імовірністю p_4 $B_4 = 101$, для p_3 $B_3 = 100$, для p_2 $B_2 = 11$, для p_1 $B_1 = 0$. Схему можна подати у вигляді:

$$a_1 - 0,$$
$$a_2 - 11,$$
$$a_3 - 100,$$
$$a_4 - 101.$$

Ця схема являє собою префіксний код, що є кодом Хаффмана. Символ a_1, що найчастіше зустрічається в потоці, кодується найкоротшим словом 0, а символ a_4, що зустрічається найрідше – довгим словом 101.

Для послідовності зі 100 символів, в якій символ a_1 зустрічається 50 разів, символ a_2 – 24 рази, символ a_3 – 15 разів, а символ a_4 – 11 разів, даний код дозволить одержати послідовність зі 176 біт ($100 \cdot l_{cp} = \sum_{i=1}^{4} p_i l_i$). Тобто, у середньому було витрачено 1,76 біта на символ потоку.

Класичний алгоритм Хаффмана потребує запису у файл таблиці відповідності кодування і кодувальних ланцюжків. На рис. 3.2 показано застосування кодування Хаффмана до даних після групового стиснення.



	Частота	Код Хаффмана
1	6	00
2	5	000
3	6	01
4	3	001
5	2	010
6	1	0000
7	1	0001
1	20	1
3	2	011
5	1	0010

в) двійкове дерево для кодування Хаффмана г) таблиця частот і кодувань

Рисунок 3.2 – Кодування Хаффмана

Ефективність стиснення за схемою Хаффмана змінюється з точністю алгоритму і типом зображення. Схема Хаффмана працює не так добре для

файлів, що містять довгі послідовності повторюваних величин пікселів, які можуть бути краще стиснуті з використанням групової або іншої схеми стиснення.

Коефіцієнти компресії: 8, 1.5, 1 (кращий, середній, гірший коефіцієнти).

Клас зображень: практично не застосовується до зображень у чистому вигляді. Зазвичай використовується як один з етапів компресії в більш складних схемах.

Характерні риси: єдиний алгоритм, що у найгіршому випадку не збільшує розміру вихідних даних (якщо не вважати необхідності зберігати таблицю перекодування разом з файлом).

Lossless JPEG

Цей алгоритм був розроблений групою експертів в області фотографії (Joint Photographic Expert Group). Lossless JPEG орієнтований на повнокольорові 24-бітові або 8-бітові в градаціях сірого зображення без палітри. Він являє собою спеціальну реалізацію JPEG без втрат.

Коефіцієнти стиснення: 20, 2, 1.

Клас зображень: Lossless JPEG рекомендується застосовувати в тих додатках, де необхідна побітова відповідність вихідного і декомпресованого зображень.

3.6.2 Алгоритми стиснення з втратами

Одна із проблем комп'ютерної графіки полягає в тому, що не знайдено адекватного критерію оцінювання втрат якості зображення. Оскільки втрата якості відбувається досить часто – при оцифровуванні, при переведенні в обмежену палітру кольорів, при перетворенні в іншу систему кольорів для друку, і, звичайно, при стисненні з втратами.

Можна навести приклад простого критерію: середньоквадратичне відхилення значень пікселів (root mean square – RMS):

$$d(x,y) = \sqrt{\frac{\sum_{i=1,j=1}^{n,n} \left(x_{ij} - y_{ij}\right)^2}{n^2}}.$$

Відповідно до цього критерію зображення буде сильно зіпсовано при зниженні яскравості всього на 5% (око цього не помітить – у різних моніторів настроювання яскравості варіюється набагато сильніше). У той же час зображення зі «снігом» – різкою зміною кольору окремих точок, слабкими смугами будуть визнані такими, що майже не змінилися.

Іншим критерієм є максимальне відхилення:

$$d(x,y) = \max_{i,j} \left| x_{ij} - y_{ij} \right|.$$

Ця міра украй чутлива до різкої зміни значень окремих пікселів. Тобто, у всьому зображенні може істотно змінитися тільки значення одного пікселя (що практично непомітно для ока), однак відповідно до цієї міри, зображення буде сильно зіпсовано.

Критерій, що найчастіше використовують на практиці, називається мірою відношення сигналу до шуму (signal-to-noise ratio – SNR):

$$d(x,y) = 10 \cdot \log_{10} \frac{255^2 \cdot n^2}{\sum_{i=1,j=1}^{n,n} \left(x_{ij} - y_{ij} \right)^2}.$$

Дана міра, по суті, аналогічна середньоквадратичному відхиленню, проте користуватися нею зручніше за рахунок логарифмічного масштабу шкали. Їй властиві ті ж недоліки, що й середньоквадратичному відхиленню.

Найкраще втрати якості зображень оцінюють наші очі. Відмінною вважається архівація, при якій неможливо на око розрізнити первісне й разархівоване зображення. При подальшому збільшенні ступеня стиснення, як правило, стають помітними побічні ефекти, характерні для даного алгоритму. На практиці навіть при відмінному збереженні якості у зображення можуть бути внесені регулярні специфічні зміни. Тому алгоритми архівації із втратами не рекомендується використовувати при стисненні зображень, які надалі потрібно друкувати з високою якістю, або обробляти програмами розпізнавання образів. Небажані ефекти з такими зображеннями можуть виникнути навіть при простому масштабуванні зображення.

Алгоритм JPEG

JPEG – один із найновіших і досить потужних алгоритмів. Алгоритм оперує областями розміром 8×8, на яких яскравість і колір змінюються порівняно плавно. Внаслідок цього при розкладанні матриці такої області в подвійний ряд за косинусами значимими виявляються тільки перші коефіцієнти. Таким чином, стиснення в JPEG здійснюється за рахунок плавності зміни кольорів у зображенні. У цілому алгоритм оснований на дискретному косинусоїдальному перетворенні (ДКП), що застосовується до матриці зображення для отримання деякої нової матриці коефіцієнтів. Для одержання вихідного зображення застосовується обернене перетворення.

ДКП розкладає зображення за амплітудами деяких частот, таким чином, при перетворенні ми одержуємо матрицю, у якій багато

коефіцієнтів, що близькі або дорівнюють нулю. Крім того, людська система колірного сприйняття слабо розпізнає певні частоти. Тому можна апроксимувати деякі коефіцієнти більш грубо, без помітної втрати якості зображення.

Для цього використовується квантування коефіцієнтів (quantization). У найпростішому випадку – це арифметичний побітовий зсув вправо. При цьому перетворенні втрачається частина інформації, але можуть досягатися більші коефіцієнти стиснення.

Розглянемо алгоритм детальніше. Нехай потрібно стиснути 24-бітове зображення.

Крок 1. Потрібно перевести зображення з колірного простору RGB у колірний простір YCrCb (іноді називають YUV). У ньому Y – складова яскравості, а Cr, Cb – компоненти, що відповідають за колір (хроматичний червоний і хроматичний синій). За рахунок того, що людське око менш чутливе до кольору, ніж до яскравості, з'являється можливість архівувати масиви для Cr і Cb компонент із більшими втратами й, відповідно, більшими коефіцієнтами стиснення. Спрощено перехід з RGB у YCrCb можна подати так:

$$\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} = \begin{vmatrix} 0.299 & 0.587 & 0.114 \\ 0.5 & -0.4187 & -0.0813 \\ 0.1687 & -0.3313 & 0.5 \end{vmatrix} \times \begin{vmatrix} R \\ G \\ B \end{vmatrix} + \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix}.$$

Обернене перетворення здійснюється множенням вектора YUV на обернену матрицю.

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{vmatrix} \times \left(\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} - \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \right).$$

Крок 2. Потрібно розбити вихідне зображення на матриці 8×8. При цьому сформуються три робочі матриці ДКП – по 8 біт окремо для кожного компонента. При більших коефіцієнтах стиснення цей крок можна виконувати складніше. Зображення ділиться за компонентами Y – як і в першому випадку, а для компонентів Cr і Cb матриці набираються через рядок і через стовпець. Тобто, з вихідної матриці розміром 16×16 виходить тільки одна робоча матриця ДКП. При цьому втрачається 3/4 корисної інформації про колірні складові зображення й отримуємо відразу стиснення у два рази.

Крок 3. Потрібно застосувати ДКП до кожної робочої матриці. При цьому отримується матриця, у якій коефіцієнти в лівому верхньому куті відповідають низькочастотній складовій зображення, а в правому

нижньому – високочастотній. У спрощеному вигляді це перетворення можна подати так:

$$Y[u,v] = \frac{1}{4}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} C(i,u)\times C(j,v)\times \quad y[i,j],$$

де $C(i,u) = A(u)\times \quad \cos\left(\frac{(2\times i+1)\times u\times \pi}{2\cdot n}\right)$, $A(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for } u \equiv 0 \\ 1, & \text{for } u \neq 0 \end{cases}$.

Крок 4. Потрбіно виконати квантування. Для кожного компонента (Y, U і V), у загальному випадку, задається своя матриця квантування q[u,v] (МК):

$$Yq[u,v] = IntegerRound \quad \left(\frac{Y[u,v]}{q[u,v]}\right).$$

На цьому кроці здійснюється керування ступенем стиснення, і відбуваються найбільші втрати. Зрозуміло, що, задаючи МК із більшими коефіцієнтами, отримуємо більше нулів і, отже, більший ступінь стиснення.

Крок 5. Потрібно перевести матрицю 8×8 в 64-елементний вектор за допомогою «зигзаг»-сканування, тобто беремо елементи з індексами (0,0), (0,1), (1,0), (2,0) ... .

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{3,0}$			
$a_{3,0}$	$a_{3,0}$	$a_{3,0}$	$a_{3,0}$				
$a_{4,0}$	$a_{4,1}$	$a_{4,2}$					
$a_{5,0}$	$a_{5,1}$						
$a_{6,0}$	$a_{6,1}$						
$a_{7,0}$	$a_{7,1}$						

Таким чином, на початку вектора одержуємо коефіцієнти матриці, що відповідають низьким частотам, а наприкінці – високим.

Крок 6. Потрібно згорнути вектор за допомогою алгоритму групового кодування. При цьому одержуємо пари типу («пропустити», «число»), де

«пропустити» є лічильником нулів, що пропускаються, а «число» – значення, яке необхідно поставити в наступну комірку. Так, вектор 42 3 0 0 0 -2 0 0 0 0 1 ..., буде згорнутий у пари (0,42) (0,3) (3,-2) (4,1) ... .

Крок 7. Потрібно згорнути отримані пари кодуванням за Хаффманом з фіксованою таблицею.

Процес відновлення зображення в цьому алгоритмі повністю симетричний. Метод дозволяє стискати деякі зображення в 10 – 15 разів без серйозних втрат.

Коефіцієнти компресії: 2 – 200 (задаються користувачем).

Клас зображень: повнокольрові 24 бітові зображення, або зображення в градаціях сірого без різких переходів кольорів (фотографії).

Характерні риси: у деяких випадках, алгоритм створює «ореол» навколо різких горизонтальних і вертикальних границь у зображенні (ефект Гіббса). Крім того, при високому ступені стиснення зображення розпадається на блоки розміром 8×8 пікселів.

Фрактальний алгоритм

Фрактальна архівація основана на тому, що зображення подається в більш компактній формі – за допомогою коефіцієнтів системи ітерованих функцій (Iterated Function System(IFS)).

IFS являє собою набір тривимірних афінних перетворень, у даному випадку таких, що переводять одне зображення в інше. Перетворенню піддаються точки в тривимірному просторі (x_координата, y_координата, яскравість).

Найбільш відомими є два зображення, отримані за допомогою IFS: «трикутник Серпінського» і «папороть Барнслі» (рис. 3.3). «Трикутник Серпінського» задається трьома, а «папороть Барнслі» чотирма афінними перетвореннями. Кожне перетворення кодується буквально ліченими байтами, у той час, як зображення, побудоване з їхньою допомогою, може займати й декілька мегабайт.

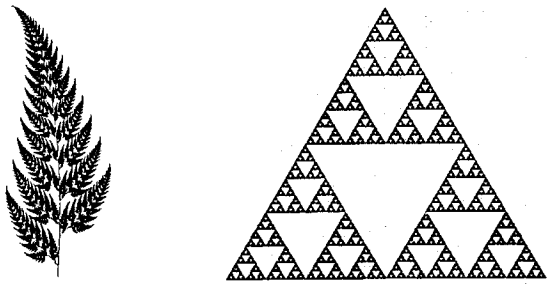


Рисунок 3.3 – «Папороть Барнслі» і «трикутник Серпінського»

Фактично фрактальна компресія – це пошук собіподібних областей у

зображенні і визначення для них параметрів афінних перетворень. Уперше можливість застосування теорії IFS до проблеми стиснення зображення була досліджена Майклом Барнслі. Джеквін подав метод фрактального кодування, у якому використовуються системи доменних і рангових блоків зображення (domain and range subimage blocks), блоків квадратної форми, що покривають все зображення. Цей підхід став основою для більшості методів фрактального кодування. Відповідно до методу зображення розбивається на множину неперекриваних рангових підзображень, також визначається множина перекриваних доменних підзображень. Для кожного рангового блока алгоритм кодування знаходить найбільш підходящий доменний блок і афінне перетворення, що переводить цей доменний блок у даний ранговий блок. Структура зображення відображається в систему рангових блоків, доменних блоків і перетворень.

Фрактальний алгоритм дозволяє стискати зображення в сотні і навіть тисячі разів. Основна складність фрактального стиснення полягає в тому, що для знаходження відповідних доменних блоків потрібен повний перебір.

Оскільки при цьому переборі щоразу повинні порівнюватися два масиви, дана операція виходить досить тривалою і потребує значних обчислювальних ресурсів.

Декомпресія вихідного зображення досить проста. Для цього необхідно виконати декілька ітерацій тривимірних афінних перетворень, коефіцієнти яких були отримані на етапі компресії.

Переважна більшість досліджень в області фрактального стиснення зараз спрямована на зменшення часу архівації, необхідного для одержання якісного зображення. На даний момент відома досить велика кількість алгоритмів оптимізації перебору, що виникає при фрактальному стисненні. Для збільшення швидкості кодування велася робота із двох напрямків. Перший підхід розв'язував завдання класифікації доменів (classification of domains), при якому за рахунок зменшення кількості доменів, серед яких ведеться пошук, скорочується кількість обчислень. При застосуванні другого підходу, який оснований на методі виділення особливостей (feature extraction), збільшення швидкості кодування відбувається за рахунок порівняння доменних і рангових блоків.

На сьогоднішній день фрактальні методи найкраще пристосовані для додатків архівування, таких як цифрові енциклопедії, у яких кодування необхідне лише один раз, а декодування відбувається безліч разів.

Коефіцієнти компресії: 2–2000.

Клас зображень: повнокольрові 24 бітові зображення або зображення в градаціях сірого.

Характерні риси: подібність між елементами зображення.

Рекурсивний (хвильовий) алгоритм

Англійська назва рекурсивного стиснення – wavelet. Алгоритм

орієнтовано на кольорові й чорно-білі зображення із плавними переходами. Ідеальний для картинок типу рентгенівських знімків. Коефіцієнт стиснення задається й варіюється в межах 5–100 разів. При спробі задати більший коефіцієнт, на різких границях, що особливо проходять по діагоналі, проявляється «ефект сходинок» – сходи різної яскравості розміром у декілька пікселів.

Ідея алгоритму полягає в тому, що у файлі зберігається різниця – число між середніми значеннями сусідніх блоків у зображенні, що зазвичай приймає значення, близькі до 0.

Два числа a_{2i} і a_{2i+1} завжди можна подати у вигляді $b^1_i = (a_{2i} + a_{2i+1})/2$ і $b^2_i = (a_{2i} - a_{2i+1})/2$. Аналогічно послідовність a_i може бути попарно переведена в послідовність $b^{1,2}_i$.

Приклад: нехай потрібно стиснути рядок з 8 значень яскравості пікселів (a_i): (220, 211, 212, 218, 217, 214, 210, 202). Тоді можна отримати такі послідовності b^1_i, і b^2_i: (215,5; 215; 215,5; 206) і (4,5; -3; 0,5; 4). Помітно, що значення b^2_i досить близькі до 0. Далі потрібно повторити операцію, розглядаючи b^1_i як a_i. Дана дія виконується рекурсивно, звідки й походить назва алгоритму. В результаті з (215,5; 215; 215,5; 206) отримується: (215,25; 210,75) (0,25; 4,75). Розраховані коефіцієнти, округливши до цілих і стиснувши, наприклад, за допомогою алгоритму Хаффмана з фіксованими таблицями, потрібно помістити у файл. При цьому перетворення до ланцюжка було використано тільки два рази. Отже, можна дозволити застосування wavelet-перетворення до 4 – 6 разів. Більше того, додаткове стиснення можна одержати, використовуючи таблиці алгоритму Хаффмана з нерівномірним кроком (тобто, доведеться зберігати код Хаффмана для найближчого в таблиці значення). Ці прийоми дозволяють досягти помітних коефіцієнтів стиснення.

Алгоритм для двовимірних даних реалізується аналогічно. Якщо є квадрат з 4 точок з яскравостями $a_{2i,2j}$, $a_{2i+1,\,2j}$, $a_{2i,\,2j+1}$, і $a_{2i+1,\,2j+1}$, то

$$b^1_{i,j} = (a_{2i,2j} + a_{2i+1,2j} + a_{2i,2j+1} + a_{2i+1,2j+1})/4,$$

$$b^2_{i,j} = (a_{2i,2j} + a_{2i+1,2j} - a_{2i,2j+1} - a_{2i+1,2j+1})/4,$$

$$b^3_{i,j} = (a_{2i,2j} - a_{2i+1,2j} + a_{2i,2j+1} - a_{2i+1,2j+1})/4,$$

$$b^4_{i,j} = (a_{2i,2j} - a_{2i+1,2j} - a_{2i,2j+1} + a_{2i+1,2j+1})/4.$$

До переваг цього алгоритму можна віднести те, що він дуже легко дозволяє реалізувати можливість поступового прояву зображення при передачі зображення по мережі. На відміну від JPEG і фрактального алгоритму даний метод не оперує блоками, наприклад, 8×8 пікселів. Точніше використовуються блоки розміром 2×2, 4×4, 8×8 і т. д. Однак за рахунок того, що коефіцієнти для цих блоків зберігаються незалежно,

можна досить легко уникнути дроблення зображення на «мозаїчні» квадрати.

Коефіцієнти компресії: 5–100.

Клас зображень: кольорові й чорно-білі зображення із плавними переходами.

Характерні риси: плавні переходи кольорів і відсутність різких границь.

Контрольні питання

1. У чому різниця між алгоритмами з втратою інформації і без втрати інформації?
2. Наведіть приклади критеріїв оцінювання втрат інформації і опишіть їхні недоліки.
3. На який клас зображень орієнтований алгоритм RLE?
4. За рахунок чого стискає зображення алгоритм JPEG?
5. У чому полягає ідея фрактального алгоритму компресії?
6. У чому полягає ідея рекурсивного (хвильового) стиснення?
7. Порівняйте наведені в цьому розділі алгоритми стиснення зображень.

РОЗДІЛ 4 РАСТРОВА ГРАФІКА. БАЗОВІ АЛГОРИТМИ РАСТРОВОЇ ГРАФІКИ

4.1 Растрове подання зображень

Цифрове зображення – набір точок (пікселів) зображення; яке характеризується координатами x і y і яскравістю $V(x,y)$, це зазвичай цілі дискретні величини. У випадку кольорового зображення кожний піксель характеризується координатами x і y, і трьома рівнями яскравості: яскравістю червоного, синього і яскравістю зеленого (V_R, V_B, V_G). Комбінуючи ці три кольори, можна одержати велику кількість різних відтінків.

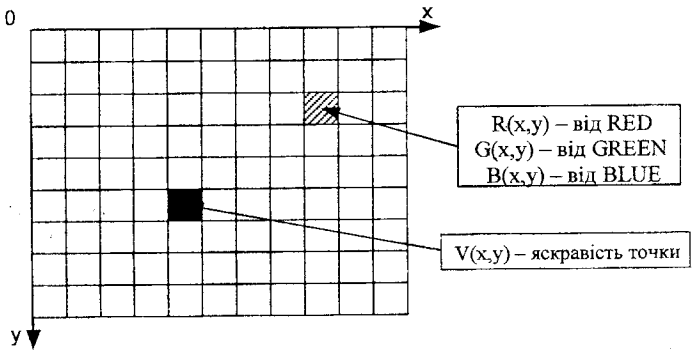


Рисунок 4.1 – Подання цифрового зображення

Під градацію яскравості зазвичай відводиться 1 байт, причому 0 – чорний колір, а 255 – білий (максимальна інтенсивність). У випадку кольорового зображення відводиться по байту на градації яскравості всіх трьох кольорів. Можливе кодування градацій яскравості іншою кількістю бітів (4 або 12), але людське око здатне розрізняти лише 8 біт градацій на кожний колір, хоча спеціальна апаратура може потребувати більш точної передачі кольорів.

Колірний простір, утворений інтенсивностями червоного, зеленого і синього кольорів, подають у вигляді колірного куба.

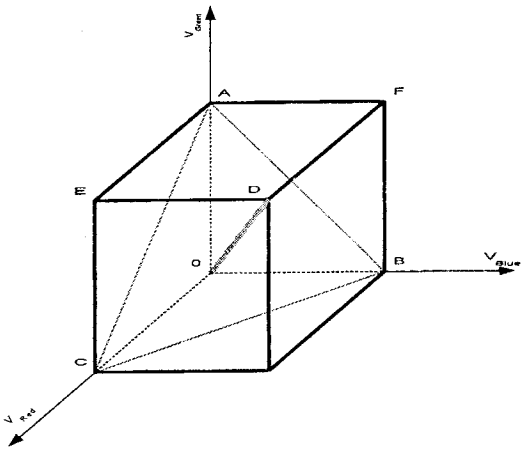


Рисунок 4.2 – Колірний куб

Вершини куба A, B, C є максимальними інтенсивностями зеленого, синього і червоного, відповідно, а трикутник, який вони утворюють, називається трикутником Паскаля, периметр цього трикутника відповідає максимально насиченим кольорам. На відрізку OD знаходяться відтінки сірого, причому точка O відповідає чорному, а точка D – білому кольору.

Растр – це порядок розташування точок (растрових елементів). На рис. 4.1 зображений растр, елементами якого є квадрати, такий растр називається квадратним, саме такі растри найчастіше використовуються. Хоча як растровий елемент можливе використання фігури іншої форми, що відповідає таким вимогам:

1. Всі фігури повинні бути однакові;
2. Повинні повністю покривати площину (без просвітів).

Як растровий елемент можливе використання рівностороннього трикутника, правильного шестикутника (гексаедра), рис. 4.3. Можна будувати растри, використовуючи неправильні багатокутники, але практичний зміст у подібних растрах відсутній.

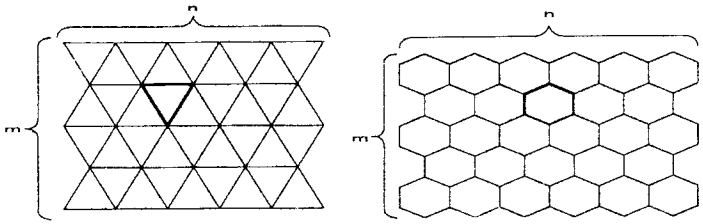


Рисунок 4.3 – Трикутний і гексагональний растри

4.2 Геометричні характеристики растра

Для растрових зображень, які складаються з точок, особливо важливим є поняття **роздільної здатності**, яка виражає кількість точок, що припадає на одиницю довжини. При цьому слід розрізняти:

– роздільну здатність оригіналу;

– роздільну здатність екранного зображення;

– роздільну здатність надрукованого зображення.

Роздільна здатність оригіналу вимірюється в точках на дюйм (dots per inch – dpi) і залежить від вимог до якості зображення та розміру файлу, способу оцифровування і створення початкової ілюстрації, вибраного формату файлу і залежить від інших параметрів. У загальному випадку діє правило: чим вище вимога до якості, тим вищою повинна бути роздільна здатність оригіналу і мащтабу відображення.

Роздільна здатність екранного зображення. Для екранних копій зображення елементарну точку растра прийнято називати пікселем. Розмір пікселя коливається в залежності від вибраної екранної роздільної здатності (з діапазону стандартних значень), роздільної здатності оригіналу і масштабу відображення.

Монітори для обробки зображень з діагоналлю 20–21 дюйм, як правило, забезпечують стандартну екранну роздільну здатність 640×480, 800×600, 1024×768, 1280×1024, 1600×1200, 1600×1280, 1920×1200, 1920×1600 точок. Відстань між сусідніми точками люмінофора у якісного монітора складає 0,22 – 0,25 мм.

Для екранної копії достатньо роздільної здатності 72 dpi, для роздрукованого варіанта на кольоровому або лазерному принтері 150–200 dpi, для виведення на пристрої для фото – 200–300 dpi.

Розмір растра (рис. 4.4) вимірюється кількістю пікселів по горизонталі та вертикалі. Для комп'ютерної графіки найбільш зручний растр із однаковим кроком для обох осей, тобто dpiX = dpiУ.

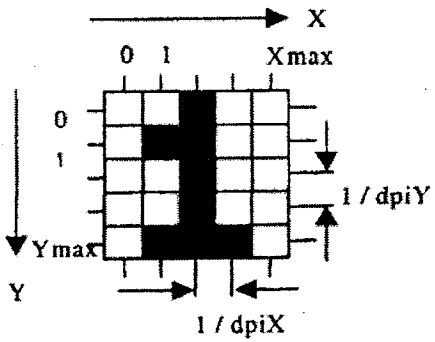


Рисунок 4.4 – Зображення растра

Фізичний розмір зображення може вимірюватися як у пікселях, так і в одиницях довжини (міліметрах, сантиметрах, дюймах). Він задається при створенні зображення й зберігається разом з файлом.

