

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Кафедра ПЗ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни «Методи та алгоритми комп'ютерної графіки»,
для студентів за спеціальностями

121 «Інженерія програмного забезпечення»

122 «Комп'ютерні науки»

усіх форм навчання

2017

Конспект лекцій з дисципліни «Методи та алгоритми комп'ютерної графіки», для студентів за спеціальностями 121 «Інженерія програмного забезпечення» 122 «Комп'ютерні науки» усіх форм навчання /Укладач: А.В. Пархоменко. – Запоріжжя: ЗНТУ, 2017. – 45 с.

Укладач: А.В. Пархоменко, к.т.н., доцент кафедри ПЗ,

Рецензент: Г.М. Шило, к.т.н., доцент кафедри ІТЕЗ.

Відповідальний

за випуск: С.О. Субботін, зав. каф. ПЗ, д.т.н., професор.

Затверджено
на засіданні кафедри
“Програмних засобів”

Протокол № 11
від 06.06.2017

Вступ.....	4
1 Основні концепції графічного програмування	5
1.1 Графічні бібліотеки	5
1.2 Системи координат	6
1.2.1 Системи координат для плоских зображень.....	6
1.2.2 Системи координат для роботи в 3D просторі	7
1.2.3 Перетворення між системами координат.....	9
1.3 Вікно та видовий екран	10
1.4 Графічні примітиви	11
1.5 Дисплейний файл	14
1.6 Текст та маркери на малюнках	14
1.7 Перетворення координат	15
1.7.1 Трансляція	16
1.7.2 Обертання	16
1.7.3 Відображення (mapping)	17
1.7.4 Інші матриці перетворення	18
2 Базові алгоритми комп'ютерної графіки	19
2.1 Малювання прямої лінії.....	19
2.2 Відсікання ліній	21
2.3 Алгоритми зафарбовування	24
2.3.1 Алгоритми заповнення для піксельно визначених областей.....	24
2.3.2 Алгоритми заповнення для символічно визначених областей.....	27
2.4 Стиль лінії. Перо.....	29
2.5 Стиль заповнення. Кисть. Текстура	30
3 Видалення невидимих ліній та поверхонь.....	33
3.1 Алгоритм видалення невидимих граней	33
3.2 Алгоритм художника (сортування по глибині)	34
3.3 Алгоритм видалення невидимих ліній.....	34
3.4 Метод Z – буфера (буфер глибини)	36
4 Рендерінг (візуалізація)	38
4.1 Затуюсування.....	38
4.2 Заломлення світла прозорими об'єктами	42
4.3 Трасування променів	42
РЕКОМЕНДОВАНА Література	45

ВСТУП

Мета - вивчення та практична реалізація програмних інструментів для створення і обробки комп'ютерних зображень.

У результаті вивчення навчальної дисципліни студент повинен знати:

- особливості та галузі використання різних видів комп'ютерної графіки;
- методи, моделі, алгоритми візуалізації та обробки комп'ютерних зображень;
- теорію кольору та особливості побудови реалістичних зображень;
- математичні аспекти 2-D та 3-D графіки;
- основи введення/виводу графічної інформації при автоматизованому створенні зображень.

вміти:

- виконувати обробку та створення комп'ютерних зображень за допомогою програм растрової та векторної графіки;
- виконувати розробку графічних програм з використанням графічної бібліотеки Open GL;
- виконувати розробку анімаційних фільмів, презентаційних слайдів та графіки на web-сторінках.

1 ОСНОВНІ КОНЦЕПЦІЇ ГРАФІЧНОГО ПРОГРАМУВАННЯ

1.1 Графічні бібліотеки

Графічне програмування вимагає крім основного програмного забезпечення (операційна система, редактор, компілятор) наявності спеціальних графічних програм двох класів (драйвери пристроїв і графічні бібліотеки).

Драйвер пристрою – набір апаратно-залежних кодів, які безпосередньо керують процесором графічного пристрою.

Графічна бібліотека – набір підпрограм, призначених для рішення визначених задач (наприклад, креслення окружності, прямої і т.д.), які ґрунтуються на командах драйвера пристрою.

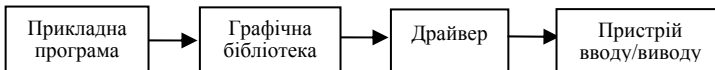


Рисунок 1.1 – Використання графічної бібліотеки

Проблема: назви й способи виклику підпрограм графічної бібліотеки різні, крім цього кожна бібліотека має своє коло даних, які вона підтримує, це означає обмежений набір графічних пристроїв.

Для рішення цієї проблеми близько 30 років розробляються різні бібліотеки (CORE, GKS-3D, PHIGS, PEX).

Open GL (Open Graphic Library) – комерційна графічна бібліотека, популярна завдяки гнучкості в керуванні робочими станціями й ПК у мережевому середовищі; здобуває статус графічного стандарту де-факто. (1992, Silicon Graphics, Ins).

Переваги бібліотеки Open GL:

- доступність;
- простота у вивченні та встановленні;
- довговічність у якості стандартного API, що підтримується експертною комісією з архітектури Open GL;
- апаратна незалежність /мобільність;
- великий і зручний набір API функцій для 2D і 3D графіки й обробки зображень.

1.2 Системи координат

Для виводу зображення об'єкта на екран графічного пристрою необхідно вирішити дві основні задачі:

- указати положення всіх точок об'єкта в просторі;
- визначити положення їхніх образів на моніторі (спроектувати об'єкт на екран).

Для цього використовуються різні системи координат, які пов'язані між собою.

1.2.1 Системи координат для плоских зображень

Система координат пристрою (device coordinate system (dcs)) визначає положення точки на екрані двома цілими числами (u , v), що дорівнюють числу пікселів (складається з горизонтальної осі u та вертикальної осі v , а початок відліку-довільний).

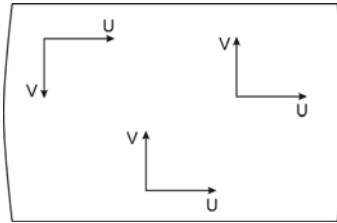


Рисунок 1.2 – Системи координат пристрою

Та сама точка може задаватися різними параметрами залежно від положення початку координат, напрямку осей і масштабу (ці параметри для різних графічних пристроїв встановлюються довільно). Тому, апаратні координати в графічній програмі можуть вимагати зміни при зміні графічного пристрою.

Віртуальна система координат пристрою (virtual dcs) – фіксує точку відліку (зазвичай лівий нижній кут), напрямок і масштаб осей (параметри u та v змінюються від 0 до 1). Графічна програма передає віртуальні координати підпрограмі драйвера пристрою, яка перетворює її в координати конкретного пристрою.

1.2.2 Системи координат для роботи в 3D просторі

Зовнішня (світова) система координат (world coordinate system – wcs) – це опорна система, що використовується для опису об'єктів світу, що цікавлять нас.

Система координат моделі (model coordinate system) – дозволяє описати форму кожного об'єкта світу шляхом завдання координат характеристичних точок об'єкта стосовно самого себе; вона пов'язана з об'єктом і переміщується разом з ним.

Розташування та орієнтація будь-якого об'єкта задаються відносним положенням і орієнтацією модельної системи координат даного об'єкта відносно зовнішньої системи координат, які у свою чергу визначаються матрицею перетворення.

Система координат спостережника (viewing cs) – полегшує розрахунок проєкції спостереження й використовується для проєцювання 3D об'єктів або їхніх точок на екран монітора.

Система координат спостережника X_v Y_v Z_v будується так:

- початок розташовується у точці, що розглядається;
- вісь Z_v спрямована з початку координат у точку зору;
- вісь Y_v паралельна вертикальній осі екрана;
- вісь X_v визначається як векторний добуток перших двох.

У комп'ютерній графіці використовується два види проєкцій (перспективна і паралельна), які вимагають завдання двох точок: точки зору й точки спостереження.

Точка зору (viewpoint) – око спостерігача.

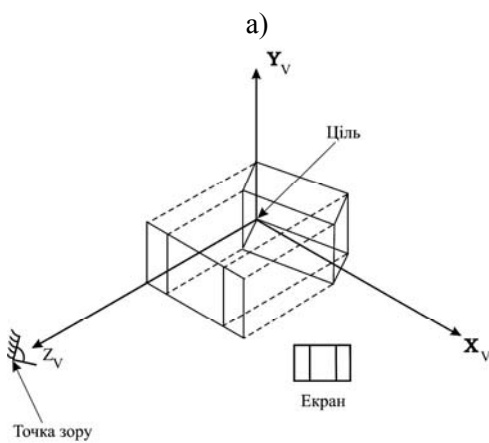
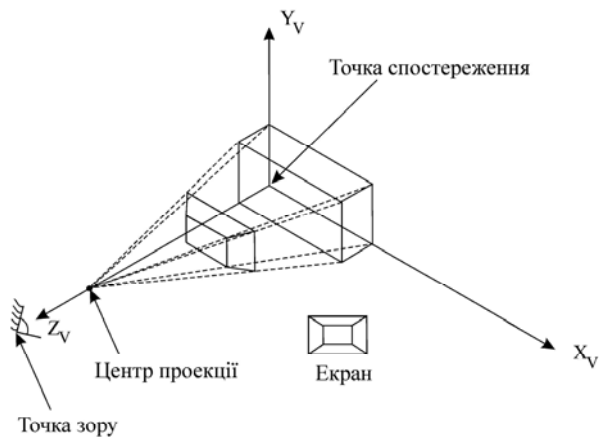
Точка спостереження (viewsite) – точка об'єкта, що визначає напрямок променя зору.

