МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

Є. Ю. БЕРКУНСЬКИЙ

Комп'ютерна графіка та діалогові системи

Методичні вказівки для студентів
напрямку "Комп'ютерні науки"

Рекомендовано Методичною радою НУК

Миколаїв 2009

УДК 681.3.06

Беркунський Є.Ю. Комп'ютерна графіка та діалогові системи: Методичні вказівки для студентів напрямку "Комп'ютерні науки". – Миколаїв: НУК, 2009. – 40 с.

Кафедра інформаційних управляючих систем і технологій

Методичні вказівки містять завдання та теоретичні відомості з програмування задач комп'ютерної графіки. Методичні вказівки можуть бути корисними при виконанні лабораторних робіт, при роботі над завданнями курсових робіт а також при самостійному вивченні основ комп'ютерної грпафіки.

Методичні вказівки призначені для студентів усіх форм навчання напрямку "Комп'ютерні науки".

Рецензент доктор техн. наук, професор К.В. Кошкін

Згідно з наказом ректора НУК № 08 від 09.01.2008 методичні вказівки друкуються в авторській редакції і відповідальність за їх редагування несе автор.

ВСТУП

Методичні вказівки з курсу "Комп'ютерна графіка та діалогові системи" містять лабораторні роботи, метою проведення яких є ознайомлення студентів з алгоритмами і методами вирішення завдань комп'ютерної графіки. Лабораторні роботи націлені на вироблення навичок, необхідних при вирішенні практичних завдань комп'ютерної графіки з використанням сучасних мов програмування та інтегрованих середовищ розробки.

Перша лабораторна робота, що знайомить з основними методами побудови графіків функцій, базується на відомостях, які студенти отримують на заняттях з вищої математики, та відрізняється невисоким рівнем складності. При її виконанні студенти повинні познайомитися з поняттями, необхідними для виконання наступних лабораторних робіт. Друга лабораторна робота знайомить студентів з інкрементними алгоритмами – основою комп'ютерної графіки. У третій роботі представлені завдання пов'язані з алгоритмами афінних перетворень на площині, які крім суто теоретичних знань, надають можливості які у подальшому будуть використовуватися як "останній крок" усіх візуалізацій. Четверта, п'ята та шоста лабораторні роботи знайомлять студентів з різними аспектами комп'ютерної візуалізації тривимірних об'єктів.

Практикум включає короткі теоретичні відомості по кожній з лабораторних робіт. Теоретичний матеріал викладений стисло і розрахо-

ваний на студентів, які вперше вивчають даний предмет. Наведені приклади фрагментів програм із використанням псевдокоду та мови Java або C++.

Для виконання практикуму студенти повинні мати навички роботи на ЕОМ та володіти основами мов програмування.

Лабораторний практикум проводиться кожним студентом індивідуально. Для цього студент отримує одне із завдань відповідно до вказаного викладачем варіанта. Робота виконується згідно до даних методичних вказівок під керівництвом викладача. Перед виконанням роботи на ЕОМ студентові слід отримати до неї допуск. При виконанні лабораторної роботи студент повинен пред'явити робочу версію програми і результат її роботи на ЕОМ.

Лабораторна робота вважається виконаною після її захисту. Для захисту роботи необхідно представити результати виконання програми на ЕОМ і звіт, оформлений відповідно до вимог.

Лабораторна робота № 1.
Побудування графіків функцій

Завдання

1. Створити програму із зручним інтерфейсом користувача, що буде реалізовувати можливість побудови графіка функції відповідно до варіанта завдання.

2. Реалізувати в програмі можливість діалогового режиму введення меж інтервалу, на якому будується графік функції.

Варіанти завдань

№ варіанту	Функція	Інтервал
1	$y = \cos(x^2)/(x+1)$	[0, 2.0]
2	$y = \sin(x^2 - 1)/\sqrt{x}$	[0.1, 3.0]
3	$y = \cos(x + 0{,}1)/(x^2 + 1)$	[1.0, 3.0]
4	$y = (1 + 0{,}55x^2)/(1{,}5 + \sqrt{0{,}2 + x^2})$	[0, 3.0]
5	$y = (0{,}1 + 0{,}3x^3)/(5 + \sqrt{0{,}15 + x^4})$	[1.0, 5.0]
6	$y = \lg(x^2 + 2)/(x + \sin x)$	[1.0, 4.5]
7	$y = \operatorname{tg}(x^2)/(x^2 + 1)$	[0, 1.4]
8	$y = \lg(x^3 - 1{,}2)/(x^2 + \cos x)$	[1.2, 5.0]
9	$y = \cos(1 + 0{,}5x^2)/(1{,}5 + \sqrt{0{,}2 + x})$	[1.0, 3.0]
10	$y = (1{,}3 + 0{,}5x^4)/\ln(1{,}14 + \sqrt{0{,}34 + x^2})$	[0, 3.0]

Теоретичні відомості

Для побудови графіка функції $y = f(x)$ на папері обирається прямокутний діапазон $(X_1, X_2)*(Y_1, Y_2)$. Головна проблема, що виникає при побудові графіків функцій на екрані монітора, зумовлена необхідністю перетворення прямокутника $(X_1, X_2)*(Y_1, Y_2)$ у прямокутник на екрані $(Xs_1, Xs_2)*(Ys_1, Ys_2)$. Це перетворення складається з операцій масштабування і переносу, які можна реалізувати таким чином [3]:

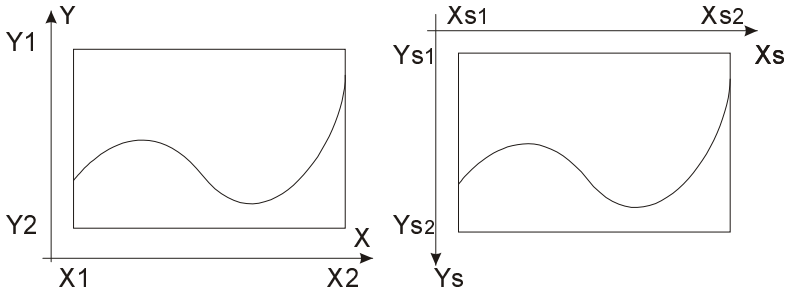

Рис. 1. Координатні простори: світовий та екранний

Масштабування по осях OX та OY реалізується за допомогою лінійних залежностей

$$\frac{x - X_1}{X_2 - X_1} = \frac{xs - Xs_1}{Xs_2 - Xs_1}$$
$$\frac{y - Y_1}{Y_2 - Y_1} = \frac{ys - Ys_1}{Ys_2 - Ys_1}$$

При обчисленні вертикальних екранних координат необхідно за допомогою знаку враховувати, що нумерація рядків іде згори донизу. Наведемо приклад класу Java, що може бути використаний для виконання операцій масштабування.

```
public class WindowParams {
  private double minX,  maxX,  minY,  maxY;
  private int i1,  i2,  j1,  j2;
  public WindowParams(double minX, double maxX,
      double minY, double maxY,
      int i1, int i2, int j1, int j2) {
        this.minX = minX;
        this.maxX = maxX;
        this.minY = minY;
        this.maxY = maxY;
        this.i1 = i1;
        this.i2 = i2;
        this.j1 = j1;
        this.j2 = j2;
  }
```

```
    public int toScreenX(double x) {
       return  (int)(i1+Math.round((x-minX)*(i2-i1)/
           (maxX-minX)));
    }
    public int toScreenY(double y) {
      return   (int)(j2+Math.round((y-minY)*(j1-j2)/
          (maxY - minY)));
    }
    public double toRealX(int i) {
      return minX + (i-i1)*(maxX-minX)/(i2-i1);
    }
    public double toRealY(int j) {
      return minY + (j-j2)*(maxY-minY)/(j1-j2);
    }
    public double getMaxX() {
      return maxX;
    }
    public double getMaxY() {
      return maxY;
    }
    public double getMinX() {
      return minX;
    }
    public double getMinY() {
      return minY;
    }
  }
```

Методи toScreenX() та toScreenY() наведеного класу виконують перерахунок координат із реальних (світових) у екранні. А методи toRealX() та toRealY() – зворотне перетворення.