Якщо зображення готують для демонстрації на екрані, то його ширину й висоту задають у пікселях, щоб знати, яку частину екрана воно займає. Якщо зображення готують для друку, то його розмір задають в одиницях довжини, щоб знати, яку частину аркуша паперу воно займе. Неважко перерахувати розмір зображення з пікселів в одиниці довжини й навпаки, якщо відома роздільна здатність зображення (див. таблицю 4.1).

Таблиця 4.1 – Зв'язок між розміром зображення (у пікселях) і розміром віддрукованої копії, (у мм) при різних роздільних здатностях

Розмір зображення в пікселях	Розмір віддрукованої копії, мм при роздільних здатностях			
	75 dpi	150 dpi	300 dpi	600 dpi
640×480	212×163	108×81	55×40	28×20
800×600	271×203	136×102	68×51	34×26
1024×768	344×260	173×130	88×66	44×33
1152×864	390×292	195×146	98×73	49×37
1600×1200	542×406	271×203	136×102	68×51

Інтенсивність тону прийнято поділяти на 256 рівнів. Більше число градацій не сприймається зором людини і є надлишковим. Менше число погіршує сприйняття зображення (мінімально припустимим для якісної півтонової ілюстрації прийняте значення 150 рівнів). Неважко підрахувати, що для відтворення 256 рівнів тону досить мати розмір комірки растра 256 = 16×16 точок.

Відстань між центрами растрових комірок однакова, їхнє число на одиницю довжини називається *лініатурою растра* і виміряється в лініях на дюйм (lpi – lines per inch). Чим вище значення lpi растра, тим більш чітким виглядає зображення, тому що дрібні деталі попадають у декілька комірок растра. Сучасне якісне поліграфічне устаткування може мати лініатуру растра до 300 lpi. При друці на принтері лініатура растра становить порядка 65–90 lpi. Для отримання якісного зображення треба знати залежність між лініатурою, роздільною здатністю і тоновим діапазоном:

$$Число\ \ рівнів = \left(\frac{роздільна\cdot здатність,\ dpi}{лініатура,\ lpi} \right)^2 + 1.$$

Динамічний діапазон. Якість відтворення тонових зображень прийнято оцінювати динамічним діапазоном (D). Це оптична щільність, яка чисельно рівна десятковому логарифму величини, оберненої коефіцієнту

пропускання (для оригіналів, що розглядаються «на просвіт», наприклад, слайдів) або коефіцієнта відбиття (для інших оригіналів, наприклад поліграфічних відбитків). Для оптичних середовищ, що пропускають світло, динамічний діапазон лежить у межах від 0 до 4. Для поверхонь, що відбивають світло, значення динамічного діапазону становить від 0 до 2. Чим вищий динамічний діапазон, тим більше число півтонів присутнє у зображенні й тим краща якість його сприйняття.

Існує також таке поняття як коефіцієнт прямокутності пікселів. На відміну від коефіцієнта прямокутності зображення, він відноситься до реальних розмірів відеопікселя і є відношенням реальної ширини до реальної висоти. Даний коефіцієнт залежить від розміру дисплея і поточної роздільної здатності, і тому на різних комп'ютерних системах приймає різні значення. Колір будь-якого пікселя растрового зображення запам'ятовується в комп'ютері за допомогою комбінації бітів. Чим більше бітів для цього використовується, тим більше відтінків кольорів можна одержати. Число бітів, що використовуються комп'ютером для будь-якого пікселя, називається *бітовою глибиною* пікселя.

Найбільш просте растрове зображення складається з пікселів, що мають тільки два можливих кольори – чорний та білий, і тому зображення, що складаються з пікселів цього виду, називаються однобітовими зображеннями. Число доступних кольорів або градацій сірого кольору дорівнює 2 у степені, рівному кількості бітів у пікселі. Кольори, описані 24 бітами, забезпечують більше 16 мільйонів доступних кольорів і їх часто називають природними кольорами.

Кількість кольорів (глибина кольору) – також одна з найважливіших характеристик растра. За кількістю кольорів зображення класифікується таким способом:

– двоколірні (бінарні) – 1 біт на піксель. Серед двоколірних найчастіше зустрічаються чорно-білі зображення.

– півтонові – градації сірого або іншого кольору. Наприклад 256 градацій (1 байт на піксель).

– кольорові зображення. Від 2 біт на піксель і вище. Глибина кольору 16 біт на піксель (65 536 кольорів) одержала назву **High Color**, 24 біти на піксель (16,7 млн кольорів) – **True Color**. У комп'ютерних графічних системах використовують і більшу глибину кольору – 32, 48 і більше біт на піксель.

4.3 Побудова лінії у квадратному растрі

Оскільки екран растрового дисплея можна розглядати як матрицю дискретних елементів (пікселів), кожний з яких може бути підсвічений, не можна безпосередньо провести відрізок з однієї точки в іншу. Процес визначення пікселів, що найкраще апроксимують заданий відрізок,

називається розкладанням у растр. У поєднанні із процесом порядкової візуалізації зображення він відомий як перетворення растрової розгортки. Для горизонтальних, вертикальних і нахилених під кутом 45° відрізків вибір растрових елементів очевидний. При будь-якій іншій орієнтації вибрати потрібні пікселі важче, що показано рис. 4.5.

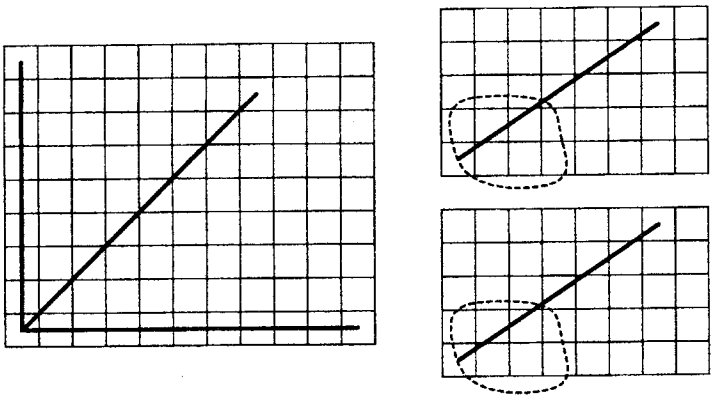


Рисунок 4.5 – Розкладання відрізків в растр

4.3.1 Параметричний алгоритм рисування лінії

Необхідно провести лінію із точки (x_1, y_1) у точку (x_2, y_2) з лінійною інтерполяцією за яскравістю (рис. 4.6).

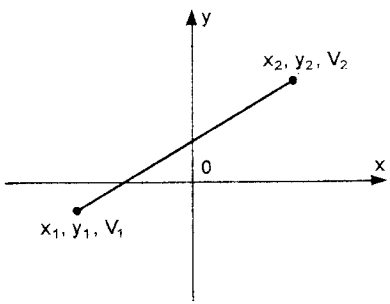


Рисунок 4.6 – Параметричний алгоритм рисування лінії

Будь-яку точку на цій лінії можна подати у вигляді:

$$\begin{aligned} x_{i+1} &=]x_i + t \cdot (x_2 - x_1)[\\ y_{i+1} &=]y_i + t \cdot (y_2 - y_1)[, \\ V_{i+1} &=]V_i + t \cdot (V_2 - V_1)[\end{aligned} \qquad (4.1)$$

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

$$t = \frac{1}{\max\{|x_2 - x_1|, |y_2 - y_1|\}} = \frac{1}{N-1}, \qquad (4.2)$$

де N – довжина лінії в пікселях.

Можна проводити обчислення через приріст координат.

$$\Delta x = \frac{x_2 - x_1}{N-1}; \Delta y = \frac{y_2 - y_1}{N-1}; \Delta V = \frac{V_2 - V_1}{N-1}. \qquad (4.3)$$

Значення приростів підраховуються на початку функції і не входять у цикл побудови лінії на екрані, за рахунок чого підвищується швидкодія.

Недоліки алгоритму:

- необхідність працювати з комплексними числами;
- в алгоритмі є операція ділення, що значно ускладнює апаратну організацію й збільшує час роботи алгоритму.

Переваги алгоритму:

- простота програмної реалізації;
- простота реалізації лінійної інтерполяції за яскравістю.

4.3.2 Цифровий диференціальний аналізатор

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

$$dy/dx = Const \qquad або \qquad \Delta y/\Delta x = (y_2 - y_1)/(x_2 - x_1).$$

Розв'язання подаються у вигляді:

$$y_{i+1} = y_i + \Delta y = y_i + \Delta x(y_2 - y_1)/(x_2 - x_1), \qquad (4.4)$$

де x_1, y_1, x_2, y_2 – кінці відрізка, що розкладається;

y_i – початкове значення для чергового кроку уздовж відрізка. Вираз (4.4) подає рекурентне співвідношення для послідовних значень y уздовж потрібного відрізка. Цей метод називається цифровим диференціальним аналізатором (Digital Differential Analyzer, ЦДА).

Роботу алгоритму можна проілюструвати такими прикладами.

Приклад 4.1. Відрізок прямої йде з точки (0; 0) в точку (7; 5), довжина = 7. $\Delta x = 1$ $\Delta y = 0{,}71$.

x = 0,5; y = 0,5 .

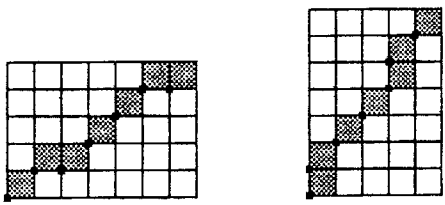
i	*x*	*y*	*Plot*
1	0,5	0,5	(0,0)
2	1,5	1,21	(1,1)
3	2,5	1,93	(2,1)
4	3,5	2,64	(3,2)
5	4.5	3,36	(4,3)
6	5,5	4,07	(5,4)
7	6.5	4,79	(6,4)

Приклад 4.2. Відрізок прямої йде з точки (0; 0) в точку (-5; -7), довжина = 7; $\Delta x = -0{,}71$ $\Delta y = -1$.

x = -0,5; y = -0,5.

i	*x*	*y*	*Plot*
1	-0,5	-0,5	(-1,-1)
2	-1,21	-1,5	(-2,-2)
3	-1,93	-2,5	(-2,-3)
4	-2,64	-3,5	(-3,-4)
5	-3,36	-4.5	(-4,-5)
6	-4,07	-5,5	(-5,-6)
7	-4,79	-6.5	(-5,-7)

Результати обох прикладів показані на рис. 4.7 (а – приклад 1; б – приклад 2).



а) б)

Рисунок 4.7 – Результати прикладів

Нижче наведена підпрограма побудови вектора із точки (x_n,y_n) у точку (x_k, y_k) у першому квадранті методом несиметричного цифрового диференціального аналізатора з використанням тільки цілочислової арифметики.

Побудова ведеться від точки з меншими координатами до точки з більшими координатами, з одиничним кроком по координаті, з більшим

приростом. Окремо виділяється випадок вектора з $d_x == d_y$. Усього треба вивести пікселі в $d_x = x_k - x_n + 1$ позиції по осі X і в $d_y = y_k - y_n + 1$ позиції по осі Y. Для визначеності розглядається випадок dx > dy. При збільшенні X-координати на 1, Y-координата повинна збільшитися на величину, меншу одиниці й рівну dy/dx. Після того, як Y- збільшення стане більше або буде дорівнювати 1,0 то Y-координату пікселя треба збільшити на 1, а з накопиченого збільшення відняти 1,0 і продовжити побудову. Тобто, збільшення Y на 1 виконується за умови: dy/dx + dy/dx + ... + dy/dx > = 1,0.

```
void V_DDA (xn, yn, xk, yk)
int  xn, yn, xk, yk;
{  int  dx, dy, s;
/* Упорядкування координат  і обчислення приросту */
  if (xn > xk) {
    s= xn;  xn= xk;  xk= s;
    s= yn;  yn= yk;  yk= s;
  }
  dx= xk - xn;  dy= yk - yn;
/* Занесення початкової точки вектора */
  PutPixLn (xn, yn, Pix_C);
  if (dx==0 && dy==0) return;
/* Обчислення кількості позицій за X і Y */
  dx= dx + 1;  dy= dy + 1;
/* Генерація вектора */
  if (dy == dx) {              /* Нахил == 45 градусів */
    while (xn < xk) {
      xn= xn + 1;
      PutPixLn (xn, xn, Pix_C);
    }
  } else if (dx > dy) {        /* Нахил <  45 градусів */
    s= 0;
    while (xn < xk) {
      xn= xn + 1;
      s= s + dy;
      if (s >= dx) { s = s - dx;  yn= yn + 1; }
      PutPixLn (xn, yn, Pix_C);
    }
  } else {                     /* Нахил >  45 градусів */
    s= 0;
    while (yn < yk) {
      yn= yn + 1;
      s= s + dx;
      if (s >= dy) { s= s - dy;  xn= xn + 1; }
      PutPixLn (xn, yn, Pix_C);
    }}}  /* V_DDA */
```

4.4 Інкрементні алгоритми

Джек Е. Брезенхем запропонував підхід, що дозволяє розробляти так звані інкрементні алгоритми растеризації. Основною метою при розробленні таких алгоритмів була побудова циклів обчислення координат на основі тільки цілочислових операцій додавання і віднімання без використання множення і ділення. Були розроблені інкрементні алгоритми не тільки для прямих, але і для кривих ліній.

Інкрементні алгоритми виконуються як послідовне обчислення координат сусідніх пікселів шляхом додавання приростів координат. Прирости розраховуються на основі аналізу функції похибки. У циклі виконуються тільки цілочислові операції порівняння і додавання/віднімання. Досягається підвищення швидкодії для обчислень кожного пікселя в порівнянні з прямим способом.

4.4.1 Алгоритм Брезенхема для рисування лінії

В 1965 році Брезенхемом був запропонований простий цілочисловий алгоритм для растрової побудови відрізка. В алгоритмі використовується керуюча змінна d_i, що на кожному кроці пропорційна різниці між s і t (рис. 4.8). На рис. 4.8 наведений i-ий крок, коли піксель P_{i-1} уже знайдений як найближчий до реального зображуваного відрізка, і тепер потрібно визначити, який з пікселів повинен бути встановлений наступним – T_i або S_i.

Якщо $s < t$, то S_i ближче до відрізка і потрібно вибрати його, в іншому випадку ближче буде T_i. Іншими словами, якщо $s\text{-}t < 0$, то вибирається S_i; в іншому випадку – T_i.

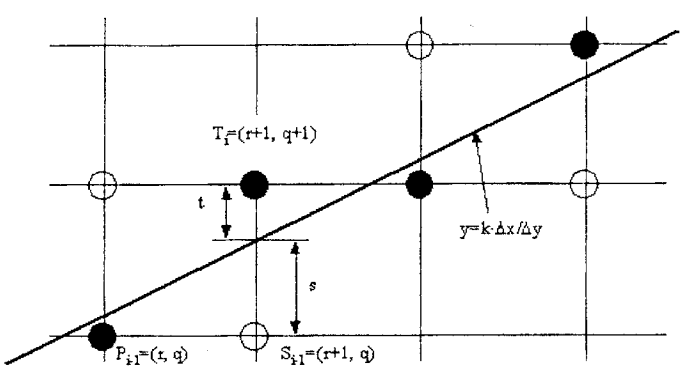


Рисунок 4.8 – Алгоритм Брезенхема для рисування лінії

Зображуваний відрізок проводиться із точки (x_1, y_1) (на рис. 4.8) у точку (x_2, y_2). Нехай перша точка перебуває ближче до початку координат, тоді перенесемо обидві точки $T(x_1, y_1)$ так, щоб початкова точка відрізка

виявилася на початку координат, тоді кінцева виявиться в $(x, \Delta y)$, де $\Delta x = x_2 - x_1$, $\Delta y = y_2 - y_1$. Рівняння прямої тепер має вигляд $y = x \cdot \Delta y / \Delta x$.

З рисунка 4.8 слідує, що

$$s = \frac{\Delta y}{\Delta x}(r+1) - q; \; t = q + 1 - \frac{\Delta y}{\Delta x}(r+1),$$

тому $s - t = 2\frac{\Delta y}{\Delta x}(r+1) - 2q - 1$ помножимо на Δx:

$$\Delta x(s-t) = 2(\Delta y \cdot r - q \cdot \Delta x) + 2\Delta y - \Delta x.$$

Оскільки $\Delta x > 0$, то величину $\Delta x(s-t)$ можна використовувати як критерій для вибору пікселя. Якщо позначити цю величину d_i

$$d_i = 2(\Delta y \cdot x_{i-1} - y_{i-1} \cdot \Delta x) + 2\Delta y - \Delta x,$$

оскільки $r = x_{i-1}$, $q = y_{i-1}$, то можна отримати:

$$\Delta_i = d_{i+1} - d_i = 2\Delta y(x_i - x_{i-1}) - 2\Delta x(y_i - y_{i-1}).$$

Відомо, що $x_i - x_{i-1} = 1$. Якщо $d_i > 0$, то вибираємо T_i, тоді $\Delta_i = 2(\Delta y - \Delta x)$. Якщо $d_i < 0$, то вибираємо S_i, тоді $\Delta_i = 2\Delta y$.

Таким чином, ми одержали ітераційну формулу для обчислення критерію d_i. Початкове значення $d_1 = 2\Delta y - \Delta x$.

Лінійну інтерполяцію за яскравістю при побудові відрізка за алгоритмом Брезенхема можна отримати параметричним способом, тобто

$$V = V_{i-1} + \Delta V; \Delta V = \frac{V_2 - V_1}{N - 1},$$

де N – довжина відрізка в пікселях.

Блок-схема алгоритму Брезенхема для рисування лінії наведена на рис. 4.9.

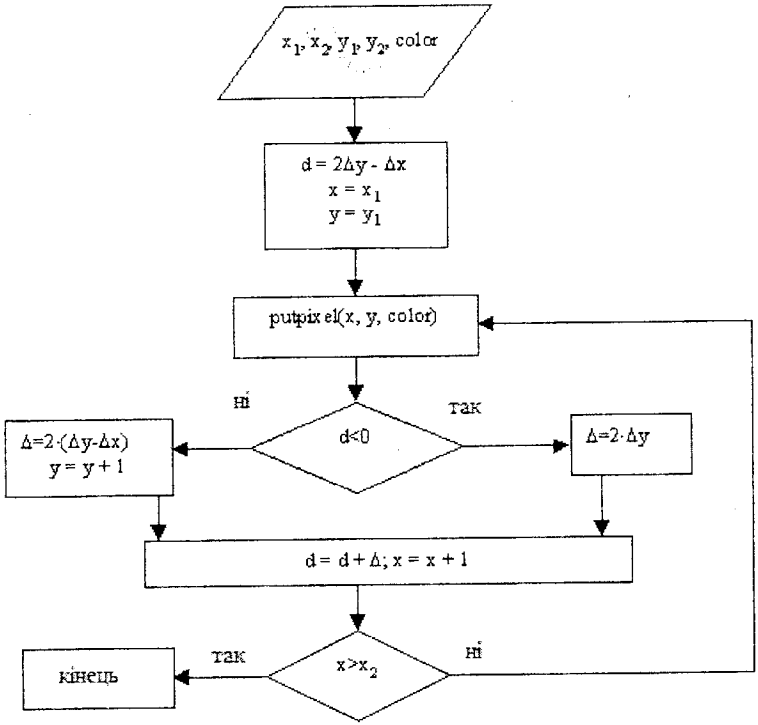
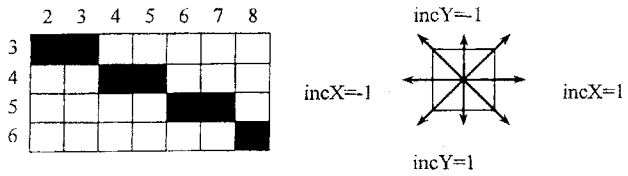


Рисунок 4.9 – Блок-схема алгоритму Брезенхема для рисування лінії

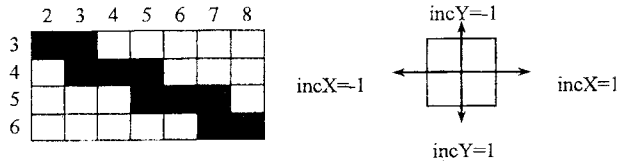
Реалізація алгоритму Брезенхема виконується таким чином:

xerr:=0; yerr:=0;
dx:=x2-x1; dy:=y2-y1;
Якщо dx>0, то incX:=1;
dx=0, то incX:=0;
dx<0, то incX:=-1;
Якщо dy>0, то incY:=1;
dy=0, то incY:=0;
dy<0, то incY:=-1;
dx:=|dx|; dy:=|dy|;
Якщо dx>dy, то d:=dx в іншому випадку d:=dy;
x:=x1; y:=y1;
Зафарбувати піксель з координатами (x, y);
Виконати цикл d раз:
xerr:=xerr+dx;
yerr:=yerr+ dy;
Якщо xerr>=d, то xerr:=xerr-d, x:=x+incX;
Якщо yerr>=d, то yerr:=yerr-d, y:=y+incY;
Зафарбувати піксель з координатами (x, y).

Як приклад можна навести використання алгоритму Брезенхема для побудови відрізка (2;3) – (8;6). Цей алгоритм восьмизв'язний, тобто при обчисленні збільшень координат для переходу до сусіднього пікселя можливі вісім випадків:



Відомі також чотиризв'язні алгоритми. Вони більш прості, але генерують менш якісне зображення ліній за більшу кількість тактів. Для наведеного прикладу чотиризв'язний алгоритм працює 10 тактів, а восьмизв'язний – тільки 7:



Підпрограма, наведена нижче, ілюструє побудову вектора з точки (x_n, y_n) в точку (x_k, y_k) методом Брезенхема. Побудова ведеться від точки з меншими координатами до точки з більшими координатами, з одиничним кроком по координаті, з більшим приростом.

У загальному випадку вихідний вектор проходить не через вершини растрової сітки, а перетинає її сторони. Нехай приріст по X більше приросту по Y і обидва вони більші нуля. Для чергового значення X потрібно вибрати одну з двох найближчих координат сітки по Y. Для цього перевіряється, як проходить вихідний вектор – вище або нижче середини відстані між найближчими значеннями Y. Якщо вище середини, то Y-координату треба збільшити на 1, якщо ні – залишити попередньою. Для цієї перевірки аналізується знак змінної s, що відповідає різниці між дійсним положенням і серединою відстані між найближчими Y-вузлами сітки.

```
void V_Bre (xn, yn, xk, yk)
int  xn, yn, xk, yk;
{  int  dx, dy, s, sx, sy, kl, swap, incr1, incr2;

/* Обчислення приростів і кроків */
  sx= 0;
```

```
  if ((dx= xk-xn) < 0) {dx= -dx; --sx;} else if (dx>0) ++sx;
  sy= 0;
  if ((dy= yk-yn) < 0) {dy= -dy; --sy;} else if (dy>0) ++sy;
/* Врахування нахилу */
  swap= 0;
  if ((kl= dx) < (s= dy)) {
    dx= s;  dy= kl;  kl= s; ++swap;
  }
  s= (incr1= 2*dy)-dx; /* incr1 – константа переобчислення */
                       /* різниці, якщо поточне s < 0  і  */
                       /* s – початкове значення різниці */
  incr2= 2*dx;         /* константа для переобчислення    */
                       /* різниці, якщо поточне s >= 0    */
  PutPixLn (xn,yn,Pix_C); /* Перший піксель вектора  */
  while (--kl >= 0) {
    if (s >= 0) {
      if (swap) xn+= sx; else yn+= sy;
      s-= incr2;
    }
    if (swap) yn+= sy; else xn+= sx;
    s+= incr1;
    PutPixLn (xn,yn,Pix_C); /* Поточна точка вектора   */
  }
} /* V_Bre */
```

4.4.2 Алгоритм Брезенхема для генерації кола

Існує декілька дуже простих, але неефективних способів перетворення кіл в растрову форму. Наприклад, для кола з центром на початку координат, яке описується рівнянням $x^2 + y^2 = R^2$, можна записати такий вираз:

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

Щоб зрисувати четверту частину кола, необхідно змінювати x з одиничним кроком від 0 до R і на кожному кроці обчислювати y. Другим простим методом растрової розгортки кола є використання обчислень x і y за формулами $x = R\cos\alpha$, $y = R\sin\alpha$ при покроковій зміні кута α від 0° до 90°.

Для спрощення алгоритму растрової розгортки стандартного кола можна скористатися його симетрією відносно координатних осей і прямих $y = \pm x$; у випадку, коли центр кола не збігається з початком координат, ці прямі необхідно посунути паралельно так, щоб вони пройшли через центр кола. Тим самим достатньо побудувати растрове зображення для 1/8 частини кола, а решту точок отримати симетрією (рис. 4.10).

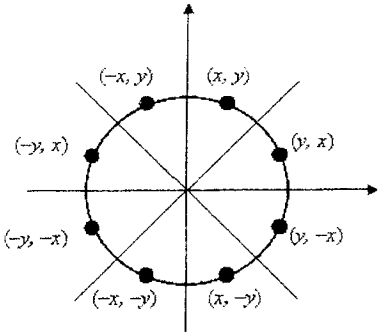


Рисунок 4.10 – Восьмистороння симетрія

Нехай потрібно описати алгоритм Брезенхема для ділянки кола з другого октанта $x \in \left[0, R/\sqrt{2}\right]$. На кожному кроці алгоритм вибирає точку P_i (x_i, y_i), яка є найближчою до дійсного кола. Ідея алгоритму полягає у виборі найближчої точки за допомогою керуючих змінних, значення яких можна обчислити в покроковому режимі з використанням невеликого числа додавань, віднімань і зсувів.