Напрямок спостереження – вектор, проведений від точки зору до цілі.

У **перспективній проєкції** всі точки розглянутого об'єкта з'єднуються із центром проєкції, який зазвичай лежить на лінії, що з'єднує точку зору й ціль (у протилежному випадку проєкція називається косокутною); точки перетинання цих ліній з екраном і утворюють проєкцію. Екран розташовується між точкою зору й цілі.

У **паралельній проєкції** лінії від всіх точок об'єкта проводяться в напрямку спостерігача паралельно напрямку спостереження, а точки перетинання цих ліній з екраном формують проєкцію (екран

перпендикулярний напрямку проєцювання тобто проєкція ортогональна).



б)

Рисунок 1.3 – Перспективна (а) і паралельна (б) проєкції.

У більшості графічних бібліотек користувачеві доводиться задавати вектор вертикалі (up-vector), положення точки зору й точки спостереження у світових координатах. Після визначення системи координат спостережника і обчислення координат точок об'єктів, що цікавлять нас, обчислюють положення їхніх проекцій на екрані.

Для паралельної проекції координати точок дорівнюють відповідним координатам X_v та Y_v точок об'єкта, а для перспективної – вимагають перерахування по формулах:

$$X_s = \frac{S}{L - Z_v} \times X_v$$

$$Y_s = \frac{S}{L - Z_v} \times Y_v,$$

де L – відстань між точкою спостереження й центром проекції;
 S – відстань між центром проекції й екраном.

1.2.3 Перетворення між системами координат

Розглянуті системи координат зв'язані матрицями перетворення. Співвідношення систем координат можна проілюструвати в такий спосіб (див. рис.1.4):

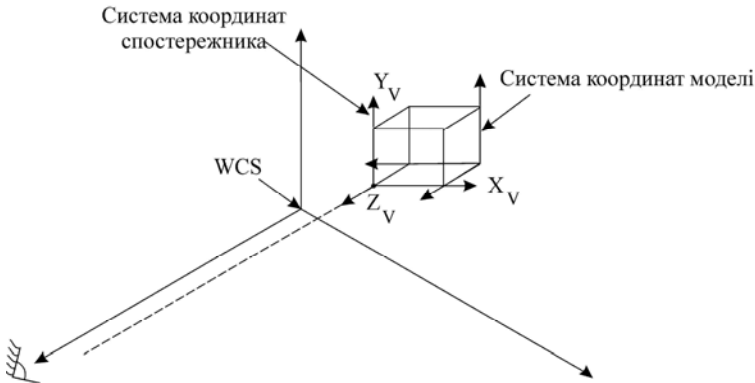


Рисунок 1.4 – Співвідношення систем координат

Всі ці перетворення виконуються усередині графічної бібліотеки, а програміст тільки вказує відомості, що наведені на рис. 1.5 у дужках.



Рисунок 1.5 – Процедура розрахунку точок проєкції з використанням матриць перетворення

1.3 Вікно та видовий екран

Світове вікно (world window) – встановлює яку частину "зовнішнього світу" слід намалювати, тобто відсікає невидиму частину простору.

Екранне вікно (графічне вікно) (screen window) – визначається як прямокутник, що лежить на екрані і заданий значеннями ширини і висоти в пікселях.

Об'єм, що переглядається – це видима частина простору, що залежить від типу проєкції (для обмеження об'єму, що проглядається вводять поняття найближчої та найдальшої площини).

Видовий екран (viewport, демонстраційне вікно, порт перегляду, вікно виводу) – це область екрана, в якій відображається зображення, що проєцюється.

Відображення складається з трансляції та масштабування, що враховують відстань між центром видового екрана і центром монітора, а також різницю розмірів світового вікна і видового екрана.

Приклад коду, що задає світове вікно і видовий екран з використанням графічної бібліотеки Open GL:

```
static GLint viewport[] = [0, 0, 400, 400];
/* видовий екран задається координатами вікна */
static GLclampd depth_range[] = {0.0, 1.0};
/* поправки до мінімального і максимального значень, які
можуть зберігатися у буфері глибини */
static GLdouble viewing_volume[] = {-100.0, 100.0, -100.0, 100.0, -
10.0, 10.0};
/* аргументи в координатах перегляду,
(1 – 4) – горизонтальний і вертикальний розміри вікна,
(5 – 6) – відстань від екрана до найближчої і найдальшої
площини */
glOrtho (viewing_volume[0], viewing_volume[1], ...,
viewing_volume[5]);
/* визначає тип проекції, створює матрицю об'єму, що
переглядається*/
glViewport (viewport[0], ..., viewport[3]);
/* визначає прямокутник вікна і відображає в нього зображення
*/
glDepthRange (depth_range[0], depth_range[1]);
/* визначає кодування z-координат при перетворенні
перегляду */
```

1.4 Графічні примітиви

Графічні примітиви (primitives) – це елементи графіки (точки, лінії, ламані, полігони), які можуть відображатись бібліотекою і задаються однією або декількома вершинами (vertices). Вивід примітивів в OpenGL оформлюється парою функцій:

```
glBegin (mode);
... // вершини об'єкта
glEnd ();
де
mode – код типу об'єкта.
```

Таблиця 1.1 – Трактовка списку вершин для різних аргументів функції `glBegin`

mode	Як виконується графічний вивід
GL_POINTS	Вершини-окремі точки 
GL_LINES	Вершини, з'єднані відрізками прямої 
GL_LINE_STRIP	Полілінія 
GL_LINE_LOOP	Замкнена полілінія 
GL_TRIANGLES	Трійки вершин, що створюють трикутники 
GL_TRIANGLE_STRIP	Зв'язані трикутники (полоса) 
GL_TRIANGLE_FAN	Зв'язані трикутники зі спільною вершиною (веер) 
GL_QUADS	Чотири вершини, що створюють чотирикутники 
GL_QUAD_STRIP	Зв'язані чотирикутники 
GL_POLYGON	Багатокутник 

Наприклад, програмний код для малювання сузір'я Велика Ведмедиця:

```
glBegin (GL_POINTS);
glVertex2i (289, 190);
/* Dubhe */
glVertex2i (320, 128);
/* Merak */
glVertex2i (230, 67);
/* Phecda */
glVertex2i (194, 101);
/* Megrez */
glVertex2i (129, 83);
```

```

/* Alioth */
glVertex2i (75, 73);
/* Mizar */
glVertex2i (74, 74);
/* Alcor */
glVertex2i (209, 10);
/* Alcaid */
glEnd ();

```

glVertexXX (...) – функція завдання координат, яка має декілька варіантів суфікса в залежності від типу і кількості аргументів, наприклад:

```

i – 32-х бітне ціле;
f – 32-х бітне дійсне;
d – 64-х бітне дійсне.

```

Якщо ми інкапсулюємо команди Open GL для малювання точки в базову функцію drawDot(), то така підпрограма буде виглядати наступним чином:

```

void drawDot (GLint x, GLint y);
{
    glBegin (GL_POINTS);
    glVertex2i (x, y);
    glEnd ();
}

```

OpenGL відстежує множину **змінних стану**: поточний розмір точки, поточний колір малювання, колір фону.

Розмір точки встановлюється за допомогою функції glPointSize(), колір для малювання: glColor3f (red, green, blue).

Наприклад, деякі з кольорів можуть бути задані так:

```

glColor3f (1.0, 0.0, 0.0) // червоний
glColor3f (1.0, 1.0, 1.0) // білий

```

Колір фону встановлюється за допомогою функції

glClearColor (red, green, blue, alpha), де змінна alpha визначає ступінь прозорості.

1.5 Дисплейний файл

Дисплейний файл (display list) – це група команд графічної бібліотеки, збережена для подальшого виконання.

Дисплейний файл забезпечує зручність та ефективність упорядкування та обробки команд бібліотеки.

Створення дисплейного файлу:

```
glNewList (AREA_FILL, GL_COMPILE_END_EXECUTE);
    glBegin (GL_POLYGON);
        glVertex2i (point1);
        glVertex2i (point2);
        ...
    glEnd ();
glEndList ();
```

Дисплейний файл Open GL орієнтований на оптимізацію продуктивності для роботи в мережі, яка забезпечується завдяки тому, що дисплейний файл працює дуже швидко та зберігається у виді списку команд, а не у виді динамічної бази даних.