Лабораторна робота № 2.
Інкрементні алгоритми

Завдання

1. Створити програму із зручним інтерфейсом користувача, що буде реалізовувати можливість побудови рисунку відповідно до варіанта завдання.

2. Єдиною графічною операцією, що дозволяється використовувати, є операція побудови точки. Всі інші графічні елементи – лінії, що утворюють багатокутники контурів символів, повинні будуватися із використанням інкрементних алгоритмів Брезенхема.

3. Реалізувати в програмі можливість діалогового режиму вибору кольору зображення.

Варіанти завдань

№ варіанту	Завдання
1	Перша літера вашого прізвища та цифра 1
2	Перша літера вашого імені та цифра 2
3	Перша літера вашого прізвища та цифра 3
4	Перша літера вашого імені та цифра 4
5	Перша літера вашого прізвища та цифра 5
6	Перша літера вашого імені та цифра 6
7	Перша літера вашого прізвища та цифра 7
8	Перша літера вашого імені та цифра 8
9	Перша літера вашого прізвища та цифра 9
10	Перша літера вашого імені та цифра 0

Примітка. *Літера і цифра повинні бути зображені як такі, що при вирізанні з листкового металу не розпадаються на частини, наприклад, літера Ю, може бути зображена так* (*рис. 2*)*:*

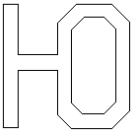

Рис. 2. Приклад зображення літери

Теоретичні відомості

При побудові графічних зображень будь-яка графічна бібліотека використовує операції рисування графічних примітивів типу "точка", "пряма", "коло", "еліпс" і т.п. Проте, в основі будь-якої графічної бібліотеки лежить операція відображення точки. А інші, більш складні фігури зображуються за допомогою виведення багатьох точок.

Розглянемо, наприклад, як використовуючи лише операцію виведення точки, можна намалювати відрізок прямої лінії (рис. 3).

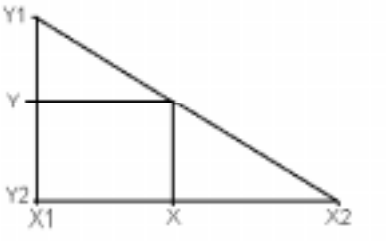

Рис. 3. Зображення відрізка прямої ліній

Кожен відрізок визначається координатами початку і кінця: (*x1, y1*) та (*x2, y2*). Як відомо, через кожні дві точки проходить єдина пряма, її рівняння можна представити у вигляді $\frac{y-y1}{y2-y1}=\frac{x-x1}{x2-x1}$, або $(x2-x1)(y-y1)=(x-x1)(y2-y1)$, звідки випливає $y=y1+$ $+\frac{x-x1}{x2-x1}(y2-y1)$, тобто $y = f(x)$ або $x=x1+(y-y1)\frac{x_2-x_1}{y_2-y_1}$, тобто $x = f1(y)$.

В залежності від кута нахилу прямої виконується цикл вздовж вісі *x* або *y*, якщо $|x_2 - x_1| > |y_2 - y_1|$, то виконується цикл по *x*, якщо ні – то по *y*.

На основі цих формул можливо побудувати алгоритм виведення прямої лінії, у якому, як легко бачити, буде міститись операція ділення, проте, оскільки на екран треба виводити тільки цілі координати, то доведеться ще використовувати операцію округлення. Крім того, як відомо, операції з дійсними числами на більшості комп'ютерних платформ виконуються повільніше, ніж із цілими числами. Тому, бажано розробити такий алгоритм побудови прямих ліній, який би не використовував операцій ділення та округлення. Оскільки початкові та кін-

цеві координати відрізка прямої є цілими числами, то всі дії в ньому будуть лише над цілими числами.

Брезенхем запропонував підхід, що дозволяє розробляти так звані інкрементні алгоритми растеризації. Основною метою для розробки таких алгоритмів була побудова циклів обчислення координат на основі тільки операцій з цілими числами – додавання/віднімання, та без застосування множення та ділення. На теперішній час відомі інкрементні алгоритми не тільки для відрізків прямих, але і для кривих ліній.

Інкрементні алгоритми виконуються як послідовне обчислення координат сусідніх пікселів шляхом додавання приростів координат. Прирости обчислюються на основі аналізу функції похибки. В циклі виконуються лише цілочислові операції порівняння і додавання/віднімання. Таким чином досягається збільшення швидкості обчислення кожного пікселя у порівнянні з прямим шляхом на основі формули рівняння прямої.

Наведемо один з варіантів алгоритму Брезенхема для прямої (всі змінні, що використовуються в цьому алгоритмі, лише цілі):

```
xErr = 0; yErr = 0;
dx = x2 – x1; dy = y2 – y1;
Якщо dx > 0 то incX = 1;
     dx = 0 то incX = 0;
     dx < 0 то incX = -1;
Якщо dy > 0 то incY = 1;
     dy = 0 то incY = 0;
     dy < 0 то incY = -1;
dx = |dx|; dy = |dy|;
Якщо dx > dy
  то d = dx
  інакше d = dy;
x = x1; y = y1;
малюватиПіксел(x,y);
Виконати цикл d разів {
     xErr += dx;
     yErr += dy;
     Якщо xErr > d, то {
          xErr -= d; x += incX;
      }
      Якщо yErr > d, то {
           yErr -= d; y += incY;
      }
      малюватиПіксел(x,y);
}
```

Для реалізації цього алгоритму за допомогою будь якої мови програмування, треба представити інструкції псевдокоду у вигляді інструкцій мови, крім того, треба використати одну з існуючих графічних бібліотек обраної мови програмування. В обраній бібліотеці треба скористатися функцією, що малює точку на екрані та замінити нею інструкцію "малюватиПіксел".

Крім алгоритмів побудови прямих ліній, Брезенхем запропонував також алгоритми побудови деяких кривих: кола, еліпса та ін.