Потім потрібно розглянути невелику ділянку сітки пікселів, а також можливі способи (від т. *A* до т. *E)* проходження дійсного кола через сітку (рис. 4.11). Нехай точка P_{i-1} була вибрана якнайближча до кола при $x = x_{i-1}$. Тепер знайдемо, яка з точок (S_i або T_i) розташована ближче до кола при $x = x_{i-1} + 1$.

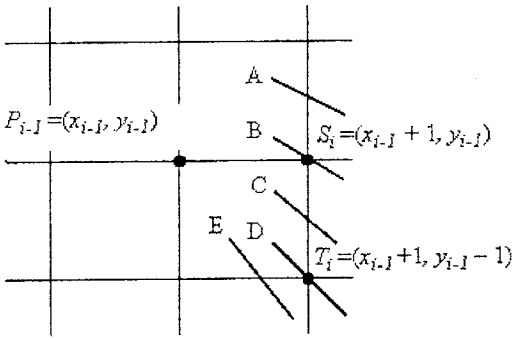


Рисунок 4.11 – Варіанти проходження кола через растрову сітку

Причому похибка при виборі точки P_i (x_i, y_i) дорівнює:

$$D(P_i) = (x_i^2 + y_i^2) - R^2.$$

Тоді вираз для похибок, які отримуємо в результаті вибору точки S_i або T_i, можна записати:

$$D(S_i) = [(x_{i-1} + 1)^2 + (y_{i-1})^2] - R^2;$$
$$D(T_i) = [(x_{i-1} + 1)^2 + (y_{i-1} - 1)^2] - R^2.$$

Якщо $| D(S_i) | \geq | D(T_i) |$, то T_i ближче до реального кола, в іншому випадку – вибираємо S_i.

Нехай $d_i = | D(S_i) | - | D(T_i) |$. Тоді T_i буде вибиратись при $d_i \geq 0$, в іншому випадку буде виставлятись S_i.

Далі потрібно записати d_i і d_{i+1} для різних варіантів вибору точки S_i або T_i.

$$D_1 = 3 - 2R.$$

Якщо вибираємо S_i (коли $d_i < 0$), то $d_{i+1} = d_i + 4x_{i-1} + 6$.

Якщо вибираємо T_i (коли $d_i \geq 0$), то $d_{i+1} = d_i + 4(x_{i-1} - y_{i-1}) + 10$.

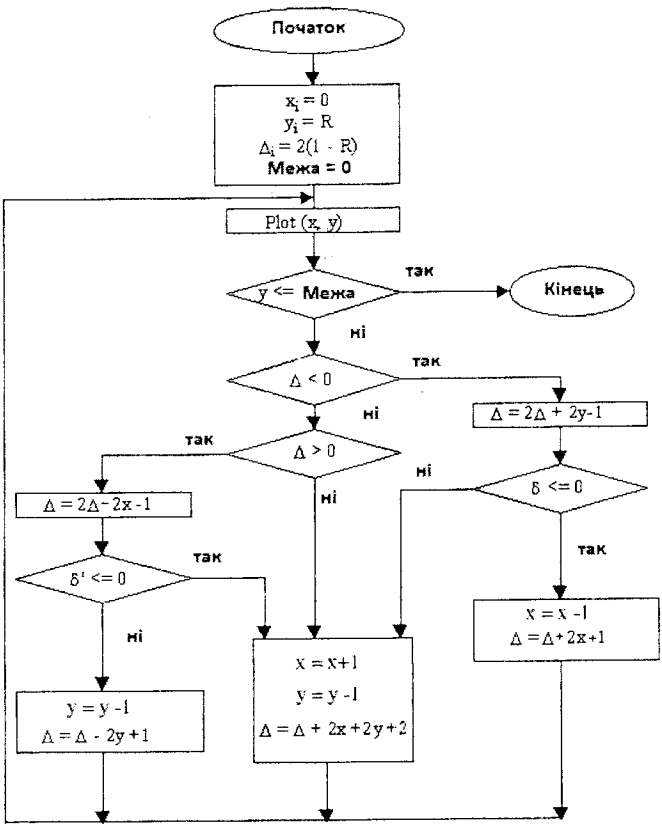


Рисунок 4.12 – Блок-схема алгоритму Брезенхема для побудови растрового подання кола

Запис інкрементного алгоритму Брезенхема для побудови кола мовою С++ можна подати таким чином. Для побудови більш ефективного алгоритму потрібно скористатись симетрією кола. Якщо точка (*x, y*) лежить на колі, то легко обчислити сім точок, що належать колу і є симетричними заданій точці. Процедура *Pixel_circle* заносить пікселі кола за годинниковою стрілкою.

```
static void Pixel_circle (xc, yc, x, y, pixel)
int  xc, yc, x, y, pixel;
{
  putpixel(xc+x, yc+y, pixel);
  putpixel(xc+y, yc+x, pixel);
  putpixel(xc+y, yc-x, pixel);
  putpixel(xc+x, yc-y, pixel);
  putpixel(xc-x, yc-y, pixel);
  putpixel(xc-y, yc-x, pixel);
  putpixel(xc-y, yc+x, pixel);
  putpixel(xc-x, yc+y, pixel);
} /* Pixel_circle */
```

Процедура *V_BRcirc* генерує 1/8 кола за алгоритмом Брезенхема. Процедура може будувати 1/4 кола, для цього цикл *while* потрібно замінити на *for* (;;) і після *Pixel_circle* перевіряти досягнення кінця за умовою *If (y <= end) break*; де *end* встановлюється рівним нулю. У цьому випадку не потрібний і останній оператор *If (x == y) Pixel_circle (xc, yc, x, y, pixel)*. Генерацію 1/8 можна забезпечити, задавши *end = r / sqrt* (2), і заносити пікселі кола за годинниковою стрілкою:

```
void V_BRcirc (xc, yc, r, pixel)
int  xc, yc, r, pixel;
{  int  x, y, z, Dd;
   x= 0;  y= r;  Dd= 2*(1-r);
   while (x < y) {
     Pixel_circle (xc, yc, x, y, pixel);
     if (!Dd) goto Pd;
     z= 2*Dd - 1;
     if (Dd > 0) {
       if (z + 2*x <= 0) goto Pd; else goto Pv;
     }
     if (z + 2*y > 0) goto Pd;
Pg:  ++x;      Dd= Dd + 2*x + 1;   continue; /* Горизонт */
Pd:  ++x; --y; Dd= Dd + 2*(x-y+1); continue; /* Діагональ */
Pv:  --y;      Dd= Dd - 2*y + 1;            /* Вертикаль */
   }
   if (x == y) Pixel_circle (xc, yc, x, y, pixel);
} /* V_BRcirc */
```

4.5 Алгоритми відсікання відрізків. Алгоритм Коена – Сазерленда

Відсікання (clipping) – метод оптимізації в комп'ютерній графіці, коли комп'ютер вирисовує тільки ту частину сцени, що перебуває в полі зору користувача.

У двовимірній графіці, якщо користувач збільшив зображення і на екрані залишилася видна тільки невелика частина, програма може заощадити процесорний час і пам'ять і не прорисовувати ті частини зображення, які залишилися «за кадром».

Аналогічно у тривимірній графіці сцена може складатися з об'єктів (зазвичай трикутників), розташованих з усіх боків віртуальної камери, але програмі досить відображати тільки ті об'єкти, які перебувають у полі зору. У тривимірній графіці для кожного трикутника в сцені потрібно визначити, входить він у поле зору чи ні. Якщо трикутник частково входить у поле зору, то його частину потрібно відсікти.

Існує декілька алгоритмів відсікання.

1. Відсікання відрізків:
   - алгоритм Коена – Сазерленда
   - алгоритм Ляна -- Барски
   - швидке відсікання (FC-алгоритм (Fast Clipping))
   - алгоритм Сайреса – Бека
2. Відсікання багатокутників:
   - алгоритм Сазерленда – Ходгмана
   - алгоритм Уайлера – Атертона

Алгоритм Коена – Сазерленда

Операція «вікно» проводить операцію відсікання точок, ліній, ланцюжків, літер для типового вертикального прямокутного вікна (рис. 4.13). Точка *A* перебуває усередині вікна і тому вона відображається усередині поля виведення на видовій поверхні. Точка *B* перебуває поза вікном і тому вона не зображується. Відрізок прямої *EF* зображується повністю, відрізки *GH* і *IJ* зображуються частково (відсікаються), а *CD* і *KL* не зображуються зовсім. Можна відзначити, що точки і відрізки, які перебувають на границі вікна, вважаються внутрішніми і тому відображаються.

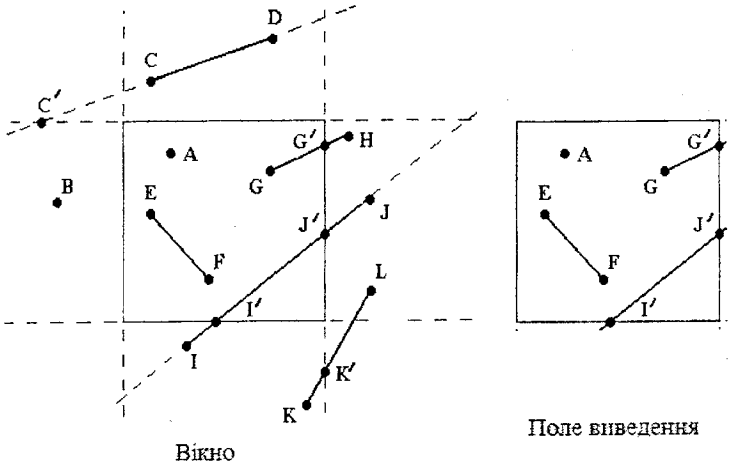


Рисунок 4.13 – Двовимірне відсікання

Відсікання точок здійснюється досить просто. Якщо x_{max} і x_{min}, y_{max} і y_{min} – координати границь вікна, то для того щоб точка була видимою, повинні виконуватися нерівності:

$$x_{min} \le x \le x_{max}, \; y_{min} \le y \le y_{max}.$$

Якщо хоча б одна із цих нерівностей неправильна, точка не відображається.

Алгоритм Коена – Сазерленда здійснює відсікання відрізка відносно прямокутника зі сторонами, паралельними координатним осям.

Якщо пронумерувати сторони прямокутника так, як це показано на рис. 4.14, то всі точки залежно від їхнього положення, можна класифікувати стосовно відсічних прямих 1, 2, 3, 4. Для цього, потрібно зіставити кожну область, на які прямі розбивають площину, з 4-бітовим кодом.

Ці біти визначаються таким чином:

0 біт – якщо точка лежить лівіше прямої 1 ($x < x_{вліво}$);

1 біт – якщо точка лежить нижче прямої 2 ($y < y_{вниз}$);

2 біти – якщо точка лежить правіше прямої 3 ($x > x_{вправо}$);

3 біти – якщо точка лежить вище прямої 4 ($y > y_{уверх}$);

Тоді для відрізка AB на рис. 4.14 будемо мати: код точки A = 0011, код точки B = 0100.

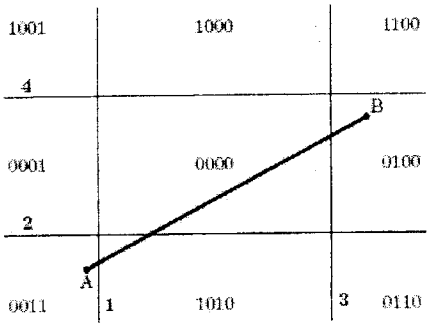


Рисунок 4.14 – Алгоритм Коена – Сазерленда

Визначення того, чи лежить відрізок повністю усередині вікна або повністю поза вікном, виконується таким способом:

1. Якщо коди обох кінців відрізка дорівнюють 0, то відрізок цілком усередині вікна, відсікання не потрібно, відрізок приймається як видимий (відрізок *EF* на рис. 4.13);
2. Якщо логічне & кодів обох кінців відрізка не дорівнює нулю, то відрізок повністю поза вікном, відсікання не потрібно, відрізок відкидається як невидимий (відрізок *CD* на рис. 4.13);
3. Якщо логічне & кодів обох кінців відрізка дорівнює нулю, то відрізок може бути частково видимим (відрізки *IJ* і *GH*) або повністю невидимим (відрізок *KL*); для нього потрібно визначити координати перетину зі сторонами вікна і для кожної отриманої частини визначити видимість або невидимість.

При безпосередньому використанні описаного вище способу відбору повністю видимого або повністю невидимого відрізка після розрахунку перетину треба було б обчислити код розташування точки перетину. Для прикладу розглянемо відрізок *GH*. Точку перетину позначена як *G'*. У силу того, що границя вікна вважається належною вікну, можна просто прийняти тільки частину відрізка *GG'*, що потрапила у вікно. Частина ж відрізка *G'H*, що насправді виявилася поза вікном, потребує подальшого розгляду, тому що логічне І кодів точок *G* і *H* дасть 0, тобто відрізок *GH* не можна просто відкинути. Для рішення цієї проблеми Коен і Сазерленд запропонували заміняти кінцеву точку з ненульовим кодом кінця, на точку, що лежить на стороні вікна, або на її продовженні.

У цілому схема алгоритму Коена – Сазерленда така:

1. Розрахувати коди кінцевих точок відрізка, що відсікається (у циклі повторювати пункти 2–6).
2. Якщо логічне & кодів кінцевих точок не дорівнює 0, то відрізок повністю поза вікном. Він відкидається і відсікання закінчене.

3. Якщо обидва коди дорівнюють 0, то відрізок повністю видимий. Він приймається і відсікання закінчене.
4. Якщо початкова точка усередині вікна, то вона міняється місцями з кінцевою точкою.
5. Аналізується код початкової точки для визначення сторони вікна з якою є перетин і виконується розрахунок перетину. При цьому обчислена точка перетину заміняє початкову точку.
6. Визначення нового коду початкової точки.

Реалізація алгоритму відсікання Коена – Сазерленда з кодуванням кінців відрізка, що відсікається, програмними засобами мови С++ подається такою функцією:

```
int  V_CSclip (float *x0, float *y0, float *x1, float *y1)
```

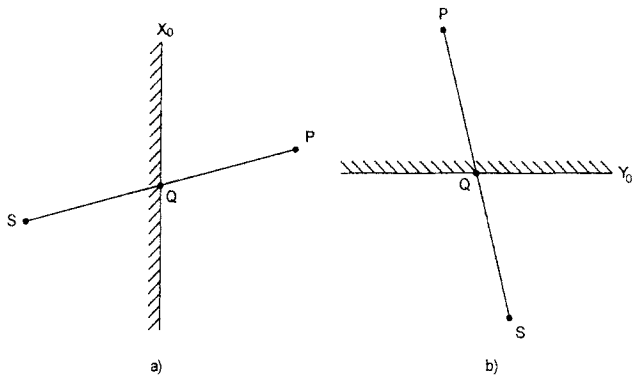
Ця функція відсікає відрізок, заданий значеннями координат його точок (x_0, y_0), (x_1, y_1), по вікну відсікання, заданому глобальними скалярами *Wxlef*, *Wybot*, *Wxrig*, *Wytop*. Кінцевим точкам відрізка приписуються коди, що характеризують його положення щодо вікна відсікання за правилом Коена – Сазерленда.

Допоміжна процедура Code обчислює код положення для кінця відрізка.

```
static float  CSxn, CSyn;   /* Координати початку відрізка */
static int  CScode (void)  /* Визначає код точки xn, yn */
{  register int  i;
   i= 0;
   if (CSxn < Wxlef) ++i; else
   if (CSxn > Wxrig) i+= 2;
   if (CSyn < Wybot) i+= 4; else
   if (CSyn > Wytop) i+= 8;
   return (i);
}  /* CScode */
int  V_CSclip (x0, y0, x1, y1)
float *x0, *y0, *x1, *y1;
{
   float  CSxk, CSyk;   /* Координати кінця відрізка */
   int    cn, ck,       /* Коди кінців відрізка */
          visible,      /* 0/1 -- видимий або невидимий*/
          ii, s;        /* Робочі змінні */
   float  dx, dy,       /* Збільшення координат */
          dxdy,dydx,    /* Нахили відрізка до сторін */
          r;            /* Робоча змінна */
   CSxk= *x1; CSyk= *y1;
   CSxn= *x1; CSyn= *y1; ck= CScode ();
   CSxn= *x0; CSyn= *y0; cn= CScode ();
/* Визначення збільшень координат і нахилів відрізка до осей. Заодно відразу на
побудову передається відрізок, що складається з єдиної точки, що потрапила у вікно*/
```

```
  dx= CSxk - CSxn;
  dy= CSyk - CSyn;
  if (dx != 0) dydx= dy / dx; else {
    if (dy == 0) {
      if (cn==0 && ck==0) goto out; else goto all;
    }
  }
  if (dy != 0) dxdy= dx / dy;
/* Основний цикл відсікання */
  visible= 0;  ii= 4;
  do {
    if (cn & ck) break;      /* Повністю поза вікном */
    if (cn == 0 && ck == 0) { /* Повністю усередині вікна */
      ++visible;  break;
    }
    if (!cn) {               /* Якщо Pn усередині вікна, то */
      s= cn; cn= ck; ck= s;  /* перемістити точки Pn,Pk і */
      r=CSxn; CSxn=CSxk; CSxk=r;  /* їхні коди, щоб Pn */
      r=CSyn; CSyn=CSyk; CSyk=r;  /* виявилася поза вікном */
    }
    /* Тепер відрізок розділяється. Pn поміщається в точку
     * перетинання відрізка зі стороною вікна.      */
    if (cn & 1) {        /* Перетинання з лівою стороною */
      CSyn= CSyn + dydx * (Wxlef-CSxn);
      CSxn= Wxlef;
    } else if (cn & 2) {  /* Перетинання з правою стороною */
      CSyn= CSyn + dydx * (Wxrig-CSxn);
      CSxn= Wxrig;
    } else if (cn & 4) {  /* Перетинання з нижньою стороною */
      CSxn= CSxn + dxdy * (Wybot-CSyn);
      CSyn= Wybot;
    } else if (cn & 8) {  /* Перетинання з верхньою стороною */
      CSxn= CSxn + dxdy * (Wytop-CSyn);
      CSyn= Wytop;
    }
    cn= CScode ();       /* Переобчислення коду точки Pn */
  } while (--ii >= 0);
  if (visible) {
out:  *x0= CSxn;  *y0= CSyn;
    *x1= CSxk;  *y1= CSyk;
  }
all:
  return (visible);
}  /* V_CSclip */
```

На кожному кроці відсікання обчислюються нові координати однієї точки, тоді можна знайти формули для обчислення нових координат.



Формули для розрахунку нових координат:

$$Q(x_0, y_Q, V_Q)$$

$$\text{а) } y_Q = y_S + \frac{x_0 - x_S}{x_p - x_S} \cdot (y_P - y_S),$$

$$V_q = V_S + \frac{x_0 - x_S}{x_p - x_S} \cdot (V_P - V_S),$$

$$Q(x_Q, y_0, V_Q)$$

$$\text{б) } x_Q = x_S + \frac{y_0 - y_S}{y_p - y_S} \cdot (x_P - x_S),$$

$$V_q = V_S + \frac{y_0 - y_S}{y_p - y_S} \cdot (V_P - V_S).$$

Ефективність алгоритмів відсікання набуває особливого значення в анімаційних системах графіки, коли сотні і тисячі відрізків повинні відсікатися з високою швидкістю для плавного переходу від одного кадру до іншого.

4.6 Алгоритми зафарбовування областей

У більшості систем використовується одна із значних переваг растрових пристроїв – можливість заповнення областей екрана.

Існує два різновиди заповнення.

1. Пов'язане як з інтерактивною роботою, так і з програмним синтезом зображення, використовується для заповнення внутрішньої частини багатокутника, заданого координатами його вершин.

2. Пов'язане, в першу чергу, з інтерактивною роботою, використовується для заливки області, яка або обкреслена границею з кодом пікселя, що відрізняється від кодів будь-яких пікселів усередині області, або зафарбована пікселями із заданим кодом.

Можна розглянути область, обмежену набором пікселів заданого кольору, і точку (x, y), що лежить усередині цієї області.

```
void PixelFill(int x, int y, int BorderColor, int color)
{
  int c=getpixel(x,y);
  if((c!=BorderColor) && (c!=color))
  {
        putpixel(x,y,color);
        PixelFill(x-1,y,BorderColor,color);
        PixelFill(x+1,y,BorderColor,color);
        PixelFill(x,y-1,BorderColor,color);
        PixelFill(x,y+1,BorderColor,color);
}}
```

Найпростіший алгоритм, що показаний вище, хоч коректно заповнює навіть найскладніші області, є занадто неефективним, тому що вже для нарисованого пікселя функція викликається ще три рази, і, крім того, потребує занадто великого стека через велику глибину рекурсії.

Тому для рішення завдання зафарбування області перевагу надають алгоритмам, які здатні обробляти відразу цілі групи пікселів.

Одним із найбільш популярних алгоритмів подібного типу є алгоритм, в якому для заданої точки (x, y) визначається й заповнюється максимальний відрізок $[x_l, x_r]$, що містить цю точку і лежить усередині області. Після цього в пошуках ще не заповнених пікселів перевіряються відрізки, що лежать вище й нижче. Якщо такі пікселі знаходяться, то функція рекурсивно викликається для їхньої обробки.

```
//File fill2.cpp
#include <conio.h>
#include <graphics.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>

int BorderColor  = WHITE;
int Color        = GREEN;
int LineFill(int x, int y, int dir, int PrevX1, int PrevXr)
{
  int x1=x;
  int xr=x;
  int c;
  do
        c=getpixel(--x1,y);
```

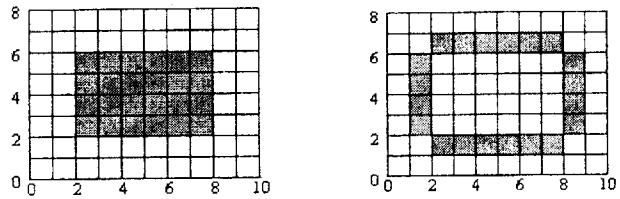
```
    while((c!=BorderColor)&&(c!=Color));
    do
            c=getpixel(++xr,y);
    while((c!=BorderColor)&&(c!=Color));
    x1++;
    xr++;
    line(x1,y,xr,y);
    for(x=x1;x<=xr;x++)
    {
            c=getpixel(x,y+dir);
            if((c!=BorderColor) &&(c!=Color))
                    x=LineFill(x,y+dir,dir,x1,xr);
}
    for(x=x1;x<PrevX1;x++)
    {
            c=getpixel(x,y-dir);
            if((c!=BorderColor) &&(c!=Color))
                    x=LineFill(x,y-dir,-dir,x1,xr);
}
    for(x=PrevXr;x<xr;x++)
    {
            c=getpixel(x,y-dir);
            if((c!=BorderColor) &&(c!=Color))
                    x=LineFill(x,y-dir,-dir,x1,xr);
}
return xr;
}
void Fill(int x,int y);
{
    LineFill(x, y, 1, x, x);
}
main()
{
    int driver=DETECT;
    int mode;
    int res;
    initgraph(&driver, &mode, "");
    if((res=graphresult())!=grOk)
    {
    printf("\nGraphics error: %s\n", grapherrormsg(res));
    exit(1);
          }
          circle(320,200,140);
          circle(260,200,40);
          circle(380,200,40);
          getch();
          setcolor(Color);
          Fill(320,300);
          getch();
          closegraph();
}
```

4.6.1 Алгоритм заповнення області із затравкою

У алгоритмах заповнення із затравкою передбачається, що відомий хоч би один піксель з області багатокутника. Алгоритм намагається знайти і зафарбувати всі інші пікселі, що належать внутрішній області. Області можуть бути внутрішньо- або гранично визначеними. Якщо область відноситься до внутрішньовизначених, то всі пікселі, що належать внутрішній частині, мають один і той же колір або інтенсивність, а всі пікселі, зовнішні відносно області, мають інший колір (рис. 4.15, а).

Якщо область відноситься до гранично визначених, то всі пікселі на межі області мають певне значення або колір (рис. 4.15, б). Жоден з пікселів з внутрішньої частини такої області не може мати цього певного значення. Проте пікселі, зовнішні відносно межі, також можуть мати граничне значення. Алгоритми, що заповнюють внутрішньовизначені області, називаються такими, що внутрішньо заповнюють, а алгоритми для гранично визначених областей – такими, що гранично заповнюють.

Далі йтиме мова про гранично заповнювальні алгоритми. Проте внутрішньозаповнювальні алгоритми можна отримати аналогічно.



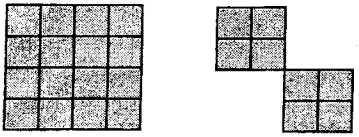
а) б)

Рисунок 4.15 – Внутрішньовизначена область а) і гранично визначена область б)

Внутрішньо- або гранично визначені області можуть бути чотири- або восьмизв'язними. Якщо область чотиризв'язна, то будь-якого пікселя можна досягти за допомогою комбінації рухів лише в 4 напрямах: наліво, направо, вгору, вниз. Для восьмизв'язної області пікселя можна досягти за допомогою комбінації рухів в двох горизонтальних, двох вертикальних і чотирьох діагональних напрямах. Алгоритм заповнення восьмизв'язної області заповнить і чотиризв'язну область, проте обернене правило неправильне.

На рис. 4.16 показані прості приклади чотири- і восьмизв'язних внутрішньовизначених областей. Хоча кожна з підобластей восьмизв'язаної області на рис. 4.16, б є чотиризв'язною, для переходу з однієї підобласті в іншу потрібний восьмизв'язний алгоритм.

Проте у ситуації, коли треба заповнити різними кольорами дві окремі чотиризв'язні підобласті, використання восьмизв'язного алгоритму викличе неправильне заповнення обох областей одним і тим же кольором.