При передачі підпрограми Open GL по мережі, дисплейні файли зберігаються на сервері, завдяки чому скорочується мережевий трафік.

До створеного дисплейного файлу можна застосувати наступні операції:

- множинне виконання;
- ієрархічне виконання;
- видалення.

1.6 Текст та маркери на малюнках

Більшість графічних бібліотек підтримує два види тексту: екранний або 2D, текст для пояснень, 3D - об'ємний текст.

Атрибути теста:

- шрифт;
- відношення висоти до товщини;
- кут нахилення літер;
- положення ы напрямком рядка тексту.

Текст може буди представлений символами двох видів: апаратними (складаються з окремих відрізків), та

програмними(заздалегіть розробляються графічними програмами та зберігаються в пам'яті комп'ютера.)

Маркери різних типів (точка, коло, хрестик, сніжинка) використовуються для виділення точок на графіках. Open GL не підтримує маркери явно, але надає механізм їх збереження маркерів у растрових файлах і виведення їх на екран.

1.7 Перетворення координат

Афінні перетворення – лінійні перетворення координат, які надають гнучкий спосіб керування розмірами, орієнтацією та розміщенням об'єктів, що нас цікавлять.

Область застосування:

- складання картинки з однакових об'єктів простої форми;
- використання «мативу» для створення малюнка;
- перегляд сцени з різних точок спостереження;
- анімація шляхом перетворення форм(робота з окремими локальними системами координат об'єктів);
- перетворення координат з однієї системи в іншу.

Перетворення координат з однієї системи в іншу вимагають застосування операцій трансляції, повороту і відображення.

Ці перетворення виконуються графічною бібліотекою самостійно, а програміст повинен тільки вказати:

- при перетворенні модельних координат в світові – трансляцію (переміщення) і поворот;
- при перетворенні з світових координат в координати спостережника – точки зору, точки спостереження та напрямок вектора вертикалі.

1.7.1 Трансляція

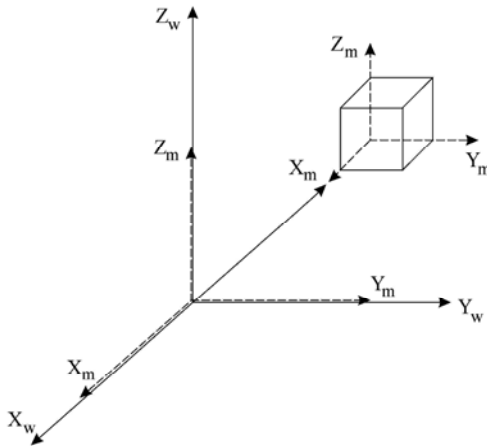


Рисунок 1.6 – Трансляція (переміщення) об'єкта

При переміщенні об'єкта його світові координати в новому положенні визначаються так:

$$X_w = X_m + a$$

$$Y_w = Y_m + b$$

$$Z_w = Z_m + c$$

w_0 в матричній формі:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix}$$

⇓

Trans (a, b, c), де

X_m, Y_m, Z_m – модельні координати точки об'єкта;

a, b, c – величини переміщення у напрямках x, y, z;

Trans (a, b, c) – однорідна матриця трансляції.

Операція трансляції в OpenGL реалізується наступним чином
 glTranslated (GLdouble offset_x, GLdouble offset_y,
 GLdouble offset_z);

1.7.2 Обертання

При обертанні об'єкту навколо осі X світової системи координат на кут θ координати можуть бути визначені так:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix}$$

↓

$Rot(X, \theta)$

$Rot(x, \theta)$ – однорідна матриця обертання навколо осі X.

При обертанні навколо осі Y:

$$Rot(Y, \theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

При обертанні навколо осі Z:

$$Rot(Z, \theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Матриця перетворення для довільної осі визначається перемноженням наведених вище трьох матриць.

У бібліотеці Open GL **обертання** реалізується так:

`glRotated (GLdouble angle, GLdouble X, GLdouble Y, GL double Z)`

1.7.3 Відображення (mapping)

Полягає в обчисленні координат точок в деякій системі координат по відомих координатах тієї ж точки в іншій системі координат (наприклад, світові координати → координати спостережника).

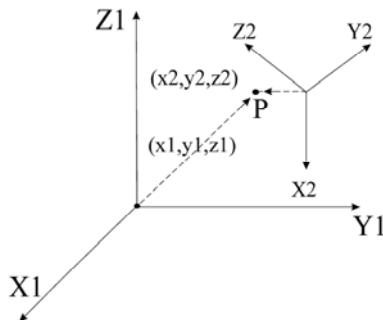


Рисунок 1.7– Відображення між системами координат.

$$\begin{bmatrix} x_2 & y_2 & z_2 & 1 \end{bmatrix}^T = T_{1-2} \begin{bmatrix} x_1 & y_1 & z_1 & 1 \end{bmatrix}^T$$

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

Припустимо $x_1 = 0$, $y_1 = 0$, $z_1 = 0$, тоді $x_2 = p_x$, $y_2 = p_y$, $z_2 = p_z$, тобто p_x, p_y, p_z – визначають координати початку відліку системи $X_1 Y_1 Z_1$ в системі координат $X_2 Y_2 Z_2$.

Якщо підставити $x_1 = 0$, $y_1 = 0$, $z_1 = 0$, то: $x_2 = n_x + p_x$; $y_2 = n_y + p_y$, де n_x, n_y, n_z – компоненти x_2 , y_2 , z_2 одиничного вектора, направлено уздовж осі x_1 системи координат $X_1 Y_1 Z_1$ (обчислюються з урахуванням взаємної орієнтації систем координат).

Аналогічно, o_x, o_y, o_z – компоненти x_2 , y_2 , z_2 одиничного вектора осі Y_1 ; a_x, a_y, a_z – компоненти x_2 , y_2 , z_2 одиничного вектора осі Z_1 .

1.7.4 Інші матриці перетворення

Для *масштабування* об'єкту відносно початку координат з коефіцієнтом S_x по осі X , S_y по осі Y , S_z по осі Z застосовується така матриця перетворення:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Відображення об'єкту відносно дзеркальної площини XY полягає у зміні знаку координати z та виконується так:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Відображення відносно дзеркальних XZ , YZ площин виводиться аналогічно.

2 БАЗОВІ АЛГОРИТМИ КОМП'ЮТЕРНОЇ ГРАФІКИ

2.1 Малювання прямої лінії

Це фундаментальна задача комп'ютерної графіки, для розв'язку якої використовуються наступні алгоритми.

Прямий алгоритм. Хай задані координати кінців відрізка прямої $(x_1, y_1)(x_2, y_2)$. Щоб намалювати відрізок необхідно визначити які пікселі мають бути задіяні між двох кінцевих точок та зафарбувати їх. Для горизонтальних та вертикальних ліній це нескладно, а для похилих – використовується спеціальний алгоритм.

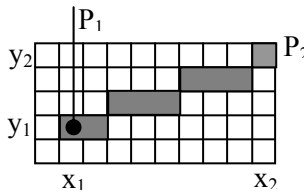


Рисунок 2.1 – Малювання відрізка прямої лінії

З елементарної алгебри відомо:

$$y = y_1 + m(x - x_1),$$

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

де m -кутовий коефіцієнт,

x - змінюється від x_1 до x_2 .

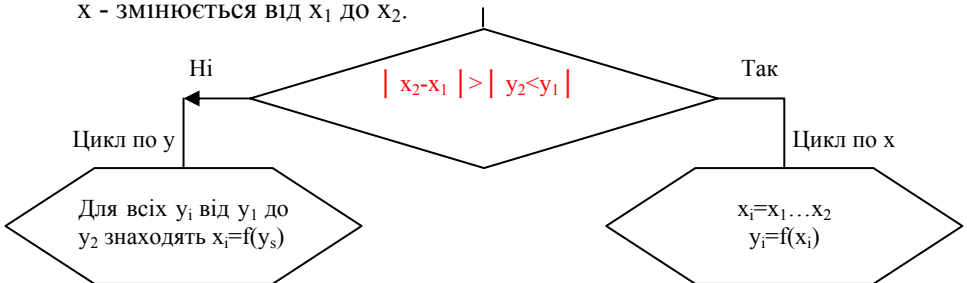


Рисунок 2.2– Прямий алгоритм

Можемо рухатись по x від x_1 до x_2 (при $x_1 < x_2$) з одиничним кроком, знаходити y та округлювати його до найближчого цілого числа.

В залежності від кута нахилу прямої, виконується цикл по осі x або y . (рис.2.2)

Недоліки: у циклі виконується багато зайвих операцій (наприклад: розрахунок m , ділення, множення), що обумовлює малу швидкість алгоритму. Якщо винести m за цикл, то доведеться використовувати операції з «плаваючою крапкою», а це й похибка представлення чисел й помилки виконання арифметичних операцій з «плаваючою крапкою». При обчисленні координат шляхом додавання прирощень може накопичуватися помилка обчислень координат і на останньому кроці виявиться, що $y \neq y_2$.

Переваги: простота та зрозумілість побудови алгоритму; можливість роботи з дробовими (нецілими) значеннями координат відрізка.

Інкрементні алгоритми (підхід Брезенхема) дозволяють побудувати цикл розрахунку координат на основі тільки цілочисельних операцій (+, - та порівняння).

Інкрементний алгоритм виконується як послідовний розрахунок координат сусідніх пікселів, шляхом додавання прирощень координат, котрі розраховуються на основі аналізу функцій похибки (положення кожного пікселя вдовж прямої, розраховується на основі інформації про попередній піксел).

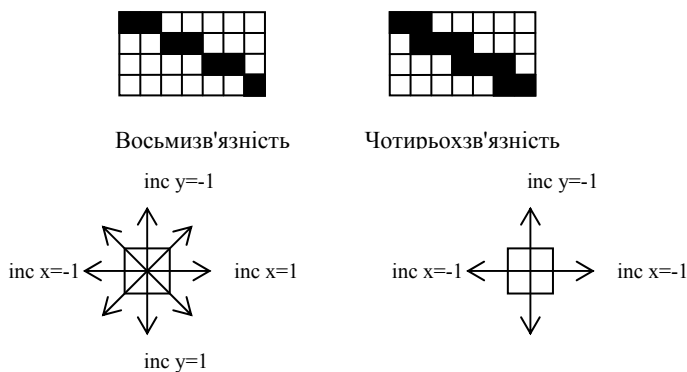


Рисунок 2.3– Випадки при обчисленні прирощень координат для переходу до сусіднього пікселя

Лістинг програми:

```

хегг = 0, уегг = 0;
dx = x2 - x1, dy = y2 - y1;
Якщо dx > 0, incx = 1;      Якщо dy > 0, incy = 1;
        dx = 0, incx = 0;      dy = 0, incy = 0;
        dx < 0, incx = -1;     dy < 0, incy = -1;
dx = | dx |, dy = | dy |;
Якщо dx > dy, то d = dx, навпаки d = dy;
x = x1, y = y1
Піксел (x,y)
Цикл d разів:
{
    хегг = хегг + dx;
    уегг = уегг + dy;
    Якщо хегг > d, то хегг = хегг - d
                    x = x + incx
    Якщо уегг > d, то уегг = уегг - d
                    y = y + incy
    Піксел (x,y) }
}
```

Переваги: підвищення швидкодії; можливість використовувати різні випадки при обчисленні прирощень координат для переходу до сусідніх пікселів (чотирьохз'язні – простіші, але генерують менш якісне зображення за більшу кількість тактів роботи).

2.2 Відсікання ліній

Це одна з основних задач комп'ютерної графіки, що застосовується для запобігання малювання частин об'єкту, які розташовуються поза заданою областю.

Алгоритм відсікання прямої передбачає визначення частини відрізка з кінцевими точками p_1 , p_2 , яка лежить усередині світового вікна і повертає кінцеві точки такої частини відрізка.

Програма приймає в якості параметрів дві 2D точки та вирівняний прямокутник (вікно), після чого відсікає межами вікна відрізок прямої, що визначається кінцевими точками p_1 , p_2 .

Якщо деяка частина даної прямої лишається у межах вікна, то змінним p_1 , p_2 присвоюються нові значення кінцевих точок та функція

повертає одиницю; якщо пряма відсікається повністю – то функція повертає нуль.

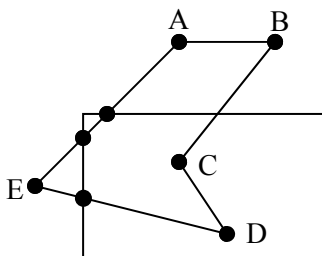


Рисунок 2.4– Відсікання прямих межами вікна

На рис.2.12 наведені різні випадки розташування відрізків прямої щодо вікна:

- відрізок CD – пряма повністю належить вікну (функція повертає 1);
- відрізок AB – пряма повністю за межами вікна (функція повертає 0);
- відрізок ED – одна кінцева точка всередині, друга за межами (функція відсікає частину відрізка за межами вікна і повертає 1);
- відрізок AE – дві кінцеві точка поза межами вікна, але відрізок проходить крізь вікно (функція відсікає два кінця і повертає 1);

Постановка завдання: необхідно ефективно ідентифікувати ситуацію (зліва, справа, над, під вікном, перетин однієї або двох границь вікна) і обчислити нові кінцеві точки для відсіченого відрізка.

Алгоритм Кохена-Сазерленда застосовує швидкий метод – «розділай і володарюй», виявляє і відкидає два розповсюджені випадки:

- тривіальний прийом (обидві кінцеві точки відрізка розташовані усередині вікна, що означає і сам відрізок теж усередині вікна);
- тривіальне відхилення(обидві кінцеві точки лежать по один бік вікна, що означає і відрізок поза вікном).

Для здійснення перевірки на тривіальний прийом або тривіальне відхилення виконується наступне.

Етап 1: для кожної кінцевої точки відрізка обчислюється «кодове слово - всередині / зовні».

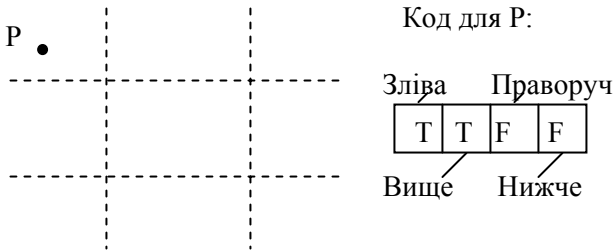


Рисунок 2.5 – Кодування розміщення точки Р відносно вікна

Таблиця 2.1 –Коди “всередині\зовні” для точки

TTFF	FTFF	FTTF
TFFF	FFFF	FFTF
TFFT	FFFT	FFTT

Етап 2: перевірка, якщо обидва кодові слова дорівнюють FFFF, то це тривіальний прийом; якщо кодові слова мають букву Т в одній і тій же позиції, то це тривіальне відхилення.

Третій випадок – не мають місце ні тривіальний прийом, ні тривіальне відхилення.

Стратегія «розділяй і володарюй» полягає в тому, що відрізок ділиться на дві частини по різні боки однієї з границь вікна. Одна його частина розташовується поза вікном, тому відкидається, а друга є потенційно видимою. Процес повторюється з відрізком, що залишився, відносно інших границь вікна.

Алгоритм припиняє роботу, виконавши цикл не більше чотирьох разів, тобто для всіх границь вікна, після чого забезпечується тривіальний прийом або тривіальне відхилення.

Розглянемо приклад для правої межі вікна(рис.2.6 б). Вибір порядку не істотний, може навіть бути випадок, коли потрібні всі чотири відсікання (рис.2.6 а).

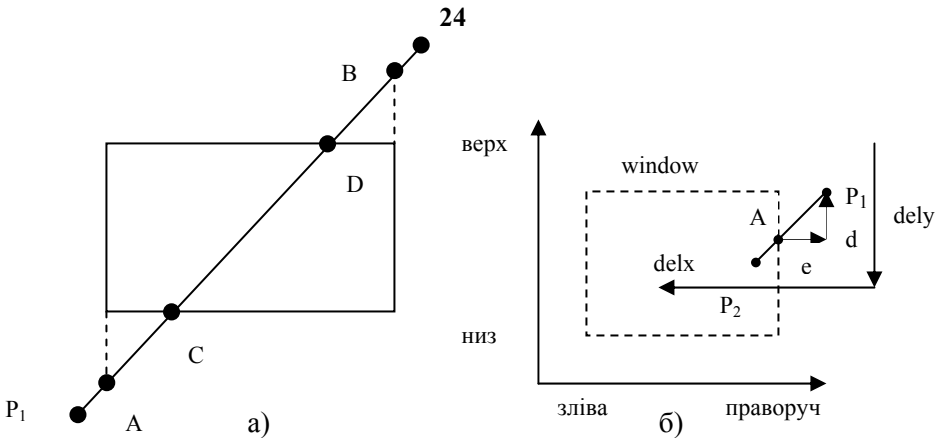


Рисунок 2.6 – Відсікання відрізка відносно правої границі вікна

Розраховуємо положення точки А. Її координата по x дорівнює правій координаті вікна $x_A = x_{wpr}$, а координату y необхідно обчислити через подобу трикутників:

$$\frac{d}{dely} = \frac{e}{delx}, \text{ де } delx, dely - \text{прирошення координат для пари}$$

кінцевих точок $delx = x_{P2} - x_{P1}$, $dely = y_{P2} - y_{P1}$;

- $e = x_{P1} - x_{wpr}$; - поправка по координаті x ;
- d - шукана поправка по координаті y .

