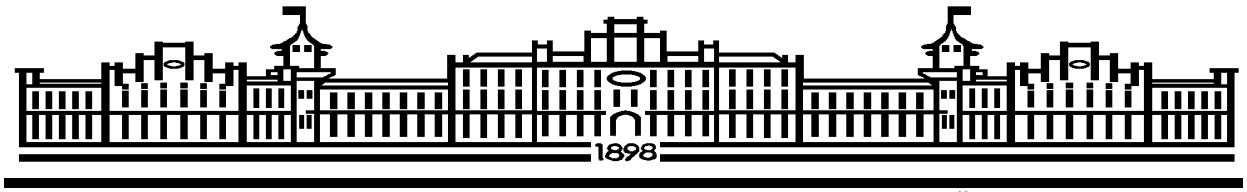


МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Кафедра обчислювальної техніки

КОНСПЕКТ ЛЕКЦІЙ

з програмного модулю
"Комп'ютерна графіка"

Розробник: старший викладач Саверченко В.Г.
(посада, вчена ступінь та звання П.І.Б.)

Затверджено на засіданні кафедри
Протокол № 1 від 30 серпня 2017 р.

Завідувач кафедри ОТ
_____ Стіренко С.Г.
(підпис) (прізвище, ініціали)

Київ – 2017.

ЗМІСТ

Вступ. 4
1. Візуалізація зображень. 6
2. Особливості програмування графіки на мові PASCAL. 7
3. Перетворення координат. 12
4. Перетворення на площині (2 D - перетворення). 13
5. Перетворення в просторі (3 D - перетворення) 15
6. Вилучення невидимих ліній і поверхонь. 19
7. Растрова візуалізація. 20
8. Фізична модель світла. 25
9. Властивості світла. 27
10. Методи заповнення. 30
11. Методи зафарбування. 33
12. Метод Гуро. 34
13. Метод Фонга. 35
14. Моделювання текстур і інших елементів реалістичних зображень. 37
15. Проективні текстури (projective texture). 38
16. Принцип дії відеосистеми ПК. 41
17. Пристрої вводу/виводу графічної інформації. 43
18. Робота відеосистеми. 46
Література. 49

ВСТУП

Поняття комп'ютерної графіки

Швидкий розвиток обчислювальних засобів, розширення їхніх можливостей є головними факторами усе більш широкого впровадження ПК в різні сфери наукової і практичної діяльності. Винятково інтенсивно розвивається напрямок комп'ютерної графіки.

Комп'ютерною графікою (КГ) називається наукова дисципліна про математичне моделювання геометричних форм і вигляду об'єктів, а також методів ***візуалізації*** різної інформації за допомогою технічних і програмних засобів комп'ютера.

Зорове сприйняття інформації для людини відіграє виняткову роль. Ємність і швидкість сприйняття зорових образів дуже великі.

Розвиток КГ почалося з появою пристроїв графічного висновку - дисплеїв (пристроїв побудови зображень на ЕЛТ). Термін «інтерактивна» КГ визначає, що можна вести діалог і впливати на формування зображення.

Місце комп'ютерної графіки

При обробці інформації, пов'язаної з зображенням на моніторі, прийнято виділяти три основних напрямки: машинну графіку, розпізнавання образів і обробку зображень.

Комп'ютерна (машинна) графіка відтворює зображення у випадку, коли вихідної є інформація необразотворчої природи. Наприклад, візуалізація експериментальних даних у виді графіків, гістограм чи діаграм, вивод інформації на екран у комп'ютерних іграх, синтез сцен для тренажерів.

А ще є комп'ютерний живопис, комп'ютерна анімація і так далі, аж до віртуальної реальності.

Можна сказати, що комп'ютерна графіка малює, спираючись на формальні правила і наявний набір засобів.

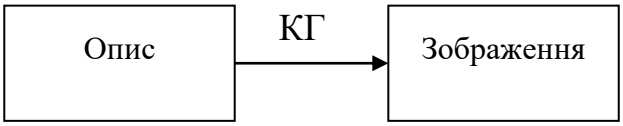


Рис. 1. Система комп’ютерної графики

Символічно систему комп'ютерної графіки КГ можна представити в такий спосіб (рис. 1)

- символьний опис;
- зображення (синтезоване зображення).

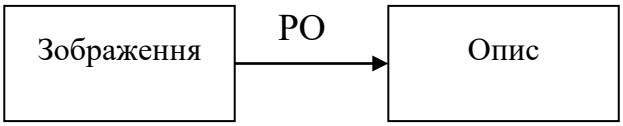


Рис. 2. Система розпізнавання зображень

Основна задача розпізнавання образів складається в перетворенні вже наявного зображення на формально зрозумілу мову символів. Розпізнавання образів (зображень) РЗ є сукупність методів, що дозволяють одержати опис зображення, поданого на вхід, або віднести задане зображення до деякого класу (так надходять, наприклад, при сортуванні чи пошти в медичній діагностиці, де шляхом аналізу томограм оцінюють наявність відхилень від норми). При цьому розглянуте зображення часто перетвориться в більш абстрактний опис - набір чисел, набір символів або у граф. Однієї з цікавих задач розпізнавання образів є так називана скелетизація об'єктів, при якій відновлюється деяка основа об'єкта, його "кістяк".

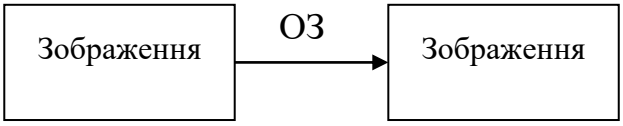


Рис. 3. Система обработки зображень

Система розпізнавання зображень має наступну структуру (рис. 2):

- зображення;
- опис.

Таким чином, задачі КГ – це задачі синтезу зображень.

Обробка зображень ОЗ розглядає задачі, у яких і вхідні і вихідні дані є зображеннями. Прикладами обробки зображень можуть служити: передача зображень разом з усуненням шумів і стиском даних, перехід від одного виду зображення (напівтонового) до іншого (каркасному), контрастування різних знімків, а також синтезування наявних зображень у нові, наприклад, за набором поперечних перерізів об'єкта побудувати поздовжні перетини або відновити сам об'єкт.

Система обробки зображень має наступну структуру (рис. 3):

- зображення;
- зображення (перетворене зображення)

Таким чином, задачі КГ - це задачі синтезу зображень, задачі РО - це задачі аналізу зображень, і а задачі ОЗ - це задачі перетворення зображень.

Однак при обробці графічної інформації представляється скрутним чітко визначити коло задач, що відносяться до конкретної області., оскільки вони взаємозалежні. Тому зручно намалювати загальну схему, що вміщає в себе опис і функції РИ, ОЗ і КГ, тим більше, що різких границь між ними провести не можна (рис. 4).

Історія і перспективи розвитку КГ

Початком сучасної КГ можна вважати створення Сазерлендом (1963 р.) першого спеціалізованого пакета програмного забезпечення машинної графіки.

У 60-і рр. були сформульовані принципи малювання відрізками, видалення невидимих ліній, методи відображення складних поверхонь, формування тіней і обліку освітленості. Перші роботи були в основному спрямовані на розвиток векторної графіки, тобто малювання відрізками. Найбільший вплив на розвиток векторної графіки зробили алгоритми Брезенхема.

У 70-і роки одержала подальший розвиток 3-х мірна КГ (3 D), тобто відображення просторових форм і об'єктів з урахуванням розташування джерел висвітлення і спостереження. З'явилася велика кількість робіт з методів апроксимації і представлення складних поверхонь, генеруванню текстур і рельєфу, моделюванню умов освітленості.

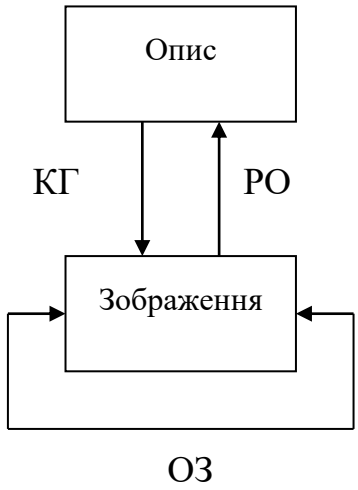


Рис. 4. Система обработки графичнкої інформації

У 80-і роки поява персонального комп'ютера (ПК) сприяла розширенню сфер застосування КГ. Причому КГ стали займатися не тільки програмісти, але і фахівці інших галузей, не зв'язаних із програмуванням. Цьому сприяло збільшення швидкості обробки інформації, збільшення обсягу пам'яті, а головне - різноманітність програмних продуктів з КГ. У цей час одержали подальший розвиток принципи і методи формування реалістичних зображень, тобто зображень, де забезпечується вся сукупність образотворчих засобів: об'ємність, взаємне розташування предметів, півтону, світло, текстура поверхні.

Подальший розвиток КГ пов'язаний з відображенням динамічних сюжетів, у яких здійснюється швидка зміна зображень. При цьому підвищення продуктивності і збільшення обсягу пам'яті ПК створюють нові передумови для подальшого розвитку КГ.

Основні області застосування комп'ютерної графіки

КГ широко використовується в системах автоматизації проектування і наукового експерименту (графіки, креслення, архітектура, дизайн), у геоінформаційних системах (візуалізація об'єктів на поверхні Землі) на телебаченні (рекламні ролики), у рекламі (спецефекти), у відеофільмах (відєоефекти), у комп'ютерних іграх, тренажерах створюючи віртуальну реальність, використання КГ в Інтернету і т.д..

Графічні стандарти

CGA (Color Graphics Adapter) - початок 80 років.
MCGA (Multicolor Graphics Array) - 1987 р. 320*200 256 чи квітів 640*480 монохром.
HGC (Hercules Graphics Card) - перший монохромний графічний стандарт.
EGA (Enhanced Graphics Adapter) - 1985 р. , розроблений фірмою IBM
PGA (Professional Graphics Adapter чи EGA-Plus
VGA (Video Graphics Array) - основний графічний стандарт
SVGA (Super VGA)
XGA (Extend Graphics Adapter)

Стандарт	Здатність	Кількість кольорів	частота рядків, кГц	частота кадрів, Гц
CGA	320*200	4 из16	15,75	60
Hercules	720*348	монохром	18,43	50
EGA	640*350	16 из 64	21,85	60
PGA	640*480	16 из 256	25,00	60
VGA	640*480 320*200	16 из 256 256 из 262144(2^{18})	31,50	70

1. Візуалізація зображень

Однією з основних задач обробки зображень за допомогою комп'ютера є візуалізація, тобто безпосереднє створення зображень. При цьому дані можуть бути інформацією образотворчої, так і необразотворчої природи, що пов'язано зі специфікою представлення конкретного зображення. Відомо, що близько 50 % нейронів людського мозку зв'язано з обробкою візуальної інформації. Тому візуалізація повинна забезпечувати ефективну взаємодію людини з комп'ютером.

Зображення реальних об'єктів у комп'ютері можуть формуватися чи прямо побічно з різних скануючих пристроїв у виді двовимірного масиву значень яскравості або кольору відповідних точок екрана дисплея. Крім того, можуть створюватися і штучні зображення об'єктів, що існують як абстрактні сукупності деяких графічних примітивів (точок, прямих ліній, окружностей і т.д.) у пам'яті комп'ютера. Після візуалізації і сприйняття зображень користувач має можливість динамічно змінювати зображення, що характерно для інтерактивної комп'ютерної візуалізації (рис.5).

Розрізняють основні два принципи візуалізації: векторний і растровий.

Векторний принцип формування зображень заснований на обході контуру відтвореної фігури. Примітиви, що складають зображення, кодуються командами, у яких використовуються координати кінцевих точок векторів. Регенерація зображення здійснюється шляхом багаторазового повторення програми відтворення чергового кадру, примітив за примітивом, оскільки електронний промінь може рухатися по екрані дисплея довільним образом.

При растровому принципі формування зображень кодування істотне простіше: примітиви, що малюються на екрані, розбиваються на складові піксели, утворюють матрицю станів вузлів координатної сітки. У растровому дисплеї електронний промінь рухається по фіксованій траєкторії (по рядках растра, а не по примітивах, як при векторному принципі), здійснюючи сканування точок координатної сітки, переходячи з однієї горизонталі на наступну.

Якщо при реалізації зображення його опис не адекватний використовуваному принципу візуалізації, то необхідно здійснити конвертацію: растерізацію (перетворення з векторного в растровий опис) чи векторізацію (перетворення з растрового у векторний опис). Оскільки найбільше поширення одержали растрові пристрої, то необхідність растерізація зображення виникає частіше, ніж векторізація.

У цьому зв'язку особлива увага приділяється методам і алгоритмам растрової візуалізації, що лежить

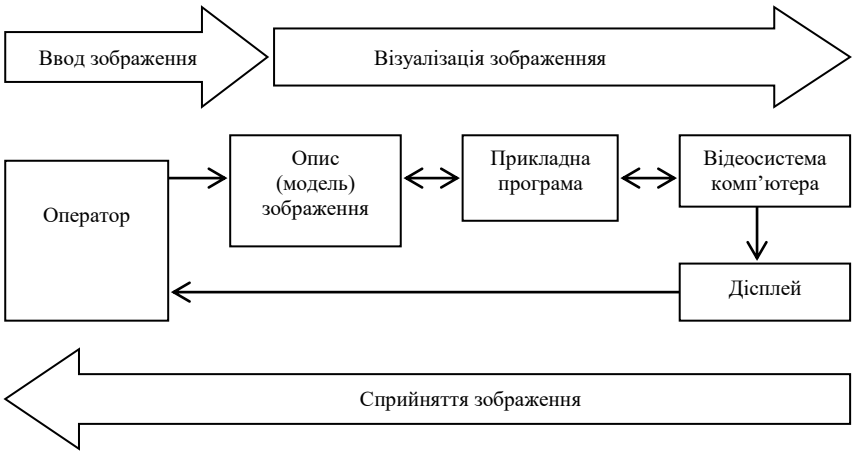


Рис. 5. Модель інтерактивної візуалізації

в основі сучасних систем обробки зображень. При цьому варто виділити базові растрові алгоритми, такі як малювання відрізка та кола, відсікання і заповнення області.

Візуалізації різних двовимірних і тривимірних об'єктів припускає виконання цілого ряду перетворень з метою найбільш повного представлення про їхню форму і про взаємне розташування. При цьому слід зазначити афінні перетворення на площині й у просторі, видалення невидимих ліній і поверхонь.

Для створення реалістичних зображень використовують різні моделі висвітлення, здійснюють зафарбовування поверхонь, що обмежують побудовані об'єкти (зафарбування методом Гуро і Фонга) та використовують текстури. Можливість зміни зображень у часі на екрані використовується при анімації і створенні комп'ютерного відео.

2. Особливості програмування графіки на мові PASCAL

Формування зображення на екрані дисплея здійснюється за допомогою ***відєоадаптера*** (відеокарти), роботу якого забезпечують ***графічні драйвери***.

Кожен драйвер містить виконуваний код і дані. Драйвера зберігаються на дисках в окремих файлах.

У мовах програмування є засоби, що ідентифікують графічну апаратуру, роблять завантаження й ініціалізацію відповідного графічного драйвера, переводять систему в графічний режим. По завершенню роботи програми вивантажують драйвер з пам'яті і відновлюють попередній відєорежим.

У ***мові PASCAL*** мається бібліотека, що включає близько 100 графічних процедур і функцій, реалізованих ***модулем GRAPH*** і який зберігається на диску у файлі ***GRAPH.TPU***. Модуль GRAPH дозволяє виконувати процедури і функції самого широкого застосування. Це можуть бути програми, орієнтовані на роботу з окремими крапками зображень, оперуючими частиною або всім полем графічного екрана. Модуль GRAPH підтримує кілька видів заповнення і типів ліній, кілька шрифтів, які можна змінювати по величині й орієнтувати чи горизонтально вертикально. ***Для графічних програм необхідно вказати компілятору ім'я директорії, де зберігається файл GRAPH.TPU*** (У меню Options-Directories-EXE and TPU directory:).

Ініціалізація графічної системи і переклад апаратури в графічний режим здійснюється процедурою ***InitGraph(Driver**:integer**, Mode**:integer**, Path**:string)***, у якій вказуються імена драйвера, що завантажується, режиму і шлях для пошуку графічного драйвера.

Графічні драйвери звичайно зберігаються на диску в ***директорії BGI***. Назва директорії відповідає ***розширенню BGI*** файлів графічних драйверів. У цій директорії також зберігаються файли шрифтів, що мають ***розширення CHR***. Наприклад:

cga.bgi	***bold.chr***
svga256.bgi	***goth.chr***
egavga.bgi	***sans.chr***
vesa16.bgi	***trip.chr***

Завершення роботи графічної програми здійснюється процедурою ***CloseGraph.*** При виконанні цієї процедури відновлюється попередній відєорежим і вивантажується графічний драйвер з пам'яті.

Структура графічної PASCAL - програми

```
Program Pr1;
Uses Graph; {підключення графічної бібліотеки}
Var Driver, Mode: integer; {змінні графічного драйвера і режиму }
Begin
    Driver:=Detect;  {автовиявлення}
    InitGraph(Driver, Mode,'C:\BP\BGI\'); {завантаження драйвера і  перехід у графічний режим}
                                . . .
     {  процедури і функції графічної програми }
                                . . .
    CloseGraph; {завершення - вивантажується драйвер з пам'яті і відновлюється попередній
відєорежим }
End.
```

Підключення стандартних драйверів

Установка відповідного графічного режиму може здійснюватися автоматично Driver:=Detect (режим автовиявлення) або вручну, використовуючи константи (як буквені, так і числові) графічного драйвера і режиму , наприклад: Driver:= VGA; Mode:=VgaHi; чи Driver:= 9; Mode:=2;

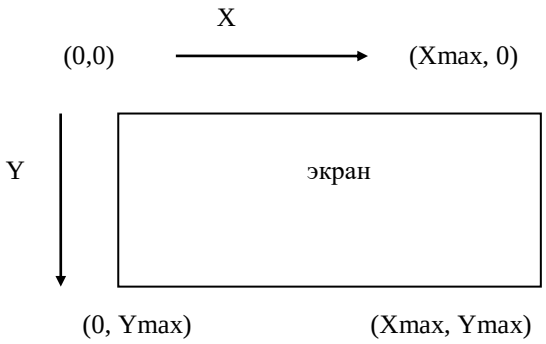
Підключення нестандартних драйверів

Підключення не вхідних у стандартний набір драйверів ***svga256.bgi*** *і* ***vesa16.bgi*** (за умови, якщо адаптер підтримує ці драйвери) може здійснюватися в такий спосіб:

Driver:=InstallUserDriver('SVGA256', nil); {замість адреси функції, що перевіряє, передається nil, функція перевіряє наявність відповідної відєокарти і повертає відєорежим}

```
Mode:=0;
InitGraph(Driver, Mode,'C:\BP\BGI\');
```

Система координат



Поточний покажчик

Для графічного режиму використовується поняття поточного покажчика CP (Current Pointer), що аналогічно поняттю курсору для текстового режиму. Але CP ми не бачимо і його значення не змінюється в залежності від його положення.