а) б)

Рисунок 4.16 – чотири- (а) та (б) восьмизв'язні внутрішньовизначені області

На рис. 4.17 показана восьмизв'язна область з рис. 4.15, перевизначена у вигляді граничновизначеної області. На рис. 4.17 ілюструється той факт, що для восьмизв'язної області, в якої дві підобласті дотикаються кутами, границя чотиризв'язна, а границя чотиризв'язної області – восьмизв'язна.

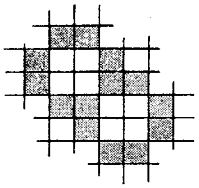


Рисунок 4.17 – чотири- та восьмизв'язні гранично визначені області

Далі мова піде про алгоритми для чотиризв'язних областей, але їх можна легко переробити для восьмизв'язних областей, якщо заповнення проводити не в чотирьох, а у восьми напрямах.

4.6.2 Простий алгоритм заповнення із затравкою

Використовуючи стек, можна розробити простий алгоритм заповнення гранично визначеної області. *Стек* – це просто масив або інша структура даних, в яку можна послідовно поміщати значення і з якого їх можна послідовно витягувати. Коли нові значення додаються або поміщаються в стек, всі останні значення опускаються вниз на один рівень. Коли значення видаляються або витягуються із стека, останні значення випливають або піднімаються вгору на один рівень. Такий стек називається стеком прямої дії або стеком з дисципліною обслуговування «першим прийшов, останнім оброблений» (FILO).

Простий алгоритм заповнення із затравкою можна подати так:

- помістити затравочний піксель в стек;
- поки стек не пустий, витягнути піксель зі стека;
- присвоїти пікселю необхідне значення;
- для кожного із сусідніх до поточних чотиризв'язних пікселів перевірити, чи не є він граничним або чи не присвоєно вже цьому

пікселю необхідне значення. Проігнорувати піксель в будь-якому з цих випадків. В протилежному випадку помістити піксель в стек.

Алгоритм можна модифікувати для восьмиз'язних областей, якщо переглядати восьмизв'язні пікселі, а не лише чотиризв'язні.

Приклад 4.3

Як приклад використання алгоритму дано гранично визначену полігональну область, задану вершинами (1, 0), (7, 0), (8, 1), (8, 4), (6, 6), (1, 6), (0,5) і (0, 1). Область наведена на рис. 4.18.

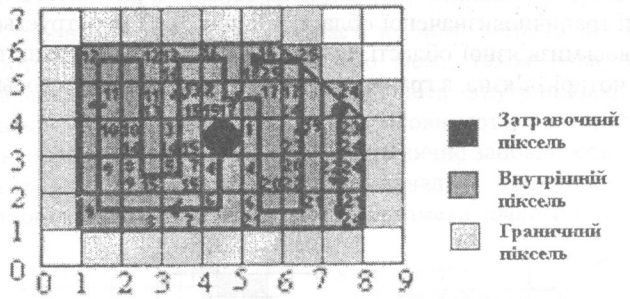


Рисунок 4.18 – Затравочне заповнення за допомогою простого стекового алгоритму

Затравочний піксель – (4, 3). Область заповнюється піксель за пікселем в порядку, вказаному на рис. 4.18 лінією зі стрілками. Числа, наведені на пікселі, позначають поточну при роботі алгоритму позицію пікселя в стеку. Для деяких пікселів таких чисел декілька. Це означає, що піксель поміщали в стек більше ніж 1 раз. Коли алгоритм досягає пікселя (5, 5), стек нараховує 25 рівнів глибини і містить пікселі (7, 4), (7, 3), (7, 2), (7, 1), (6, 2), (6, 3), (5, 5), (6, 4), (5, 5), (4, 4),(3, 3), (3, 4), (3, 5), (2, 4), (2, 3), (2, 2), (2, 2), (3, 2), (5, 1), (3, 2), (5, 2), (3, 3), (4, 4), (5, 3).

Оскільки всі пікселі, що оточують (5, 5), містять або граничні, або нові значення кольору, то жоден з них в стек не поміщається. Отже, зі стека витягується піксель (7, 4) і алгоритм продовжує заповнювати колонку (7, 4) (7, 3) (7, 2) (7, 1). Досягнувши пікселя (7, 1), перевірка знову показує, що навколишні пікселі або вже заповнені, або є граничними пікселями.

Оскільки у цей момент багатокутник повністю заповнений, то витягання пікселів зі стека до повного його спустошення не викликає появу додаткових пікселів, які слід запам'ятати. Коли стек стає порожнім, алгоритм завершує роботу.

4.6.3 Порядковий алгоритм заповнення із затравкою

Даний алгоритм застосовується до граничновизначених областей. Граничновизначена чотиризв'язна область може бути як опуклою, так і

випуклою, а також може містити дірки. У зовнішній і сусідній до граничновизначеної області, не повинно бути пікселів з кольором, яким область або багатокутник заповнюється. Схему роботи алгоритму можна розбити на такі етапи.

1. Піксель затравки на інтервалі витягується зі стека, що містить пікселі затравок.
2. Інтервал з пікселем затравки заповнюється вліво і вправо від затравки вздовж скануючого рядка до тих пір, поки не буде знайдена межа. У змінних $X_{лів}$ і $X_{прав}$ запам'ятовуються крайній лівий і крайній правий пікселі інтервалу.
3. У діапазоні $X_{лів} < x < X_{прав}$ перевіряються рядки, розташовані безпосередньо над і під поточним рядком. Визначається, чи є на них ще не заповнені пікселі. Якщо такі пікселі є (тобто не всі пікселі граничні або вже заповнені), то у вказаному діапазоні крайній правий вказаний піксель в кожному інтервалі визначається як затравка і поміщається в стек.

При ініціалізації алгоритму в стек поміщається єдиний піксель затравки, робота завершується при спустошенні стека. Як показано на рис. 4.19 і в прикладі нижче, алгоритм ефективно працює з дірками і зубцями на границях області.

Приклад 4.4. Нехай дано гранично визначену область, зображену на рис. 4.19. При ініціалізації в стек поміщається піксель затравки, помічений як затравка (6, 7) на рис. 4.19, а. Спочатку як затравка інтервалу зі стека витягується цей піксель. Інтервал заповнюється справа і зліва від затравки. Знайдені при цьому кінці інтервалу $X_{прав} = 9$ і $X_{лів} = 1$.

Потім перевіряється рядок, розташований вище поточного і виявляється, що він граничний і незаповнений. Крайнім правим пікселем в діапазоні $1 < x < 9$ виявляється піксель (9, 9), помічений цифрою 1 на рис. 4.19, а. Цей піксель поміщається в стек. Потім аналогічним чином обробляється рядок нижче поточного. У діапазоні $X_{лів} < x < X_{прав}$ є два підінтервали на цьому рядку. Для лівого підінтервалу як затравка виступає піксель (3, 6), помічений цифрою 2 на рис. 4.19, а, він теж поміщається в стек. Затравка для правого підінтервалу – піксель (9, 6), він поміщається в стек. Відмітимо, що піксель (9, 6) – це не крайній правий піксель на інтервалі, проте це крайній піксель в діапазоні $X_{лів} < x < X_{прав}$, тобто в діапазоні $1 < x < 9$. На цьому завершується перший прохід алгоритму.

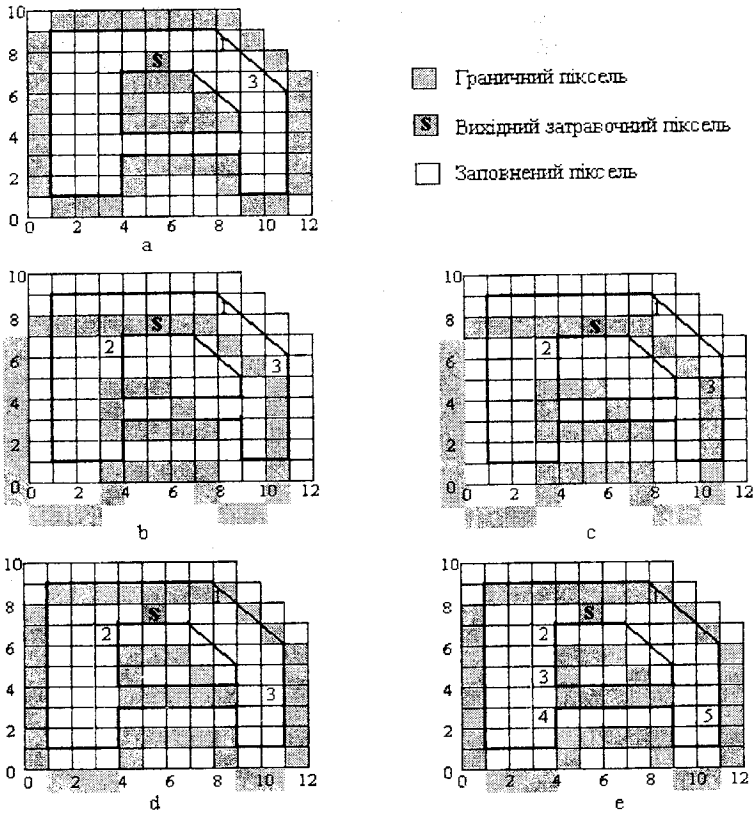


Рисунок 4.19 – Порядковий алгоритм заповнення із затравкою для багатокутника

Далі зі стека витягується верхній піксель. Тут заповнюються інтервали, розташовані в правій частині багатокутника на послідовних скануючих рядках (рис. 4.19, b, c, d). Для рядка 3 затравкою служить піксель (10, 3) (рис. 4.19, d). В результаті заповнення інтервалу справа і зліва від затравки отримуємо нові межі діапазону: $X_{лів} = 1$ і $X_{прав} = 10$. Обробляючи рядок вище, отримуємо для лівого підінтервалу піксель (3, 4) затравки, який поміщається в стек. Правий підінтервал до цього часу вже заповнений. Обробка рядка нижче дає затравку (3, 2) для лівого і (10, 2) для правого підінтервалів. Ці пікселі також поміщаються в стек. Саме зараз досягається максимальна глибина стека для оброблюваного багатокутника.

Тепер залишається лише один цікавий момент. Після заповнення чотиризв'язних полігональних підобластей з пікселями затравок 5, 4 і 3 на рис. 4.19, e зі стека витягується піксель, помічений цифрою 2. Тут виявляється, що всі пікселі на цьому рядку і на сусідніх рядках (вище і

нижче) вже заповнені. Таким чином, жоден піксель в стек не поміщається. Зі стека витягується піксель 1 і рядок обробляється, при цьому знову додаткових пікселів не з'являється. Тепер стек пустий, багатокутник заповнений і робота алгоритму завершена.

Очевидний недолік простого алгоритму зі затравкою, що безпосередньо використовує зв'язність зафарбовуваної області – великі витрати пам'яті на стек, оскільки на кожен зафарбований піксель в стеку по максимуму буде занесена інформація про ще три сусідніх. Крім того, інформація про деякі пікселі може записуватися в стек багато разів. Це приведе не лише до перевитрати пам'яті, але і втрат швидкодії за рахунок багатократного розфарбовування одного і того ж пікселя.

Значно економнішим є розглянутий порядковий алгоритм заливки. Він використовує просторову когерентність:

• пікселі в рядку змінюються лише на границях;

• при переміщенні до наступного рядка розмір рядка, що заливається, швидше за все або незмінний, або змінюється на 1 піксель. Таким чином, на кожен зафарбовуваний фрагмент рядка в стеку зберігаються координати лише одного початкового пікселя, що приводить до істотного зменшення розміру стека.

Контрольні питання та завдання

1. Алгоритми растрової графіки. Типи областей. Зв'язність.
2. Цифровий диференціальний аналізатор.
3. Алгоритм Брезенхема для побудови лінії.
4. Алгоритм Брезенхема для побудови кола.
5. Суть алгоритмів відсікання відрізків.
6. Простий алгоритм двовимірного відсікання.
7. Алгоритм Коена – Сазерленда.
8. Суть алгоритмів зафарбовування областей.
9. Простий алгоритм заповнення із затравкою.
10. Порядковий алгоритм заповнення із затравкою.
11. Використовуючи простий алгоритм відсікання визначте видимість відрізків та координати пересікання відрізків з вікном. Координати вікна (0,0), (4,0), (0,4), (4,4), координати відрізків A(2,5), B(-2,-2); C(3/2, 6), D(6,-1).
12. Побудувати коло радіусом 5 і з центром біля початку координат використовуючи алгоритм Брезенхема для генерації кола.
13. Розкласти відрізок A(0, 0), B(-3, -5) в растр за допомогою узагальненого алгоритму Брезенхема.
14. Використовуючи алгоритм відсікання Коена – Сазерленда, визначте видимість відрізків та координати пересікання відрізків з вікном. Координати вікна (2; 2), (2; -2), (-2; 2), (-2; -2), координати відрізків A(3; 3/2), B(-2; -3); C(-1; 3), D(-3; -1).

Векторна графіка (об'єктно-орієнтовна графіка) – це створення зображення з сукупності геометричних примітивів (точок, ліній, кривих, полігонів), тобто об'єктів, які можна описати математичним рівнянням.

Для векторної графіки характерним є разбиття зображення на ряд графичних примітивів – точки, прямі, ламані, дуги, полігони. Таким чином, з'являється можливість зберігати не всі точки зображення, а координати вузлів примітивів та їхні властивості (колір, зв'язок з іншими вузлами тощо).

Важливою деталлю є те, що об'єкти задаються незалежно один від одного, а отже, можуть перекриватися між собою.

Примітив будується навколо його вузлів (*nodes*). Координати вузлів задаються в координатній системі макета. А зображення буде являти собою масив описів типу:

- відрізок (20, 20 – 100, 80);
- коло (50, 40 – 30);
- крива Без'є (20, 20 – 50, 30 – 100, 50).

Кожному вузлу приписується група параметрів в залежності від типу примітива, що задають його геометрію щодо вузла. Наприклад, коло задається одним вузлом і одним параметром – радіусом.

Такий набір параметрів, що відіграють роль коефіцієнтів та інших величин у рівняннях і аналітичних співвідношеннях об'єкта даного типу, називають аналітичною моделлю примітива. Відрисувати примітив – значить побудувати його геометричну форму за його параметрами відповідно до його аналітичної моделі.

Зрозуміло, що векторним можна назвати лише спосіб опису зображення, проте саме зображення для людського ока завжди растрове. Таким чином, завданнями векторного графічного редактора є растрове пририсовування графічних примітивів і надання користувачеві сервісу зі зміни параметрів цих примітивів. Все зображення являє собою базу даних примітивів і параметрів макета (розміри полотна, одиниці вимірювання тощо). Відрисувати зображення – значить виконати послідовно процедури прорисовування всіх його деталей.

Коли зображення, створене у векторній програмі, виводиться на друк, його якість залежить не від вихідної роздільної здатності зображення, а визначається роздільною здатністю пристроїв виведення (монітора, принтера, плотера тощо). Саме завдяки тому, що якість векторного зображення не пов'язана з роздільною здатністю, файли векторних зображень мають, як правило, менший обсяг у порівнянні з файлами растрових редакторів.

5.1 Застосування векторної графіки та засоби для створення векторних зображень

Сфери застосування векторної графіки є дуже широкими. У поліграфії – від створення кольорових ілюстрацій до роботи зі шрифтами. Все, що є машинною графікою, 3D-графікою, графічними засобами комп'ютерного моделювання та САПР – належить до сфер пріоритету векторної графіки, бо ці галузі комп'ютерних наук розглядають зображення виключно з позиції його математичного подання.

Крім того векторні зображення застосовуються в: комп'ютерній поліграфії, системах комп'ютерного проектування, комп'ютерному дизайні та рекламі.

Векторні графічні зображення є оптимальним засобом зберігання високоточних графічних об'єктів (креслення, схеми тощо), для яких має значення збереження чітких і ясних контурів.

На сьогоднішній час створено безліч пакетів ілюстративної графіки, які містять прості в застосуванні, розвинені і потужні інструментальні засоби векторної графіки, призначеної як для підготовки матеріалів до друку, так і для створення сторінок в Інтернеті.

Для обробки зображень на комп'ютері використовуються спеціальні програми – графічні редактори. Графічні редактори також можна розділити на дві категорії: растрові і векторні.

Насправді, при більш детальному розгляді стає очевидно, що жоден сучасний професійний графічний пакет не є чисто векторним або чисто растровим, а поєднує в собі елементи як того, так і іншого виду графіки. Наприклад, векторний редактор Corel DRAW має як власні, так і компоненти (plug-ins) – інструменти для редагування растрових зображень, а, починаючи з шостої версії растрового редактора Photoshop, розширені інструментальні можливості для роботи з векторними об'єктами.

На сьогодні найбільш широко використовуються такі векторні редактори.

Corel Draw. Пакет Corel Draw завжди справляє сильне враження. У комплект фірма Corel вмістила безліч програм, зокрема Corel Photo-Paint. Новий пакет має в своєму розпорядженні безперечно найпотужніший інструментарій серед всіх програм огляду, а при цьому, у порівнянні з попередньою версією, інтерфейс став простіший, а інструментальні засоби рисування і редагування вузлів – гнучкішими. Проте, що стосується нових функцій, зокрема підготовки публікацій для Web: тут Corel Draw поступається Corel Xara.

Adobe Illustrator. Adobe Illustrator був задуманий як редактор векторної графіки, проте дизайнери використовують його в найрізноманітніших цілях, у тому числі й у вигляді ілюстратора. Він дуже зручний для швидкої розмітки сторінки з логотипом і графікою – простого односторінкового документа. Програма має інтуїтивно зрозумілий інтерфейс, з легким

доступом до багатьох функцій, широким набором інструментів для рисування і покращеними можливостями управління кольором, текстом, що дозволяє створювати векторні зображення будь-якого рівня складності. Adobe Illustrator є одним з найбільш зручних редакторів для створення різних макетів для преси або зовнішньої реклами.

Corel Xara. Служить в першу чергу для створення графічного зображення на сторінці за один раз і формування блока тексту за один раз. Програма дозволяє виконувати найрізноманітніші дії з рисунками, градієнтним заповненням, зображеннями і діапозитивами. Хоча Corel рекламує Corel Xara як доповнення до Corel Draw 7 для створення Web-графіки, по суті, завдяки високій продуктивності, засобам для роботи з Web і спеціалізованому інструментарію Corel Xara перевершує Corel Draw у багатьох відношеннях.

5.2 Порівняння механізмів формування зображень в растровій та векторній графіці

Є велика різниця між механізмами створення растрової і векторної графіки, яка і визначає їх переваги та недоліки і область використання кожного з типів.

Растрова графіка – графіка, подана у машинній пам'яті у вигляді растра. Растровий масив подає сукупність бітів, розташованих на сітчастому полі-канві, інакше кажучи, графічний об'єкт подано у вигляді комбінації точок (пікселів, які мають свій колір та яскравість, та певним чином розташовані на координатній сітці. Такий підхід є ефективним у разі, коли графічне зображення має багато півтонів і інформація про колір важливіша за інформацію про форму (фотографії та поліграфічні зображення).

При редагуванні растрових об'єктів, користувач змінює колір точок, а не форми ліній. Растрова графіка залежить від оптичної роздільності, оскільки її об'єкти описуються точками у координатній сітці певного розміру. Роздільна здатність вказує на кількість точок на одиницю площі. Колір кожного пікселя растрового зображення запам'ятовується за допомогою комбінації бітів, число яких називається бітовою глибиною.

Приклад растрового зображення подано нижче (рис. 5.1). На прикладі видно, що зображення складається із пікселів, які стає видно при зміні розміру зображення.

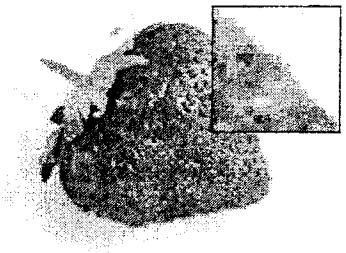


Рисунок 5.1 – Растрове зображення

При створенні векторної графіки зображення описується лініями та фігурами (можливо зафарбованими), з використанням комп'ютерних команд та математичних формул, що описують властивості фігур чи ліній. Це дає можливість пристроям відображення (моніторам, принтерам тощо) при прорисовуванні цих об'єктів визначити розміщення усіх точок. Приклад векторного зображення подано нижче (рис. 5.2).

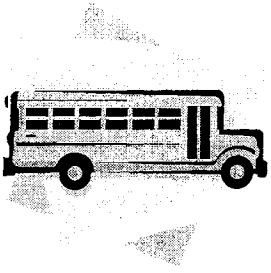


Рисунок 5.2 – Векторне зображення

Різниця у механізмах роботи растрових і векторних редакторів на прикладі опису одного і того ж відрізка прямої полягає:

- *у векторному форматі* – задаються координати початку і кінця прямої, колір і товщина лінії. Для збереження такої інформації на диску буде потрібно всього кілька байтів пам'яті;
- *у растровому форматі* – задаються координати і колір кожної точки (пікселя), що входить в цей відрізок прямої. А оскільки кількість вхідних в неї пікселів залежить від роздільної здатності, то об'єм інформації, необхідної для опису відрізка прямої (а значить, необхідний для її запам'ятовування обсяг пам'яті), буде визначатися встановленою роздільною здатністю.

З наведеного прикладу видно, що векторний формат, як правило, більш компактний (хоча складні рисунки, що містять сотні і тисячі об'єктів, можуть мати розміри, які перевищують розміри растрових зображень). Разом з тим він абсолютно непридатний для зберігання сканованих

зображень, наприклад фотографій. А ось рисунки і креслення набагато зручніше і практичніше робити саме у векторному вигляді.

5.3 Математичні основи векторної графіки

Якщо основним елементом растрової графіки є піксель (точка), то для векторної графіки в ролі базового елемента виступає лінія. Це пов'язано з тим, що у векторній графіці будь-який об'єкт складається з набору ліній, з'єднаних між собою вузлами. Окрема лінія, що з'єднує сусідні вузли, називається сегментом (в геометрії їй відповідає відрізок). Сегмент може бути заданий за допомогою рівняння прямої або рівняння кривої лінії, що потребують для свого опису різної кількості параметрів.

Для більш повного розуміння механізму формування векторних об'єктів потрібно розглянути способи подання основних елементів векторної графіки: точки, прямої лінії, відрізка прямої, кривої другого порядку, кривої третього порядку, кривих Без'є.

У векторній графіці точці відповідає вузол. На площині цей об'єкт описується двома числами (X, Y), що задають його положення відносно початку координат.

Для опису прямої лінії використовується рівняння типу $Y = aX + b$. Тому для побудови даного об'єкта достатньо всього двох параметрів: a і b. Результатом буде побудова нескінченної прямої в декартових координатах. На відміну від прямої, відрізок прямої потребує для свого опису двох додаткових параметрів, що відповідають початку і кінцю відрізка (наприклад, X_1 і X_2).

До класу кривих другого порядку відносяться параболи, гіперболи, еліпси і кола, тобто всі лінії, рівняння яких містять змінні в степені не вище другого. У векторній графіці ці криві використовуються для побудови базових форм (примітивів) у вигляді еліпсів і кіл. Криві другого порядку не мають точок перегину. Канонічне рівняння, що використовується для опису цих кривих, потребує п'ять параметрів:

$$x_2 + a_1 y_2 + a_2 xy + a_3 x + a_4 y + a_5 = 0. \qquad (5.1)$$

Для побудови відрізка кривої потрібно задати два додаткові параметри.

На відміну від кривих другого порядку криві третього порядку можуть мати точку перегину. Наприклад, графік функції $Y = X^3$ має точку перегину на початку координат (0,0). Саме ця особливість даного класу функцій дозволяє використовувати їх як основні криві для моделювання різних природних об'єктів у векторній графіці. Слід зазначити, що згадані раніше прямі і криві другого порядку є окремим випадком кривих третього порядку.

Канонічне рівняння, використовуване для опису рівняння третього порядку, потребує для свого задання дев'яти параметрів:

$$x_3 + a_1 y_3 + a_2 x_2 y + a_3 x y_2 + a_4 x_2 + a_5 y_2 + a_6 xy + a_7 x + \ldots$$
$$\ldots + a_8 y + a_9 = 0. \qquad (5.2)$$

Для опису відрізка кривої третього порядку потребується на два параметри більше.

Криві Без'є – це частковий вид кривих третього порядку, що потребує для свого опису меншої кількості параметрів – восьми замість одинадцяти. В основі побудови кривих Без'є лежить використання двох дотичних, проведених до крайніх точок відрізка лінії (рис. 5.3, праворуч).

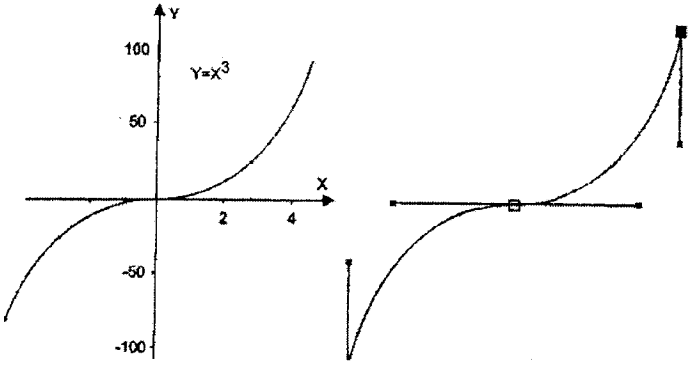


Рисунок 5.3 – Подання кривої лінії за допомогою кривих третього порядку: зліва – класичний варіант, праворуч – крива Без'є

На кривизну (форму) лінії впливає кут нахилу і довжина відрізка дотичної, значеннями яких можна керувати в інтерактивному режимі, шляхом перетягування їх кінцевих точок. Таким чином, дотичні виконують функції віртуальних важелів, що дозволяють керувати формою кривої.

5.4 Елементи (об'єкти) векторної графіки

До основних елементів векторного зображення можна віднести такі.