Наведемо інкрементний алгоритм виведення кола з центром у точці **xc**, **yc** та радіусом **radius**, при цьому будемо вважати, що в нас є визначена така функція sim, що малює точки симетрично відносно точки xc, yc та прямих, що проходять через цю точку паралельно осям координат.

```
sim(int x, int y, Color col) {
    малюватиПіксел(x+xc, y+yc, col);
    малюватиПіксел(x+xc, -y+yc, col);
    малюватиПіксел(-x+xc, -y+yc, col);
    малюватиПіксел(-x+xc, y+yc, col);
    малюватиПіксел(y+xc, x+yc, col);
    малюватиПіксел(y+xc, -x+yc, col);
    малюватиПіксел(-y+xc, -x+yc, col);
    малюватиПіксел(-y+xc, x+yc, col);
}

d = 3 – 2 * y;
x = 0;
y = r;
поки (x <= y) {
      sim(x,y);
      якщо (d<0)
      то {
          d = d + 4 * x + 6;
      }
      інакше {
          d = d + 4 * (x - y) + 10;
          dec(y);
      }
      x++;
}
```

В цьому алгоритмі використано симетрію кола – в основному циклі обчислюються координати точок кола тільки для одного октанта (між променями OA і OB) і одразу малюються вісім симетрично розташованих пікселів (рис. 4):

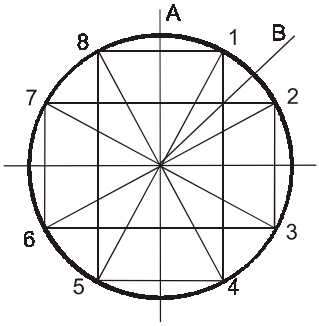

Рис. 4. Симетрія кола

Лабораторна робота № 3. Афінні перетворення на площині

Завдання

1. Створити програму, що матиме зручний інтерфейс (вікно, головне меню, рядок статусу та ін.). У вікні програми повинно створюватися та відображатись зображення фігури, створеної відповідно завдання лабораторної роботи № 2.

2. За допомогою операцій з матрицями і векторами реалізувати афінні перетворення фігури (переміщення по екрану, збільшення/зменшення, повороти). Програма повинна надавати можливість виконання над фігурою трьох основних афінних перетворень, при цьому вибір точки, відносно якої буде відбуватися поворот також повинен обиратись користувачем (приклад зображено на рис. 5).

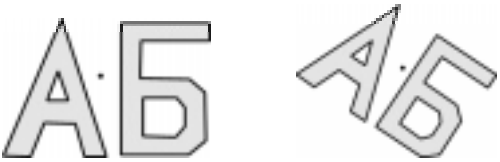

Рис. 5. Завдання до лабораторної роботи № 3

Теоретичні відомості

Задамо якусь двовимірну систему координат (x, y). Афінне перетворення на площині описується формулами

$$X = Ax + By + C$$

$$Y = Dx + Ey + F\,,$$

де A, B, C, D, E, F – константи. Значення (X, Y) можна розглядати, як координати в новій системі координат. Обернене перетворення (X, Y) у (x, y) також є афінним.

Афінні перетворення зручно записувати у матричному вигляді. Константи A, B, D, E утворюють матрицю перетворення, котра, будучи помножена на матрицю-стовпець координат (x, y) дає матрицю-стовпець (X, Y). Однак щоб урахувати константи C та F, необхідно перейти до так званих ***однорідних координат*** – додамо ще один рядок у матрицях координат:

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

Матричний запис дає можливість наочно описувати декілька перетворень, що йдуть одні за одними. Наприклад, якщо необхідно спочатку виконати перетворення

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} & A & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

а потім — інше перетворення

$$\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \begin{bmatrix} & B & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix},$$

то це можна описати як

$$\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \begin{bmatrix} & B & \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} & B & \end{bmatrix} \begin{bmatrix} & A & \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Однак замість двох перетворень можна виконати тільки одне

$$\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \begin{bmatrix} & C & \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

де матриця С дорівнює добутку ВА.

Тепер розглянемо окремі випадки афінного перетворення.

1. Паралельний зсув координат (рис. 6)

На рис. 6 точка *A* перетворюється у точку *B*.

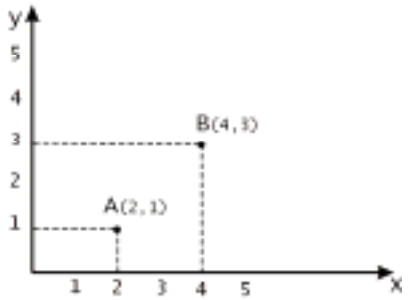

Рис. 6. Операція паралельного зсуву точки А в точку В

$$\begin{cases} X = x + dx \\ Y = y + dy \end{cases}$$ У матричній формі: $$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}.$$

2. Розтягнення-стискання, або масштабування (рис. 7)

Масштабуванням об’єктів називається розтягнення об’єктів вздовж відповідних осей координат відносно початку координат. Ця операція застосовується до кожної точки об’єкта, тому можна також говорити про масштабування точки. При цьому, звісно, мова не іде про зміну розмірів самої точки. Масштабування досягається множен-

ням координат точок на деякі константи. В тому випадку, коли ці константи однакові, масштабування називають однорідним. На рис. 7 наведено приклад однорідного масштабування трикутника ABC.

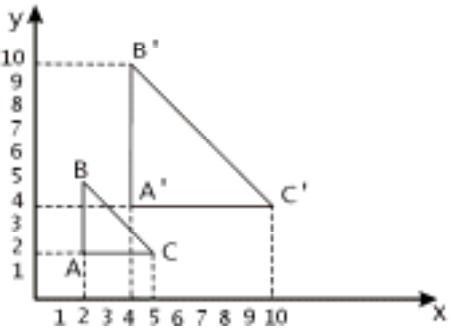

Рис. 7. Операція масштабування

Після застосування операції однорідного масштабування з коефіцієнтом 2 він переходить в трикутник A'B'C'.

Масштабування можна представити такими формулами:

$$\begin{cases} X = x * S_x \\ Y = y * S_y \end{cases}$$, або у матричній формі $$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Коефіцієнти S_x та S_y можуть бути від'ємними. Наприклад, $S_x = -1$ відповідає дзеркальному відбиттю відносно осі Y.

3. **Поворот** (рис. 8)

На рисунку 8 точка A(x, y) переходить в точку B(x', y') поворотом на кут α.

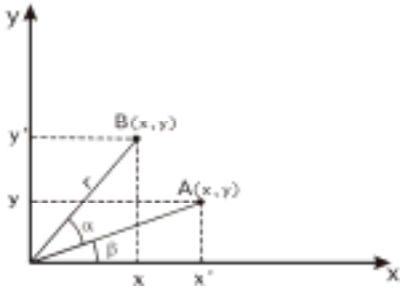

Рис. 8. Операція повороту точки A на кут α

Знайдемо перетворення координат точки A в точку B. Позначимо кут, що утворює радіус-вектор OA з віссю Ox як β. Нехай r – довжина радіус-вектора OA, тоді

$$x' = r \cdot \cos(\alpha + \beta) = r \cdot (\cos\alpha \cdot \cos\beta - \sin\alpha \cdot \sin\beta)$$
$$y' = r \cdot \sin(\alpha + \beta) = r \cdot (\sin\alpha \cdot \cos\beta + \cos\alpha \cdot \sin\beta).$$

Оскільки $\cos\beta = \frac{x}{r}$ та $\sin\beta = \frac{y}{r}$, то підставляючи ці вирази в рівняння для x' и y', отримуємо:

$$x' = x \cdot \cos\alpha - y \cdot \sin\alpha$$
$$y' = x \cdot \sin\alpha + y \cdot \cos\alpha.$$

У матричному вигляді:

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

Розглянемо, яким образом за допомогою композиції матричних перетворень можна одержати одне загальне результуюче перетворення. Для цього будемо використовувати матриці T, S та R. З обчислювальної точки зору набагато простіше й швидше застосовувати матрицю вже готового перетворення замість того, щоб застосовувати їх послідовно одну за іншою. До точки більш ефективно застосовувати одне результуюче перетворення, ніж ряд перетворень один за одним.