Процедури і функції, що визначають параметри графічного режиму і драйвера

GetMax: integer; - повертає розмір по осі X
GetMax: integer; - повертає розмір по осі Y
GetDriverName:string; - повертає ім'я поточного драйвера
GetGraphMode:integer; - повертає поточний графічний режим
GetModeName(Mode:integer):string; - повертає ім'я заданого графічного режиму
GetMaxMode:word; - повертає максимальне значення номера режиму для поточного драйвера
GetMaxColor:word; - повертає максимальне значення кольору
GetModeRange(Driver:integer;Var MinMode,MaxMode:integer); - повертає найменше і найбільше значення графічного режиму
Get:integer; - повертає значення X поточного покажчика
Get:integer; - повертає значення Y поточного покажчика

Настановні процедури

ClearDevice: - очищає екран
ClearViewPort; - очищає вікно
MoveTo(X,Y:integer) - установка поточного покажчика
SetBkColor(Color:word) - колір тла
SetColor(Color:word) - колір ліній
SetFillStyle(Pattern: Word; Color: Word); - заповнення
SetLineStyle(LineStyle: Word; Pattern: Word; Thickness: Word); - вибір стилю ліній (стандартний шаблон, шаблон користувача, товщина)
SetRGBPalette(ColorNum, RedValue, GreenValue, BlueValue: Integer);- палітра
SetTextStyle(Font, Direction: Word; CharSize: Word); - вибір шрифту
SetUserCharSize(Mult, Div, Mult, Div: Word); - розмір шрифту
SetViewPort(x1, y1, x2, y2: Integer; Clip: Boolean); - установка вікна
SetActivePage(Page: Word);- установка активної сторінки
SetVisualPage(Page: Word););- установка візуальної сторінки

Графічні примітиви

PutPixel(X, Y: Integer; Color: Word); - крапка
Line(x1, y1, x2, y2: Integer); - лінія
LineTo(x1, y1: Integer); - лінія від поточного покажчика
Circle(X,Y: Integer; Radius: Word); - окружність
Rectangle(x1, y1, x2, y2: Integer); - прямокутник
PieSlice(X, Y: Integer; StAngle, EndAngle, Radius: Word); - сектор
Ellipse(X, Y: Integer; StAngle, EndAngle: Word; XRadius, YRadius: Word); -еліпс
Arc (X,Y; Integer; StAngle, EndAngle, Radius; Word);- дуга окружності
Bar(x1, y1, x2, y2: Integer); - 2-D прямокутник
Bar3D(x1, y1, x2, y2: Integer; Depth: Word; Top: Boolean); - 3-D прямокутник

Фрагменти зображень

ImageSize(x1, y1, x2, y2: Integer): Word; - розмір зображення

GetImage(x1, y1, x2, y2: Integer; var BitMap); - збереження зображення
PutImage(X, Y: Integer; var BitMap; BitFun: Word); - висновок зображення

Текст

OutText (TextString: string); - висновок тексту від поточного покажчика
OutTextXY(X,Y: Integer; TextString: string); - висновок тексту в координати
Str(X [: Width [: Decimals]]; var S:string); - перетворення числа в рядок

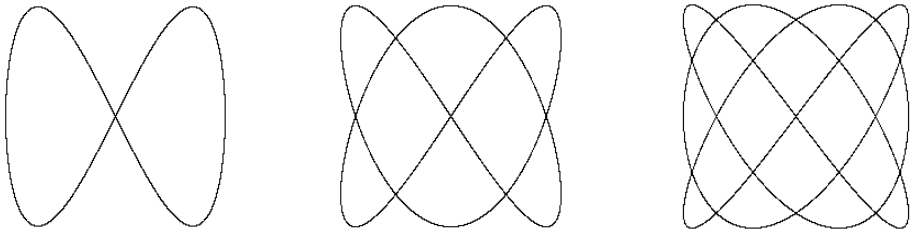
Фрагмент програми висновку графіка функції

```
SetViewPort(X1,Y1,X2,Y2,ClipOn);
Mx:=(X2-X1)/(Xmax-Xmin);
My:=(Y2-Y1)/(Ymax-Ymin);
D:=(Xmax-Xmin)/N;
X0:=(X2-X1) div 2 ;
Y0:=(Y2-Y1) div 2
MoveTo(X0+Round(Xmin*Mx), Y0– Round(Fun(Xmin)*My));
For i:=1 to N do
Begin
X;=X0+Round((Xmin+D*i)*Mx);
Y:=Y0 -Round(Fun(Xmin+D*i)*My);
LineTo(X,Y);
End;
```

Фігури Лісажу

Рівняння	Фрагмент програми

X=sin w_1t
Y= sin w_2t ,
де $2\pi \geq t \geq 0$
	MoveTo(X0,Y0);
	for i:=0 to N do
	begin
	X:=X0 + Round (sin(W1*2*Pi*i/N) *Mx);
	Y:=Y0 - Round (sin(W2*2*Pi*i/N) *My);
	LineTo(X,Y)
	end;



w_1=1; w_2=2 w_1=2; w_2=3 w_1=3; w_2=4

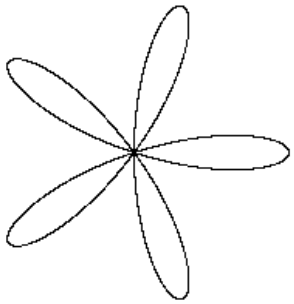
Рис. 1. Фігури Лісажу

Спірограф

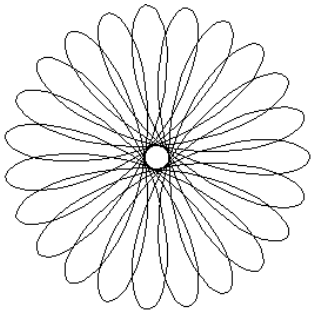
Рівняння	Фрагмент програми

X=(A - B) cos t + D cos f
Y=(A - B) sin t + D sin f ,
где f = (A/B) t,
D < B < A, 2π M ≥ t ≥ 0,
M = B/H
(H - найбільший загальний дільник A и B)
	dT:=2*Pi*M/N;
	MoveTo(X0+Round(Mx*((A-B)+D),Y0);
	for i:=0 to N do
	begin
	X:=X0+Round(Mx*((A-B)*cos(dT*i)+D* cos(A/B*dT*i)));
	Y:=Y0-Round(My*((A-B)*sin(dT*i)+D* sin(A/B*dT*i)));
	LineTo(X,Y)
	end;

Мережива



A=15; B=10; D=5; M=3



A=75; B=40; D=30; M=8

Рис.2. Спірограф

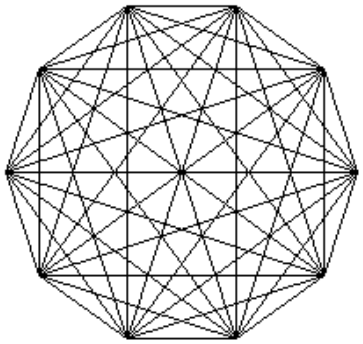


Рис. 3. Мережива

Рівняння	Фрагмент програми

x_i=R *cos (2π/N*i)
y_i=R* cos (2π/N*i),
где R –радиус,
N – количество вершин
	T:=2*Pi/N;
	MoveTo(X0+R,Y0);
	for i:=0 to N do
	for j:=i+1 to N do
	begin
	X1:=X0+Round(R* cos(T*i));
	Y1:=Y0-Round(R* sin(T*i));
	X2:=X0+Round(R* cos(T*j));
	Y2:=Y0-Round(R* sin(T*j));
	Line(X1,Y1,X2,Y2,)
	end;

Фрагмент програми вводу-виводу зображення

```
Program Pr2;
Uses Graph;
Const N=200;
      M=100; { N,M - розміри графічного зображення }
Var A: pointer;
Driver, Mode: integer;
Begin
      Driver:=Detect;
      InitGraph(Driver, Mode,'C:\BP\BGI\');
                                  . . .
       {  процедури для формування графічного зображення
                 на ділянці 0,0 - N*M}
                                  . . .
   GetMem(A, ImageSize(0,0,N,M)); {виділяється пам'ять для зображення}
   GetImage(0,0,N,M,A^);{зображення запам'ятовується в А від крапки 0,0 до N,M }
      ClearDevice;
   PutImage(100,100,A^,XorPut); { XorPut =1} {зображення виводиться з А в крапку 100,100}
      FreeMem(A, ImageSize(0,0,N,M)); {звільняється пам'ять}
      CloseGraph;
End.
```

Можна також малювати криві в тривимірному просторі для представлення залежності між декількома змінними. Як приклад, розглянемо графіки функції Z = Z0 + H * SIN (W * SQRT(X * X + Y * Y)) у заданому діапазоні значень координат X и Y при обраних фіксованих значеннях Z0, H і W. Діапазон зміни значень координати X обчислюється з вираження XR*SQRT(1 — (Y *Y)/(YR * YR)). Це рівняння еліпса визначає діапазон X в залежності від координати Y. Процес починається з креслення коротких кривих (при мінімальних значеннях Y, рівних - YR), потім їхня довжина збільшується до максимуму (при Y = 0) і знову зменшується до первісних розмірів (при максимальному Y, рівному YR). Остаточний вид отриманої поверхні показаний на мал. 4. Для визначення діапазону значень X можна використовувати будь-яку іншу

функціональну залежність (наприклад, лінійну чи константу) для всіх значень Y. Діапазон зміни значень координати Y і постійні X0, Z0, H, W, XR і YR вибираються з таким розрахунком, щоб зображення заповнило весь екран. Вичерчуючи X + Z і Y + Z/2 замість **X та Y,** зрушуємо малюнок вправо і нагору, щоб розосередити криві і створити відчуття тривимірного простору.

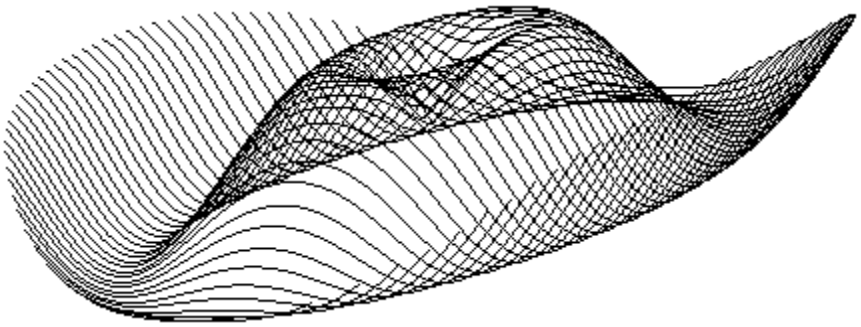


Рис. 4. Тривимірний графік функції

Можна не вичерчувати невидимі лінії (мал. 5). Це досягається послідовним кресленням кривих, починаючи з передніх, за винятком схованих ділянок, тобто ділянок кривих, що будуть перекриті раніше накресленими лініями. Для кожного значення X запам'ятовуються максимальне і мінімальне значення Y для раніше намальованих кривих. Усі наступні обчислені відрізки кривої, що попадають у цей інтервал, не малюються. Відрізки кривої, розташовані поза границями діапазону, малюються, при цьому границі діапазону відповідно змінюються. Розмірність масивів, призначених для збереження значень границь, вибирається виходячи з максимального діапазону значень X. Для обчислення максимального діапазону можна скористатися аналітичними чи методами написати коротку програму, що обчислює значення X і друкуючу його мінімальне і максимальне значення.

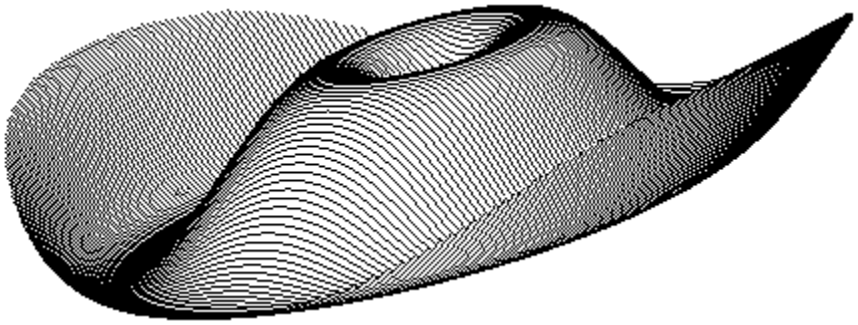


Рис.5. Тривимірний графік функції
(з вилученням невидимих ліній)

3. Перетворення координат

Координатний метод був введений у 17 столітті французькими математиками П. Ферма і Р. Декартом. На цьому методі ґрунтується аналітична геометрія, яку можна вважати фундаментом КГ.

У сучасної КГ широко використовується координатний метод по наступним причинах:

- кожна точка на екрані має свої координати;
- координати використовуються для опису об'єктів як на площині, так і в просторі;
- при використанні більшості проміжних дій у КГ використовуються системи координат і перетворення з однієї системи в іншу.

Спочатку розглянемо загальні питання перетворення координат.

Нехай буде задана n-мірна система координат. Тоді кожній точці М ставиться у відповідність безліч чисел (m_1, m_2, … , m_n) її координат. У КГ найчастіше використовується двовимірна (n=2) і тривимірна (n=3) системи координат. Далі введемо ще нову N-мірну систему координат. Тоді тій же крапці М ставиться у відповідність інша безліч чисел - (m^*_1, m^*_2, … , m^*_N)...

Перехід від n - ої системи координат до N -ої описується наступними співвідношеннями:

$$m^*_1 = f_1(m_1, m_2, \ldots, m_n) ;$$
$$m^*_2 = f_2(m_1, m_2, \ldots, m_n) ;$$
…
$$m^*_N = f^*_N (m_1, m_2, \ldots, m_n) ,$$

де f_i - функція перерахування i-ої координати в N -мірній системі , аргументами якої є координати m_i.

Зворотна задача, тобто перехід від N - мірної координат до n -мірного описується співвідношеннями:

$$m_1 = F_1(m^*_1, m^*_2, \ldots, m^*_N) ;$$
$$m_2 = F_2(m^*_1, m^*_2, \ldots, m^*_N) ;$$
…
$$m_n = F_n(m^*_1, m^*_2, \ldots, m^*_N) ;$$

де F_i - функція зворотного перерахування i-ої координати в n-мірній системі , аргументами якої є координати m^*_i.

Якщо розмірність систем координат не збігається ($n \neq N$), то часто таке перетворення є неоднозначним.

Якщо функції перетворень f_i і F_i лінійні ($f_i = a_{1i}m_1 + a_{2i}m_2 + \ldots a_{ni}m_n + a_{ni+1}$, $F_i = b_{1i}m^*_1 + b_{2i}m^*_2 + \ldots b_{Ni}m^*_N + b^*_{Ni+1}$) і $n = N$, то такі перетворення називають афінними.

Лінійні перетворення описуються в матричній формі, наприклад:

$$\begin{pmatrix} m^*_1 & m^*_2 & \ldots & m^*_N \end{pmatrix} = \begin{pmatrix} m_1 & m_2 & \ldots m_n & 1 \end{pmatrix} \cdot \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1N} \\ a_{21} & a_{22} & .. & a_{2N} \\ & & \ldots & \\ a_{n1} & a_{n2} & & a_{3N} \\ a_{n1+1} & a_{n2+1} & \ldots & a_{nN+1} \end{pmatrix} \text{ чи } M^* = M \cdot A .$$

Іноді в літературі використовується інший запис – по стовпцях:

$$\begin{pmatrix} m^*_1 \\ m^*_2 \\ \\ m^*_N \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} & a_{1n+1} \\ a_{21} & a_{22} & \ldots & a_{2n} & a_{2n+1} \\ & & \ldots & & \\ a_{N1} & a_{N1} & & a_{Nn} & a_{Nn+1} \end{pmatrix} \cdot \begin{pmatrix} m_1 \\ m_2 \\ .. \\ m_n \\ 1 \end{pmatrix} \text{ чи } M^* = A \cdot M .$$

Перетворення на площині (2 D - перетворення)

У комп'ютерній графіці усе, що відноситься до двовимірного випадку прийнято позначати символом 2D.

Припустимо, що на площині введена прямокутна система координат. Тоді кожна точка М визначається упорядкованою парою чисел М(х,у) її координат (рис. І). При зміні положення точки їй ставиться у відповідність інша пара чисел М*(х*,у*) .

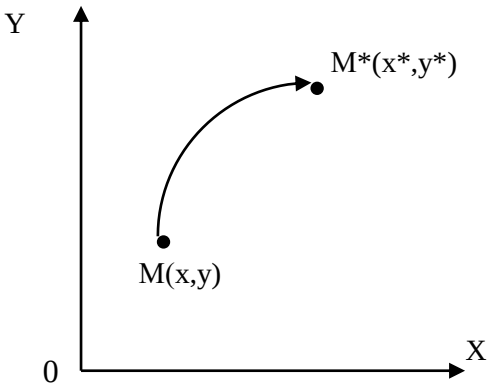


Рис. 1. Зміна положення точки

Перехід від одного положення точки до іншого на площині описується співвідношеннями:

$$x^* = \alpha x + \beta y + \lambda,$$

$$y^* = \gamma x + \delta y + \mu,$$

де $\alpha, \beta, \lambda, \gamma, \delta, \mu$ - деякі константи, причому $\alpha, \beta, \gamma, \delta$ зв'язані нерівністю

$$\begin{vmatrix} \alpha & \beta \\ \gamma & \delta \end{vmatrix} \neq 0 \ .$$

Ці формули можна розглядати не тільки як зміна координат точки, але і як перехід до іншої прямокутної системи координат при незмінному положенні точки (рис. 2).

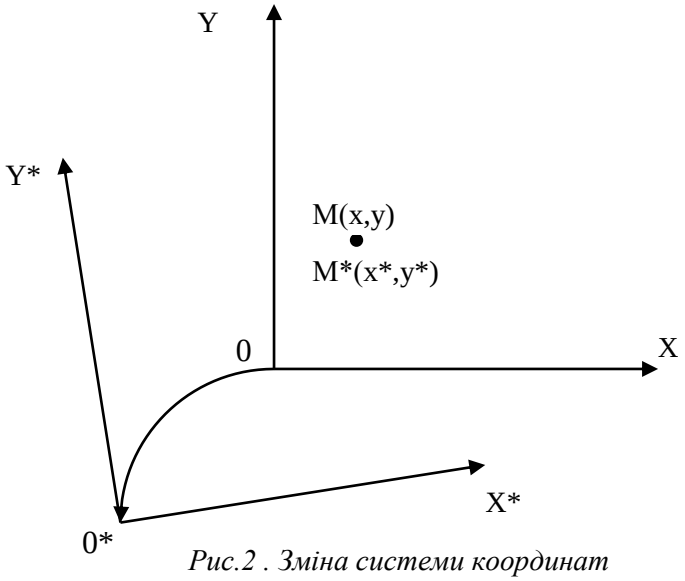


Рис.2 . Зміна системи координат

Надалі будемо розглядати перший випадок, коли в заданій системі координат змінюються координати точки.

Будь-яке перетворення на площині можна представити як послідовність 4 найпростіших перетворень:

1. Поворот
2. Розтягування
3. Відбиття
4. Перенос

Вибір цих 4-х базових перетворень визначається наступними обставинами:

- Перетворення мають простий і наочний геометричний зміст.
- Будь-яке перетворення можна представити як послідовність цих базових перетворень.

Дані перетворення можуть описуватися за допомогою математичних моделей, заснованих на використанні матриць. При цьому слід зазначити, що перші три перетворення (поворот, розтягання і відображення) можуть бути описані двовимірною матрицею, тоді як четверте перетворення (перенос) двовимірною матрицею представити неможливо. Тому для опису зазначених геометричних перетворень використовують метод однорідних координат.

В основі цього методу лежить представлення про те, що кожна точка в N - мірному просторі може розглядатися як проекція точки з (N + 1) - мірного простору. Зокрема, точка в 2-х мірному просторі представляється трьома складовими hx, hy, h , де $h \neq 0$. На практиці звичайно приймають h=1, що відповідає нормалізованим координатам (x,y,1).

Однорідні координати дозволяють виражати за допомогою 3-х мірних матриць 4 основні перетворення на площині, а також будь-які сполучення цих перетворень у виді добутку відповідних матриць. Тому можливо застосування єдиного математичного апарата для просторових перетворень на площині. Якщо операції переносу не використовуються, то необхідність в однорідних координатах відпадає.

Математичний запис 2 D – перетворень у матричному виді з використанням однорідних координат виглядає в такий спосіб:

$$(X^*, Y^*, 1) = (X, Y, 1) * A \ ,$$

де (X, Y, 1) - координати вихідної точки, A - матриця перетворення, $(X^*, Y^*, 1)$ - координати точки після перетворення.

Відрізок на площині в однорідних координатах задається матрицею $\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{pmatrix}$, трикутник -

матрицею $\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}$, а n-кутник – матрицею $\begin{pmatrix} x_1 & y_1 & 1 \\ \dots & \dots & \dots \\ x_i & y_i & \dots \\ \dots & \dots & \dots \\ x_n & y_n & 1 \end{pmatrix}$.

У загальному виді перетворення на площині можна записати:

$$S=Q* A,$$

де Q – матриця вихідних координат двовимірної фігури, S - матриця координат двовимірної фігури після перетворень, A - матриця перетворень.

Якщо відомі початкові Q і кінцеві S координати двовимірної фігури, то можна знайти матрицю перетворення, помноживши на Q^{-1} обох частин вираження (1), тобто

$$Q^{-1}Q* A=Q^{-1}S$$

чи

$$A=Q^{-1}S.$$

При виконанні послідовності перетворень на площині:

$$S=Q* A1*A2*\dots*An,$$

результуюче перетворення буде дорівнює:

$$S=Q* A,$$

де A=A1*A2*…*An.

Розглянемо вираження і матричні представлення основних перетворень на площині з використанням однорідних координат.

Обертання

Рівняння обертання	Матриця обертання
$x^* = x \cdot cos\varphi - y\, sin\varphi$ $y^* = x \cdot sin\varphi + y\, cos\varphi$	$R = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Розтягування

Рівняння розтягування	Матриця розтягування
$x^* = \alpha x$ $y^* = \beta y$ $\alpha > 0, \quad \beta > 0$	$D = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Відбиття

(щодо осі абсцис і ординат)

Рівняння відбиття щодо осі		Матриця відбиття щодо осі	
абсцис	ординат	абсцис	ординат
$x^* = x$ $y^* = -y$	$x^* = -x$ $y^* = y$	$M_Y = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$M_X = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Перенос

Рівняння переносу	Матриця переносу
$x^* = x + \lambda$ $y^* = y + \mu$	$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \lambda & \mu & 1 \end{pmatrix}$

5. Перетворення в просторі (3 D - перетворення)

Припустимо, що в просторі введена прямокутна система координат. Тоді кожна точка М визначається упорядкованою парою чисел М(x,y,z) її координат (мал. I). При зміні положення точки їй ставиться у відповідність інша пара чисел М*(x*,y*,z*) .

Перехід від одного положення точки до іншого в просторі описується співвідношеннями:

$$x^* = a_1 x + a_2 y + a_3 z + a_4,$$
$$y^* = b_1 x + b_2 y + b_3 z + b_4,$$
$$z^* = c_1 x + c_2 y + c_3 z + c_4$$

де a_i , b_i , $i = \overline{1,4}$ - деякі константи, причому зв'язані нерівністю

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} \neq 0 .$$

Використання однорідної системи координат дозволяє застосовувати єдиний математичний апарат для просторових перетворень.

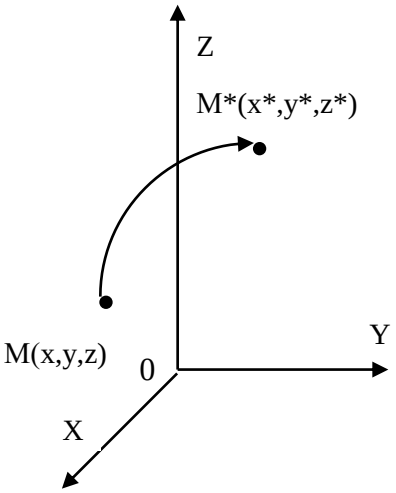


Рис. 1. Зміна положення точки у просторі

При використанні методу однорідних координат точка у тривимірному просторі представляється чотирма складовими hx, hy, hz, h , де h - може приймати будь-яке значення. На практиці h=1, що відповідає нормалізованим координатам (x, y, z, 1).