Лінії. В основі векторної графіки лежить використання математичних уявлень про властивості контурів, основу яких становить елементарний об'єкт векторної графіки – лінія. За її допомогою можна легко побудувати будь-який більш складний об'єкт. Наприклад, об'єкт «чотирикутник» можна створити за допомогою чотирьох ліній, а куб – за допомогою 12 ліній або 6 чотирикутників. Таким чином, ілюстрація складається із простих об'єктів, як із кубиків.

Завдяки цьому процес рисування у векторних редакторах фактично зводиться до створення контурів (об'єктів) потрібної форми і присвоєння

їм певних заливок і обведень. Цей принцип лежить в основі всіх програм векторної графіки. Розрізняються лише прийоми роботи і деякі спеціальні ефекти.

У той же час побудова лінії поряд з використанням для її опису математичного апарату припускає задання ряду додаткових атрибутів, що визначають її основні властивості: форму, товщину, колір, стиль (суцільна, пунктирна тощо). Кількість перерахованих атрибутів залежить від виду лінії. Незамкнені лінії, наприклад, на відміну від замкнених не мають атрибуту заливки (рис. 5.4). Замкнені контури крім обведення можуть мати визначену користувачем заливку.

За замовчуванням контури об'єктів зазвичай не мають товщини. Щоб контур був видимий на екрані, йому надають обведення (абрис) певної товщини, стилю (наприклад, суцільна або пунктирна) і кольору. У більшості редакторів вибір перерахованих атрибутів лінії виконується шляхом використання спеціальних бібліотек, доступ до яких реалізується за допомогою відповідних вікон діалогу.

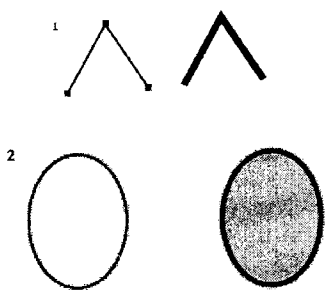


Рисунок 5.4 – Ілюстрація властивостей векторного об'єкта «Лінія»:
1) прямолінійний незамкнений контур (лінія), намальований в програмі CorelXARA 2 інструментом «Рисунок від руки» (Freehand) при натиснутій клавіші Alt без атрибуту обведення (ліворуч) і з доданням обведення (праворуч), товщиною 4 пункти (4pt) ;
2) замкнена лінія у вигляді еліпса без заливки (ліворуч) і із заливкою (праворуч)

Криві Без'є. На початку 70-х років професор П'єр Без'є, проектуючи на комп'ютері корпуси автомобілів «Рено», вперше застосував для цієї мети особливий вид кривих, що описуються рівнянням третього порядку, які згодом стали відомими під назвою «криві Без'є» (функція Bezier).

Оскільки ці лінії мають особливе значення як для векторної, так і растрової графіки, має сенс розглянути їх більш детально.

В наш час криві Без'є подані в будь-якому сучасному графічному пакеті. Досить сказати, що всі комп'ютерні шрифти складаються з кривих

Без'є. Криві Без'є знаходять також широке застосування і в растровій графіці. Так, у програмі Photoshop використовується термін контур (path), що базується на кривих Без'є. Саме за допомогою цього інструменту можна виділити на сканованій фотографії потрібний об'єкт (наприклад, для його вирізання), який буде використаний при створенні фотомонтажу.

Відрізками такої кривої можна апроксимувати який завгодно складний контур. У цьому випадку він буде складатися з набору кривих Без'є. У місцях з'єднань сформована з відрізків кривої Без'є лінія може мати злами. Однак за допомогою функції згладжування (smooth) керуючі точки сусідніх відрізків легко вибудовуються в одну лінію, після чого злам зникає.

Криві Без'є мають ряд властивостей, що визначають можливість їх використання у векторній графіці. З геометричної точки зору, похідною кривої Без'є буде інша крива Без'є, вектори керувальних точок якої визначаються обчисленням різниць векторів керувальних точок вихідної кривої.

Основні властивості кривих Без'є:

– безперервність заповнення сегмента між початковою і кінцевою точками;

– крива завжди розташовується усередині фігури, утвореної лініями, що сполучають контрольні точки;

– за наявності лише двох контрольних точок сегментом є відрізок прямої лінії;

– пряма лінія утворюється при колінеарному (на одній прямій) розміщенні керувальних точок;

– крива Без'є симетрична, тобто обмін місцями між початковою і кінцевою точками (зміна напряму траєкторії) не впливає на форму кривої;

– масштабування і зміна пропорцій кривої Без'є не порушує її стабільності, оскільки вона, з математичної точки зору, «афінно інваріантна»;

– зміна координат хоч би однієї з точок веде до зміни форми всієї кривої Без'є;

– величина кривої завжди на одиницю менша числа опорних точок (тобто при трьох опорних точках форма кривої – парабола);

– розміщення додаткових опорних точок поблизу однієї позиції збільшує її «вагу» і приводить до наближення траєкторії кривої до даної позиції;

– коло не може бути описане параметричним рівнянням кривої Без'є;

– неможливо створити паралельні криві Без'є, за винятком тривіальних випадків (прямі лінії і збіжні криві).

Поява кривих Без'є спричинила справжній переворот у відео- і тривимірній графіці. Це пов'язано з тим, що до появи формул Без'є

контури комп'ютерних персонажів були ламаними, поверхні – гранованими, а рух – переривчастим, стрибкоподібним, неприродним. Використання кривих Без'є дозволило реалізувати найбільш загальний і інтуїтивно зрозумілий спосіб керування рухом. Відповідно до цього параметра кривій можна поставити у відповідність параметри руху комп'ютерного персонажа. В результаті рух відбуватиметься за тим же розглянутими правилами. Таким чином, знаменита крива використовується не тільки в двовимірній комп'ютерній графіці, але й в тривимірній графіці, відео та анімації.

Вузли (Опорні точки). Поряд з лінією (line) іншим основним елементом векторної графіки є вузол (опорна точка). Як уже зазначалося, лінії й вузли використовуються для побудови контурів, які можуть бути подані у вигляді прямої, кривої або форми. Кожен контур має декілька вузлів.

У векторних редакторах (як і в растрових) форму контуру змінюють шляхом маніпуляції вузлами. Це можна зробити одним із таких способів:

- переміщенням вузлів;
- зміною властивостей вузлів (в тому числі атрибутів пов'язаних з ними дотичних ліній і керувальних точок – рис. 5.5);
- додаванням або видаленням вузлів.

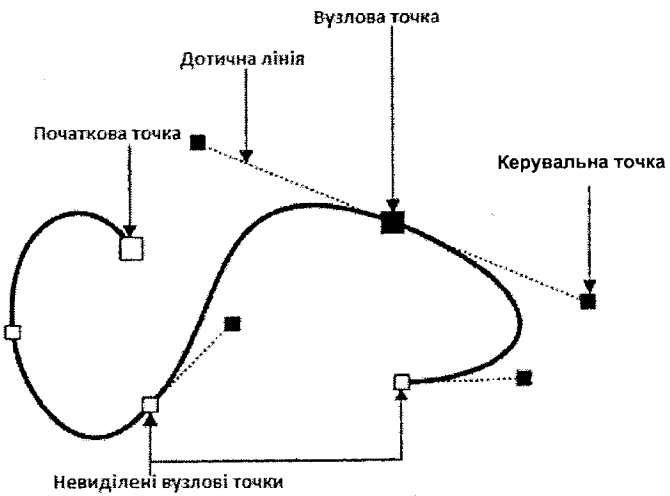


Рисунок 5.5 – Невиділені вузлові точки

Таким чином, в основі всіх процедур, пов'язаних з редагуванням (в тому чилі і створенням) будь-якого типу контурів, лежить робота з вузлами.

Дотичні лінії і керувальні точки. При виділенні вузлової точки криволінійного сегмента у неї з'являються одна або дві керувальні точки, сполучені з вузловою точкою дотичними лініями, Керувальні точки зображуються чорними зафарбованими точками. Розташування дотичних ліній і керувальних точок визначає довжину і форму (кривизну) криволінійного сегмента, а їх переміщення приводить до зміни форми контуру.

Форма і колір керувальних точок також залежать від редактора, що використовується. Якщо в Corel DRAW вони, як і виділені вузли, позначаються чорними квадратиками, але меншого розміру, то в Corel XARA вони зафарбовані червоним кольором, а виділений вузол відображається незафарбованим квадратом.

Розрізняють три типи вузлових точок:

- гладкий вузол (smoothnode);
- симетричний вузол (symmetricalnode);
- гострий вузол (cuspnode).

У *симетричного вузла* обидва відрізки дотичних по обидві сторони точки прив'язки мають однакову довжину і лежать на одній прямій, яка показує напрямок дотичної до контуру в даній вузловій точці (рис. 5.6). Це означає, що кривизна сегментів з обох боків точки прив'язки однакова (в даній точці не зазнають розриву перша і друга похідні кривої).

Зміна положення керувальної точки приводить до відповідної зміни кута нахилу дотичної до кривої. Зміна довжини дотичної лінії з одного боку точки прив'язки шляхом переміщення керувальної точки приводить до відповідної зміни і другої дотичної лінії, що змінює радіус кривизни лінії в точці прив'язки.

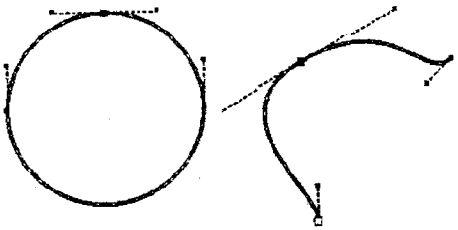


Рисунок 5.6 – У симетричної вузлової точки довжина обох відрізків дотичних однакова і вони лежать на одній прямій

У *гладкої вузлової точки* обидва відрізки дотичних ліній по обидві сторони точки прив'язки лежать на одній прямій, яка показує напрямок дотичної до кривої в даній точці, але довжина керувальних ліній різна (рис. 5.7). Це говорить про те, що кривизна криволінійних ділянок,

прилеглих до цієї опорної точки, різна з різних її сторін. Математично це означає, що в даній точці немає розриву першої похідної, але друга похідна кривої має розрив.

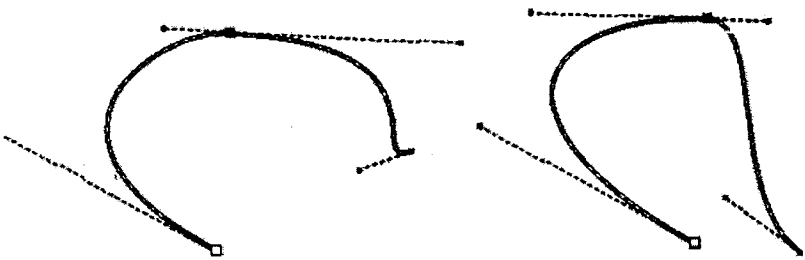


Рисунок 5.7 – У гладкої вузлової точки дотичні лінії лежать на одній прямій, але мають різну довжину

Зміна довжини дотичної лінії з одного боку точки прив'язки шляхом перемещення керувальної точки призводить до відповідної зміни радіуса кривизни цього криволінійного сегмента з одного боку вузлової точки. При цьому довжина другого відрізка дотичної лінії (з іншого боку вузлової точки) не змінюється.

У *гострого вузла* дотичні лінії з різних сторін цієї точки не лежать на одній прямій. Тому два криволінійних сегменти, прилеглих до опорної точки, мають різну кривизну з різних сторін вузлової точки і контур в цій точці утворює різкий злам (рис. 5.8).

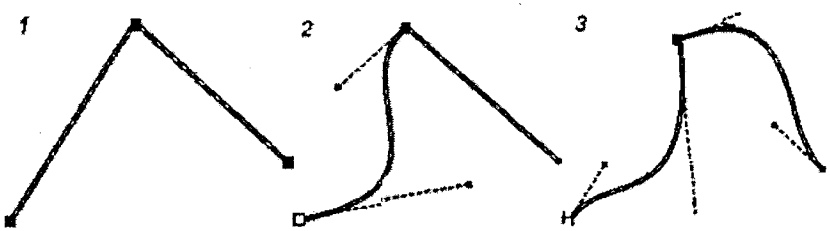


Рисунок 5.8 – Три варіанти гострих вузлів: без керувальних точок (1), з однією керувальною точкою (2) і двома (3). В останньому випадку кривизну сегментів контуру в гострій вузловій точці можна змінювати незалежно для кожного сегмента

Тут радіус кривизни і кут нахилу дотичної для кожного криволінійного сегмента можна регулювати незалежно один від одного відповідною зміною довжини і кута нахилу дотичної лінії для кожного прилеглого до опорної точки криволінійного сегмента окремо. Зокрема один з відрізків

дотичних може дорівнювати нулю (рис. 5.9). У цьому випадку форма сегмента кривої буде регулюватися тільки одним відрізком дотичної, а не двома, як це було в попередніх випадках.

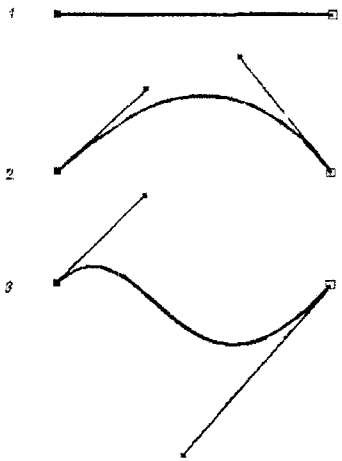


Рисунок 5.9 – Ілюстрація рисування (1) і редагування (2, 3) кривої Без'є в програмі Corel XARA за допомогою інструменту ShapeEditor (редагування фігур)

Примітиви (форми). Разом з різноманітними кривими, основу яких складають криві Без'є, векторні редактори мають у своєму складі спеціальні інструментальні засоби для створення простих форм (графічних примітивів), що спрощує побудову складних об'єктів. Як приклад такого примітиву, можна вказати на еліпсоподібні форми, використовувані при малюванні бруньок на гілці сакури.

Часто поряд зі своїм прямим призначенням прості форми використовуються вихідні заготовки для створення на їх базі більш складних об'єктів. У цьому випадку для подальшого редагування створених заготовок необхідним є залучення розглянутої вище технології редагування кривих Без'є за допомогою переміщення вузлів і керувальних точок.

Для виконання цієї процедури в деяких редакторах потрібно виконати спеціальне перетворення примітивів в криві Без'є, оскільки їх математичний опис у деяких редакторах відрізняється від формул, закладених в побудову кривих Без'є.

Комбіновані об'єкти. Векторне зображення може складатися з десятків і сотень об'єктів (контурів). Всі вони спочатку створюються як прості об'єкти, з яких потім формується складний об'єкт. Досягнутий в результаті цих дій результат необхідно зафіксувати, щоб уникнути при виконанні подальших операцій спотворення рисунка через можливу зміну

співвідношення пропорцій між об'єктами або їх взаємного розташування. Для цих цілей у векторних редакторах передбачена група базових операцій, які включають:

– групування об'єктів;

– об'єднання об'єктів;

– використання складених контурів.

Операція *групування* полягає в об'єднанні двох або більше об'єктів (контурів) в одну групу. З отриманим згрупованим об'єктом можна працювати як з єдиним об'єктом. Його можна переміщувати, повертати, розтягувати і виконувати багато інших операцій без спотворення взаємного розташування і пропорцій вхідних у нього об'єктів.

При реалізації операції групування можна використовувати кілька рівнів групування. У цьому випадку розгрупування об'єктів відбувається в зворотному порядку зі збереженням ієрархії угруповання.

Об'єднаний об'єкт (контур) створюється шляхом використання однієї або декількох операцій з об'єднання двох або декількох контурів. У результаті такої операції з декількох об'єктів виходить новий об'єкт, що має властивості найвищого з вихідних об'єктів, що беруть участь в операції. Тому на відміну від розглянутої раніше операції групування тут властивості складових об'єктів губляться.

У сучасних векторних редакторах передбачені різні варіанти злиття об'єктів. Найбільш поширеними з них є три процедури, принцип дії яких оснований на використанні базових логічних операцій АБО, І, І-НЕ.

5.5 Переваги та недоліки векторної графіки

Переваги векторної графіки:

– перетворення без спотворень. Векторні рисунки можуть бути збільшені або зменшені без втрати якості. Це можливо за рахунок того, що зміна розміру рисунка здійснюється за допомогою простого множення координат точок графічних об'єктів на коефіцієнт масштабування;

– невеликий графічний файл. Невеликий інформаційний обсяг файлів в порівнянні з об'ємом файлів, що містять растрові зображення (розмір не залежить від величини об'єкта);

– для векторних редакторів характерна висока якість друку рисунків і відсутність проблем з експортом векторного зображення в растрове;

– векторну графіку значно легше редагувати, оскільки готове зображення не є «плоскою» картинкою з пікселів, а складено з об'єктів, які можуть накладатися один на одний, перекриватися, залишаючись в той же час абсолютно незалежними один від одного;

– процес рисування – швидкий та простий;

– висока точність прорисовування (до 1 000 000 точок на дюйм);

– редактор швидко виконує операції.

Недоліки векторної графіки:

– векторні зображення виглядають штучно;

– обмеженість у засобах рисування;

– практично неможливо здійснити експорт зображення з растрового формату у векторний;

– векторний принцип опису зображення не дозволяє автоматизувати введення графічної інформації, як це робить сканер для растрової графіки. На жаль, не існує, наприклад, векторних моніторів або векторних сканерів;

– у векторній графіці неможливе застосування великої бібліотеки ефектів (фільтрів), використовуваних при роботі з растровими зображеннями;

– кількість пам'яті і часу на відображення залежить від числа об'єктів і їх складності.

Контрольні питання та завдання

1. Охарактеризуйте термін «векторна графіка», її особливості.
2. Що таке примітив? Наведіть приклади.
3. Охарактеризуйте сфери застосування векторної графіки.
4. Які графічні редактори векторних зображень Ви знаєте? Коротко охарактеризуйте кожен з них.
5. В чому полягає різниця у формуванні растрових та векторних зображень?
6. Охарактеризуйте різницю у механізмах роботи растрових і векторних редакторів на прикладі лінії.
7. Які Ви знаєте способи подання основних елементів векторної графіки?
8. Дайте характеристику елемента векторної графіки – лінії.
9. Дайте характеристику елемента векторної графіки – кривої Без'є.
10. Дайте характеристику елемента векторної графіки – вузла.
11. Які види вузлів існують? В чому особливість кожного виду?
12. Дайте характеристику елемента векторної графіки – форми.
13. В чому полягає різниця між операціями групування та об'єднання елементів векторного зображення?
14. Які переваги та недоліки векторної графіки Ви знаєте?

РОЗДІЛ 6 КОЛІРНІ МОДЕЛІ КОМП'ЮТЕРНОЇ ГРАФІКИ

6.1 Поняття кольору в комп'ютерній графіці

Колір – надзвичайно складна проблема як для фізики, так і для фізіології, тому що він має як психофізіологічну, так і фізичну природу. Сприйняття кольору залежить від фізичних властивостей світла, тобто електромагнітної енергії, від його взаємодії з фізичними речовинами, а також від їхньої інтерпретації зоровою системою людини. Інакше кажучи, колір предмета залежить не тільки від самого предмета, але також і від джерела світла, що освітлює предмет, і від системи людського бачення. Більше того, одні предмети відбивають світло (дошка, папір), а інші його пропускають (скло, вода). Отже, термін «колір» можна визначити так.

Колір – це властивість тіл викликати те або інше зорове відчуття відповідно до спектрального складу відбитого або випромінюваного ними світла.

Природні кольори розділяють на дві групи: хроматичні й ахроматичні.

До групи *ахроматичних* належать білий, сірий і чорний кольори. Вони характеризуються лише кількістю відбитого світла, або, інакше кажучи, неоднаковим коефіцієнтом відбиття. Ахроматичні (безбарвні) кольори відрізняються один від одного лише яскравістю, тобто вони відбивають різну кількість падаючого на них світла. Наприклад, білі поверхні й предмети відбивають 70–90 % падаючого на них світла, а чорні – 3–4 %.

Між найбільш яскравими – білими і найбільш темними – чорними поверхнями є різні відтінки сірого кольору: світло-сірі з коефіцієнтом відбиття 50 – 60 %, темно-сірі з коефіцієнтом відбиття 15 – 20 %. Людське око розрізняє в гамі ахроматичних кольорів близько 300 відтінків.

Хроматичні кольори – це ті кольори і їхні відтінки, які можна розрізнити в спектрі (червоний, жовтогарячий, жовтий, зелений, блакитний, синій, фіолетовий). Хроматичний колір визначається такими фізичними поняттями: колірний тон, насиченість, яскравість, колірна температура.

Колірний тон є таким атрибутом, що дозволяє розрізняти кольори як червоний, жовтий, зелений, синій або як проміжний між двома сусідніми парами цих кольорів. Різниця в колірних тонах у першу чергу залежить від довжини хвилі світла, що попадає в око.

Яскравість визначається енергією, інтенсивністю світлового випромінювання. Виражає кількість сприйнятого світла. Чим вища яскравість, тим світліше колір.

Насиченість або чистота тону показує частку присутності білого кольору. В ідеально чистому кольорі домішки білого відсутні. Чим нижча насиченість, тим більш сірим виглядає колір. При нульовій насиченості колір стає повністю сірим.

Колірна температура характеризує спектральний склад джерела світла, виражається в градусах за шкалою Кельвіна і базується на уявному об'єкті, що називається чорним тілом.

Зустрічаються не чисті монохроматичні кольори, а їхні суміші. В основі трикомпонентної теорії світла лежить припущення про те, що в центральній частині сітківки ока перебувають три типи чутливих до кольору колбочок. Перша сприймає зелений колір, друга – червоний, а третя – синій колір. Відносна чутливість ока максимальна для зеленого кольору й мінімальна для синього. Якщо на всі три типи колбочок впливає однаковий рівень енергетичної яскравості, то світло здається білим. Відчуття білого кольору можна одержати, змішуючи будь-які три кольори, якщо жоден з них не є лінійною комбінацією двох інших. Такі кольори називають основними.

6.2 Колірні моделі

Колірні моделі використовуються для математичного опису певних колірних областей спектра. Більшість комп'ютерних колірних моделей базуються на використанні трьох основних кольорів, що відповідає сприйняттю кольору людським оком.

Кожному основному кольору присвоюється певне значення цифрового коду, після чого всі інші кольори визначаються як комбінація основних кольорів.

Незалежно від того, що лежить у її основі, будь-яка модель повинна задовольняти три вимоги:

1) реалізовувати визначення кольору деяким стандартним способом, що не залежить від можливостей конкретного пристрою;

2) точно відтворювати діапазон відтворених кольорів, оскільки жодна множина кольорів не є нескінченною;

3) враховувати механізм сприйняття кольору – випромінювання або відбиття.

Більшість графічних пакетів дозволяє оперувати широким колом колірних моделей. За принципом дії колірні моделі можна умовно розбити на три класи:

1) адитивні (RGB), основані на додаванні кольорів;

2) субтрактивні (CMY, CMYK), основу яких становить операція віднімання кольорів (субтрактивний синтез);

3) перцепційні (HSB, HLS, Lab, YCC), що базуються на сприйнятті.

Адитивні колірні моделі

Адитивний колір отримують шляхом поєднання променів світла різних кольорів. В основі цього явища лежить той факт, що більшість кольорів видимого спектра можуть бути отримані шляхом змішування в різних пропорціях трьох основних колірних компонентів. Цими компонентами,

які в теорії кольору називаються первинними (основними), є червоний (Red), зелений (Green) і синій (Blue). При попарному змішуванні первинних кольорів утворюються вторинні кольори: блакитний (Cyan), пурпуровий (Magenta) і жовтий (Yellow).

Для одержання нових кольорів за допомогою адитивного синтезу можна використовувати і різні комбінації із двох основних кольорів, варіювання складу яких приводить до зміни підсумкового кольору.

У графічних пакетах колірна модель RGB використовується для створення кольорів зображення на екрані монітора, основними елементами якого є три електронних прожектори і екран з нанесеними на нього трьома різними люмінофорами, що мають різні спектральні характеристики. Один люмінофор під дією падаючого на нього електронного променя випромінює червоний колір, другий – зелений, третій – синій.

Субтрактивні колірні моделі

Для опису друкованих кольорів використовується модель CMY, що базується на субтрактивних кольорах. Субтрактивні кольори отримують видаленням вторинних кольорів із загального променя світла. У цій системі білий колір з'являється як результат відсутності всіх кольорів, тоді як їхня присутність дає чорний колір.

Існують дві найпоширеніші версії субтрактивної моделі CMY і CMYK. Перша з них використовується в тому випадку, коли зображення буде виводитися на чорно-білому принтері. У її основі лежить використання трьох вторинних кольорів (блакитного, пурпурового та жовтого). Теоретично при змішуванні трьох цих кольорів у рівній пропорції на білому папері виходить чорний колір. Однак у реальному технологічному процесі одержання чорного кольору шляхом змішування трьох основних кольорів для паперу неефективне – отримується коричневатий колір. Тому при друці чистого чорного кольору використовується додатково чорний компонент, що приводить до зміни назви колірної моделі: від CMY до CMYK.

Перцепційні колірні моделі

Для усунення апаратної залежності колірних моделей був створений ряд так званих перцепційних колірних моделей. У їхню основу закладене роздільне визначення яскравості й кольоровості. Такий підхід забезпечує ряд переваг:

- дозволяє поводитися з кольорами на інтуїтивно зрозумілому рівні;
- значно спрощує проблему узгодження кольорів, оскільки після установлення значення яскравості можна зайнятися настроюванням кольору.

Модель HSB на відміну від моделей RGB і CMYK носить абстрактний характер. Частково це пов'язано з тим, що колірний тон і насиченість кольору не можна виміряти безпосередньо. Будь-яка форма введення колірної інформації завжди починається з визначення червоної, зеленої та

синьої складових, на базі яких потім за допомогою математичного перерахунку одержують компоненти HSB-моделі.