Для прикладу розглянемо завдання повороту об'єкта на площині щодо деякої довільної точки (X,Y). Поки ми вміємо повертати об'єкти тільки навколо початку координат. Але можна представити це завдання як послідовність кроків, на кожному з яких буде застосовуватися тільки елементарна операція: зсув, масштабування або обертання.

Ось ця послідовність елементарних перетворень (рис. 9):

1. Зсув, при якому точка p_0 переходить в початок координат.
2. Поворот на вказаний кут.
3. Зсув, при якому точка з початку координат повертається в початкове положення p_0.

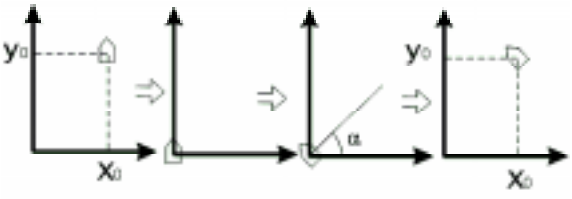

Рис. 9. Послідовність перетворень
при повороті навколо точки $p_o = (x_0, y_0)$ на кут α

Точка $p_o = (x_0, y_0)$. Перший зсув відбувається на вектор $[-x_0, -y_0]$, а зворотний перенос - на вектор $[x_0, y_0]$.

Лабораторна робота №4.
Побудова проекцій тривимірних об’єктів

Завдання

1. Розробити програму, що будує точкову (перспективну) проекцію тривимірного об’єкта. Об’єкт для побудови вибрати самостійно та затвердити у викладача. Об’єкт повинен мати понад 50 базових точок і складатися із 20 або більше багатокутників (полігонів) .

2. Створена програма повинна надавати можливість афінних перетворень фігури, що відображає збудовану проекцію.

Теоретичні відомості

Процес виведення тривимірної графічної інформації по суті є складнішим, ніж відповідний двовимірний процес. Складність, характерна для тривимірного випадку, ґрунтується на тому, що поверхня виведення не має графічного третього виміру. Така невідповідність між просторовими об’єктами і пласкими зображеннями усувається шляхом введення проекцій, які відображують тривимірні об’єкти на двовимірній проекційній картинній площині. В процесі виведення тривимірної графічної інформації ми задаємо видимий об'єм в світовому просторі, проекцію на картинну плоскість і поле виводу на видовій поверхні. У загальному випадку об'єкти, визначені в тривимірному світовому просторі, відсікаються по межах тривимірного видимого об'єму і після цього проектуються. Те, що потрапляє в межі вікна, яке само є проекцією видимого об'єму на картинну плоскість, потім перетвориться в полі виводу і відображується на графічному пристрої. У загальному випадку операція проекції перетворює точки, задані в системі координат розмірності n, в точки системи координат розмірності меншій, ніж n. У нашому випадку точка тривимірного простору відображується в двовимірний простір. Проекція тривимірного об’єкта будується за допомогою прямих променів, що його проецирують. Ці промені називаються проекторами і вони виходять з центру проекції, проходять через кожну точку об’єкта і, при перетинанні картинної площини, створюють проекцію. На рис. 10 представлені дві різні проекції одного й того самого відрізку і проектори, які проходять через його кінцеві точки.

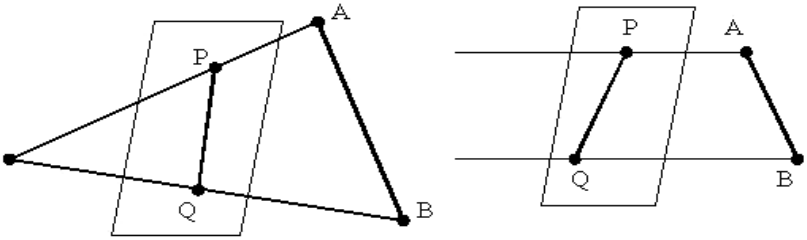

Рис. 10. Побудова проекцій

Визначений таким чином клас проекцій відомий під назвою плоских геометричних проекцій, оскільки проектування здійснюється на площину, а не на викривлену поверхню і як проектори використовують прямі лінії. Пласкі геометричні проекції можна розділити на два основні класи: центральні (перспективні) і паралельні (ортогональні). Відмінність між ними визначається співвідношенням між центром проекції і проекційною плоскістю. Так, якщо відстань між ними, звичайна, то проекція буде центральною, якщо ж вона нескінчена, то – паралельною. При описі центральної проекції ми явно задаємо її центр, тоді як для паралельної проекції ми вказуємо лише напрям проектування. Центр проекції породжує візуальний ефект, аналогічний тому, до якого приводять фотографічні системи і використовується у випадках, коли бажано досягти деякої міри реалізму. Слід зауважити, що розмір центральної проекції об'єкту змінюється зворотно пропорційно до відстані від центру проекції до об'єкту. Паралельна проекція породжує менш реалістичне зображення, оскільки відсутнє перспективне "укорочення" об'єкту. Проекція фіксує дійсні розміри об'єкту, і паралельні лінії залишаються паралельними.

У загальному випадку завдання побудови центральної проекції полягає в тому, щоб визначити проекцію точки об'єкту, розташованої в довільному місці тривимірного простору, на деяку площину в цьому ж просторі. Цю площину називають картинною. Знаходження центральної проекції є окремим випадком завдання визначення перетинання променя **L** з площиною α в тривимірному просторі (рис. 11).

У комп'ютерній графіці завдання обчислення центральної проекції, як правило, сильно спрощене. В даному випадку центр проекції, який також називають точкою зору, знаходиться на одній з осей си-

стеми координат, картинна (проекційна) площина перпендикулярна оптичній осі. Як правило, точку зору (центр проекції) розташовують на осі **OZ**, тоді картинна площина буде паралельна площині **OXY** системи координат (рис. 12).