Для 3D - перетворення (X^*, Y^*, Z^*, 1) = (X, Y, Z, 1) * | A | , де (X, Y, Z, 1) - координати вихідної точки, | A | - матриця перетворення, (X^*, Y^*, Z^*, 1) - координати точки після перетворення.

У просторі, як і на площині, теж можна виділити 4 основні операції: обертання, розтягування, відображення і переносу.

У загальному виді, з використанням однорідних координат ці перетворення описуються матрицею виду:

$$\begin{matrix} \textit{поворот} \\ \\ \\ \textit{сдвиг} \end{matrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ \lambda & \mu & \nu & 1 \end{pmatrix} \begin{matrix} \textit{проекция} \\ \\ \\ \textit{множитель} \end{matrix}$$

Розглянемо матриці для кожного виду перетворень у просторі.

Матриця обертання навколо осі X: $|R_x| = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$.

Матриця обертання навколо осі Y: $|R_y| = \begin{vmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$.

Матриця обертання навколо осі Z: $|R_z| = \begin{vmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$.

Матриця розтягування: $|D| = \begin{vmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$.

Матриця відображення щодо площини XY: $|M_Z| = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$.

Матриця відображення щодо площини YZ: $|M_X| = \begin{vmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$.

Матриця відображення щодо площини ZX: $|M_Y| = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$.

Матриця переносу: $|T| = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & \mu & \nu & 1 \end{vmatrix}$.

При виконанні послідовності перетворень у просторі результуюча матриця буде дорівнювати добутку відповідних матриць. Матриця обертання одночасно навколо двох осей: щодо осі Y на кут w і щодо осі X на кут f буде дорівнює:

$$A = R_y \cdot R_x = \begin{pmatrix} \cos w & 0 & -\sin w & 0 \\ 0 & 1 & 0 & 0 \\ \sin w & 0 & \cos w & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos f & \sin f & 0 \\ 0 & -\sin f & \cos f & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos w & \sin w * \sin f & -\sin w * \cos f & 0 \\ 0 & \cos w & \sin f & 0 \\ \sin w & -\sin w * \sin f & \cos w * \cos f & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Як приклад, розглянемо побудову матриці обертання деякої точки $M(x, y, z)$ на кут φ навколо довільної прямої L, що проходить через точку $A(a,b,c)$ і має одиничний направляючий вектор (p,m,n), де $p^2 + m^2 + n^2 = 1$(мал.2). Рішення цієї задачі розбивається на ряд послідовних перетворень:

1. Перенос вихідної прямої так, щоб вона проходила через початок координат.
2. Сполучення цієї прямої з однієї з осей координат.
3. Поворот точки навколо цієї осі координат.
4. Виконання перетворень у зворотному порядку.

Перенос точки $A(a,b,c)$ в початок координат здійснюється за допомогою матриці

$$T(-a,-b,-c) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{pmatrix}.$$

Після такого перетворення пряма буде проходити через початок координат.

Далі, сполучення прямої з віссю Z можна здійснити

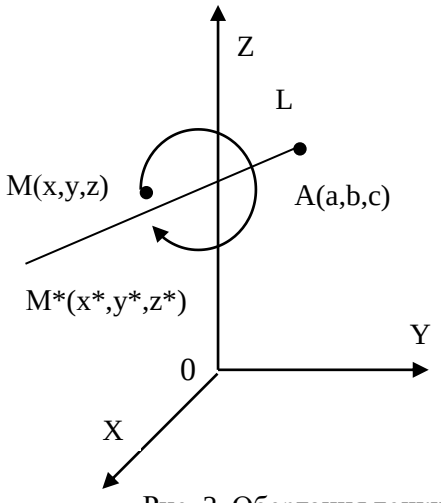


Рис. 2. Обертання точки вокруг прямой

двома поворотами. Наприклад, поворотом навколо осі X на кут ψ (пряма сполучається з площиною XOZ) і поворотом навколо осі Y на кут θ (у площині XOZ до сполучення з віссю Z).

Кут ψ знаходиться з проекції точки (p,m,n) на площину XOZ. Оскільки поворот здійснюється проти годинної стрілки, то кут ψ позитивний і $\cos\psi = \frac{n}{d}$, $\sin\psi = \frac{m}{d}$, де $d = \sqrt{m^2 + n^2}$. Матриця такого обертання буде:

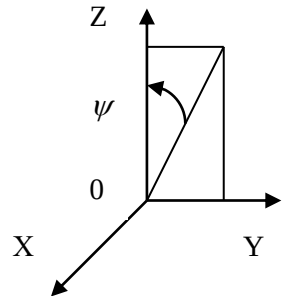


Рис.3. Совмещение точки с плоскостью XOZ

$$R_x(\psi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{n}{d} & \frac{m}{d} & 0 \\ 0 & -\frac{m}{d} & \frac{n}{d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$ Під дією перетворення, яке описується матрицею $R_x(\psi)$, координати вектора (p,m,n) зміняться, тобто $(p,m,n,1)\cdot R_x(\psi) = (p,0,d,1)$.

Кут θ знаходиться з положення точки ($p,0,d,1$) на площині XOZ (мал.4). Оскільки поворот здійснюється за годинною стрілки, то кут θ негативний ($\cos\theta = d$, $\sin\theta = p$. Матриця такого обертання

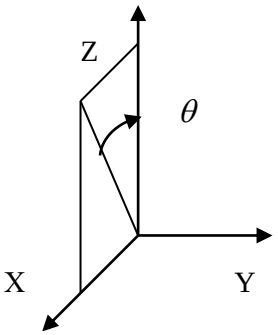


Рис.4. Совмещение точки с осью Z

$$R_y(-\theta) = \begin{pmatrix} d & 0 & p & 0 \\ 0 & 1 & 0 & 0 \\ -p & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$ Під дією перетворення, яке описується матрицею $R_y(\theta)$, координати вектора ($p,0,d,1$) зміняться, тобто $(p,0,d,1)\cdot R_y(-\theta) = (0,0,p^2+d^2,1)$, де $p^2 + d^2 = 1$. Пряма сполучилася з віссю Z. Подібні перетворення використовують у нарисній геометрії, коли потрібно знайти натуральну довжину відрізка.

Матриця повороту точки на кут φ навколо осі Z дорівнює:

$$R_z(\varphi) = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Тепер виконуються зворотні перетворення $R_y(\theta) = \begin{pmatrix} d & 0 & -l & 0 \\ 0 & 1 & 0 & 0 \\ l & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $R_x(-\psi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{n}{d} & -\frac{m}{d} & 0 \\ 0 & \frac{m}{d} & \frac{n}{d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ і

$$T(a,b,c) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{pmatrix}.$$

Таким чином, загальне перетворення A виходить з перемножування матриць:

$A = T(-a,-b,-c)\cdot R_X(\psi)\cdot R_Y(-\theta)\cdot R_Z(\varphi)\cdot R_Y(\theta)R_X(-\psi)\cdot T(a,b,c)$.

Розглянемо приклад PASCAL-програми, що реалізує обертання куба навколо двох осей Y і X з

використанням матриці $A = \begin{pmatrix} \cos w & \sin w * \sin f & 0 & 0 \\ 0 & \cos w & 0 & 0 \\ \sin w & -\sin w * \sin f & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, що враховує проектування просторових координат куба на екран монітора уздовж осі Z.

```
Program Demo3D_1;
Uses Crt,Graph;
Type Vector=array [1..3] of real;
Type Massiv=array [1..3] of Vector;
Const
K:array [1..5,1..4] of vector=
      (((0,100,0),(0,100,100),(100,100,100),(100,100,0)),
       ((100,0,0),(100,100,0),(100,100,100),(100,0,100)),
      ((0,0,100),(0,100,100),(100,100,100),(100,0,100)),
      ((0,0,0),(0,100,0),(0,100,100),(0,0,100)),
       ((0,0,0),(0,100,0),(100,100,0),(100,0,0)));
Var Miz:Massiv;
   T,Ti:Vector;
   X,Y,Z:real;
   x0,y0,dr,md,i,j,p,ap,vp,w1,f1:integer;
Procedure MultVector(q1:integer;Var V1,V2:Vector;Var sm:real);
 Var i:integer;
 Begin
   sm:=0;
   for i:=1 to q1 do sm:=sm+V1[i]*V2[i]
 End;
Procedure MultVectorMassiv(q1:integer;Var V1:Vector; Var M1:Massiv; Var V2:Vector);
Var n:integer;
   i:integer;
   Sum:real;
 Begin
  n:=q1;
              for i:= 1 to n do
               Begin
                 MultVector(n,V1,M1[i],sum);
                 V2[i]:=sum;
               end
 End;
 Procedure FigRot;
 var w,f:real;
 begin
   W:=2*PI/500*w1;
   f:=2*PI/500*f1;
   Miz[1,1]:=cos(w);          Miz[1,2]:=0;            Miz[1,3]:=sin(w);
   Miz[2,1]:=sin(f)*sin(w);   Miz[2,2]:=cos(w);       Miz[2,3]:=-sin(w)*cos(w);
   Miz[3,1]:=0;               Miz[3,2]:=0;            Miz[3,3]:=0;
 For j:=1 to 5 do
     begin
     for i:=1 to 4 do
           begin
   T:=K[j,i];
   MultVectorMassiv(3,T,Miz,Ti);
   if i=1 then
            begin x:=Ti[1];y:=Ti[2];z:=Ti[3];
               MoveTo(x0+round(x),Y0-round(y/1.37));
            end
         else
   LineTo(x0+round (Ti[1]),Y0-round (Ti[2]/1.37));
```

```
        end;
   LineTo(x0+round (x),y0-round (Y/1.37));
   end;
 end;
{*********************************************************************}
begin
 dr:=9;
 md:=1;
 InitGraph (Dr,Md,'d:\bp\bgi');
 ap:=1;
 vp:=0;
   x0:= getMaxX div 2;
   y0:= getMaxY div 2;
 w1:=0;
 f1:=0;
Repeat
clearviewport;
FigRot;
p:=ap;
ap:=vp;
vp:=p;
setvisualpage(vp);
setactivepage(ap);
w1:=w1+1;
f1:=f1+1;
until KeyPressed;
CloseGraph;
end.
```

6. Вилучення невидимих ліній і поверхонь

(Відсікання нелицьових граней)

Розглянемо опуклий багатогранник. Для кожної грані побудуємо вектор зовнішньої нормалі. Якщо вектор нормалі грані становить із вектором проектування тупий кут (90 - 270), то ця грань не видима й вона називається нелицьовою. Якщо (є гострим (0 - 90 або 270 - 360), то грань видима й називається лицьовий. Таким чином, якщо cos > 0, то грань лицьова, а якщо cos < 0, то грань нелицьова.

Якщо точку зору вибрати на осі Z, тобто напрямок проектування буде перпендикулярно площини XOY, то видимість площин можна визначити за допомогою координати Z їхніх нормалей. Якщо координата Z нормалі > 0, то грань видима, а якщо координата Z нормалі < 0, то грань невидима.

Пронумеруємо всі видимі вершини опуклого багатокутника проти годинникової стрілки, а невидимі - по ходу годинникової стрілки. Для визначення координата Z нормалі досить 3-х вершин

Знайдемо: d1=X 2-X1; d1=Y 2-Y1; d2=X 3-X2; d2=Y 3-Y2;

Обчислимо Z=d1* d2 - d2* d1

Якщо Z > 0, то грань видима. Якщо Z > 0, то грань невидима.

7. Растрова візуалізація

Для роботи з пристроями растрової візуалізації використовуються спеціальні алгоритми генерації зображень, креслення прямих ліній і окружностей, зафарбування багатокутників.

Екран растрового дисплея можна розглядати як матрицю дискретних елементів (пікселів). Процес визначення пікселів, щонайкраще апроксимуючих задане зображення, називається розкладанням у растр. У сполученні з процесом порядкової візуалізації зображення він відомий як растерізація зображення. Якщо при цьому використовувати тільки цілочислену арифметику, то можна збільшити швидкодія алгоритму растерізації.

Здійснюючи послідовне збільшення координат на основі аналізу деякої функції помилки, растрові алгоритми виконують обчислення координат сусідніх пікселів . При цьому варто розрізняти алгоритми, що використовують у якості сусідніх 4-х чи зв'язкові 8-ми зв'язкові піксели (мал. 3.1). Координати $(x_i,\ y_i)$ і $(x_{i+1},\ y_{i+1})$ будь-яких двох 4-х зв'язкових пікселів зв'язані співвідношенням $|x_{i+1}-x_i|+|y_{i+1}-y_i|\le 1$. А для будь-яких двох 8-х зв'язкових пікселів справедливі співвідношення $|x_{i+1}-x_i|\le 1;\ |y_{i+1}-y_i|\le 1$.

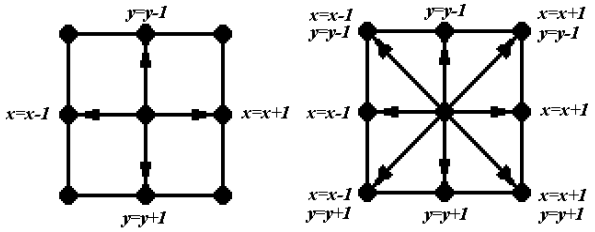
Цілочислений алгоритм Брезенхема для генерації прямих ліній.

При розкладанні відрізка в растр можна скористатися рекурентним співвідношенням виду [Роджерс]:

$$y_{i+1}=y_i+\Delta y;$$

$$y_{i+1}=y_i+\frac{y_2-y_1}{x_2-x_1}\Delta x\ ,$$

де $x_1,\ y_1$ і $x_2,\ y_2$ – кінці відрізка, що розкладається, y_i і - початкове значення для чергового кроку уздовж відрізка. Однак, такий підхід має недолік, що виявляється у використанні дійсної арифметики для роботи на цілочисленій сітці.



Чотирисвязність Восьмисвязніс

Рис. 3.1. З‘єднання сусідніх

У 1965 році Брезенхейм запропонував простий цілочислений алгоритм для растрової побудови відрізка. Алгоритм Брезенхема вибирає оптимальні растрові координати для представлення відрізка. Для горизонтальних, вертикальних і нахилених під кутом 45° прямих ліній вибір растрових елементів очевидний. Однак при іншій орієнтації вибір необхідних пікселів може виявитися скрутним (мал. 3.2).

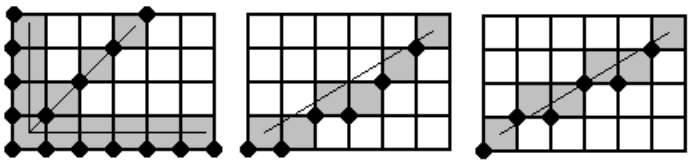


Рис. 3.2. Разклад в растр відрізків

Припустимо, що кутовий коефіцієнт відрізка менше одиниці. Тоді в процесі роботи алгоритму *x* збільшується на одиницю, а *y* – або не змінюється, або збільшується на одиницю в залежності від помилки (відстані між дійсним положенням відрізка і найближчої координати сітки). Алгоритм використовує лиш знак помилки *e* (мал. 3.3). Якщо *e<0,* то відрізок проходить нижче середньої крапки, тобто *в* не збільшується. Якщо *e>0,* то відрізок проходить вище середини й *y* збільшується на одиницю. Оскільки перевіряється лише знак помилки, те її початкове значення $e_0=-1/2$.

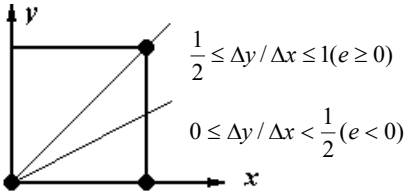


Рис. 3.3. Визначення знаку помилки

Приведемо цілочислений алгоритм Брезенхема для першого квадранта (мал. 3.4, а). Передбачається, що кінці відрізка $x_2>x_1$ і $y_1>y_2$ і всі перемінні - цілі [Роджерс].

Узагальнений цілочислений алгоритм Брезенхема для всіх квадрантів можна реалізувати, якщо враховувати номер квадранта, у якому лежить відрізок, і його кутовий коефіцієнт (мал. 3.4, б).

Побудова растрового представлення відрізка не є єдиним. На основі використання алгоритму Брезенхема можливі різні способи його реалізації. [Шикин, Порев].

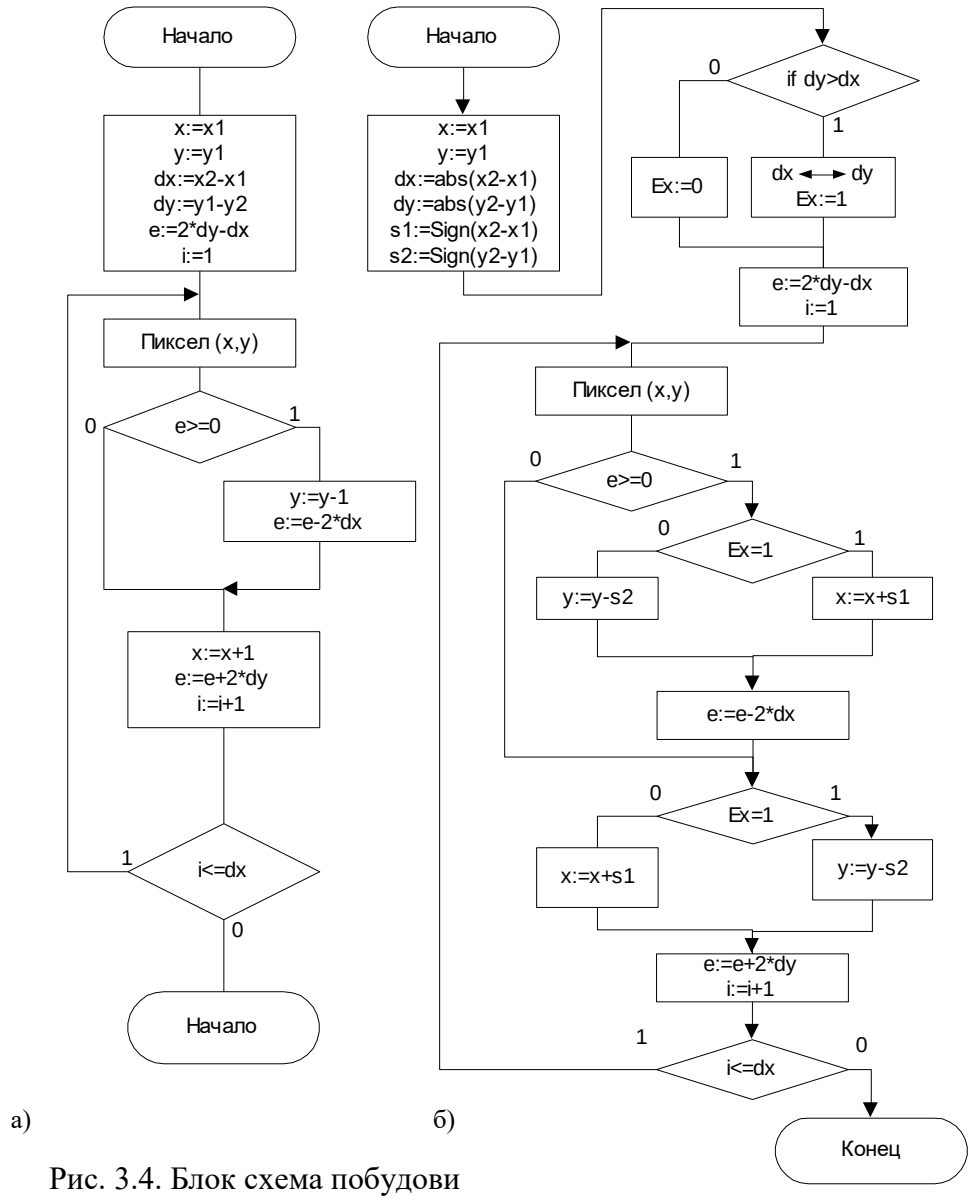


Рис. 3.4. Блок схема побудови

Наприклад, для побудови 4-х зв'язковий розгорнення відрізка приведемо фрагмент програми мовою ПАСКАЛЬ (четвертий квадрант для $x_2>x_1$ і $y_2>y_1$).

```
Procedure LineBR_4(x1,y1,x2,y2:integer);
Var i,dx,dy,e,e1,e2,x,y:integer;
begin
    dx:=x2-x1;
    dy:=y2-y1;
    e:=0;
    PutPixel(x1,y1,15);
    x:=x1;
    y:=y1;
    for i:=1 to dx+dy do
        begin
            if e>0 then
             begin
                e:=e-dx;
                y:=y+1;
            end
                    else
                begin
                e:=e+dy;
                x:=x+1;
            end;
```

```
                PutPixel(x,y,15);
            end;
    end;
```

Для загального випадку довільного відрізка фрагмент програми 8-ми зв'язного алгоритму Брезенхема може мати вид.

```
Procedure LineBR_8(x1,y1,x2,y2:integer);
Var dx,dy,s1,s2,e,i,dx,dy,x,y:integer;
Function Sign(z:integer):integer;
begin
    if z>0 then sign:=1;
    if z=0 then sign:=0;
    if z<0 then sign:=-1;
end;
    begin
        sx:=0; sy:=0;
        dx:=x2-x1; dy:=y2-y1;
        s1:=sign(dx); s2:=sign(dy);
        dx:=abs(dx); dy:=abs(dy);
        if dx>dy then e:=dx else e:=dy;
        x:=x1;
        y:=y1;
        PutPixel(x,y,15);
        for i:=1 to e do
            begin
                sx:=sx+dx; sy:=sy+dy;
                if sx>e then
                    begin
                        sx:=sx-e; x:=x+s1;
                    end;
                if sy>d then
                    begin
                        sy:=sy-e; y:=y+s2;
                    end;
                PutPixel(x,y,15);
            end;
    end;
```

8-х зв'язкові алгоритми більш складні і реалізують краще зображення відрізка при меншому числі пікселів у порівнянні з 4-х зв'язковими алгоритмами.