Десяткове і шістнадцяткове подання кольору

Десяткове подання – це трійка десяткових чисел, що розділені комами. Перше число відповідає яскравості червоної складової, друге – зеленої, а третє – синьої.

Наприклад, червоний колір: (255,0,0); зелений колір: (0,255,0), синій колір: (0,0,255)

Код кольору в шістнадцятковому поданні має вигляд 0xXXXXXX. Префікс 0x вказує на те, що в коді використано власне шістнадцяткове число. Після префіксу зазначено шість шістнадцяткових цифр (0, 1, 2 ..., 9, A, B, C, D, E, F).

Перші дві цифри – шістнадцяткове число, що подає яскравість червоної складової, друга і третя пари відповідають яскравості зеленої та синьої складових.

Наприклад, червоний колір: 0xFF0000; зелений: 0x00FF00; синій: 0x0000FF.

Якщо всі складові мають максимальну яскравість (255,255,255 – в десятковому поданні; 0xFFFFFF – в шістнадцятковому поданні), виходить білий колір. Мінімальна яскравість (0, 0, 0 або 0x000000) відповідає чорному кольору.

6.2.1 Закон Грассмана (закон змішування кольорів)

У більшості колірних моделей для опису кольору використовується тривимірна система координат. Вона утворює колірний простір, у якому колір можна подати у вигляді точки з трьома координатами. Для оперування кольором у тривимірному просторі Т. Грассман вивів три закони (1853 р.):

1. Колір тривимірний – для його опису необхідні три компоненти. Будь-які чотири кольори перебувають у лінійній залежності, хоча існує необмежене число лінійно незалежних сукупностей із трьох кольорів. Іншими словами, для будь-якого заданого кольору можна записати таке колірне рівняння, що виражає лінійну залежність кольорів.

2. Якщо в суміші трьох колірних компонентів один змінюється безупинно, у той час, як два інших залишаються постійними, колір суміші також змінюється безупинно.

3. Колір суміші залежить тільки від кольорів компонентів, що змішуються, і не залежить від їхніх спектральних складів.

Зміст третього закону стає більш зрозумілим, якщо врахувати, один і той же колір (у тому числі й колір компонентів, що змішуються,) може бути отриманий різними способами. Наприклад, змішувальний компонент може бути отриманий, у свою чергу, змішуванням інших компонентів.

6.3 Колірна модель RGB

Це одна з найпоширеніших і часто вживаних моделей. Вона застосовується в приладах, що випромінюють світло, таких, наприклад, як монітори, прожектори, фільтри та інші подібні пристрої.

Дана колірна модель базується на трьох основних кольорах: Red – червоному, Green – зеленому і Blue – синьому. Кожна з перерахованих складових може варіюватися в межах від 0 до 255, утворюючи різні кольори і забезпечуючи, таким чином, доступ до 16 мільйонів кольорів (повна кількість кольорів, що подаються цією моделлю дорівнює 256×256×256 = 16 777 216.).

Колір, що створюється змішуванням трьох основних компонентів, можна подати вектором у тривимірній системі координат *R*, *G* і *B*, що зображена на рис. 6.1.

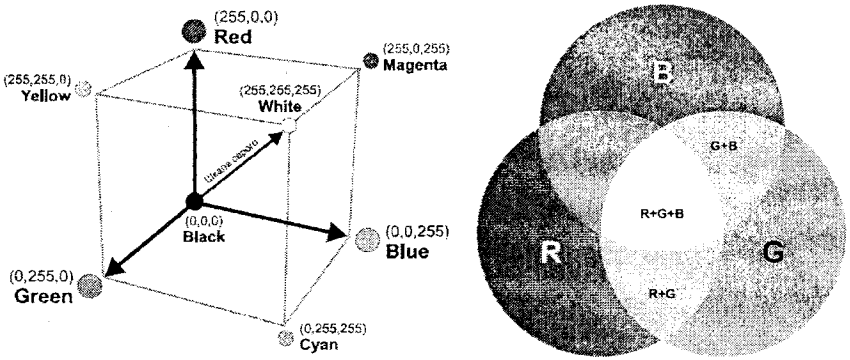


Рисунок 6.1 – Колірна модель RGB

Чорному кольору відповідає центр координат – точка (0,0,0). Білий колір виражений максимальним значенням компонентів. Нехай це максимальне значення уздовж кожної осі дорівнює одиниці. Тоді білий колір – це вектор (1, 1, 1). Точки, які лежать на діагоналі куба від чорного до білого, мають однакові значення координат: $R_I = G_I = B_I$. Це градації сірого – їх можна вважати білим кольором різної яскравості. Отже, якщо всі компоненти вектора *(r, g, b)* помножити на однаковий коефіцієнт (k = 0..1...1), то колір *(kr, kg, kb)* зберігається, змінюється тільки яскравість. Тому для аналізу кольору важливе співвідношення компонентів. Якщо в колірному рівнянні

$$K = rR + gG + bB$$

розділити коефіцієнти r, g і b на їх суму:

$$r' = \frac{r}{r+g+b}, r' = \frac{r}{r+g+b}, r' = \frac{r}{r+g+b},$$

то можна записати таке колірне рівняння:

$$K = r' R + g' G + b' B.$$

Це рівняння являє вектори кольору (r', g', b'), які лежать в одиничній площині $r' + g' + b' = 1$. Іншими словами, від куба отримуємо трикутник Максвелла.

Трикутник Максвелла (рис. 6.2) – один із видів подання колірних моделей. Вершини трикутника Максвелла відповідають положенню трьох основних кольорів: червоного (R), зеленого (G) і синього (B). У центрах сторін трикутника розташовуються додаткові жовтий (Y), пурпуровий (M) і блакитний (C) кольори, а в точці перетину перпендикулярів, проведених від сторін трикутника, знаходиться білий колір (W), що задається координатами $r = g = b = 1/3$. На лініях W-R, W-G, W-B розташовуються червоний, зелений і синій кольори, а на лініях W-C, W-M, W-Y – додаткові – жовтий, пурпуровий і блакитний кольори за ступенем збільшення їхньої насиченості. Колір $A1$ визначаємо координатами кольоровості r = 0,1; g = 0,2 і отримується змішуванням червоного, зеленого і синього кольорів у відповідних пропорціях, а колір $A2$ з координатами кольоровості r = 0,8; g = -0,2 лежить поза колірним трикутником і не може бути отриманий змішуванням червоного, зеленого і синього кольорів (лежить поза їхнім колірним охопленням)

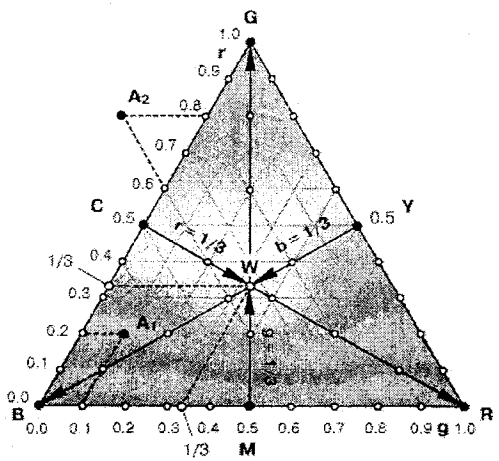


Рисунок 6.2 – Трикутник Максвелла

Модель *RGB* є найпоширенішою і офіційним стандартом (згідно з рішенням Міжнародної комісії з освітлення – МКО (Commision International de VEclairage) в 1931 році були стандартизовані основні кольори, які було рекомендовано використовувати як *R*, *G* і *B*), проте у той же час їй властивий важливий недолік: не всі кольори, видимі людиною, подані в цій моделі. Наприкінці 1920- х років В. Д. Райтом і Дж. Гілдом були проведені експерименти, у яких спостерігачеві пропонувалося кожний монохроматичний колір фіксованої яскравості у видимому діапазоні зіставити з кольором, складеним із суміші основних кольорів *R*, *G* і *B* з деякими вагами, регульованими спостерігачем. Виявилося, що для деяких кольорів необхідно було додати окремо яскравості випробуваного світла і одного з базисних кольорів (був вибраний *R*), для того щоб одержати однакове сприйняття. Такий ефект повязаний з тим, що хвилі з видимого діапазону впливають відразу на всі типи колбочок і не завжди можна обмежитися додатними коефіцієнтами для подання деяких кольорів з видимого спектра.

6.4 Колірні моделі CMY і CMYK

Колірну модель *CMY* (від англ. Cyan, Magenta, Yellow – блакитний, пурпурний, жовтий) можна отримати з моделі *RGB* за допомогою перетворення:

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

Аналогічно виразити модель *RGB* через набір компонент моделі *CMY* можна так:

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

Ця колірна модель використовується для опису кольору при одержанні зображень на пристроях, які реалізують принцип поглинання (видалення) кольорів.

Цю модель використовують для підготовки не екранних, а друкованих зображень. Вони відрізняються тим, що їх бачать у відбитому світлі. Чим більше фарби покладено на папір, тим більше світла вона поглинає і менше відбиває. Сполучення трьох основних кольорів поглинає майже все падаюче світло і з боку зображення виглядає майже чорним. На відміну від моделі *RGB*, збільшення кількості кольору приводить не до збільшення візуальної яскравості, а навпаки, до її зменшення. Тому для підготовки

друкованих зображень використовується не адитивна (підсумовувальна) модель, а субтрактивна (та, що віднімає). Колірними компонентами цієї моделі є не основні кольори, а ті, які виходять у результаті віднімання основних кольорів з білого:

Блакитний (Cyan) = Білий – Червоний = Зелений + Синій
Пурпурний (Magenta) = Білий – Зелений = Червоний + Синій
Жовтий (Yellow) = Білий – Синій = Червоний + Зелений

Ці три кольори називаються додатковими, тому що доповнюють основні кольори до білого.

Істотні труднощі в поліграфії являє чорний колір. Теоретично його можна одержати сполученням трьох основних або додаткових кольорів, але на практиці результат виявляється незадовільним. Тому в колірну модель *CMY* доданий четвертий компонент – чорний, саме система *CMYK* зобов'язана буквою *K* у назві (blacK).

Модель *CMY* зазвичай використовується в тому випадку, якщо зображення або рисунок будуть виводитися на чорно-білому принтері, що дозволяє заміняти чорний картридж на кольоровий (color upgrade).

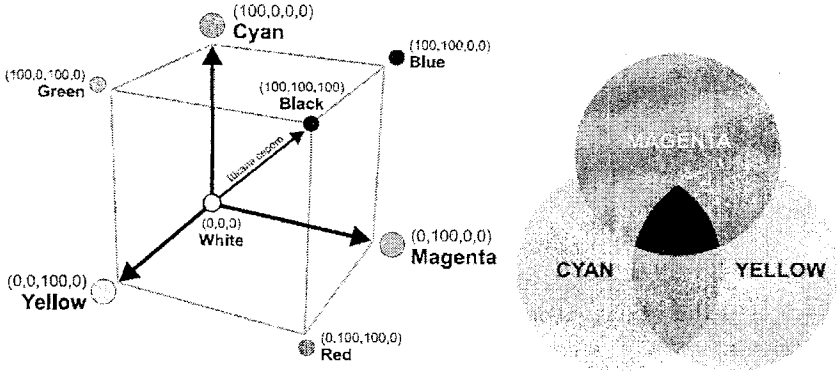


Рисунок 6.3 – Колірна модель *CMYK*

CMYK-модель має ті ж два типи обмежень, що й *RGB*-модель: апаратна залежність та обмежений колірний діапазон.

В *CMYK*-моделі також не можна точно передбачити підсумковий колір тільки на базі числових значень його окремих компонентів. У цьому сенсі вона є навіть більше апаратно-залежною моделлю, ніж *RGB*. Це пов'язане з тим, що в ній є більша кількість дестабілізуючих факторів. До них можна віднести варіацію складу кольорових барвників, що використовуються для створення друкованих кольорів. Колірне відчуття визначається ще й типом застосовуваного паперу, способом друку, а також зовнішнім освітленням.

У силу того, що кольорові барвники мають гірші характеристики в порівнянні з люмінофорами, колірна модель *CMYK* має більш вузький колірний діапазон у порівнянні з *RGB*- моделлю. Зокрема вона не може відтворювати яскраві насичені кольори, а також ряд специфічних кольорів, таких, наприклад, як металевий або золотавий.

Не зважаючи на ці недоліки, варто пам'ятати, що якщо при підготовці зображення до друку, потрібно працювати з *CMYK*, тому що в іншому випадку те, що ви побачите на моніторі, і те, що одержите на папері, буде відрізнятися настільки сильно, що вся робота може зійти нанівець. Отже, модель *CMYK* – це субтрактивна колірна модель, що описує реальні барвники, які використовуються в поліграфічному виробництві.

6.5 Колірна модель HSB (Hue, Saturation, Brightness)

Якщо дві вищеописані моделі подати у вигляді єдиної моделі, то можна отримати усічений варіант колірного кола, у якому кольори розташовуються у такому порядку: червоний (*R*), жовтий (*Y*), зелений (*G*), блакитний (*C*), синій (*B*) (рис. 6.4).

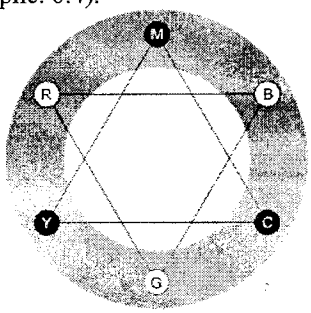


Рисунок 6.4 – Колірне коло

На колірному колі основні кольори моделей *RGB* і *CMY* перебувають у такій залежності: кожний колір розташований напроти доповнювального його (комплементарного) кольору; при цьому він перебуває між кольорами, за допомогою яких він був отриманий. Щоб підсилити певний колір, потрібно послабити колір, що його доповнює (розташований напроти нього на колірному колі). Наприклад, щоб змінити загальне колірне рішення у бік блакитних тонів, варто знизити в ньому вміст червоного кольору.

По краю цього колірного кола розташовуються так звані спектральні кольори або **колірні тони (Hue),** які визначаються довжиною світлової хвилі, відбитої від непрозорого об'єкта або хвилі, що пройшла через прозорий об'єкт. Колірний тон характеризується положенням на колірному колі і визначається величиною кута в діапазоні від 0 до 360 градусів. Ці кольори мають максимальну насиченість, тобто синій колір ще більш синім бути не може.

Наступним параметром є **насиченість кольору (Saturation)** – це параметр кольору, що визначає його чистоту. Зменшення насиченості кольору означає його освітлювання. Колір зі зменшенням насиченості стає пастельним, бляклим, розмитим. На моделі всі однаково насичені кольори розташовуються на концентричних колах, тобто можна говорити про однакову насиченість, наприклад, зеленого і пурпурового кольорів, і чим ближче до центра кола, тим все більше освітленими стають кольори. У самому центрі будь-який колір максимально освітлюється й стає білим кольором. Роботу з параметром насиченості можна характеризувати як додавання в спектральний колір певного відсотка білого кольору.

Ще одним параметром є **яскравість (Brightness)** – це параметр кольору, що визначає освітленість або затемненість. Зменшення яскравості кольору означає його затемнення. Роботу з параметром яскравості можна характеризувати як додавання в спектральний колір певного відсотка чорного кольору. У загальному випадку будь-який колір виходить зі спектрального кольору додаванням певного відсотка білої й чорної фарб, тобто фактично сірої фарби.

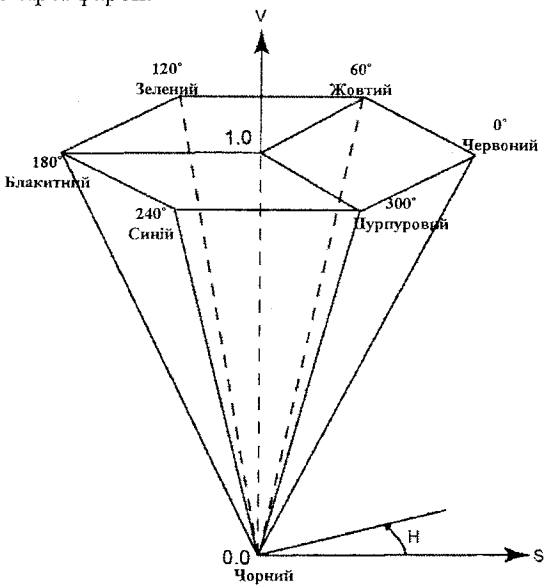


Рисунок 6.5 – Колірний простір HSB

Моделі *RGB*, *CMY* і *CMYK* орієнтовані на роботу з кольоропередавальною апаратурою і незручні для задання кольору людиною. З іншого боку, модель, названа HSB, більше орієнтована на роботу з людиною й дозволяє задавати кольори, опираючись на інтуїтивні поняття тону, насиченості і яскравості. Для більш детального

ознайомлення з моделлю *HSB* варто розглянути циліндричну систему координат, де множина всіх припустимих кольорів являє собою шестигранний конус, поставлений на вершину (рис. 6.5).

Можна навести приклади кодування кольрів для моделі *HSB*. При $S = 0$ (тобто на осі B) – сірі тони. Значення $B = 0$ відповідає чорному кольору. Білий колір кодується як $S = 0$, $B = 1$. Кольори, розташовані по колу навпроти один одного, тобто відрізняються по H на 180°, є додатковими. При цьому червоному кольору відповідає кут 0°, зеленому – кут 120° і т. д. Процес додавання білого кольору до заданого можна подати як зменшення насиченості S, а процес додавання чорного кольору – як зменшення яскравості B.

Задання кольору за допомогою параметрів *HSB* досить часто використовується в графічних системах, причому звичайно показується розгортка конуса. Недоліком цієї моделі є необхідність перетворювати її в модель *RGB* для відображення на екрані монітора або в модель *CMYK* для одержання поліграфічного зображення.

Нижче наведено процедуру переведення з *RGB* моделі в *HSB* і навпаки. Спочатку розглянемо перетворення з моделі *HSB* у модель *RGB*.

0 <= hue <= 360 градусів – відтінок.

Основні кольори: 0 – червоний, 60° – жовтий, 120° – зелений, 180° – блакитний, 240° – синій, 300° – пурпуровий. Інші кольори між ними:

0.0 <= sat <= 1.0 – насиченість

0.0 <= br <= 1.0 – яскравість

```
int  V_HSBRGB (r, g, b, hue, sat, br)
float *r, *g, *b, hue, sat, br;
{  int ii, otw;
   float c1, c2, c3, fr;

   otw= 0;
   if (sat == 0.0) {          /* Ахроматичний колір */
     *r= val;  *g= br;  *b= br;
     if (hue != UNDEFINED) ++otw;
   } else {                   /* Хроматичний колір */
     hue-= (ii= (int)(hue/360.0)) * 360.0;
     if (ii < 0) hue= -hue;
     ii= (int)(hue /= 60,0);
     fr= hue - ii;
     c1= br*(1.0 - sat);
     c2= br*(1.0 - sat*fr);
     c3= br*(1.0 - sat*(1.0 - fr));
     switch (ii) {
       case 0: *r= br; *g= c3;  *b= c1;  break;
       case 1: *r= c2;  *g= br; *b= c1;  break;
       case 2: *r= c1;  *g= br; *b= c3;  break;
       case 3: *r= c1;  *g= c2;  *b= br; break;
       case 4: *r= c3;  *g= c1;  *b= br; break;
```

```
      case 5: *r= br; *g= c1;  *b= c2;  break;
    }
  }
  return (otw);
}
```

Перетворення з моделі *HSB* у модель *RGB*.

```
int  V_RGBHSV (r, g, b, hue, sat, br)
float r, g, b, *hue, *sat, *br;
{  int  otw;
  float minc, maxc, h, s, v, dmax, rc, gc, bc;
  otw= 0;
  if (r < 0.0 || r > 1.0) ++otw;   /* Перевірка значень */
  if (g < 0.0 || g > 1.0) otw= 2;
  if (b < 0.0 || b > 1.0) otw= 3;
  if (!otw) {
    if ((maxc= r) < b) maxc= b;   /* Пошук максимального значення */
    if (maxc < g) maxc= g;
    if ((minc= r) > b) minc= b;   /* Пошук мінімального значення */
    if (minc > g) minc= g;
    s= 0.0;                       /* Насиченість      */
    if (maxc != 0.0) s= (maxc-minc)/maxc;
    if (s == 0.0) h= UNDEFINED;   /* Ахроматичний колір */
    else {                        /* Хроматичний  колір */
      dmax= maxc-minc;
      rc= (maxc-r)/dmax;          /* rc – віддаленість   */
      gc= (maxc-g)/dmax;          /* кольору від червоного */
      bc= (maxc-b)/dmax;
      if (r == maxc) h= bc-gc; else   /* Колір між жовтим і пурпуровим */
      if (g == maxc) h= 2+rc-bc; else /* Колір між блакитним і жовтим    */
      h= 4+gc-rc;                 /* Колір між пурпуровим і блакитним*/
      if ((h*= 60.0) < 0.0) h+= 360.0;
    }
    *hue= h;  *sat= s;  *br= maxc;
  }
  return (otw);
}
```

6.6 Інші колірні моделі

Колірна модель CI XYZ

З урахуванням особливостей сприйняття кольору людиною, в 1931 році CI (Commission Internationale de l'Eclairage) затвердила стандарт (колірну модель CI XYZ), що описує будь-який колір за допомогою незалежних змінних *X, Y* і *Z*. Їхній спектр (рис. 6.6) вибраний таким чином, що дозволяє описати будь-який колір з видимого діапазону. Величини *X, Y* і *Z* визначаються через спектральний розподіл енергії відповідного кольору, а

фізичний зміст їх такий, що Y являє собою складову яскравості, а X і Z – кольору. Величини X, Y і Z задаються співвідношеннями:

$$X = \int I(\lambda)\bar{x}(\lambda)d\lambda,$$
$$Y = \int I(\lambda)\bar{y}(\lambda)d\lambda,$$
$$Z = \int I(\lambda)\bar{z}(\lambda)d\lambda.$$

де $I(\lambda)$ – спектральна функція розподілу для кольору, що подається. Цими трьома числами X, Y і Z будь-який колір, що сприймається людським оком, можна охарактеризувати однозначно.

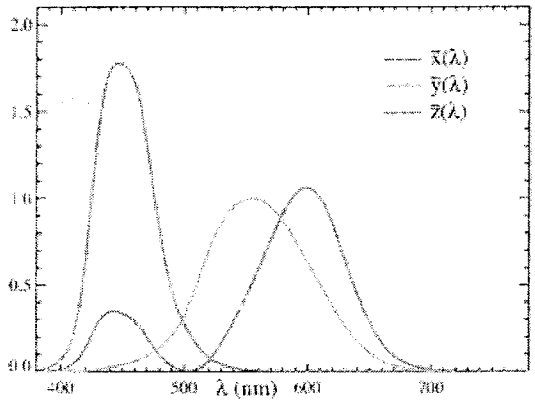


Рисунок 6.6 – Інтенсивності незалежних змінних СІ XYZ

Формули для переходу від системи СІ *XYZ* до системи *RGB:*

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3{,}240479 & -1{,}537156 & -0{,}498535 \\ -0{,}969256 & 1{,}875992 & 0{,}041556 \\ 0{,}055648 & -0{,}204043 & 1{,}057311 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

Якщо певний колір не може бути поданий в *RGB* моделі, то в нього хоча б один з компонентів буде або від'ємним, або більшим одиниці.

Обернене перетворення з *RGB* в СІ *XYZ* має такий вигляд:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0{,}412453 & 0{,}357580 & 0{,}180423 \\ 0{,}212671 & 0{,}715160 & 0{,}072169 \\ 0{,}019334 & 0{,}119193 & 0{,}950227 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}.$$

Варто відмітити, що СІ *XYZ* – це єдина відома колірна модель, здатна описати будь-який колір з видимого діапазону і всі інші моделі є її

підмножинами. Однак користуватися тривимірною моделлю не зовсім зручно, і, крім того, вона нелінійна стосовно кольоросприйняття людини. Це означає, що допуски на колір (область нерозрізнених відтінків), виражені через евклідову відстань, у різних колірних діапазонах мають різне значення. CI згодом доробила цю модель із урахуванням описаних недоліків. У результаті з'явилися моделі L*u*v і L*a*b. Перша використовується для джерел світла, наприклад, дисплеїв, а друга – для відбитого світла.

Колірна модель CIE L*a*b

Щоб усунути нелінійність *XYZ* в 1960 році Мак-Адам запропонував простір *UVW*. В 1964 році Вишецьким була запропонована модель *U*V*W*. В 1966 році був запропонований Hunter L,a, b, а в 1976 році після усунення розбіжностей була розроблена модель CIE L*a*b, що є зараз міжнародним стандартом.

Всі ці колірні простори прагнули зменшити нелінійність зміни кольору в різних частинах області колірного простору, однак, ідеальнішого цього стандарту так і не з'явилося. В Hunter Lab спостерігається стиснення у жовтій частині й розширення в синій. В CIE L*a*b, хоч вона розроблена на основі Hunter Lab і повинна була усунути наявні недоліки, відзначається розширення в жовтій частині.

Обидва колірних простори обчислюються із простору CI 1931 *XYZ*, однак перетворення в CIE L*a*b здійснюються з використанням кубічного кореня, у той час як в Hunter Lab використані квадратні.

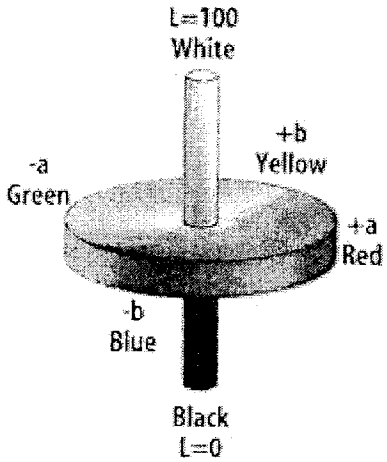


Рисунок 6.7 – Колірна модель CIE L*a*b

Колірна модель CIE L*a*b визначається яскравістю (Lightness), що змінюється від 0 до 100, і двома колірними діапазонами із числовими значеннями від -60 до 60: a – від зеленого до пурпурного (magenta) і b – від синього до жовтого. Модель є апаратно незалежною й тому часто застосовується для перенесення даних між пристроями.

Колірне охоплення моделі CIE L*a*b значно перевершує можливості моніторів і друкованих пристроїв, тому перед виведенням зображення, поданого в цій моделі, його доводиться перетворювати. Дана модель була розроблена для узгодження кольорових фотохімічних процесів з поліграфічними. Сьогодні вона є прийнятим за замовчуванням стандартом для програми Adobe Photoshop. Даний графічний редактор при переході від режиму *RGB* до *CMYK* використовує Lab як проміжний етап.

Модель *RGB* орієнтована, в основному, на особливості випромінюваного світла (монітор), а *CMYK* – на особливості світла, що поглинається (принтер). Колірні діапазони цих моделей не збігаються: RGB добре відтворює кольори в діапазоні від синього до зеленого і дещо гірше – жовті і оранжеві відтінки, а в моделі *CMYK* бракує багатьох відтінків. Модель L*a*b позбавлена таких недоліків. В рамках цієї моделі працюють багато професіоналів комп'ютерної графіки.

Зір людини має одну особливість: око більш сприйнятливе до зміни яскравості, ніж кольору. Колірна модель **YCbCr** використовує цю особливість, що дозволяє більш ефективно стискати зображення. Термін *Y* означає яскравість (luminosity), *Cb* – кольоровість (chrominance) для вихідного синього, а *Cr* – кольоровість для вихідного червоного. Вихідний зелений може бути отриманий з комбінації цих трьох значень.

Перетворення колірного простору: [*R G B*] в [*Y Cb Cr*]

$$\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} = \begin{vmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}1687 & -0{,}3313 & 0.5 \\ 0.5 & -0{,}4187 & -0{,}0813 \end{vmatrix} \begin{vmatrix} R \\ G \\ B \end{vmatrix} \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix}$$

Нова величина $Y = 0{,}299 \times R + 0{,}587 \times G + 0{,}114 \times B$.

Величини $Cb = -0{,}1687 \times R - 0{,}3313 \times G + 0{,}5 \times B + 128$,

і $Cr = 0{,}5 \times R - 0{,}4187 \times G - 0{,}0813 \times B + 128$.

Перетворення *[Y,Cb,Cr]* в *[R,G,B]*

$R = Y + 1{,}402 \times (Cr - 128)$,

$G = Y - 0{,}34414 \times (Cb - 128) - 0{,}71414 \times (Cr - 128)$,

$B = Y + 1{,}772 \times (Cb - 128)$.

Колірна модель Grayscale

Сіра шкала (Grayscale) застосовується для відображення чорно-білих фотографій або зображень для чорно-білої поліграфії.

Традиційна сіра шкала, що використовує на кожний піксель зображення в один байт інформації, може передавати 256 відтінків (градацій) сірого кольору або яскравості (Brightness): значення 0 являє чорний колір, а значення 255 – білий. Сіра шкала може виражатися й у процентних відношеннях, у цьому випадку 0% являє білий колір (відсутність фарби на білому папері), а 100% – чорний колір.

У таблиці 6.1 наведено порівняння різних колірних моделей.

Таблиця 6.1 – Порівняння колірних моделей

Колірна модель	Лінійність	Незалежність від пристрою	Базується на	Тип
Grayscale	+	–		півтонова
Index colors	–	–	RGB	
CIE XYZ	–	+		всі кольори
CIE L*a*b	+	+		субтрактивна, всі кольори
CIE L*u*v	+	+		адитивна, всі кольори
RGB	–	–		адитивна
HSB	–	–	RGB	
YCbCr	+	+		адитивна
CMYK	–	–		субтрактивна

Перетворення колірних моделей

Перетворення зображення з однієї колірної моделі в іншу в Adobe Photoshop виконується винятково просто. Для цього призначені команди списку Mode (Режим) меню Image (Зображення). У ньому знаходяться команди *RGB* Color (Кольори *RGB*), *CMYK* Color (Кольори *CMYK*) і Lab Color (Кольори Lab). Проте, простота перетворення колірних моделей оманна і не варто звертатися до такого перетворення без особливої необхідності. Будь-яке перетворення з *RGB* в *CMYK* або навпаки пов'язане зі зміною колірного охоплення, яке щоразу погіршує якість зображення. Перехід між колірними моделями допустимо виконувати тільки один раз. Якщо, наприклад, готується зображення до друку, тоді можливе перетворення в модель *CMYK*. Модель Lab має настільки широке колірне охоплення, що воно повністю вміщає як колірний простір *CMYK*, так і RGB. Тому перетворення в Lab і навпаки не змінює якість зображення і цілком безпечне.

6.7 Колірні палітри

Для зберігання кожного з компонентів кольору приділяється фіксоване число *n* біт пам'яті. Тому вважається, що допустимий діапазон значень для компонентів кольору не [0; 1], а [0; 2*n*-1].

Практично будь-який відеоадаптер здатний відобразити значно більшу кількість кольорів, ніж ту, що визначається розміром відеопам'яті, яка відводиться під один піксель. Для використання цієї можливості вводиться поняття палітри.

Палітра – масив, у якому кожному можливому значенню пікселя ставиться у відповідність значення кольору (*r*, *g*, *b*). Розмір палітри і її організація залежать від типу використовуваного відеоадаптера.

Найбільш простою є організація палітри на *EGA*-адаптері. Під кожний з 16 можливих логічних кольорів (значень пікселя) приділяється 6 біт, по 2 біти на кожний колірний компонент. При цьому колір у палітрі задається байтом вигляду 00*rgbRGB*, де *r*, *g*, *b*, *R*, *G*, *B* можуть приймати значення 0 або 1. Таким чином, для кожного з 16 логічних кольорів можна задати будь-який з 64 можливих фізичних кольорів.

16-кольорова стандартна палітра для *відеорежимів EGA, VGA.* Реалізація палітри для 16-кольорових режимів адаптерів *VGA* набагато складніша. Крім підтримки палітри адаптера *EGA* відеоадаптер додатково містить 256 спеціальних *DAC*-регістрів, де для кожного кольору зберігається його 18-бітове подання (по 6 біт на кожний компонент). При цьому з вихідним логічним номером кольору з використанням 6-бітових регістрів палітри *EGA* зіставляється, як і раніше, значення від 0 до 63, але воно вже є не *RGB*-розкладанням кольору, а номером *DAC*-регістра, що містить фізичний колір.

256-кольорова для VGA. Для 256-*VGA* значення пікселя безпосередньо використовується для індексації масиву *DAC*-регістрів.

У цей час досить розповсюдженим є формат True Color, у якому кожний компонент поданий у вигляді байта, що дає 256 градацій яскравості для кожного компонента: R = 0...255, G = 0...255, B = 0...255. Кількість кольорів становить 256×256×256 = 16,7 млн (2^{24}). Такий спосіб кодування можна назвати *компонентним.*

При роботі із зображеннями в системах комп'ютерної графіки часто доводиться шукати компроміс між якістю зображення і ресурсами, необхідними для зберігання і відтворення зображення. При обмеженні кількості кольорів використовують палітру, що надає набір кольорів, необхідний для даного зображення. Палітру можна сприймати як таблицю кольорів. Палітра встановлює взаємозв'язок між кодом кольору і його компонентами у вибраній колірній моделі.

Комп'ютерні відеосистеми звичайно надають можливість програмістові встановити власну колірну палітру. Кожний колірний відтінок подається одним числом, причому це число виражає не колір пікселя, а індекс

кольору (його номер). Сам же колір розшукується за цим номером в супровідній колірній палітрі. Такі колірні палітри називають індексними палітрами.

Індексна палітра – це таблиця даних, у якій зберігається інформація про те, яким кодом закодований той або інший колір. Ця таблиця створюється і зберігається разом із графічним файлом.

Різні зображення можуть мати різні колірні палітри. Наприклад, в одному зображенні зелений колір може кодуватися індексом 64, а в іншому цей індекс може відповідати рожевому кольору. Якщо відтворити зображення з "чужою" колірною палітрою, то зелена ялинка на екрані може виявитися рожевою.

Зображення, які призначені для публікації в Інтернет, прийнято створювати в так званій *безпечній палітрі кольорів*. Вона є варіантом розглянутої вище індексної палітри. Але тому, що файли зображень в Web-графіці повинні мати мінімальний розмір, необхідно було відмовитися від введення в їхній склад індексної палітри. Для цього була прийнята єдина фіксована палітра кольорів, названа «безпечною», тобто така, що забезпечує правильне відображення кольорів на будь-яких пристроях (у програмах), що підтримують єдину палітру. Безпечна палітра містить усього 216 кольорів, що пов'язано з обмеженнями, які накладаються вимогами сумісності з комп'ютерами, що не відносяться до класу IBM PC.

Контрольні питання та завдання

1. Поняття кольору. Особливості хроматичних та ахроматичних кольорів.
2. Види колірних палітр. Глибина кольору.
3. Різниця між колірними палітрами HighColor і TrueColor.
4. Десяткове і шістнадцяткове подання кольору.
5. Закон Грассмана (закон змішування кольорів).
6. Колірні моделі. Види колірних моделей.
7. Колірні моделі RGB і CMYK (особливості, порівняння, сфери застосування).
8. Види перцепційних моделей, особливість і сфери застосування.
9. Колірні моделі HSB і HSL.
10. Колірна модель CI XYZ.
11. Перетворення колірних моделей CMYK в RGB, RGB в CMYK.
12. Переведення з RGB моделі в HSB, з HSB в RGB.

7.1 Графічна бібліотека OpenGL

Графічна бібліотека OpenGL (Open Graphics Library) – це базовий стандарт, що надає широкі можливості із програмування зображення реалістичних 3D-образів. Вона містить у собі більше ста функцій і процедур, що дозволяють розробнику визначати об'єкти і складні операції для створення високоякісних образів.

Графічний стандарт OpenGL розроблений і затверджений у 1992 році дев'ятьма фірмами, серед яких Digital Equipment Corporation, Hewlett-Packard Corporation, IBM Corporation, Silicon Corporation, Sun Microsystems Inc., Microsoft. В основу стандарту була покладена бібліотека IrisGL, розроблена Silicon Graphics.

Основними перевагами OpenGL є:

1) стабільність (доповнення і зміни в стандарті реалізуються в такий спосіб, щоб зберегти сумісність із розробленим раніше програмним забезпеченням, тобто нова версія OpenGL відбувається тільки за рахунок розширення);

2) надійність і мобільність (додатки гарантують однаковий візуальний результат незалежно від конкретної апаратної і програмної платформи, типу використовуваної операційної системи. Для перенесення додатків достатньо просто перекомпілювати вихідний текст на новій платформі);

3) OpenGL має продуману структуру, інтуїтивно зрозумілий інтерфейс, що дозволяє з меншими затратами створювати ефективні додатки, які містять менше рядків коду в порівнянні з використанням інших графічних бібліотек. Її драйвери містять інформацію про основне устаткування, звільняючи розробника від необхідності проектування для спеціальних особливостей графічних пристроїв.

OpenGL надає користувачу достатньо потужний низькорівневий набір команд, а всі операції високого рівня виконуються в термінах цих команд. Для полегшення роботи разом з OpenGL постачаються бібліотеки додаткових команд. На даний момент реалізація Microsoft OpenGL містить такі бібліотеки:

– **OpenGL** – набір базових функцій (бібліотека opengl32.dll) для створення об'єктів та управління їх відображенням на екрані. Імена базових команд починаються з префікса gl, а всі константи – з префікса GL. Кожне слово, що входить в ім'я функції, починається з великої літери, наприклад glPolygonMode;

– **Glu-бібліотеку** (бібліотека утиліт) є невід'ємною частиною стандарта і поставляється разом з головною бібліотекою OpenGL (бібліотека glu32.dll). Команд цієї бібліотеки доповнюють базові функції OpenGL і полегшують роботу програміста. До складу бібліотеки Glu увійшла

множина більш складних функцій, таких як реалізація куба, кулі, циліндра, диска, сплайнових кривих і поверхонь, реалізація додаткових операцій над матрицями тощо. Усі вони реалізовані через базові функції OpenGL. Імена команд бібліотеки утиліт починаються з префікса glu;

– **OpenGL Utility Toolkit** (GLUT) – бібліотека утиліт для додатків під OpenGL, яка в основному відповідає за системний рівень операцій введення-виведення при роботі з операційною системою. З функцій можна навести такі: створення вікна, керування вікном, моніторинг за введенням з клавіатури і подій миші. Вона також включає функції для рисування ряду геометричних примітивів: куб, сфера, чайник. GLUT навіть містить можливість створення нескладних висхідних меню.

– **Glaux-бібліотеку.** Бібліотека Glaux має широкий набір засобів взаємодії з користувачем, вона містить функції для управління вікнами (кількома командами можна визначити вікно, в якому буде працювати Windows), меню, кольорами, палітрою тощо. Крім цього, вона надає додаткові функції для побудови складних графічних об'єктів (конуса, тетраедра, тора тощо). Команди бібліотеки Glaux починаються з префікса aux.

7.2 Синтаксис команд OpenGL

Командами в OpenGL називаються функції чи процедури. Багато еквівалентних за діями команд можуть розрізнятися типами і кількістю аргументів. Для опису таких команд було введено такий синтаксис:

rtype ***CommandName [1 2 3 4] [b s i f d ub us ui] [v]****(atype arg)*

Команди OpenGL подаються за допомогою чотирьох компонент – ім'я і трьох символів:

- *rtype* визначає тип значення, що повертається і для кожної команди вказується в явному вигляді;
- *CommandName* – ім'я команди, таке, наприклад, як glRotate;
- *[1234]* – цифра, що показує число аргументів команди;
- *[b s i f d ub us ui]* – символи, що визначають тип аргументу. Допускаються такі типи аргументів:

Таблиця 7.1

Суфікс	Тип даних	Тип у С	Тип в OpenGL
1	2	3	4
b	8-бітове ціле	signed char	GLbyte
s	16-бітове ціле	short	GLshort
i	32-бітове ціле	int или long	GLint, GLsizei

Продовження таблиці 7.1

1	2	3	4
f	32-бітове число з плаваючою точкою	float	GLfloat, GLclampf
d	64-бітове число з плаваючою точкою	double	GLdouble, GLclampd
ud	8-бітове беззнакове ціле	unsigned char	GLubyte, GLboolean
us	16-бітове беззнакове ціле	unsigned short	GLushort
ui	32-бітове беззнакове ціле	unsigned long	GLuint, GLenum,

- *[v]* – буква, яка показує, що як аргумент використовується покажчик на масив значень;
- *atype* і *ard* визначаються типом і числом аргументів, відповідно.

Наприклад, команда визначення кольору glColor* може мати одну з нижчеперелічених форм запису:

glColor3d(1,1,1);
glColor3f(0.5,0.5,0.5);
glColor4b(1,1,1,0);

7.3 Ініціалізація OpenGL. Створення мінімальної програми

Типова програма, що використовує OpenGL, починається з визначення вікна, в якому відбуватиметься відображення. Потім створюється контекст (клієнт) OpenGL і асоціюється з цим вікном. Далі програміст може вільно використовувати команди і операції OpenGL API.

Перш за все необхідно навчитися найголовнішого – це створення вікна, в якому можна буде створювати об'єкти за допомогою OpenGL. Мінімальна програма складається з таких кроків:

1. Ініціалізація GLUT.
2. Встановлення параметрів вікна.
3. Створення вікна.
4. Встановлення функцій, що відповідають за рисування у вікні і зміну форми вікна.
5. Вхід до головного цикл GLUT.

Розглянемо всі 5 пунктів детальніше.

1. Ініціалізація GLUT проводиться командою:

*void glutInit (int * argcp, char ** argv);*

Перший параметр являє собою покажчик на кількість аргументів в командному рядку, а другий – покажчик на масив аргументів. Зазвичай ці

значення беруться з головної функції програми: int main (int argc, char * argv []).

2. Установлення параметрів вікна містить в собі декілька етапів. Перш за все, необхідно вказати розміри вікна:

void glutInitWindowSize (int width, int height);

Перший параметр width – ширина вікна в пікселях, другий height – висота вікна в пікселях. Якщо цю команду опустити, то GLUT сам встановить розміри вікна за замовчуванням, зазвичай це 300×300.

Далі можна задати положення вікна щодо верхнього лівого кута екрана за допомогою команди:

void glutInitWindowPosition (int x, int y);

Необхідно також встановити для вікна режим відображення інформації – встановити для вікна такі параметри, як: колірна модель, кількість різних буферів, і т. д. Для цього в GLUT існує команда:

void glutInitDisplayMode (unsigned int mode);

У команди є єдиний параметр, який може бути поданий однією з наступних констант або комбінацією цих констант за допомогою побітового АБО.

Таблиця 7.2

Константа	**Значення**
1	2
GLUT_RGB	Для відображення графічної інформації використовуються три компоненти кольору RGB.
GLUT_RGBA	Те ж, що і RGB, але використовується 4-ий компонент ALPHA (прозорість).
GLUT_INDEX	Колір задається не за допомогою RGB компонентів, а за допомогою палітри. Використовується для старих дисплеїв, де кількість кольорів, наприклад, 256.
GLUT_SINGLE	Виведення у вікно здійснюється з використанням одного буфера. Зазвичай використовується для статичного виведення інформації.
GLUT_DOUBLE	Виведення у вікно здійснюється з використанням двох буферів. Застосовується для анімації, щоб усунути ефект мерехтіння.
GLUT_ACCUM	Використовується буфер накопичення (Accumulation Buffer). Цей буфер застосовується для створення спеціальних ефектів, наприклад відображення і тіні.

Продовження таблиці 7.2

1	2
GLUT_ALPHA	Використовується буфер ALPHA для таких ефектів, як прозорість об'єктів.
GLUT_DEPTH	Створення буфера глибини. Цей буфер використовується для відсікання невидимих ліній в 3D просторі при виведенні на плоский екран монітора
GLUT_STENCIL	Буфер трафарету використовується для таких ефектів, як вирізання частини фігури, роблячи цей шматок прозорим.

Ось приклад використання цієї команди:

void glutInitDisplayMode (GLUT_RGB | GLUT_DOUBLE);

3. Створення вікна.

*int glutCreateWindow (const char * title);*

Ця команда створює вікно із заголовком, який вказується як параметр, і повертає HANDLER вікна у вигляді числа int. Цей HANDLER зазвичай використовується для подальших операцій над цим вікном, таких як зміна параметрів вікна і закриття вікна.

4. Установлення функцій, що відповідають за рисування у вікні і зміну форми вікна.

Після того як вікно, в яке буде виводитися графічна інформація, підготовлено і створено, необхідно пов'язати з ним процедури, які відповідатимуть за виведення графічної інформації, стежити за розмірами вікна, стежити за натисканнями на клавіші і т. д. Найперша і найнеобхідніша функція, яку ми розглянемо, відповідає за рисування. Саме вона завжди буде викликатися операційною системою, щоб нарисувати (перерисувати) вміст вікна. Отже, задається ця функція командою:

void glutDisplayFunc (void (func) (void));*

Єдиний параметр цієї функції – це покажчик на функцію, яка буде відповідати за рисування у вікні. Наприклад, щоб функція void Draw (void), визначена у вашій програмі, відповідала за рисування у вікні, треба приєднати її до GLUT таким чином: glutDisplayFunc (Draw);

Ще одна важлива функція – це функція, яка відстежує зміни вікна. Як тільки у вікна змінилися розміри, необхідно перебудувати виведення графічної інформації вже в нове вікно з іншими розмірами. Якщо цього не зробити, то, наприклад, збільшивши розміри вікна, виведення інформації буде проводитися в стару область вікна, з меншими розмірами. Визначити

функцію, відповідальну за зміну розмірів вікна, потрібно такою командою:

void glutReshapeFunc (void (func) (int width, int height));*

Єдиний параметр – це покажчик на функцію, відповідальну за зміну розмірів вікна, яка як видно повинна приймати два параметри width і height, відповідно ширину і висоту нового (зміненого) вікна.

5. Вхід до головного циклу GLUT. Цей цикл запускає на виконання так зване «серце» GLUT, яке забезпечує взаємозв'язок між операційною системою і тими функціями, які відповідають за вікно, отримують інформацію від пристроїв введення / виведення. Для того щоб перейти в головний цикл GLUT, треба виконати команду:

void glutMainLoop (void);

Нижче наведена найпростіша програма з використанням засобів бібліотеки GLUT, яка рисує у вікні зображення сфери.

Команда *glFlush ()* гарантує, що команда рисування буде виконана негайно, а не збережена в буфері.

Перш ніж використовувати функції з GLUT, необхідно до програми підключити заголовок: *#include <GL / glut.h>*.

Приклад 7.1. Виведення на екран зображення сфери

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
#include <GL/glut.h>
void resize(int width,int height)
{
}
void  display(void)
{
    glColor3d(1,1,0);
    glutSolidSphere(1.0, 25, 25);
       glFlush();
}
void init(void)
{
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_DEPTH_TEST);
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
```

```
  glOrtho(-5.0,5.0,-5.0,5.0,2.0,12.0);
  gluLookAt(0,0,5, 0,1,0, 0,1,0);
  glMatrixMode(GL_MODELVIEW);
}
int main(int argc,char ** argv)
{
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
 glutInitWindowPosition(50,10);
 glutInitWindowSize(400,400);
 glutCreateWindow(«Hello»);
 glutReshapeFunc(resize);
 init();
  glutDisplayFunc(display);
 glutMainLoop();
 return 0;
 }
```

7.4 Матриці і видові перетворення координат

Усі перетворення над координатами бібліотека OpenGL робить із використанням відповідних матриць перетворення (розмірністю 4×4). Передбачено три типи матриць – видова, проекцій і текстури. Перш ніж почати роботу з кожною з них, її необхідно зробити поточною. Команда:

void glMatrixMode(Glenum mode)

робить поточною задану параметром mode матрицю. При цьому даний параметр може приймати одне з таких значень:

- GL_MODELVIEW – поточною буде видова матриця;
- GL_PROJECTION – поточною буде матриця проекцій;
- GL_TEXTURE – поточною буде матриця текстури.

Наприклад, команда glMatrixMode(GL_MODELVIEW) робить поточною видову матрицю перетворення.

Перш, ніж використовувати матрицю, її потрібно ініціалізувати за допомогою команди:

void glLoadIdentity(void)

яка замінює поточну матрицю перетворення на одиничну (яку називають матрицею ідентичності).

При виконанні різних перетворень часто виникає необхідність збереження значень поточної матриці з можливим наступним їхнім відновленням. Це виконується за допомогою таких команд:

void glPushMatrix(void);
void glPopMatrix(void);

які запам'ятовують у стеці і відновлюють значення поточної матриці.

Бібліотека OpenGL дозволяє виконувати такі видові перетворення:

- обертання (glRotate*);
- перенесення (glTranslate*);
- масштабування (glScale*).

void glRotate[f d](GLtype angle, GLtype x, GLtype y, GLtype z)

здійснює поворот вектора проти часової стрілки на кут angle (у градусах) щодо точки (x, y, z). Наприклад, після виконання команди glRotatef(45, 0, 0, 1) усі об'єкти будуть зображені поверненими на кут 45° щодо осі Z.

void glTranslate[f d](GLtype x, GLtype y, GLtype z)

здійснює перенесення об'єкта на відстані x, y і z уздовж осей X, Y і Z, відповідно. Наприклад, команда glTranslatef(1, 0, 0) зрушує об'єкт на відстань 1 уздовж осі абсцис.

void glScale[f d](GLtype x, GLtype y, GLtype z)

здійснює масштабування об'єкта уздовж кожної з координатних осей на значення, які задані відповідними параметрами.

7.5 Задання області виведення і виду проекції

Область виведення визначається як прямокутник із заданими координатами верхнього лівого кута, а також шириною й висотою в пікселях. За замовчуванням, область виведення OpenGL збігається з клієнтською (робочою) частиною вікна, у яке виводиться зображення. Для її задання використовується спеціальна команда glViewport:

void glViewport(GLint x, Glint y, Glint width, Glint height)

Ця команда задає перетворення з нормалізованих координат у віконні. Причому, x і y – задають координати верхнього лівого кута області виведення, а width і height – відповідно її висоту й ширину.

Наприклад, команда glVievport(0,0, ClientWidth / 2, ClientHeight / 2) задає як область виведення ліву половину клієнтської частини вікна.

Після того як установлена область виведення, потрібно установити проекційну (картинну) площину й обсяг видимості. Для різних типів проекції це здійснюється по-різному.

Ортографічна проекція

Для задання ортографічної проекції необхідне виконання таких стандартних дій:

- задання області виведення;
- встановлення як поточної матриці проекцій і її ініціалізація;
- встановлення матриці перспектив та обсягу видимості.

Для виконання останнього етапу використовується команда

void glOrtho(Gldouble left, Gldouble right, Gldouble bottom, Gldouble top, Gldouble near, Gldouble far)

де параметри left і right визначають координати лівої і правої вертикальних, а bottom і top – нижньої й верхньої горизонтальних площин відсікання. Відстань до ближньої і дальньої площин відсікання визначається параметрами near і far.

Приклад 7.2. Задання ортографічної проекції

```
// Визначення області виведення
glViewport(0,0, ClientWidth, ClientHeight)
// Встановлення як поточної матриці проекцій
glMatrixMode(GL_PROJECTION);
// Її ініціалізація
glLoadIdentity();
glOrtho(-10, 10, -10, 10, 20, 50);
// Встановлення як поточної видової матриці
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// Поворот на 45 градусів навколо осі Y
glRotatef(45,0,1,0);
```

Перспективна проекція

Встановлення даного виду проекції є аналогічним ортографічній за винятком обсягу видимості – для цього типу проектування визначається усічений конус видимості. У OpenGL реалізована спеціальна команда:

void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble znear, GLdouble zfar)

яка створює матрицю перспективи. Параметри left і right визначають координати лівої і правої вертикальних, а bottom і top – нижньої і верхньої горизонтальних площин відсікання. Відстань до ближньої і дальньої площин відсікання по глибині визначається параметрами znear і zfar (які повинні завжди бути додатніми). Вважається, що спостерігач знаходиться в точці з координатами (0, 0, 0). Точність подання глибини залежить від значень, визначених у znear і zfar, чим більше відношення zfar / znear, тим менш ефективно будуть розрізнятися в буфері глибини, розташовані поруч поверхні.

Приклад 7.3. Задання перспективної проекції

```
// Визначення області виведення
glViewport(0,0, ClientWidth, ClientHeight)
// Встановлення як поточної матриці проекцій
glMatrixMode(GL_PROJECTION);
// Її ініціалізація
glLoadIdentity();
glFrustum(-2, 2, -2, 2, 2, 50);
// Встановлення як поточної видової матриці
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Оскільки перспективна проекція використовується частіше, у бібліотеку OpenGL введені команди *gluPerspective* і *gluLookAt*, що полегшують її завдання.

void gluPerspective(Gldouble angley, Gldouble aspect, Gldouble znear, Gldouble zfar)

визначає усічений конус видимості у видовій системі координат. Параметр angley задає кут видимості (у градусах) у напрямку осі Y. У напрямку X кут видимості задається через відношення aspect, що визначається співвідношенням сторін області виведення. Параметри znear, zfar визначають відстань від спостерігача до ближньої і дальньої площин відсікання, і їхні значення повинні бути завжди додатними.

void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz)

створює видову матрицю, що залежить від точки спостереження (дозволяє в явному вигляді задати положення спостерігача сцени). Параметри eyex, eyey, eyez задають координати точки спостереження, centerx, centery, centerz – центр сцени, upx, upy, upz – вектор «вверх», що визначає додатний напрямок осі Y сцени.

Приклад 7.4. Подання перспективної проекції і точки спостереження

```
// Визначення області виведення
glViewport(0,0, ClientWidth, ClientHeight);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
  // Подання конуса видимості
        gluPerspective(40.0, ClientWidth / ClientHeight, 0.1, 25);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
  // Подання точки спостереження і центра сцени
        gluLookAt(0, 0, 3, 0, 0, 0, 0, 1, 0);
```

7.6 Графічні примітиви OpenGL

Примітиви – це ті базові елементи, з яких будуються графічні об'єкти. До примітивів у OpenGL відносяться точки, лінії, багатокутники і бітові масиви.

Для задання примітива в OpenGL передбачені спеціальні командні дужки glBegin – glEnd, між якими і міститься група команд, що задає примітив:

void glBegin(GLenum mode);
void glEnd(void);

Параметр mode визначає тип примітива, що буде будуватися з описаних між командними дужками вершин.

Таблиця 7.3

Значення mode	**Опис**
GL_POINTS	Кожна вершина розглядається як окрема точка
GL_LINES	Кожна пара вершин розглядається як незалежний відрізок. Остання непарна вершина ігнорується.
GL_LINE_STRIP	Рисується зв'язна послідовність відрізків. Кінець попереднього відрізка є початком наступного.
GL_LINE_LOOP	Аналогічно попередньому за винятком того, що остання вершина з'єднується з першою (замкнута послідовність відрізків)
GL_TRIANGLES	Кожна трійка вершин розглядається як незалежний трикутник. Зайві вершини (одна чи дві ігноруються).
GL_TRIANGLE_STRIP	Рисується група зв'язних трикутників, що мають загальну грань.
GL_TRIANGLE_FAN	Рисується група зв'язних трикутників. Перші три вершини – перший трикутник. Перша, третя і четверта – другий і т. д.
GL_QUADS	Кожна група з чотирьох вершин розглядається як незалежний чотирикутник. Зайві вершини ігноруються.
GL_QUAD_STRIP	Рисується група зв'язних чотирикутників (аналогічно GL_TRIANGLE_STRIP)
GL_POLYGON	Рисується окремий опуклий багатокутник, заданий усіма вершинами.

Точки

Як уже відзначалося раніше, вершина -- це точка в тривимірному просторі. Для її визначення в бібліотеці реалізована спеціальна команда glVertex*:

void glVertex[234] [v] (type coord)

Виклик будь-якої команди glVertex* визначається чотирма аргументами: x, y, z, w, що задають однорідні координати вершини. При цьому дотримуються угоди:

glVertex2* – задає координати x і y, z = 0,w = 1;

glVertex3* – задає координати x, y, z, а w = 1;

glVertex4* – задає всі чотири координати x, y, z і w.

Для задання товщини точки в OpenGL є спеціальна команда

void glPointSize(GLfloat size)

де параметр size задає значення товщини (за замовчуванням приймається 1.0).

Приклад 7.5. Зображення червоної точки в просторі товщиною 2.5

```
glPointSize(2.5);
glColor3b(1,0,0);
glBegin(GL_POINTS);
        glVertex3d(1,1,1);
glEnd();
```

Лінії

Для задання ширини ліній використовується команда

void glLineWidth(GLfloat width);

цілком аналогічна відповідній команді для точок.

Приклад 7.6. Зображення двох не зв'язаних між собою відрізків різного кольору

```
glLineWidth(2.0);
glBegin(GL_LINES);
  glColor3f(0.5,0.5,1);
  glVertex2f(-1,0);
  glVertex2f(1,0);
  glColor3f(1,0.5,0);
  glVertex2f(0,-1);
  glVertex2f(0,1);
glEnd();
```

Багатокутники

Для демонстрації використання багатокутних примітивів розглянемо такі приклади.

Приклад 7.7. Зображення різнобарвного трикутника

```
// Включаємо режим плавного сполучення кольорів
glEnable(GL_SMOOTH);
// Задаємо координати вершин і їхнього кольору
glBegin(GL_TRIANGLES);
  glColor3b(1,0,0);
  glVertex2f(0,0);
  glColor3b(0,1,0);
  glVertex2f(1,0);
  glColor3b(0,0,1);
  glVertex2f(1,1);
glEnd();
```

Приклад 7.8. Зображення двох білих суміжних чотирикутників

```
glBegin(GL_QUAD_STRIP);
  glVertex3f(0,0,0);
  glVertex3f(1,0,0);
  glVertex3f(1,1,0);
  glVertex3f(0,1,0);
  glVertex3f(1,1,1);
  glVertex3f(0,1,1);
glEnd();
```

Приклад 7.9. ***Зображення сірого п'ятикутника***

```
glColor3f(0.5,0.5,0.5);
  glBegin(GL_POLYGON);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);
    glVertex3f(1,1,0);
    glVertex3f(0,1,0);
    glVertex3f(-1,0,0);
  glEnd();
```

Растрові примітиви

Для зображення растрових примітивів використовується команда:

*void glDrawPixels(GLsizei width, GLsizei height, GLenum format, GLenum type, const GLvoid *pixels)*

де параметри width і height задають висоту і ширину бітового масиву в пікселях, format – визначає спосіб збереження бітів у масиві (GL_RGBA – найчастіше використовуваний формат), type – тип елементів бітового масиву (наприклад, GL_UNSIGNED_BYTE) і pixels – вказує адресу, починаючи з якої, зберігається бітовий образ.

При відображенні растровий примітив виводиться відносно поточної позиції растра, що задається командою

void glRasterPos[234] [v] (type coord)
де coord – координати початкової позиції растра.

Приклад 7.10. Зображення растрового примітива
```
// Задання елементів бітового масиву
GLubyte bmp[] = {
                    0x00, 0x00, 0x00, 0x00,
                    0x00, 0xFF, 0xFF, 0x00,
                    0x00, 0xFF, 0xFF, 0x00,
                    0x00, 0x00, 0x00, 0x00,
                  };
        // Задання початкової позиції растра
glRasterPos2i(0,0);
// Виведення растрового примітива
        glDrawPixels(4,4,GL_RGBA,GL_UNSIGNED_BYTE, bmp);
```

Крім описаних базових, бібліотека OpenGL містить ряд додаткових квадратичних примітивів, таких як циліндр, сфера, диск, сектор і т. п. Для їхнього задання використовуються такі команди, як: gluCylinder, gluSphere, gluDisk, gluPartialDisk, gluNewQuadric і т. п.

Перш ніж використовувати такі примітиви, їх потрібно створити за допомогою команди

GLUquadricObj gluNewQuadric(void)*

яка повертає покажчик на квадратичний примітив. Потім покажчик використовується для створення безпосередньо конкретної фігури.

Приклад 7.11. Зображення сфери
```
// Оголошення покажчика на квадратичний примітив
GLUquadricObj* quadObj;
// Створення примітива
quadObj = gluNewQuadric();
// Зображення сфери радіусом 1.5, що складається з 16 меридіанів і 16 паралелей
gluSphere (quadObj, 1.5, 16, 16);
// Видалення об'єкта й очищення пам'яті
gluDeleteQuadric(quadObj);
```

7.7 Колір та освітлення в OpenGL

У OpenGL використовується колірна модель RGB. Однак для задання конкретного кольору допускається використання четвертого компонента – так званої альфи, що визначає ступінь прозорості кольору. Така колірна

схема називається RGBA. Причому альфа може змінюватися від 0.0 (повна прозорість) до 1.0 (повна непрозорість, значення за замовчуванням).

Для роботи в режимі RGBA OpenGL надає таку команду:

void glColor[3 4][b s i f d][v](GLtype components)

яка задає інтенсивність компонентів кожного базового кольору (і альфи при необхідності). Наприклад, для задання жовтого кольору необхідно скористатися командою glColor3b(1, 1, 0).

Після того, як деякий колір установлений як поточний, він поширюється на всі створювані після цього об'єкти. При рисуванні суцільних примітивів (наприклад, багатокутників) допускається використання плавного сполучення кольорів між різнобарвними вершинами. Даний режим вмикається командою glEnable(GL_SMOOTH).

При візуальному моделюванні реалістичних образів із використанням графічної бібліотеки OpenGL важливим моментом є відтворення властивостей поверхні об'єктів, що рисуються. Тип поверхні об'єкта сильно впливає на його візуальне сприйняття. Основними параметрами, що впливають на це сприйняття, є:

- матеріал, із якого зроблений об'єкт;
- ступінь шорсткості його поверхні;
- форма об'єкта (радіус кривизни);
- положення і кількість джерел світла.

Нормалі

Для задання форми об'єкта використовується поняття нормалі. *Нормаллю* називається вектор, що однозначно визначає орієнтацію грані в просторі.

Вектор нормалі асоціюється з кожною вершиною і у явному вигляді задається за допомогою команди:

void glNormal3[b s i f d][v](type coords),

де coords – компоненти вектора нормалі. Фактично за допомогою даної команди можна задати локальний радіус кривизни в кожній вершині примітива, що рисується.

Приклад 7.12. Зображення трикутника із заданням нормалей

```
// Задаємо координати вершин і їхні нормалі
glBegin(GL_TRIANGLES);
  glNormal3b(1,0,0);
  glVertex3f(0,0,0);
  glNormal3b(0,1,0);
  glVertex3f(1,0,0);
  glNormal3b(0,0,1);
  glVertex3f(1,1,1);
glEnd();
```

Властивості матеріалів

Різні матеріали по-різному відбивають світло, а деякі його навіть випромінюють. Для задання властивостей матеріалу в OpenGL є спеціальна команда

void glMaterial[i f][v](GLenum face, GLenum pname, GLfloat param);

Параметри матеріалу в загальному випадку розрізняються для лицьової і зворотної поверхні. Перший аргумент face визначає, до якої поверхні будуть застосовуватися параметри, що задаються іншими аргументами:

- GL_FRONT – лицьові поверхні;
- GL_BACK – зворотні поверхні;
- GL_FRONT_AND_BACK– обидві поверхні.

Аргумент pname визначає, які параметри матеріалу будуть задаватися за допомогою третього аргументу params, і може приймати такі значення:

Таблиця 7.4

Значення pname	Відповідні значення params
GL_AMBIENT	params містить 4 цілі чи дійсні значення кольорів RGBA, що визначають розсіяний колір матеріалу. За замовчуванням – (0.2,0.2,0.2,1.0)
GL_DIFFUSE	params містить значення, що визначають колір дифузійного відображення матеріалу. За замовчуванням – (0.8,0.8,0.8,1.0)
GL_SPECULAR	params містить значення, що визначають колір дзеркального відображення. За замовчуванням – (0.0,0.0,0.0,1.0)
GL_EMISSION	params містить значення, що визначають інтенсивність випромінюваного світла матеріалу. За замовчуванням – (0.0,0.0,0.0,1.0)
GL_SHININESS	params містить значення, що визначають ступінь дзеркального відображення матеріалу. Допускаються значення від 0 до 128. За замовчуванням – 0.
GL_AMBIENT_AND_DIFFUSE	еквівалентне двом викликам команди з аргументом pname, рівними GL_AMBIENT і GL_DIFFUSE, і однаковими значеннями параметра params
GL_COLOR_INDEXES	параметр params містить 3 цілі чи речовинні значення, що визначають індекси кольорів для розсіяного, дифузійного і дзеркального відображення, відповідно

Джерело світла

Для визначення параметрів джерела світла використовується команда:

void glLight[i f][v](GLenum light, GLenum pname, GLfloat param)

Перший аргумент light вказує номер джерела світла, для якого задаються параметри (може бути до 8 джерел світла). Як і у випадку з командою glMaterial*, значення другого і третього параметрів пов'язані між собою таким способом:

Таблиця 7.5

Значення pname	**Відповідне значення params**
GL_AMBIENT	params містить 4 цілі чи дійсні значення кольорів RGBA, що визначають інтенсивність фонового висвітлення. За замовчуванням – (0.0,0.0,0.0,1.0)
GL_DIFFUSE	params містить 4 цілі чи дійсні значення RGBA, що визначають інтенсивність дифузійного висвітлення. За замовчуванням – (1.0,1.0,1.0,1.0) для нульового джерела, і (0.0,0.0,0.0,1.0) – для всіх інших
GL_SPECULAR	params містить 4 цілі чи дійсні значення квітів RGBA, що визначають інтенсивність дзеркального відображення. За замовчуванням – (1.0,1.0,1.0,1.0) для нульового джерела, і (0.0,0.0,0.0,1.0) – для всіх інших
GL_POSITION	params містить 4 цілі чи дійсні значення, що задають положення джерела світла в однорідних світових координатах. За замовчуванням джерело розташовується в кращі (0.0, 0.0, 1.0, 0.0)
GL_SPOT_DIRECTION	params містить 3 цілі чи дійсні значення, що задають напрямок світла у світових координатах. За замовчуванням (0.0, 0.0, -1.0)

Після того, як задані ці параметри, джерела світла можна вмикати і вимикати в будь-який час виконання програми. Джерело світла n включається / виключається командами glEnable / glDisable з аргументом GL_LIGHTn (n = 0, 1, 2, …, 8).

Приклад 7.13. Задання властивостей матеріалу і джерела світла

```
  // Вказуємо параметри матеріалу і координати джерела світла
GLfloat emission[4] = {0.0, 0.0, 0.0, 1.0},
                      ambient[4]  = {0.2, 0.2, 0.0, 1.0},
                      diffuse[4]  = {1.0, 1.0, 0.0, 1.0},
                      specular[4] = {1.0, 1.0, 1.0, 1.0},
                      shiness = 70.0,
        lpt[] = { -6, 20, 18 };
// Задаємо параметри матеріалу
glMaterialfv(GL_FRONT, GL_EMISSION, emission);
glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);
          glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
```

```
        glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
        glMaterialfv(GL_FRONT, GL_SHININESS, &shiness);
    // Включаємо нульове джерело світла і задаємо його положення
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_POSITION, lpt);
```

7.8 Взаємодія з користувачем та анімація в OpenGL

Функції бібліотеки GLUT реалізують так званий подієво-керований механізм. Це означає, що є певний внутрішній цикл, який запускається після відповідної ініціалізації та обробляє одну за одною всі події, оголошені під час ініціалізації. До подій відносяться: клацання миші, закриття вікна, зміна властивостей вікна, пересування курсора, натискання клавіші, «порожня» (idle) подія, коли нічого не відбувається. Для проведення періодичної перевірки здійснення тієї чи іншої події потрібно зареєструвати функцію, яка буде його обробляти. Для цього використовуються функції вигляду:

void glutDisplayFunc (void (func) (void))*
void glutReshapeFunc (void (func) (int width, int height))*
void glutMouseFunc (void (func) (int button, int state, int x, int y))*
void glutIdleFunc (void (func) (void))*
void glutKeyboardFunc (void (func) (unsigned int key, int x, int y)*
void glutMotionFunc (void (func) (int x, int y))*

void glutKeyboardFunc (void (func) (unsigned int key, int x, int y)*
визначає функцію (func), яка викликається, коли натиснута клавіша на клавіатурі. Повертаються параметри: key – згенерований клавіатурою ASCII код; x, y – координати положення миші в координатах вікна в момент, коли була натиснута кнопка на клавіатурі.

void glutMouseFunc (void (func) (int button, int state, int x, int y))*
визначає функцію (func), яка викликається, коли кнопка миші натиснута або відпущена. Параметр button, що повертається функцією, може приймати значення GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON або GLUT_RIGHT_BUTTON. Значення параметра state є GLUT_UP або GLUT_DOWN залежно від того, чи була кнопка миші натиснута або відпущена, x and y параметри вказують на координати в поточному вікні, де знаходилася миша в момент натискання або відпускання кнопки.

void glutMotionFunc (void (func) (int x, int y))*
визначає функцію (func), яка викликається, коли покажчик миші переміщається в межах вікна при натиснутій одній або більше кнопок, x та

у параметри вказують на координати в поточному вікні, де знаходилася миша в момент початку події void glutPostRedisplay (void).

Якщо вміст буфера кадру змінюється в процесі регенерації зображення, то глядач може побачити зовсім небажані ефекти, наприклад, тремтіння картинки.

Цю проблему можна вирішити з використанням ***подвійної буферизації*** – стандартної технології організації комп'ютерної анімації. У цьому випадку використовуються два буфери кадру, які прийнято називати робочим і фоновим. Робочий буфер – це той, з якого виконується регенерація зображення на екрані, а у фоновому буфері зображення формується програмою. Механізм подвійної буферизації встановлюється в процесі ініціалізації аргументом функції *glutInitDisplayMode ()*. Замість константи GLUT_SINGLE потрібно задати константу GLUT_DOUBLE.

Перемикання буферів виконується функцією *glutSwapBuffers ()*. Всі оператори формування зображення входять в функцію display (), але при використанні подвійної буферизації в цій функції спочатку потрібно очистити робочий буфер, викликавши команду *glClear ()*, а останнім оператором викликати функцію перемикання буферів *glutSwapBuffers ()*.

Приклад 7.14. Рисування на екрані обертового квадрата.

```
#include <GL/glut.h>
#include <stdlib.h>
static GLfloat spin = 0.0;
void spinDisplay(void)
{
   spin = spin + 2.0;
  if (spin > 360.0)
     spin = spin – 360.0;
  glutPostRedisplay();
}
void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glPushMatrix();
  glRotatef(spin, 0.0, 0.0, 1.0);
  glColor3f(1.0, 1.0, 1.0);
  glBegin(GL_POLYGON);
   glVertex2f(-25.0,-25.0);
   glVertex2f(25.0,-25.0);
   glVertex2f(25.0,25.0);
   glVertex2f(-25.0,25.0);
   glEnd( );
  glPopMatrix();
  glutSwapBuffers();
}
void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}
void reshape(int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
```

```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (250, 250);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutIdleFunc(spinDisplay);
  glutMainLoop();
  return 0;
}
```

Контрольні питання та завдання

1. Яке призначення графічної бібліотеки OpenGl ?
2. Поясніть синтаксис команд OpenGL.
3. Назвіть категорії команд (функцій) бібліотеки.
4. Навіщо потрібні різні варіанти команд OpenGL, що відрізняються тільки типами параметрів?
5. Що таке примітив в OpenGL?
6. Що таке атрибут? Перелічіть відомі вам атрибути вершин в OpenGL.
7. Які системи координат використовуються в OpenGL?
8. Перелічіть види матричних перетворень в OpenGL. Яким чином відбуваються перетворення об'єктів в OpenGL?
9. Що таке матричний стек?
10. Перелічіть способи зміни положення спостерігача в OpenGL.
11. Яка послідовність викликів команд glTranslate (), glRotate () і glScale () відповідає команді gluLookAt (0, 0, -10, 10, 0, 0, 0, -1, 0)?
12. Які стандартні команди для задання проекцій ви знаєте?
13. Що таке видові координати?
14. Для чого потрібна команда glColorMaterial?
15. Як задати положення джерела світла таким чином, щоб він завжди знаходився в точці положення спостерігача?
16. Як задати фіксоване положення джерела світла? Чи можна задавати положення джерела відносно локальних координат об'єкта?

ЛІТЕРАТУРА

1. Блинова Т. А. Компьютерная графика / Т. А. Блинова, В. Н. Порев. – К. : Издательство Юниор, 2006. – 520 с.
2. Божко А. Н. Компьютерная графика: учеб. пособие для вузов. / Божко А. Н., Жук Д. М., Маничев В. Б. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2007. – 392 с.
3. Вельтмандер П. В. Машинная графика : учеб. пособие в 3-х книгах. Книга 2. Основные алгоритмы / Вельтмандер П. В. – Новосиб. ун-т: Новосибирск, 1997. – 193 с.
4. Гайдуков С. А. OpenGL. Профессиональное программирование трехмерной графики на C++ / Гайдуков С. А. – СПб. : Питер, 2004. – 716 с.
5. Эгрон Д. Синтез изображений. Базовые алгоритмы / Эгрон Д. – М. : Радио и связь, 1993. – 216 с.
6. Эйнджел Э. Интерактивная компьютерная графика. Вводный курс на базе OpenGL, 2 изд. Пер. с англ. / Эйнджел Э. – Москва : «Вильямс», 2001. – 592 с.:ил.
7. Заргарян Ю. А. Компьютерная графика в практических приложених : учебное пособие для вузов. / Ю. А. Заргарян, Е. В. Заргарян. – Таганрог: Изд-во Технологического института ЮФУ, 2009. – 255 с.
8. Мосин В. Г. Математические основы компьютерной графики: Монография. / Мосин В. Г. – Самара : СГАСУ, 2005. – 227 с.
9. Пантюхин П. Я. Компьютерная графика. В 2 частях. / Пантюхин П. Я., Быков А. В., Репинская А. В. – М. : Издательство Инфра-М, 2007. – 288 с
10. Петров М. Н. Компьютерная графика. Учебник. / М. Н. Петров, В. П. Молочков. – Издательство Питер, 2004. – 812 с.
11. Поляков А. Ю. Методы и алгоритмы компьютерной графики в примерах на Visual C++, 2-е изд., перераб. И доп. / А. Ю. Поляков, В. А. Брусенцев. – СПб. : ЮХВ-Петербург, 2003. – 560 с.
12. Порев В. Н. Компьютерная графика / Порев В. Н. – Спб. : БХВ-Петербург, 2002. – 432 с.
13. Постнов К. В. Компьютерная графика / Постнов К. В. – Москва, 2009. – 247 с.
14. Роджерс Д. Алгоритмы машинной графики/ Роджерс Д. – М. : Мир, 1989. – 230 с.
15. Романюк О. Н. Комп'ютерна графіка / Романюк О. Н. – Вінниця : ВНТУ, 2001. – 200 с.
16. Тихомиров Ю. В. OpenGL. Программирование трехмерной графики / Тихомиров Ю. В. – СПб. : БХВ-Петербург, 2002. – 304 с.
17. Фоли Дж. Основы интерактивной машинной графики / Дж. Фоли, А. вэн Дэм. – М. : Мир, 1985, том2. – 287 с.

18. Хери Я. Компьютерная графика. Пер. с англ. / Я. Хери, Х. Бейкер. – М. : Вильямс, 2005. – 365 с.
19. Херн Д. Компьютерная графика и стандарты OpenGL, 3-е издание.: Пер. С англ. / Д. Херн, М. П. Бейкер. – М. : «Вильямс», 2005. – 1168 с.
20. Хилл Ф. OpenGL. Программирование компьютерной графики. Для профессионалов / Хилл Ф. – Спб. : Питер, 2002. –1088 с. ил.
21. Шикин А. В. Компьютерная графика. Полигональные модели / А. В. Шикин, А. В. Боресков. – Москва : ДИАЛОГ-МИФИ, 2001. – 464 с.

Навчальне видання

Коцюбинський Володимир Юрійович
Софина Ольга Юріївна
Мельник Людмила Миколаївна

КОМП'ЮТЕРНА ГРАФІКА

Навчальний посібник

Редактор В. Дружиніна
Коректор З. Поліщук
Оригінал-макет підготовлено О. Софиною

Підписано до друку 20.11.2015 р.
Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний. Ум. друк. арк. 9,8.
Наклад 75 пр. Зам. № 2015-132.

Вінницький національний технічний університет,
навчально-методичний відділ ВНТУ.
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, к. 2201.
Тел. (0432) 59-87-36.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.

Віддруковано у Вінницькому національному технічному університеті
в комп'ютерному інформаційно-видавничому центрі
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, ГНК, к. 114.
Тел. (0432) 59-87-38.
publish.vntu.edu.ua; email: kivc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.