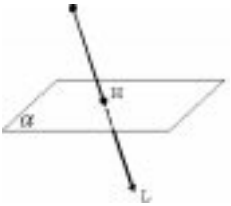

Рис. 11. Перетинання променя з площиною

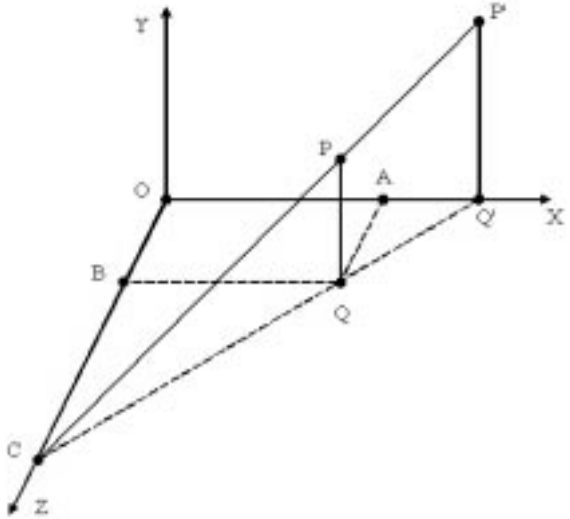

Рис. 12. Схема побудови центральної проекції

В нашому випадку точка C = (0, 0, c) – центр проекції (положення спостерігача), площина z = 0 – картинна площина. Нехай точка P = (x, y, z) має проекцію P' = (x', y', 0). Розглянемо два подібних трикутника *CPQ* та *CP'Q'*, і запишемо відношення катетів: $\frac{P'Q'}{PQ} = \frac{C'Q'}{CQ}$. Розглянемо два інших подібних трикутника *CQ'O* і *CQB*, і запишемо відношення катетів для них: $\frac{OQ'}{BQ} = \frac{CQ'}{CQ}$. З іншого боку маємо: $\frac{OQ'}{BQ} =$

$= \frac{OC}{BC} = \frac{c}{c-z}$. Оскільки $OQ' = x'$, $BQ = x$, $P'Q' = y'$, $PQ = y$ маємо

$$\begin{cases} \frac{x'}{x} = \frac{c}{c-z} \\ \frac{y'}{y} = \frac{c}{c-z} \end{cases},$$

або після перетворень

$$\begin{cases} x' = x\frac{c}{c-z} = x\frac{1}{1-\frac{z}{c}} \\ y' = y\frac{c}{c-z} = y\frac{1}{1-\frac{z}{c}} \end{cases}.$$

Якщо тепер $c \to \infty$, то отримаємо формулу паралельної проекції:

$$\begin{cases} x' = x \\ y' = y \end{cases}.$$

Наступним кроком необхідно спроектоване зображення перевести в координати екрану. Це можна виконати таким чином:

$$\begin{cases} X = X_0 + x'l \\ Y = Y_0 + y'l \end{cases},$$

де (X_0, Y_0) – середина екрану, l – кількість пікселів в одиниці.

Існує зв'язок однорідних координат з операціями центральної і паралельної проекцій, який може бути виражений так:

$$\begin{pmatrix} X \\ Y \\ Z \\ H \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{c} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1-\frac{z}{c} \end{pmatrix}.$$

Для переходу від однорідних координат до звичайних, необхідно розділити всі компоненти точки на четверту координату:

$$\begin{cases} x' = \dfrac{x}{1 - \dfrac{z}{c}} \\ y' = \dfrac{y}{1 - \dfrac{z}{c}} \end{cases}.$$

Для паралельної проекції матриця перетворення буде мати вигляд:

$$\begin{pmatrix} X \\ Y \\ Z \\ H \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix}.$$

Таким чином, крок проектування можна описати в термінах матричної операції добутку. В результаті цього ми можемо поєднати разом операції перетворення об'єкта (перенесення, масштабування, обертання які потрібно буде реалізувати в наступній лабораторній роботі) і операцію проектування в одну загальну матрицю перетворення. Аналогічно можна зробити з приведенням точок, які проектуються, до екранних координат:

$$\begin{pmatrix} X \\ Y \\ Z \\ H \end{pmatrix} = \begin{pmatrix} l & 0 & 0 & X_0 \\ 0 & l & 0 & Y_0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} x'l \\ X_0 y' l + Y_0 \\ 0 \\ 1 \end{pmatrix}$$

$$\begin{cases} X_{екр} = x'l + X_0 \\ Y_{екр} = y'l + Y_0 \end{cases}.$$

Таким чином, всі операції перетворення об'єкту тривимірного простору на картинну площину (екран) можна описати в термінах матричних добутків.

Лабораторна робота № 5.
Афінні перетворення у просторі

Завдання

1. Створити програму, що матиме зручний інтерфейс (вікно, головне меню, рядок статусу та ін.). У вікні програми повинно створюватися та відображатись зображення фігури, створеної відповідно завдання лабораторної роботи №4.

2. За допомогою операцій з матрицями і векторами реалізувати афінні перетворення фігури (переміщення у просторі, збільшення або зменшення, повороти). Програма повинна надавати можливість виконання над фігурою трьох основних афінних перетворень.

Теоретичні відомості

Якщо задана тривимірна система координат (x, y, z), то будь-яке перетворення у просторі буде афінним, якщо воно може бути описаним за допомогою формул:

$$X = Ax + By + Cz + D$$
$$Y = Ex + Fy + Gz + H$$
$$Z = Kx + Ly + Mz + N,$$

де A, B, C, D, E, F, G, H, K, L, M, N – константи. Значення (X,Y, Z) можна розглядати, як координати в новій тривимірній системі координат. Обернене перетворення (X, Y, Z) у (x, y, z) також є афінним. Як зазначалося у лабораторній роботі №3, афінні перетворення зручно записувати у матричному вигляді. Константи A, B, C, E, F, G, K, L, M утворюють матрицю перетворення, котра, після множення на стовпець (x, y, z) утворює стовпець (X, Y, Z). Так само, як і для перетворень на площині, щоб урахувати константи D, H, N необхідно перейти до однорідних координат. Для цього додамо ще один рядок у матрицях координат:

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

Як уже розглядалось у лабораторній роботі №3, усі афінні перетворення можна представити через послідовне застосування трьох основних перетворень: паралельного зсуву, масштабування та повороту.

Розглянемо їх докладніше:

1. Паралельний зсув

Аналогічно до випадку перетворень на площині, паралельний зсув у просторі можна представити за допомогою рівнянь:

$$\begin{cases} X = x + dx \\ Y = y + dy \\ Z = z + dz \end{cases}$$, або у матричній формі: $$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2. Розтягнення/стискання (Масштабування)

Як і у випадку перетворень на площині, масштабуванням об'єктів будемо називати їхнє розтягнення вздовж відповідних осей координат відносно початку координат. Звичайно, така операція застосовується до кожної точки об'єкта, тому говорять про масштабування точок. Найчастіше використовують однорідне масштабування, коли коефіцієнти розтягнення вздовж усіх трьох координатних осей однакові. Масштабування у просторі можна представити такими формулами:

$$\begin{cases} X = x * S_x \\ Y = y * S_y \\ Z = z * S_z \end{cases}$$, або у матричній формі: $$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Звичайно, як і у випадку на площині, коефіцієнти *Sx, Sy, Sz* можуть бути як додатними, так і від'ємними.

3. Повороти

Тривимірні перетворення обертання є складнішими, ніж їх двовимірні аналоги. В даному випадку необхідно додатково задати вісь обертання. Розглянемо спочатку прості випадки, коли вісь обертання збігається з однією з координатних осей.