Алгоритм Брезенхема для генерації окружності.

Використовуючи рівняння окружності з центром на початку координат $y = \pm\sqrt{R^2 - x^2}$, можна шляхом прямого обчислення координат перетворити окружність у растрову форму. Однак цей підхід не є ефективний, тому що він використовує операції витягу кореня і множення.

Одним з найбільш ефективних для генерації окружності є алгоритм Брезенхема. На кожнім кроці вибирається крапка, що є найближчої до щирої окружності. При цьому використовуються перемінні, значення яких обчислюються в покроковому режимі з використанням невеликого числа цілочислених операцій і аналізу знака помилки. Оскільки окружність має осі симетрії, те раціонально сформувати тільки одну восьму її частину, а інші – одержати відповідним відображенням.

Припустимо, що формується 1-а частину окружності (мал.3.5). Тоді вибір крапок растра може здійснюватися на основі наступних рекурентних співвідношень [Дж. Фоли]. Крапка $(x_i + 1, y_i)$ вибирається, коли $e_i < 0$,

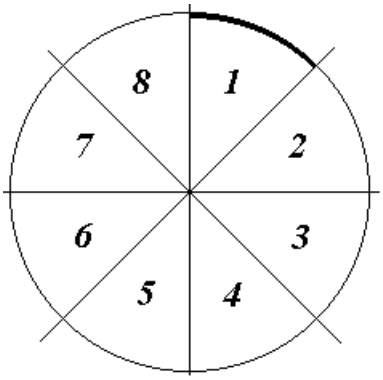
$$e_{i+1} = e_i + 4x_i + 6 \ ,$$

а крапка $(x_i + 1, y_i - 1)$ вибирається, коли $e_i \geq 0$,

$$e_{i+1} = e_i + 4(x_i - y_i) + 10 \ ,$$

де $d_1 = 3 - 2R$.

Нижче приводиться фрагмент програми мовою ПАСКАЛЬ для реалізації окружності відповідно до алгоритму Брезенхема (x1, x2 – координати центра кола, R – радіус окружності, усі перемінні – цілі). У програмі обчислюються координати тільки 1 частини окружності, а інші 2-8 частини малюються послідовно як симетричні піксели.



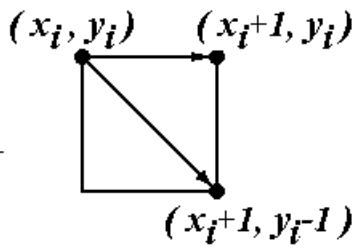


Рис. 3.5. Растеризація кола у першому квадранті

```
Procedure CircleBR(x1,y1,R:integer);
Var e,x,y:integer;
  begin
    x:=0;
    y:=R;
    e:=3-2*R;
    while x<y do
      begin
        PutPixel(x1+x,y1-y,15);{1}
        PutPixel(x1+y,y1-x,15);{2}
        PutPixel(x1+y,y1+x,15);{3}
        PutPixel(x1+x,y1+y,15);{4}
        PutPixel(x1-x,y1+y,15);{5}
        PutPixel(x1-y,y1+x,15);{6}
        PutPixel(x1-y,y1-x,15);{7}
        PutPixel(x1-x,y1-y,15);{8}
        if e<0 then
                    e:=e+4*x+6
              else
                    begin
                      e:=e+4*(x-y)+10;
                      y:=y-1
                    end;
        x:=x+1
      end;
  end;
```

При реалізації розглянутих алгоритмів спостерігається сходовий ефект, основна причина появи якого полягає в тім, що відрізки, що мають безупинну природу, відображаються на дискретному растровому дисплеї. Поліпшити зображення можна за допомогою двох основних методів усунення ступінчастості. Перший метод припускає, що растр потрібно обчислювати з більш високим дозволом, а відображати з більш низьким, використовуючи різні усереднення характеристик пікселів. Другий метод розглядає піксел не як крапку, а як кінцеву область і відображає його з інтенсивністю пропорційно площі піксела, зайнятої відрізком багатокутника (мал. 3.6). Такий підхід еквівалентний префільтрації зображення і згладжування ліній зв'язане з деяким розмиванням їхніх границь.

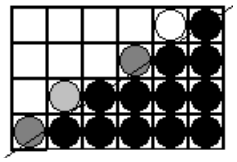


Рис. 3.6. Усунення ступеневості

Цілочислений алгоритм Брезенхема для генерації прямих ліній з усуненням ступінчастості.

У результаті модифікації алгоритму Брезенхема можна одержати апроксимацію площі частини піксела, зайнятого лінією, і використовувати цю апроксимацію для модуляції інтенсивності відображення піксела. Крім того, можна компенсувати яскравість відрізка в залежності від тангенса кута нахилу (враховувати розподіл інтенсивності уздовж довжини відрізка), малюючи лінії постійної яскравості. Наприклад, для побудови відрізка з використанням *N* рівнів інтенсивності приведемо фрагмент програми мовою ПАСКАЛЬ (четвертий квадрант для $x_2>x_1$ і $y_2>y_1$).

```
Procedure LineBR_N(x1,y1,x2,y2,N:integer);
Var x, y, dx, dy ,e, m,w: integer;
Begin
    x:=x1;
    y:=y1;
    dx:=x2-x1;
    dy:=y2-y1;
    m:=Round((N*dy)/dx);
```

```
    w:=N-m;
    e:=N div 2;
    InputPixel(x, y, m div 2);
    While (x<x2) do
        begin
            if e<w  then
             begin
                x:=x+1;
                e:=e+m;
            end
                    else
                begin
                x:=x+1;
                y:=y+1;
                e:=e-w;
            end;
            InputPixel(x, y, e);
        end;
end;
```

Для висновку піксела з інтенсивністю *I* тут використовується процедура *InputPixel(x, y, I),* що не входить у стандартний набір, а реалізується додатково. Цей алгоритм для реалізації необхідних рівнів інтенсивності враховує тангенс кута нахилу (m), ваговий коефіцієнт (w) і значення помилки (e). Його можна узагальнити для всіх квадрантів тим же способом, що і для основного алгоритму Брезенхема (мал. 3.4, б).

8. Фізична модель світла

Світло - це дуже складна система, щоб змоделювати її в досконалості. Саме тому ми рідко можемо бачити створені комп'ютером тривимірні зображення, що були б по сьогоденню фотореалістичні. В усіх випадках, чим складніше і більш реалістичні створювана вами віртуальна сцена, тим більше обчислень ви повинні зробити, і тем повільніше вона буде відтворюватися на екран. Як програміст, ви повинні будете вирішити, чим ви більше готові пожертвувати: якістю чи зображення швидкістю його прорахунку на комп'ютері; чи хочете ви, щоб ваша програма привела усіх у здивування своєю красою, але вимагала майже годину для промальовування одного єдиного зображення, чи могла працювати зі швидкістю висновку 60 кадрів у секунду, але при цьому була схожа на карикатуру.

Ця лекція буде присвячений деяким з фізичних принципів, реалізація яких при комп'ютерному моделюванні навколишньої дійсності дозволить об'єктам виглядати саме так, як вони повинні виглядати. Ми так само поговоримо про деяких часто використовувану спрощеннях, що дозволяють збільшити швидкість прорахунку зображення.

Одиничний Фотон

Світло складається з дрібних згустків енергії (часток), називаних фотонами. Фотон, з одного боку, це частка, з іншого боку - хвиля, це означає, що він має властивості, властиві як хвилям, так і часткам. Ці енергетичні згустки відриваються від джерела енергії і прямолінійно поширюються в просторі, поки не відбудеться зіткнення з зовнішнім об'єктом у просторі.

Об'єкти

При зіткненні фотона з зовнішніми об'єктами може відбутися:

- відображення (reflection) - фотон відскакує від поверхні
- поглинання (absorption) - фотон поглинається і віддає свою енергію об'єкту
- переломлення (refraction) - фотон проходить крізь об'єкт і змінює напрямок руху в залежності від властивостей об'єкта й оточення
- відхилення (diffraction) --фотон може відхилитися і змінити напрямок у випадку, коли він проходить на дуже близькій відстані від поверхні об'єкта.

Сукупність фотонів

У дійсності, фотонів дуже багато. Так багато, що ми можемо сказати - їх невиразно багато. Виходячи з цього, ми можемо зневажити фактом, що світло складається з одиничних фотонів і розглянути світло як безупинний потік енергії. У цьому випадку до світла можна застосувати статистичні закони, і отримані результати будуть досить акуратні саме завдяки величезній кількості використаних фотонів. Таким чином, світло може бути (легко?) змодельований на комп'ютері.

Взаємодія світлового потоку з навколишніми предметами (об'єктами) дозволяє нам бачити їх. Світло виходить із джерела світлової енергії. Трильйони фотонів вириваються і з величезною швидкістю несуться від джерела, взаємодіючи з предметами, ударяючи кожну дрібну деталь навколишнього оточення. Невелика кількість з них попадає в маленьку темну пляму в середині нашого ока. Це зіниця. По дуже вагомій причині, суть якої буде пояснена нижче, наша зіниця чорний. Око улаштоване таким чином, що він трохи підправляє напрямок руху фотона перед тим, як він досягне задньої частини ока. Тут фотон поглинається світлочутливими рецепторами. Ці рецептори дають відповідні сигнали нашому мозку. Мозок інтерпретує послідовність сигналів, що надійшла, і постачає нас докладною інформацією про наше оточення. Зображення, що ми бачимо насправді, не є відповідним йому набором фізичних предметів. Усе, що ми одержуємо, насправді лише його енергетичний відбиток, що пройшов величезну кількість дуже складних перетворень у нашому мозку. Синій об'єкт - не є в дійсності синій. Він вважається синім тому, що ми інтерпретуємо світло, що прийшло від нього, як синій.

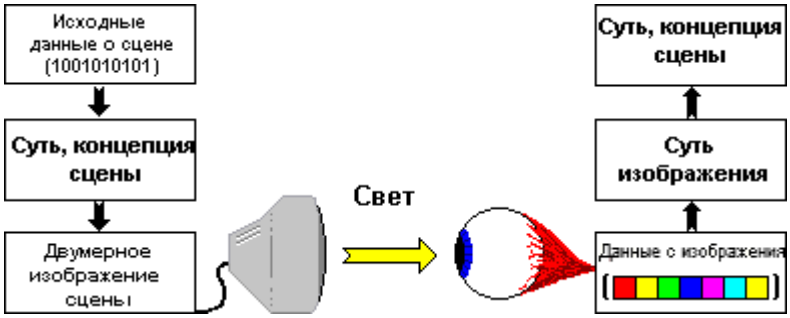
Через досвід наш мозок учиться визначати і розпізнавати безліч образів і відбитків, що створює світло про навколишню нас дійсності. Дитина бере предмет, дивиться на нього мить, потім тягне в рот. Його мова - це прекрасний датчик, і може визначати форму і вид поверхні предмета практично так само, як і око, а іноді і краще. Дитина учиться асоціювати те, що він бачить з тією формою, що йому описав мову. Згодом дитина довідається, що той самий предмет може виглядати по-різному в залежності від того, як його тримати, хоча він як і раніше є тим же самим предметом. Це очевидно - подумаєте ви, але було виявлено, що сліпим з народження людям, яким медицина повернули зір, зрозуміти вищевикладене дуже складно. Їм також складно засвоїти зміст тіні і відображення, суть яких видючі люди пізнали ще від народження. І сам факт того, що ви можете бачити, ще не означає, що ви можете зрозуміти те, що бачите.

У цьому і полягає різниця між Даними (Data) і Інформацією (Information). Дані - це світловий образ, що формується на сітківці ока. Інформація - це інтерпретація цього образа нашим мозком.

Створюючи зображення будь-якого виду, ви намагаєтеся сформувати світловий образ на сітківці ока таким чином, щоб він інтерпретувався мозком як предмет, що відображає це зображення. Тренований

мозок може витягти величезна кількість інформації з зображення. Завдяки цьому в голові ми можемо одержати повне тривимірне представлення сцени, зображеної на двомірній картинці. Щоб одержати це, наш мозок аналізує порядок взаємодії світла зі сценою (набором об'єктів зображених на картинці) і на основі такого аналізу даних видає нам кінцеве тривимірне представлення сцени.

Розмаїтість моделей висвітлення, застосовуваних у процесі формування зображень комп'ютером, -- це спроба збільшити кількість інформаціїї, що мозку зможе витягти. Коли ви, як програміст, будете писати фрагменти коду, що відповідає за графіку, вам не слід думати: "Я пишу процедуру затінення по Фонгу", замість цього вам належить міркувати так: "Я використовую візуальний трюк для коректної інтерпретації мозком".



Яку інформацію може витягти мозок?

Людський мозок може витягти й інтерпретувати 4 інформаційних ресурси з потоку видимих даних.

Форма

Це зовнішній вигляд об'єкта (предмета) у сцені, його видимі границі і краї. Око людини має здатність поліпшувати чіткість сприйманого зображення, що дозволяє впевненіше розпізнавати краю предметів; (до місця сказати, що багато комп'ютерних програм для обробки зображень використовують алгоритми, що дозволяють одержувати поліпшення чіткості, подібні тим, які робить око людини.)

Відтінки

Відблиски і тіні. Тон і структура поверхонь.

Колір

Три кольори можуть бути виявлені людським оком - червоний, зелений і синій.

Рух Мозок людини особливо сприйнятливий до руху об'єктів. Прекрасно "замаскована" тварина миттєва буде виявлено, якщо воно поворухнеться. Дуже часто, якщо ви втратили курсор на екрані монітора, кращий спосіб знайти його - рушити мишкою.

Спеціальні відділи головного мозку відповідають за обробку цих чотирьох інформаційних ресурсу. Це було неодноразово доведене у випадках аналізу черепно-мозкових травм, одержуваних людиною. Як тільки людина одержує травму і позбавляється відділу головного мозку, що відповідає за кожною з перерахованих вище ресурсів, то він відразу втрачає здатність до сприйняття цієї інформації. Наприклад, в одному випадку жінка втратила здатність відчувати рух. Вона могла бачити так само, як усі, за винятком здатності чуйно визначати рух об'єктів. Наприклад, вона могла бачити автомобілі на дорозі, але ніколи не могла сказати з першого погляду - рухаються вони чи ні.

Здатність до сприйняття приймається людиною як саме собою що розуміється. Прийнято вважати, якщо ви можете бачити, те, виходить, ви в стані визначити форму, відтінки, колір і рух. Але це не завжди так.

Вам простили…

Не менш важливої є інформація, що мозок чи додає видаляє під час аналізу. Коли ми споглядаємо, ми маємо справу з гігантськими обсягами інформації. Було б просто неможливим проаналізувати і запам'ятати всі зведення до дрібних деталей. Так це і не потрібно. Велика частина зведень (даних), що надходять нам через зір, не мають яку-небудь цінність. Мозок автоматично робить фільтрацію цього "сміття", дозволяючи нам сконцентруватися на більш значимій інформації. Що ще більш важливо, мозок також додає відсутню інформацію. Людський зір має "мертві зони", але, проте, ми цього не зауважуємо, тому що пробіли будуть завжди заповнені придатною інформацією. Наш мозок багато прощає.

Для програміста це означає те, що йому зовсім не потрібно прорисовувати зображення з точністю до дрібних деталей. Більшість з цих деталей буде просто зігнороване і "заповнено" чим - те іншим. Ваша картина може бути значно спрощена. От, наприклад, у фільмі "Повернення Джедая" зі знаменитих "Зоряних Воєн" один з космічних кораблів у просторі - це звичайний черевик. Але ніхто цього не помітив, тому що очікували бачити космічний корабель, і в тім місці дійсно був об'єкт, що нагадує його своєю формою, тому всі і бачили саме космічний корабель.

Ви можете ще більш спростити своє кінцеве зображення, якщо сцена знаходиться в русі. Натисніть паузу на відеомагнітофоні і подивитеся на нерухоме зображення, воно виглядає нікуди негідним, але ми цього не зауважуємо, коли воно в русі.

Ціль програміста, що відповідає за висновок графіки в реальному часі, - забезпечити такі процедури апроксимації у візуалізованих фрагментах коду, що поліпшують реалізм і точно передають атмосферу, дух створюваного вами світу. Інше нехай робить мозок. Ціль програміста фотореалістичної графіки -

спробувати змоделювати взаємодія світла з об'єктами сцени настільки акуратно, щоб воно могло витримати скрупульозну перевірку людським мозком.

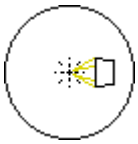
9. Властивості світла

У цій частині ми обговоримо деякі з основних принципів, що ви можете застосувати при програмуванні висновку графіки на екран.

Закон зворотної пропорційності квадрату відстані

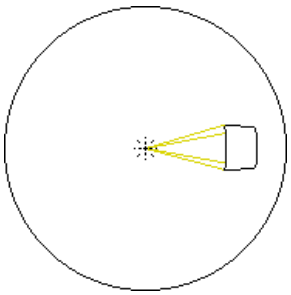
Як визначити яскравість світла?

Представте, що у вас є ідеальне джерело світла. Таке джерело не має обсягу і розмірів, а існує у виді крапки в просторі. Його можна включити і виключити, і це переключення відбувається миттєво, без утрат часу на перехідні процеси. Це саме те джерело світла, з яким можливо працювати усередині віртуального світу комп'ютера. У реальному світі такі джерела неможливі. Надалі ми також побачимо, що і реальні джерела, у свою чергу, дуже складно створити у віртуальному світі.



А тепер представимо, що ми включили джерело на дуже короткий час, коротке настільки, наскільки можна собі представити. У цей момент світло починає поширюватися в різні сторони від джерела, утворити сферу. Представимо, що ми розглядаємо невеликий фрагмент цієї сфери.

У міру того, як промені світла усе більше і більше віддаляються від джерела, розмір сфери росте, відповідно, росте і розмір досліджуваного нами фрагмента. Яскравість цього фрагмента прямо пропорційна щільності фотонів, що містяться в ньому.



Зрозуміло, якщо розмір фрагмента росте, а кількість фотонів залишається незмінним, то щільність фотонів у ньому зменшується.

Площа поверхні сфери прямо пропорційна квадрату її радіуса. Таким чином, яскравість маленького фрагмента буде назад пропорційна квадрату відстані від джерела світла.

$$Brightness = \frac{K}{r^2}$$

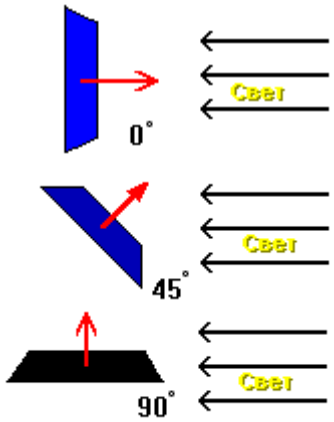
де

- Brightness - величина, що визначає яскравість (інтенсивність) світла в крапці віддаленої від джерела світла на відстані r;
- k - деяка константа, що визначає яскравість (інтенсивність) самого джерела світла.

Це і є закон зворотної пропорційності квадрату відстані. Цей закон застосуємо до всіх джерел світла, крім лазерів.

Закон косинуса

Яка освітленість поверхні?



Тепер, після того, як світло залишило джерело, воно може взаємодіяти і навколишніми предметами. Зараз ми обговоримо теорію взаємодії світла з поверхнею непрозорого предмета. Тут дуже важливо знати, як багато світла буде в будь-якій крапці на поверхні цього об'єкта.

Коли поверхня цілком звернена до світла - максимальна кількість світла досягає її. Уся поверхня освітлена.

Коли поверхня розташована під деяким кутом до падаючого на неї світла, площа перетину, зверненого до світла, стає менше. Що виражається в меншій кількості світлової енергії, що впливає на поверхню.

Коли вектор нормалі до площини поверхні знаходиться під прямим кутом до падаючого світлу, то світло просто-напросто проходить повз поверхню, і вона зовсім не висвітлюється.

Таким чином, кількість світлової енергії, що впливає на поверхню, є функція від орієнтації поверхні стосовно променів світла, що впливають.

Illumination= cos(a) * brightness ;

де:

- illumination - освітленість поверхні;
- a - кут між нормаллю до поверхні і напрямком світла;
- brightness - яскравість (інтенсивність) світла.

А що ж далі?

А от тепер припустимо, що у світла є вибір. Він може бути поглинений поверхнею, відбитий чи пропущений крізь неї.

Поглинання

Деяка кількість світла може бути поглинено поверхнею. У цьому випадку відбувається звичайне нагрівання поверхні. Остільки, оскільки ми говоримо тільки про комп'ютерні зображення, те найчастіше ми можемо просто ігнорувати це явище.

Відображення

Велика частина світла "відскочить" від поверхні. Напрямок відбитого світла до деякої міри залежить від самої

поверхні.
Якщо поверхня зовсім гладка (абсолютно блискуча), світло відіб'ється від поверхні під точно таким же кутом до нормалі, під яким кутом він до неї прийшов. При цьому нормаль буде бісектрисою кута між напрямком приходу лучачи і напрямком його відображення. Це явище можна спостерігати на дзеркальній чи полірованій металевій поверхнях. Ми зможемо помітити яскраве відображення від поверхні, тільки дивлячись на неї під визначеним кутом.
Якщо поверхня шорстка (абсолютно розсіює), то відбите світло буде поширюватися в багатьох напрямках. Тут ні в якому разі не затверджується, що в природі існують абсолютно розсіюють поверхні. Грубо оброблене дерево прекрасне розсіює світло, як і матова фарба, але обидва матеріали все-таки мають якийсь (ненульовий) блиск (shininess). Найбільш яскраве відображення від цих поверхонь буде помітно під різними кутами зору.
Більшість природних і штучних матеріалів знаходяться десь посередині між цими двома крайностями. Вони одночасно мають властивості блиску (shininess) і розсіювання (diffuse). Щоб помітити розсіяної світло від поверхні, положення ваших очей не має значення, для того, щоб помітити відблиск, кут зору повинний бути строго визначеним.

Переломлення

Коли світло проходить крізь поверхню, він проходить з одного середовища в іншу. У момент проходу через границю середовищ виникаючі квантові ефекти змушують світло змінити свій напрямок. Така зміна напряму руху світла називається переломленням (refraction). Точне значення величини кута зміни напрямку залежить від взаємного розташування поверхонь середовищ і властивості середовища за назвою коефіцієнт переломлення. Порожнеча (вакуум) має коефіцієнт, дорівнює одиниці. У повітря цей коефіцієнт трохи нижче. Більш тверді матеріали і середовища мають більш низькі коефіцієнти переломлення.
Переломлення - дуже складне явище, вимагає великих обчислювальних потужностей при його моделюванні. Для висновку на екран у реальному часі більш придатним є застосування технології ray tracing. Ми не будемо тут поглиблюватися в подробиці, усьому свій час :)

Ще далі…

Після взаємодії з поверхнею, якщо, звичайно, він не був поглинений, світло продовжує свій шлях і продовжує взаємодіяти з іншими предметами. Одиничний фотон буде продовжувати відбиватися від багатьох і багатьох поверхонь, поки остаточно не розтратить свою енергію. Ці численні ітерації складно моделювати, та й займуть вони колосальний час на візуалізацію. Роблячи рендеринг графіки в реальному часі, думають, що світло взаємодіє з поверхнею один раз.

Колір

До дійсного моменту ми говорили про однорідний світловий потік. Фактично, він і є однорідний, але може виявляти себе в нескінченній безлічі різних варіацій.

Світловий спектр

Тому що світло є ще і хвилею, то, зрозуміло, він має довжину хвилі. Довжин хвиль нескінченна безліч, але наше око в стані реєструвати тільки їхній невеликий діапазон, відомий за назвою видимої частини спектра. Узагалі ж, довжини хвиль можуть бути від дуже коротких (мільйонні частки міліметра) до наддовгих (кілометри).

Триколірна модель (RGB Model)

Людське око в стані реєструвати три основних колірних смуги в діапазоні хвиль від 400 нм (нанометрів) до 680 нм. Ми звикли ототожнювати їх з назвами наступних квітів: червоний (R), зелений(G) і синій(B). (Забудьте, якщо ви чули від художників, що існує три основних кольори, - червоний, жовтий, синій. Такий підхід актуальний тільки для барвників). Причина в наявності тільки трьох основних квітів криється не тільки в "підступ" фізиків, але й у хімічному складі органічної матерії сітківки ока, здатному реагувати тільки на визначені довжини хвиль, що відповідають цим квітам. Весь не основний кольори, такі, як жовтий чи рожевий - це просто комбінації основних квітів.
Саме ці кольори використовуються в телебаченні і висновку зображень на екран монітора. Ці три кольори дають можливість відтворити більшість квітів, що ви можете бачити. Ще раз повторимося - більшість, але не усі. Кольору, вироблені монітором, не є абсолютно чистими, тому і усі вироблені ними відтінки не можуть бути відтворені з точністю.
Більш того, якрісний діапазон моніторів сильно обмежений. Людське око в стані розрізняти набагато більше градацій яскравості. Максимальна яскравість монітора навряд чи відповідає і половині максимальної яскравості, що наше око здатне розрізнити. Це часто може привести до складностей при відображенні сцен з реального світу, що містять широкі варіації яскравості. Наприклад, фотографія пейзажу з фрагментом неба і ділянками землі, що знаходяться в повній тіні.
При моделюванні світла на комп'ютері всі три кольори обробляються окремо, за винятком яких-небудь нестандартних ситуацій, коли кольору не впливають один на одного. Іноді повноколірні зображення одержують шляхом послідовного прорахунку червоного, зеленого і синього зображень і їх подальшим комбінуванням.

Звичайно комп'ютери оперують зі світлом у виді величин, що визначають кількість вміщуваного в ньому червоного, зеленого і синього кольорів. Наприклад, білий - це рівна кількість усіх трьох, Жовтий - рівна кількість червоного і зеленого і повна відсутність синього. Усі колірні відтінки можна візуально представити у виді куба, де по осях координат будуть відкладені відповідні величини трьох вихідних квітів. Це і є триколірна світлова модель (RGB Model).
Однак є ще цілий ряд колірних моделей світла, що можуть бути навіть більш зрозумілі для деяких людей. От, наприклад, модель HSV (від англійських: -Hue - відтінок, Saturation - насиченість, Value - кількість). Снову ми бачимо три значення, виходить, усі можливі колірні відтінки можна знову укласти усередину куба.

Ця модель також іноді відома як HSL, де L - luminance, слово інше, а суть та ж.
Hue: Колір, колірний відтінок
Saturation: Колірна насиченість. Еквівалент, який відповідає.органу керування на багатьох телевізорах і моніторах.
Value: Інтенсивність. Нуль - значить чорний, більш високі значення характеризують більш яскраві значення.
(Хочу звернути увагу, щоб читач не плутав колірні моделі світла, описані в статті, з моделями передачі кольору, такими, як CMYK, наприклад. Моделі передачі кольору застосовні в основному для змішання барвників і сфера їхнього застосування - кольорова печатка і поліграфія.)

Використання світла на практиці

Ну що ж, тепер, після того, як основи світла нам відомі, ми готові йти далі.

Допущення і спрощення

Як було сказано раніше, конкретний варіант моделювання світла залежить від вимог до додатка, що ви створюєте. Існує цілий ряд допущень, які можна застосувати для того, щоб збільшити швидкість прорахунку і висновку на екран.

Крапкові джерела світла

Для спрощення математичних розрахунків джерела світла звичайно розглядають у виді крапки в просторі. У переважній більшості випадків це буде не занадто далеко від реальності. Лампочки і ліхтарі на вулицях дуже малі в порівнянні з об'єктами, що вони висвітлюють. Проблема виникає тоді, коли ви хочете зобразити сцену з джерелом світла у виді довгої люмінесцентної чи лампи сцену, рівномірно освітлювану природним небесним висвітленням. У цьому випадку вам доведеться застосувати групу у виді декількох, більш слабких джерел - для того, щоб вони могли імітувати один великий.

Багаторазові відображення

Прорахунки ефектів, вироблених світлом при відображенні від однієї поверхні на іншу, тривалі і складні. Тому для великих просторів ми можемо не прораховувати множинні відображення, через те, що різниця між однократним і багаторазовим відображенням, у космосі, наприклад, зовсім непомітна. Інша справа, якщо ми моделюємо світло в маленької кімнаті. Тут ця різниця буде більш, чим помітна, тому що об'єкти, що знаходяться в зоні безпосередньої тіні, будуть усе рівно освітлені за рахунок відбитих променів від інших поверхонь.

Тіні

Незважаючи на те, що тіні можуть дати спостерігачу додаткову інформацію про глибину сцени, їхня відсутність іноді може бути незначною втратою. У залежності від ситуації часто можливо зробити внести спрощення в задачу прорахунку тіней. Наприклад, ви працюєте над авіасимулятором. У цьому випадку спостереження тіні літака може бути дуже важливим індикатором висоти польоту. Але навколишній світ містить всього одне джерело світла - сонце, а інші предмети дуже малі і розкидані. Таким чином, вам не треба думати над імітацією тіней на самому літаку і кожнім маленькому будиночку далеко внизу на землі. Вам досить спроектувати тінь тільки на площину землі.

Статичні (нерухомі) тіні

Сцени з нерухомими джерелами світла й об'єктами мають статичні тіні. Мається можливість заздалегідь прорахувати всі тіні в сцені. А потім використовувати цю інформацію для швидкого промальовування цих тіней на екрані. Гарним прикладом подібного підходу є ігри типу Quake. Рівень заздалегідь обраховувався утилітами прорахунку висвітлення, і в реальному часі движок гри вже не витрачав дорогоцінний процесорний час на їхнє створення. Усі тіні зберігалися у виді "карти тіней" у самому файлі рівня й у процесі гри комбінувалися з відповідними текстурами.

10. МЕТОДИ ЗАПОВНЕННЯ

Формування складних графічних зображень у ряді випадків вимагає реалізації зафарбування областей площини і поверхонь.

Існує два основних види зафарбування:

- зафарбування області площини, заданої контуром одним кольором з постійним рівнем яскравості;
- зафарбування поверхонь різними кольорами чи різними рівнями яскравості з урахуванням взаємного положення й орієнтації цих поверхонь і джерел світла.

Перший вид зафарбування прийнятий називати *заповненням області.* При цьому область може бути задана обмежуючим її багатокутником чи сукупністю крапок (піселів) визначеної яскравості і кольору на дискретній площині.

Існуючі в даний час методи заповнення областей засновані на двох основних принципах:

- *перевірці на парність;*
- *критерії зв'язаності.*

Методи, що використовують перевірку на парність, базуються на тім, що будь-яка пряма перетинає контур, що обмежує область, парне число раз. Знаючи, що перша крапка прямої, що перетинає контур, лежить поза областю, зробивши прохід по цій лінії, можна шляхом підрахунку кількості перетинань визначити відрізки, розташовані поза і усередині області. Якщо кількість перетинань парне, то відрізок лежить поза областю, якщо непарне – усередині.

Методи, засновані на критерії зв'язаності, припускають завдання крапки, що лежить усередині області. Дана крапка називається *затравочною.* Ці методи реалізують обхід площини, на якій знаходиться заповнювана область, з метою перебування пікселей, які досяжні затравочною крапкою.

Багато замкнутих контурів є або простими багатокутниками, або їхній можна апроксимувати придатними багатокутниками. Найпростіші методи заповнення багатокутника базуються на приналежності пікселов растра внутрішньої області багатокутника. Ці методи характеризуються великими обчислювальними витратами. Більш ефективні алгоритми растрового розгорнення суцільних областей називаються алгоритмами з упорядкованим списком ребер. Ефективність цих алгоритмів залежить від ефективності сортування крапок перетинань ребер багатокутника зі скануючими рядками.

Розглянемо метод і відповідний йому алгоритм заповнення області, обмеженої сторонами багатокутника, що використовує принцип парності. Будемо вважати, що сторони багатокутника задані координатами вершин багатокутника x_i, y_i, яким відповідає максимальне значення *в* чи мінімальне значення *x,* якщо сторона горизонтальна, а також збільшеннями Δx, Δy для визначення координат іншої вершини.

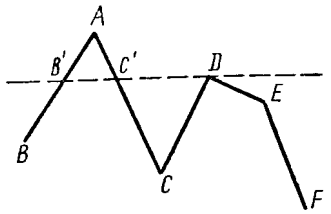


Рис. 4.8. Заполнение области по критерию четности

Першим **кроком алгоритму** є сортування сторін відповідно до максимальних значень y_i, а при їхній рівності по мінімуму x_i. У випадку рівності і x_i, використовуються значення Δx, а потім, Δy. При цьому вибирається сторона з найменшим алгебраїчним значенням. У результаті одержуємо упорядкований список сторін контуру – список черговості.

Другий **крок алгоритму** виконується для всіх суміжних сторін списку з однаковими значеннями *y*, що відповідає локальному максимуму. Через вершину багатокутника, що відповідає цим сторонам, проводиться пряма, рівнобіжна осі *X*, і визначаються крапки її перетинання з іншими

Таблиця 4.2

Описание границ, заполняемой области

Сторона:	X	Y	X	Y	Метка 1	Метка 2	Метка 3
A ***B***	x_A	y_A	$x_B - x_A$	$y_B - y_A$	2	y_D	1
AC	x_A	y_A	$x_C - x_A$	$y_C - y_A$	1	–	0
DC	x_D	y_D	$x_C - x_D$	$y_C - y_D$	2	–	0
DE	x_D	y_D	$x_E - x_D$	$y_E - y_D$	1	–	1
EF	x_E	y_E	$x_F - x_E$	$y_F - y_E$	1		1
…	…	…	…	…	…	…	…

сторонами, що розташовані в списку вище розглянутих. Дані прямі поділяють контур на шари, що містять парне число границь. Кожному шару відповідає спеціальний список, називаний поточним, котрий містить границі; які визначають цей шар. Для формування поточного списку список черговості включає мітки трьох типів. Табл. 4.2 ілюструє формування списку черговості і міток для одержання поточного списку на прикладі фрагмента області, приведеної на мал. 4.8.

Мітка *M1* визначає число границь, переданих зі списку черговості в поточний список: дві – якщо крапка відповідає максимуму; одна – якщо вона не є максимумом; більш двох, якщо одному значенню у відповідає більш одного максимуму.

Мітка *M2* служить для введення в поточний список наступної пари границь при переході в наступний шар. Її значення відповідає значенню *y*, при який цей перехід повинний відбутися. Зазначеними мітками мітяться перші сторони зі списку черговості, з якими перетинаються прямі, проведені через вершини багатокутника, що є локальними максимумами. При це значення міток дорівнює координаті *y* вершини, з якої проведена пряма.

Мітка М3 визначає число нових границь, до яких виробляється звертання при завершенні роботи з поточною границею.

Третьим шагом алгоритма является непосредственное заполнение области последовательно для каждого слоя. Этот процесс начинается с первой пары границ списка очередности путем формирования и обновления текущего списка, используя имеющиеся метки, которые позволяют удалять и включать новые границы. Организация текущего списка дает возможность реализовать для каждого слоя принцип четности, в связи с тем, что все локальные максимумы определены. Необходимо отметить, что приведенный алгоритм не учитывает наличие горизонтальных сторон многоугольника. Однако модификация алгоритма для учета этих ситуаций не представляет существенных трудностей.

Трудности возникают при использовании принципа четности при разработке алгоритма заполнения области, заданной на дискретной плоскости. Это связано с необходимостью определения точек касаний контура, а также нахождения пересечений прямых и кривых, что является нетривиальной задачей. При необходимости с соответствующими алгоритмами можно ознакомиться в специальной литературе, посвященной вопросам машинной графики [17, 21, 26].

Рассмотрим теперь метод и алгоритм, работающие по критерию связности. Как было отмечено, данный метод предполагает наличие затравочного пиксела, лежащего внутри контура. Для его задания существует несколько способов. В ряде режимов он может быть указан оператором. В других случаях в качестве затравочного пикселя можно взять точку непосредственно смежную с пикселем контура. Что касается задания контура, то он, как правило, определяется совокупностью точек поэлементного эквивалента изображения. В случае задания контура многоугольником предварительно осуществляется процедура получения его поэлементного эквивалента с помощью генератора векторов. Положим, что контур и затравочный пиксел сгенерированы, причем имеют уровень яркости или цвет, необходимый для заполнения области.

Простейший рекурсивный алгоритм заключается в рассмотрении четырех соседних к затравочному пикселу элементов на предмет их принадлежности контуру. Пикселы, не принадлежащие контуру, в свою очередь могут использоваться в качестве затравочных.

Приведем пример простого рекурсивного алгоритма для заполнения 4-связной области цветом color1 до границы цвета color2.

```
Procedure Flood_Fill_4(x, y, color1, color2:integer);
Begin
 If color2<>GetPixel(x, y) and color1<>GetPixel(x, y) then
        Begin PutPixel(x, y, color1);
          Flood_Fill_4(x, y-1, color1,color2);
          Flood_Fill_4(x, y+1, color1,color2);
          Flood_Fill_4(x-1, y, color1,color2);
          Flood_Fill_4(x+1, y, color1,color2)
          end
End;
```

Нетрудно прийти к выводу, что этот алгоритм при своей внешней простоте требует значительного объема вычислений. Это связано в первую очередь с дублированием рассмотрения соседних элементов для соседних затравочных пикселей.

Альтернативный нерекурсивный алгоритм состоит в следующем. Начиная с затравочного пиксела производится анализ на принадлежность пикселей контуру, которые находятся влево, а затем вправо от затравочного с одновременной закраской. Далее осуществляется переход на нижнюю сторону с повторением процедуры. Этот процесс продолжается до полного заполнения нижней по отношению к затравочному пикселу части области. Затем проводится заполнение верхней части области.

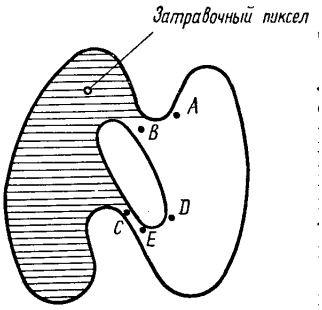


Рис. 4.9. Заполнение области по критерию связности

В случае заполнения невыпуклых областей необходимо реализовать стек, в котором будут храниться пикселы, расположенные вблизи максимумов и минимумов контура. Указанные пикселы используются в качестве затравочных, последовательно извлекаясь из стека. Очевидно, что задача нахождения этих пикселей не составляет особых проблем.

На рис. 4.9 приведен пример, иллюстрирующий реализацию метода. Пикселы *A, B* и *C* помещаются в стек при заполнении заштрихованной области, *D* и *E* — при заполнении незаштрихованной части.

Более детально этот метод и реализующий его алгоритм рассмотрены в [21, 26], где также представлены различные комбинации методов и алгоритмов и приведен их сравнительный анализ.

11. МЕТОДИ ЗАФАРБУВАННЯ

Однією з визначальних проблем на шляху створення реалістичних зображень є проблема зафарбовування поверхонь, що обмежують побудовані об'єкти. Ми зупинимося на описі деяких найпростіших моделей, що вимагають порівняно невеликих обчислювальних витрат.

Світлова енергія, що падає на поверхню від джерела світла, може бути поглинена, відбита і пропущена. Кількість поглиненої, відбитої і пропущеної енергії залежить від довжини світлової хвилі. При цьому колір поверхні об'єкта визначається довжинами хвиль, що поглинаються.

Властивості відбитого світла залежать від форми і напрямку джерела світла, а також від орієнтації освітлюваної поверхні і її властивостей. Світло, відбитий від об'єкта, може бути дифузійним і дзеркальної: дифузно відбите світло розсіюється рівномірно в усіх напрямках, дзеркальне відображення походить від зовнішньої поверхні об'єкта.

Світло крапкового джерела відбивається від ідеального розсіювача за законом косинусів Ламберта:

$$I = I_1 k_d \cos\theta \ , \tag{1}$$

де I - інтенсивність відбитого світла; I_1 - інтенсивність точечного джерела; k_d - коефіцієнт дифузійного відображення (постійна величина, $0 \le k_d \le 1$); θ - кут між напрямком на джерело світла і (зовнішньої) нормаллю до поверхні ($0 \le \theta \le \frac{\pi}{2}$, мал. 1).

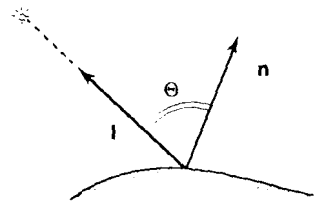


Рис. 1

На обєкти реальних сцен падає ще і розсіяному світлу, відповідному відображенню світла від інших об'єктів. Оскільки точний розрахунок розсіяного висвітлення вимагає значних обчислювальних витрат, у комп'ютерній графіці при обчисленні інтенсивності надходять так :

$$I = I_a k_a + I_1 k_d \cos\theta \tag{2}$$

де I_a - інтенсивність розсіяного світла; k_a - (постійний) коефіцієнт дифузійного відображення розсіяного світла, $0 \le k_a \le 1$.

Інтенсивність світла, природно, залежить від відстані d від об'єкта до джерела світла. Для того щоб врахувати це, користаються наступною моделлю висвітлення:

$$I = I_a k_a + \frac{I_1 k_d \cos\theta}{d + K} \tag{3}$$

де K - довільна постійна.

Інтенсивність дзеркально відбитого світла залежить від кута падіння, довжини хвилі і властивостей речовини. Тому що фізичні властивості дзеркального відображення досить складні, то в простих моделях висвітлення звичайно користаються наступною емпіричною моделлю (моделлю Фонга):

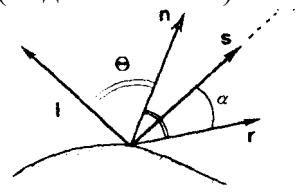


Рис. 2

$$I_S = I_1 k_S \cos^P \alpha \ , \tag{4}$$

де k_S - експериментальна постійна; α - кут між відбитим променем і вектором спостереження; p - ступінь, що апроксимує просторовий розподіл світла (мал. *2*).

Поєднуючи останні дві формули, одержуємо модель висвітлення (функцію зафарбування), використовувану для розрахунку інтенсивності (чи тону) крапок поверхні об'єкта (чи пикселів зображення):

$$I = I_a k_a + \frac{I_1}{d + K}(k_d \cos\theta + k_S \cos^P \alpha) \tag{5}$$

Функцію зафарбування, використовуючи одиничні вектори зовнішньої нормалі n, а також одиничні вектори, що визначають напрямки: на джерело (вектор l), відбитого променя (вектор r) і спостережень (вектор s), можна записати в наступному виді:

$$I = I_a k_a \frac{I_1}{d + K}\left\{k_d (n - l) + k_S (r - s)^P\right\} \tag{6}$$

Зауваження: *Щоб одержати кольорове зображення, необхідно знайти функції зафарбування для*

кожного з трьох основних квітів - червоного, зеленого і синього. Оскільки колір дзеркально відбитого світла визначається кольором падаючого, те постійна k_S вважається однакової для кожного з цих квітів.

Якщо крапкових джерел світла трохи, скажемо m, то модель висвітлення визначається так

$$I = I_a k_a + \sum_{j=1}^{m} \frac{I_{1j}}{d+K}(k_d \cos\theta_j + k_S \cos^{Pj} \alpha_j) \ . \qquad (7)$$

Якщо освітлювана поверхня в розглянутій крапці гладка (має дотичну площину), то вектор зовнішньої нормалі обчислюється безпосередньо (мал. 3). У випадку багатогранної поверхні вектори зовнішніх нормалей можна знайти тільки для її граней. Що стосується напрямків векторів зовнішніх нормалей на ребрах і у вершинах цієї поверхні, те їхнього значення можна знайти тільки приблизно.

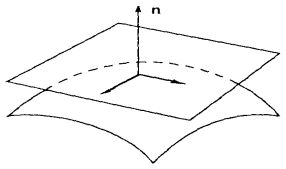


Рис. 3

Нехай, наприклад, m граней, що сходяться в даній вершині, мають нормалі N_i, де $i = 1,...,m$, є векторами зовнішніх нормалей для розглянутої багатогранної поверхні (мал. 4).

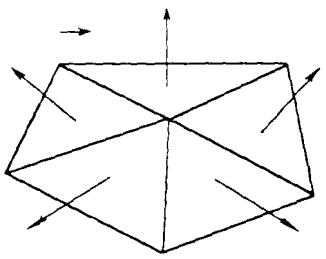


Рис. 4

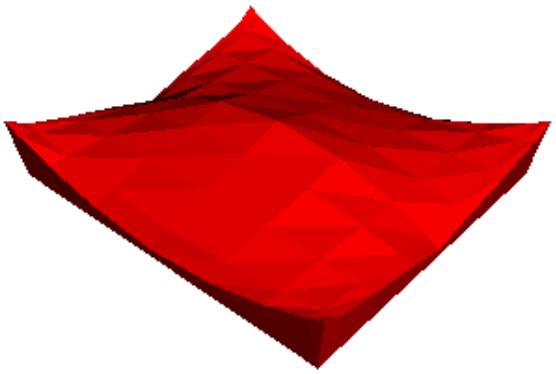
Вектор у вершині одержуємо шляхом усередненням нормалей граней:

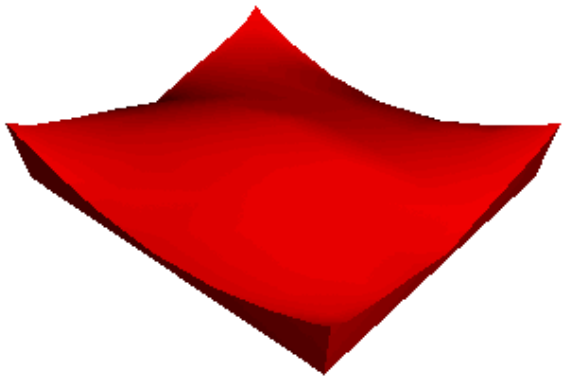
$$N = \frac{1}{m}\sum_{i=1}^{m} N_i \ .$$

12. Метод Гуро

Затінення по Гуро -- це метод лінійної інтерполяції освітленості в межах одного полігона. Він був винайдений у 1971 і має ім'я свого винахідника. Це простий і ефективний метод додання відчуття зігнутості для рівного полігона. Цей метод також часто використовується для скорочення глибини промальованої сцени шляхом імітації зникнення вилучених об'єктів у тумані.

Полігоном у тривимірній графіці прийнято вважати ділянка поверхні, що має як мінімум три вершини лежачі в одній площині. Власне кажучи, полігон може мати необмежена кількість вершин, але в тривимірній комп'ютерній графіці в переважній більшості випадків розроблювачі використовують саме трикутні полігони. По-перше, три вершини завжди однозначно визначають положення площини в просторі, а четверта вершина може лежати поза межами площини, і її положення прийдеться завжди перевіряти, що викликає додаткових труднощів. По-друге, із трикутників завжди можна одержати будь-який інший багатокутник..





Ліворуч ви бачите вигнуту поверхню (що складається з безлічі трикутних полігонів, чи попросту - трикутників), намальовану на екран звичайним образом, де кожна трикутна грань (полігон) має рівномірну освітленість(flat shading), праворуч та ж поверхня, але з застосуванням освітленості Гуро. Безсумнівно, другий варіант виглядає куди більш привабливо, тому розглянемо один з варіантів його реалізації.

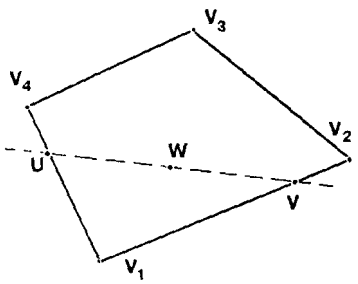


Рис. 10

На *відміну від рівномірного заповнення, у випадку методу Гуро ви насамперед зобов'язані визначити і прорахувати одним з відомих способів приведений вектор нормалі для кожної вершини полігона. Одержавши нормаль,*

ми можемо визначити (ступінь чи затінення освітленість) у вершинах, а вже потім нам залишається тільки провести інтерполяцію освітленості по полігоні.

Розглянемо більш докладно один з найбільш простих методів зафарбування - метод Гуро, що ґрунтується на визначенні освітленості грані в її вершинах з наступної білінейною інтерполяцією величин, що вийшли, на всю грань.

Звернемося до мал. 10, на якому зображена опукла чотирикутна грань. Припустимо, що інтенсивності в її вершинах V_1, V_2, V_3, V_4 відомі і рівні відповідно I_{V1}, I_{V2}, I_{V3}, I_{V4}.

Нехай W - довільна крапка грані. Для визначення .інтенсивності (освітленості) у цій крапці проведемо через неї горизонтальну пряму. Позначимо через U і V крапки перетинання проведеної прямої з границею грані.

Будемо вважати, що інтенсивність на відрізку UV змінюється лінійно, тобто

$I_W = (1-t)I_U + tI_V$, де $\mathrm{t} = \dfrac{|\mathrm{X_U} - \mathrm{X_W}|}{|\mathrm{X_U} - \mathrm{X_V}|}$, $0 \le \mathrm{t} \le 1$.

Для визначення інтенсивності в крапках U і V знову скористаємося лінійною інтерполяцією, також вважаючи, що уздовж кожного з ребер границі інтенсивності змінюються лінійно.

Тоді інтенсивність у крапках U і V обчислюється по формулах:

$I_U = (1-u)I_{V4} + uI_{V1}$,

$I_V = (1-v)I_{V1} + vI_{V2}$,

де $\mathrm{u} = \dfrac{|\mathrm{Y_{V4}} - \mathrm{Y_U}|}{|\mathrm{Y_{V4}} - \mathrm{Y_{V1}}|}$, $0 \le \mathrm{u} \le 1$, $\mathrm{v} = \dfrac{|\mathrm{Y_{V1}} - \mathrm{Y_V}|}{|\mathrm{Y_{V1}} - \mathrm{Y_{V2}}|}$, $0 \le \mathrm{v} \le 1$.

Метод Гуро забезпечує безупинна зміна інтенсивності при переході від однієї грані до іншої без розривів і стрибків.

Ще однією перевагою цього методу є його інкрементальний характер: грань малюється у виді набору горизонтальних відрізків, причому так, що інтенсивність наступного піксела відрізка відрізняється від інтенсивності попереднього на постійну величину для даного відрізка. Крім того, при переході від відрізка до відрізка значення інтенсивності в його кінцях також змінюються лінійно.

Таким чином, процес малювання грані складається з наступних кроків:

1) проектування вершин грані на екран і обчислення нормалей для кожної грані;

2) обчислення нормалей у вершинах граней як усереднення нормалей граней;

3) відшукання інтенсивностей у вершинах відповідно до обраної моделі висвітлення;

4) визначення координат кінців чергового відрізка і значень інтенсивності в них лінійною інтерполяцією;

5) малювання відрізка з лінійною зміною інтенсивності між його кінцями.

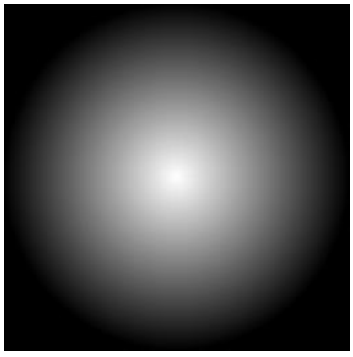
Зауваження:

1. *При визначенні освітленості у вершині, природно, устає питання про вибір нормалі. Часто як нормаль у вершині вибирається нормована сума нормалей прилягаючих граней*

   $n = \dfrac{a_1 n_1 + ... + a_k n_k}{|a_1 n_1 + ... + a_k n_k|}$, *де* $\mathrm{a_1}$,..... $\mathrm{a_k}$ - *довільні вагарні коефіцієнти.*

2. Дефекти зображення, що виникають при зафарбуванні Гуро, частково порозуміваються тим, що цей метод не забезпечує гладкості зміни інтенсивності.

13. Метод Фонга



Апроксимації Фонга, чи зафарбування по Фонгу (Phong shading), звуться по імені свого винахідника - Ву Тонг Фонга (Wu Tong Phong). Цей метод дає більш точні результати при затіненні полігонів у порівнянні з затіненням Гуро, розглянутим вище. Суть цих апроксимацій полягає у визначенні нормалі до поверхні в кожній вершині полігона і подальшої інтерполяції вектора по всьому полігоні поверхні. Далі для кожного пікселя необхідно обчислити значення яскравості, ґрунтуючись на значеннях вектора нормалі.

Насправді, цей процес займає дуже багато часу, і навряд чи його можна вважати одним зі зручних методів прорахунку затінення для

ігор, розрахованих на середній домашній комп'ютер. Тому більшість комп'ютерних програм, що обіцяють використання затінення по Фонгу в реальному часі, насправді використовують "підробку" - значно спрощений варіант апроксимацій. Такий варіант дозволяє досягти прийнятних результатів і при цьому він значно швидше.

Таким чином, як і описаний вище метод зафарбування Гуро, зафарбування Фонга при розрахунку інтенсивності також спираються на інтерполяцію. Однак на відміну від методу Гуро тут інтерполюється не значення інтенсивності по уже відомих її значеннях в опорних крапках, а значення вектора зовнішньої нормалі, що потім використовується для обчислення інтенсивності піксела. Тому зафарбування Фонга вимагає помітно більшого обсягу обчислень. Правда, при цьому і зображення виходить більш близьким до реалістичного (зокрема, при зафарбуванні Фонга дзеркальні відблиски виглядають досить правдоподібно).

Метод Фонга полягає в побудові для кожної точки вектора, що грає роль вектора зовнішньої нормалі, і використанні цього вектора для обчислення освітленості в розглянутій точці по формулі (5). При цьому схема інтерполяції, використовувана при зафарбуванні Фонга, аналогічна інтерполяції в зафарбуванні Гуро.

Для визначення вектора "нормалі" n_W у крапці W проводимо через цю крапку горизонтальну пряму і, використовуючи значення векторів "нормалей" n_U і n_V в крапках її перетинання U і V з ребрами грані, одержуємо

$$\mathrm{n_W} = \frac{(1-\mathrm{t})\mathrm{n_U} + \mathrm{tn_V}}{\left|(1-\mathrm{t})\mathrm{n_U} + \mathrm{tn_V}\right|}\ ,\ \text{де}\ \mathrm{t} = \frac{\left|\mathrm{X_U} - \mathrm{X_W}\right|}{\left|\mathrm{X_U} - \mathrm{X_V}\right|},\ 0 \le \mathrm{t} \le 1$$

а вектори зовнішніх нормальний у крапках U і V знаходяться, у свою чергу (також лінійною інтерполяцією), по векторах нормалей у кінцевих крапках відповідних ребер розглянутої багатокутної грані:

$n_U = (1-u)n_{V4} + un_{V1}$,

$n_V = (1-v)n_{V1} + vn_{V2}$,

$$\text{де}\ \mathrm{u} = \frac{\left|\mathrm{Y_{V_4}} - \mathrm{Y_U}\right|}{\left|\mathrm{Y_{V_4}} - \mathrm{Y_{V_1}}\right|},\ 0 \le \mathrm{u} \le 1\ ,\ \ \mathrm{v} = \frac{\left|\mathrm{Y_{V_1}} - \mathrm{Y_V}\right|}{\left|\mathrm{Y_{V_1}} - \mathrm{Y_{V_2}}\right|},\ 0 \le \mathrm{v} \le 1.$$

Нормування вектора n_W необхідно в наслідку того, що у формулах (1)-(5) використовується одиничний вектор нормалі.

Зауваження:

1. Як і метод Гуро метод Фонга також у значній мірі носить інкрементальний характер.

2. Застосовуючи метод Фонга, ми фактично будуємо на багатогранній моделі безупинне поле одиничних векторів, використання якого в якості полючи зовнішніх нормалей забезпечує гладкість одержуваного зображення.

3. Ясно, що вимоги до якості зображення прямо зв'язані з точністю розглянутої моделі й обсягом відповідних їй обчислень. Безсумнівним достоїнством запропонованих моделей зафарбування (Гуро і Фонга) є їхня порівняльна простота. Однак унаслідок значних спрощень одержуваний результат не завжди виявляється задовільним. Переборювати цей бар'єр якості найкраще шляхом використання більш зроблених моделей і методів.

4. Незважаючи на те, що затінення Гуро далеко від ідеалу, воно все-таки дозволяє створювати значно більш реалістичні поверхні. Недоліки затінення Гуро починають виявлятися тоді, коли ви намагаєтеся сполучити обчислення освітленості великих по розмірі полігонів. Освітленість, створювана одним джерелом світла у вершинах полігона, буде дуже малої через значну далекість від джерела світла. Виведений на екран полігон буде темним, хоча це не правильно, адже центр полігона - освітлений! Цей недолік ви можете помітити, наприклад, у грі Descent. Спалаху від пострілів яскраво висвітлюють кути стін і практично не впливають на зміну висвітлення, якщо ви вистрілюєте в середину чи стіни підлоги. (І навпаки, великий полігон буде завжди цілком освітлений якщо з його протилежних країв коштує по джерелу світла, навіть *незважаючи на те, що джерела малопотужні і середина полігона просто зобов'язана бути в тіні)*

Однак, якщо ви використовуєте велику кількість невеликих полігонів разом з віддаленим джерелом світла, то затінення Гуро може виглядати дуже і дуже непогано. Практично - чим

менше розмір полігона, тим точніше і більш якісне буде згладжування, тим більше воно буде схоже по якості на затінення по Фонгу.

5. Затінення по Фонгу - це досить популярний метод для реалізації зафарбовування, особливо в програмах, що застосовують non-realtime rendering, наприклад, 3D Studio. Цей метод значно точніше затінення Гуро. Зафарбування по Фонгу -і це логічна реалізація фізичної моделі світла, що ми розглянули раніше. Кожен одиночний піксел має своє власне значення яскравості, ретельно перелічене при використанні інтерпольованого вектора нормалі.

7. Моделювання текстур і інших елементів реалістичних зображень

Для додання більш природного виду (реалістичних зображень) сцені бажано мати можливість змінювати параметри поверхні (у найпростішому випадку - колір) у залежності від положення крапки на ній. Нижче будуть розглянуті різні способи досягнення цього.

Для досягнення реалістичності відображення поверхонь застосовуються різні методи:

-текстурірованіе (texture mapping);
- (mip mapping) - предфільтрація bitmap текстур зі зменшеним дозволом.
- фільтрація (звичайно bi- чи tri-лінійна);
- anti-aliasing – це спосіб обробки (інтерполяції) пікселов для одержання більш чітких границь зображення;
- fogging (затуманення);
- alpha blending - спосіб передачі інформації про прозорість об'єктів.
- double buffering – спосіб промальовування зображення, що випереджає.

Текстурирование - це самий істотний і найбільш розповсюджений метод для моделювання поверхонь. Наприклад, фасад будинку зажадав би відображення безлічі граней для моделювання цеглин, вікон і двер. Однак текстура (зображення, що накладається на всю поверхню відразу) дає більше реалізму. При цьому не потрібно великих обчислювальних витрат, тому що дозволяє оперувати з усім фасадом як з єдиною поверхнею. Перш ніж зобразити поверхні на екрані, вони попередньо текстуріруються і розрахується їхня освітленість. Усі текстури зберігаються в пам'яті, звичайно встановленої на відеокарті. До речі, тут слід зазначити, що застосування AGP уможливлює збереження текстур у системній пам'яті, а її обсяг набагато більше.

Очевидно, що коли поверхні текстуріруються, необхідний облік перспективи, наприклад, при відображенні дороги з розділовою смугою, що іде за обрій. Перспективна корекція необхідна для того, щоб текстуровані об'єкти виглядали правильно. Вона гарантує, що bitmap правильно наляже на різні частини об'єкта - і ті, котрі ближче до спостерігача, і на більш далекі. Корекція з урахуванням перспективи дуже трудомістка операція, тому нерідко можна зустріти спрощену її реалізацію.

Існують різні способи моделювання текстури, але практично усі вони підрозділяються на два основних класи:

- проективні текстури:
- процедурні (суцільні - solid) текстури.

Уявимо собі, що необхідно задати визначену текстуру (наприклад, мармур) якому-небудь об'єкту. Можливі два шляхи:

1. Узяти зображення реальної мармурової поверхні і відобразити (спроектувати) його яким-небудь образом на поверхню об'єкта. Тобто перевести вихідні тривимірні координати точки в двовимірні і використовувати останні для індексації в зображення.

2. Побудувати деяку функцію Color(x, y, z), що визначає для кожної крапки простору (x, y, z) колір таким чином, щоб об'єкт, колір якого задається цією функцією, мав вид об'єкта, зробленого з мармуру.

Перший шлях відповідає проективним текстурам. Він найбільш простий, однак має цілий ряд істотних недоліків: вимагає великого обсягу пам'яті для збереження використовуваних зображень, має порівняно невелику гнучкість і до того ж сполучений з великими складностями в підборі способу проектування для об'єктів складної форми.

Тому в практичних задачах, як правило, використовується лише невелика кількість стандартних варіантів проектування: плоске (рівнобіжне проектування уздовж заданого напрямку), циліндричне і сферичне. Для параметрично заданих поверхонь часто як проекцію крапки ($x(u,v)$, $y(u,v)$, $z(u,v)$) виступають значення параметрів (u,v).

Другий шлях не вимагає великих витрат пам'яті й однаково добре працює з об'єктами кожної (як завгодно складної) форми. Оскільки подібна функція звичайно залежить від великої кількості параметрів, то, змінюючи їх, можна легко змінювати параметри текстури. Основним недоліком цього підходу є складність підбора відповідної функції.

Розглянемо більш докладно проективні структури.

15. Проективні текстури (projective texture)

У загальному випадку текстура попередньо проектується на поверхню, а потім проектується на 2-х мірний екран. Іншими словами, спочатку проектується проектором деяке зображення на поверхню, а потім дивимося на неї з довільної крапки (див. мал.1. При побудові зображення ця ситуація моделюється вкрай просто - проекція примітивів поверхні на екран справа звичайне, а роль другої проекції (проектування зображення на поверхню) грає прив'язка відповідного місця текстури з зображенням на примітиви.

Нам залишилося лише навчитися правильно прив'язувати текстуру з вихідним зображенням до нашої поверхні.

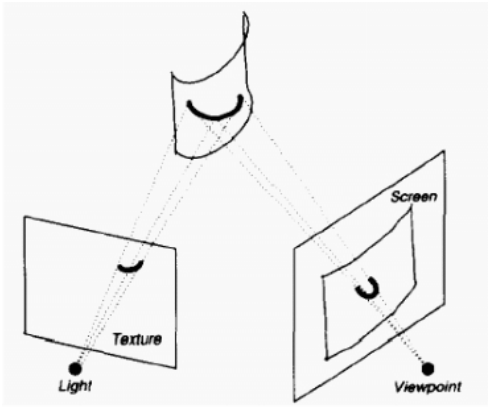


Рис. 1

Усього ми маємо справу з чотирма координатними системами.

1. Спостережницька система ("clip" чи "projection") - є звичайним для графіки 4-х координатним представленням 3-х мірного (об'ємного) простору. Нехай початок цієї координатної системи знаходиться в точці спостереження, а координати точки в цій системі позначимо $(\mathrm{x},\mathrm{y},\mathrm{z},\mathrm{w})$.

2. Екранна система ("screen") - 2-х мірний екран, що бачить спостерігач. Ці координати виходять зі спостережницької системи шляхом розподілу x і y на w, тобто $\mathrm{x}^s = \mathrm{x}/\mathrm{w}$, $\mathrm{y}^s = \mathrm{y}/\mathrm{w}$, де індекс "s" позначає екранну систему).

3. Система джерела світла ("light") - це друга об'ємна система координат (x^l, y^l, z^l, w^l). На початку цієї системи координат знаходиться джерело світла.

4. Текстурна система (texture) - координати на площині проектованої текстури (той слайд, крізь яку світить уявлюване джерело світла). Текстурні координати виходять як $x^t = x^l / w^l$, $y^t = y^l / w^l$, (також можна обчислити $z^t = z^l / w^l$, якщо ми вирішили не обмежуватися плоскою текстурою).

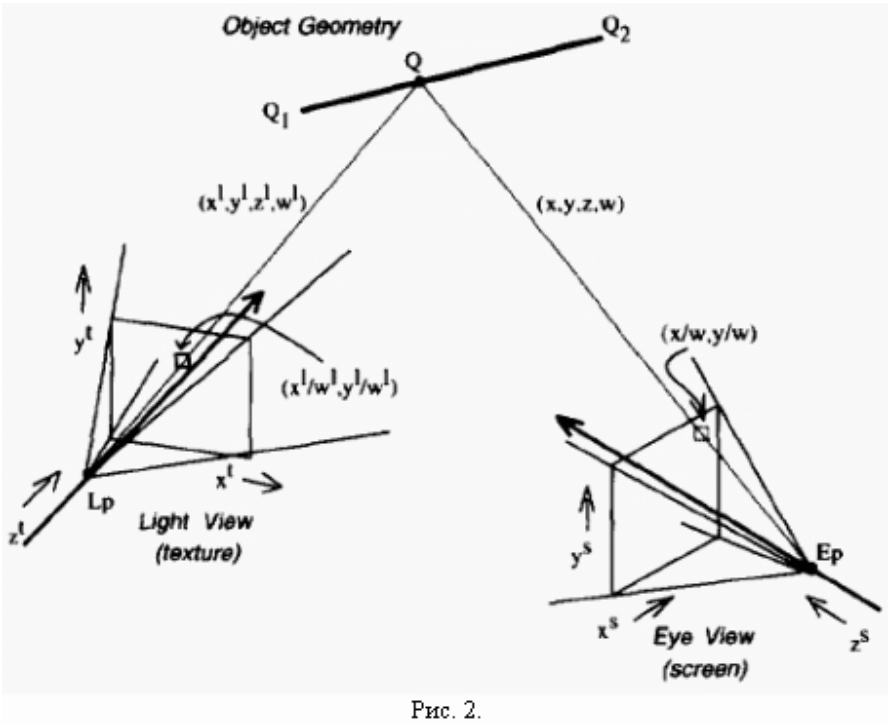


Рис. 2.

Наша задача: маючи точку (x^s, y^s) на екрані, нам необхідно знайти відповідну їй точку (x^t, y^t) на текстурі.

На мал. 2 показаний сегмент лінії в нашому тривимірному просторі і його проекції на 2-х мірний екран. Цей сегмент - горизонтальна смуга сканування на екрані, розташована між двома ребрами полігона. Координати його кінців у спостережницькій системі:

Нам необхідно знайти координати нашої довільної точки відрізка в координатній системі джерела світла. Будемо вважати, що, так чи інакше, ми уже визначили координати кінців відрізка в системі джерела

$Q_1 = (x_1, y_1, z_1, w_1)$ и $Q_2 = (x_2, y_2, z_2, w_2)$

Произвольная точка Q на этом отрезке может быть задана параметрическим образом:

$$Q = (1-t)Q_1 + tQ_2 \qquad (1)$$

для $t \in [0,1]$. В экранной системе координат (*screen*) эта точка задется следующим образом:

$$Q^s = (1-t^s)Q_1^s + t^s Q_2^s \qquad (2)$$

где

$Q_1^s = Q_1/w_1$ и $Q_2^s = Q_2/w_2$ (общее правило преобразования из наблюдательской системы в экранную)

світла. Для початку нам необхідно знайти параметр t , відповідний t^s (у загальному випадку екранне $t \neq t^s$ спостережницькому).

Для цього запишемо

$$Q^s = \frac{(1-t^s)Q_1}{w_1} + \frac{t^s Q_2}{w_2} = \frac{(1-t)Q_1 + tQ_2}{(1-t)w_1 + tw_2} \qquad (3)$$

і вирішимо відносно t.

Обчислення t.

Задамо a і b, таким чином, що $1-t^s = a/(a+b)$, $t^s = b/(a+b)$

Задамо A і B так, що $1-t = A/(A+B)$, $t = B/(A+B)$

Тоді:

$$Q^s = \frac{aQ_1/w_1 + bQ_2/w_2}{(a+b)} = \frac{AQ_1 + BQ_2}{Aw_1 + Bw_2} \qquad (4)$$

Легко перевірити, що $A = aw_1$ і $B = bw_2$ задовільнюють цьому рівнянню, дозволяючи нам одержати шуканий параметр t і, далі, координати Q.

Продовжимо. У нас є матриця М, що переводить координати із системи джерела в спостережницьку:

Рівняння (6) виражає координати на поверхні текстури, що відповідають будь-якій точці сегмента обираної (лінійно інтерполюємої) параметром t в екранних координатах.

$$Q^l = MQ = \frac{A}{A+B}Q_1^l + \frac{B}{A+B}Q_2^l \qquad (5)$$

где $Q_1^l = (x_1^l, y_1^l, z_1^l, w_1^l)$ и $Q_2^l = (x_2^l, y_2^l, z_2^l, w_2^l)$ координаты в системе источника света точек Q_1 и Q_2 (концы нашего отрезка) из наблюдательской системы. В итоге мы получаем:

$$Q^t = Q^l/w^l = \frac{AQ_1^l + BQ_2^l}{Aw_1^l + Bw_2^l} = \frac{aQ_1^l/w_1 + bQ_2^l/w_2}{a(w_1^l/w_1) + b(w_2^l/w_2)} \qquad (6)$$

Для того, щоб одержати координати, ми повинні лінійно інтерполювати x^l/w, y^l/w і w^l/w.

Для кожного піксела: $x^t = x^l/w$ і $y^t = y^l/w$, при цьому

$$x^l/w^l = \frac{x^l/w}{w^l/w} \text{ і } y^l/w^l = \frac{y^l/w}{w^l/w} \qquad (7)$$

Якщо w^l постійна на всьому полігоні, то рівняння (7) здобуває вид

$$s = \frac{s/w}{1/w} \text{ і } t = \frac{t/w}{1/w} \qquad (8)$$

відкіля ми маємо $s = x^l / w^l$ і $t = y^l / w^l$. Тут (s,t) - текстурні координати, синоніми (x^l, y^l)

Рівняння (8) і визначає текстурні координати, які можна прив'язати до вершин переданого на прискорювач полігона. У більш загальному складному випадку проективної текстури, що виражається рівнянням (7), потрібно розподіл на w^l / w, а не на $1/w$.

При накладанні текстур, у принципі, також можна побачити шви між двома найближчими bitmap. Чи, що буває частіше, у деяких іграх при зображенні чи дороги довгих коридорів помітне мерехтіння під час руху. Для усунення цих недоліків застосовується фільтрація (звичайно bi- чи tri-лінійна).

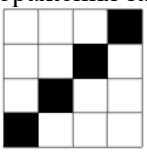
Білінійна фільтрація - метод усунення перекручувань зображення. При повільному чи обертанні русі об'єкта можуть бути помітні перескакування пікселов з одного місця на інше, що і викликає мерехтіння. Для зниження цього ефекту при білінейній фільтрації для відображення точки поверхні береться зважене середнє чотирьох суміжних текстурних пікселов.

Трилінійна фільтрація трохи складніше. Для одержання кожного піксела зображення береться зважене середнє значення результатів двох рівнів білінійної фільтрації. Отримане зображення буде ще більш чітке і менш мерехтливе.

Текстури, за допомогою яких формується поверхня об'єкта, змінюють свій вид у залежності від зміни відстані від об'єкта до положення очей глядача. При зображенні, що рухається, наприклад, у міру того, як об'єкт віддаляється від глядача, текстурний bitmap повинний зменшуватися в розмірах разом зі зменшенням розміру відображуваного об'єкта. Для того щоб виконати це перетворення, графічний процесор перетворить bitmap текстур аж до відповідного розміру для покриття поверхні об'єкта, але при цьому зображення повинне залишатися природним, тобто об'єкт не повинний деформуватися непередбаченим образом.

Для того щоб уникнути непередбачених змін, більшості випадках створюють серії предфільтрованих bitmap текстур зі зменшеним дозволом, це називається (mip mapping). При цьому, необхідно визначити, яку текстуру використовувати, ґрунтуючись на деталях зображення на екрані монітора. Відповідно, якщо об'єкт зменшується в розмірах, розмір його текстурного bitmap теж зменшується.

При промальовування прямих ліній, не рівнобіжних краям екрану,. з'являється "сходовий ефект". Для боротьби з цим недоліком зображення застосовується anti-aliasing.



Рвані краї

Рівні краї

Це спосіб обробки (інтерполяції) пікселов для одержання більш чітких країв (границь) зображення (об'єкта). Найбільше часто використовувана техніка - створення плавного переходу від кольору лінії чи краю до кольору фону. Колір точки, що лежить на границі об'єктів визначається як середнє кольорів двох граничних точок. Однак у деяких випадках, побічним ефектом anti-aliasing є змазування (blurring) країв.

Крім перерахованих вище основ, тривимірні графічні плати звичайно мають можливість відтворення деякої кількості додаткових функцій. Наприклад, для реалізації деяких ефектів застосовується fogging (затуманення). Він утвориться за рахунок комбінування змішаних комп'ютерних колірних пікселов з кольором тумана (fog) під керуванням функції, що визначає глибину затуманення. За допомогою цього ж алгоритму далеко віддалені об'єкти занурюються в димку, створюючи ілюзію відстані.

Реальний світ складається з прозорих, напівпрозорих і непрозорих об'єктів. Для обліку цієї обставини, застосовується alpha blending - спосіб передачі інформації про прозорість об'єктів. Ефект прозорості створюється шляхом об'єднання кольору вихідного піксела з пікселом, що вже знаходиться в буфері. У результаті колір крапки є комбінацією кольорів переднього і заднього плану. Звичайно, коефіцієнт alpha має нормалізоване значення від 0 до 1 для кожного кольорового піксела. Новий піксел = (alpha)(колір піксела A) + (1 - alpha)(колір піксела B).

Очевидно, що для створення реалістичної картини необхідно часте відновлення, що відбувається на екрані, його вмісту. Для додання плавності руху використовують метод – double buffering. Він полягає в тім, що наступне положення об'єкта вже намальовано, до того, як поточна відеосторінка сторінка буде змінена. Без застосування подвійний буферизації зображення не буде мати необхідної плавності, тобто буде переривчастим. Для подвійний буферизації потрібно наявність двох областей, зарезервованих у буфері кадрів. Обидві області повинні відповідати розміру зображення, виведеного на екран. Метод використовує два буфери для одержання зображення: один для відображення картинки, іншої для рендеринга. У той час як відображається вміст одного буфера, в іншому відбувається рендеринг. Коли черговий кадр оброблений, буфери переключаються (міняються місцями).

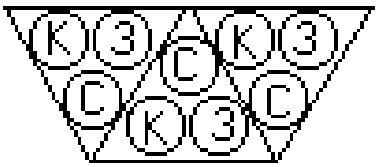
16. Принцип дії відеосистеми ПК

Розрізняють ***графічні*** і ***растрові дисплеї*** в залежності від використовуваного способу формування зображення: векторний чи растровий.

Дисплеї мають багаті можливості. Це керування кольором і яскравістю зображення, швидка зміна відображуваних кадрів, комбінування графічних елементів з алфавітно-цифровою інформацією. Усе це дозволяє будувати на екрані динамічні малюнки, що моделюють поводження відображуваного об'єкта, створювати зображення об'ємних тіл, змінювати їхнє положення в просторі. Зображення формуються за допомогою електронного променя, що направляється в задані точки люмінофора на екрані.

Найбільше поширення одержали растрові дисплеї, оскільки вони більш дешеві в порівнянні з графічними. Графічні дисплеї застосовуються в дорогих графічних системах.

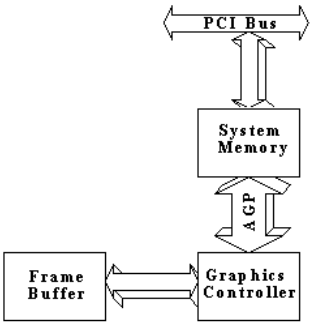
У растрових дисплеях електронний промінь на екрані переміщається в деякій прямокутній області. Ця область називається ***робочим полем***. Мінімально припустимі переміщення луча в робочому полі чи по кожній з координат визначають на дисплеї прямокутну сітку. Вузлами координатної сітки на робочому полі є точки растра.



У відповідні вершини піксела направляються промені від 3-х електронних пушек, кожна з який керує інтенсивністю свого променя. Змішання червоного, зеленого і синього кольорів у різній пропорції дозволяють одержувати досить багату палітру, що нараховує від кілька десятків до кілька мільйонів відтінків. Статичне зображення на робочому полі може бути описане за допомогою матриці станів, елементи якої відповідають вузлам координатної сітки. Значення кожного елемента визначає атрибути відповідного піксела (колір, яскравість).

ВІДЕОСИСТЕМА КОМП'ЮТЕРА

Схема підключення відеосистеми в сучасному комп'ютері зображена на малюнку. Для підвищення операцій обміну з пам'яттю використовується шина AGP, до якої підключається плата відеоадаптера комп'ютера.



На малюнку зображена структура типового сучасного відеоадаптера. Він складається з наступних основних частин:

1. Відеопам'ять (буфер кадру, Z-буфер, буфер текстур).
2. RAMDAC (RAM+DAC) – пристрій, що складається з RAM (пам'ять) DAC (ЦАП – перетворювач цифрового сигналу в аналоговий).
3. 2D- і 3D-акселератори (процесори: растеризацыъ, що генерує трикутники, рендеринга (промальовування), геометричних перетворень).
4. TV-тюнер (на платі чи окремо).

Основні пристрої – буфер кадру і RAMDAC.

Буфер кадру – пам'ять, де розміщаються відеодані, готові до виводу на екран. Розрізняють Frontbuffer – з нього дані безпосередньо виводяться на екран, і Backbuffer - у який виконує розрахунок наступного кадру. Це дозволяє поліпшити плавність виводу зображення на екран. Іноді використовується і Framebuffer – де зберігається кілька кадрів, готових до виводу на екран.

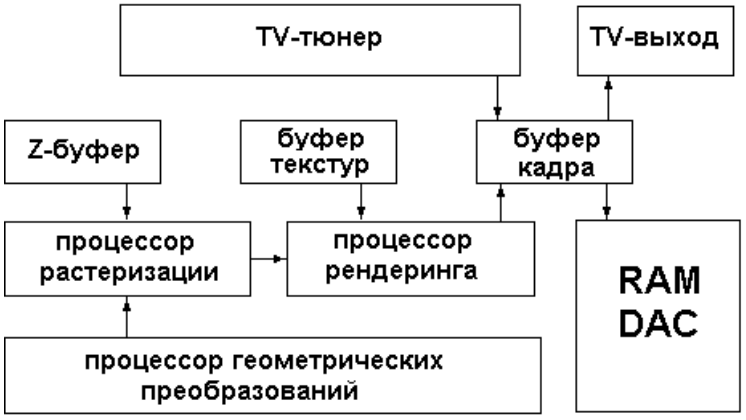


Рис. Структура видеоадаптера

Сформоване зображення попіксельно надходить у RAMDAC, де колірний код перетвориться в аналогові сигнали трьох основних кольорів (R, G, B

– червоного, зеленого і синього). RGB-сигнал подається в монітор для відображення.

Розглянемо роботу відеоадаптера на прикладі створення об'ємного зображення. Формування об'ємного зображення можна розділити на 2 етапи: побудова каркаса і рендеринг – процес зафарбування каркасної моделі об'єкта .

Каркас – геометрична модель поверхні об'єкта, що складає з трикутників. Вершини трикутників – основна інформація про об'єкт. Ця інформація обробляється процесором при переміщенні об'єкта. Кількість трикутників визначають ступінь деталізації поверхні. При збільшенні деталізації об'єкта (кількості трикутників) збільшується якість відображення і вимоги до продуктивності відеосистеми. Дані про координати трикутників надходять у 2D прискорювач, де перетворюються в растрове зображення (процес генерації трикутників). При цьому швидкість генерації трикутників складає кілька одиниць до кілька десятків млн/с.

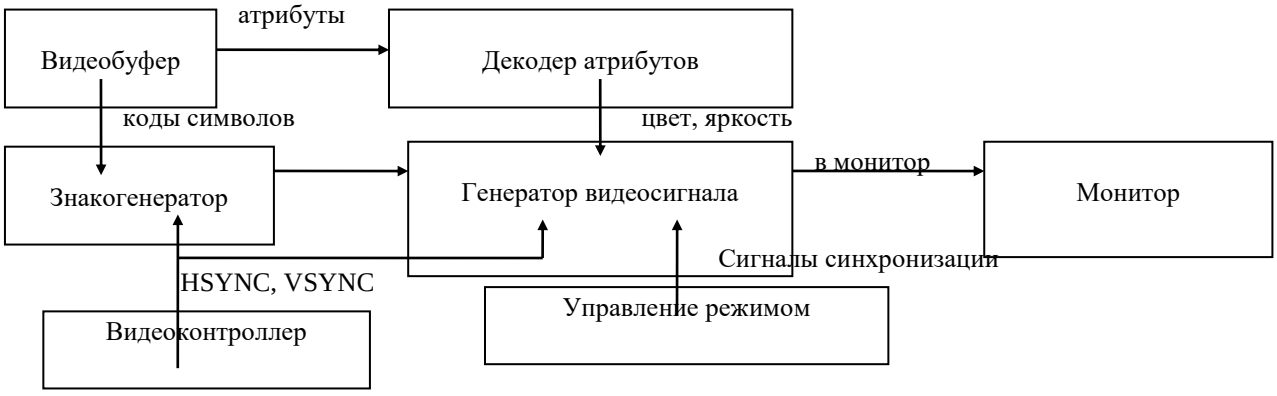
При обробці каркасної моделі враховуються тип і розташування джерела світла, текстура, вид проектування (перспективне). Процесор виконує необхідні обчислення і визначає колір і інтенсивність фарбування растрового представлення трикутника. Останнім часом випускаються відеоакселератори, оснащених процесорами геометричних перетворень і динамічного висвітлення (T&L). При рендерингу враховують:

- Глибину об'єкта (далекість по координаті Z);
- Матеріал об'єкта, форму, тип поверхні;
- Джерела висвітлення;
- Видимість трикутників.

Основна характеристика рендерингу - швидкість заповнення. Вона складає кілька сотень млн. pixel/c. Крім того, слід зазначити, що під буфер текстур відеосистеми приділяється більше пам'яті (приблизно в два рази), чим під буфер кадру і Z-буфер. Як параметри відеосистеми приводять значення fps- flip per second, тобто кількість анімованих кадрів у секунду (швидкість з який відеосистема може обробляти кадри анімації). Цей параметр не слід плутати з частотою кадру (f_k=85-120 Гц, а f_v=60 fps – частота відновлення кадру у відеобуфері).

Для прискорення обробки у відеосистемі використовуються різні види розпаралелювання обчислень, апаратна підтримка функцій API (Application Programming Interface – інтерфейс програмування додатків). На сьогоднішній день стандартами стали два основних API – Direct від Microsoft і OpenGL від Silicon Graphics. Виключенням стало створення компанією 3Dfx програмні продукти Glide OGL, що виконує тієї ж функції.

Структура RAM DAC



Інформація у відеобуфері зберігається в байтах. Кожен байт може визначати колір одного чи трохи пікселів.

Для відеобуфера, де зберігаються атрибути зображення, застосовуються спеціальні мікросхеми, що мають 2 вхідних канали:

1. Для регенерації зображення (адаптер тільки зчитує інформацію);
2. Для запису зображення.

Обидва канали діють паралельно і незалежно.

Функції відеоконтролера: (формування сигналів горизонтальної і вертикальної синхронізації, лічильник адресу відеобуфера, формує форму і позицію курсору).

17. Пристрої вводу/виводу графічної інформації.

Існує багато різноманітних пристроїв для вводу/виводу зображень. Як приклади можна привести пір'яні графобудівники, точечно-матричні, електростатичні і лазерні друкувальні пристрої, фильмирующие пристрою, дисплеї на запам'ятовуючій трубці, векторні дисплеї з регенерацією зображення і растрові дисплеї на ЭЛТ .

У сучасних ПК використовуються різні засоби відображення графічної інформації, у яких для фіксації зображень застосовуються різні фізичні принципи.

В основному розрізняють 2 типи графічних пристроїв:

1. Пристрої, що забезпечують одержання графічного документа (твердої копії).
2. Графічні дисплеї.

Пристрою для одержання графічного документа (твердої копії) використовують різні носії інформації: звичайна чи электро-хімічний папір, фоточуттєва плівка й ін. Сюди відносяться різні графопобудовачи чи координатографи. Вони можуть використовувати різні фізичні принципи нанесення зображення. Наприклад:

1. Шляхом безпосереднього контакту з носіями за допомогою звичайних кулькових або стрижнів ампул з чорнилом.
2. Без безпосереднього контакту з носіями за допомогою спеціальних самописців, що розпорошують барвну чи рідину пасту.
3. Випалюючи зображення за допомогою іскрового розряду шляхом використання спеціальних голчастих матриць.
4. Шляхом використання різців, що гравірують захисний шар на пластині.
5. Використання фотоголівок з керованою конфігурацією і розмірами лучачи й ін.

Розміри робітників полів (прямокутна область, у якій переміщається пишучий вузол) графобудівників можуть бути різні. Настільні графобудівники забезпечують висновок на аркуші формату А4 (210*297) чи А3 (297*520), а робочі поля координатографів мають розміри від 800 мм до 1200 мм. Спеціальні графопобудовачи, які використовують в судо- та авіабудуванні здійснюють вивод документації шириною до 2 м , та довжиною до 8 - 10 м.

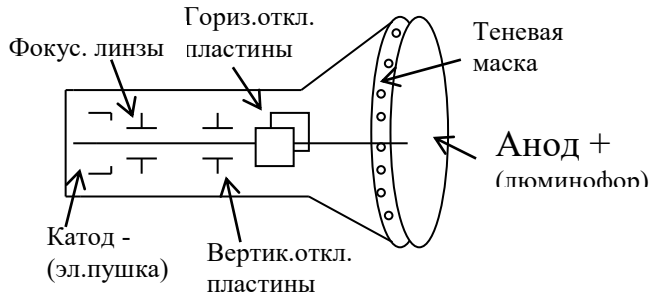
Графічні дисплеї мають багаті можливості по формуванню різних графічних зображень. Це керування кольором і яскравістю, швидка зміна відображуваних кадрів, комбінування графічної і текстової інформації. Зображення на екрані дисплея формується за допомогою електронного променя, що направляється у визначені точки люмінофора на екрані. Розміри робочого поля графічних дисплеїв складає 40 - 60 см.

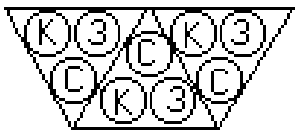
Приведемо просту класифікацію дисплеїв.

Устройство ЭЛТ

Катод (негативно заряджений) нагрівається доти, поки збуджені електрони не створять хмари, яка розширюється (електрони відштовхуються друг від друга, тому що мають однаковий заряд) . Ці електрони притягаються до позитивного аноду - внутрішня поверхня, на яку нанесений люмінофор. Електронний промінь фокусується і відхиляється.

У векторному дисплеї електронний промінь відхиляється в будь-яку довільну позицію. Оскільки люмінофорне покриття на цих ЕЛТ нанесено суцільним шаром, то виходить майже ідеальна пряма. Кольорове зображення у векторних дисплеях виходить за рахунок того, що існують червоний, зелений і синій шари люмінофора. Електронні промені засвічують визначений шар люмінофора. На відміну від цього, у растрових дисплеях промінь рухається по фіксованій траєкторії – растру, тобто промінь відхиляється у визначені точки. Люмінофорне покриття в растрової ЕЛТ являє собою безліч поруч розташованих точок люмінофора.





У кольорових дисплеях у відповідні вершини пиксела направляються промені від 3-х електронних пушок, кожна з який керує інтенсивністю свого променя. Змішання червоного, зеленого і синього кольорів у різній пропорції дозволяють одержувати досить багату палітру, що нараховує від кілька десятків до кілька мільйонів відтінків. Для ПК випускають спеціальні растрові дисплеї з підвищеною здатністю.

Кількість рядків на робочому полі перевищує звичайний телевізійний стандарт (625 рядків).

Електронні пушки поєднуються в блок, що відповідає подібному блоку точок люмінофора.

Тіньові маски:

- точечні;
- щілинні;
- апертурні (змішані)

Колірні пушки розташовані таким чином, що їхні промені сходяться і перетинаються в площині тіньової маски. Після проходження через отвір кожен електронний промінь захищений від перетинання не зі своєю точкою люмінофора (маскований). Таким чином, електронний промінь може перетнути тільки свою точку.

Основні технічні характеристики відеосистем

До основних технічних характеристик варто віднести:

- довільна здатність;
- ємність відеопам'яті;
- частота рядків і частота кадрів;
- смуга пропущення відеоканалу

Довільна здатність визначається ємністю відеопам'яті (число крапок по горизонталі X, число крапок по вертикалі Y, число біт квітів на одну точку), смугою пропущення, частотою рядків і кадрів. При відтворенні графіки довільну здатність бажано мати якнайвище, оскільки користувачі ПК знаходяться на незначній відстані від екрана і краще бачать усі недосконалості зображення.

Для якісного відтворення графічних зображень треба використовувати як можна більшу кількість точок зображення, високу кількість та рівнів градації кольорів). Довільна здатність, кількість кольорів визначає ємність відеопам'яті. Причому зі збільшенням числа точок обсяг відеопам'яті збільшується по квадратичному законі, а зі збільшенням кількості кольорів - по логарифмічному. Як правило, технічні можливості відеопам'яті вище, ніж у монітора.

Інформація з відеопам'яті зчитується послідовно, за допомогою використання синхросигналів: горизонтального і вертикального розгорнення. При цьому важливе значення має

стабільність формованого зображення, тобто відсутності мерехтіння екрана. Основна причина мерехтіння – це гашення променя при зворотньому його ході. Мерехтіння найбільшою мірою виявляється у світлих частинах екрана, причому зі зменшенням відстані око до екрана цей ефект зростає. Для зменшення мерехтіння потрібно підвищувати частоту кадрів. Вона повинна бути більш 60 гц. Збільшення частоти кадрів при збереженні здатності, вимагає збільшення смуги пропущення відеоканалу. Наприклад, при частоті кадрів 60 Гц і здатності, 1280x1024 точок смуга пропущення відеоканалу складає 110 Мгц. В даний час смуга пропущення складає більш 200 Мгц.

Для подолання труднощів зв'язаних з розширенням смуги пропущення, раніше використовувався метод черезстрочной розгорнення (Interlace), при якому спочатку формувалися непарні, а потім усі парні рядки. В даний час використовується метод безупинного розгорнення (Non-Interlace), при якому побудова кадру відбувається послідовно без розподілу на частині.

У залежності від області застосування ПК до здатності, висувають різні вимоги. Наприклад, при моделюванні в автомобілебудуванні здатність, повинна бути не нижча за 1024x768 точок з відображенням 256 відтінків, в офісних додатків - не нижче 640x480, а точність передачі кольору не має великого значення, а для рекламної графіки має велика кількість квітів і стійкість зображення.

Чи частота швидкість висновку пікселов на екран називається шириною смуги пропущення відеоканалу. Здатність $R = Nx \cdot Ny$, смуга $f_П$ пропущення , частота f_S рядків і f_K кадрів зв'язані між собою наступними співвідношеннями:

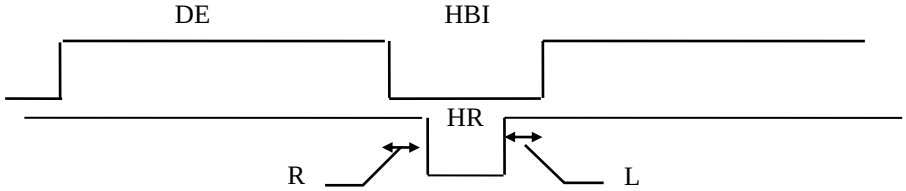
1. Число крапок в одному рядку: $Nx = \frac{f_П}{f_S}$.
2. Число рядків розгорнення в одному кадрі $Ny = \frac{f_S}{f_K}$.
3. $R = Nx \cdot Ny = \frac{f_П}{f_S}\frac{f_S}{f_K} = \frac{f_П}{f_K}$.

При виборі монітору треба звертати увагу не тільки на технічні, а і на геометричні характеристики. Це геометрія, зведення та фокусування електронних променів.

18. РОБОТА ВІДЕОСИСТЕМИ

Горизонтальна синхронізація (VSYNC)

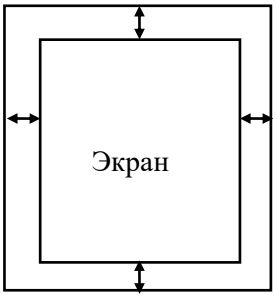
Відеоконтролер формує сигнал дозволу відображення DE (Display Enable). По цьому сигналі електронний промінь включається на початку кожного рядка. При проходженні лучачи по рядку зчитуються послідовності байтів з відеобуфера по адресах, формованим лічильником адреси відеоконтролера. Дані декодуються і посилаються в монітор у виді сигналів кольору і яскравості. У правого краю екрана відеоконтролер виключає сигнал DE і після цього дані з буфера не відображаються. Потім відеоконтролер формує горизонтальний синхросигнал, що відхиляє промінь вліво і вниз до початку наступної рядка растра. Після цього для відображення наступної рядка даних включається сигнал DE.



Часовий інтервал між кінцем одного рядка розгорнення відеоданих і початком наступної рядка називається інтервалом горизонтального гасіння HBI (horizontal blanking interval). Тому що час зворотного ходу HR (horizontal retrace) лучачи менше тривалості інтервалу горизонтального гасіння, то на кінець і початок кожного рядка приходиться горизонтальне перекриття ходу лучачи (праве R і ліве L облямівку).

Вертикальна синхронізація (VSYNC)

Після того як промінь пройде всі горизонтальні рядки сигнал DE відключається. Відеоконтролер формує вертикальний синхросигнал, по якому промінь повертається з нижньої правої частини екрана у верхню ліву частину. Через те, час зворотного ходу лучачи менше тривалості інтервалу вертикального гасіння лучачи, у верхній і нижній частині екрана виникають полючи вертикального перекриття (верхня і нижня облямівка).

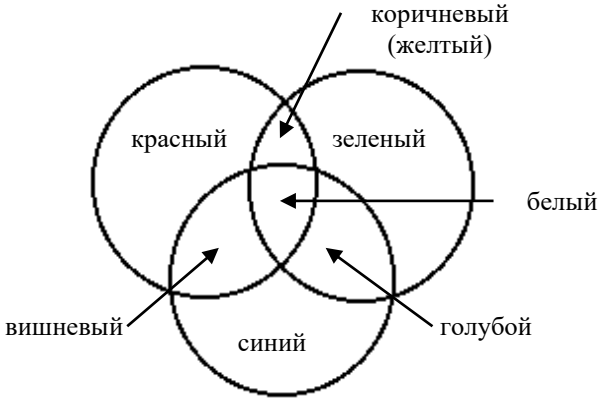


Облямівка дає можливість центрувати растр без утрат даних на екрані і зменшити нелінійні перекручування.

Кодування кольорів

Біти, що знаходяться у відеобуфері називаються відеоатрибутами і вони визначають, як ці піксели виглядають на екрані дисплея.

Відомо, що будь-який колір є комбінацією трьох основних кольрів – червоного (Red), зеленого (Green) і синього (Blue). У залежності від того, який «вага» має кожний з цих кольорів, виходить різноманітна колірна гама. У найпростішому випадку для кодування кожного з основних квітів досить по одному біті (0 – колір виключений, 1 – колір включений), називаних бітами R, G, B. З трьох основних квітів із двоїчним кодуванням, тобто з 3-х біт виходить 8 колірних комбінацій (мал.). Коли всі кольори виключені, виходить чорний колір, а коли усі включені – білий, а при інших комбінаціях – ще 6 кольорів. Якщо ввести ще один біт - чи інтенсивність яскравість (I), то можна одержати 16 кольорів.



Таким чином, для 4-х бітної комбінації, називаної IRGB – кольором, будуть отримані кольори, що представлені в таблиці.

Номер	I	R	G	B	Назва	Компоненти
0	0	0	0	0	Чорний	Немає
1	0	0	0	1	Синій	Синій
2	0	0	1	0	Зелений	Зелений
3	0	0	1	1	Голубий	Зелений + синій
4	0	1	0	0	Червоний	Червоний
5	0	1	0	1	Вишневий	Червоний + синій

6	0	1	1	0	Коричневий	Червоний + зелений
7	0	1	1	1	Білий	Червоний + зелений + синій
8	1	0	0	0	Сірий	I
9	1	0	0	1	Яскраво-синій	I + синій
10	1	0	1	0	Яскраво-зелений	I + зелений
11	1	0	1	1	Яскраво-голубий	I + зелений + синій
12	1	1	0	0	Яскраво-червоний	I + червоний
13	1	1	0	1	Яскраво-вишневий	I + червоний + синій
14	1	1	1	0	Жовтий	I + червоний + зелений
15	1	1	1	1	Яскраво-білий	I + червоний + зелений + синій

Відзначимо, що сприйняття квітів залежить від індивідуальних особливостей людини, а також апаратного настроювання монітора. Тому вишневий (magenta) колір називається пурпурним, малиновим і бузкової. Коричневий (brown) колір називається - золотавим, голубой (cyan) - бірюзовим чи ціановим, яскраво-червоний (bright red) - рожевим і т.д. Чим більше елементів могтиме 1, тим яскравіше буде світіння, але одночасно тим більше «розмитим» буде сприйматися колір. Для ока людини чисті основні кольори R, G, B виявляються візуально більш сприймані, чим будь-які зі змішаних кольорів і навіть чим «яскравий» варіант основного кольору.

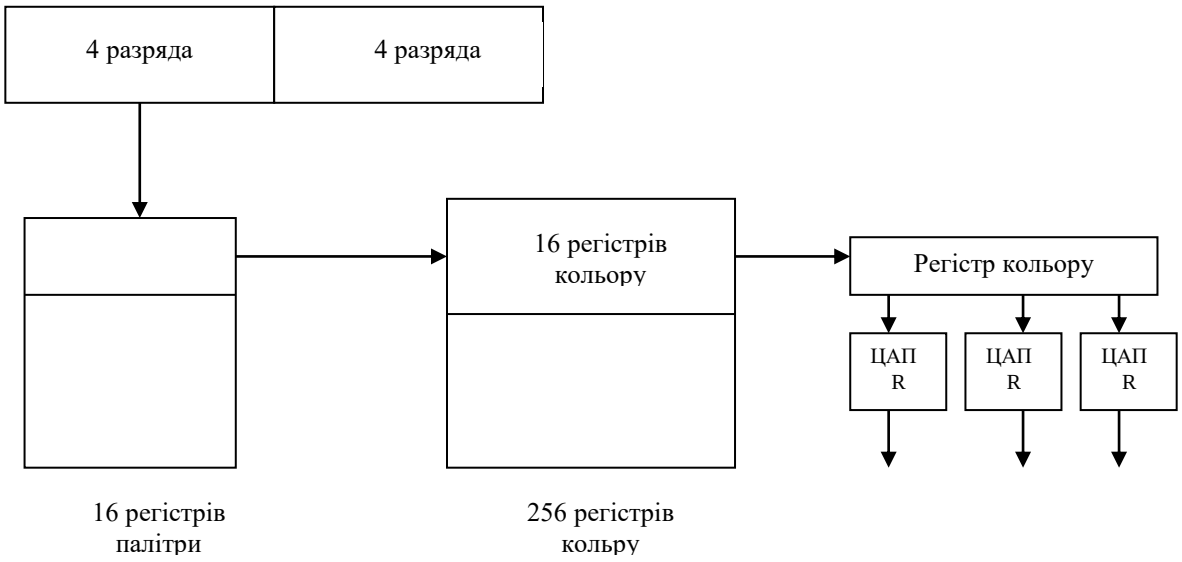
У 16-кольоровому режимі можна користатися всіма квітами з номерами 0 – 15, приведеними в таблиці. У 8-кольоровому режимі немає керування інтенсивністю, тому застосовують кольору 0 – 7. У 4-кольоровому режимі можна вибрати будь-які 4 кольору з 16, причому обрані кольори утворять так називану палітру (palette). Нарешті, у 2-кольоровому режимі можна вибрати тільки два кольори 0 і 7 – чорний і звичайний білий колір.

В адаптері EGA маються режими, у яких для кодування кожного з основних квітів відведено по двох біта, тобто повний колір кодується шістьма бітами RrGgBb (00 – колір виключений, 01 – слабкий колір, 10 – звичайний, 11 - яскравий). Таке кодування розширює загальне число квітів до 64. Проте одночасно на екрані може спостерігатися тільки 16 квітів, тому що у відеобуфері пікселі кодуються 4-бітними значеннями .

В адаптері VGA уведені режими, у яких для кодування кожного з основних квітів відведено по 6 біт, тобто можливе число квітів зросло до величини 256ДО. Однак одночасно на екрані спостерігається тільки 256 квітів, тому що в режимі максимальною «кольоровістю» кожен піксел кодується у відеобуфері 8 бітами.

Схема декодування атрибутів пікселів в адаптері VGA
(16-кольровий режим)

Звичайний режим VGA - це 16-кольоровий режим 640*480. У цьому режимі кожен байт визначає колір 2-х пікселів.

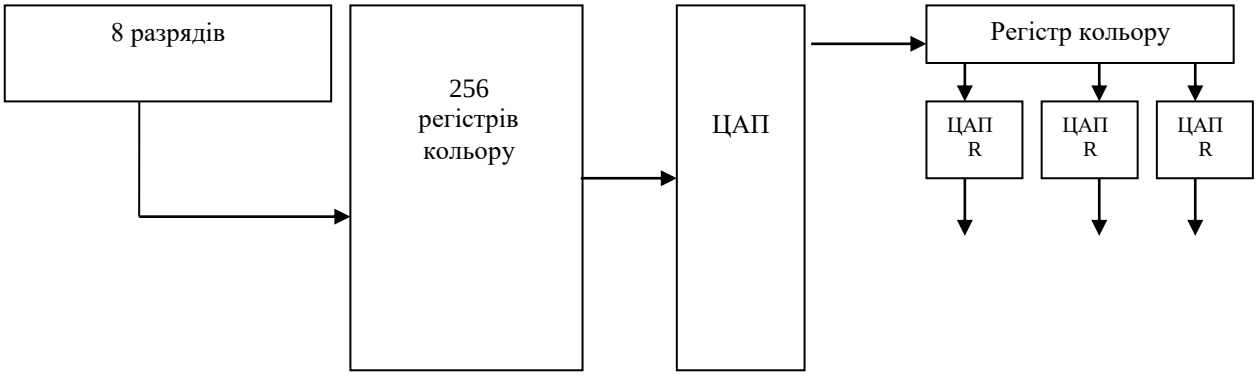


Адаптер VGA при декодуванні пікселів використовує 16 регістрів палітри. Кожне значення, що міститься в регістрі палітри вибирає один з 16 регістрів кольору (регістри 18-розрядні). Усього регістрів 256. Вміст обраного регістра визначає відображуваний колір. ЦАП перетворить 18-бітове значення цифрового RGB-сигналу (на кожний приділяється 6 розрядів) у відповідні аналогові RGB-сигнали, що подаються на монітор.

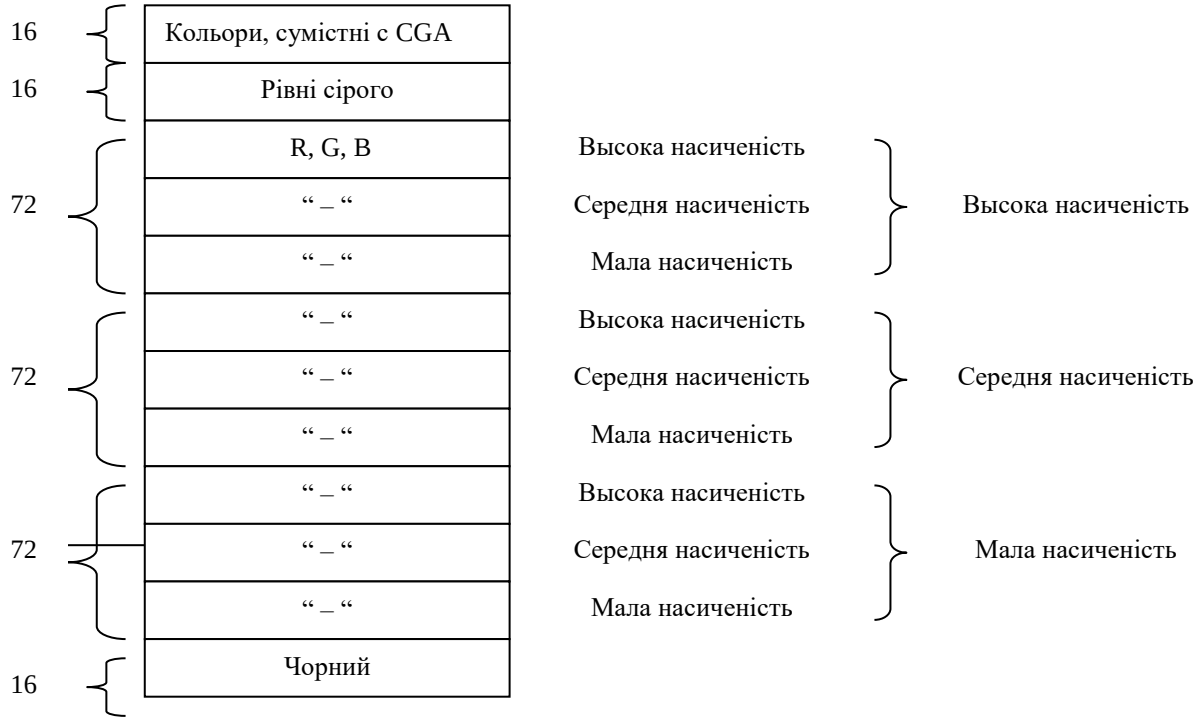
Схема декодування атрибутів пикселів в адаптері VGA

(256-кольоровий чи режим SVGA)

Режим SVGA - це 256-кольоровий режим 320*200. У цьому режимі кожен байт визначає колір 1-го піксела.



У цьому режимі кожне значення піксела безпосередньо вибирає один з 256 колірних регістрів. У кожен регістр кольору можна завантажити кожне з 256К значень квітів (2^{18}=256К). Перші 16 регістрів містять значення квітів сумісні з квітами адаптера CGA. Другі 16 регістрів містять різні рівні сірого з поступово збільшується яскравістю. Наступні 256 регістрів містять по 3 групи з 72 квітів з яскравістю, що збільшується. Останні 16 регістрів містять значення чорного кольору.



Це розподіл кольорів задається за замовчуванням. Однак уміст цих регістрів можна змінити, використовуючи засобу для формування довільної палітри.

ЛІТЕРАТУРА

1. Маценко В.Г. Комп’ютерна графіка: Навчальний посібник. – Чернівці: Рута, 2014 – 343 с. http://arr.chnu.edu.ua/handle/123456789/141
2. Блинова Т.А., Порев В.Н. Компьютерная графика/Под ред. В. Н. Порева— К.: Издательство Юниор, СПб.:КОРОНА принт. К.: Век+, 2012. — 520 с, ил.
3. Дональд Херн, М. Паулин Бейкер. Компьютерная графика и стандарт OpenGL = Computer Graphics with OpenGL. — 3-е изд. — М.: «Вильямс», 2015. — С. 1168.