2.3 Алгоритми зафарбовування

2.3.1 Алгоритми заповнення для піксельно визначених областей

Застосовуються для піксельно-визначеної області, яка характеризується поточними кольорами пікселів в піксельній карті, або для гранично-визначеної області довільного контура, який вже намальований в растрі.

Алгоритм зафарбовування “пробирається навпомацки”, крізь задану область, від внутрішньої точки до границь фігури: знаходять піксел всередині контура фігури; змінюють колір цього пікселя на необхідний; аналізують кольори сусідніх пікселів доти, доки всередині контуру усі пікселі не перефарбуються в колір заповнення.

При визначенні сусідніх пікселів може використовуватися два випадки: 4-х та 8-ми зв'язність(рис.2.3).

Пікселі контура – границя, за яку заборонено виходити в процесі послідовного перебору усіх сусідніх пікселів (але не будь-який контур може бути границею зафарбовування(рис.2.7а).

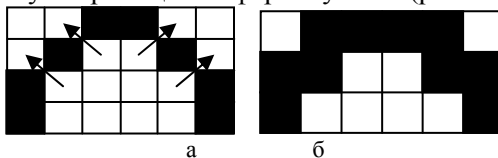


Рисунок 2.7 – Особливість 8-ми зв'язного зафарбовування

Алгоритм заливання. Заливання – це найлегший алгоритм зафарбовування, що використовується в інтерактивних системах малювання.

Етап1: визначити початкову точку всередині контура з координатами (x_0, y_0) ;

Етап2: виконати функцію.

Функція зафарбовування (x, y)

```
{
    Якщо колір пікселя  $(x, y)$  не дорівнює границі, то
    {
        Встановити для пікселя  $(x, y)$  коліром заливання;
        Зафарбовування  $(x+1, y)$ ;
        зафарбовування  $(x-1, y)$ ;
        зафарбовування  $(x, y+1)$ ;
        зафарбовування  $(x, y-1)$ ;
    }
}
```

Недоліки алгоритму:

- алгоритм рекурсивний, що спричиняє переповнення стеку комп'ютера в ході виконання програми та збій алгоритму;
- алгоритм працює «всліпу», перевіряючи сусідів незалежно від того, чи тестувались вони раніше.

Переваги алгоритму:

- простота;
- можливо побудувати цей алгоритм без рекурсії, якщо замість стеку комп'ютера використовувати окремі масиви.

Хвильовий алгоритм зафарбовування. Сутність його полягає в тому, що для початкової точки(вершини на графі) знаходяться сусідні точки (або інші вершини графа), які відповідають наступним умовам:

- ці вершини пов'язані з початковою;
- ці вершини ще не відмічені, тобто розглядаються вперше.

Хвиля пікселів зафарбовування розповсюджується від початкової точки у вигляді ромбу (рис. 2.8).

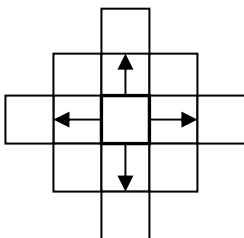


Рисунок 2.8– Розповсюдження “хвилі” кольору

Сусідні вершини поточної ітерації відзначаються в масиві опису вершин (кольором зафарбовування прямо в растрі) і кожна з них стає поточною точкою для пошуку нових сусідніх вершин наступної ітерації.

Якщо в спеціальному масиві відзначати кожен вершину номером ітерації (номер фронту хвилі), то коли буде досягнута кінцева точка, можна зробити зворотний цикл, по убаванню номерів ітерації (знайти найкоротшу відстань між двома заданими точками).

Як робочі масиви для поточного збереження координат пікселів фронтів хвиль, можна використовувати динамічні масиви місткістю, наприклад по 10 тис. елементів (максимальна місткість обумовлюється розмірами контура і розраховується емпірично).

Перевага: не використовується рекурсія, що обумовлює ефективнішу послідовність обробки пікселів при зафарбовуванні.

Недолік: мала швидкість (особливо, якщо використовувати повільну функцію setPixel для малювання окремих пікселів в програмах для Windows).

Алгоритм зафарбовування лініями. Сутність його полягає в тому, що на кожному кроці малюється горизонтальна лінія, яка розташовується між пікселями контура.

Алгоритм є рекурсивним, але оскільки виклик функції здійснюється для лінії, а не для кожного окремого пікселя, то кількість вкладених викликів зменшується пропорційно довжині лінії, що зменшує навантаження на стекову пам'ять комп'ютера.

(наприклад функція FloodFillAPIWindows)

2.3.2 Алгоритми заповнення для символічно визначених областей

В цьому випадку використовується векторна форма завдання фігури:

- а) за допомогою математичної формули $y=f(x)$;
- б) за допомогою множини точок (координат вершини полігону або багатокутника).

Алгоритми даного класу не передбачають обов'язкове попереднє створення пікселів контура растру (він може взагалі не виводитися в растр).

Заповнення прямокутників полягає в послідовному малюванні горизонтальних ліній заданого кольору.

```
for(y=y1;y<=y2;y++);
```

```
//малюємо горизонтальну лінію з координатами (x1,y)-(x2,y)
```

Заповнення кола: можна виконати на основі інкрементного алгоритму Брезенхема. В процесі цього алгоритму послідовно розраховуються координати пікселів контура у межах одного октанта.

Для заповнення виводяться горизонталі, які сполучають пари точок на контурі, які розташовані симетрично осі у.

Заповнення полігонів виконується шляхом зафарбовування відрізками прямих ліній (горизонталями) на основі алгоритму ХУ (рис.2.9).

P_1 P_3

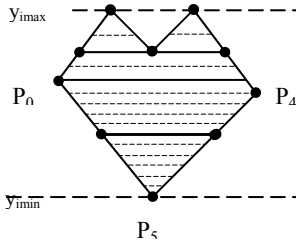


Рисунок 2.9 – Заповнення полігону

Виконується цикл уздовж осі y , в ході якого виконується пошук точок перетину ліній контура з відповідними горизонталями:

Крок 1: Знайти y_{imin} і y_{imax} серед всіх вершин P_i ;

Крок 2: Виконати цикл по осі y від y_{imin} до y_{imax} ;

{

Крок 3: Знаходження точок перетину всіх відрізків контура з горизонталлю y_i . Координати x_j точок перетину записуємо в масив;

Крок 4: Сортування масиву x_j за збільшенням x ;

Крок 5: Виведення горизонтальних відрізків з координатами $(x_0, y)-(x_1, y)$, $(x_2, y)-(x_3, y) \dots (x_{2k}, y)-(x_{2k+1}, y)$. Кожен відрізок виводиться кольором заповнення.

}

Алгоритм використовує наступну властивість топології контура фігури: будь-яка пряма перетинає будь-який замкнутий контур парну кількість разів. Для опуклих фігур на Кроці 3 алгоритму, в масив x_j завжди повинне записуватися парне число точок з'єднання.

Особливі випадки:

а) P_0, P_4 - горизонталь проходить крізь вершину і перетинає контур (в масив записується тільки одна точка перетину, а інша-вершина полігону).

б) P_1, P_2, P_3, P_5 – горизонталь торкається вершини контура (координати точки дотику або не записуються або записуються два рази)

Ця процедура визначення особливих точок уповільнює роботу алгоритму. Вирішити проблему можна таким чином:

а) змістити координати вершин контура або горизонталі заповнення так, щоб горизонталі не потрапляли у вершини (це призведе до спотворення форми полігону);

б) можна врахувати, що кожна горизонталь в більшості випадків перетинає не велику кількість ребер полігону, тобто якщо виконати попередній відбір ребер, які знаходяться навколо кожної горизонталі, можна добитися зменшення кількості тактів роботи алгоритму.

Для визначення координат x точок перетину для кожної горизонталі в загальному випадку необхідно перебирати всі n відрізків полігону.

Координата перетину відрізка $P_i - P_k$ з горизонталлю y може бути визначена так:

$$x = x_i + (y_k - y)(x_k - x_i) / (y_k - y_i)$$

Кількість тактів роботи цього алгоритму визначається таким чином

$$N_{\text{тактів}} \approx (y_{\text{imax}} - y_{\text{imin}}) * N_{\text{гор}},$$

де $y_{\text{imax}}, y_{\text{imin}}$ – діапазон координат y ;

$N_{\text{гор}} \approx k * n$ – кількість тактів для однієї горизонталі;

n – число вершин полігону;

k – коефіцієнт пропорційності.

Якщо виконати попередній відбір ребер, то $N_{\text{гор}} \approx k * n_p$, де n_p – кількість відібраних ребер. Наприклад, якщо розділити діапазон $y_{\text{imin}}, y_{\text{imax}}$ навпіл, то кількість тактів алгоритму: $N_{\text{тактів}} \approx (y_{\text{imax}} - y_{\text{imin}}) * N_{\text{гор}} / 2 + N_{\text{доп}}$, де $N_{\text{доп}}$ – кількість тактів для створення списку ребер. Цей спосіб підвищення швидкодії ефективний для великої кількості вершин полігону, причому контур можна ділити і на дрібніші частини.

2.4 Стилль лінії. Перо

Використовується для опису різних за виглядом зображень на основі ліній. Для тонкої неперервної лінії перо відповідає одному пікселу, а для товстих – це фігура або відрізок, що ковзає вздовж осі і лишає слід.

Малювання товстої лінії.

1 спосіб: можна побудувати алгоритм виведення товстої лінії на основі алгоритму Брезенхема, тільки замість виведення кожного окремого піксела поставити виведення фігури або лінії, яка відповідає перу (відрізок, прямокутник, коло). Спосіб ефективний для пір'їв у вигляді відрізків.

Превага: можливо напряму використовувати ефективні алгоритми для обчислення координат точок лінії осі (наприклад, алгоритм Брезенхема).

Недолік: підхід не ефективний для деяких форм пера(квадрат, коло) оскільки кількість тактів роботи алгоритму пропорційна квадрату товщини лінії.

2 спосіб: виведення товстої лінії, як полігону із заповненням.

Перевага: немає розривів у вузлах полілінії і добре промальовуються торці.

Недолік: менша швидкість виводу через необхідність визначення координат вершин і заповнення полігону.

3 спосіб: малювання товстої лінії накладенням декількох тонких.

Малювання штрихової тонкої лінії: алгоритм базується на інкрементному алгоритмі Брезенхема, але вводиться нова змінна – лічильник пікселів лінії, а також логічна умова, що визначає стиль лінії

Недоліки: необхідно запобігти обнуленню значення лічильника на початку кожного відрізка при виведенні поліліній; використання змінної – лічильника є складним в алгоритмах, які використовують симетрію.

Малювання штрихової товстої лінії: здійснюється шляхом об'єднання попередніх алгоритмів.

2.5 Стиль заповнення. Кисть. Текстура

Кисть(2D) – поняття для визначення стилів заповнення, що відрізняються від суцільного стилю.

Алгоритм заповнення в узагальненій формі передбачає виведення пікселів заповнення кольору C з координатами (x, y) .

Якщо $C = \text{const}$, то одержуємо суцільне заповнення, а якщо будуємо візерунок, то:

$C = f(x, y)$, де $f(x, y)$ – функція, що визначає стиль заповнення.

Наприклад, щоб отримати штрихування:

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

де S – період, T – товщина штрихів, $C_{\text{ш}}$ – колір штрихів, $C_{\text{ф}}$ – колір фону.

Для ілюзії шорсткої матової поверхні, колір C обчислюють, як випадкове значення в певних межах $C = \text{random}()$.

Для ілюзії напівпрозорої фігури: не малюють пікселі фону.

Для копіювання невеликих растрових зображень виконують наступні дії: координати пікселу заповнення (x,y) перетворюємо в растрові координати зразка кисті (x_T, y_T) ; по координатам (x_T, y_T) визначаємо колір C пікселів в зразку кисті; виводимо пікселі заповнення кольору C з координатами (x,y) .

Перетворення координат піксела заповнення (x,y) , в координати усередині зразка кисті можна виконати так:

$$x_T = x \bmod m,$$

$$y_T = y \bmod n,$$

де m, n – розміри растру кисті по горизонталі і вертикалі.

x_T, y_T – лежать в діапазоні для $x_T = 0 \dots (m-1)$, $y_T = 0 \dots (n-1)$.

Текстура (3D) – стиль заповнення, який ілюструє складну рел'єфну об'ємну поверхню, виконану з будь-якого матеріалу.

Тексел – це піксел текстури.

При відображенні 3D об'єктів у вигляді полігональних площин, кожна грань відображається шляхом накладення текстури, причому використовуються проєктивні текстури.



Рисунок 2.10 – Накладання текстур

В цьому випадку растровий зразок представляє всю грань, а перетворення координат з (x,y) в (x_T, y_T) є афінним:

$$x_T = Ax + By + C,$$

$$y_T = Dx + Ey + F,$$

де A, B, C, D, E, F - константи, при перерахунку координат всіх пікселів для окремої текстурованої грані.

Проблеми:

а) необхідний значний об'єм пам'яті комп'ютера;

б) повороти, масштабування, і інші перетворення растрових зразків приводять до погіршення якості растру;
в) складність накладання текстур в перспективній проекції, оскільки перетворення координат з видових в координати площини нелінійне.

Методи усунення:

- а) апаратна підтримка операцій з текстурами;
- б) використання методів фільтрації (інтерполяції)растрів текстур;
- в) застосування декількох зразків текстур для різних ракурсів показу(mipmaps).

3 ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

Полягає в блокуванні відображення відрізків або поверхонь, прихованих від спостерігача, з метою підвищення наочності проєкцій та полегшення сприйняття об'єкта.

3.1 Алгоритм видалення невидимих граней

Принцип заснований на тому, що грань об'єкта може бути видимою тоді, коли вектор зовнішньої нормалі цієї грані спрямований до спостерігача, інакше грань невидима.

Етап 1. Класифікація граней на видимі або невидимі. Верхня грань на рис.3.1 вважається видимою, оскільки вектор зовнішньої нормалі N має позитивну складову в напрямку вектора M , який проведено з точки зору на грані до спостерігача.

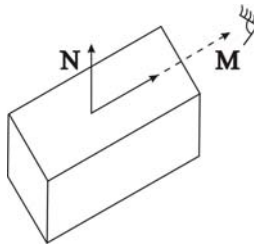


Рисунок 3.1 – Вектори, що визначають видимість граней

В загальному випадку можливі такі ситуації:

$N \cdot M > 0$ – видима поверхня;

$N \cdot M = 0$ – поверхня проєцюється у відрізок;

$N \cdot M < 0$ – невидима поверхня.

Етап 2. Після того, як всі грані класифіковані як видимі або невидимі, на екран виводяться ребра видимих граней і таким чином, отримуємо рисунок без невидимих ліній. Якщо треба одержати рисунок без невидимих поверхонь, то видимі поверхні заливуються вибраними кольорами.

Область застосування: для об'єктів, обмежених плоскими поверхнями; лише до одного випуклого об'єкту.

Обмеження алгоритму:

- увігнуті об'єкти, тому що грань, спрямована до спостерігача, може бути закрита іншою гранню цього об'єкту;
- декілька випуклих об'єктів, які можуть закривати грані один одного;
- об'єкти, для яких вектор зовнішньої нормалі визначається неоднозначно;
- криволінійні поверхні (в цьому випадку грань треба розділити силуетною лінією, на якій виконується рівність $N \cdot M = 0$).

Переваги:

- простота реалізації.

3.2 Алгоритм художника (сортування по глибині)

Принцип: полягає у тому, що поверхні об'єктів сортуються по віддаленості від спостерігача і заповнюються відповідними кольорами, розпочинаючи з найдальшої. Принцип полягає у тому, що необхідно порівняти Z_v -координати всіх поверхонь, та відобразити їх, розпочинаючи з поверхні з мінімальним Z_v .

Обмеження алгоритму:

- у більшості випадків діапазон значень Z_v точок однієї поверхні перекривається з діапазоном значень Z_v інших поверхонь.

Вирішення:

- всі поверхні перетворюються в набори трикутників (триангуляція або фасетування) таким чином, щоб діапазони Z_v різних трикутників не перекривалися один з одним, потім трикутники зафарбовуються у відповідні кольори в потрібній послідовності.

3.3 Алгоритм видалення невидимих ліній

Принцип: полягає у тому, що для кожного ребра кожного об'єкту проводиться перевірка, чи не закрите воно гранями будь-яких об'єктів. Закриті частини ребер послідовно виключаються доти, доки не залишиться поверхонь, що не перевірені. Частини ребер, що залишилися, виводяться на екран.

Етап 1. За допомогою алгоритму невидимих граней виділяються плоскі поверхні, направлені до спостерігача і зберігаються в масив FACETABLE (поверхні, направлені від спостерігача, враховувати не треба, оскільки вони є невидимими і не можуть закривати ребра інших граней). Для кожної грані зберігається

Z_v максимальне і Z_v мінімальне. Криволінійні поверхні розділяються по силуетних лініях, а їх видимі частини зберігаються в той же масив.

Етап 2. Ребра граней з масиву FACETABLE виділяються зі всієї решти ребер і збираються в окремий список.

Етап 3. Для кожного ребра із списку проводиться перевірка: чи не закривається це ребро гранню з FACETABLE. Можливі такі випадки розташування ребра:

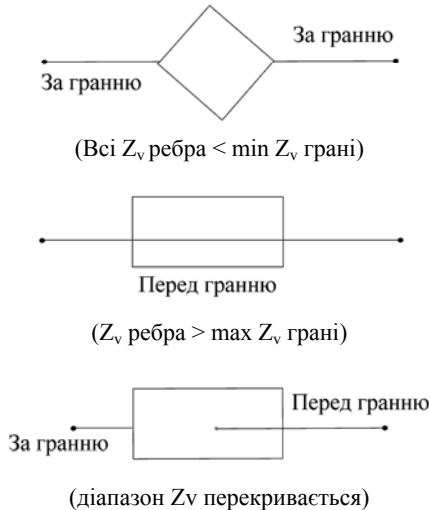


Рисунок 3.2 –Варіанти розташування ребер та граней

Якщо ребро перед гранню, що перевіряється, то з масиву FACETABLE вибирається наступна грань і виконується перевірка по ній. Якщо ребро за гранню або проходить її наскрізь – виконуються додаткові дії.

Етап 4. Ребро і грань проєціюються на екран, після чого виконується перевірка перекриття. Якщо перекриття немає, то ребро не закривається гранню, що перевіряється. Далі обирається наступна грань і перевіряється згідно етапу 3.

Якщо проєкції перекриваються – ребро ділиться на частини, при цьому невидима частина відкидається, видима додається в список ребер. Етап 3 повторюється для нових елементів списку.

Етап 5. Ребра, що пройшли перевірку з усіма гранями з FACETABLE, вважаються видимими і виводяться на екран.

Переваги:

– позбавлений обмежень алгоритму невидимих граней, тобто не залежить від кількості об'єктів, їх випуклості та наявності криволінійних площин.

3.4 Метод Z – буфера (буфер глибини)

Z – буфер – область пам'яті, подібна буферу кадру, в якому зберігаються Z – координати об'єктів, до проєкцій яких відносяться пікселі зображення. Об'єм Z – буфера визначається кількістю пікселів, для кожного з яких треба зберегти дійсне число.

Цей метод призначений для видалення прихованих поверхонь але може бути використаний для видалення прихованих ліній.
Принцип: (той же, що і у алгоритма сортування по глибині: на будь-якій ділянці екрану виявляється проєкція елемента (точки, кривої, або поверхні), розташованого ближче за інших до спостерігача. Відмінність полягає в тому, що порядок проєцювання не має значення.

Етап 1. Ініціалізація Z - буфера, координатами Z_v , відповідними дальній площині. Проецювання площини на екран; фарбування пікселів проєкції в колір дальньої площини.

Етап 2. Проецювання довільно вибраної поверхні. Всі не зафарбовані піксели, що відносяться до нової поверхні, зафарбовуються в її колір. Якщо піксели вже були зафарбовані, то відповідні їм значення Z_v порівнюються з поточними Z_v . Якщо збережене Z_v більше поточного (тобто точка на попередній поверхні ближче до спостерігача) – колір пікселя не змінюється, інакше піксел зафарбовується в колір поточної поверхні. Повторення етапу 2 для всіх поверхонь дозволяє пофарбувати піксели екрану в кольори найближчих поверхонь.

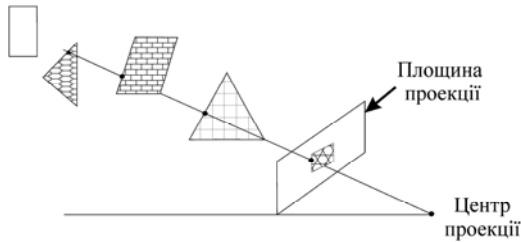


Рисунок 3.3 – Основи методу Z – буфера

Метод може бути застосований для видалення невидимих ліній.

Етап 1. Проецювання всіх поверхонь на екран, при цьому піксели зафарбовуються в колір фону. При цьому Z – буфер заповнюється значеннями Z_v найближчих до спостерігача поверхонь.

Етап 2. Проецювання на екран ребер поверхонь; порівняння Z_v – ребер і Z_v – поверхонь; фарбування пікселів видимих ребер.

Недоліки:

- значні витрати часу на візуалізацію "непотрібних" поверхонь;
- значний обсяг пам'яті для підтримки Z – буфера з достатньою точністю;
- ефекти «сходинок».

Висновок: розглянуті алгоритми видалення невидимих ліній або поверхонь мають різну форму залежно від наступних критеріїв:

- точність по об'єкту – алгоритм обчислює координати кожного видимого ребра кожної грані; ці алгоритми реалізуються в системі координат об'єкту; використовуються в САПР;
- точність по зображенню – алгоритм визначає, який елемент, якій грані є ближче до спостерігача, а необхідний піксел зафарбовується відповідним кольором; реалізується з точністю до віконної системи координат.

4 РЕНДЕРІНГ (ВІЗУАЛІЗАЦІЯ)

Для відтворення ефектів, створюваних світлом, що падає на поверхні об'єктів сцени необхідно виконати візуалізацію або рендерінг (тонування).

Існують дві основні технології візуалізації: зафарбовування та трасування променів

4.1 Затушовування

Полягає в тому, що пікселі однієї поверхні зафарбовуються різними кольорами відповідно до **кольору та інтенсивності** відбитого світла в точці, що проєцюється на цей піксел.

Етап 1. Розрахунок інтенсивності.

Поверхня будь-якого об'єкта може бути освітлена прямим світлом (від джерела) і навколишнім освітленням, що відбивається від інших поверхонь. Тому світло, що відбивається даною точкою об'єкта, отримується додатком відбитих променів двох типів (рис.4.1).

Навколишнє освітлення будемо вважати рівномірним. Інтенсивність відбиття навколишнього освітлення визначається так:

$$R_a = K_d \cdot I_d, \quad (4.1)$$

де R_a – інтенсивність відбиття;

K_d – коефіцієнт відбиття поверхні;

I_d – інтенсивність навколишнього освітлення.

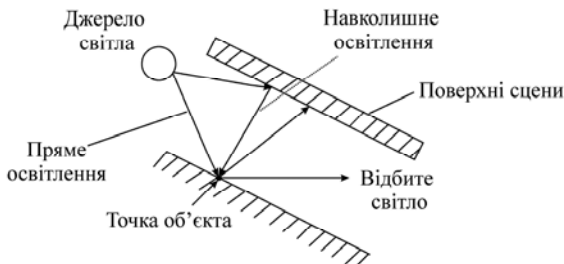


Рисунок 4.1 – Пряме й навколишнє освітлення

При будь-якому положенні спостерігача ця інтенсивність однакова.

При прямому освітленні, відбиття світла від крапкового джерела являє собою комбінацію двох видів відбиття: **дифузного й дзеркального**.

Дифузне відбиття полягає в тому, що поверхня поглинає світло, а потім перевипромінює його у всіх напрямках; дозволяє побачити особливості структури поверхні (колір і текстуру); його інтенсивність так само як і для відбиття навколишнього освітлення не залежить від положення спостерігача; характерне для грубих поверхонь (матових).

Дзеркальне відбиття – це пряме відбиття світла поверхнею; характерне для блискучих поверхонь; інтенсивність дзеркального відбиття сприймається по різному залежно від положення спостерігача.

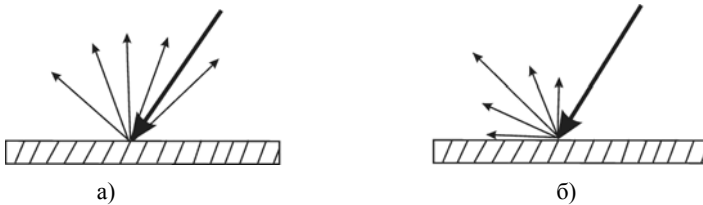


Рисунок 4.2 – Дифузне (а) та дзеркальне (б) відбиття

Інтенсивність дифузного відбиття прямо пропорційна $\cos$ кута падіння й описується співвідношенням, що називається **законом косинусів Ламберта** (рис. 4.3). З нахилом поверхні кількість променів, що падають на неї, зменшується, це означає, що зменшується і інтенсивність відбитого світла.

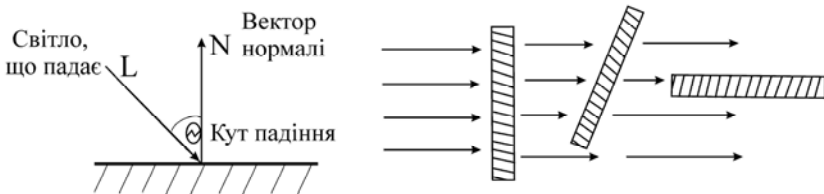


Рисунок 4.3– Ілюстрація закону Ламберта

Математично цей закон записується так:

$$I_p = E_p / D_I^2, \quad (4.2)$$

де I_p – інтенсивність падаючого світла в потрібній точці;
 D_I – відстань до джерела світла;
 E_p – інтенсивність джерела.

На практиці вважають, що джерела світла розташовані далеко від поверхонь так, що D_1 однакове для всіх точок і його можна не враховувати:

$$R_d = K_d \cdot E_p (N \cdot L), \quad (4.3)$$

де R_d – інтенсивність дифузного відбиття;

K_d – коефіцієнт відбиття поверхні;

N – одиничний вектор нормалі до поверхні в точці, що цікавить нас;

L – одиничний вектор, спрямований від точки до джерела світла;

$(N \cdot L)$ – косинус кута падіння (скалярний добуток $N \cdot L = |N| \cdot |L| \cdot \cos \theta$)

Для того, щоб з 2-х паралельних площин, які частково перекриваються на екрані, найближча була яскравішою, тобто мала більше значення R_d формулу треба змінити тким чином:

$$R_d = (K_d \cdot E_p / (D + D_0)) \cdot (N \cdot L), \quad (4.4)$$

де D – відстань між точкою, що цікавить нас, і точкою зору спостерігача;

D_0 – константа, що дозволяє уникнути ділення на 0

Інтенсивність дзеркального відбиття може бути розглянута на моделі Фонга.

Більша частина відбитого світла попадає в невеликий конічний об'єм з віссю R (рис. 4.4).

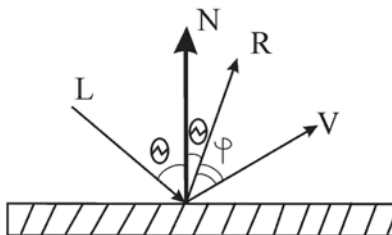


Рисунок 4.4 – Вектори, що використовуються в моделі Фонга.

Інтенсивність сприйманого спостерігачем світла зменшується по мірі віддалення вектора зору V від вектора R . Фонг описав цей ефект так:

“Інтенсивність світла, що сприймається спостерігачем, вектор зору якого відхиляється від осі R на кут φ пропорційна $(\cos \varphi)^n$.”

$$R_d = (K_s \cdot E_p / D + D_0) \cdot (V \cdot R)^n, \quad (4.5)$$

де K_s – коефіцієнт дзеркального відбиття;
 n – коефіцієнт, що характеризує тип поверхні.

Загальна інтенсивність відбиття в даній точці визначається як сума розглянутих інтенсивностей:

$$I = K_d \cdot I_d + (E_p / D + D_0) \cdot [K_d(N \cdot L) + K_s(V \cdot R)^n] \quad (4.6)$$

Етап 2. Для урахування кольору отриманий вираз записується окремо для R, G, B-складаових. Інтенсивності з індексом R, G, B (I_r, I_g, I_b) використовуються для завдання інтенсивності електронних гармат на кольорових моніторах стандарту RGB.

Наведений розрахунок інтенсивності та кольору відбитого світла стосується однієї точки об'єкта. Рішення цієї задачі для всіх точок вимагає значних обчислювальних ресурсів. Для рішення цієї проблеми треба: представити поверхню у вигляді наборів трикутних граней і розрахувати I_r, I_g, I_b один раз для грані. Для рішення проблеми плавного переходу між гранями можна використовувати два підходи:

- по методу Фонга – вектор одиничної нормалі для грані обчислюється інтерполяцією векторів нормалі у вершинах цієї грані; а вектор нормалі у вершині обчислюється усередненням векторів нормалей плоских граней, яким ця вершина належить;
- по методу Гуро інтенсивність світла обчислюється для кожного пікселя шляхом інтерполяції інтенсивностей у вершинах граней.

Недоліки методу Гуро: зафарбовування передньої грані одним кольором (рис. 4.5).

Метод Фонга більш складний, вимагає більше обчислень, але дає кращі результати, особливо для дзеркальних поверхонь.

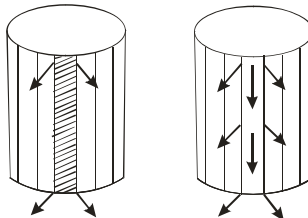
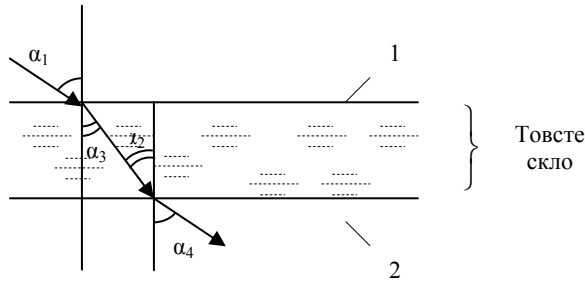


Рисунок 4.5 – Ілюстрація методу Гуро (а) і Фонга (б)

4.2 Заломлення світла прозорими об'єктами

Існує дві моделі заломлення:

а) модель ідеального заломлення – промінь відхиляється на границі двох середовищ, причому промінь, що падає, заломлений і нормаль лежать в одній площині (тут також лежить і зеркально відображений промінь)



1,2 – границі розділу середовищ

Рисунок 4.6 – Заломлення променя

Закон Снелліуса: $n_1 \sin \alpha_1 = n_2 \sin \alpha_2$, де n_1, n_2 – абсолютні показники заломлення відповідних середовищ (залежать також від стану середовища, наприклад від температури).

При проходженні крізь скло, первинний промінь світла паралельно зсувається, величина зсуву залежить від товщини скла та співвідношення показників заломлення середовищ (n_1/n_2 – відносний показник заломлення).

При заломленні відхилення променя також залежить від довжини хвилі випромінювання (чим менше довжина хвилі, тим більше він відхиляється)

б) модель дифузного заломлення – промінь, що падає, заломлюється на всі боки (молочне скло; скло вкрите льодом). Складна в реалізації.

4.3 Трасування променів

Це наймогутній підхід до візуалізації сцен і створення реалістичного зображення.

Недоліки:

- вимагає значного обсягу обчислень;
- повільний та коштовний;
- не використовується, коли об'єктів багато, частина з них прозорі, а інші заломлюють промені.

Метод трасування променів передбачає:

1. Джерела світла випромінюють промені в усіх напрямках і ці промені простежуються доти, доки не потраплять в екран. В процесі відслідковування обчислюються зміни інтенсивності і напрямку променя.

2. Кожен піксел зафарбовується у колір, відповідний інтенсивності пучку світла у момент його падіння на екран.

3. Оскільки кількість пікселів на екрані кінцева, то можна обмежитися кінцевою кількістю променів, тому будемо в зворотному напрямку відслідковувати шлях променя (із центру проекції крізь кожен піксел).

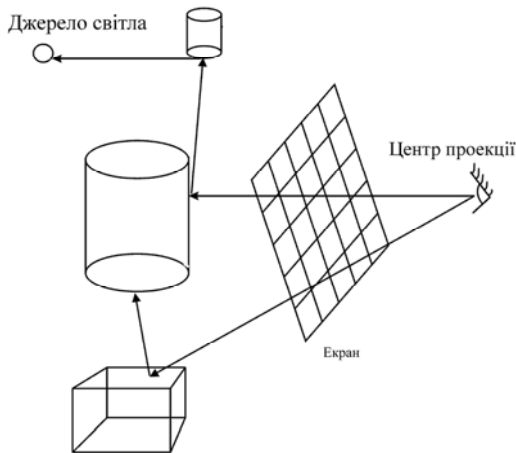


Рисунок 4.7 – Ілюстрація методу трасування променів

4. Можливі наступні варіанти траси променя:

- промені світла досягають джерела (в цьому випадку піксел фарбується в колір джерела світла);

- промінь виходить з переглянутого об'єму, тобто ніякі об'єкти на даний піксел не проєціюються (піксел фарбується в колір фону);
- промінь падає на поверхню, що розсіює (піксел фарбується в колір світла відбитого, даною поверхнею).

Переваги:

- невидимі поверхні будь-якої форми видаляються автоматично;
- оскільки траєкторію променя можна простежити по всій сцені – це дозволяє додавати інші візуальні ефекти (затемнення, відбиття, заломлення);
- дозволяє накладати на об'єкти двовимірні та тривимірні текстури;
- універсальність і можливість застосування для синтезу достатньо складних просторових об'єктів;
- для одного пікселя растрового зображення можливо трасувати декілька близько розташованих променів, що дозволяє усереднити колір для усунення ефекту «сходинок».

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Компьютерная графика / М.Н. Петров, В.П. Молочков - СПб.: Питер, 2002.-736 с.
2. Порев В.Н. Компьютерная графика.- СПб.: БХВ - Петербург, 2002.-432 с.
3. Коцюбинский А.О., Грошев С.В. Компьютерная графика. Практическое пособие. - М.: ТЕХНОЛОДЖИ-3000, 2001.-752 с.
4. Френсис Хилл. Орен GL. Программирование компьютерной графики. Для профессионалов. - СПб.: Питер, 2002.-1088 с.
5. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики.- СПб.: БХВ-Петербург, 2003.-560 с.
6. Тихомиров Ю.В. Программирование трёхмерной графики.-СПб.: БХВ-Петербург,2001.-256 с.