Знайдемо матрицю повороту довкола осі OZ на кут φ. Запишемо матрицю перетворення для лівобічної системи координат. Слід зазначити, що в лівобічній системі координат позитивними будуть поворо-

ти, що виконуються за годинниковою стрілкою, якщо дивитися з кінця позитивної піввісі у напрямку початку координат (рис. 13).

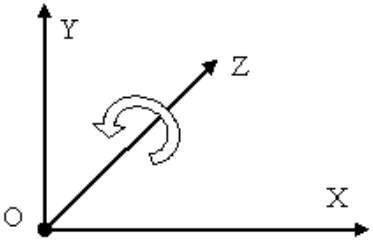

Рис. 13. Поворот навколо вісі Z

В даному випадку вісь повороту перпендикулярна до плоскості рисунка, і оскільки ми використовуємо лівобічну систему координат, то обертання довкола осі *OZ* зводиться до повороту точки на площині *XOY* на кут φ. При цьому координата *z* точки обертання не змінюється. Таким чином, формулу повороту точки *(x,y,z)* довкола осі *OZ* на кут φ можна записати таким чином:

$$\begin{cases} x' = x\cos\varphi - y\sin\varphi \\ y' = x\sin\varphi + y\cos\varphi \\ z' = z \end{cases}$$

або в матричній формі

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}.$$

Змінимо тепер положення координатних осей лівобічної системи координат так, щоб вісь **OY** була направлена в площину рисунка. Тоді позитивна піввісь **OZ** буде направлена горизонтально праворуч, а позитивна піввісь **OX** – вертикально вгору (рис. 14).

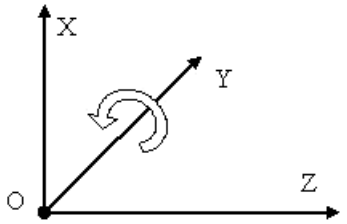

Рис. 14. Поворот навколо вісі Y

Отримати формулу повороту точки довкола осі *OY* на кут β можна замінивши *x* на *z*, *y* на *x* у формулі двовимірного повороту. При цьому координата *y* точки в при такому обертанні не змінюється. Внаслідок чого формула обертання точки *(x, y, z)* довкола осі *OY* на кут β матиме наступний вигляд:

$$\begin{cases} x' = x\cos\beta + z\sin\beta \\ y' = y \\ z' = x\sin(-\beta) + z\cos\beta \end{cases}$$

або в матричній формі

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}.$$

Аналогічно поступаємо з віссю обертання *OX*. Змінимо положення координатних осей так, щоб вісь *OX* була направлена в площину рисунка, вісь *OY* - горизонтально праворуч, вісь *OZ* - вертикально вгору (рис. 15).

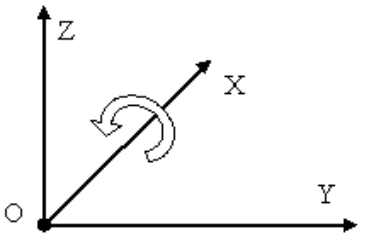

Рис. 15. Поворот навколо вісі X

Замінивши у формулі двовимірного повороту y на z, x на y, отримаємо формулу обертання точки (x, y, z) довкола осі OX на кут α:

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

або в матричній формі

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}.$$

Спосіб двовимірного плоского повороту довкола довільної точки може бути узагальнений на випадок повороту довкола довільної осі тривимірного простору. Хай довільна вісь повороту задається вектором P_1P_2, причому $P_1 = (a,b,c)$ – точка, що визначає початок вектора, а $P_2 = (m,l,n)$ – кінець вектора (рис. 16).

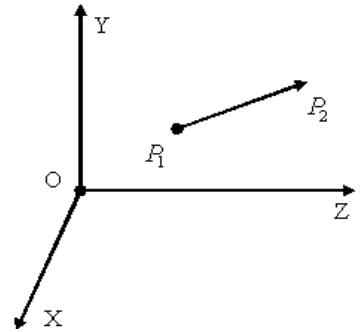

Рис. 16. Вектор осі повороту у просторі

Поворот навколо вказаної вісі (вектора P_1P_2) на кут θ виконується в декілька етапів:

1. Паралельний зсув вектора так, щоб початок вектора (точка P_1) співпав з початком системи координат. Це здійснюється за допомогою операції переносу $T(-a, -b, -c)$;

2. Поворот навколо вісі OY на кут β так, щоб вектор (m, l, n) опинився в площині OYZ: $R_y(\beta)$;

3. Поворот навколо вісі *OX* на кут α так, щоб вектор *(m', l', n')* співпав з віссю *OZ*: $R_x(\alpha)$;

4. Поворот навколо вісі *OZ* на вказаний кут θ: $R_z(\theta)$;

5. Виконання перетворення, зворотного тому, що було виконано на кроці 3. Тобто поворот навколо вісі *OX* на кут $-\alpha$;

6. Виконання перетворення, зворотного тому, що було виконано на кроці 2. Тобто поворот навколо вісі *OY* на кут $-\beta$;

7. Виконання перетворення, зворотного тому, що було виконано на кроці 1. Тобто паралельний зсув на вектор *(a,b,c):T(a,b,c)*

Наведений алгоритм обертання навколо довільної осі можна записати за допомогою добутку послідовності елементарних матриць:

$$V' = T(a,b,c) \cdot R_y(-\beta) \cdot R_x(-\alpha) \cdot R_z(\theta) \cdot R_x(\alpha) \cdot R_y(\beta) \cdot T(-a,-b,-c) \cdot V \ ,$$

де V – початкова точка, V' – точка після обертання.

Залишилось визначити, чому дорівнюють кути обертання α і β (рис. 17).

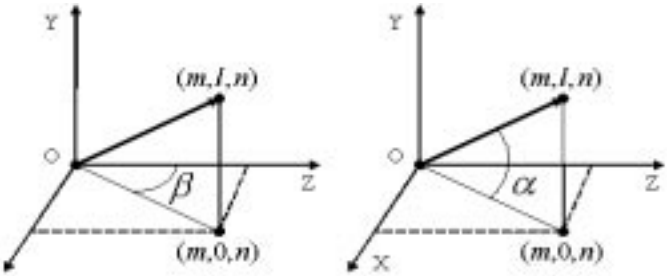

Рис. 17. Кути при повороті навколо вказаної осі

З простих тригонометричних співвідношень можна отримати наступні формули:

$$\cos\beta = \frac{n}{\sqrt{m^2 + n^2}}, \quad \cos\alpha = \frac{\sqrt{m^2 + n^2}}{\sqrt{m^2 + n^2 + l^2}}.$$

Як видно, операції тривимірного масштабування і повороту можуть бути реалізовані за допомогою множення вектора-стовпця (точки) на матрицю перетворення. Проте, операція паралельного зсуву реалізується через складання двох векторів-стовпців. Аналогічно тому, як всі двовимірні перетворення (паралельний зсув, масштабування

і поворот) описуються матрицями розміром 3x3 (через однорідні координати), тривимірні перетворення можуть бути представлені у вигляді матриць розміром 4x4. Тоді точка тривимірного простору *(x,y,z)* записується в однорідних координатах як *Wx,Wy,Wz,W*, де *W≠0*. Якщо *W≠1*, то для отримання тривимірних декартових координат точки *(x, y, z)* перші три однорідні координати потрібно розділити на *W*. Звідси витікає, що дві точки H_1 і H_2 в просторі однорідних координат описують одну і ту ж точку тривимірного простору в тому і лише тому випадку, коли $H_1 = c * H_2$ для будь-якої константи *c*, що не дорівнює нулю. Таким чином, перетворення точки тривимірного простору *P=(x,y,z)* в нову точку *P'=(x',y',z')* з використанням однорідних координат можна записати як:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Рівняння тривимірного паралельного зсуву, масштабування і повороту записуються у вигляді матриць перетворення однорідних координат таким чином:

1) Паралельний перенос:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = T(Tx, Ty, Tz) * P.$$

2) Масштабування:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = S(Sx, Sy, Sz) * P.$$

3) Повороти:

- Навколо осі X:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = R_x(\alpha) * P.$$

- Навколо осі Y:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = R_y(\beta) * P.$$

- Навколо осі Z:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = R_z(\varphi) * P,$$

де *Tx,Ty,Tz* – величини перенесення вздовж осей *OX, OY, OZ* відповідно, *Sx,Sy,Sz* – масштабні множники вздовж осей *OX, OY, OZ* відповідно, *Rx*(α), *Ry*(β), *Rz*(φ) – матриці обертання навколо осей *OX, OY, OZ* на кути α, β, φ відповідно.

Як і в двовимірному випадку, матричний підхід дозволяє поєднати два або більш елементарних перетворення в одне. Таким чином, послідовне застосування двох перетворень T_1 та T_2 може бути замінено застосуванням одного перетворення T, причому матриця T дорівнюватиме добутку матриць перетворень T_1 та T_2. Це легко побачити на простому прикладі. Нехай точка *(x,y,z)* трансформується в точку *(x',y',z')* за допомогою перетворення T_1:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = T_1 \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Застосувавши далі перетворення T_2 до точки *(x', y', z')*, отримаємо точку

$$\begin{pmatrix} x'' \\ y'' \\ z'' \\ 1 \end{pmatrix} = T_2 \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}.$$

Тепер, підставивши перший вираз у другий, отримаємо:

$$\begin{pmatrix} x'' \\ y'' \\ z'' \\ 1 \end{pmatrix} = T_2 T_1 \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Причому порядок застосування перетворень має бути збережений при обчисленні добутку відповідних матриць.

Лабораторна робота №6. Алгоритми видалення невидимих ліній

Завдання

1. Створити програму, що матиме зручний інтерфейс (вікно, головне меню, рядок статусу та ін.). У вікні програми повинно створюватися та відображатись зображення фігури, створеної відповідно завдань лабораторних робіт №№4,5.

2. За допомогою методів та алгоритмів видалення невидимих ліній та поверхонь створити більш реалістичне зображення тривимірної фігури Програма повинна надавати можливість виконання над фігурою трьох основних афінних перетворень, як у лабораторній роботі № 5, а також вмикати та вимикати режим видалення невидимих ліній.

Теоретичні відомості

Задача видалення невидимих ліній та поверхонь є однією з найбільш цікавих та складних у комп'ютерній графіці. Алгоритми ви-

далення полягають у визначенні ліній ребер, поверхонь або об'ємів, які є видимими або невидимими для спостерігача, що знаходиться в указаній точці простору.

Необхідність видалення невидимих ліній, ребер, поверхонь або об'ємів проілюстрована на рис.18. Рисунок наочно демонструє, що зображення без видалення невидимих ліній сприймається неоднозначно.

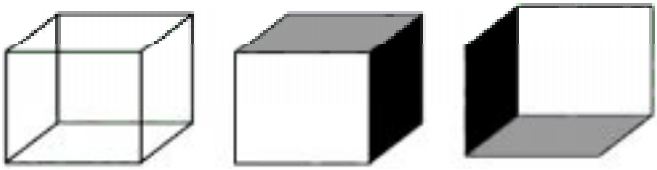

Рис. 18. Неоднозначність сприйняття зображення куба

Складність задачі видалення невидимих ліній та поверхонь призвела до появи великої кількості різноманітних способів її розв'язання. Багато з них орієнтовані на спеціалізовані додатки. Єдиного (загального) розв'язку цієї задачі, який був би придатним для усіх різноманітних випадків, природно, не існує. Для кожного випадку обирається метод, що найбільш підходить. Наприклад, для моделювання процесів у реальному часі потрібні швидкі алгоритми, в той час як для формування складного реалістичного зображення, в якому представлені тіні, прозорість і фактура, що враховують ефекти віддзеркалення і заломлення кольору у найменших відтінках, фактор часу виконання вже не такий важливий. Подібні алгоритми працюють повільно, і часто на обчислення потрібно декілька хвилин або навіть годин. Існує тісний взаємозв'язок між швидкістю роботи алгоритму і детальністю його результату. Жоден з алгоритмів не може досягнути гарних оцінок для цих двох показників одночасно. По мірі створення все більш швидких алгоритмів можна будувати все більш детальні зображення. Реальні завдання, однак, завжди будуть вимагати враховувати ще більшу кількість деталей.

Всі алгоритми такого роду так чи інакше включають в себе сортування, причому головне сортування ведеться по геометричній відстані від тіла, поверхні, ребра чи точки до точки спостерігання або картинної площини. Основна ідея, що покладена в основу сортування по відстані, полягає в тому, що чим далі розташований об'єкт від точки

спостерігання, тим більше ймовірність того, що він буде повністю або частково закритий одним з об'єктів, більш близьких до точки спостерігання. Після визначення відстаней або пріоритетів за глибиною залишається провести сортування по горизонталі та по вертикалі, щоб з'ясувати, чи буде об'єкт, що розглядається, дійсно закритий об'єктом, який розташований ближче до точки спостерігання. Ефективність будь-якого алгоритму видалення в значній мірі залежить від ефективності процесу сортування.

Докладно більшість цих алгоритмів описані у роботі [1]. Тут розглянемо лише основні. Більшість з них можна значно поліпшити, якщо виконати попередні обчислення щодо видалення нелицевих граней, тобто тих граней, які повністю перекриваються іншими.

Алгоритм видалення нелицевих граней спочатку видаляє з кожного багатогранника ті ребра або грані, які екрануються самим тілом. Для цього можна використати простий тест: якщо одна або обидві суміжні грані обернені своєю зовнішньою поверхнею до спостерігача, то ребро є видимим. Цей тест виконується обчисленням скалярного добутку координат вектора, що направлений від багатогранника до спостерігача, на вектор зовнішньої нормалі грані: якщо результат додатний, то грань видима.

Розглянемо цей тест більш докладно. Нехай грань утворена вершинами, що мають координати (X_0, Y_0, Z_0), (X_1, Y_1, Z_1), (X_2, Y_2, Z_2),…,($X_к$, $Y_к$, $Z_к$). Вектор зовнішньої нормалі $\overline{N}$, що має координати (X_n, Y_n, Z_n) можна обчислити, наприклад через векторний добуток векторів ребер, що утворюють грань.

$$\overline{N} = \overline{V}_1 \times \overline{V}_2, \text{ де } \overline{V}_1(X_1 - X_0, Y_1 - Y_0, Z_1 - Z_0),$$
$$\overline{V}_2(X_2 - X_0, Y_2 - Y_0, Z_2 - Z_0).$$

З курсу математики відомо, що координати вектора N можна обчислити за формулами:

$$N_x = (Y_1 - Y_0)(Z_2 - Z_0) - (Y_2 - Y_0)(Z_1 - Z_0),$$
$$N_y = (Z_1 - Z_0)(X_2 - X_0) - (Z_2 - Z_0)(X_1 - X_0),$$
$$N_Z = (X_1 - X_0)(Y_2 - Y_0) - (X_2 - X_0)(Y_1 - Y_0).$$

Таким чином, отримаємо вектор $\overline{N}$ *(N_x, N_y, N_z)*. Якщо спостерігач знаходиться в точці *C(0,0,c)* (як на рис. 12), то вектор, що направлений до спостерігача $\overline{V}(V_x, V_y, V_z)$ буде мати такі координати: V_x = *0-*X_0, V_y=*0-*Y_0, V_z=*c-*Z_0. Таким чином маємо $\overline{V}(-X_0, Y_0, c - Z_0)$. Тепер залишилось обчислити скалярний добуток векторів $\overline{V} \cdot \overline{N} = (N_x \cdot V_x + N_y \cdot V_y + N_z \cdot V_z)$. Як було зазначено вище, якщо цей добуток додатний, то грань є видимою, якщо добуток від'ємний, або дорівнює нулю – грань невидима.

Після визначення тих граней, які є повністю невидимими, оскільки обернені до спостерігача нелицевою стороною, малювати всі інші грані враховуючи той факт, що вони можуть бути частково екрановані іншими гранями. Для цього можна скористатися методом *Z*-буфера.

Це один з найпростіших алгоритмів видалення невидимих поверхонь. Вперше він був запропонований Кетмулом в 1975 р. (докладніше про цей алгоритм можна прочитати в [4]). Працює цей алгоритм в просторі зображення. Ідея *Z*-буфера є простим узагальненням ідеї про буфер кадру. Буфер кадру використовується для запам'ятовування атрибутів кожного пікселя в просторі зображення, а *Z*-буфер призначений для запам'ятовування глибини (відстані від картинної плоскості) кожного видимого пікселя в просторі зображення. Оскільки достатньо поширеним є використання координатної площини *XOY* в якості картинної площини, то глибина дорівнює координаті *Z* точки, звідси і назва буфера. В процесі роботи значення глибини кожного нового пікселя, котрий потрібно занести в буфер кадру, порівнюється з глибиною того пікселя, котрий вже занесений до *Z*-буферу. Якщо це порівняння показує, що новий піксель розташований попереду пікселя, що знаходиться в буфері кадру, то новий піксель заноситься в цей буфер і, крім того, виконується корегування *Z*-буфера новим значенням глибини. Якщо ж порівняння дає протилежний результат, то жодних дій не виконується. Насправді, алгоритм є пошуком по **x** та **y** найбільшого значення функції *z(x,y)*.

Головна перевага алгоритму – його простота. Крім того, цей алгоритм вирішує задачу про видалення невидимих поверхонь і робить тривіальною візуалізацію складних поверхонь. Сцени можуть бути будь-якої складності. Оскільки габарити простору зображення фіксо-

вані, оцінка обчислювальної трудомісткості алгоритму не більше ніж лінійна. Оскільки елементи сцени або картинки можна заносити в буфер кадру або в *Z*-буфер в довільному порядку, їх не потрібно заздалегідь сортувати за пріоритетом глибини. Тому економиться обчислювальний час, що витрачається на сортування за глибиною.

Основний недолік алгоритму – великий обсяг необхідної пам'яті. Останнім часом у зв'язку з швидким зростанням можливостей обчислювальної техніки цей недолік стає менш лімітуючим. Але в той час, коли алгоритм ще лише з'явився, доводилося винаходити способи створення буфера як можна більшого об'єму при наявному ресурсі пам'яті.

Наприклад, можна розбивати простір зображення на 4, 16 або більше прямокутників або смуг. У граничному варіанті можна використовувати буфер розміром в один рядок розгортки. Для останнього випадку був розроблений алгоритм порядкового сканування. Оскільки кожен елемент сцени обробляється багато раз, та сегментація *Z*-буфера, взагалі кажучи, призводить до збільшення часу, необхідного для обробки сцени. Інший недолік алгоритму полягає в трудомісткості реалізації ефектів, пов'язаних з напівпрозорістю, і рядом інших спеціальних завдань, що підвищують реалістичність зображення. Оскільки алгоритм заносить пікселі в буфер кадру в довільному порядку, то досить складно отримати інформацію, яка необхідна для методів, що ґрунтуються на попередньому аналізі сцени.

В цілому алгоритм виглядає так:

1. Заповнити буфер кадру фоновим значенням кольору.
2. Заповнити *Z*-буфер мінімальним значенням *z* (глибини) .
3. Перетворити зображувані об'єкти в растрову форму в довільному порядку.
4. Для кожного об'єкта:

4.1. Для кожного пікселя *(x,y)* образу обчислити його глибину *z(x,y)*.

4.2. Порівняти глибину *z(x,y)* зі значенням глибини, що зберігається в *Z*-буфері в тій самій позиції.

4.3. Якщо *z(x,y)>Z-буфер(x,y)*, то занести атрибути пікселя в буфер кадру і замінити *Z-буфер(x,y)* на *z(x,y)*. Інакше жодних дій не виконувати.

Існують і інші алгоритми видалення невидимих ліній та поверхонь, докладніше про них можна прочитати у літературі, наприклад [1,2].

СПИСОК ЛІТЕРАТУРИ

1. *Блінова Т.О., Порєв В.М.* Комп'ютерна графіка. – К.:"Юніор", 2004. – 456 с., іл.

2. *Порев В.Н.* Компьютерная графика. – СПб.: БХВ-Петербург, 2002. – 432 с., ил.

3. *Тюкачев Н., Свиридов Ю.* Delphi 5. Создание мультимедийных приложений. Учебный курс. – СПб.:Питер, 2001. – 400 с., ил.

4. *Хилл Ф.* OpenGL. Программирование компьютерной графики. Для профессионалов. – СПб.: Питер, 2002. – 1088 с., ил.

ЗМІСТ

Навчальне видання

БЕРКУНСЬКИЙ Євген Юрійович

Комп'ютерна графіка та діалогові системи

Методичні вказівки для студентів
напрямку "Комп'ютерні науки"
(*українською мовою*)

Комп’ютерна верстка *В.Г. Мазанко*

Коректор *М.О. Паненко*

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців,
виготівників і розповсюджувачів видавничої продукції
ДК № 2506 від 25.05.2006 р.

Підписано до друку 16.04.09. Папір офсетний. Формат 60×84/16.
Друк офсетний. Гарнітура "Таймс". Ум. друк. арк. 2,3. Обл.-вид. арк. 2,5.
Тираж 100 прим. Вид. 18. Зам. № 164. Ціна договірна

Видавець і виготівник Національний університет кораблебудування,
54002, м. Миколаїв, вул. Скороходова, 5

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК