МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ»
ФИЗИКО-ТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ
КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

ИНЖЕНЕРНАЯ И КОМПЬЮТЕРНАЯ ГРАФИКА

КУРС ЛЕКЦИЙ

(ЧАСТЬ 2)

Донецк
ГОУ ВПО «ДонНУ»
2017

УДК 004.92 (042.4)
ББК Ж11я73-2 + З973.2-018.3я73-2
К 731

Авторы:
В. Н. Котенко, старший преподаватель кафедры;
Ю. В. Котенко, ассистент кафедры

Ответственный за выпуск:
Т. В. Шарий, канд. техн. наук, доц.

Утверждено на заседании ученого совета
физико-технического факультета ГОУ ВПО «ДонНУ»
Протокол № 5 от 24.01.2017

К 731 Котенко В. Н.

Инженерная и компьютерная графика: курс лекций (часть 2) для студентов укрупненной группы направлений подготовки 09.00.00 «Информатика и вычислительная техника» направления подготовки 09.03.01 «Информатика и вычислительная техника» квалификационного уровня «Бакалавр» / В. Н. Котенко. – Донецк: ГОУ ВПО «ДонНУ», 2017. – 125 с.

Курс лекций содержит подробное описание каждой темы и примеры, разъясняющие теоретический материал.

Предназначен для студентов всех направлений подготовки, изучающих инженерную и компьютерную графику.

УДК 004.92 (042.4)
ББК Ж11я73-2 + З973.2-018.3я73-2

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данный курс лекций предназначен для формирования знаний студента о фундаментальных методах в графике, об общих принципах хранения, отображения и преобразования графической информации, о методах, средствах и технологиях графического и геометрического моделирования и построения интерактивных графических систем.

В курсе лекций использованы таблицы, схемы, графики, а также примеры программных кодов, которые разъясняют и закрепляют теоретический материал.

В издании излагаются основные понятия и принципы графического и геометрического моделирования. Рассматриваются такие темы, как перенос и поворот в двумерном пространстве, отсечение линий, автоматический подбор размеров и позиций, сглаживание кривых, геометрический инструмент трёхмерной графики, перенос и поворот в трёхмерном пространстве, перспективные изображения, вычерчивание проволочных моделей.

Курс лекций построен следующим образом. В начале каждой темы приводятся ключевые определения, осуществляется вывод формул графических преобразований, а затем детально рассматривается программный код, реализующий механизм этих графических преобразований.

В конце курса лекций приводится список рекомендуемой литературы для изучения дисциплины.

Тема 17. Перенос и поворот в двумерном пространстве

17.1 Преобразование и новые координаты точки

Рассмотрим следующую систему уравнений:

$$\begin{cases} x' = x + a \\ y' = y \end{cases}$$

Эти уравнения можно интерпретировать двояким образом:

1) все точки на плоскости XY перемещаются вправо на расстояние *a*;

2) координатные оси X и Y перемещаются влево на расстояние *a*.

На рис. 17.1 показаны перенос и изменение координат.

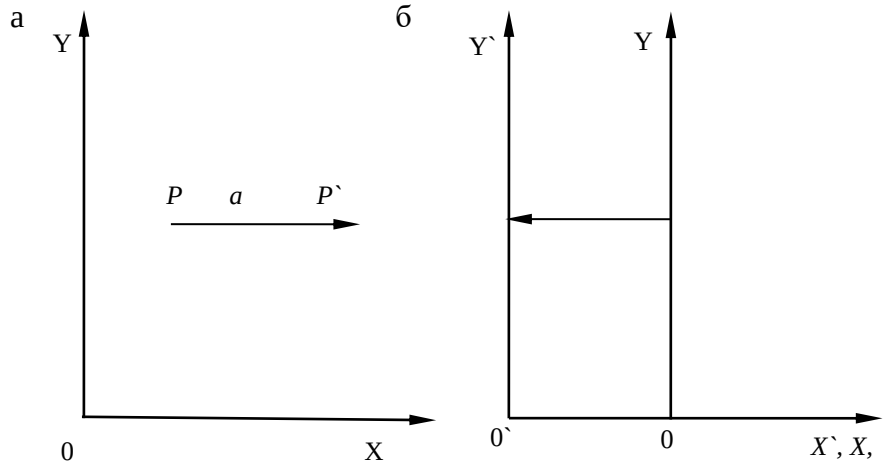


Рис. 17.1 – Перенос и изменение координат:
а – перенос; б – изменение координат

Этот пример иллюстрирует принцип, применимый и в более сложных ситуациях. Далее будем рассматривать системы уравнений, записываемых в виде произведений матриц, интерпретируя их как преобразования всех точек в фиксированной системе координат. Однако та же самая система уравнений может интерпретироваться и как изменение системы координат.

Пусть необходимо повернуть точку *P(x, y)* вокруг начала координат O на угол φ. Изображение новой точки обозначим через *P′ (x′,y′)*. На рис. 17.2 показан поворот точки вокруг точки начала координат на угол φ.

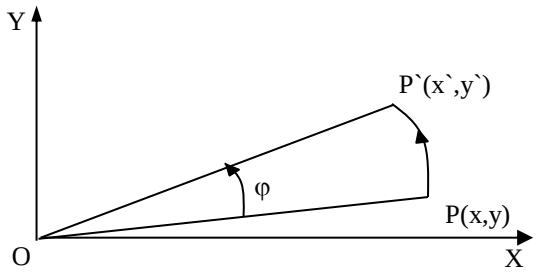


Рис. 17.2 – Поворот вокруг точки O на угол φ

Существуют четыре числа *a, b, c, d* такие, что новые координаты *x′* и y` могут быть вычислены по значениям старых координат *x* и *y* из следующих уравнений:

$$\begin{cases} x' = a * x + b * y \\ y' = c * x + d * y \end{cases} \qquad (17.1)$$

Для получения значений a, *b, c, d* рассмотрим вначале точку *(x, y) = (1,0)*. Полагая *x* = 1 и *y = 0* в уравнении (17.1), получим

$$x' = a$$
$$y' = c$$

Но в этом простом случае, как видно из рис. 17.3 (а), значения *x′* и *y′* равны соответственно cos(φ) и sin(φ).

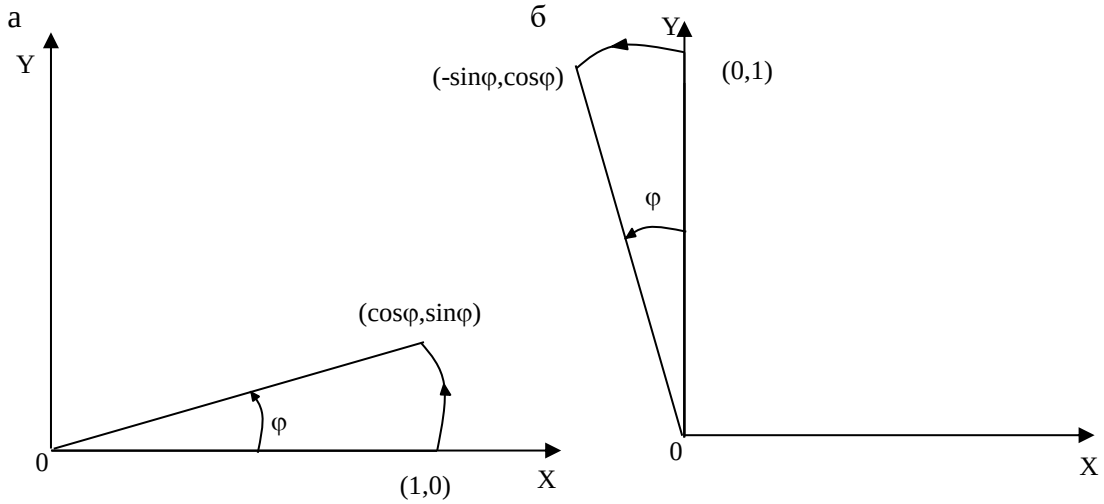


Рис. 17.3 – Отображение точек при повороте:
а – отображение точки (1,0); б – отображение точки (0,1)

Тогда будем иметь:

$$a = \cos(\varphi)$$
$$c = \sin(\varphi)$$

Аналогичным образом из рис. 17.3 (б) следует:

$$b = -\sin(\varphi)$$
$$d = \cos(\varphi)$$

Тогда вместо системы уравнений (1.1) можем записать:

$$\begin{cases} x' = x\cos(\varphi) - y\sin(\varphi) \\ y' = x\sin(\varphi) + y\cos(\varphi) \end{cases} \qquad (17.2)$$

В приведенной ниже программе изображение стрелки вычерчивается после предварительного поворота вокруг точки O на 6°.

```
/* Arrow14: Программа чертит 14 стрелок, летящих в направлении
против часовой стрелки относительно точки O первого квадранта
координатной системы */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Arrow14
{
  public partial class Form1 : Form
  {
    Graphics dc; Pen p;
    public Form1()
    {
      InitializeComponent();
      dc = pictureBox1.CreateGraphics();
      p = new Pen(Brushes.Black, 1);
    }
/* Функции преобразования вещественных координат X и Y в целые
для того, чтобы превратить область вывода в прямоугольник
размером 10x7 условных единиц с началом координат в точке (0.0,
0.0), находящемся в левом нижнем углу области вывода. Оси
координат направлены теперь как в математической системе
координат. Координаты точек теперь вещественные: по оси X от 0.0
до 10.0, по оси Y от 0.0 до 7.0. */
```

```
    private int IX(double x)
    {
      double xx = x * (pictureBox1.Size.Width / 10.0) + 0.5;
      return (int)xx;
    }
    private int IY(double y)
    {
      double yy = pictureBox1.Size.Height - y *
          (pictureBox1.Size.Height / 7.0) + 0.5;
      return (int)yy;
    }
  /* Функция вычерчивания линии (область вывода 10x7 усл.ед.*/
    private void Draw(double x1,double y1,double x2,double y2)
    {
      Point point1 = new Point(IX(x1), IY(y1));
      Point point2 = new Point(IX(x2), IY(y2));
      dc.DrawLine(p, point1, point2);
    }
  /* Функция рисования 14 повернутых стрелок */
    private void DrawArrow14()
    {
      double[] x = new double[4] {6.0, 6.0, 5.9, 6.1};
      double[] y = new double[4] {-0.25, 0.25, 0.0, 0.0};
      int i, j;   double phi, sin_phi, cos_phi, xold, yold;
      phi = 6 * Math.PI / 180;
      sin_phi = Math.Sin(phi); cos_phi = Math.Cos(phi);
      for (i = 1; i <= 14; i++)
      {
        /* Находим координаты очередной стрелки, повернутой
        относительно предыдущей на угол phi */
        for (j = 0; j <= 3; j++)
        {
           xold = x[j]; yold = y[j];
           x[j] = xold * cos_phi - yold * sin_phi;
           y[j] = xold * sin_phi + yold * cos_phi;
        }
        /* Рисуем повернутую стрелку */
        xold = x[0]; yold = y[0];
        for (j = 1; j <= 3; j++)
        {
          Draw(xold, yold, x[j], y[j]);
          xold = x[j]; yold = y[j];
        }
        Draw(xold, yold, x[1], y[1]);
      }
    }
  /* Вызов функции рисования стрелок при нажатии на кнопку */
    private void button1_Click(object sender, EventArgs e)
    {
      DrawArrow14();
    }
  }
}
```

В геометрии точки объекта обычно обозначаются прописными буквами A, B... . Здесь будем обозначать их цифрами 0, 1... . В начальной позиции стрелка указывает вверх, ее центр расположен в точке (6.0, 0.0). Значения координат *x* и y для вершин ломаной линии, изображающей стрелку, записаны в элементах массивов `x[i] и y[i] (i = 0, 1, 2, 3)`. Нумерация индексов элементов массива начинается с нуля. Таким образом, описание массива `double[] x = new double[4] {6.0, 6.0, 5.9, 6.1};` определяет четыре элемента массива с начальными значениями `x[0] = 6.0, x[1] = 6.0, x[2] = 5.9, x[3] = 6.1`.

То обстоятельство, что начальное значение `y[0]` задано отрицательным, а все координаты для вычерчивания должны быть положительными, не вызывает сложностей, поскольку перед вычерчиванием стрелка подвергается повороту, который переносит ее в область над осью X. На рис. 17.4 показан результат работы этой программы.

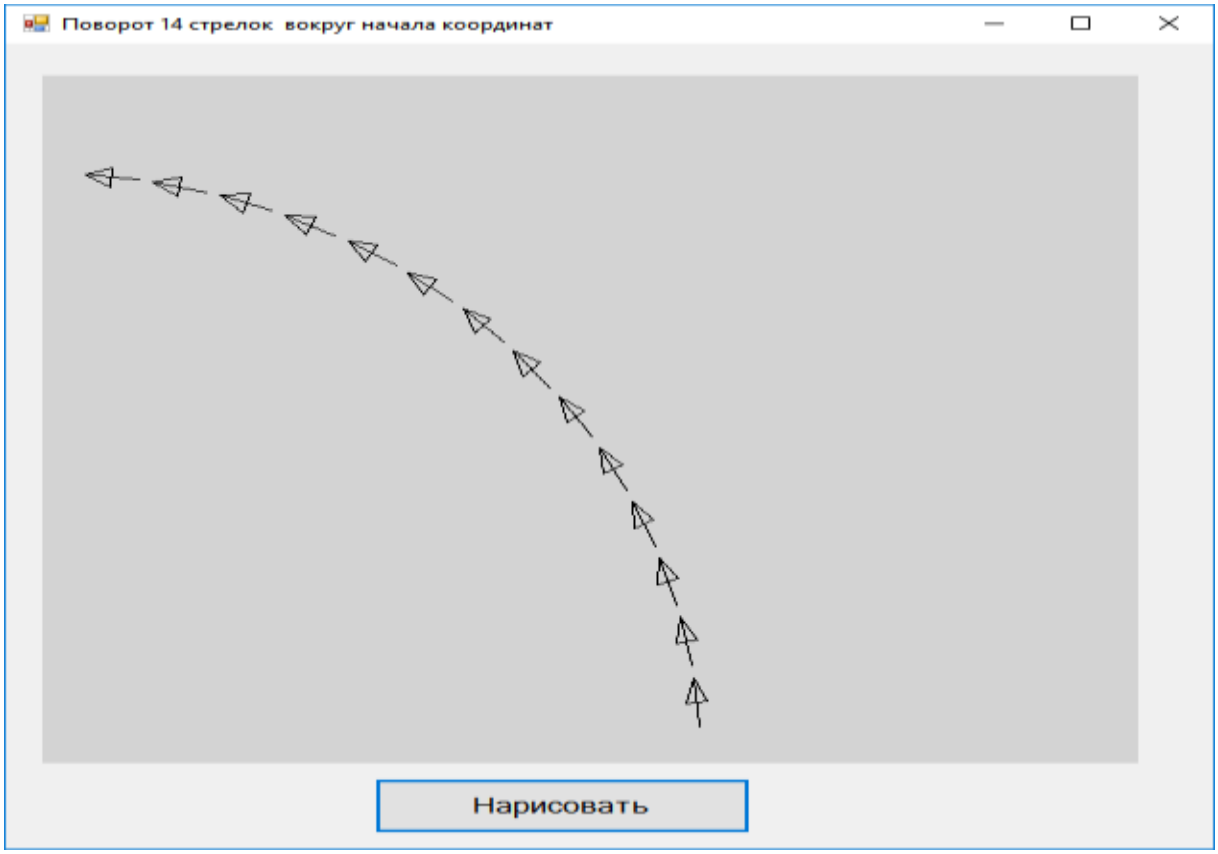


Рис 17.4 – Результат работы программы Arrow14

17.2 Поворот вокруг заданной точки

Система уравнений (17.2) описывает поворот вокруг точки – начала системы координат. Но часто требуется выполнить поворот относительно заданной точки (x_0, y_0). Тогда в этих уравнениях можно заменить: x на $(x - x_0)$, y – на $(y - y_0)$, x' – на $(x' - x_0)$ и y' – на $(y' - y_0)$.

$$\begin{cases} x' - x_0 = (x - x_0)\cos(\varphi) - (y - y_0)\sin(\varphi) \\ y' - y_0 = (x - x_0)\sin(\varphi) + (y - y_0)\cos(\varphi) \end{cases}$$
$$\begin{cases} x' = x_0 + (x - x_0)\cos(\varphi) - (y - y_0)\sin(\varphi) \\ y' = y_0 + (x - x_0)\sin(\varphi) + (y - y_0)\cos(\varphi) \end{cases} \qquad (17.3)$$

Заменим функцию `DrawArrow14()` на функцию `DrawArrow30()`.

```
/* Функция рисования 30 повернутых стрелок */
private void DrawArrow30()
{
  double[] x = new double[4] { 0.0, 0.0, -0.08, 0.08 };
  double[] y = new double[4] { -0.25, 0.25, 0.0, 0.0 };
  double phi, sin_phi, cos_phi, dx, dy; int i, j;
  phi = 12 * Math.PI / 180;
  sin_phi = Math.Sin(phi); cos_phi = Math.Cos(phi);
  double x0 = 5.0, y0 = 3.5, r = 3, xold = 0.0, yold = 0.0;
  /* Перенос в начальную позицию (x0 + r, y0) */
  for (j = 0; j < 4; j++) { x[j] += x0 + r; y[j] += y0; }
  /* Цикл прописовки стрелок */
  for (i = 0; i < 30; i++)
  {
       /* Выполняем поворот вокруг точки (x0,y0) на phi
         градусов относительно предыдущей стрелки */
       for (j = 0; j <= 3; j++)
       {
         dx = x[j] - x0; dy = y[j] - y0;
         x[j] = x0 + dx * cos_phi - dy * sin_phi;
         y[j] = y0 + dx * sin_phi + dy * cos_phi;
       }
       /* Рисуем повернутую стрелку */
       xold = x[0]; yold = y[0];
       for (j = 1; j <= 3; j++)
       {
         Draw(xold, yold, x[j], y[j]);
         xold = x[j]; yold = y[j];
       }
       Draw(xold, yold, x[1], y[1]);
  }
}
```

Результат работы программы показан на рис. 17.5.

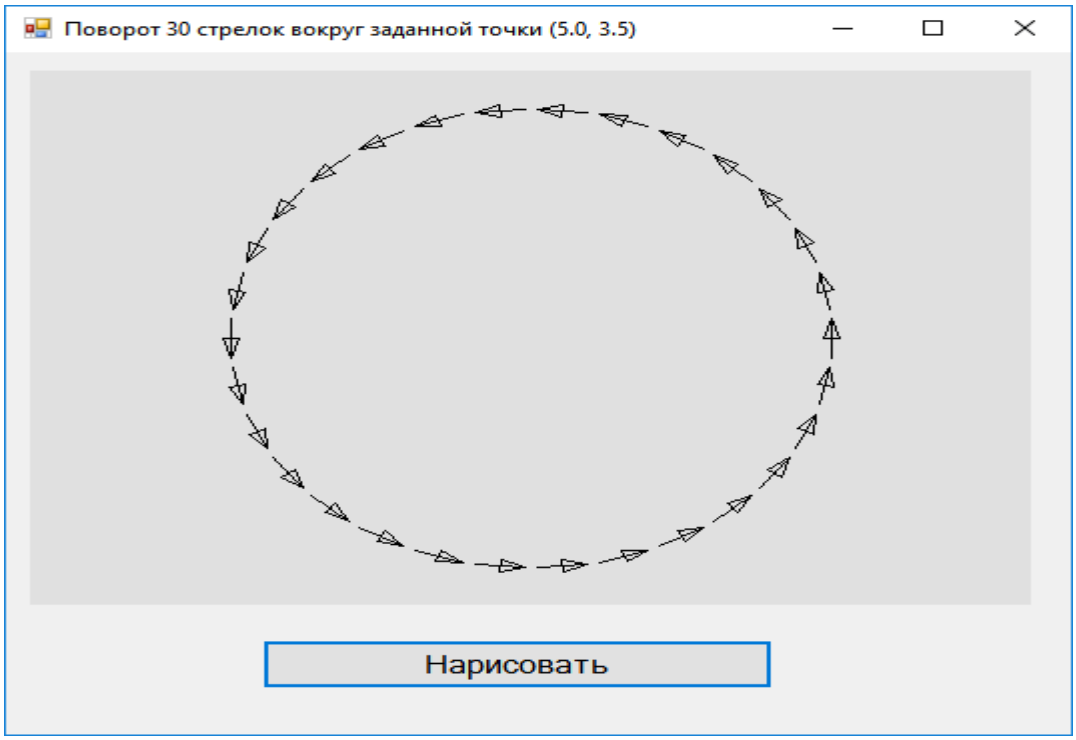


Рис. 17.5 – Результат работы Arrow30

17.3 Матричная запись поворота и переноса

Система уравнений (17.2) может быть записана в виде одного матричного уравнения:

$$[x', y'] = [x, y]\begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \qquad (17.4)$$

или с использованием вектора-столбца:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad (17.5)$$

В книгах по машинной графике запись с вектором-строкой (17.4) встречается чаще, чем с вектором-столбцом (17.5). Далее будем использовать запись типа (17.4). В такой записи *i-я* строка квадратной матрицы всегда является отображением *i-го* единичного вектора (i = 1, 2).

Можно записать в матричной форме систему уравнений (17.3):

$$[x', y'] = [x_0, y_0] + [x - x_0, y - y_0] \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \quad (17.6)$$

Однако первая часть этого уравнения не является чисто матричным произведением. В более сложных ситуациях, когда поворот совмещается с другими преобразованиями, было бы более удобно иметь единое матричное произведение для каждого элементарного преобразования.

На первый взгляд это кажется невозможным, если преобразование включает операцию переноса. Но, как увидим ниже, с помощью матрицы преобразования размера 3x3 это вполне реально. Начнем с простого переноса. Пусть точка *P(x, y)* переносится в точку *P'(x', y')*, где

$$\begin{cases} x' = x + a \\ y' = y + b \end{cases} \quad (17.7)$$

Эти уравнения можно переписать в виде:

$$[x', y'] = [x, y, 1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ a & b \end{bmatrix}$$

Но с учетом будущих потребностей это уравнение лучше переписать в следующей форме:

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix} \quad (17.8)$$

Легко проверить, что уравнения (17.7) и (17.8) эквивалентны. Такую запись принято называть записью в системе «однородных координат». Однородные координаты более подробно будут обсуждаться далее.

Запись каждого преобразования в форме произведения матриц позволяет совмещать несколько преобразований в одном. Чтобы показать такое совмещение преобразований, объединим поворот с двумя переносами. Поворот на угол φ вокруг начала координат О был описан уравнением (17.4). Заменим это уравнение следующим:

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (17.9)$$

Теперь выведем новую версию уравнений (17.6) для описания поворота на угол φ вокруг точки (x_0, y_0). Это уравнение может быть выражено формулой:

$$[x', y', 1] = [x, y, 1] R, \qquad (17.10)$$

где через R обозначена матрица размером 3х3.

Для нахождения этой матрицы R будем считать, что преобразование состоит из трех шагов с промежуточными точками (u_1, v_1) и (u_2, v_2).

1) преобразование для переноса точки (x_0, y_0) в начало координат О:

$$[u_1, v_1, 1] = [x, y, 1] T',$$

где

$$T' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_0 & -y_0 & 1 \end{bmatrix};$$

2) поворот на угол φ относительно точки начала координат О:

$$[u_2, v_2, 1] = [u_1, v_1, 1] R_0,$$

где

$$R_0 = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix};$$

3) перенос из начала координат в точку (x_0, y_0):

$$[x', y', 1] = [u_2, v_2, 1]\, T\ ,$$

где

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_0 & y_0 & 1 \end{bmatrix}.$$

Возможность комбинации этих шагов основана на свойстве ассоциативности матричного умножения, то есть

$$(AB)C=A(BC)$$

для любых трех матриц *A, B* и *C,* имеющих размерности, допускающие такое умножение. Для любой части этого уравнения мы можем просто записать *ABC.*

Теперь найдем:

$$[x', y', 1] = [u_2, v_2, 1]\ T = \{[u_1, v_1, 1]\, R_0\}\, T = [u_1, v_1, 1]\, R_0\, T = \{[x, y, 1]\, T'\}\, R_0\, T =$$
$$= [x, y, 1]\, T' R_0\, T = [x, y, 1]\, R$$

где $R = T'\, R_0\, T$.

Это и будет искомая матрица, которая после выполнения двух матричных умножений дает:

$$R = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ c_1 & c_2 & 1 \end{bmatrix},$$

где $c_1 = x_0 - x_0 \cos(\varphi) + y_0 \sin(\varphi)$,

$c_2 = y_0 - x_0 \sin(\varphi) - y_0 \cos(\varphi)$.

Тема 18. Отсечение линий

18.1 Окна и области вывода. Мировые и экранные координаты

Часто встречаются ситуации, когда требуется вычертить объекты, размеры которых заданы в единицах совершенно несовместимых с экранной системой координат. Например, размеры здания могут быть в сотни раз больше размеров желаемого изображения. С другой стороны, молекула в реальности значительно меньше ее изображения на картинке.

Наконец, имеются такие приложения, в которых объект является не какой-либо физической реальностью, а лишь графическим представлением соотношений между некоторыми значениями, например, на рис. 18.1 показана динамика доходов некоей фирмы в начале двадцать первого века.

Проблемно-ориентированные размеры выражаются в так называемых *мировых координатах.* На рис. 18.1 числа 2001, 2002, 2003, 2004 и 50 000, 100 000, 150 000, 200 000, 250 000 выражают значения в мировых координатах.

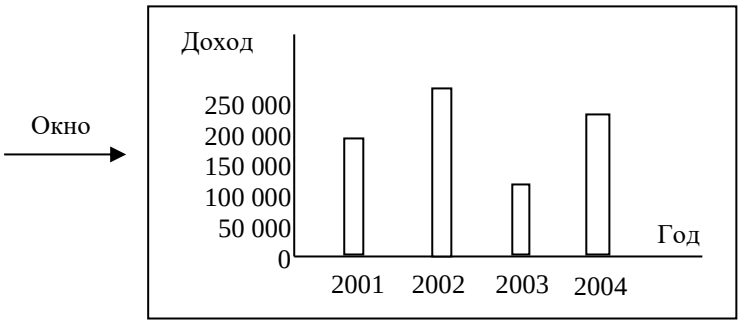


Рис. 18.1 – Столбчатая диаграмма в окне

Введем теперь концепцию *окна. Окно – это прямоугольник, в пределах которого вычерчивается объект (или его часть)*, как показано на рис. 18.1. Стороны прямоугольника параллельны координатным осям. Во избежание затруднений в понимании очень важно отметить, что окно относится к объекту, но не к изображению, которое будет сформировано. Если, как

обычно, введем горизонтальную ось x и вертикальную ось y, то окно на рис. 18.1 полностью определится значениями:

$$x_{\min} = 1998$$
$$x_{\max} = 2008$$
$$y_{\min} = -150\,000$$
$$y_{\max} = 325\,000$$

Очевидно, что размеры и положение окна определяются в системе мировых координат. Эти значения могут показаться неожиданными, поскольку окно вводится для определения желаемого изображения на картинке и, на первый взгляд, более приемлемым было бы задать эти значения в сантиметрах, чем указать какое-то фиктивное значение дохода, равное – 150 000, в качестве минимального значения $y_{\min}$ по оси y. Однако задание окна в системе мировых координат является обычным и удобным на практике.

Необходимо также задать прямоугольную область на экране, которая определит размеры желаемой картинки. Эта область называется *областью вывода.* Она задается аналогично окну, то есть указываются минимальные и максимальные значения по координатным осям x и y в единицах измерения на экране. Эти значения будут обозначаться прописными буквами X и Y. Типичным примером задания области вывода могут быть значения:

$$X_{\min} = 1.5$$
$$X_{\max} = 7.5$$
$$Y_{\min} = 1.0$$
$$Y_{\max} = 6.0$$

Теперь окно нужно отобразить на область вывода. Например, заданное значение в мировых координатах $x = 1998$ должно быть преобразовано в экранную координату $X = 1.5$.

Вначале вычисляются коэффициенты масштабирования по осям:

$$f_x = \frac{X_{\max} - X_{\min}}{x_{\max} - x_{\min}}$$

$$f_y = \frac{Y_{\max} - Y_{\min}}{y_{\max} - y_{\min}}$$

В нашем случае найдем, что:

$$f_x = \frac{7.5 - 1.5}{2008 - 1998} = 0.6,$$

$$f_y = \frac{6.0 - 1.0}{325\,000 - (-150\,000)} = 0.0000105.$$

Расстояние $X - X_{\min}$ точки изображения от левого края области вывода вычисляется умножением коэффициента f_x на соответствующее расстояние $x - x_{\min}$ от исходной точки до левого края окна. Расстояние $Y - Y_{\min}$ находится аналогично. Следовательно, координаты точки изображения будут определены из соотношений:

$$\begin{aligned} X &= X_{\min} + f_x * (x - x_{\min}) \\ Y &= Y_{\min} + f_y * (y - y_{\min}) \end{aligned} \qquad (18.1)$$

Замечания:

1. Окно не обязательно должно охватывать весь объект целиком. Если оно не охватывает весь объект, то части объекта, находящиеся вне окна, не вычерчиваются, – они должны быть отсечены. Эта операция носит название *отсечение*. Более детально она обсуждается далее.

2. В общем случае коэффициенты f_x и f_y различны. Для столбчатой диаграммы – это то, что нужно. Но совсем не годится, когда угловые соотношения на изображении должны быть точно такими же, как на объекте. В этом случае в качестве коэффициента масштабирования следует выбрать наименьшее из значений f_x и f_y. Поэтому рекомендуется заменить выражение (18.1) на формулы, основанные на пересчете координат относительно центров окна и области вывода.

3. Размеры и положение окна не всегда известны заранее. Далее будет показан способ их вычисления вместо задания пользователем.

18.2 Отсечение линий. Алгоритм Коэна-Сазерленда

Предположим, что мировые и экранные координаты одинаковы, то есть окно и область вывода совпадают, поэтому термин «окно» везде может быть заменен на термин «область вывода». Однако обычно считается, что отсечение выполняется по границам окна, а не области вывода, это и учитывается в данном случае.

На рис. 18.2 показан прямоугольник ABCD, который является окном. Все видимые отрезки прямых линий должны лежать внутри окна, то есть при вычерчивании отрезка прямой линии его части, лежащие вне окна, должны быть отсечены. Процесс отсечения должен выполняться автоматически. Команды на вычерчивание треугольника PQR на рис. 18.2 интерпретируются как команды на вычерчивание отрезков прямых линий P'P, PQ, QQ'. Прямоугольник ABCD вычерчивается раньше, так что вместо треугольника PQR нужно вычертить ломаную линию P'PQQ'.

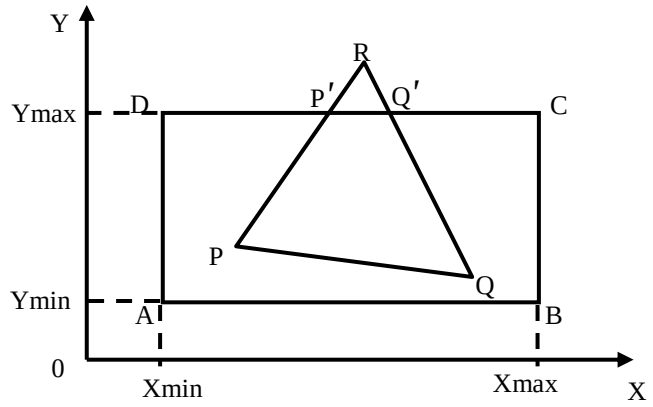


Рис. 18.2 – Треугольник, подвергающийся операции отсечения

Поскольку заданы только три точки P, Q R, координатная пара чисел $(x_{P'}, y_{P'})$ должна быть вычислена из значения координат точек (x_P, y_P) и (x_R, y_R). Из рис. 18.2 видно, что наклон отрезка PR можно вычислить двумя способами, что приводит к следующему уравнению:

$$\frac{y_{P'} - y_P}{x_{P'} - x_P} = \frac{y_R - y_p}{x_R - x_P}$$

Совмещая это уравнение с соотношением:

$$y_{P'} = y_{\max}$$

получим:

$$x_{P'} = x_P + \frac{(x_R - x_P)(y_{\max} - y_P)}{y_R - y_p}$$

Отсюда легко вычисляются координаты точки P', если известно, что концевая точка P находится внутри окна, а другая концевая точка $R(x_R, y_R)$ удовлетворяет неравенствам:

$$\frac{x_{\min} < x_R < x_{\max}}{y_R > y_{\max}}$$

Однако необходимо рассмотреть значительно больше ситуаций. Большое разнообразие логических операций, которые нужно выполнить для решения этой задачи, делают проблему отсечения линий очень интересной с алгоритмической точки зрения. Из рис. 18.3 очевидно, что недостаточно отсечь отрезок прямой линии PQ относительно прямой линии CD.

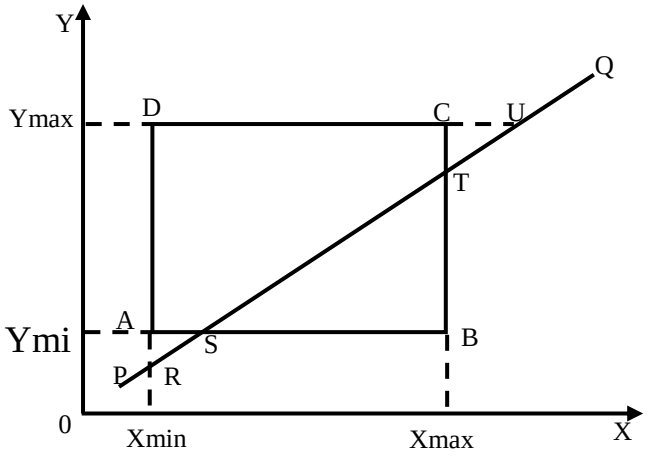


Рис. 18.3 – Последовательные шаги отсечения

Алгоритм для отсечения отрезков прямых линий разработали Коэн и Сазерленд. С любой точкой P(x, y) будем ассоциировать четырёхбитовый код: $b_3b_2b_1b_0$, где b_i может быть либо 0, либо 1 $(i = 0, 1, 2, 3)$.

Этот код содержит полезную информацию о положении точки P относительно окна ABCD. В языке C# условные выражения вырабатывают

значения. Так значение выражения $x < x_{\min}$ «истина», если оно верно, и «ложь», если условие не выполняется. Используя это правило, можно записать:

$b_3 = (x < x_{\min})$ – точка P слева от AD;
$b_2 = (x > x_{\max})$ – точка P справа от BC;
$b_1 = (y < y_{\min})$ – точка P ниже AB;
$b_0 = (y > y_{\max})$ – точка P над CD.

Фактически могут существовать только девять из 16 возможных битовых комбинаций. Они показаны на рис. 18.4.

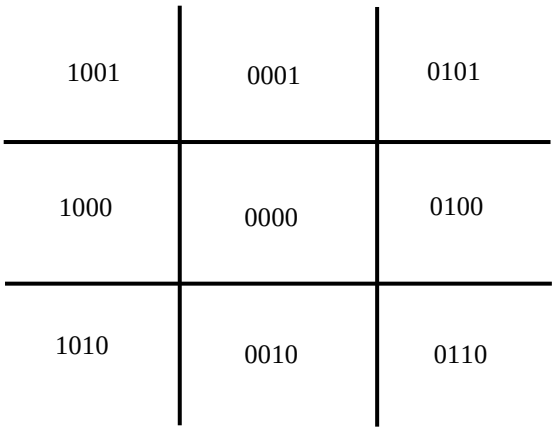


Рис. 18.4 – Значения кодов

Значения кодов вырабатываются функцией:

```
private byte code(double x, double y)
{
   return (byte)((Convert.ToByte (x < xmin) << 3) |
                 (Convert.ToByte (x > xmax) << 2) |
                 (Convert.ToByte (y < ymin) << 1) |
                  Convert.ToByte (y > ymax));
}
```

Для понимания этого выражения необходимо знать, что выражение $b << n$ означает, что битовое значение b сдвигается на n позиций влево. Кроме битовой операции сдвига << есть еще битовый оператор побитовой операции *ИЛИ*, записываемый как знак вертикальной черты (|). Менее эффективной записью вышеприведенного выражения в операторе возврата было бы выражение:

$$(x < x_{\min}) * 8 + (x > x_{\max}) * 4 + (y < y_{\min}) * 2 + (y > y_{\max})$$

Это выражение приведено исключительно с целью облегчения восприятия предыдущего выражения.

Описанная функция *code* будет использоваться в функции *clip,* в задачу которой входит анализ заданного отрезка прямой линии и вычерчивание только той его части, которая заключена внутри окна ABCD, если такая часть существует. Эта функция работает следующим образом.

Если хотя бы один из кодов для точек P_1 и P_2 содержит единичный бит, то либо P_1, либо P_2 перемещается из области вне окна к одной из границ окна или к ее продолжению. В последнем случае точка по-прежнему будет находиться вне окна и понадобится еще одно перемещение. Например, в случае, изображенном на рис. 18.3, за перемещением из точки *P* в точку *R* должно последовать перемещение из точки *R* в точку *S*. Затем может потребоваться отсечение другого конца отрезка, как это видно на рис. 18.3.

Таким образом, процесс отсечения может быть многоступенчатым, – на каждом шаге расстояние между точками P_1 и P_2 уменьшается. Процесс завершается, как только обе точки окажутся в пределах окна. Оставшаяся часть отрезка P_1P_2 будет вычерчена.

Однако существует еще один важный случай, когда цикл должен быть завершен, а именно, когда обе точки P_1 и P_2, находятся вне окна и в то же время по одну сторону от окна. Эту ситуацию нельзя различить вначале, но она может возникнуть в процессе отсечения. Если концевые точки отрезка прямой линии находятся вне окна, то отрезок может пересекать окно, но может оказаться и полностью вне окна, как показано на рис. 18.3 и 18.5.

На рис. 18.5 точки P_1 и P_2 сначала не находятся одновременно ниже окна. Но в процессе отсечения точки *Q* и *S* определяют новые позиции точек P_1 и P_2 соответственно. Поскольку теперь обе точки находятся ниже окна, то можно сделать вывод, что ничего не надо вычерчивать. Такое решение принимается на основе анализа значений *code(x1, y1*) и *code(x2, y2).* Точки P_1 и P_2 на находятся по одну сторону от окна тогда и только тогда, если их коды содержат единицу в одной и той же позиции.

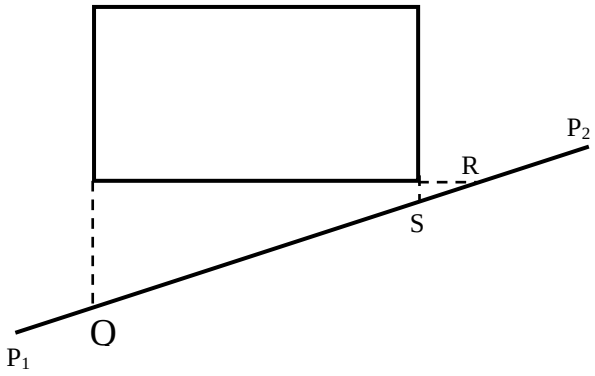


Рис. 18.5 – Отрезок вне окна

Для трех точек P_1, Q, S третий бит слева *(b1)* в их кодах равен 1, тогда как для точки P_2 бит равен 0. Поэтому точка P_2 должна быть перемещена в точку S. Перемещение точки P_1 в точку Q не является необходимым, но это перемещение выполняется потому, что оно не приносит вреда, а реализация алгоритма получается проще. Подобно оператору побитовой операции *ИЛИ,* обозначаемому вертикальной чертой (|) и упомянутому ранее, в языке C# имеется оператор побитовой операции *И,* обозначаемый знаком &. Побитовые операторы, обозначаемые знаками | и &, дают в результате битовые последовательности. Таким образом, циклическая конструкция

```
while (c1 | c2)...
```

будет выполнять действия, обозначенные многоточием (...), до тех пор, пока вычисление побитовой операции c1 | c2 будет выдавать битовую последовательность, содержащую 1 в каком-либо бите, то есть пока в аргументах c1 или c2 существует хотя бы один бит, содержащий 1. С другой стороны, оператор

```
if (c1 & c2) return;
```

вызывает прямой выход из функции, если битовая последовательность, полученная в результате выполнения операции *c1 & c2,* содержит хотя бы один единичный бит, то есть тогда, когда оба значения *c1* и *c2* содержат единичный бит в одной и той же позиции. Полный текст функции *clip* содержится в следующей программе, которая выполняет отсечение вложенных друг в друга пятиугольников.

```
/* Демонстрация работы алгоритма отсечения линий Коэна – Сазерленда */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ClipDemo
{
  public partial class Form1 : Form
  {
    // Задание координат окна отсечения
    float xmin = 1.0f, xmax = 9.0f, ymin = 1.0f, ymax = 6.0f;
    Graphics dc; Pen p;
    public Form1()
    {
      InitializeComponent();
      dc = pictureBox1.CreateGraphics();
      p = new Pen(Brushes.Black, 1);
    }
  /* Метод преобразования вещественной координаты X в целую */
    private int IX(double x)
    {
      double xx = x * (pictureBox1.Size.Width / 10.0) + 0.5;
      return (int)xx;
    }
  /* Метод преобразования вещественной координаты Y в целую */
    private int IY(double y)
    {
      double yy = pictureBox1.Size.Height - y *
                  (pictureBox1.Size.Height / 7.0) + 0.5;
      return (int)yy;
    }
  /* Функция вычерчивания линии (экран 10x7 условн. единиц) */
    private void Draw(double x1,double y1,double x2,double y2)
    {
      Point point1 = new Point(IX(x1), IY(y1));
      Point point2 = new Point(IX(x2), IY(y2));
      dc.DrawLine(p, point1, point2);
    }
  /* Функция получение кода положения точки относительно окна */
    private byte code(double x, double y)
    {
      return (byte)(
                  (Convert.ToByte(x < xmin) << 3) |
                  (Convert.ToByte(x > xmax) << 2) |
                  (Convert.ToByte(y < ymin) << 1) |
                   Convert.ToByte(y > ymax)
                   );
    }
```

```
/* Функция отсечения линий */
private void clip(double x1,double y1,double x2,double y2)
{
  byte c1, c2;
  double dx, dy;
  // Вызов функции получения четырёхбитных кодов точек
  c1 = code(x1, y1);  c2 = code(x2, y2);
  /* Цикл, пока обе точки не окажутся в границах окна */
  while ((c1 | c2) != 0)
  {
    /* Если оба кода содержат единицу в одной и той же
       позиции, то ничего не вычерчиваем */
    if ((c1 & c2) != 0) return;
    dx = x2 - x1;
    dy = y2 - y1;
    /* Пересчет координат для первой точки */
    if (c1 != 0)
    {
      if (x1 < xmin) {y1 += dy * (xmin - x1) / dx; x1 = xmin;}
      else
      if (x1 > xmax) {y1 += dy * (xmax - x1) / dx; x1 = xmax;}
      else
      if (y1 < ymin) {x1 += dx * (ymin - y1) / dy; y1 = ymin;}
      else
      if (y1 > ymax) {x1 += dx * (ymax - y1) / dy; y1 = ymax;}
      c1 = code(x1, y1);
    }
    else
    /* Пересчет координат для второй точки */
    {
      if (x2 < xmin) {y2 += dy * (xmin - x2) / dx; x2 = xmin;}
      else
      if (x2 > xmax) {y2 += dy * (xmax - x2) / dx; x2 = xmax;}
      else
      if (y2 < ymin) {x2 += dx * (ymin - y2) / dy; y2 = ymin;}
      else
      if (y2 > ymax) {x2 += dx * (ymax - y2) / dy; y2 = ymax;}
      c2 = code(x2, y2);
    }
  } /* Конец цикла while */
  Draw(x1, y1, x2, y2);
}
/* Функция рисования вложенных пятиугольников */
private void drawPentagon()
{
  int i;
  double r, alpha, phi0, phi, x0, y0, x1, y1, x2, y2;
  alpha = 72.0 * Math.PI / 180.0;
  phi0 = 0.0; x0 = 4.0; y0 = 4.0;
/* Вычерчивание границ окна */
  Draw(xmin, ymin, xmax, ymin);
  Draw(xmax, ymin, xmax, ymax);
  Draw(xmax, ymax, xmin, ymax);
  Draw(xmin, ymax, xmin, ymin);
```

```
      /* В пределах границ окна вычерчиваются 20 правильных
         концентрических пятиугольников */
        for (r = 0.5; r < 10.5; r += 0.5)
        {
        x2 = x0 + r*Math.Cos(phi0); y2 = y0 + r*Math.Sin(phi0);
          for (i = 1; i <= 5; i++)
          {
            phi = phi0 + i * alpha;
            x1 = x2; y1 = y2;
            x2 = x0 + r*Math.Cos(phi); y2 = y0 + r*Math.Sin(phi);
           /* Вызов функции отсечения линии и рисования */
            clip(x1, y1, x2, y2);
          }
        }
      }
    /* Вызов функции рисования прямоугольников при нажатии на кнопку */
      private void button1_Click(object sender, EventArgs e)
      {
        drawPentagon();
      }
    }
}
```

На рис. 18.6 показан результат работы этой программы.

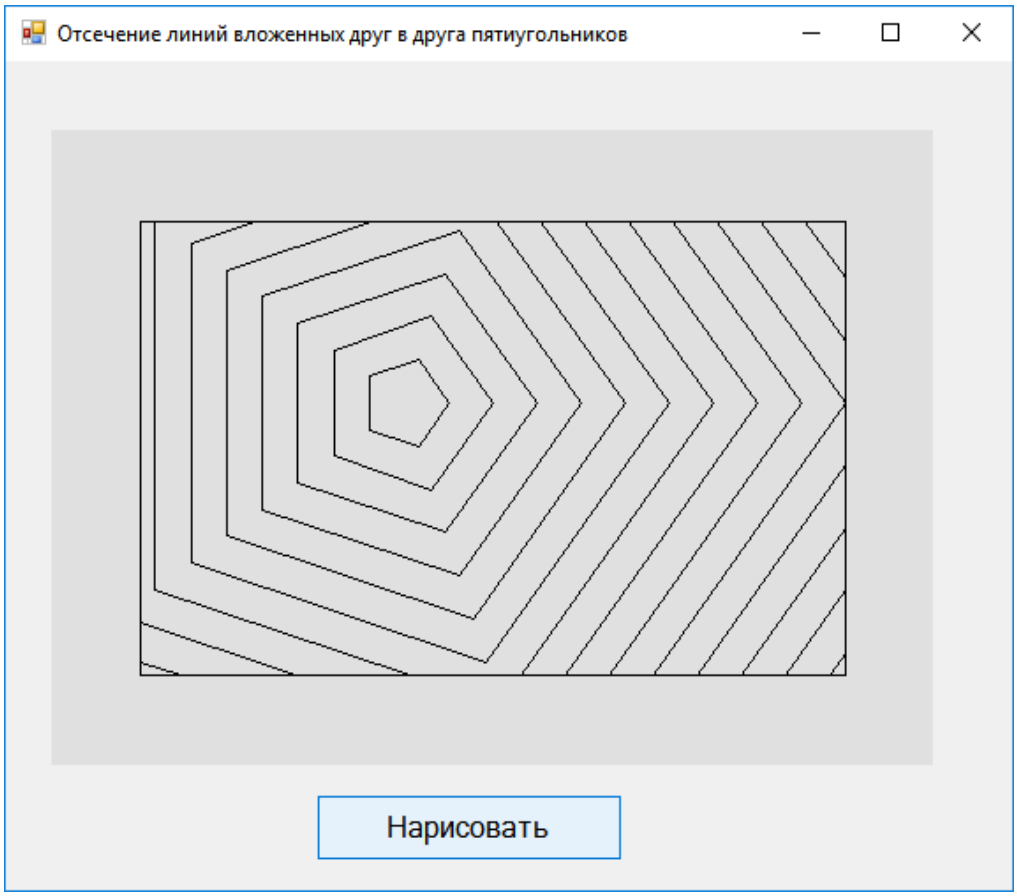


Рис. 18.6 – Результат работы программы ClipDemo

Тема 19. Автоматический подбор размеров и позиций

19.1 Автоматический подбор размеров и позиций для построения изображения: подбор коэффициентов и центра вывода изображения

Чтобы картинка была нарисована в пределах границ области вывода, необходимо сначала выполнить отсечение по заданному окну, как было описано ранее, а затем отразить окно и его содержимое на область вывода. Для большинства применений такая процедура вполне достаточна. Но сейчас опишем иной подход, который отличается по следующим аспектам:

1) объект будет вычерчиваться целиком, отсечение не понадобится;

2) окно будет определено расчётом, а не задано заранее;

3) при отражении окна на область вывода будет использован одинаковый коэффициент масштабирования по обеим осям в горизонтальном и вертикальном направлениях.

Из первого пункта следует, что объект должен быть конечным. Для большинства применений это ограничение не является серьёзным, но оно исключает панорамирование.

Второй пункт может быть реализован путём двойного просмотра данных, описывающих объект. Во время первого просмотра определяются границы окна $x_{\min}, x_{\max}, y_{\min}, y_{\max}$.Вычерчивание производится во время второго просмотра.

Третий пункт предполагает, что любой треугольник на картинке будет подобен исходному треугольнику на объекте, что означает неизменность угловых соотношений при отображении.

Предположим, что на рис. 19.1 имеем заданный треугольник PQR, координаты вершин которого определены в системе мировых координат:

$$x_P = 1.0,\ x_Q = 1.5,\ x_R = 1.2$$
$$y_P = 0.8,\ y_Q = 0.9,\ y_R = 1.1$$

Тогда расчётными значениями параметров окна будут:

$$x_{\min} = 1.0,\ \ x_{\max} = 1.5$$
$$y_{\min} = 0.8,\ \ y_{\max} = 1.1$$

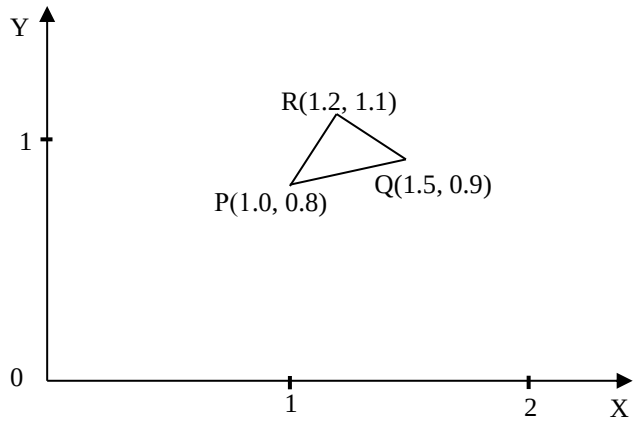


Рис. 19.1 – Объект, подлежащий масштабированию

Заметим, что крайние точки объекта располагаются на границах окна, чего не было в случае, рассмотренном в предыдущей теме.

Чтобы предусмотреть некоторое свободное пространство со всех сторон экрана или листа бумаги, необходимо задать размеры области вывода несколько меньше, чем они могли быть на самом деле. Например, можно задать $x_{\min} = 0.2$ вместо $x_{\min} = 0.0.$ Область вывода будет полностью определена при выборе:

$$X_{\min} = 0.2,\ \ X_{\max} = 8.2$$
$$Y_{\min} = 0.5,\ \ \ Y_{\max} = 6.5$$

Как и ранее, выполним расчёт коэффициентов масштабирования:

$$f_x = \frac{X_{\max} - X_{\min}}{x_{\max} - x_{\min}} = \frac{8.2 - 0.2}{1.5 - 1.0} = 16$$
$$f_y = \frac{Y_{\max} - Y_{\min}}{y_{\max} - y_{\min}} = \frac{6.5 - 0.5}{1.1 - 0.8} = 20$$

В качестве коэффициента масштабирования f выберем наименьшее из значений f_x и f_y. Напомним, что все расстояния умножаются на коэффициент масштабирования, так что если будет выбран коэффициент масштабирования больше, чем f_x или f_y, то часть картинки определённо выйдет за пределы области вывода. В данном примере получим: $f = f_x = 16$.

Понятно, что такой одинаковый коэффициент масштабирования приведёт к отображению треугольника, который точно равен по ширине размеру области вывода по оси X, но в направлении оси Y останется свободное пространство. Его желательно распределить поровну между нижней и верхней частями области вывода. Это можно реализовать, если для расчёта константы c_2 вместо минимального значения Y, как в предыдущей теме, выбрать позицию центра Y. Аналогичным образом вычисляется и значение c_1 .

$$x_C = 0.5*(x_{\min} + x_{\max}) = 0.5*(1.0+1.5) = 1.25$$
$$y_C = 0.5*(y_{\min} + y_{\max}) = 0.5*(0.8+1.1) = 0.95$$
$$X_C = 0.5*(X_{\min} + X_{\max}) = 0.5*(0.2+8.2) = 4.2$$
$$Y_C = 0.5*(Y_{\min} + Y_{\max}) = 0.5*(0.5+6.5) = 3.5$$
$$c_1 = X_C - f*x_C = 4.2 - 16*1.25 = -15.8$$
$$c_2 = Y_C - f*y_C = 3.5 - 16*0.95 = -11.7$$

Теперь для любой точки объекта (x, y) позиция отображаемой точки (X, Y) рассчитывается по формулам:

$$X = f*x + c_1 = 16*x - 15.8$$
$$Y = f*y + c_2 = 16*y - 11.7$$

19.2 Генерация случайной кривой

Составим программу, которая будет вычерчивать картинку, окно для которой не может быть задано заранее.

Используем случайные числа для генерации кривой непредсказуемой формы и размеров, где, как обычно, кривая аппроксимируется большим числом отрезков прямых линий.

Автоматическое масштабирование и позиционирование позволяют решить задачу, практически не разрешимую иным путём. Для каждого отрезка будут генерироваться значения координат x и y, которые будут записываться в файл на диске. Точнее говоря, будем записывать так называемые *структуры,* содержащие тройки:

x y code

где пара (x, y) определяет координаты точки, в которую должно перемещаться перо, а параметр *code* может принимать значение 0 или 1, означающее состояние пера «поднято» или «опущено» соответственно.

Другими словами,

$x\ y\ 0$ – означает *move(X, Y),*

$x\ y\ 1$ – означает *draw(X, Y),*

где X и Y – экранные координаты, соответствующие мировым координатам x и y.

В примере будем использовать специальную программу *CurvGen*, которая генерирует кривую и записывает тройки чисел в файл *Scratch.*

За выполнением этой программы должен последовать запуск общей программы вычерчивания *GenPlot,* которая дважды считывает тройки чисел. Первый раз – для определения параметров окна $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ и второй раз – для фактического выполнения операций перемещения пера и вычерчивания, используя экранные координаты X, Y, полученные путём пересчёта координат x, y в системе мировых координат.

В программе генерацию кривой будем начинать с точки начала координат и перемещаться каждый раз на одну единицу расстояния.

Всегда существует текущее направление φ и текущий угол поворота α. В исходном состоянии оба эти значения равны нулю. Перед каждым шагом угол α увеличивается на случайно выбранное значение угла в пределах от – 6° до + 6° (целое число). Полученный новый угол поворота добавляется к текущему направлению φ для получения нового направления.

Ограничим максимальное значение кривизны и уменьшим шанс для вырождения кривой в окружность. Для этого модифицируем описанный алгоритм, и будем задавать значение 0 для угла α каждый раз, когда его абсолютное значение превышает 15°.

Код программы *CurvGen*, генерирующей случайную кривую и записывающей тройки $x\ y\ code$ в файл *Scratch* приведен ниже:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace CurvGen
{
  public partial class Form1 : Form
  {
    struct Simple
    {
      public double xx; public double yy; public int ii;
    };
    Simple s;
    FileInfo my_file = new FileInfo("SCRATCH");
    BinaryWriter fw;
    Random rnd;
    int first = 1, phi = 0, alpha = 0;
    public Form1()
    {
      InitializeComponent();
    }
    /* Создание файла Scratch и открытие его на запись */
    void pfopen()
    {
      fw = new BinaryWriter(my_file.Open(FileMode.Create,
                                         FileAccess.Write));
    }
    /* Запись в файл точки с флагом перемещения */
    void pmove(double x, double y)
    {
      s.xx = x; s.yy = y; s.ii = 0;
      fw.Write(s.xx); fw.Write(s.yy); fw.Write(s.ii);
    }
    /* Запись в файл точки с флагом рисования */
    void pdraw(double x, double y)
    {
      s.xx = x; s.yy = y; s.ii = 1;
      fw.Write(s.xx); fw.Write(s.yy); fw.Write(s.ii);
    }
    /* Закрытие файла */
    void pfclose()
    {
      fw.Close();
    }
    /* Функция, возвращающая новое направление кривой */
    double direction()
    {
```

```
      if (first == 1) { first = 0; rnd = new Random(); }
      alpha += rnd.Next(10000) % 13 - 6;
      if (Math.Abs(alpha) > 15) alpha = 0;
      phi += alpha;
      return ((double)phi * Math.PI / 180.0);
    }
    /* Главная функция генерации точек случайной кривой */
    void curvgen()
    {
      int i, N = 500;
      double x = 0.0, y = 0.0, x0, y0, phi = direction();
      pfopen(); pmove(x, y);
      for (i = 1; i <= N; i++)
      {
        x0 = x; y0 = y;
        phi = direction();
        x = x0 + Math.Cos(phi); y = y0 + Math.Sin(phi);
        pdraw(x, y);
      }
      pfclose();
    }
    /* Вызов функции генерации случайной кривой */
    private void button1_Click_1(object sender, EventArgs e)
    {
      curvgen();
      MessageBox.Show("Случайная кривая успешно сгенерирована
                       в файл Scratch");
    }
  }
}
```

На рисунке 19.1 показана работа программы *CurvGen*.

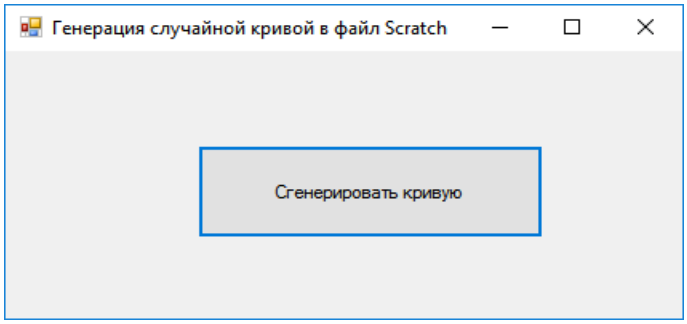


Рис. 19.1 – Генерация и запись точек случайной кривой в файл Scratch

В функции *direction* показано обращение к функции инициализации датчика случайных чисел. Таким образом, будут генерироваться различные кривые при каждом новом запуске программы. Функция выдаёт большое неотрицательное целое число. Оно преобразуется в целое число в диапазоне от 0 до 12 путём использования остатка от деления на 13, то есть

$$0 <= rnd.Next(10000) \ \% \ 13 <= 12$$
$$-6 <= rnd.Next(10000) \ \% \ 13 - 6 <= 6$$

Переменная s обозначает структуру, содержащую три числа *x y code*, которые будут записываться в файл.

19.3. Общий алгоритм определения размеров и черчения

Рассмотрим теперь общую программу *GenPlot*, которая будет считывать тройки из файла *Scratch*, определять размеры окна, выполнять пересчёт мировых координат в экранные координаты и вычерчивать картинку в пределах заданной области вывода. Также вычерчиваются небольшие уголки в углах области вывода и в середине нижней границы, так что абстрактную картинку можно ориентировать относительно верха и низа. Если уголки нежелательны, их легко можно исключить.

Заметим, что в этой программе файл *Scratch* открывается дважды. Операции закрытия и открытия приводят к «перемотке» файла к началу.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace GenPlot
{
  public partial class Form1 : Form
  {
    struct Simple
    {
      public double xx;  public double yy;  public int ii;
    };
    Simple s;
    FileInfo my_file = new FileInfo("SCRATCH");
    BinaryReader fr;
    double x, y, xmin, xmax, ymin, ymax;
    double X, Y, Xmin, Xmax, Ymin, Ymax;
    double fx, fy, f, xC, yC, XC, YC, c1, c2;
    Graphics dc; Pen p;
```

```
  public Form1()
  {
    InitializeComponent();
    /* Задание границ области вывода по умолчанию  */
    Xmin = 0.2; Xmax = 8.2; Ymin = 0.5; Ymax = 6.5;
    textBox1.Text = Xmin.ToString();
    textBox2.Text = Ymin.ToString();
    textBox3.Text = Xmax.ToString();
    textBox4.Text = Ymax.ToString();
    dc = pictureBox1.CreateGraphics();
    p = new Pen(Brushes.Black, 1);
  }
/* Метод преобразования вещественной координаты X в целую */
  private int IX(double x)
  {
    double xx = x * (pictureBox1.Size.Width / 10.0) + 0.5;
    return (int)xx;
  }
/* Метод преобразования вещественной координаты Y в целую */
  private int IY(double y)
  {
    double yy = pictureBox1.Size.Height - y *
                (pictureBox1.Size.Height / 7.0) + 0.5;
    return (int)yy;
  }
/* Функция вычерчивания линии (экран 10x7 условн. единиц) */
  private void Draw(double x1,double y1,double x2,double y2)
  {
    Point point1 = new Point(IX(x1), IY(y1));
    Point point2 = new Point(IX(x2), IY(y2));
    dc.DrawLine(p, point1, point2);
  }
/* Функция прорисовки меток-уголков области вывода */
  private void initViewPort(double Xmin, double Ymin,
                            double Xmax, double Ymax)
  {
    Draw(Xmin,Ymin,Xmin+0.2,Ymin);
    Draw(Xmin, Ymin, Xmin, Ymin+0.2);
    Draw(Xmin, Ymax, Xmin + 0.2, Ymax);
    Draw(Xmin, Ymax, Xmin, Ymax - 0.2);
    Draw(Xmax, Ymin, Xmax - 0.2, Ymin);
    Draw(Xmax, Ymin, Xmax, Ymin + 0.2);
    Draw(Xmax, Ymax, Xmax - 0.2, Ymax);
    Draw(Xmax, Ymax, Xmax, Ymax - 0.2);
    Draw(Xmax / 2 - 0.2, Ymin, Xmax / 2 + 0.2, Ymin);
    Draw(Xmax / 2, Ymin, Xmax /2 , Ymin + 0.2);
  }
/* Чтение из файла Scratch и вычерчивание кривой */
  private void button2_Click(object sender, EventArgs e)
  {
    double Xold = 0, Yold = 0;
/* Первый проход для определения границ окна xmin, ymin,
                                              xmax, ymax */
```

```
        fr = new BinaryReader(my_file.Open(FileMode.Open,
                                           FileAccess.Read));
        while (fr.BaseStream.Position < fr.BaseStream.Length)
        {
          s.xx = fr.ReadDouble(); s.yy = fr.ReadDouble();
          s.ii = fr.ReadInt32();
          x = s.xx; y = s.yy;
          if (x < xmin) xmin = x;
          if (x > xmax) xmax = x;
          if (y < ymin) ymin = y;
          if (y > ymax) ymax = y;
        }
        fr.Close();
    /* Вызов функции отображения границ области вывода */
        initViewPort(Xmin, Ymin, Xmax, Ymax);
    /* Получение коэффициентов формулы перевода мировых
       координат в экранные */
        fx = (Xmax - Xmin) / (xmax - xmin);
        fy = (Ymax - Ymin) / (ymax - ymin);
        f = (fx < fy ? fx : fy);
        xC = 0.5 * (xmin + xmax); yC = 0.5 * (ymin + ymax);
        XC = 0.5 * (Xmin + Xmax); YC = 0.5 * (Ymin + Ymax);
        c1 = XC - f * xC;  c2 = YC - f * yC;
    /* Второй проход для вычерчивания */
        fr = new BinaryReader(my_file.Open(FileMode.Open,
                                           FileAccess.Read));
        while (fr.BaseStream.Position < fr.BaseStream.Length)
        {
          s.xx = fr.ReadDouble();  s.yy = fr.ReadDouble();
          s.ii = fr.ReadInt32();
          x = s.xx; y = s.yy;
          X = f * x + c1; Y = f * y + c2;
          if (s.ii == 1) { Draw(Xold, Yold, X, Y); }
          Xold = X; Yold = Y;
        }
        fr.Close();
      }
    /* Очистка области вывода */
      private void button3_Click(object sender, EventArgs e)
      { dc.Clear(Color.White); }
      /* Изменение границ области вывода */
      private void button1_Click(object sender, EventArgs e)
      {
        Xmin = Convert.ToDouble(textBox1.Text);
        Ymin = Convert.ToDouble(textBox2.Text);
        Xmax = Convert.ToDouble(textBox3.Text);
        Ymax = Convert.ToDouble(textBox4.Text);
        MessageBox.Show("Координаты области вывода изменены");
      }
    }
}
```

На рисунке 19.2 показана работа программы *CurvGen*. Область вывода:

$$X_{\min} = 0.2,\ \ X_{\max} = 8.2$$
$$Y_{\min} = 0.5,\ \ \ Y_{\max} = 6.5$$

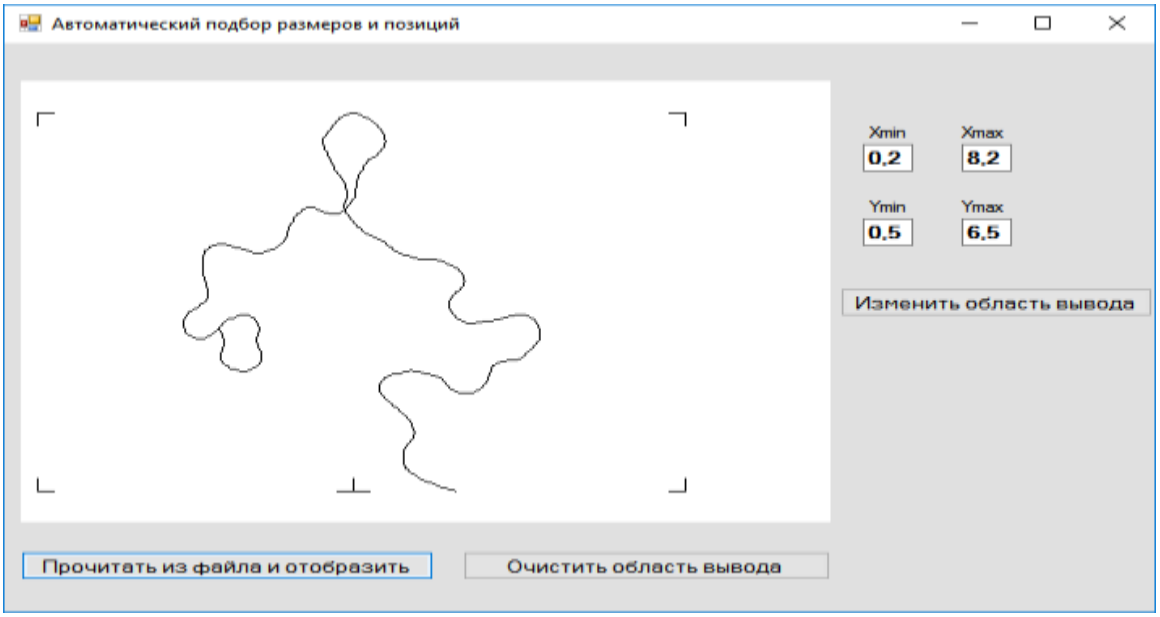


Рис. 19.2 – Результат работы программы автоматического подбора размеров и позиций

На рисунке 19.3 показана работа программы *CurvGen*. Область вывода:

$$X_{\min} = 0.2,\ \ X_{\max} = 5.2$$
$$Y_{\min} = 0.5,\ \ \ Y_{\max} = 6.5$$

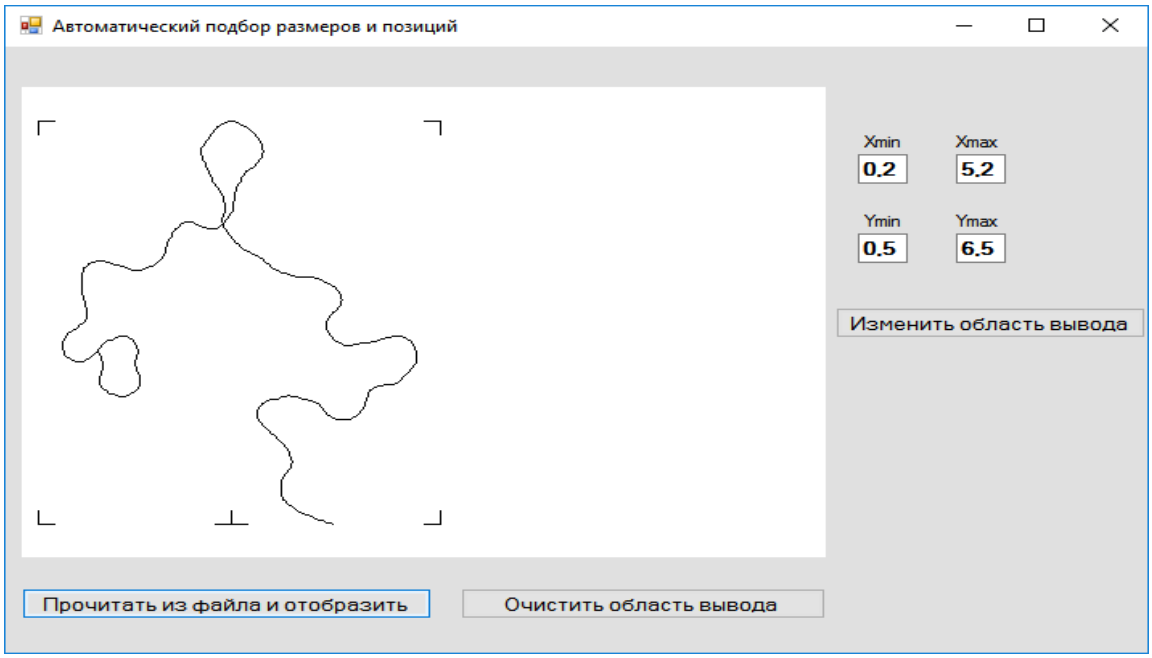


Рис. 19.3 – Результат работы программы *CurvGen* с измененной областью вывода

Тема 20. Сглаживание кривых

20.1 Применение рекурсии для решения задач

Ряд задач может быть решен с применением рекурсии. Например, для заданных трех чисел X_C, Y_C, r соединить точки с координатами:

$$X_i = X_C + r * \cos(\varphi_i)$$
$$Y_i = Y_C + r * \sin(\varphi_i)$$
$$(i = 0, 1, 2, 3, 4, 5; \varphi_i = i * 144^o),$$

что в результате дает звезду. Последовательно, как часть задачи, выполнить подобную работу еще пять раз, но теперь с другими тремя числами:

$$X_{C'} = X_C + 2 * r * \cos(\varphi_j)$$
$$Y_{C'} = Y_C + 2 * r * \sin(\varphi_j)$$
$$(j = 0, 1, 2, 3, 4; \varphi_j = 36^o + j * 72^o)$$

Выполнение задачи продолжается до тех пор, пока задаваемая величина *r* будет не меньше 0.1. Для начальной (главной) задачи задаваемые числа будут равны $X_C = 0, Y_C = 0, r = 1$. Как и ранее, будем использовать автоматическое масштабирование и размещение. Ниже приводится программа *Stars*, решающая поставленную задачу.

```
/*Stars: Звёзды различных размеров */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace Stars
{
  public partial class Form1 : Form
  {
    struct Simple
    {
      public double xx; public double yy; public int ii;
    };
    Simple s;
```

```
    FileInfo my_file = new FileInfo("SCRATCH");
    BinaryWriter fw;
    public Form1()
    { InitializeComponent(); }
  /* Создание файла Scratch и открытие его на запись */
    void pfopen()
    {
      fw = new BinaryWriter(my_file.Open(FileMode.Create,
                                         FileAccess.Write));
    }
  /* Запись в файл точки с флагом перемещения */
    void pmove(double x, double y)
    {
      s.xx = x; s.yy = y; s.ii = 0;
      fw.Write(s.xx); fw.Write(s.yy); fw.Write(s.ii);
    }
  /* Запись в файл точки с флагом рисования */
    void pdraw(double x, double y)
    {
      s.xx = x; s.yy = y; s.ii = 1;
      fw.Write(s.xx); fw.Write(s.yy); fw.Write(s.ii);
    }
  /* Закрытие файла */
    void pfclose()
    { fw.Close(); }
  /* Главная функция генерации звезд */
    void star(double xC, double yC, double r)
    {
      double phi, r_half, r_double, factor = Math.PI / 180.0;
      int i;
      if (r < 0.1) return;
      pmove(xC + r, yC);
      for (i = 1; i <= 5; i++)
      {
        phi = i * 144 * factor;
        pdraw(xC + r * Math.Cos(phi), yC + r * Math.Sin(phi));
      }
      r_half = 0.5 * r; r_double = 2 * r;
      for (i = 0; i < 5; i++)
      {
        phi = (36 + i * 72) * factor;
        star(xC + r_double * Math.Cos(phi), yC + r_double *
                                    Math.Sin(phi), r_half);
      }
    }
  /* Вызов функции генерации случайной кривой */
    private void button1_Click(object sender, EventArgs e)
    {
      pfopen();  star(0.0, 0.0, 1.0); pfclose();
      MessageBox.Show("Звезды сгенерированы в файл Scratch");
    }
  }
}
```

На рис. 20.1 показан результат работы программы *Stars*.

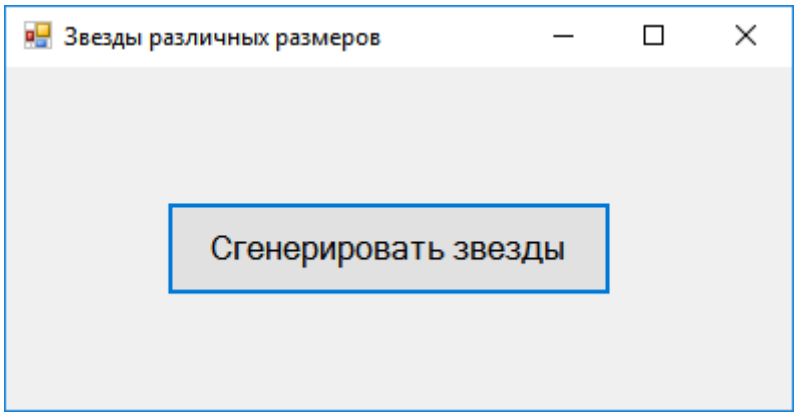


Рис. 20.1 – Результат работы программы Stars

После этой программы должна быть вызвана программа *GenPlot*, описанная ранее. На рис. 20.2 показан результат работы программы *GenPlot*, читающей информацию из файла *Scratch*, созданного программой *Stars*.

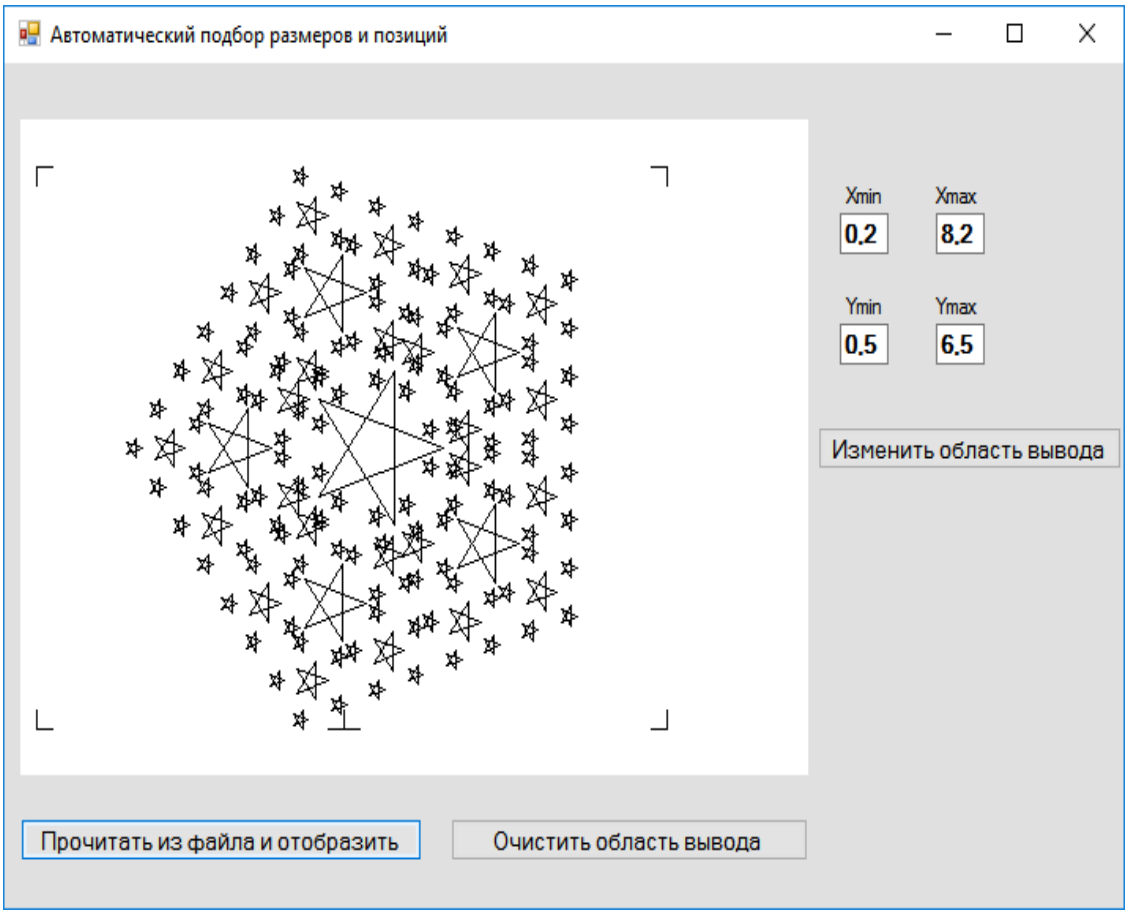


Рис. 20.2 – Результат совместной работы программ Stars и GenPlot

20.2 Построение дерева Пифагора

Следующий пример рекурсии – «Пифагорово дерево». Оно часто изображается так, как показано на рис. 20.3. Каждый из прямоугольных треугольников в этом дереве имеет внутренний угол, равный 45°.

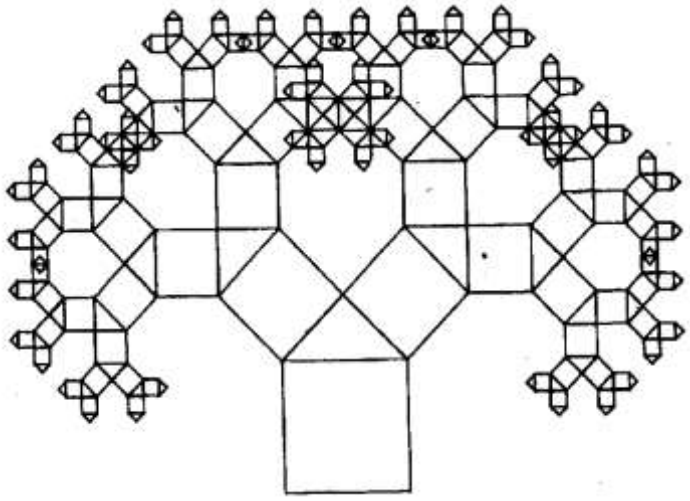


Рис. 20.3 «Пифагорово дерево», регулярная версия

Воспользуемся генератором случайных чисел для создания более общей программы, которая может сформировать не только рисунок 20.3, но также генерировать и менее регулярные деревья.

Углы, задаваемые равными 45° на рисунке 20.3, в общем случае будут задаваться случайным образом в пределах между (45 – *delta)*° и (45 + *delta)*°, где значение *delta* задается в качестве входного параметра вместе с параметром *n*, определяющим глубину рекурсии. Регулярная версия, изображенная на рис. 20.3, получается при задании *delta* = 0 и n = 7. На рисунке параметр *n* определяет количество треугольников на пути от корня до листьев дерева. Сердцевиной программы будет рекурсивная функция square_and_triangle («квадрат и треугольник») с параметром *n*, определяющим глубину рекурсии, в качестве первого аргумента. Если значение параметра *n* больше нуля, то задачей функции square_and_triangle будет вычертить квадрат и над ним треугольник, а затем дважды обратиться

к самой себе с соответствующими новыми аргументами, первый из которых задается равным *n – 1*. Размер и положение квадрата полностью определяются четырьмя параметрами: X_0, Y_0, a, φ (см. рисунок 20.4).

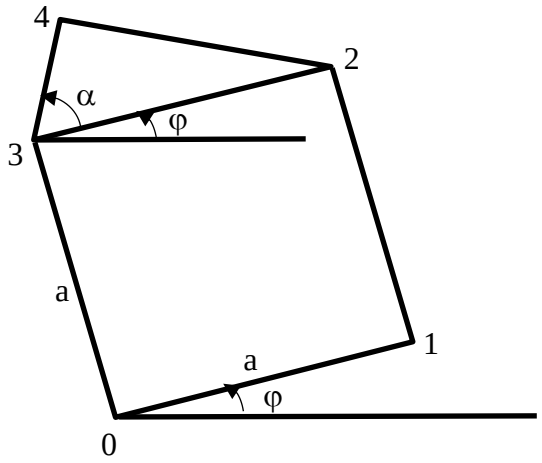


Рис. 20.4 – Нумерация точек

Для вычерчивания треугольника необходимо знать угол α. Этот угол в градусах равен 45 + *deviation*, где *deviation* равно одному из целых чисел ряда – *delta*, – *delta* + 1, ... , *delta*, выбираемому случайным образом.

На рисунке 20.4 необходимые точки пронумерованы последовательными числами 0, 1, 2, 3, 4. Координаты X_0, Y_0 точки O задаются в обращении к функции. Для вычисления остальных точек вначале рассмотрим более простую ситуацию при $\varphi = 0$, то есть когда сторона (0,1) квадрата занимает горизонтальное положение.

В этом положении координаты точек определить очень просто. Они записываются в массивах *x* и *y*. Затем все точки поворачиваются вокруг точки O на угол φ. Результат поворота записывается в массивах *xx* и *yy*.

Ниже приведен код программы.

```
/* Pyth_Tree: Вариант дерева Пифагора */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace Pyth_Tree
{
  public partial class Form1 : Form
  {
    struct Simple
    {
      public double xx; public double yy; public int ii;
    };
    Simple s;
    FileInfo my_file = new FileInfo("SCRATCH");
    BinaryWriter fw;
    Random rnd;
    int delta;
    public Form1()
    { InitializeComponent(); }
  /* Создание файла Scratch и открытие его на запись */
    void pfopen()
    {
      fw = new BinaryWriter(my_file.Open(FileMode.Create,
                                         FileAccess.Write));
    }
  /* Запись в файл точки с флагом перемещения */
    void pmove(double x, double y)
    {
      s.xx = x; s.yy = y; s.ii = 0;
      fw.Write(s.xx); fw.Write(s.yy); fw.Write(s.ii);
    }
  /* Запись в файл точки с флагом рисования */
    void pdraw(double x, double y)
    {
      s.xx = x; s.yy = y; s.ii = 1;
      fw.Write(s.xx); fw.Write(s.yy); fw.Write(s.ii);
    }
  /* Закрытие файла */
    void pfclose()
    { fw.Close(); }
  /* Главная функция генерации квадрата и треугольника */
    void square_and_triangle(int n, double x0, double y0,
                             double a, double phi)
    {
      double []x = new double[5]; double[] y = new double[5];
      double[] xx = new double[5]; double[] yy = new double[5];
      double cphi, sphi, c1, c2, b, c, alpha, calpha, salpha;
      int i, deviation;
      if (n == 0) return;
      /* углы phi и alpha в радианах, угол delta в градусах */
      deviation = rnd.Next(1000) % (2 * delta + 1) - delta;
      alpha = (45 + deviation) * Math.PI / 180.0;
      x[0] = x[3] = x0; x[1] = x[2] = x0 + a;
      y[0] = y[1] = y0; y[2] = y[3] = y0 + a;
```

```
        calpha = Math.Cos(alpha); salpha = Math.Sin(alpha);
        c = a * calpha; b = a * salpha;
        x[4] = x[3] + c * calpha;
        y[4] = y[3] + c * salpha;
        /* Поворот вокруг точки (x0, y0) на угол phi;*/
        cphi = Math.Cos(phi); sphi = Math.Sin(phi);
        c1 = x0 - x0 * cphi + y0 * sphi;
        c2 = y0 - x0 * sphi - y0 * cphi;
        for (i = 0; i < 5; i++)
        {
          xx[i] = x[i] * cphi - y[i] * sphi + c1;
          yy[i] = x[i] * sphi + y[i] * cphi + c2;
        }
        pmove(xx[3], yy[3]);
        for (i = 0; i < 5; i++) pdraw(xx[i], yy[i]);
        pdraw(xx[2], yy[2]);
        square_and_triangle(n - 1, xx[3], yy[3], c, phi + alpha);
        square_and_triangle(n - 1, xx[4], yy[4], b, phi + alpha -
                                                  0.5 * Math.PI);
      }
    /* Вызов функции генерации дерева Пифагора */
      private void button1_Click(object sender, EventArgs e)
      {
        int n;
        pfopen();
        rnd = new Random();
        n = 7;  delta = 30;
        square_and_triangle(n, 0.0, 0.0, 1.0, 0.0);
        pfclose();
        MessageBox.Show("Дерево Пифагора записано в файл Scratch");
      }
    }
}
```

Результат работы программы *Pyth_Tree* показан на рис. 20.5.

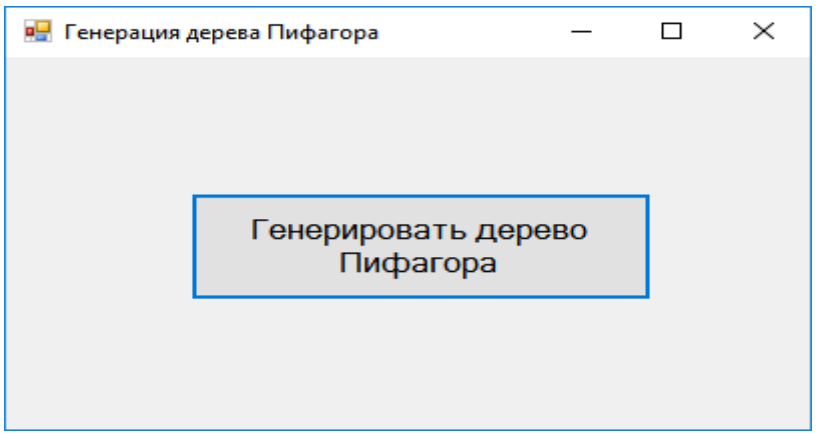


Рис. 20.5 – Результат работы программы Pyth_Tree

Эта программа формирует файл *Scratch*, который должен быть обработан программой *Genplot*. Графический результат совместной работы программ *Pyth_Tree* и *Genplot* для *delta* = 30 и n = 7 показан на рис. 20.6.

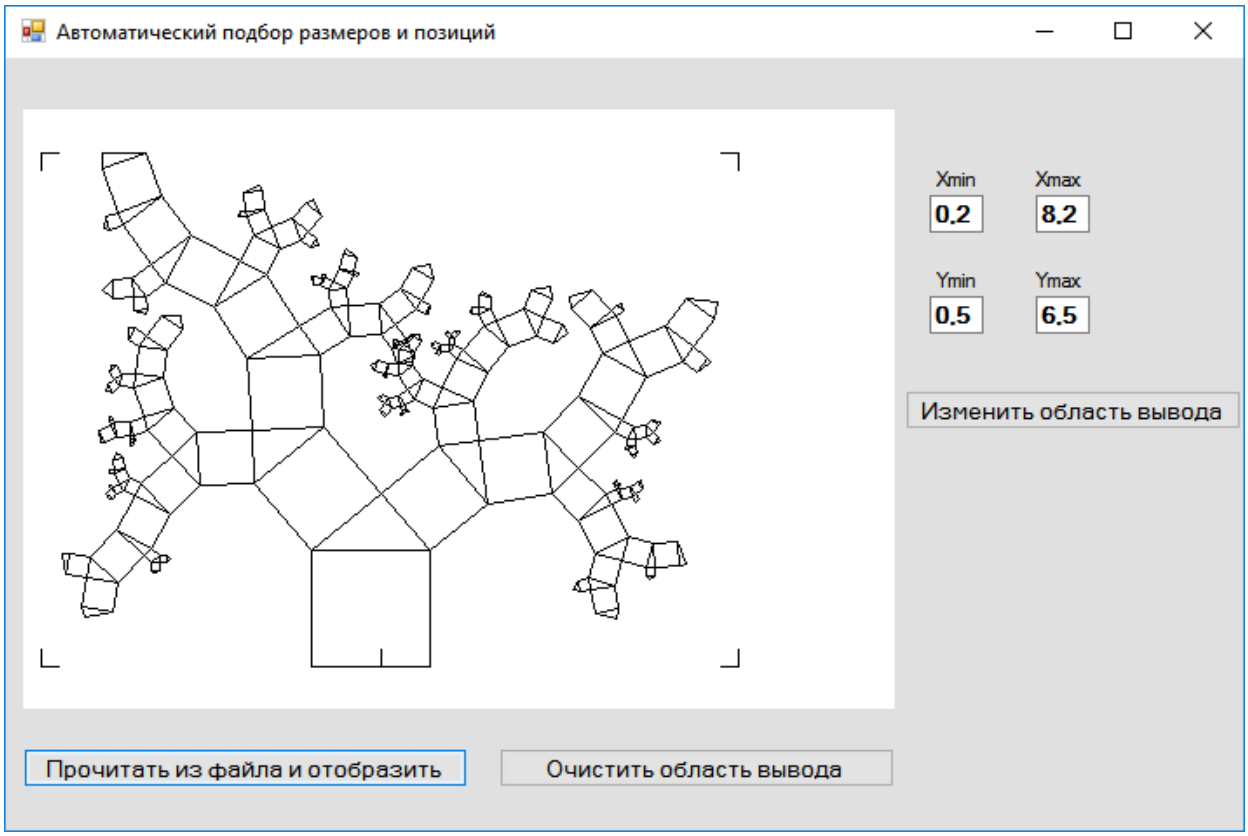


Рис. 20.6 – Результат совместной работы программ Pyth_Tree и GenPlot

20.3 Сглаживание кривых

В автоматизированном проектировании и управлении станками часто требуется построить гладкую кривую или гладкую поверхность по набору заданных точек. Рассмотрим только двухмерное пространство, поэтому ограничимся анализом кривых в плоскости *xy*. Анализ плоских кривых послужит основой для построения поверхностей на более поздней стадии.

Из нескольких возможных способов построения гладких кривых выберем форму B-сплайна. Из заданной последовательности точек выбираются две соседние точки, и между ними строится кривая кубического полинома на основе позиций четырёх точек – двух уже упомянутых и двух соседних с ними точек. B-сплайн обеспечивает

получение более гладких кривых, чем другие способы сглаживания за счет того, что получаемые кривые не проходят точно через заданные точки.

Математически гладкость кривых выражается в терминах непрерывности параметрических представлений *x(t)* и *y(f)* и их производных. Кривые типа B-сплайна обладают свойством непрерывности даже вторых производных *x''(t) и y''(t)* в точках стыковки двух соседних сегментов кривой. На рис. 20.7 можно видеть, как выгладят кривые, если их нулевая, первая и вторая производные не непрерывны в некоторой точке.

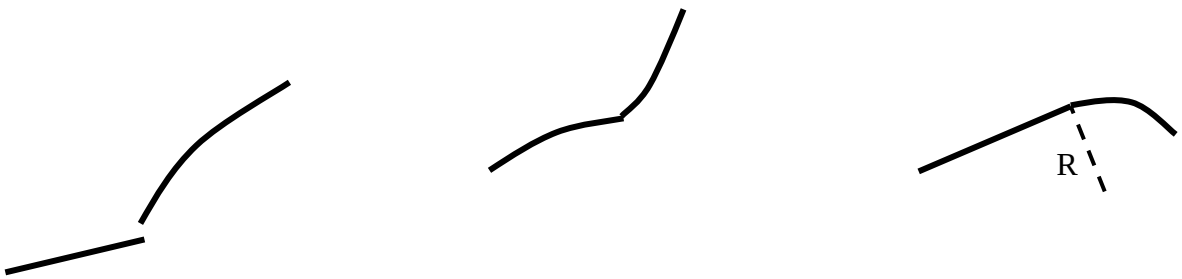


Рис. 20.7 – Производные кривых не непрерывны в некоторой точке:
а – нулевая производная; б – первая производная; в –вторая производная

Кривая на рис. 20.7 (в) может считаться гладкой, но она не удовлетворяет строгим требованиям, которые выполняются в способе B-сплайна.

Будем использовать параметрическое представление кривых. Любая точка части кривой между двумя заданными последовательными точками P и Q будет иметь координаты $x(t)$ и $y(t)$, где t увеличивается от 0 до 1, если отслеживается часть кривой от точки P до точки Q. Можно считать, что t – это время. Если имеются заданные точки:

$$P_0(x_0, y_0)$$
$$P_1(x_1, y_1)$$
$$\ldots\ldots\ldots\ldots$$
$$P_n(x_n, y_n)$$

то часть кривой B-сплайна между двумя последовательными точками P_i и P_{i+1} получается путем вычисления функций $x(t)$ и $y(t)$ для изменения t от 0 до 1:

$$x(t) = \{(a_3 * t + a_2) * t + a_1\} * t + a_0$$
$$y(t) = \{(b_3 * t + b_2) * t + b_1\} * t + b_0$$

Эти уравнения содержат следующие коэффициенты:

$$a_3 = (-x_{i-1} + 3 * x_i - 3 * x_{i+1} + x_{i+2}) / 6$$
$$a_2 = (x_{i-1} - 2 * x_i + x_{i+1}) / 2$$
$$a_1 = (-x_{i-1} + x_{i+1}) / 2$$
$$a_0 = (x_{i-1} + 4 * x_i + x_{i+1}) / 6$$

Коэффициенты b_3, b_2, b_1, b_0 вычисляются по значениям $y_{i-1}, y_i, y_{i+1}, y_{i+2}$ аналогичным образом. Вышеприведенные, формулы пригодны для эффективных вычислений. Вычисление значений $x(t)$ производится быстрее по правилу Горнера, чем по обычному полиномиальному выражению. Коэффициенты a_3, a_2, a_1, a_0 вычисляются только однажды для каждого сегмента кривой, что очень важно, поскольку на каждом сегменте кривой может вычисляться большое число промежуточных точек $x(t)$ и $y(t)$.

Для получения некоторого представления о свойствах кривой в точках стыковки двух сегментов рассмотрим функцию $x(t)$ и ее первую и вторую производные для значений $t = 0$ и $t = 1$. Функция $y(t)$ будет обладать аналогичными свойствами.

Используя вышеприведённые уравнения и упрощая, получаем:

$$x(0) = a_0 = (x_{i-1} + 4 * x_i + x_{i+1}) / 6$$
$$x(1) = a_3 + a_2 + a_1 + a_0 = a_0 = (x_i + 4 * x_{i+1} + x_{i+2}) / 6$$

Можно видеть, что значение $x(0)$ не равно в точности x-координате x_i точки P_i. Оно зависит от позиций точек P_{i-1} и P_{i+1}.

Из рисунка 20.8 видно, что точка B является конечной точкой сегмента AB, но одновременно и начальной точкой сегмента BC.

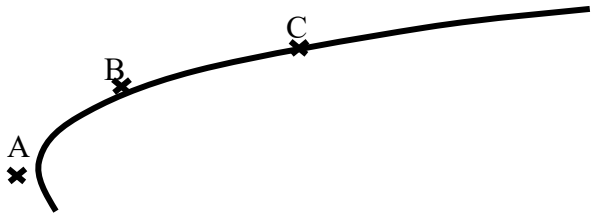


Рис.20.8 – Три последовательные точки

Для сегмента AB, для которого точка B является конечной, имеем:

$$A = P_i\ ,\ B = P_{i+1}\ ,\ C = P_{i+2}$$

$$x_B^* = x(1) = (x_i\ + 4 * x_{i+1}\ + x_{i+2}) / 6 = (x_A\ + 4 * x_B\ + x_C) / 6,$$

где через x_B^* обозначено вычисленное значение координаты x для точки B.

Рассматривая эту точку B как начальную точку сегмента BC, получим:

$$A = P_{i-1}\ ,\ B = P_i\ \ ,\ C = P_{i+1}$$

$$x_B^* = x(0) = (x_{i-1} + 4 * x_i\ + x_{i+1}) / 6 = (x_A\ + 4 * x_B\ + x_C) / 6$$

Видно, что оба способа вычисления значения x дают одинаковый результат, что означает непрерывность функции $x(t)$ в точке B. Продифференцировав $x(t)$ дважды, найдем производные $x'(t)$ и $x''(t)$. Подставляя в них значения $t = 0$ и $t = 1$, как это было сделано для $x(t)$, можно будет убедиться, что производные непрерывны в точке B. Поскольку функция $y(t)$ и ее первые две производные тоже непрерывны, то становится ясно, что кривая B-сплайна очень гладкая.

Для расчета любого сегмента кривой между точками P_i и P_{i+1} используются также точки P_{i-1} и P_{i+2}. Из этого следует, что первый сегмент кривой будет располагаться между точками P_1 и P_2, а последний – между точками P_{n-2} и P_{n-1}. Поэтому начальная и конечная точки всей кривой будут располагаться вблизи P_1 и P_{n-1}, но не вблизи P_0 и P_n. Приведенная ниже программа считывает числа

n
x_0 , y_0
x_1 , y_1
.....
x_n , y_n

из файла *Curv.dat*. При выводе каждая из $n+1$ точек обозначается маркером в виде крестика. Затем вычерчивается кривая B сплайна.

Ниже приведен код программы *CurvFit*.

```
/* CurvFit: Сглаживание кривой с применением B-сплайна */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace CurvFit
{
  public partial class Form1 : Form
  {
    int MAX = 100; int N = 30;
    Graphics dc;  Pen p;
    double[] x, y;
    double eps = 0.04, X, Y, t, xA, xB, xC, xD, yA, yB, yC,
                yD, a0, a1, a2, a3, b0, b1, b2, b3;
    int n, i, j, first;
    public Form1()
    {
      InitializeComponent();
      dc = pictureBox1.CreateGraphics();
      p = new Pen(Brushes.Black, 1);
      x = new double[MAX]; y = new double[MAX];
    }
  /* Метод преобразования вещественной координаты X в целую */
    private int IX(double x)
    {
      double xx = x * (pictureBox1.Size.Width / 10.0) + 0.5;
      return (int)xx;
    }
  /* Метод преобразования вещественной координаты Y в целую */
    private int IY(double y)
    {
      double yy = pictureBox1.Size.Height - y *
                        (pictureBox1.Size.Height / 7.0) + 0.5;
      return (int)yy;
    }
  /* Функция вычерчивания линии (область вывода 10x7 усл.ед.*/
```

```
    private void Draw(double x1, double y1, double x2, double
y2)
    {
      Point point1 = new Point(IX(x1), IY(y1));
      Point point2 = new Point(IX(x2), IY(y2));
      dc.DrawLine(p, point1, point2);
    }

    /* Функция генерации B-сплайна */
    private void curv_Fit()
    {
      double Xold=0, Yold=0;
      /* Заданные точки отмечаются маркером */
      for (i = 0; i <= n; i++)
      {
        X = x[i]; Y = y[i];
        Draw(X - eps, Y - eps, X + eps, Y + eps);
        Draw(X + eps, Y - eps, X - eps, Y + eps);
      }
      /* Отрисовка B-сплайна */
      first = 1;
      for (i = 1; i < n - 1; i++)
      { /* Вычисление коэффициентов */
        xA = x[i - 1]; xB = x[i]; xC = x[i + 1]; xD = x[i + 2];
        yA = y[i - 1]; yB = y[i]; yC = y[i + 1]; yD = y[i + 2];
        a3 = (-xA + 3 * (xB - xC) + xD) / 6.0;
        b3 = (-yA + 3 * (yB - yC) + yD) / 6.0;
        a2 = (xA -2 * xB + xC) /2.0;
        b2 = (yA - 2 * yB + yC)/ 2.0;
        a1 = (xC - xA)/ 2.0;
        b1 = (yC - yA)/ 2.0;
        a0 = (xA + 4 * xB + xC) / 6.0;
        b0 = (yA + 4 * yB + yC) / 6.0;
        /* Отрисовка сегмента дуги */
        for (j = 0; j <= N; j++)
        {
          t = (double)j / (double)N;
          X = ((a3 * t + a2) * t + a1) * t + a0;
          Y = ((b3 * t + b2) * t + b1) * t + b0;
          if (first==1) { first = 0; }
          else Draw(Xold, Yold, X,Y);
          Xold = X; Yold = Y;
        }
      }
    }
  /* Функция чтения количества и координат точек из файла
Curv.dat */
    private void read_File()
    {
      StreamReader reader = new StreamReader("Curv.dat");
      /* Чтение из файла количества точек*/
      n = Convert.ToInt32(reader.ReadLine());
```

```
/* Чтение координат точек точек*/
      for (i = 0; i <= n; i++)
      {
        string[] xy = reader.ReadLine().Split(' ');
        x[i] = Convert.ToDouble(xy[0]);
        y[i] = Convert.ToDouble(xy[1]);
      }
      reader.Close();
    }
    /* Вызов функций чтения данных и отрисовки B-сплайна */
    private void button1_Click(object sender, EventArgs e)
    {
  /* Вызов функции чтения данных из файла Curv.dat */
      read_File();
  /* Вызов функции отрисовки B-сплайна */
      curv_Fit();
    }
  }
}
```

Если в текстовом файле *Curv.dat* 6удут записаны числа:

20
0,75 1,4
0,5 1
0,75 0,6
1,5 0,4
2,25 0,7
2,5 0,9
3 0,85
5 0,75
6,25 0,8
6,5 0,85
6,5 1
6,5 1,15
6,25 1,2
5 1,25
3 1,15
2,5 1,1
2,25 1,3
1,5 1,6
0,75 1,4
0,5 1
0,75 0,6

то в результате работы программы получим изображение, показанное на рис. 20.9.

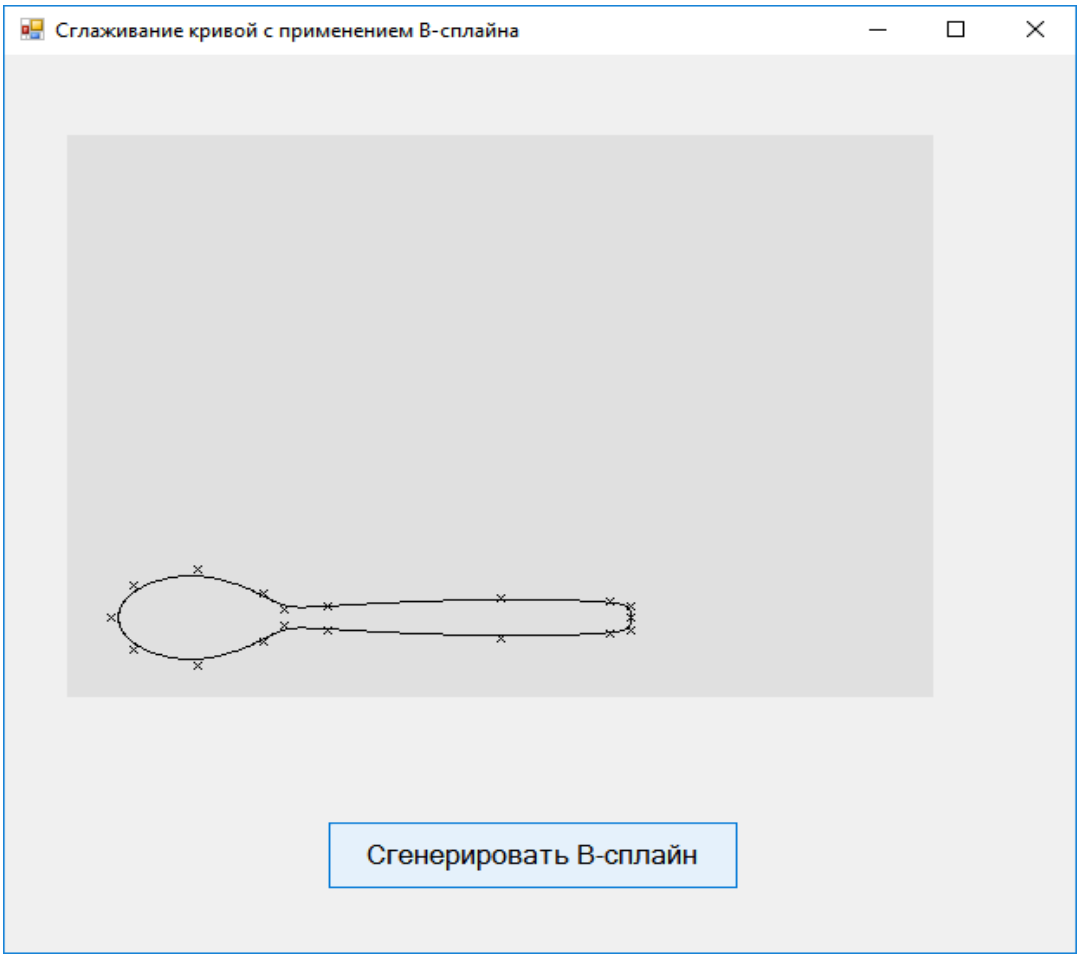


Рис. 20.9 – Результат работы программы *CurvFit*

Тема 21. Геометрический инструмент трехмерной графики

21.1 Векторы

Вектор – это направленный отрезок прямой линии, характеризуемый только его длиной и направлением. На рис. 21.1 показаны два представления одного и того же вектора `a` = PQ = `b` = RS. Следовательно, при переносе вектор не изменяется.

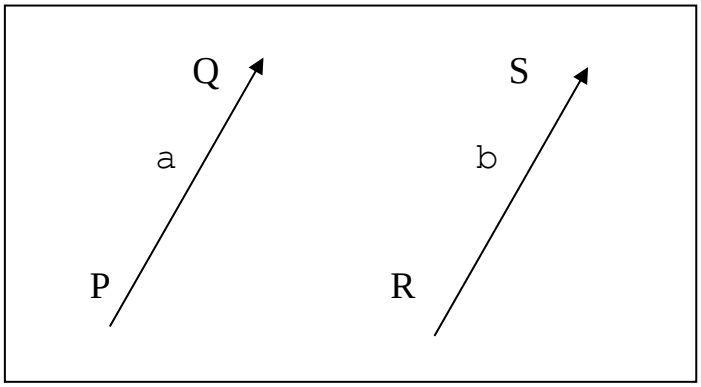


Рис.21.1 – Равные векторы

На рис. 21.2 начальная точка вектора `b` совпадает с конечной точкой вектора `a`.

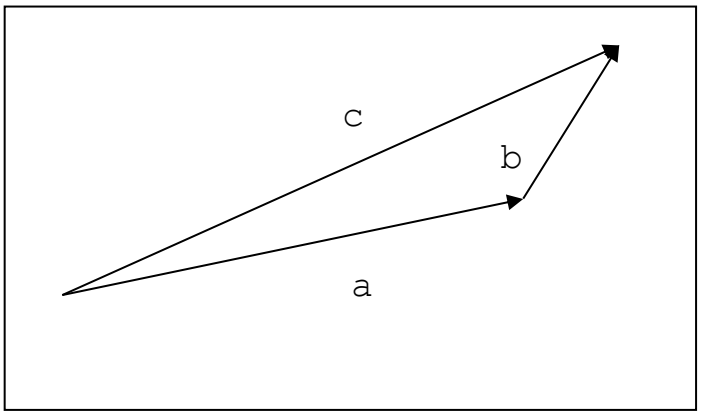


Рис.21.2 – Сложение векторов

Тогда *сумма векторов* `a` и `b` определяется как вектор `c`, проведённый из начальной точки вектора `a` в конечную точку вектора `b`, поэтому можно записать: `c = a + b`.

Длина вектора a обозначается | a | и равна расстоянию между его начальной и конечной точками. Вектор с нулевой длиной называется *нулевым* вектором и обозначается 0. Обозначение – a применяется для вектора, имеющего длину | a |, направление которого *обратно* направлению вектора a. Для любого вектора a и вещественного числа c вектор ca имеет длину | c || a |. Если a = 0 или c = 0, то ca = 0, в противном случае вектор ca совпадает по управлению с вектором a, если c > 0, или имеет противоположное направление, если c < 0. Для любых векторов u, v, w и вещественных чисел c, k будем иметь:

```
u + v = v + u
(u + v) + w = u + (v + w)
u + 0 = u
u + (-u) = 0
c(u + v) = cu + cv
(c+k)u = cu+ku
c(ku) = (ck)u
1u = u
0u = 0
```

На рис. 21.3 показаны три единичных вектора i, j, k.

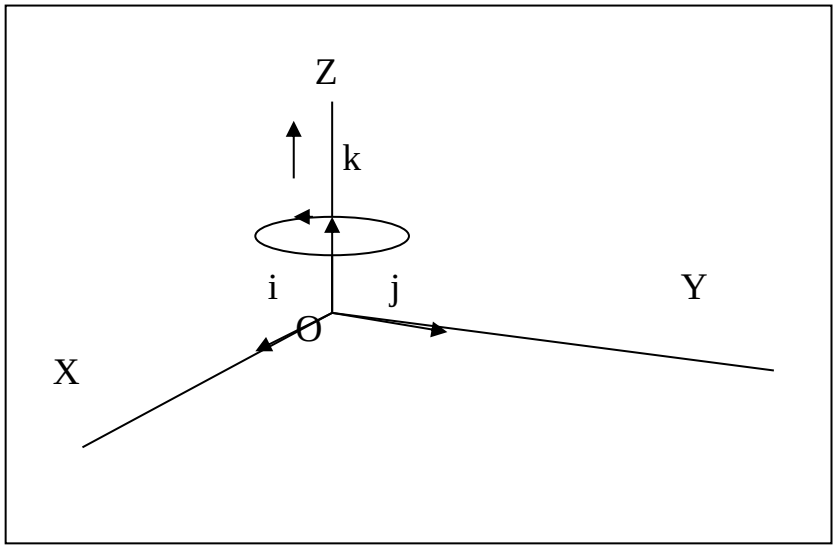


Рис. 21.3 – Правая координатная система

Они взаимно перпендикулярны, имеют длину, равную единице, и определяют направления координатных осей. Можно сказать, что векторы `i,j,k` образуют тройку ортогональных единичных векторов. Координатная система является *правой.* Это означает, что если поворот от вектора `i` к вектору `j` на 90° соответствует повороту винта с правой резьбой, то вектор `k` совпадает с направлением перемещения винта.

Точка O в начале координатной системы часто является начальной точкой всех векторов. Любой вектор v может быть записан как линейная комбинация единичных векторов `i,j,k`:

$$v = xi + yj + zk$$

Вещественные числа `x,y,z` определяют координаты конечной точки P вектора `v` = OP. Этот вектор `v` может быть обозначен либо в виде строки:

$$v = [x\ y\ z],$$

либо в виде столбца:

$$v = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Числа `x,y,z` иногда называют *элементами* вектора `v`.

21.2 Скалярное произведение

Скалярное произведение двух векторов `a` и `b` обозначается через $a \bullet b$ и определяется как:

$$\begin{aligned} &a \bullet b = |a||b|\cos(\gamma), \text{ если } a \neq 0 \text{ и } b \neq 0 \\ &a \bullet b = 0, \text{ если } a = 0 \text{ или } b = 0 \end{aligned} \qquad (21.1)$$

где γ – угол между a и b.

Применяя это правило к единичным векторам `i,j,k,` находим:

$$\begin{aligned} &i \bullet i = j \bullet j = k \bullet k = 1 \\ &i \bullet j = j \bullet i = j \bullet k = k \bullet j = i \bullet k = k \bullet i = 0 \end{aligned} \qquad (21.2)$$

Приравнивая $a = b$ в уравнении (21.1) получаем: $a \bullet a = |a|^2$, так что $|a| = \sqrt{|a \bullet a|}$.

Важные свойства скалярного произведения:

```
c(ku•v) = ck(u•v)
(cu + kv)•w = cu•w + kv•w
u•v = v•u
u•u = 0 только если u = 0
```

Скалярное произведение векторов $u = [u_1\ u_2\ u_3]$ и $v = [v_1\ v_2\ v_3]$ может быть вычислено как:

$$u \bullet v = u_1v_1 + u_2v_2 + u_3v_3$$

Это легко доказать, переписав правую часть уравнения

$$u \bullet v = (u_1i + u_2j + u_3k) \bullet (v_1i + v_2j + v_3k)$$

в виде суммы девяти скалярных произведений и проанализировав их на основании выражения (21.2).

21.3 Детерминанты

Перед описанием векторного произведения обратим внимание на детерминанты. Чтобы решить следующую систему двух линейных уравнений

$$\begin{cases} a_1x + b_1y = c_1 \\ a_2x + b_2y = c_2 \end{cases} \qquad (21.3)$$

необходимо умножить первое уравнение на коэффициент b_2, а второе – на коэффициент $-b_2$ и сложить. Тогда получим

$$(a_1b_2 - a_2b_1)x = b_2c_1 - b_1c_2$$

После этого можно первое уравнение умножить на $-a_2$, а второе – на a_1 и также сложить. В результате получим

$$(a_1b_2 - a_2b_1)y = a_1c_2 - a_2c_1$$

Если $a_1b_2 - a_2b_1$ не равно нулю, то можно выполнить деление и найти

$$x = \frac{b_2c_1 - b_1c_2}{a_1b_2 - a_2b_1}, \quad y = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1} \tag{21.4}$$

Выражение в делителе можно записать в форме

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

В этом случае оно называется *детерминантом второго порядка*.

Следовательно,

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = a_1b_2 - a_2b_1$$

С помощью детерминантов уравнение (21.4) может быть записано в виде

$$x = \frac{D_1}{D}, \; y = \frac{D_2}{D}, \; D \neq 0,$$

где $D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$ $D_1 = \begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}$ $D_2 = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}$

Заметим, что D_i получается путём замены `i`-го столбца в D на правую часть системы уравнений (21.3) (`i` = 1 или 2). Такой способ решения системы линейных уравнений называется «правилом Крамера». Этот способ пригоден не только для системы двух уравнений (хотя с точки зрения затрат машинного времени он оказывается очень дорогим для больших систем).

Определим детерминант третьего порядка в виде уравнения

$$D = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = a_1 \begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & c_1 \\ b_3 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}$$

и детерминант четвёртого порядка

$$D=\begin{vmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \end{vmatrix} = a_1\begin{vmatrix} b_2 & c_2 & d_2 \\ b_3 & c_3 & d_3 \\ b_4 & c_4 & d_4 \end{vmatrix} - a_2\begin{vmatrix} b_1 & c_1 & d_1 \\ b_3 & c_3 & d_3 \\ b_4 & c_4 & d_4 \end{vmatrix} + a_3\begin{vmatrix} b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \\ b_4 & c_4 & d_4 \end{vmatrix} - a_4\begin{vmatrix} b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \\ b_3 & c_3 & d_3 \end{vmatrix}$$

и так далее.

Детерминанты имеют много интересных свойств, некоторые из них перечислены ниже.

1. Значение детерминанта не изменится, если строки записать в виде столбцов в том же порядке.

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix}$$

2. Если произвести взаимную замену двух строк (или двух столбцов), то значение детерминанта будет умножено на –1.

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = -\begin{vmatrix} b_1 & a_1 & c_1 \\ b_2 & a_2 & c_2 \\ b_3 & a_3 & c_3 \end{vmatrix}$$

3. Если любую строку (или столбец) умножить на коэффициент, то значение детерминанта также будет умножено на этот коэффициент.

$$\begin{vmatrix} ca_1 & cb_1 \\ a_2 & b_2 \end{vmatrix} = c\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

4. Если строка (или столбец) изменяется путём добавления соответствующих элементов другой строки (или столбца), умноженных на константу, то значение детерминанта не изменится.

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 + ka_1 & b_3 + kb_1 & c_3 + kc_1 \end{vmatrix} = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

5. Если строка (столбец) является линейной комбинацией некоторых других строк (столбцов), то значение детерминанта равно нулю.

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 3a_1 - 2a_2 & 3b_1 - 2b_2 & 3c_1 - 2c_2 \end{vmatrix} = 0$$

Существует много полезных применений детерминантов. Детерминантные уравнения, выражающие геометрические свойства, элегантны и легки для запоминания. Например, *уравнение для прямой линии в двухмерном пространстве, R_2*, проходящей через две точки $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$ может быть записано в виде

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0 \qquad (21.5)$$

Такая запись становится очевидной, если, во-первых, рассматривать уравнение (21.5) как специальное обозначение линейных уравнений по x и y и, следовательно, представляющих прямую линию в пространстве R_2, и, во-вторых, можно убедиться, что координаты обеих точек P_1 и P_2 удовлетворяют этому уравнению, поскольку при их подстановке в первую строку получим две одинаковые строки. Аналогично, плоскость в трёхмерном пространстве, R_3, проходящая через три точки $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$, $P_3(x_3, y_3, z_3)$, будет описываться уравнением

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \end{vmatrix} = 0$$

21.4 Векторное произведение

Векторное произведение двух векторов `a` и `b` обозначается через $a \times b$ и равно вектору `v`, который обладает следующими свойствами.

Если $a = cb$ для некоторого скаляра c, то v = 0.

В противном случае длина вектора v равна

$$|v| = |a||b|\sin(\gamma)$$

где γ – угол между векторами a и b, а направление вектора v перпендикулярно обоим векторам a и b и таково, что a, b, v, именно в таком порядке, образуют *правостороннюю тройку*. Последнее означает, что если вектор a поворачивается на угол $\gamma < 180°$ в направлении к вектору b, то вектор v имеет направление перемещения винта с правой резьбой, поворачиваемого в том же направлении. Из этого определения можно вывести следующие свойства векторного произведения:

```
(ka) x b = k(a x b) для любого вещественного числа k
a x (b + c) = a x b + a x c
a x b = - b x a
```

В общем случае a x (b x c) ≠ (a x b) x c. Используя правую ортогональную систему координат с единичными векторами i, j, k, будем иметь

$$i \times i = j \times j = k \times k = 0$$

$$i \times j = k,\ j \times k = i,\ k \times i = j$$

$$j \times i = -k, k \times j = -i,\ i \times k = -j$$

Используя эти значения векторных произведений в расширенной записи векторного произведения

```
a x b = (a1i + a2j + a3k) x (b1i + b2j + b3k)
```

получим

```
a x b = (a2b3 - a3b2)i + (a3b1 - a1b3)j + (a1b2 - a2b1)k
```

что может быть записано в виде

$$a \times b = \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} i + \begin{vmatrix} a_3 & a_1 \\ b_3 & b_1 \end{vmatrix} j + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} k$$

или в форме, более удобной для запоминания:

$$a \times b = \begin{vmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

Это скорее мнемоническая запись, чем реальный детерминант, поскольку элементами первой строки являются векторы, а не числа. Если через векторы a и b обозначены соседние стороны параллелограмма, как на рис. 21.4, то площадь этого параллелограмма равна длине вектора a x b. Это непосредственно следует из $|a \times b| = |a||b|\sin(\gamma)$

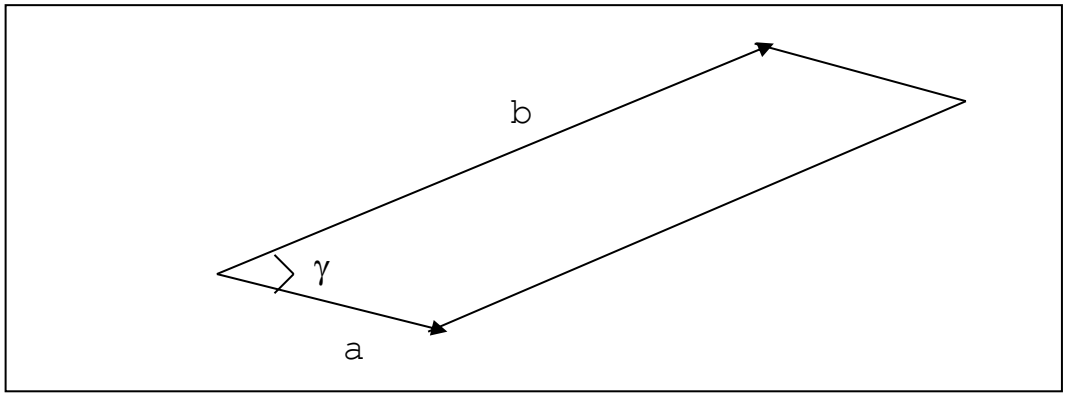


Рис.21.4 – Параллелограмм с площадью |a x b|

На рис. 21.5 векторы a и b лежат в плоскости, проходящей через оси *x* и *y*.

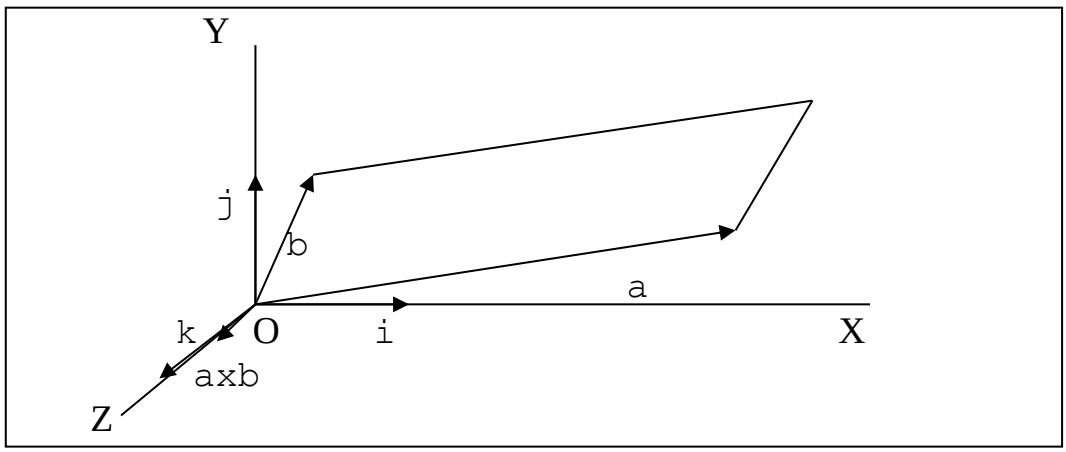


Рис.21.5 – Векторное произведение k=a x b

Предположим, что ось *z* выходит из листа бумаги к читателю и соответствует правой координатной системе. Тогда для трёхмерного пространства

$$a=[a_1\ a_2\ 0],\ b=[b_1\ b_2\ 0]$$

$$a\times b=\begin{vmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}=\begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} k$$

Таким образом, вектор $a\times b$ будет иметь то же направление, что и вектор k, но только в том случае, если детерминант

$$D=\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

имеет положительный знак. Это налагает условие, что вектор a, поворачиваемый по направлению к вектору b на угол меньше 180°, вращается в положительном направлении (против часовой стрелки) тогда, и только тогда, когда D > 0. Этот способ будем ниже использовать для определения, обходятся ли вершины треугольника A, B, C в направлении против часовой стрелки при их перечислении именно в этом порядке. На рис. 21.6 имеем

$$u=[u_1\ u_2]=AB,\ v=[v_1\ v_2]=AC$$

$$D=\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix}=\begin{vmatrix} x_A & y_A & 1 \\ x_B-x_A & y_B-y_A & 0 \\ x_C-x_A & y_C-y_A & 0 \end{vmatrix}=\begin{vmatrix} x_B-x_A & y_B-y_A \\ x_C-x_A & y_C-y_A \end{vmatrix}=\begin{vmatrix} u_1 & u_1 \\ v_2 & v_2 \end{vmatrix}$$

Вершины A, B, C, именно в этом порядке, обходятся в направлении против часовой стрелки, если, и только если, вектор u поворачивается в сторону направления вектора v на угол $\gamma < 180°$ против часовой стрелки.

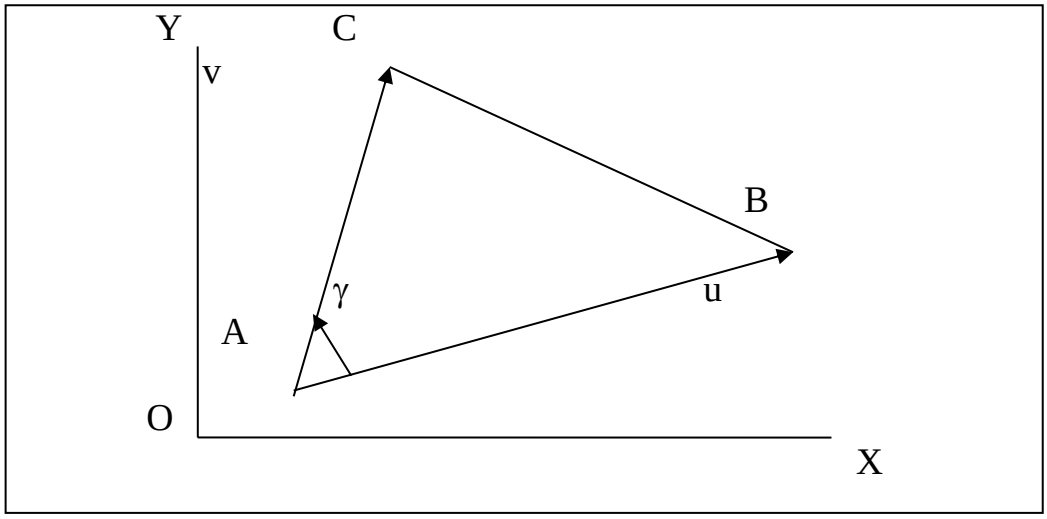


Рис. 21.6 – Точки A, B, C обходятся против часовой стрелки

Это означает, что направление обхода точек A, B, C может быть установлено на основе анализа детерминанта

$$D = \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix}$$

следующим образом:

$D > 0$ – точки A, B, C обходятся против часовой стрелки;

$D < 0$ – точки A, B, C обходятся по часовой стрелке;

$D = 0$ – точки A, B, C лежат на одной прямой.

21.5 Декомпозиция полигонов на треугольники

В дальнейшем будут формироваться изображения трёхмерных объектов, границы поверхностей которых могут быть полигонами. Это не очень серьёзное ограничение, поскольку кривые поверхности могут аппроксимироваться большим числом полигонов, точно так же, как линии аппроксимируются последовательностью отрезков прямых линий. Обработка произвольных полигонов может привести к очень сложным

ситуациям, особенно если нужно различать *видимые и невидимые* части отрезков прямых. На рис. 21.7 показан пример такой ситуации.

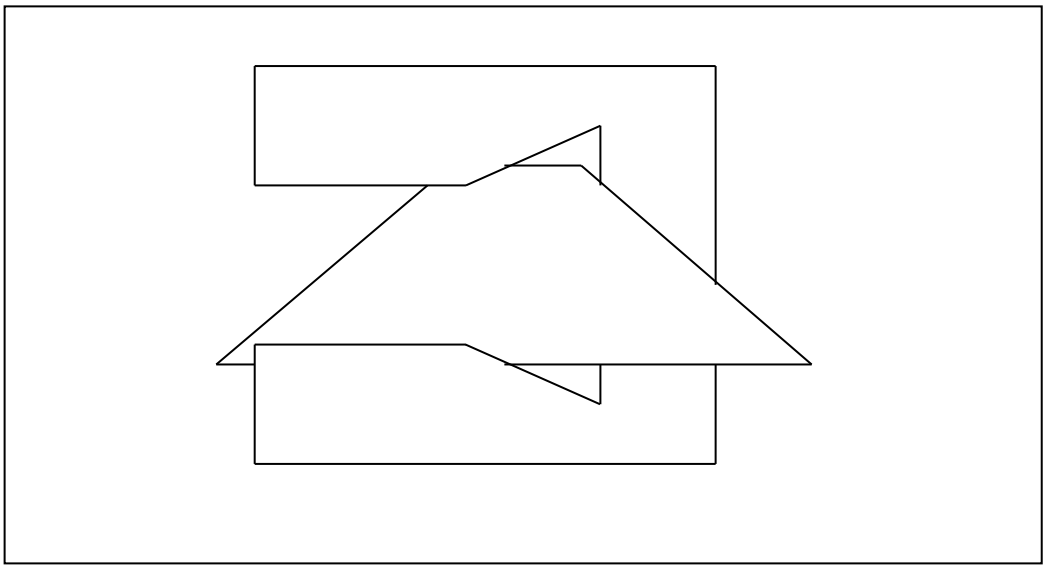


Рис.21.7 – Два полигона частично перекрывают друг друга

Если внутренние углы при всех вершинах полигона меньше 180°, то такой полигон называется *выпуклым***.** На рис. 21.8 (б) внутренний угол при вершине *P* больше 180°. Такую вершину будем называть *невыпуклой***.** Все другие вершины на рис. 21.8 – выпуклые. Если полигон имеет хотя бы одну невыпуклую вершину, то весь такой полигон будем называть невыпуклым.

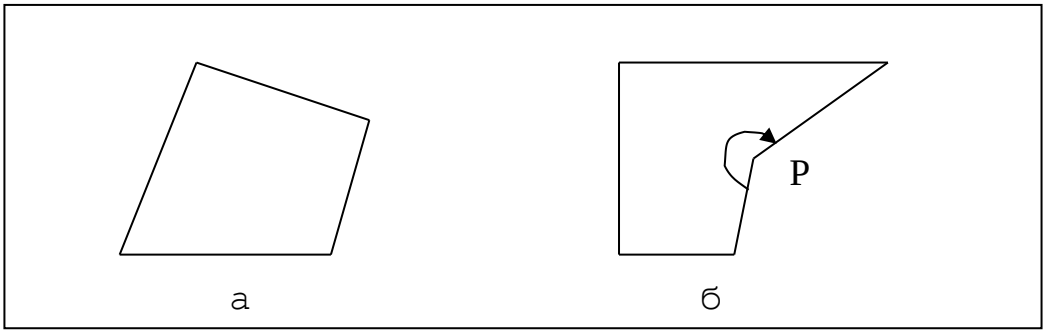


Рис.21.8 – Выпуклый и невыпуклый полигон:
а – выпуклый полигон; б – невыпуклый полигон

Если A и B – две точки на границе выпуклого полигона, то и весь отрезок AB будет принадлежать полигону. Для невыпуклых полигонов это условие может не соблюдаться.

Невыпуклость полигонов является источником сложностей, это же касается и переменного числа вершин полигонов. По этой причине уделим большое внимание треугольникам. Очевидно, что треугольники всегда имеют фиксированное число вершин, и они обязательно выпуклые. Особый интерес к ним выявляется в связи с рассмотрением произвольных полигонов, поскольку любой полигон может быть разбит на конечное число треугольников

Операция разбивки выпуклого полигона на треугольники чрезвычайно проста, как это видно из рис. 21.9.

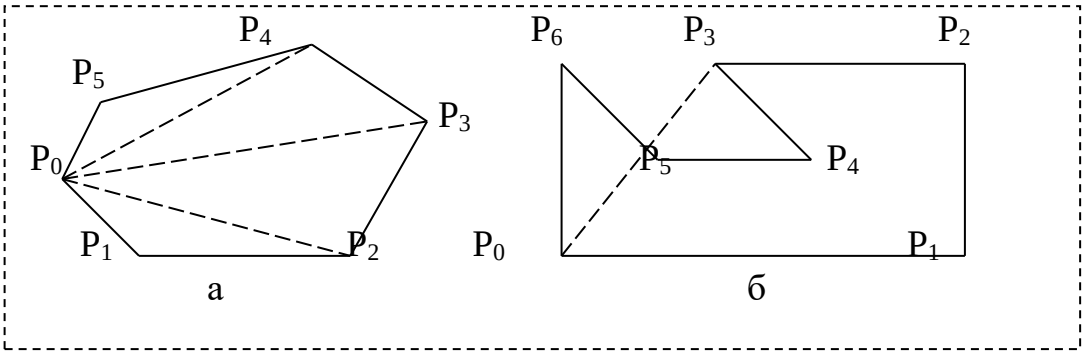


Рис. 21.9 – Разбивка выпусклого полигона на треугольники:
а – диагонали внутри полигона; б – диагональ P_0P_3 использовать нельзя

Если вершины полигона пронумеровать последовательно $P_0, P_1, ..., P_{n-1}$, а затем вычертить диагонали $P_0P_2, P_0P_3, ..., P_0P_{n-2}$ то этого будет достаточно. В невыпуклом полигоне, как на рис. 21.9 (б), этот простой способ работать не будет, поскольку некоторые из диагоналей $P_0P_2, P_0P_3, ..., P_0P_{n-2}$ могут выходить за пределы полигона.

Составим теперь программу, которая будет считывать координаты вершин полигона и выполнит разбиение полигона на треугольники. Требуется, чтобы вершины были указаны обязательно в порядке обхода против часовой стрелки. Например, у полигона на рис. 21.9(б) верной окажется последовательность $P_4P_5P_6P_0P_1P_2P_3$, а последовательность $P_6P_5P_4P_3P_2P_1P_0$ будет непригодной. Для полигона с n вершинами сначала указывается количество вершин n, затем последовательно перечисляются n

пар координат всех вершин в порядке обхода полигона против часовой стрелки. В результате будет получен чертёж полигона с вычерченными диагоналями, полностью разбивающими весь полигон на треугольники. Перед вычерчиванием диагоналей необходимо удостовериться, что все диагонали лежат полностью внутри полигона.

Предположим, что P_{i-1}, P_i, P_{i+1} обозначают три соседние вершины, причём будем считать, что $P_{-1} = P_{n-1}$ и $P_n = P_0$, чтобы можно было рассматривать случаи, когда $i = 0$ и $i = n - 1$. Напомним, что вершины должны быть перечислены в порядке обхода против часовой стрелки. В этом случае P_i будет выпуклой вершиной тогда, и только тогда, когда три вершины P_{i-1}, P_i, P_{i+1} именно в этом порядке будут обходиться в направлении против часовой стрелки.

В качестве контрпримера рассмотрим рис. 21.10, в котором обход тройки $P_1P_2P_3$ выполняется по часовой стрелке.

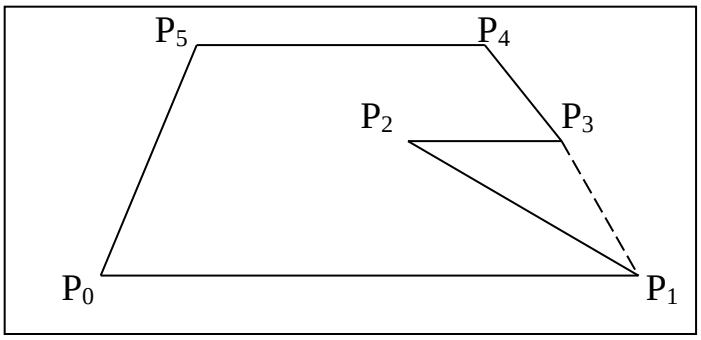


Рис.21.10 – Диагональ P_1P_3 вне полигона

Вершина P_2 является невыпуклой, и диагональ P_1P_3 лежит вне полигона. Таким образом, диагональ $P_{i-1}P_{i+1}$ может быть только кандидатом для наших целей, если три вершины $P_{i-1}(x_{i-1}, y_{i-1})$, $P_i(x_i, y_i)$, $P_{i+1}(x_{i+1}, y_{i+1})$, именно в этом порядке, обходятся в направлении против часовой стрелки, то есть если

$$D = \begin{vmatrix} x_{i-1} & y_{i-1} & 1 \\ x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \end{vmatrix}$$

Это условие является необходимым, но, к сожалению, недостаточным, как это показано на рис. 21.11.

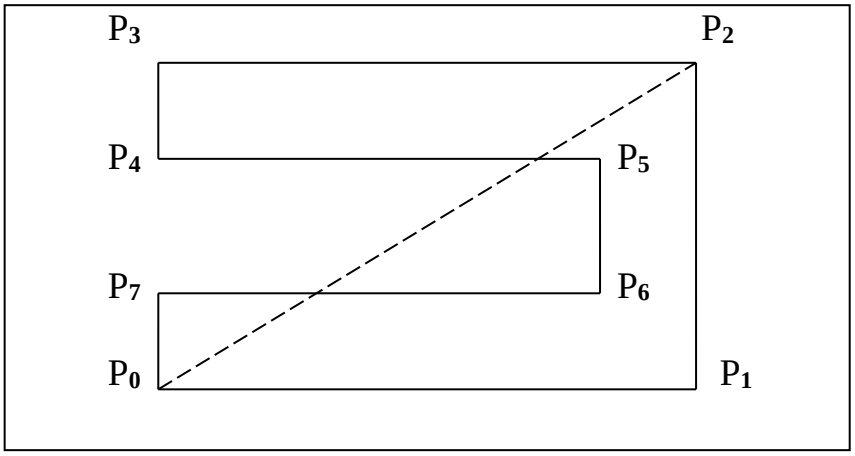


Рис. 21.11 – Диагональ P_0P_1 частично находится вне полигона

Здесь точки P_0, P_1, P_2 обходятся именно в таком порядке, в направлении против часовой стрелки, но отрезок P_0P_2 нельзя использовать для деления полигона на треугольники. Такой ситуации можно избежать, ели принимать во внимание также длину диагоналей. Будем выбирать наикратчайшую диагональ $P_{i-1}P_{i+2}$, которую может меть выпуклая вершина P_i между точками P_{i-1} и P_{i+1}. Эта диагональ используется для отсечения треугольника $P_{i-1}P_iP_{i+1}$. Затем таким же образом проверяется оставшийся полигон

$$P_0, P_1, \ldots, P_{i-1}, P_{i+1}, \ldots, P_{n-1}$$

и так далее. Технически это реализуется введением целочисленного массива $v_0, \ldots, v_{m-1}$, содержащего номера вершин оставшегося полигона. Вначале задаём $m = n$ и $v_i = i$ ($i = 0, 1, \ldots, n-1$). Каждый раз при отсечении треугольника число m уменьшается на единицу. Если подобная программа предназначается для практического применения, то желательно выполнить некоторое число испытаний на допустимость входных данных. Программа, которая надёжно отвергает любой непригодный набор данных, может быть названа устойчивой.

В нашей ситуации необходимо провести испытание, действительно ли заданный набор координат точек

$$n \ x_0 \, y_0 \ x_1 \, y_1 \ \ldots \ x_{n-1} \, y_{n-1}$$

вообще описывает полигон. Например, совершенно непригодна последовательность

4 1 1 2 2 2 1 1 2

поскольку обход точек в заданном порядке приведёт к ситуации, оказанной на рис. 21.12, но в качестве полигона такую фигуру принять нельзя.

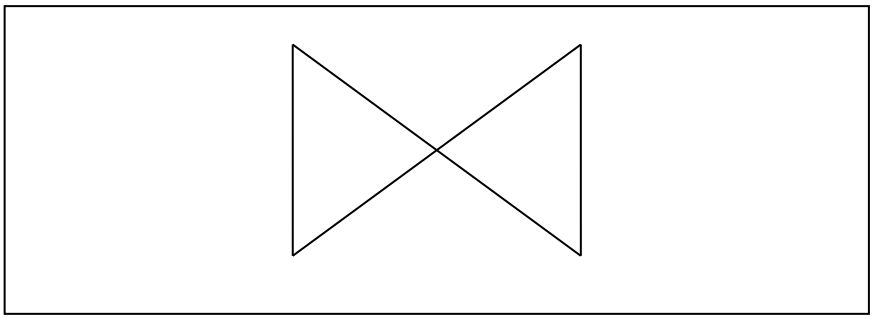


Рис.21.12 – Результат недопустимой последовательности

Из других проверок следует обратить внимание на:

– максимальное количество точек n, например, n <= 500;

– минимальное и максимальное значения координат;

– ориентацию обхода точек в направлении против часовой стрелки.

Несмотря на их очевидную важность, большинство проверок здесь опущено, и они оставлены для упражнений. Диагонали представлены в виде штриховых линий вместо сплошных. Штриховая линия не должна начинаться или кончаться пробелом, в начале и в конце штрихи должны быть полной длины, как это показано для отрезка PQ на рис. 21.13.

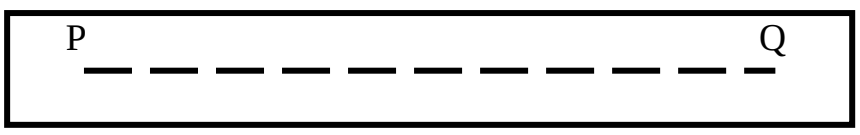


Рис.21.13 – Штриховая линия

```
/* PolyTria: Разбиение полигона на треугольники */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace PolyTria
{
  public partial class Form1 : Form
  {
    const int NMAX = 500;
    const double BIG = 1.0e30;
    Graphics dc;  Pen p;
    int   n; int[] v;
    double[] x, y;
    public Form1()
    {
      InitializeComponent();
      dc = pictureBox1.CreateGraphics();
      p = new Pen(Brushes.Black, 2);
      x = new double[NMAX];
      y = new double[NMAX];
      v = new int[NMAX];
    }
  /* Метод преобразования вещественной координаты X в целую */
    private int IX(double x)
    {
      double xx = x * (pictureBox1.Size.Width / 10.0) + 0.5;
      return (int)xx;
    }
  /* Метод преобразования вещественной координаты Y в целую */
    private int IY(double y)
    {
      double yy = pictureBox1.Size.Height - y *
                  (pictureBox1.Size.Height / 7.0) + 0.5;
      return (int)yy;
    }
  /* Функция вычерчивания линии (экран 10x7 условн. единиц) */
    private void Draw(double x1, double y1, double x2, double y2)
    {
      Point point1 = new Point(IX(x1), IY(y1));
      Point point2 = new Point(IX(x2), IY(y2));
      dc.DrawLine(p, point1, point2);
    }
    // Функция вычисления длины диагонали
     private bool counter_clock(int h, int i, int j, ref
                                              double pdist)
    {
      double xh = x[v[h]], xi = x[v[i]], xj = x[v[j]],
```

```
            yh = y[v[h]], yi = y[v[i]], yj = y[v[j]],
            x_hi, y_hi, x_hj, y_hj, Determ;
      x_hi = xi - xh; y_hi = yi - yh;
      x_hj = xj - xh; y_hj = yj - yh;
      pdist = x_hj * x_hj + y_hj * y_hj;
      Determ = x_hi * y_hj - x_hj * y_hi;
      return (Determ > 1e-6);
    }
/* Функция рисования полигона */
    private void draw_polygon()
    {
      int i;  double xold, yold;
      xold=x[n - 1]; yold=y[n - 1];
      for (i = 0; i < n; i++)
      {
        Draw(xold, yold, x[i], y[i]);
        xold = x[i]; yold = y[i];
      }
    }
/* Функция чтения информации из файла Polygon.dat */
    private void read_File()
    {
      StreamReader reader = new StreamReader("Polygon.dat");
      /* Чтение из файла количества точек*/
      n = Convert.ToInt32(reader.ReadLine());
      /* Чтение координат точек точек*/
      for (int i = 0; i < n; i++)
      {
        string[] xy = reader.ReadLine().Split(' ');
        x[i] = Convert.ToDouble(xy[0]);
        y[i] = Convert.ToDouble(xy[1]);
      }
      reader.Close();
    }
/* Главная функция разбиения полигона на треугольники */
    private void poly_Tria()
    {
      int i, h, j, m, k, imin = 0;
      double diag = 0, min_diag;
      /* Заполнение массива v номерами вершин */
      for (i = 0; i < n; i++) { v[i] = i; }
      /* Отрисовка полигона */
      p.DashStyle = System.Drawing.Drawing2D.DashStyle.Solid;
      draw_polygon();
      p.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
      m = n;
      while (m > 3){
        min_diag = BIG;
        for (i = 0; i < m; i++)
        {
        /* h - предыдущая вершина, i - текущая, j - следующая */
          if (i == 0) h = m - 1; else h = i - 1;
          if (i == m - 1) j = 0; else j = i + 1;
```

```
            /* Запоминаем самую короткую диагональ */
            if (counter_clock(h, i, j, ref diag) && (diag < min_diag))
            { min_diag = diag; imin = i; }
          }
          i = imin;
          if (i == 0) h = m - 1; else h = i - 1;
          if (i == m - 1) j = 0; else j = i + 1;
          if (min_diag == BIG)
          {
            var result = MessageBox.Show("Неправильное
            направление обхода!", "Ошибка!",MessageBoxButtons.OK,
            MessageBoxIcon.Error);
            Application.Exit();
          }
          /* Вывод шриховой линии между вершинами h и j */
          Draw(x[v[h]], y[v[h]], x[v[j]], y[v[j]]);
          /* Уменьшение количества вершин */
          m--;
          /* Исключаем из последовательности вершин вершину i */
          for (k = i; k < m; k++) v[k] = v[k + 1];
        }
      }
    /* Чтение информации о полигоне из файла и его разбиение */
      private void button1_Click(object sender, EventArgs e)
      {
    /* Вызов функции получения информации о полигоне */
       read_File();
    /* Вызов функции разбиения полигона на треугольники */
       poly_Tria();
      }
    }
}
```

Следующая входная последовательность в файле Polygon.dat

```
12
1 1
6 1
6 4
4 4
4 3
5 3
5 2
2 2
2 3
3 3
3 4
1 4
```

приведёт к выводу изображения, показанного на рис. 21.14.

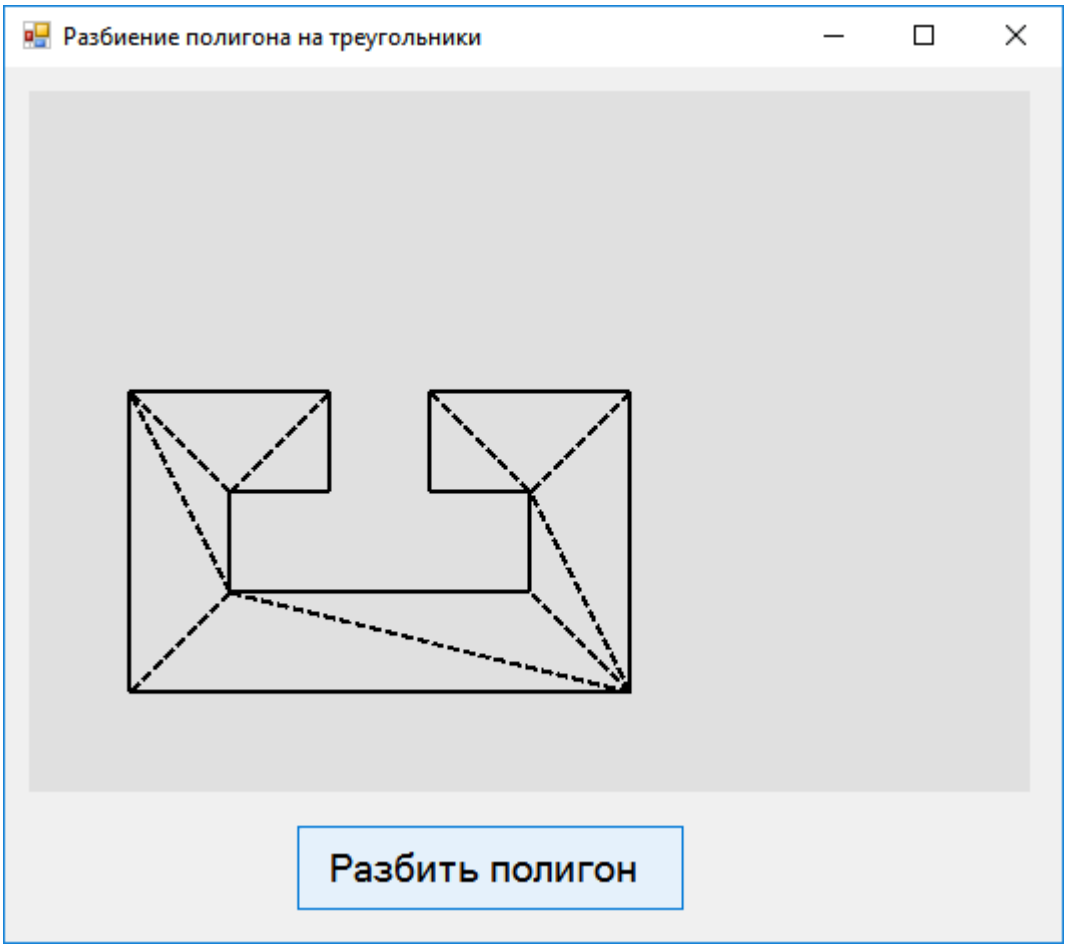


Рис. 21.14 – Результат работы программы PolyTria

Тема 22. Перенос и поворот в трёхмерном пространстве

22.1 Однородные координаты

Ранее использовалась запись [*x y* 1] для обозначения матрицы, состоящей только из одной строки, иногда называемой вектором-строкой. Такая запись может также рассматриваться как частный случай записи [*x y w*], где числа *x, y, w* называются *однородными координатами.* Эти три числа однородных координат применяются для обозначения точки в двухмерном пространстве. В проективной геометрии однородные координаты применялись задолго до возникновения и распространения машинной графики. На рис. 22.1 имеем ось *x* и ось *w*, так что точка задается парой координат *(x, w).*

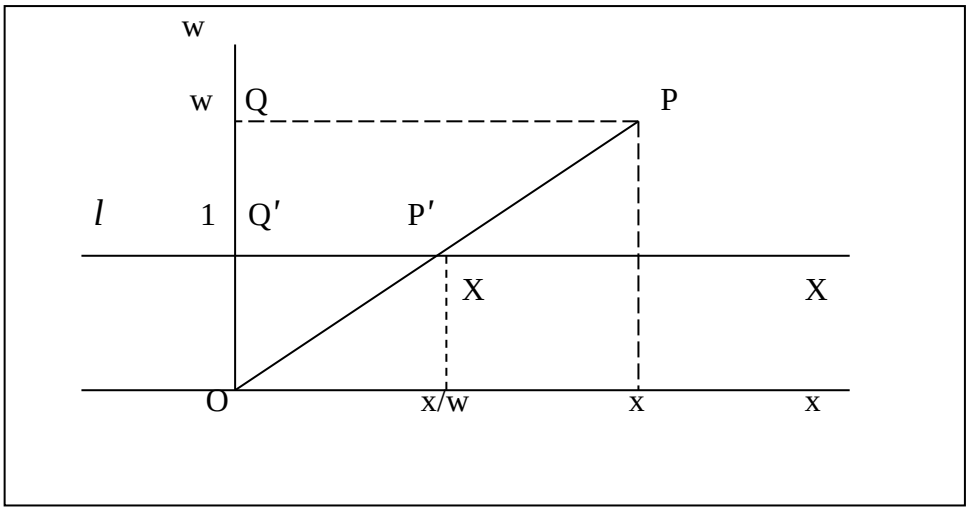


Рис.22.1 – Двумерная центральная проекция

Любая точка *P(x, y)*, не лежащая на оси *x*, имеет свою центральную проекцию *P(X, 1)*, определяемую как точка пересечения прямой линии OP с прямой линией *l*, описываемой уравнением *w* = 1. Точка начала координат является центром проекции. Отрезок прямой линии PO может рассматриваться как луч света из объекта *P* в глаз, расположенный в точке O. Для точек *Q(0, w)* и *Q' (0, 1)* получим два подобных треугольника OPQ и OP'Q', так что

$$X = \frac{X}{1} = \frac{P'Q'}{OQ'} = \frac{PQ}{OQ} = \frac{x}{w}$$

Все точки (x, w) со свойством $x = wX$ лежат на линии OP и имеют одну и ту же проекцию P'. Если будем интересоваться только проекциями на прямую линию l, а не фактическими значениями x и w, то имеет значение только их отношение. Вполне естественно использовать только одну координату X вместо пары (X, 1), если учитывать только точки на прямой линии l. Если же все-таки требуется использовать координатную пару, то подойдет любая пара чисел (x, w), удовлетворяющая условию $x/w = X$, если принять такое соглашение. В геометрическом смысле координатная пара (wX, w) любой точки P, отличной от O, на прямой линии OP' может служить обозначением точки P'. Это и реализуется в случае применения однородных координат.

В общем случае любая точка (X_1, X_2,..., X_n) в n-мерном пространстве записывается как точка (wX_1, wX_2 ..., wX_n, w) в ($n+1$)-мерном пространстве, где w — любое ненулевое вещественное число. Эта группа из $n+1$ чисел определяет однородные координаты исходной точки в n-мерном пространстве. Однородные координаты возникли из-за необходимости обратного преобразования из n-мерного пространства в (n+1)-мерное пространство. Точка (5, 7) в двухмерном пространстве, например, может быть записана в однородных координатах как (15, 21, 3) или как (500, 700, 100) и так далее. Хотя все операции относятся к двухмерному пространству, эти тройки можно принимать за точки в трехмерном пространстве. Очевидно, что обозначение (X, Y) представляет собой обычную запись точки в двухмерном пространстве для точки (X, Y, 1) в трехмерном пространстве. Точка P' является центральной проекцией для любой точки P(x, y, w) , если $x/w = X$ и $y/w = Y$. Как и ранее, точка начала координат O будет центром проекций, а все точки проецируются на плоскость w =1.

Для определения термина *однородные* координаты воспользуемся уравнением

$$aX + bY + c = 0 \tag{22.1}$$

описывающим прямую линию в двухмерном пространстве. Заменяя X и Y на x/w *и* y/w, получаем

$$a\,(x/w) + b\,(y/w) + c = 0$$

или

$$ax + by + cw = 0 \tag{22.2}$$

Уравнение (22.2) обычно принято называть однородным, поскольку оно имеет одинаковую структуру в терминах ax, by, cw. Отсюда числа x, y, w закономерно называть однородными координатами точки (X, Y). Если опять принять, что двухмерное пространство располагается в плоскости $w = 1$ в координатной системе xyw, то уравнение (22.2) описывает плоскость, проходящих через начало координат и заданную прямую линию. Если считать, что запись (x, y, w) используется как иная форма записи для (x/w, y/w), то необходимо будет потребовать, чтобы значение w не было равно нулю. Однако при этом однородные координаты едва ли имели какие-либо преимущества перед обычными координатами, и можно вообще выразить сомнение относительно наличия каких-то преимуществ. Например, рассмотрим систему из двух линейных уравнений, каждое из которых описывает прямую линию в двухмерном пространстве:

$$\begin{cases} a_1 X + b_1 Y + c_1 = 0 \\ a_2 X + b_2 Y + c_2 = 0 \end{cases} \tag{22.3}$$

Если две прямые линии параллельны, то они не пересекаются и не существует пары чисел (X, Y), удовлетворяющей системе (22.3). Таким образом, для нахождения общей точки для прямых линий необходимо применять правило с некоторым исключением, которое не совсем

элегантно. При замене координат X и Y на однородные координаты x, y, w ситуация несколько улучшится

$$\begin{cases} a_1 x + b_1 y + c_1 w = 0 \\ a_2 x + b_2 y + c_2 w = 0 \end{cases} \qquad (22.4)$$

Уравнения из системы (22.4) можно интерпретировать как плоскости, проходящие через точку начала координат O. Эта система имеет, по крайней мере, одно тривиальное решение $x = y = w = 0$.

Для геометрической интерпретации зададим для коэффициентов конкретные значения. Например, заменим систему (22.3) на систему

$$\begin{cases} 2X + 3Y - 6 = 0 \\ 4X + 6Y - 24 = 0 \end{cases} \qquad (22.5.а)$$

описывающую две параллельные прямые линии, изображенные на рис. 22.2.

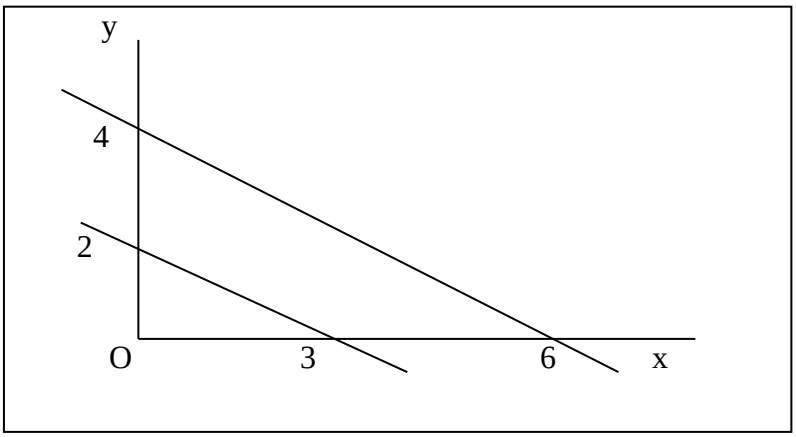


Рис. 22.2 – Параллельные прямые линии

Тогда систему уравнений (22.4) можно заменить на

$$\begin{cases} 2x + 3y - 6w = 0 \\ 4x + 6y - 24w = 0 \end{cases} \qquad (22.5.б)$$

Эта система эквивалентна системе

$$\begin{cases} 2x + 3y = 0 \\ w = 0 \end{cases}$$

Решение состоит из всех троек $(3k, -2k, 0)$, где k – любое вещественное число. В трехмерном пространстве эти точки образуют прямую линию, проходящую через точки O и (3, – 2, 0), причем эта прямая линия представляет собой линию пересечения двух плоскостей, заданных уравнениями (22.5). Возвращаясь к двухмерному пространству плоскости $w = 1$, напомним, что каждая точка (X, Y) ассоциируется с прямой линией (wX, wY, w) в трехмерном пространстве. Для ненулевых значений w эта ассоциация почти тривиальна.

Теперь станет понятной очень важная причина применения однородных координат. Для каждой прямой линии в двухмерном пространстве добавим один объект, называемый *бесконечно удаленной точкой***.** Эта бесконечно удаленная точка не может быть обозначена в обычной системе координат, а в системе однородных координат – может. Например, бесконечно удаленная точка на прямой линии, описываемой уравнением (22.5.а), записывается как (3, – 2, 0) или в виде любой тройки $(3k, -2k, 0)$ для ненулевого k. Поскольку эти тройки являются решением системы уравнений (22.5), то бесконечно удаленную точку можно считать точкой пересечения двух параллельных линий, изображенных на рис. 22.2. Считается, что бесконечно удаленная точка находится в двухмерном пространстве.

Как мы видели, каждая точка в двухмерном пространстве ассоциируется с прямой линией в трехмерном пространстве, поэтому выясним, с какой прямой линией ассоциируется бесконечно удаленная точка (3, – 2, 0). Поскольку эта линия должна проходить через точку начала координат O(0, 0, 0), то искомая линия должна быть линией, проходящей через точку O и точку (3, – 2, 0), то есть лежащей в плоскости $w = 0$.

Вполне резонно назвать точку (3, – 2, 0) бесконечно удаленной точкой, поскольку ее можно рассматривать как *предельную* точку $(3, -2, w)$ при w, стремящемся к нулю, а эта тройка в однородных координатах

эквивалентна точке (3/*w*, – 2/*w*, 1), которая при малых значениях *w* удалена очень далеко.

Введение бесконечно удаленной точки позволяет утверждать, что любые две различные прямые пересекаются в одной точке. Аналогично в проективной геометрии можно утверждать, что две любые различные плоскости имеют линию пересечения. Если плоскости параллельны, то все точки этой линии пересечения в однородных координатах записываются в виде (*x*, *y*, *z*, 0).

Вернемся к двухмерному пространству и покажем другие новые проявления возможностей, предоставляемых однородными координатами. В обычных, не однородных, координатах линейное преобразование в двухмерном пространстве может быть записано в виде

$$[X'\ Y'] = [X\ Y]\ A$$

$$A = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix}$$

Поскольку $[1\,0]\,A = [\,a_1\ \ a_2\,]$ и $[0\,1]\,A = [\,b_1\ \ b_2\,]$, то строки матрицы *A* отображают, соответственно, точки [1 0] и [0 1].

Вне зависимости от способа определения матрицы *A* начало координат О не изменяется, так что $[0\,0]\,A = [\,0\ \ 0\,]$, поэтому этим способом нельзя выразить операцию переноса. Однако в однородных координатах точка в двухмерном пространстве задается тройкой (*x*, *y*, *w*) и преобразование записывается в виде

$$[x'\ y'\ w'] = [x\ y\ w]\ A$$

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

Имеем $[1\,0\,0]\,A = [\,a_1\ a_2\ a_3\,]$

Так как точка [1 0 0] – бесконечно удаленная точка на оси *x*, то первая строка $[\,a_1\ a_2\ a_3\,]$ матрицы *A* представляет собой отображение этой бесконечно удаленной точки на оси *x*. Вторая строка матрицы $[\,b_1\ b_2\ b_3\,]$ будет отображением бесконечно удаленной точки на оси *y*. Поскольку $[\,0\,0\,1\,]\,A = [\,c_1\ c_2\ c_3\,]$, то можно считать, что третья строка матрицы $[\,c_1\ c_2\ c_3\,]$ является отображением точки начала координат [0 0 1].

Это означает, что однородные координаты позволяют выразить любые преобразования путем матричного умножения. Однако операция переноса – не единственное новое свойство, предоставляемое таким матричным умножением. С его помощью можно преобразовать параллельные прямые линии в пересекающиеся. Покажем это на примере преобразования полного прямоугольного квадранта в треугольник.

На рис. 22.3 изображен произвольный треугольник, вершины которого заданы в прямоугольной системе координат точками $A(a_1, a_2)$, $B(b_1, b_2)$ и $C(c_1, c_2)$. Добавим третью координату, равную единице, как формальное средство образования однородных координат. Исключением являются бесконечно удаленный точки (1, 0, 0) и (0, 1, 0) на координатных осях, у которых третья однородная координата равна нулю.

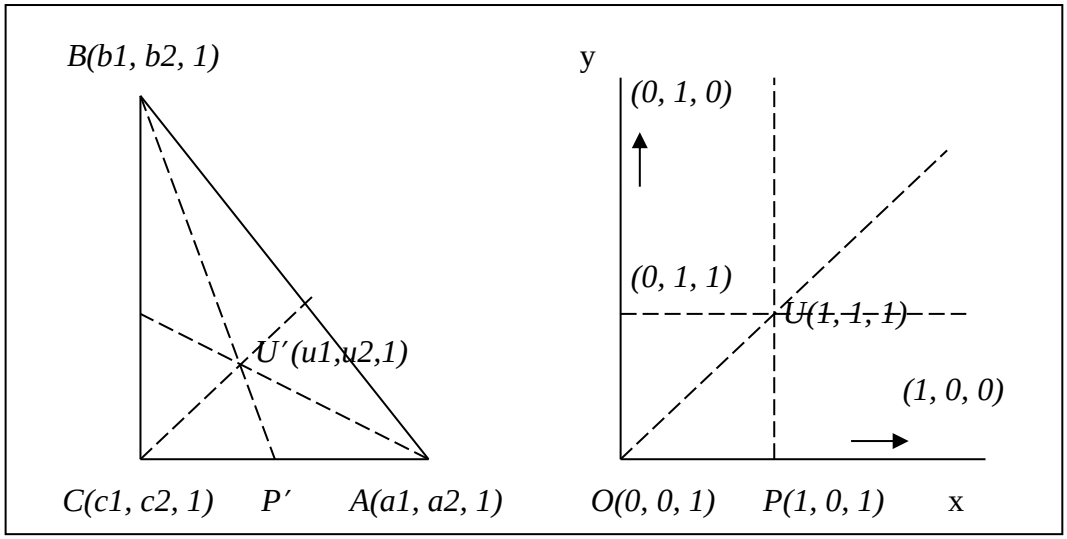


Рис.22.3 – Квадрант, преобразованный в треугольник

Образуем матрицу *M,* такую, что матричное умножение

$$[x'\ y'\ z'] = [x\ y\ z]\,M$$

отобразит точку O на точку C, точку (1, 0, 0) – на точку A и точку (0,1,0) – на точку B. Для любых ненулевых значений α, β, γ точки A, B и C будут точками требуемого изображения треугольника, если

$$M = \begin{bmatrix} \alpha a_1 & \alpha a_2 & \alpha \\ \beta b_1 & \beta b_2 & \beta \\ \gamma c_1 & \gamma c_2 & \gamma \end{bmatrix} \qquad (22.6)$$

Это легко проверить, поскольку имеем очевидное соотношение

$$[1\,0\,0]\,M = [\alpha a_1\ \alpha a_2\ \alpha]$$

правая часть которого является просто другим обозначением точки A на рис. 22.3. На первый взгляд кажется, что константы α, β, γ в уравнении (22.6) не нужны и могут быть установлены равными единице. Однако они нужны для того, чтобы можно было задать заранее установленное отображение U' для так называемой единичной точки U. Точка U' может быть выбрана в любом месте внутри треугольника. Поскольку точка U(1, 1, 1) отображается в точку U' (u_1, u_2, u_3), то имеем

$$[1\,1\,1]\,M = [u_1\ u_2\ 1]$$

– сокращенное обозначение следующей системы трех линейных уравнений относительно переменных α, β, γ:

$$a_1\alpha + b_1\beta + c_1\gamma = u_1$$
$$a_2\alpha + b_2\beta + c_2\gamma = u_2$$
$$\alpha + \beta + \gamma = 1$$

Система имеет единственное решение, которое следует из того факта, что тройки (a_1, a_2, a_3), (b_1, b_2, b_3) и (c_1, c_2, c_3) обозначают вершины

треугольника. Решение этой системы для α, β, γ и подстановка результата в уравнение (22.6) дает искомое значение матрицы *M*.

Прямые линии преобразуются в прямые линии, но их *параллелизм не сохраняется***.** Например, вертикальная линия, проходящая через точку *U* на рис. 22.3, превращается в прямую линию, проходящую через точки *B* и *U′*. Это дает геометрическое средство для нахождения проекции *P′* для точки *P*.

Имея вычисленное значение матрицы *M*, точку проекции *P′* можно найти аналитически как произведение

$$[101]M$$

Заметим, что все бесконечно удаленные точки отображаются на отрезок AB, а все бесконечно удаленный точки параллельных линий – на единственную точку на отрезке AB. Это дает основание рассматривать весь квадрант как плоскую проекцию, в которой треугольник ABC является картиной, а отрезок AB представляет собой линию горизонта.

22.2 Перенос и повороты в трехмерном пространстве

Если каждая точка *P*(*x, y, z*) отображается на точку *P′(x′, y′, z′)* в соответствии с уравнениями

$$\begin{cases} x' = x + a_1 \\ y' = y + a_2 \\ z' = z + a_3 \end{cases}$$

где a_1, a_2, a_3 – константы, то процесс называется *переносом в трехмерном пространстве*. Такой перенос может быть записан в матричной форме

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1]T$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a_1 & a_2 & a_3 & 1 \end{bmatrix} \qquad (22.7)$$

Первая, вторая и третья строки матрицы соответствуют отображениям бесконечно удаленных точек на координатных осях, а четвертая строка – отображению точки [0 0 0 1]. Последнее означает, что в однородных координатах точка $[a_1\ a_2\ a_3\ 1]$ является отображением точки начала координат O.

Поворот вокруг координатных осей может быть описан матрицей без использования однородных координат. Ради краткости так и будем поступать, обращаясь к однородным координатам только в тех случаях, когда они действительно необходимы. Будем использовать правую координатную систему, считая вращение вокруг оси положительным, если оно соответствует положительному направлению этой оси по правилу винта с правой резьбой. Это показано на рис. 22.4.

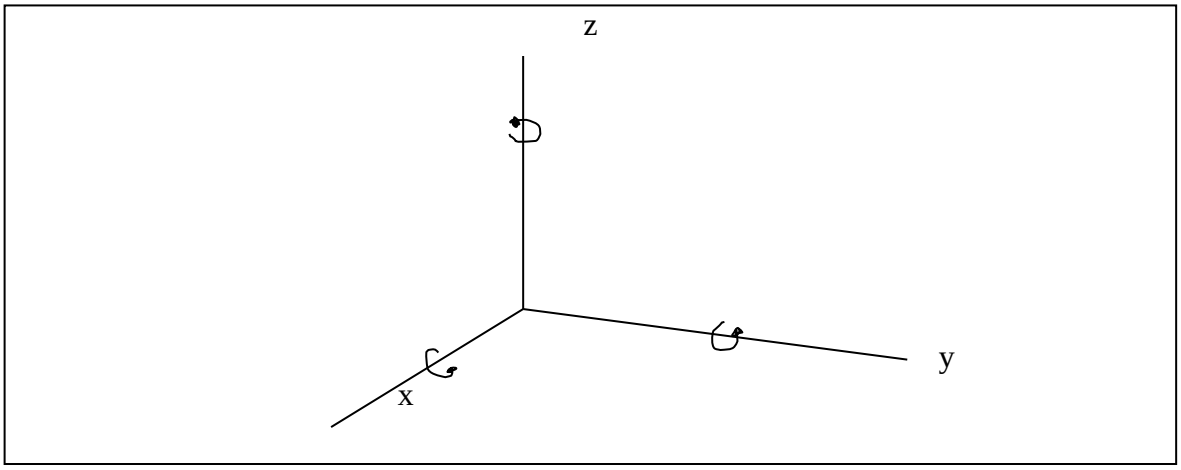


Рис.22.4 – Вращение в положительном направлении вокруг координатных осей

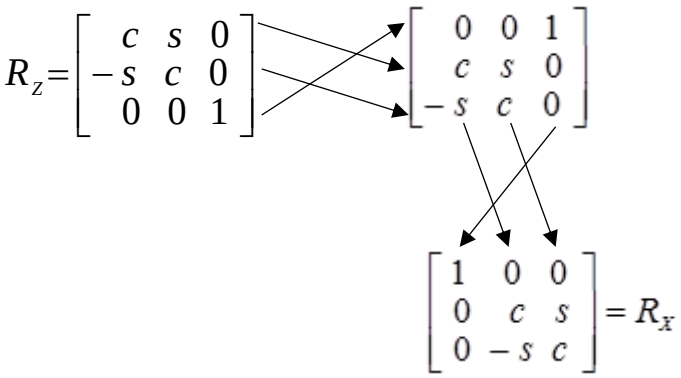
Рассмотрим поворот вокруг оси z на угол α и для сокращения обозначим $\cos(\alpha) = c$ и $\sin(\alpha) = s$. Тогда можно записать

$$[x'\ y'\ z'] = [x\ y\ z]R_z$$

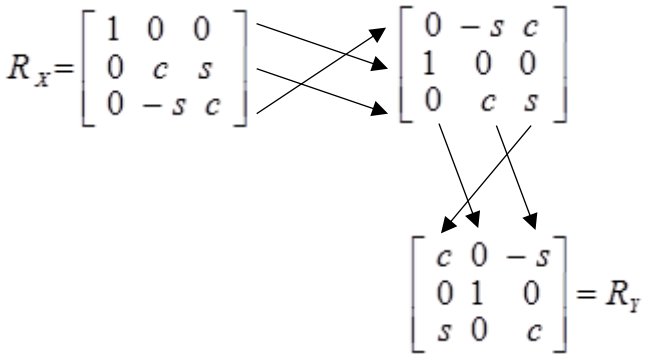
$$R_z = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Матрицу R_z можно использовать для получения матриц R_x и R_y, определяющих поворот вокруг соответствующих осей, чисто формальным образом, то есть без применения картинки. Это делается путем циклических перестановок, получаемых заменой каждой из букв *x, y, z* на последующую, считая, что за буквой *z* следует буква *x*.

Матрица R_z превратится в матрицу R_x циклическим переносом каждой строки на одну позицию и затем выполнением аналогичной операции для столбцов:



Так же матрица R_x преобразуется в «последующую» матрицу R_Y:



Суммируя сказанное, получим следующие матрицы:

$$R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{22.8}$$

$$R_Y = \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix} \tag{22.9}$$

$$R_Z = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{22.10}$$

Для поворота точки вокруг оси x на угол α матрица R_x используется следующим образом:

$$[x'\ y'\ z'] = [x\ y\ z]R_X$$

Матрицы R_y и R_z применяются аналогично.

Уравнения для преобразований могут интерпретироваться как изменения координат. Перенос точки на определенное расстояние вправо описывается теми же уравнениями, как перенос системы координат на такое же расстояние влево. На практике удобнее перемещать координатную систему вместо точки, но для этого требуется инвертирование матрицы. К счастью, инверсия матриц T, R_x, R_y, R_z (уравнения (22.7) – (22.10)) может быть записана сразу:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a_1 & -a_2 & -a_3 & 1 \end{bmatrix} \tag{22.11}$$

$$R_X^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{22.12}$$

$$R_Y^{-1} = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix} \tag{22.13}$$

$$R_Z^{-1} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{22.14}$$

Теперь можно найти матрицу R для поворота вокруг любой прямой линии, проходящей через точку начала координат О. Для определенности будем полагать, что поворот осуществляется вокруг вектора v, начало которого расположено в точке О. Тогда положительное направление вращения соответствует направлению вектора по правилу винта с правой резьбой. Как и ранее, поворот будем производить на угол α.

Если концевая точка вектора *v* задана в ортогональных координатах, то сначала вычислим его сферические координаты ρ, θ, φ (см. рис. 22.5).

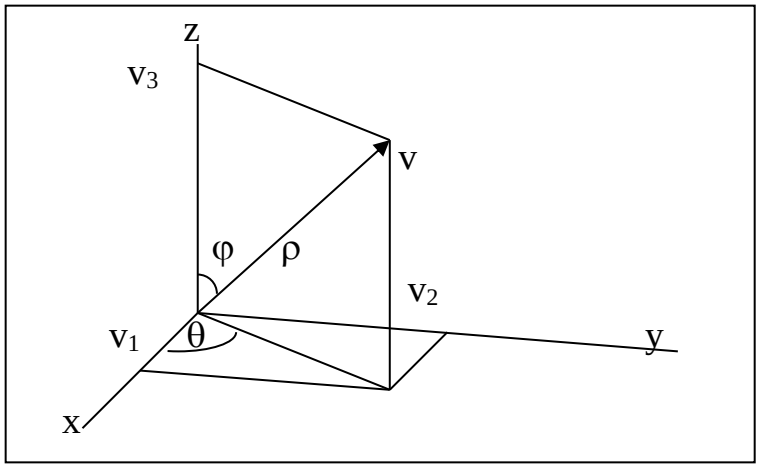


Рис.22.5 – Сферические координаты

$$\rho = |v| = \sqrt{v_1^2 + v_2^2 + v_3^2}$$

Если $\rho = 0$, то будем считать, что $\theta = \varphi = 0$. В противном случае

$$\theta = \begin{cases} \arctan(v_2 / v_1), \textit{если } v_1 > 0 \\ \pi + \arctan(v_2 / v_1), \textit{если } v_1 < 0 \\ \pi / 2, \textit{если } v_1 = 0 \textit{ и } v_2 \geq 0 \\ 3\pi / 2, \textit{если } v_1 = 0 \textit{ и } v_2 < 0 \end{cases}$$

$$\varphi = \arccos(v_3 / \rho)$$

Известно обратное вычисление

$$v_1 = \rho \sin(\varphi)\cos(\theta),\ v_2 = \rho \sin(\varphi)\sin(\theta),\ v_3 = \rho \cos(\varphi)$$

Теперь стратегия заключается в таком изменении системы координат, чтобы вектор *v* (ось вращения) совпадал с новым направлением положительной полуоси *z*. Начнем с поворота осей *x* и *y* вокруг оси *z* на угол θ. В соответствии с уравнением (22.14)

$$[x'\ y'\ z'] = [x\ y\ z] R_Z^{-1}$$

$$R_Z^{-1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ось x' имеет положительное направление вектора (v_1, v_2, 0). Теперь повернем оси x' и z' вокруг оси y' на угол φ до совпадения оси z'' с вектором v (см. рис. 22.5).

Обращаясь к уравнению (22.13), это условие запишем как

$$[x'' \; y'' \; z''] = [x' \; y' \; z'] R_Y^{-1}$$

$$R_Y^{-1} = \begin{bmatrix} \cos(\varphi) & 0 & \sin(\varphi) \\ 0 & 1 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) \end{bmatrix}$$

Фактический поворот вокруг вектора v на угол a теперь можно выполнить как поворот вокруг оси z''. Из уравнения (22.10) получим

$$[x''' \; y''' \; z'''] = [x'' \; y'' \; z''] R_V$$

$$R_V = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

К этому моменту достигнуто выполнение соотношения

$$[x''' \; y''' \; z'''] = [x \; y \; z] R_Z^{-1} \; R_Y^{-1} R_V$$

К сожалению, координаты x''', y''', z''' относятся к самой последней системе координат, тогда как их необходимо выразить в родной системе. Обозначим эти координаты в исходной системе, через x^*, y^*, z^*. Переход к исходной системе инвертированных матриц R_z^{-1} и R_y^{-1}, которые будут совпадать с матрицами R_z и R_y, в обратном порядке для преобразования точки x''', y''', z''':

$$[x^* \; y^* \; z^*] = [x''' \; y''' \; z'''] R_Y R_Z$$

Это означает, что полный поворот вокруг вектора v на угол a вычисляется по следующей формуле:

$$[x^* \; y^* \; z^*] = [x''' \; y''' \; z'''] = R_Z^{-1} R_Y^{-1} R_V R_Y R_Z$$

где

$$R_Z^{-1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_Y^{-1} = \begin{bmatrix} \cos(\varphi) & 0 & \sin(\varphi) \\ 0 & 1 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) \end{bmatrix}$$

$$R_V = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_Y = \begin{bmatrix} \cos(\varphi) & 0 & -\sin(\varphi) \\ 0 & 1 & 0 \\ \sin(\varphi) & 0 & \cos(\varphi) \end{bmatrix}$$

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

Для последующего применения запишем

$$R_Z^{-1} R_Y^{-1} R_V R_Y R_Z = R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \qquad (22.15)$$

До сих пор обсуждалось решение задачи о повороте относительно вектора, привязанного к точке начала системы координат O. Теперь нужно устранить это последнее ограничение и поставить задачу определения поворота относительно вектора, начало которого расположено в любой произвольной точке $A(a_1, a_2, a_3)$.

Для этого будем использовать вектор *v* для вычисления матрицы *R* в уравнении (22.15) таким же образом, как и ранее**.** Затем нужно выполнить три следующих шага:

1. Обращаясь к уравнению (22.11), выполним перенос из заданной точки в точку начала координат O, используя однородные координаты и следующую матрицу:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a_1 & -a_2 & -a_3 & 1 \end{bmatrix}$$

2. Теперь можем осуществить поворот относительно оси, проходящей через O, как и ранее, но матрицу R из уравнения (22.15) необходимо расширить тривиальным образом, чтобы можно было использовать однородные координаты

$$R^* = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Применить преобразование, обратное первому шагу, используя матрицу

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a_1 & a_2 & a_3 & 1 \end{bmatrix}$$

После этого матрица обобщенного поворота вычисляется как

$$R_{GEN} = T^{-1} R^* T$$

и ее можно использовать следующим образом:

$$[x^* \; y^* \; z^* \; 1] = [x \; y \; z \; 1] R_{GEN}$$

или

$$[x^* \; y^* \; z^* \; 1] = [x \; y \; z \; 1] T^{-1} (R_Z^{-1} R_Y^{-1} R_V R_Y R_Z)^* T$$

Тема 23. Перспективные изображения

23.1 Способы получения перспективных изображений

На рис. 23.1 представлено двухмерное изображение куба.

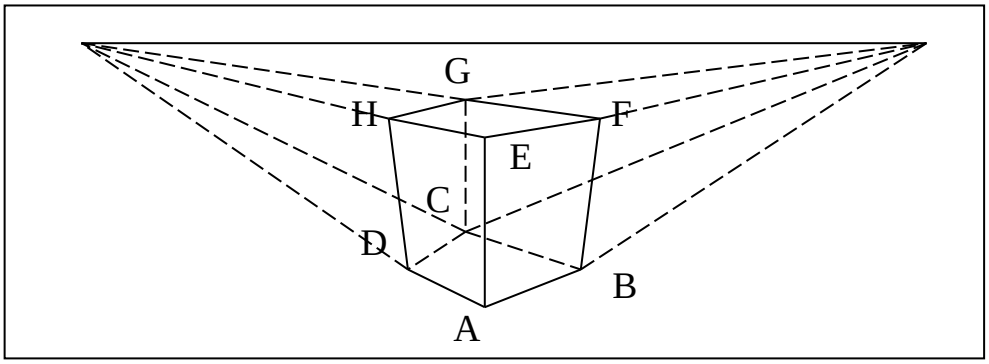


Рис.23.1 – Точки схода на горизонте

На этом изображении отрезки AB и AD не параллельны нижнему или верхнему краю листа бумаги, так что можно было бы утверждать, что они не горизонтальны. Однако они обозначают горизонтальные ребра куба ABCDEFGH в трехмерном пространстве, поэтому в принципе их можно называть горизонтальными. По этой же причине можно утверждать, что два отрезка AB и DC параллельны, неявно предполагая, что они находятся в трехмерном пространстве. По этой терминологии параллельные горизонтальные линии встречаются в так называемой *точке схода.*

Все точки схода лежат на одной прямой линии, которая называется *линией горизонта*. Линия горизонта и точка схода являются особенностью изображения и реально не существуют в трехмерном пространстве. В течение многих веков эта концепция использовалась художниками для получения изображений трехмерных объектов. Такие изображения обычно называются *перспективными*.

Изобретение фотографии предложило новый способ формирования перспективных изображений. Фотокамера является простейшей имитацией человеческого глаза. Далее слово *глаз* может быть заменено словом *камера,* если хотим подчеркнуть желание получить двухмерную твердую копию.

Очевидно, что картинка будет зависеть от положения глаза. Особо важное значение имеет расстояние между глазом и объектом, поскольку «*эффект перспективы*» будет обратно пропорционален этому расстоянию. Если глаз расположен очень близко от объекта, то получим сильный эффект перспективы, как на рис. 23.2 (а). Можно видеть, что продолжения изображений параллельных линий на картинке пересекаются. Если глаз расположен далеко от объекта (по сравнению с размером объекта), то параллельные линии объекта будут казаться параллельными и на картинке. Это показано на рис. 23.2 (б).

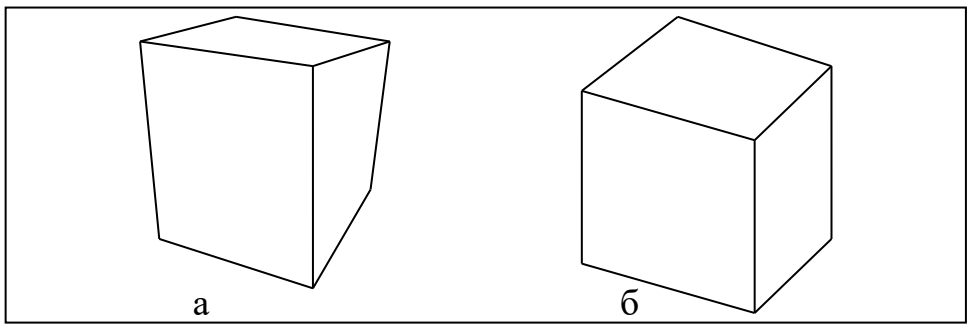


Рис.23.2 – Расположение глаза:
а – близко к объекту; б – далеко от объекта

Кроме классического и фотографического способов существует способ получения перспективных изображений на основе аналитической геометрии. Задается большое количество точек $P(x, y, z)$, принадлежащих объекту, для которых предстоит вычислить координаты точек изображения $P'(X, Y)$ на картинке. Нужно преобразовать координаты точки P из *мировых координат (x, y, z)* в *экранные координаты (X,Y)* ее центральной проекции P'. Экран расположен между объектом и глазом E. Для каждой точки P объекта прямая линия PE пересекает экран в точке P'. Это отображение удобно выполнять в два этапа. Первый этап называется видовым преобразованием – точка P остается на месте, но система мировых координат переходит в систему *видовых координат*. Второй этап – *перспективное преобразование.* Это преобразование точки P в P', объединенное с переходом из трехмерных видовых координат в систему двухмерных экранных координат:

Мировые координаты (x_w, y_w, z_w)

Видовое преобразование ↓

Видовые координаты (x_e, y_e, z_e)

Перспективное преобразование ↓

Экранные координаты (X, Y)

23.2 Видовое преобразование

Для выполнения видовых преобразований должны быть заданы точка наблюдения, совпадающая с глазом, и объект. Желательно, чтобы система мировых координат была правой. Будет удобно, если начало ее координат располагается где-то вблизи центра объекта, поскольку объект наблюдается в направлении от E к O. Предположим, что это условие выполняется. На практике это означает, что, возможно, потребуется некоторое преобразование координат, заключающееся в вычитании из исходных значений координат положения центральной точки объекта. Пусть точка наблюдения E будет задана в сферических координатах ρ, θ, φ по отношению к мировым координатам, то есть мировые координаты могут быть вычислены по формулам:

$$x_E = \rho \sin(\varphi)\cos(\theta), \quad y_E = \rho \sin(\varphi)\sin(\theta), \quad z_E = \rho \cos(\varphi) \qquad (23.1)$$

Сферические координаты схематически изображены на рис. 23.3.

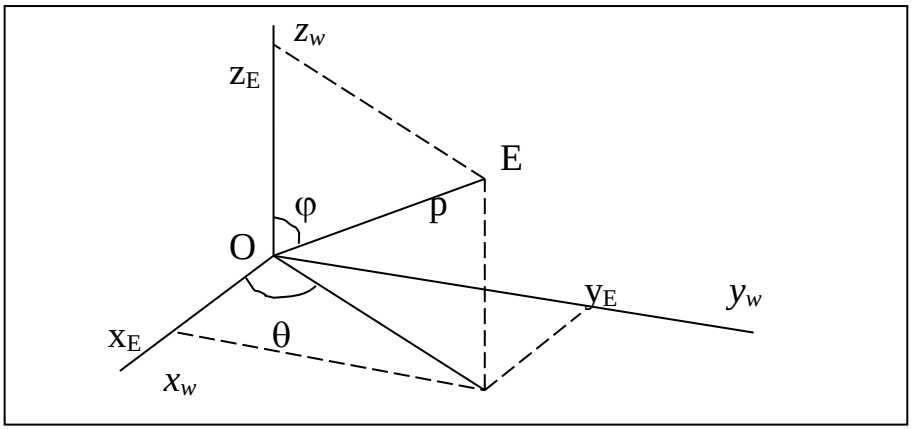


Рис.23.3 – Сферические координаты точки наблюдения E

Говорят, что вектор направления EO (равный – OE) определяет направление наблюдения. Из точки наблюдения E можно видеть точки объекта только внутри некоторого конуса, ось которого совпадает с линией EO, а вершина – с точкой E. Если заданы ортогональные координаты x_E, y_E, z_E точки E, то можно вычислить ее сферические координаты.

Конечной задачей будет вычисление экранных координат *X, Y,* для которых оси *x* и *y* лежат в плоскости экрана, расположенной между точками E и O и перпендикулярной направлению наблюдения EO. Начало системы видовых координат располагается в точке наблюдения E (рис.23.4).

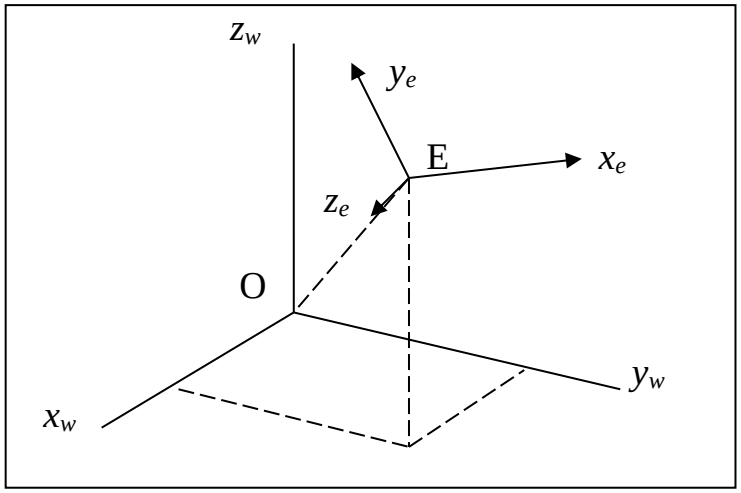


Рис. 23.4 – Система видовых координат

При направлении взгляда из E в O положительная полуось x_e направлена вправо, а положительная полуось y_e – вверх. Такое направление осей позволит нам впоследствии определить экранные оси в тех же направлениях. Направление оси z_e выбирается таким образом, что значения координат увеличиваются по мере удаления от точки наблюдения. Такие определения осей логичны и удобны, но их взаимное соответствие таково, что система видовых координат является левосторонней. Заметим, что система мировых координат всегда выбирается как правосторонняя.

Видовое преобразование может быть записано в форме

$$[x_e\ y_e\ z_e] = [x_w\ y_w\ z_w]V \qquad (23.2)$$

где *V* – матрица видового преобразования размерами 4 x 4.

Для нахождения матрицы V предположим, что преобразования отображения могут быть составлены из четырех элементарных преобразований, для которых легко написать свои матрицы преобразований. Матрица V получается путем перемножения этих четырех матриц. Фактически каждое из четырех преобразований изменяет координаты и, следовательно, определяется матрицей, обратной матрице, соответствующей преобразованию точки.

23.2.1 Перенос начала из O в E

Выполним такой перенос системы координат, при котором точка E становится новым началом координат. Матрица для такого изменения координат выглядит так:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_E & -y_E & -z_E & 1 \end{bmatrix} \qquad (23.3)$$

Новая система координат показана на рис. 23.5.

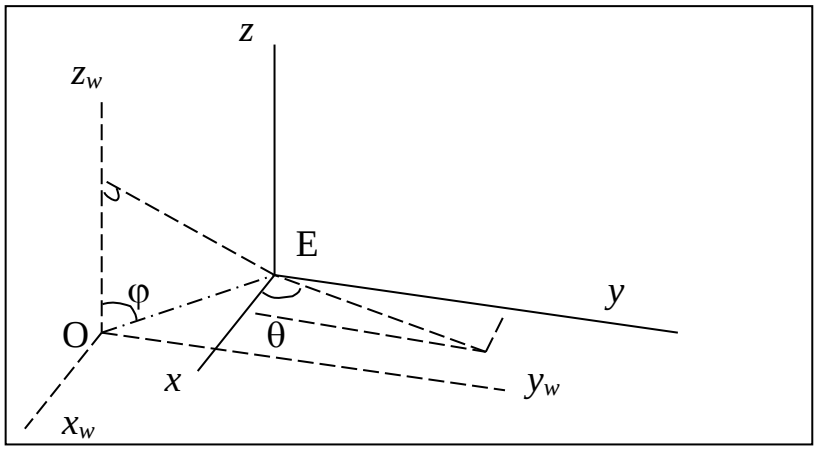


Рис.23.5 – Новые оси после переноса

23.2.2 Поворот координатной системы вокруг оси Z

Обращаясь к рис. 23.5, повернем систему координат вокруг оси z на угол $\pi/2 - \theta$ в отрицательном направлении. В результате ось y совпадет по направлению с горизонтальной составляющей отрезка OE, а ось x будет

расположена перпендикулярно отрезку OE. Матрица для такого изменения координат будет совпадать с матрицей для поворота точки на такой же угол в положительном направлении. Матрица 3 x 3 для этого поворота равна

$$R_Z = \begin{bmatrix} \cos(\pi/2-\theta) & \sin(\pi/2-\theta) & 0 \\ -\sin(\pi/2-\theta) & \cos(\pi/2-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \sin(\theta) & \cos(\theta) & 0 \\ -\cos(\theta) & \sin(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (23.4)$$

Новое положение осей показано на рис. 23.6.

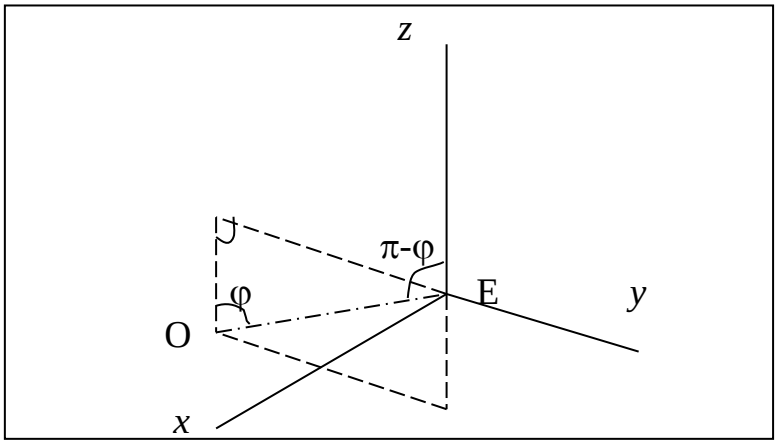


Рис.23.6 – Новые оси после поворота вокруг оси z

23.2.3 Поворот системы координат вокруг оси x

Так как новая ось *z* должна совпадать по направлению с отрезком EO, повернем систему координат вокруг оси *x* на угол $\pi - \varphi$ в положительном направлении, что соответствует повороту точки на угол $-(\pi - \varphi) = \varphi - \pi$.

Новые оси показаны на рис. 23.7.

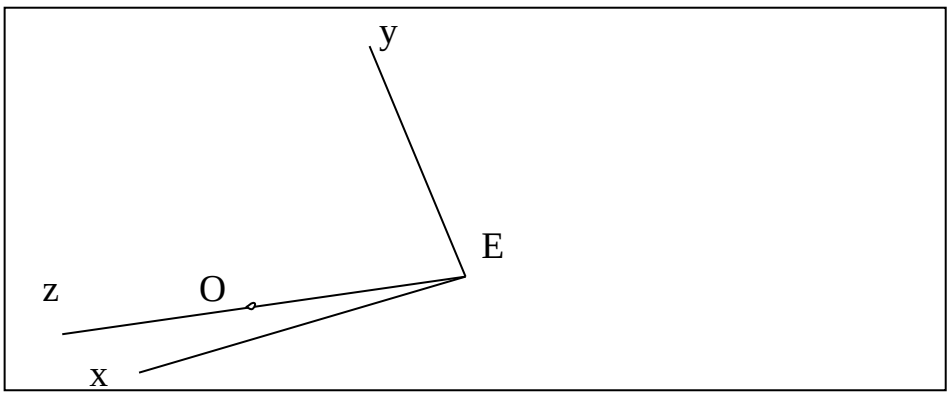


Рис. 23.7 – Новые оси после поворота вокруг оси x

Матрица поворота:

$$R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi-\pi) & \sin(\varphi-\pi) \\ 0 & -\sin(\varphi-\pi) & \cos(\varphi-\pi) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & -\cos(\varphi) \end{bmatrix} \qquad (23.5)$$

23.2.4 Изменение направления оси x

На рис. 23.7 оси *y* и *z* имеют правильную ориентацию, а ось *x* должна быть направлена в противоположную сторону. Поэтому необходима матрица для выполнения преобразования $x' = -x$, то есть

$$M_{yz} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (23.6)$$

После этого завершающего преобразования получим систему видовых координат, уже показанную на рис. 23.4.

Вычислим матрицу отображения *V* как матричное произведение

$$V = TR_Z^* R_X^* M_{YZ}^* \qquad (23.7)$$

где обозначение *R** использовано для матрицы 4 x 4, полученной путем расширения матрицы *R*, имеющей размерность 3 x 3, добавлением четвертой строки и четвертого столбца, содержащих числа 0, 0, 0, 1 именно в этом порядке. Матричное произведение не коммутативно (то есть в общем случае *A*B <> B*A)*, но ассоциативно, поэтому уравнение (23.7) можно переписать в виде

$$V = T(R_Z R_X M_{YZ})^*$$

Таким образом можно работать с матрицами 3 к 3 до тех пор, пока это возможно. Дальнейшая проработка задачи умножения матриц связана с использованием обозначений

$$\cos(\varphi) = a \quad \sin(\varphi) = b \quad \cos(\theta) = c \quad \sin(\theta) = d \qquad (23.8)$$

откуда следует, что $a^2 + b^2 = 1$ и $c^2 + d^2 = 1$.

На основе уравнения (23.1), перепишем уравнение (23.3) в виде

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\rho bc & -\rho bd - \rho a & 1 \end{bmatrix}$$

а уравнения (23.4) и (23.5) как

$$R_Z = \begin{bmatrix} d & c & 0 \\ -c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_X == \begin{bmatrix} 1 & 0 & 0 \\ 0 & -a & -b \\ 0 & b & -a \end{bmatrix}$$

Следовательно,

$$R_Z R_X == \begin{bmatrix} d & -ac & -bc \\ -c & -ad & -bd \\ 0 & b & -a \end{bmatrix}$$

Умножим эту матрицу справа на матрицу M_{yz} из уравнения (23.6):

$$R_Z R_X M_{YZ} == \begin{bmatrix} -d & -ac & -bc \\ c & -ad & -bd \\ 0 & b & -a \end{bmatrix}$$

Затем искомую матрицу отображения V найдем как произведение двух матриц

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\rho bc & -\rho bd - \rho a & 1 \end{bmatrix} \quad \text{и} \quad (R_Z R_X M_{YZ})^* = v_{41} \begin{bmatrix} -d & -ac & -bc & 0 \\ c & -ad & -bd & 0 \\ 0 & b & -a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

откуда

$$V = \begin{bmatrix} -d & -ac & -bc & 0 \\ c & -ad & -bd & 0 \\ 0 & b & -a & 0 \\ v_{41} & v_{42} & v_{43} & 1 \end{bmatrix}$$

где

$$v_{41} = \rho bcd - \rho bcd = 0$$
$$v_{42} = \rho abc^2 + \rho abd^2 - \rho ab = \rho\{ab(c^2 + d^2) - ab\} = \rho(ab - ab) = 0$$
$$v_{43} = \rho b^2 c^2 + \rho b^2 d^2 + \rho a^2 = \rho\{b^2(c^2 + d^2) + a^2\} = \rho\{b^2 + a^2\} = \rho$$

Таким образом, мы нашли

$$V = \begin{bmatrix} -\sin(\theta) & -\cos(\varphi)\cos(\theta) & -\sin(\varphi)\cos(\theta) & 0 \\ \cos(\theta) & -\cos(\varphi)\sin(\theta) & -\sin(\varphi)\sin(\theta) & 0 \\ 0 & \sin(\varphi) & -\cos(\varphi) & 0 \\ 0 & 0 & \rho & 1 \end{bmatrix} \qquad (23.9)$$

Мы получили важный результат: если были заданы сферические координаты ρ, θ, φ для точки наблюдения E, то положение точки в системе видовых координат можно вычислить по значениям ее мировых координат, используя только уравнения (23.2) и (23.9).

После только что выполненного преобразования отображения необходимо определить перспективные преобразования. Хотя уже сейчас можно использовать видовые координаты x_e и y_e просто игнорируя координату z_e . В этом случае будет получена *ортогональная проекция*. Каждая точка P объекта проецируется в точку P' проведением прямой линии из точки P перпендикулярно плоскости, определяемой осями x и y. Эту проекцию можно также считать перспективной картинкой, которая была бы получена при удалении точки наблюдения в бесконечность. Примером такой картинки может служить изображение куба на рис. 23.2(б). Параллельные линии остаются параллельными и на картинке, полученной при ортогональном проецировании.

2.3 Перспективные преобразования

Мировые координаты уже не будут затрагиваться. Поэтому видовые координаты будут обозначаться просто *(x, y, z)* вместо *(x_e, y_e, z_e)*.

На рис. 23.8 была выбрана точка Q, видовые координаты которой равны (0, 0, d) для некоторого положительного числа d. Плоскость $z = d$ определяет экран, который будем использовать. Таким образом, экран – это

плоскость, проходящая через точку *Q* и перпендикулярная оси *z*. Экранные координаты, определяются привязкой начала к точке *Q*, а оси X и Y имеют такие же направления, как оси *x* и *y* соответственно. Для каждой точки объекта *P* точка изображения *P'* определяется как точка пересечения прямой линии PE и экрана. Чтобы упростить рис. 23.8, будем считать, что точка *P* имеет нулевую *y*-координату. Но все последующие уравнения для вычисления ее *y*-координаты также пригодны и для любых других значений координаты *X*.

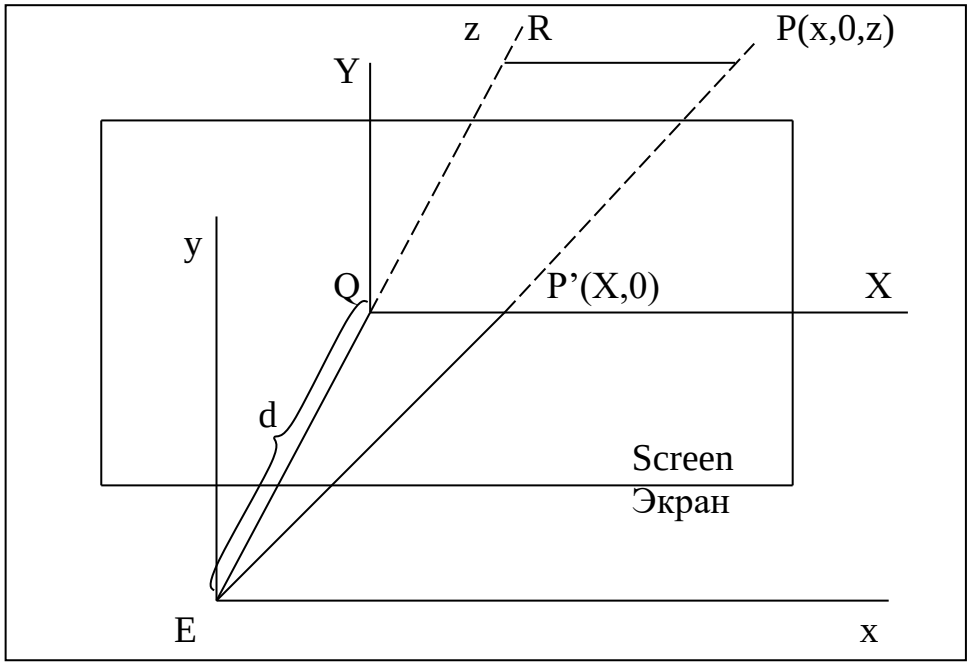


Рис.23.8 – Экран и видовые координаты

На рис. 23.8 треугольники EPR и EP'Q подобны. Следовательно,

$$\frac{P'Q}{EQ} = \frac{PR}{ER}$$

Отсюда будем иметь

$$\frac{X}{d} = \frac{x}{z}$$
$$X = d * \frac{x}{z} \qquad (23.10)$$

Аналогично можем получить

$$Y = d * \frac{y}{z} \qquad (23.11)$$

Было введено предположение, что точка O начала системы мировых координат примерно совпадает с центром объекта. Поскольку ось *z* видовой системы координат совпадает с прямой линией EO, которая пересекает экран в некоторой точке, то начало Q системы экранных координат будет находиться в центре изображения. Если бы мы потребовали, чтобы это начало координат располагалось в нижнем левом углу экрана, а размеры экрана составляли $2c_1$ по горизонтали и $2c_2$ по вертикали, то можно заменить уравнения (23.10) и (23.11) на

$$X = d * \frac{x}{z} + c_1 \qquad (23.12)$$

$$Y = d * \frac{y}{z} + c_2 \qquad (23.13)$$

Нам еще требуется определить расстояние между точкой наблюдения E и экраном. Грубо говоря, мы имеем соотношение:

$$\frac{размер\ картинки}{d} = \frac{размер\ объекта}{\rho}$$

что следует из подобия треугольников EP' Q' и EPQ на рис. 23.9.

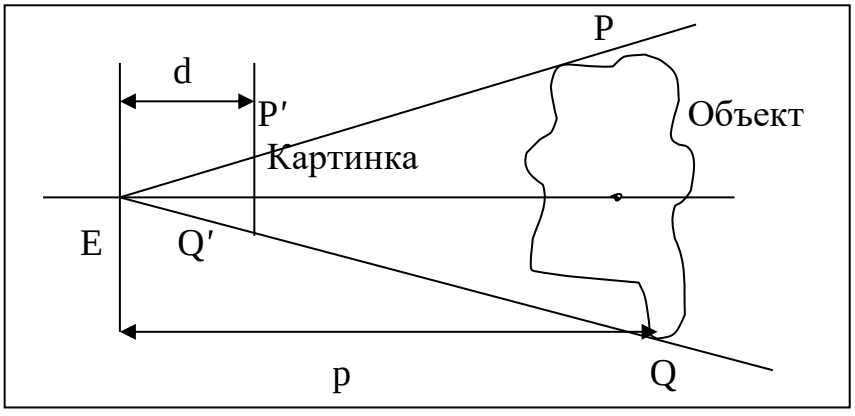


Рис. 23.9 – Размеры картинки и объекта

Отсюда получим

$$d = \rho * \frac{размер\ картинки}{размер\ объекта} \qquad (23.14)$$

Это выражение равно применимо для горизонтального и вертикального размеров. Его следует интерпретировать скорее, как средство для оценки подходящего значения d, чем точное предписание, поскольку трехмерный объект может иметь очень сложную форму и не всегда ясно, какие его размеры следует включать в это уравнение. Можно ввести грубую оценку размеров объекта по максимуму его длины, ширины и высоты. Исходя из оценки уравнения (23.14) можно сделать вывод, что размеры картинки должны быть несколько меньше размеров экрана.

Проанализируем подробнее общую концепцию относительно точек схода и горизонта.

На рис. 23.10 изображены точки наблюдения E и экрана ABCD.

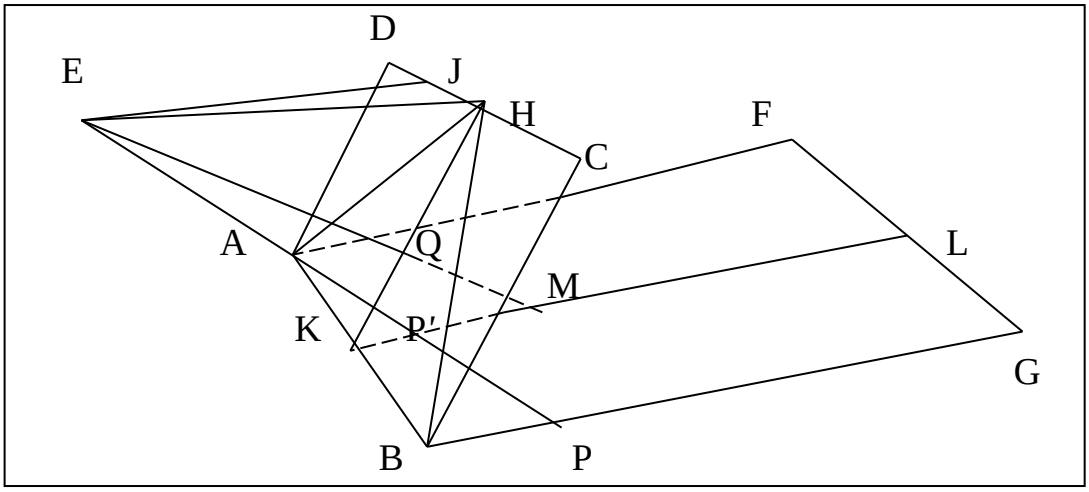


Рис. 23.10 – Точка схода H

Как и прежде, взгляд направлен из точки E в точку Q. Прямые линии AF, BG, EH параллельны, и будем считать их горизонтальными. Предположим, что имеется плоскость, проходящая через параллельные линии EH и BG. Эта плоскость пересекает плоскость экрана по прямой линии BH. Таким образом, каждая точка P на прямой линии BG будет иметь свою центральную проекцию P', лежащую на прямой BH, при условии, что точка E является центром проекции. Пусть точка P удаляется от точки B в бесконечность, тогда ее проекция P' будет приближаться к точке H. Это

означает, что *H* – точка схода для прямой линии, проходящей через точки *B* и *G*. В терминах проективной геометрии точка *H* представляет собой проекцию бесконечно удаленной точки, лежащей на прямой BG. Как уже упоминалось, параллельные линии пересекаются в бесконечно удаленной точке, так что точка *H* будет также проекцией бесконечно удаленной точки, лежащей на прямой линии AF. Если возьмем прямую линию с другим направлением, но также горизонтальную, например, прямую линию BF, то такая линия также будет иметь точку схода, лежащую на прямой линии CD. Она находится как точка пересечения прямой линии CD и прямой линии, проходящей через точку E и параллельной этой выбранной горизонтальной прямой линии. Прямая линия CD – это линия горизонта. Каждая точка J на линии горизонта является точкой схода всех прямых линий, параллельных прямой линии EJ.

Точка схода есть не только у горизонтальных линий. На рис. 23.11 проведены вертикальные прямые линии CA и DB. Точкой схода для них будет F. Это та точка, в которой вертикальная прямая линия, проходящая через точку E, пересекает экран. Отрезки прямых линий CA и DB имеют проекции CA' и DB', которые не параллельны.

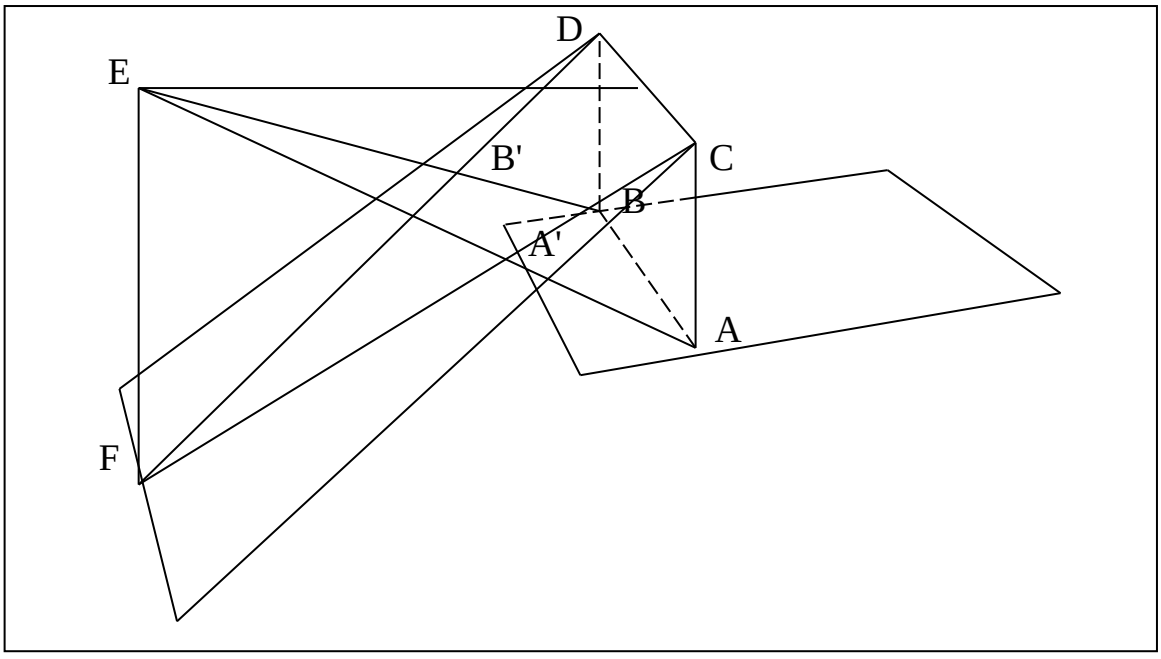


Рис. 23.11 – Точка схода F для вертикальных линий

Строгий эффект перспективы для вертикальных прямых линий не всегда приемлем. Часто предпочитают картинки, в которых вертикальные линии представляются почти вертикальными. Это происходит потому, что мы больше привыкли к горизонтальному, или почти горизонтальному, направлению взгляда. Некоторые чувствуют даже головокружение, если смотрят далеко вниз. Художники в своих картинах применяют «псевдоперспективу», когда вертикальные линии рисуются точно вертикальными, даже если направление взгляда не горизонтально. В таком случае полученная картинка отличается от действительно видимой, но выглядит достаточно правдоподобной. Пример показан на рис. 23.12(б).

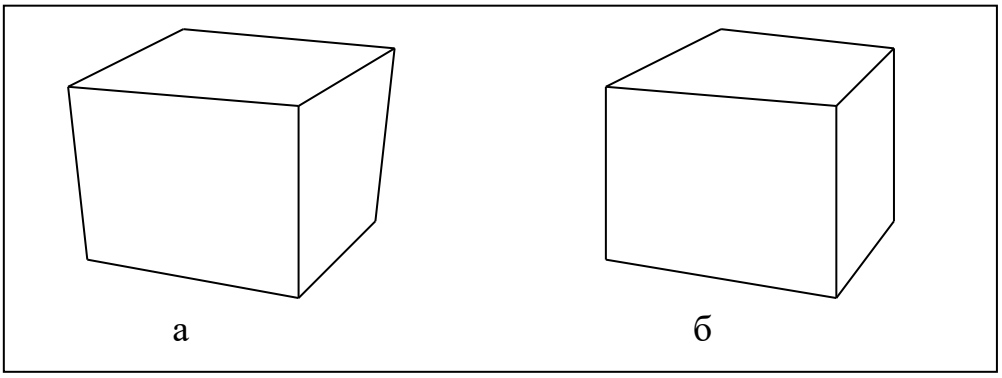


Рис. 23.12 – Перспективные изображения:
а – перспектива; б – псевдоперспектива

Рекомендуется выбирать точку наблюдения не очень близко к объекту, особенно в тех случаях, когда угол φ, показанный на рис. 23.3, примерно равен 90°. В этом заключается практический способ исключения сильного эффекта перспективы для вертикальных линий.

Другая несколько дискуссионная тема – представление линий, параллельных экрану. Они будут изображаться параллельными линиями на картинке. Рассмотрим, например, рис. 23.13 с изображениями десяти кубиков, расположенными в одну линию. Направление наблюдения – горизонтальное, то есть угол $\varphi = 90°$.

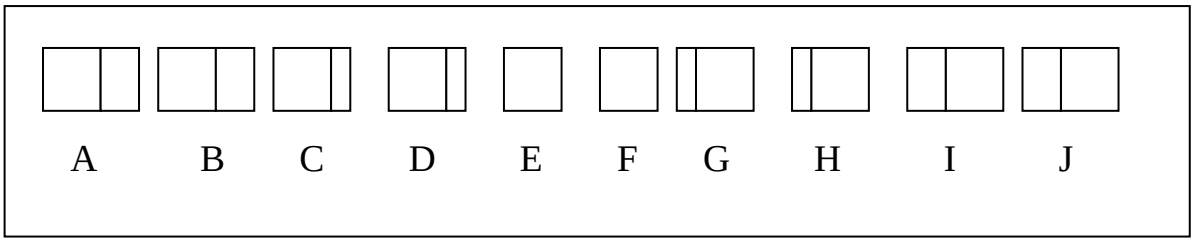


Рис. 23.13 – Десять кубиков, параллельных экрану

На этой картинке можно видеть парадоксальный эффект, относящийся к размерам кубиков. Кубик *A* удален значительно дальше, чем кубик *E*, но на картинке оба этих кубика имеют одинаковые размеры. Кажется, что это неправильно, поскольку удаленные объекты должны иметь меньшие видимые размеры, чем более близкие. Однако это является результатом центрального проецирования, полученного по точным геометрическим правилам, выведенным на основе наших представлений. Изображение на рис. 23.13 практически нереально. Особенность нашего глаза такова, что мы можем видеть только точки, расположенные внутри определенного конуса, ось которого совпадает с направлением взгляда EO. Очень важный параметр этого конуса – угол α_{max}, отмеченный на рис. 23.14.

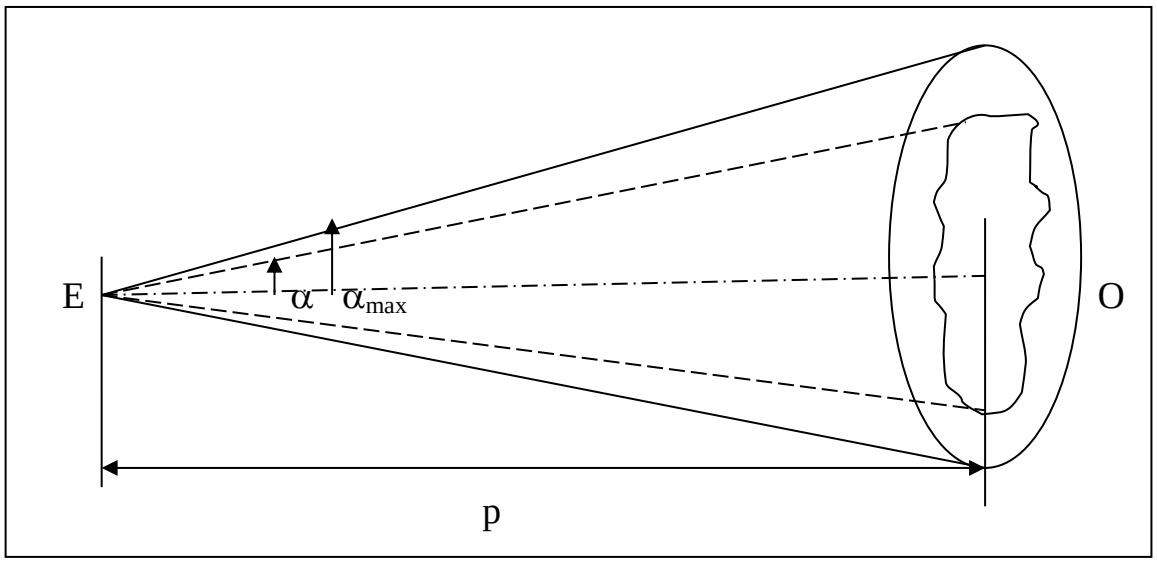


Рис.23.14 – Конус наблюдения

Глаз, как и камера, допускает только такие значения угла α, которые не превышают некоторого максимального значения α_{max}. При вычислениях

лучше ограничивать не фактическое значение угла, показанное на рис. 24.14, а примерно выдерживать отношение

$$tg(\alpha) = \frac{0.5 * размер\,объекта}{\rho}$$

Из этого опять следует, что выбор слишком малого значения расстояния ρ, может быть источником затруднений. Если же значение ρ будет выбрано сравнительно большим, то угол α может оказаться малым, чтобы избежать обсуждаемых затруднений. Если все-таки это не годится, то может быть принято решение о замене плоского экрана частью сферической поверхности с центром в точке E. Таким образом, можно допустить большие значения угла α, но при этом картинка не получится плоской. Сферическая картинка, в свою очередь, может быть спроецирована на плоскость. Далее ограничимся только плоскими картинками, которые получаются при сравнительно малых углах α.

Тема 24. Вычерчивание проволочных моделей

24.1 Программа для вычерчивания куба

Для многих задач необходимо заранее решить, насколько общими будут программы. Существуют две крайние возможности. Можно написать специфическую программу, которая не считывает входные данные и выдает один результат, скажем, некоторую картинку. С другой стороны, можно разработать очень общую программу, которая может вычертить любую картинку при условии, что задан файл с исходными данными.

Разработаем программу для вычерчивания перспективного изображения куба с длиной ребер, равной 100. Этот произвольный размер не имеет никаких ограничений в отношении картинки, поскольку нижеследующие параметры могут выбираться совершенно свободно:

1) три сферические координаты:

ρ – расстояние до точки наблюдения EO;

θ – угол в горизонтальном направлении от оси *x*;

φ – угол, измеренный по вертикали от оси z;

2) расстояние *d* между экраном и точкой наблюдения.

Начало системы координат выбирается в центре куба (см. рис. 24.1).

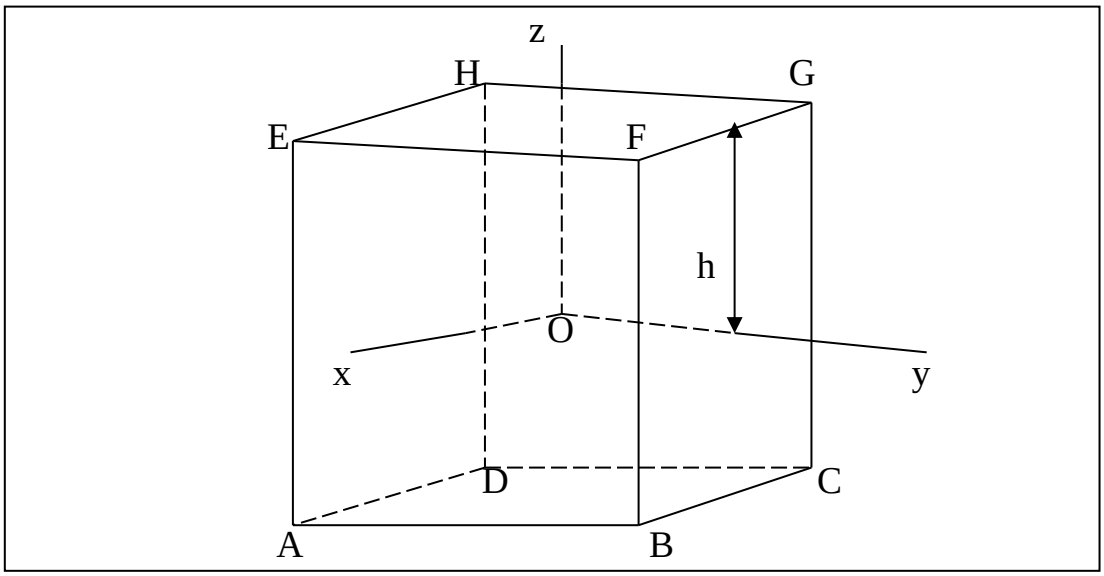


Рис.24.1– Куб и система мировых координат

Длина каждого ребра обозначается через *2h*, следовательно, $h = 50$. Тогда восемь вершин куба будут иметь следующие координаты:

$$A\,(h, -h, -h)$$
$$B\,(h, h, -h)$$
$$C\,(-h, h, -h)$$
$$D\,(-h, -h, -h)$$
$$E\,(h, -h, -h)$$
$$F\,(h, h, h)$$
$$G\,(-h, h, h)$$
$$H\,(-h, -h, h)$$

Будем вычерчивать так называемую проволочную модель, что означает отсутствие различий между видимыми и скрытыми линиями. В данной программе это будет выглядеть так, как если бы перо перемещалось в трехмерном пространстве. Для этой цели будем использовать функцию *dw(x1, y1, z1, x2, y2, z2)*, аналогичную функции *draw(x1, y1, x2, y2)* в двухмерном пространстве. В функции *dw* будет выполняться обращение к функции *perspective*, которая реализует как видовое, так и перспективное преобразование. Для эффективности все коэффициенты, не зависящие от конкретных точек от *A* до *H*, вычисляются заранее при помощи функции *coeff*. Ниже приводится текст всей программы *Cube*:

```
/* Cube: Проволочная модель куба */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace Cube
{
  public partial class Form1 : Form
  {
    Graphics dc; Pen p;
    /* Коэффициенты матрицы видового преобразования */
    double v11, v12, v13, v21, v22, v23, v32, v33, v43;
    /* Сферические координаты точки наблюдения */
    double rho = 150.0, theta = 320.0, phi = 45.0;
    /* Расстояние от точки наблюдения до экрана */
    double screen_dist = 100.0;
    /* Смещение относительно левого нижнего угла экрана */
    double c1 = 5.0, c2 = 3.5;
```

```
  /* Половина длины ребра куба */
  double h = 2;
  public Form1()
  {
    InitializeComponent();
    dc = pictureBox1.CreateGraphics();
    p = new Pen(Brushes.Black, 1);
  }
/* Функция преобразования вещественной координаты X в целую */
  private int IX(double x)
  {
    double xx = x * (pictureBox1.Size.Width / 10.0) + 0.5;
    return (int)xx;
  }
/* Функция преобразования вещественной координаты Y в целую */
  private int IY(double y)
  {
    double yy = pictureBox1.Size.Height - y *
                (pictureBox1.Size.Height / 7.0) + 0.5;
    return (int)yy;
  }
/* Вычисление коэффициентов, не зависящих от вершин куба */
  private void coeff(double rho, double theta, double phi)
  {
    double th, ph, costh, sinth, cosph, sinph, factor;
    factor = Math.PI / 180.0; // из градусов в радианы
    th = theta * factor;
    ph = phi * factor;
    costh = Math.Cos(th);
    sinth = Math.Sin(th);
    cosph = Math.Cos(ph);
    sinph = Math.Sin(ph);
    /* Элементы матрицы V видового преобразования
     | -sin(th) -cos(phi) * cos(th) -sin(phi) * cos(th)  0 |
  V= | cos(th)  -cos(phi) * sin(th) -sin(phi) * sin(th)  0 |
     | 0         sin(phi)           -cos(phi)            0 |
     | 0         0                   rho                 1 |
    */
    v11 = -sinth; v12 = -cosph * costh; v13 = -sinph * costh;
    v21 = costh; v22 = -cosph * sinth; v23 = -sinph * sinth;
    v32 = sinph;    v33 = -cosph; v43 = rho;
  }
/* Функция видового и перспективного преобразования координат */
  private void perspective(double x, double y, double z, ref
                                  double pX, ref double pY)
  {
    double xe, ye, ze;
    /*координаты точки наблюдения, вычисляемые по уравнению
     [Xe Ye Ze 1]= [Xw Yw Zw 1]*V
    */
    xe = v11 * x + v21 * y;
    ye = v12 * x + v22 * y + v32 * z;
    ze = v13 * x + v23 * y + v33 * z + v43;
```

```
      /* Экранные координаты,вычисляемые по формулам
        X= d* (x/z)+c1,  Y= d*(y/z)+c2,
        где - расстояние от точки наблюдения до экрана
      */
      pX = screen_dist * xe / ze + c1;
      pY = screen_dist * ye / ze + c2;
    }
  /* Функция вычерчивания линии (экран 10x7 условн. единиц) */
    private void dw(double x1, double y1, double z1, double
                                    x2, double y2, double z2)
    {
      double X1=0, Y1=0, X2=0, Y2=0;
      /* Преобразование мировых координат в экранные */
      perspective(x1, y1, z1, ref X1, ref Y1);
      perspective(x2, y2, z2, ref X2, ref Y2);
      /* Вычерчивание линии */
      Point point1 = new Point(IX(X1), IY(Y1));
      Point point2 = new Point(IX(X2), IY(Y2));
      dc.DrawLine(p, point1, point2);
    }
  /* Функция рисования проволочной модели куба */
    private void drawCube(double h)
    {
      dw(h, -h, -h, h, h, -h);     // Отрезок AB
      dw(h, h, -h, -h, h, -h);     // Отрезок BC
      dw(-h, h, -h, -h, h, h);     // Отрезок CG
      dw(-h, h, h, -h, -h, h);     // Отрезок GH
      dw(-h, -h, h, h, -h, h);     // Отрезок HE
      dw(h, -h, h, h, -h, -h);     // Отрезок EA
      dw(h, h, -h, h, h, h);       // Отрезок BF
      dw(h, h, h, -h, h, h);       // Отрезок FG
      dw(h, h, h, h, -h, h);       // Отрезок FE
      dw(h, -h, -h, -h, -h, -h);   // Отрезок AD
      dw(-h, -h, -h, -h, h, -h);   // Отрезок DC
      dw(-h, -h, -h, -h, -h, h);   // Отрезок DH
    }
  /* Вызов главной функции рисования проволочной модели куба */
    private void button1_Click(object sender, EventArgs e)
    {
      coeff(rho, theta, phi);
      drawCube(h);
    }
  }
}
```

Результат работы программы показан на рис. 24.2.

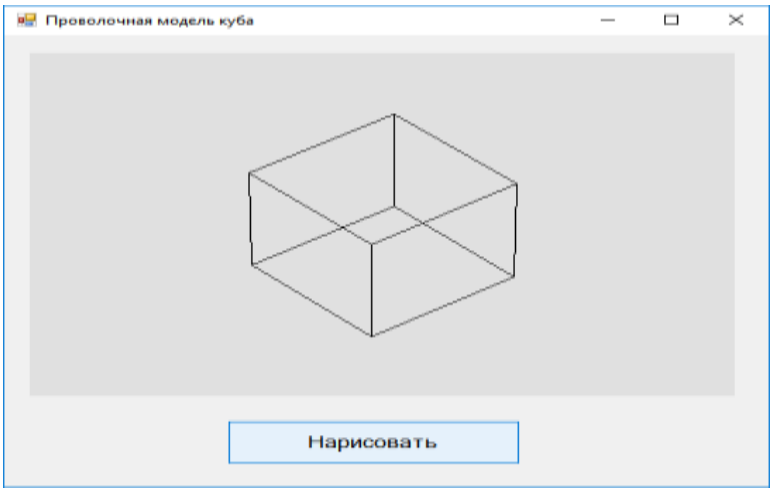


Рис. 24.2 – Результат работы программы *Cube*

Программа в результате своей работы выдала также изображения кубов, показанные на рис. 24.3.

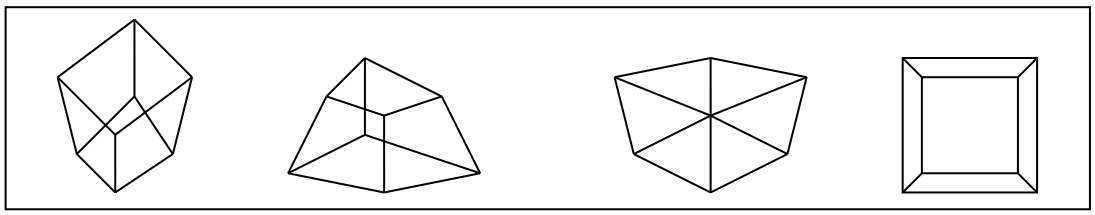


Рис. 24.3 – Кубы, видимые из различных точек

На второй картинке взгляд направлен снизу куба, а на третьей картинке стороны и диагонали лежат на одной прямой. Гораздо интереснее вычертить изображение сплошного тела, а не его проволочную модель. Если применить некоторые меры предосторожности при задании точки наблюдения, то можно вычертить изображения не очень сложных объектов с последующим ручным удалением невидимых линий. Для изображения на рис. 24.1 выбрана такая точка наблюдения, что видны только грани ABFE, BCGF, EFGH, а грани AD, CD, DH должны быть удалены. Это сделано путем удаления из программы *Cube* трех программных строк. Программа *Cube* была использована и для получения рис. 23.2 (а) при $\rho = 200$, $\theta = 30$, $\varphi = 70$, $d = 3$ и рис. 23.2(б) при значениях $\rho = 200000$, $\theta = 30$, $\varphi = 70$, $d = 3000$.

24.2 Вычерчивание проволочных моделей

Рассмотрим программу, которая может вычертить перспективное изображение любой проволочной модели, образованной из конечного числа отрезков прямых линий. Опуская определенные отрезки прямых линий, можно выполнить картинки для простых сплошных тел. Как и ранее, будем использовать точку наблюдения E и «объектную точку» O, которые определяют направление наблюдения. Для пользователя не всегда удобно, чтобы точка O совпадала с началом системы мировых координат. Поэтому будем требовать только, чтобы пользовательская ось z располагалась вертикально. Пользователь должен задать координаты x_O, y_O, z_O объектной точки O, выраженные в своей системе координат. После считывания пользовательских координат они преобразуются в систему «внутренних мировых координат» x, y, z с точкой O как начало по следующему правилу:

x = пользовательская x-координата – x_O;

y = пользовательская y-координата – y_O;

z = пользовательская z-координата – z_O;

Затем пользователь должен определить сферические координаты ρ, θ, φ точки наблюдения E относительно нового начала координат O. Также должно быть задано расстояние между точкой наблюдения и экраном. Тем самым определяется положение экрана, поскольку он перпендикулярен направлению наблюдения EO. Для определения отрезков прямых линий нам опять придется представить перемещение в трехмерном пространстве, которое ассоциируется с соответствующим перемещением пера на картинке. Для каждого перемещения задаются коды: 0 – перо поднято, 1 – перо опущено.

Для вычерчивания двух примыкающих друг к другу отрезков PQ и QR необходимо задать три входные строки следующей структуры:

x_P y_P z_P 0 (перенос пера в точку P)
x_Q y_Q z_Q 1 (вычерчивание отрезка PQ)
x_R y_R z_R 1 (вычерчивание отрезка QR)

Правая часть каждой строки является комментарием. Ее наличие во входном файле не влияет на работу программы.

Рассмотрим пример полного набора входных данных. Обратимся к рис. 23.10, который совсем не тривиален. Если вычерчивать этот рисунок вручную, то возникает проблема определения положения точки *M* на прямой линии KL, такой, чтобы в трехмерном пространстве отрезок EM был перпендикулярен плоскости BCDA.

Выберем точку *K* в качестве начала пользовательской системы координат. Пусть положительная полуось *y* в этой системе проходит через точку *L*, а положительная полуось x – через точку *B*. Вид с положительной оси *x* на плоскость yKz показан на рис. 24.4. В трехмерном пространстве точка *K* находится в середине отрезка AB, длина которого равна 10.

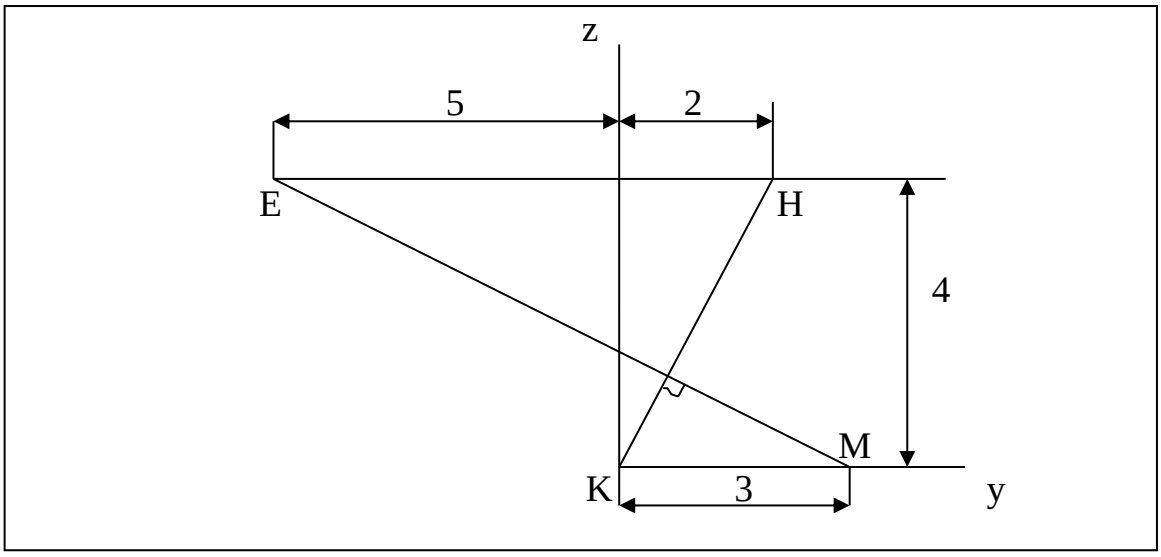


Рис.24.4 – Вид с положительной полуоси *x*

Теперь можно записать полный набор входных данных для картинки, которую собираемся начертить, например, в файл *Planes*:

0 2 2 (объектная точка O)

30 –15 75 15 (ρ, θ, φ, d)

5 0 0 0 (перенос в точку B)

5 2 4 1 (вычерчивание отрезка BC)

–5 2 4 1 (вычерчивание отрезка CD)

–5 0 0 1 (вычерчивание отрезка DA)

5 0 0 1 (вычерчивание отрезка AB)

5 9 0 1 (вычерчивание отрезка BG)

–5 9 0 1 (вычерчивание отрезка GF)

–5 0 0 1 (вычерчивание отрезка FA)

0 –5 4 0 (перенос в точку E)

0 2 4 1 (вычерчивание отрезка EH)

0 0 0 1 (вычерчивание отрезка HK)

0 9 0 1 (вычерчивание отрезка KL)

0 –5 4 0 (перенос в точку E)

0 3 0 1 (вычерчивание отрезка EM)

Для параметра d было найдено значение 15 путем подстановки в формулу 23.14 значений ρ = 30, *размер картинки* = 7, *размер объекта* = 14.

На рис. 24.5 показан результат работы будущей программы *GPers* на основании приведенных выше входных данных.

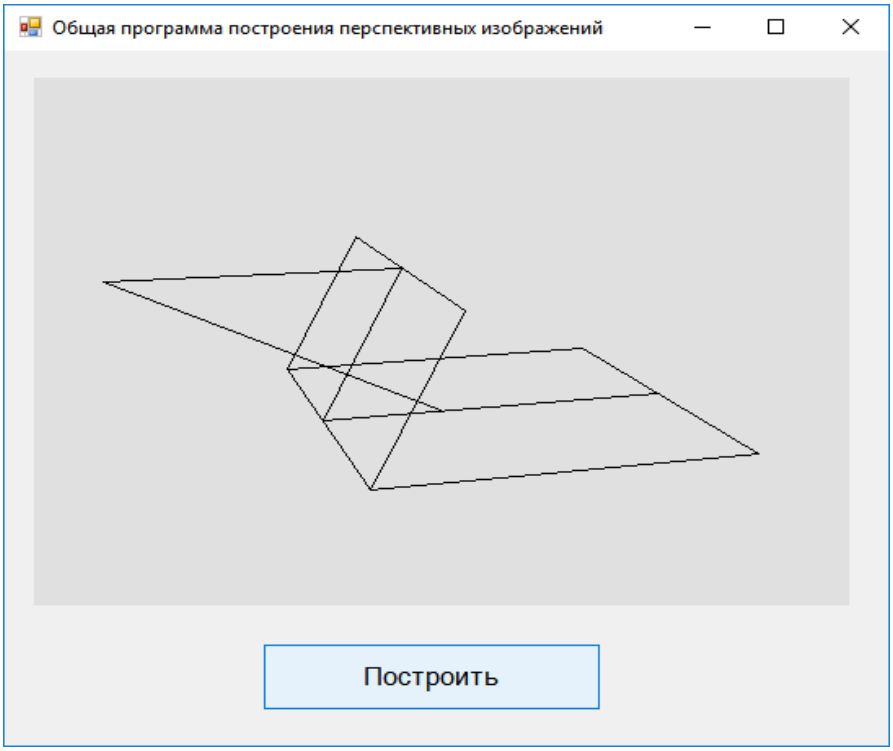


Рис.24.5 – Пример работы программы *GPers*

Программа *GPers* будет считывать данные из файла, например, *Planes*.

Видовые и перспективные преобразования выполняются так же, как в программе *Cube*.

```
/* GPers: Общая программа построения перспективных изображений */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace GPers
{
  public partial class Form1 : Form
  {
    Graphics dc; Pen p;
    /* Коэффициенты матрицы видового преобразования */
    double v11, v12, v13, v21, v22, v23, v32, v33, v43;
    /* Смещение относительно левого нижнего угла экрана */
    double c1 = 4.5, c2 = 3.5;
    /* Расстояние от точки наблюдения до экрана */
    double screen_dist;
    public Form1()
    {
      InitializeComponent();
      dc = pictureBox1.CreateGraphics();
      p = new Pen(Brushes.Black, 1);
    }
  /* Функция преобразования вещественной координаты X в целую */
    private int IX(double x)
    {
      double xx = x * (pictureBox1.Size.Width / 10.0) + 0.5;
      return (int)xx;
    }
  /* Функция преобразования вещественной координаты Y в целую */
    private int IY(double y)
    {
      double yy = pictureBox1.Size.Height - y *
                      (pictureBox1.Size.Height / 7.0) + 0.5;
      return (int)yy;
    }
  /* Вычисление коэффициентов */
    private void coeff(double rho, double theta, double phi)
    {
      double th, ph, costh, sinth, cosph, sinph, factor;
      factor = Math.PI / 180.0; // из градусов в радианы
      th = theta * factor; ph = phi * factor;
      costh = Math.Cos(th); sinth = Math.Sin(th);
      cosph = Math.Cos(ph); sinph = Math.Sin(ph);
```

```
    v11 = -sinth; v12 = -cosph * costh; v13 = -sinph * costh;
    v21 = costh; v22 = -cosph * sinth; v23 = -sinph * sinth;
    v32 = sinph; v33 = -cosph; v43 = rho;
  }
/* Функция видового и перспективного преобразования координат */
  private void perspective(double x, double y, double z, ref
                                    double pX, ref double pY)
  {
    double xe, ye, ze;
    xe = v11 * x + v21 * y;
    ye = v12 * x + v22 * y + v32 * z;
    ze = v13 * x + v23 * y + v33 * z + v43;
    pX = screen_dist * xe / ze + c1;
    pY = screen_dist * ye / ze + c2;
  }
/* Функция вычерчивания линии (экран 10x7 условн. единиц) */
  private void dw(double x1, double y1, double z1, double
                                  x2, double y2, double z2)
  {
    double X1 = 0, Y1 = 0, X2 = 0, Y2 = 0;
    /* Преобразование мировых координат в экранные */
    perspective(x1, y1, z1, ref X1, ref Y1);
    perspective(x2, y2, z2, ref X2, ref Y2);
    /* Вычерчивание линии */
    Point point1 = new Point(IX(X1), IY(Y1));
    Point point2 = new Point(IX(X2), IY(Y2));
    dc.DrawLine(p, point1, point2);
  }
/* Отрисовка перспективного изображения */
  private void button1_Click(object sender, EventArgs e)
  {
    double rho, theta, phi, x0, y0, z0, x, y, z;
    int code;
    double xold = 0, yold = 0, zold = 0;
    string file_name = "";
    /* Выбор файла с исходными данными изображения */
    OpenFileDialog openFileDialog1 = new OpenFileDialog();
    openFileDialog1.InitialDirectory =
                                    Application.StartupPath;
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    { file_name = openFileDialog1.FileName; }
    /* Чтение данных из файла */
    StreamReader reader = new StreamReader(file_name);
    /* Чтение из файла координат точки O */
    string[] xyz = reader.ReadLine().Split(' ');
    x0 = Convert.ToDouble(xyz[0]);
    y0 = Convert.ToDouble(xyz[1]);
    z0 = Convert.ToDouble(xyz[2]);
    /* Чтение параметров ro, teta, fi, screen_dist */
    xyz = reader.ReadLine().Split(' ');
    rho = Convert.ToDouble(xyz[0]);
    theta = Convert.ToDouble(xyz[1]);
    phi = Convert.ToDouble(xyz[2]);
```

```
      screen_dist = Convert.ToDouble(xyz[3]);
      coeff(rho, theta, phi);
      /* Чтение координат точек из файла */
      string line;
      while ((line = reader.ReadLine()) != null)
      {
        xyz = line.Split(' ');
        x = Convert.ToDouble(xyz[0]);
        y = Convert.ToDouble(xyz[1]);
        z = Convert.ToDouble(xyz[2]);
        code = Convert.ToInt32(xyz[3]);
        /* Отрисовка отрезков по координатам */
        if (code == 1) {
          dw(xold, yold, zold, x - x0, y - y0, z - z0);
        }
        xold = x - x0; yold = y - y0; zold = z - z0;
      }
      reader.Close();
    }
  }
}
```

Пользователь программы *Gpers* должен задать расстояние до экрана. По нескольким причинам такое решение неудовлетворительно:

1) пользователю важен размер картинки, а не расстояние до экрана;

2) Параметр «размер объекта» имеет очень неопределенный смысл, поэтому трудно вычислить точное значение для параметра *d*;

3) иногда желательно определить прямоугольную область вывода, занимающую лишь часть экрана, в которую картинка должна быть вписана целиком. В этом случае было бы очень неудобно преобразовывать размеры области вывода в расстояние до экрана.

Есть несколько способов улучшения программ в этих направлениях:

1) отсечение трехмерного объекта по пирамиде, вершина которой совпадает с точкой наблюдения E, а основанием является определенное окно, заданное в мировых координатах (описание такого способа, который здесь применяться не будет, можно найти в книге Ньюмена и Спрула);

2) отсечение картинки в двухмерном пространстве по заданной области вывода (этот способ также не будет применяться);

3) автоматический подбор размера и позиции таким образом, чтобы картинка целиком входила в заданную область вывода. Этот способ мы уже применяли для двухмерных объектов, где программа *GenPlot* действовала в качестве постпроцессора. *GenPlot* можно использовать и в данном случае.

Будем применять третий способ, используя два файла и две программы, как показано на рис. 24.6.

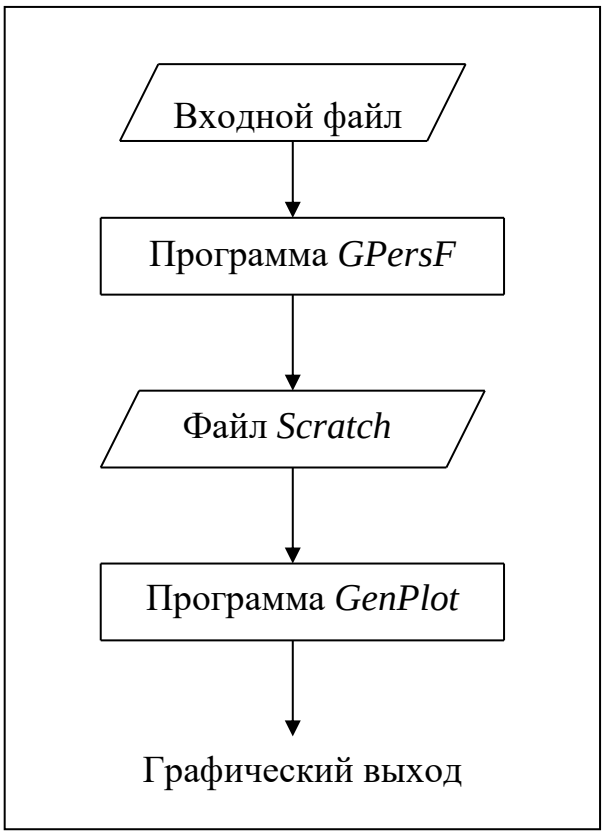


Рис.24.6 – Блок-схема для программ *GPersF* и *GenPlot*

В программе *Gpersf* значение *d* не вводится, а просто задается *d* = 1. Это значение не нужно, поскольку размеры картинки, вычисляемые программой *GenPlot*, зависят только от заданных размеров области вывода.

Следующая программа *GPersF* воспринимает тот же самый входной файл, что и программа *Gpers*, но расстояние до экрана в конце второй строки исходных данных игнорируется и может быть опущено, и записывает экранные координаты точек в файл *Scratch*.

```
/* GPersF: Общая программа построения перспективных
изображений, формирующая выходной файл A.Scratch,
предназначенный для считывания программой GenPlot */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace GPersF
{
  public partial class Form1 : Form
  {
    struct Simple
    {
      public double X; public double Y; public int code;
    };
    Simple s;
    FileInfo file_out = new FileInfo("SCRATCH");
    BinaryWriter writer;
    /* Коэффициенты матрицы видового преобразования */
    double v11, v12, v13, v21, v22, v23, v32, v33, v43;
    public Form1()
    {
      InitializeComponent();
    }
    /* Вычисление коэффициентов */
    private void coeff(double rho, double theta, double phi)
    {
      double th, ph, costh, sinth, cosph, sinph, factor;
      factor = Math.PI / 180.0; // из градусов в радианы
      th = theta * factor; ph = phi * factor;
      costh = Math.Cos(th); sinth = Math.Sin(th);
      cosph = Math.Cos(ph); sinph = Math.Sin(ph);
      v11 = -sinth; v12 = -cosph * costh; v13 = -sinph * costh;
      v21 = costh; v22 = -cosph * sinth; v23 = -sinph * sinth;
      v32 = sinph; v33 = -cosph; v43 = rho;
    }
  /* Функция видового и перспективного преобразования координат */
    private void perspect(double x, double y, double z, ref
                                    double pX, ref double pY)
    {
      double xe, ye, ze;
      xe = v11 * x + v21 * y;
      ye = v12 * x + v22 * y + v32 * z;
      ze = v13 * x + v23 * y + v33 * z + v43;
      pX =  xe / ze;
      pY =  ye / ze;
    }
```

```
  /* Чтение информации, перспективные преобразования и запись
     координат в файл Scratch */
    private void button1_Click(object sender, EventArgs e)
    {
      double rho, theta, phi, x0, y0, z0, x, y, z;
      string file_in = "";
      /* Выбор файла с исходными данными изображения */
      OpenFileDialog openFileDialog1 = new OpenFileDialog();
      openFileDialog1.InitialDirectory =
                                    Application.StartupPath;
      openFileDialog1.Title = "Выбор файла с данными";
      if (openFileDialog1.ShowDialog() == DialogResult.OK)
      { file_in = openFileDialog1.FileName; }
      /* Чтение данных из файла */
      StreamReader reader = new StreamReader(file_in);
      /* Чтение из файла координат точки O */
      string[] xyz = reader.ReadLine().Split(' ');
      x0 = Convert.ToDouble(xyz[0]);
      y0 = Convert.ToDouble(xyz[1]);
      z0 = Convert.ToDouble(xyz[2]);
      /* Чтение параметров ро, teta, fi */
      xyz = reader.ReadLine().Split(' ');
      rho = Convert.ToDouble(xyz[0]);
      theta = Convert.ToDouble(xyz[1]);
      phi = Convert.ToDouble(xyz[2]);
      coeff(rho, theta, phi);
      /* Создание файла Scratch и открытие его на запись */
      writer = new BinaryWriter(file_out.Open(FileMode.Create,
                                        FileAccess.Write));
      /* Чтение координат точек из файла */
      string line;
      while ((line = reader.ReadLine()) != null)
      {
        xyz = line.Split(' ');
        x = Convert.ToDouble(xyz[0]);
        y = Convert.ToDouble(xyz[1]);
        z = Convert.ToDouble(xyz[2]);
        s.code = Convert.ToInt32(xyz[3]);
        perspect(x-x0, y-y0, z-z0, ref s.X, ref s.Y);
        writer.Write(s.X);
        writer.Write(s.Y);
        writer.Write(s.code);
      }
      reader.Close();
      writer.Close();
      MessageBox.Show("Данные сохранены в файл SCRATCH");
    }
  }
}
```

Функция *perspect* в этой программе была получена на основе функции *perspective* из программы *GPers* подстановкой *screen_dist* = 1 и c1 = c2 = 0.

При выборе других значений содержимое файла *Scratch* будет иным, но картинка, сформированная программой *GenPlot*, окажется точно такой же!

На рис. 24.7 показана работа программы *GPersF*.

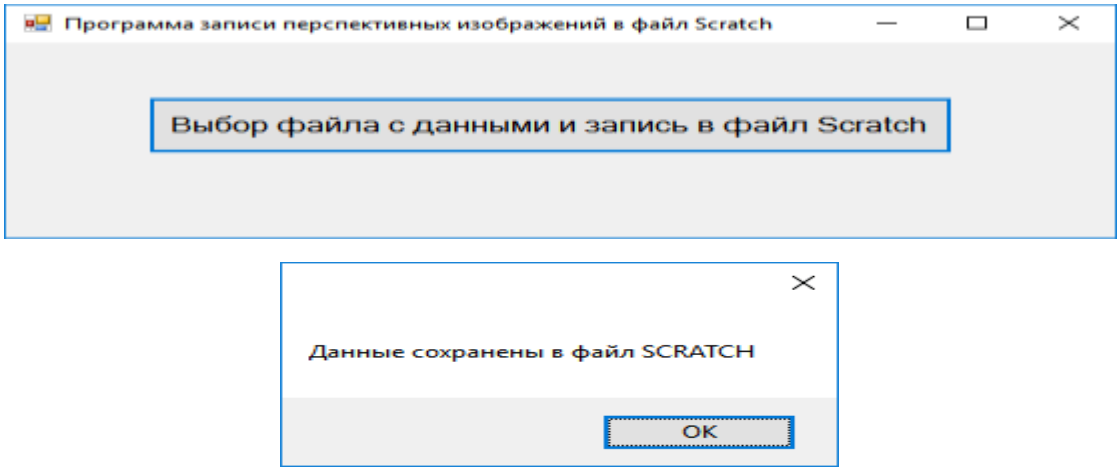


Рис. 24.7 – Результат работы программы *GPersF*

На рис. 24.8 показано перспективное изображение, сформированное программой *GenPlot* на основе данных файла *Scratch*, созданного программой *GPersF*.

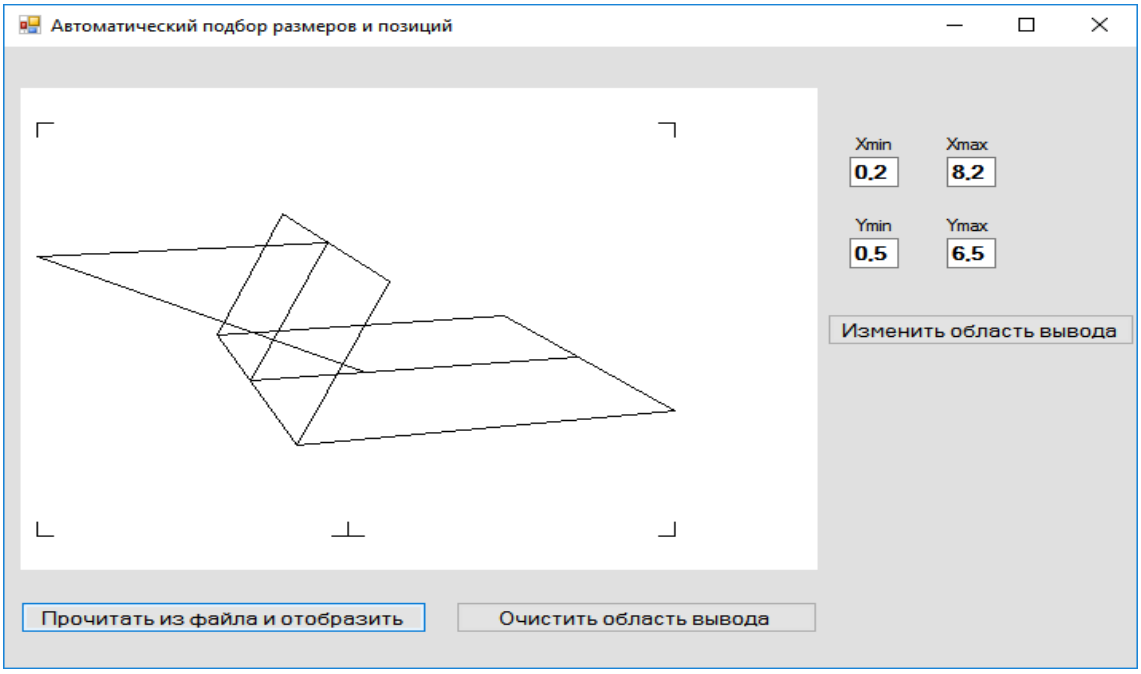


Рис. 24.8 – Результат совместной работы программ *GPersF* и *GenPlot*

24.3 Направление наблюдения, бесконечность, вертикальные линии

Используем программу *GPersF* для обсуждения некоторых новых интересных аспектов. Предположим, что объект очень длинный в направлении оси *x*, как балка на рис. 24.9, размеры которой 200 x 2 x 2.

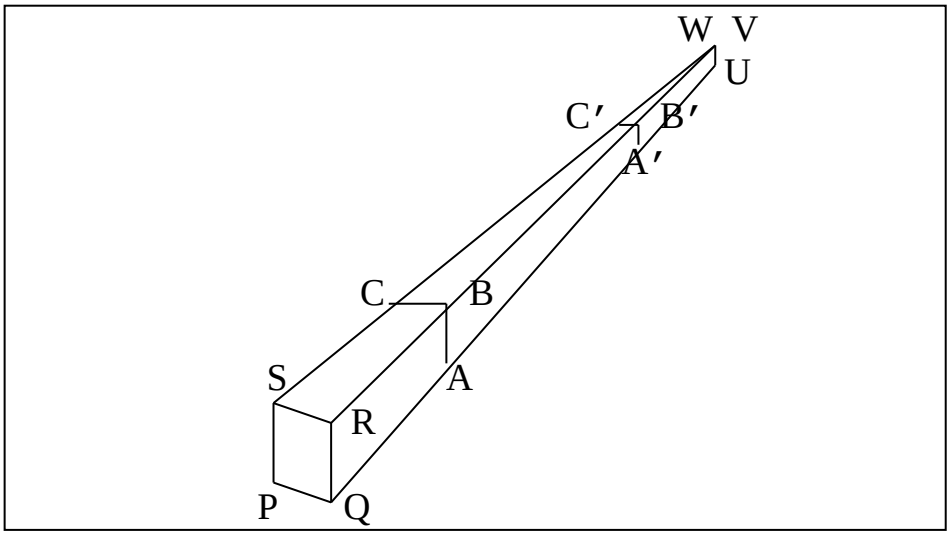


Рис. 24.9 – Длинная балка

Интересным здесь является задание точки O, которую мы должны указать для определения направления наблюдения EO. До сих пор точка O выбиралась в центре объекта. Однако фактически нужна такая точка O, изображение которой будет располагаться в центре картинки. Иногда это очень сильно влияет на изображение. Например, на рис. 24.10 точка O' оказывается по средине отрезка Q'U', хотя соответствующая ей точка O выбрана не на середине исходного отрезка QU. Если бы для нее была задана середина отрезка QU, то направление наблюдения было бы иным.

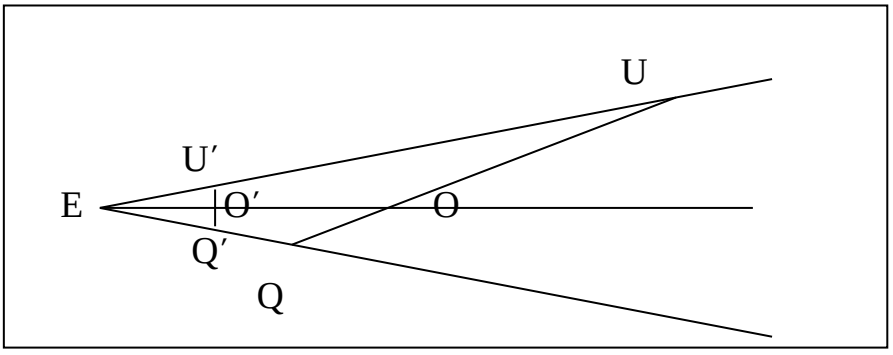


Рис. 24.10 – Объектная точка не в центре объекта

Вернемся к балке, изображенной на рис. 24.9. Очевидно, что точку O следует выбирать не в середине балки, а значительно ближе к глазу. Это возможно осуществить потому, что в нашей программе объектная точка задается пользователем, а не вычисляется как центр объекта. Реальные бесконечные линии при изображении ландшафта на картинке получаются в виде отрезков прямых конечной длины. Для бесконечных объектов нет никакого смысла говорить о центре, хотя направление наблюдения EO существует. Поэтому и требуется определить положение «центральной» точки O объекта. Балка на рис. 24.9, конечно, не является ландшафтом в точном смысле, но ее длина предполагает существование бесконечности. Для данной балки выберем точку O (– 15, 1, 1). За исключением букв *P, Q, R, S, A, B, C, A′, B′, C′, W, V, U,* вся эта картинка была вычерчена с помощью программ *GPersF* и *GenPlot*. Пересечение балки с плоскостью $x = -15$ обозначено буквами *A, B, C*, а пересечение с плоскостью $x = -100$ буквами *A', B', C'*. Следовательно, точка O лежит в плоскости ABC, а плоскость A'B'C' расположена по центру балки. Полный набор входных данных для получения рис. 24.11 определяется в файле списком:

–15 1 1 (объектная точка O)

30 20 70 (po, teta, fi)

0 2 0 0 (перемещение в точку Q)

0 2 2 1 (вычерчивание отрезка QR)

0 0 2 1 (вычерчивание отрезка RS)

0 0 0 1 (вычерчивание отрезка SP)

0 2 0 1 (вычерчивание отрезка PQ)

–200 2 0 1 (вычерчивание отрезка QU)

–200 2 2 1 (вычерчивание отрезка UV)

–200 0 2 1 (вычерчивание отрезка VW)

0 0 2 1 (вычерчивание отрезка WS)

0 2 2 0 (перемещение в точку R)

–200 2 2 1 (вычерчивание отрезка RV)

–15 2 0 0 (перемещение в точку A)

–15 2 2 1 (вычерчивание отрезка AB)

–15 0 2 1 (вычерчивание отрезка BC)

–100 2 0 0 (перемещение в точку A')

–100 2 2 1 (вычерчивание отрезка A'B')

–100 0 2 1 (вычерчивание отрезка B'C')

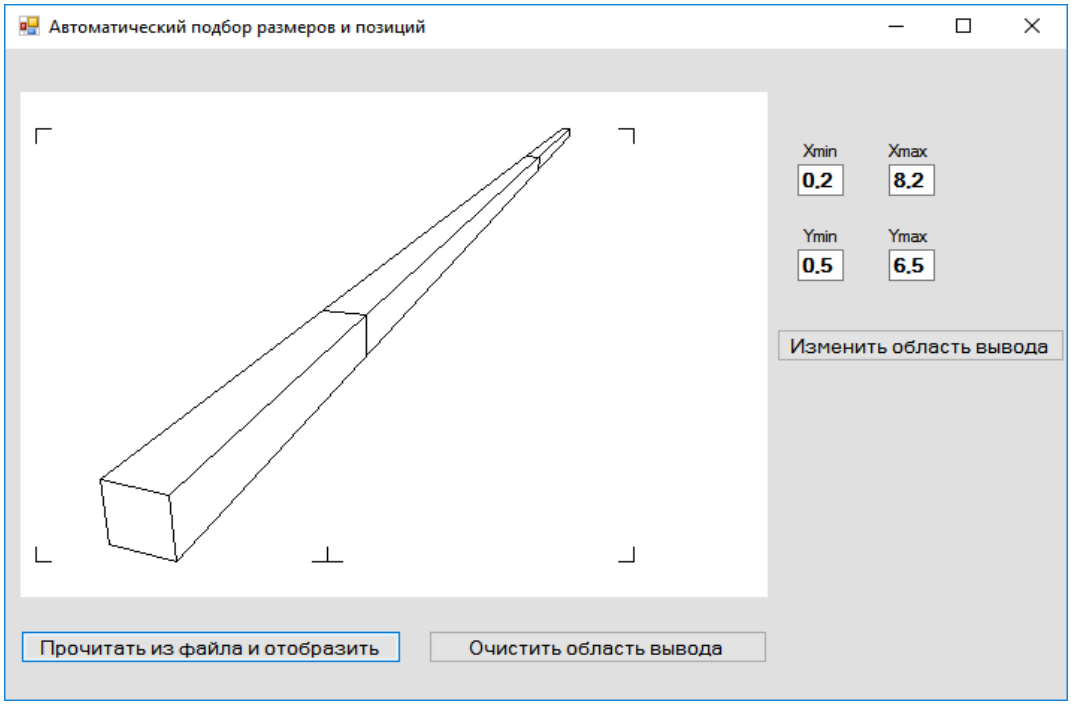


Рис. 24.11 – Результат совместной работы программ *GPersF* и *GenPlot*

Другой аспект, требующий специального рассмотрения, – это представление вертикальных линий. На рис. 23.11 мы видели, что спроецированные вертикальные линии встречаются в точке схода F. Мы также сравнивали представления кубиков. Если точка O расположена в центре кубика, то наша программа не сможет сформировать изображение рис. 23.12(б). Но точку O можно поместить над кубиком так, чтобы направление наблюдения EO было горизонтальным. Тогда экран будет занимать вертикальное положение, поскольку он перпендикулярен направлению наблюдения. Это означает, что вертикальные ребра кубика будут параллельны экрану. Следовательно, их проекции на картинке также

будут вертикальными. Курьезным является то, что, хотя линия наблюдения горизонтальная, мы будем видеть верхнюю грань кубика как бы сверху.

Получаемая картинка выглядит совершенно натурально, пока мы не преувеличим и не выберем точку наблюдения слишком высоко. На рис. 24.12 показаны оба таких случая.

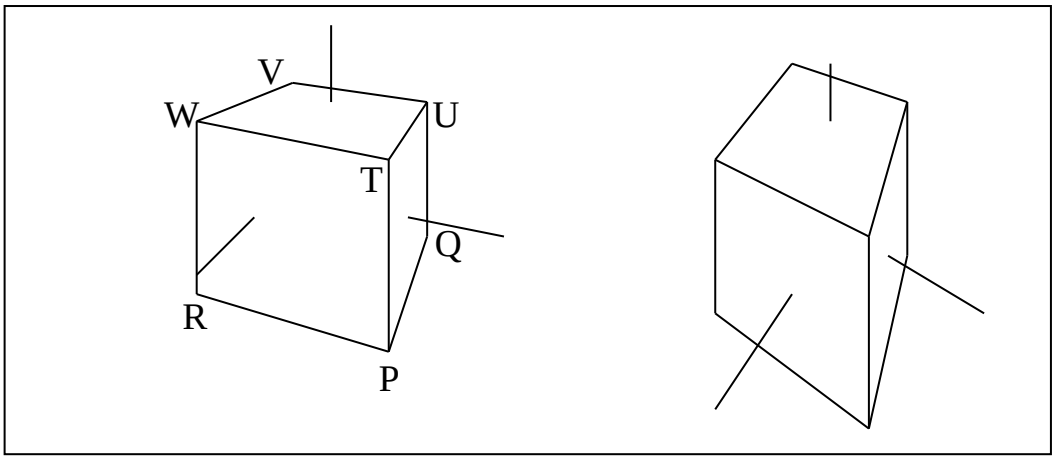


Рис. 24.12 – Зависимость вида кубика от выбора местоположения объектной точки:
а – объектная точка несколько выше кубика;
б – объектная точка очень высоко над кубиком

Линии на рис. 24.12(а) были вычерчены программами *GPersF* и *GenPlot* на основе следующего списка входных данных файла:

0 0 2 (точка O)
5 30 90 (ро, teta, fi)
1 1 –1 0 (перемещение в точку P)
–1 1 –1 1 (вычерчивание отрезка PQ)
–1 1 1 1 (вычерчивание отрезка QU)
1 1 1 1 (вычерчивание отрезка UT)
1 1 –1 1 (вычерчивание отрезка TR)
1 –1 –1 1 (вычерчивание отрезка PR)
1 –1 1 1 (вычерчивание отрезка RW)
1 1 1 1 (вычерчивание отрезка WT)
1 –1 1 0 (перемещение в точку W)

–1 –1 1 1 (вычерчивание отрезка WV)

–1 1 1 1 (вычерчивание отрезка VU)

2 0 0 0 (ось x)

1 0 0 1

0 2 0 0 (ось y)

0 1 0 1

0 0 2 0 (ось z)

0 0 1 1

Рис. 24.12(б) получен на этом же наборе данных путем выбора точки O значительно выше, а именно – с координатами (0, 0, 6) вместо (0, 0, 2). Это уже совсем нереалистичное изображение кубика. В то же время рис. 24.12(а) вполне приемлем. Некоторые даже предпочитают его по сравнению с кубиками на рис. 23.2(а) и рис. 23.12(а), где вертикальные линии имеют точку схода. Если кубик вычерчивается в перспективе вручную, то обычно вертикальные линии остаются вертикальными, как на рис. 24.12 (а). Другие представления выполнить вручную уже значительно труднее. С помощью же компьютера их выполнить просто, и пользователь может выбрать такие, какие ему нравятся.

На рис. 23.13 показано перспективное изображение, сформированное программой *GenPlot* на основе файла *Scratch*, созданного программой *GPersF* из вышеприведенного набора данных, где координаты точки O – (0, 0, 2), – несколько выше кубика.

На рис. 23.14 показано перспективное изображение, сформированное программой *GenPlot* на основе файла *Scratch*, созданного программой *GPersF* из вышеприведенного набора данных, где координаты точки O – (0, 0, 6), – высоко над кубиком.

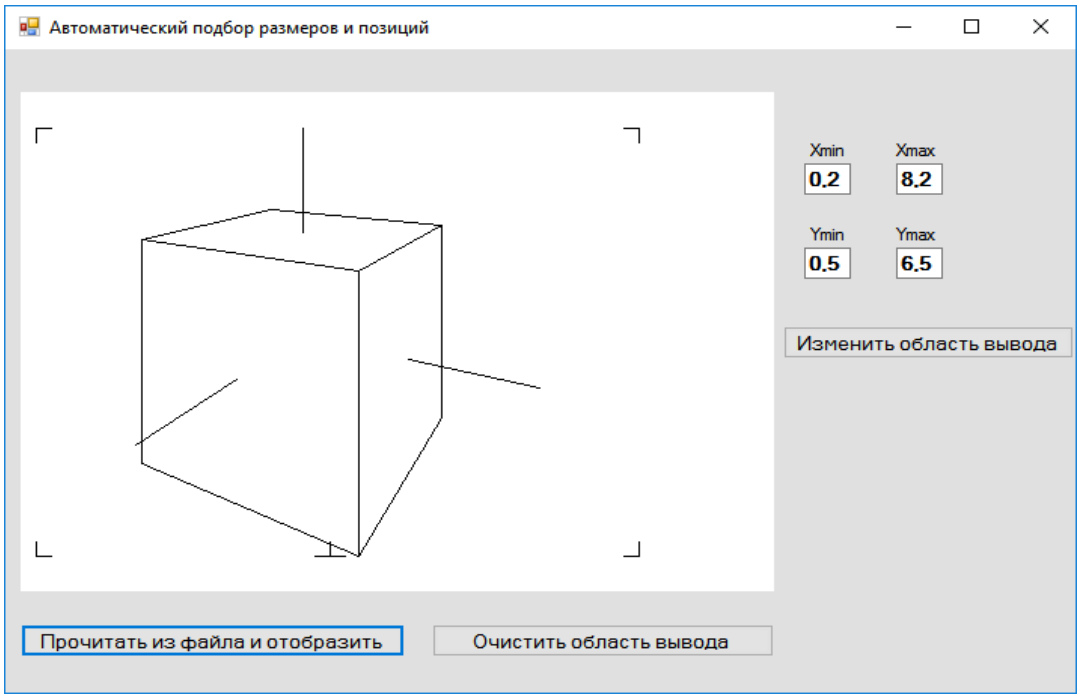


Рис. 24.13 – Объектная точка O несколько выше кубика

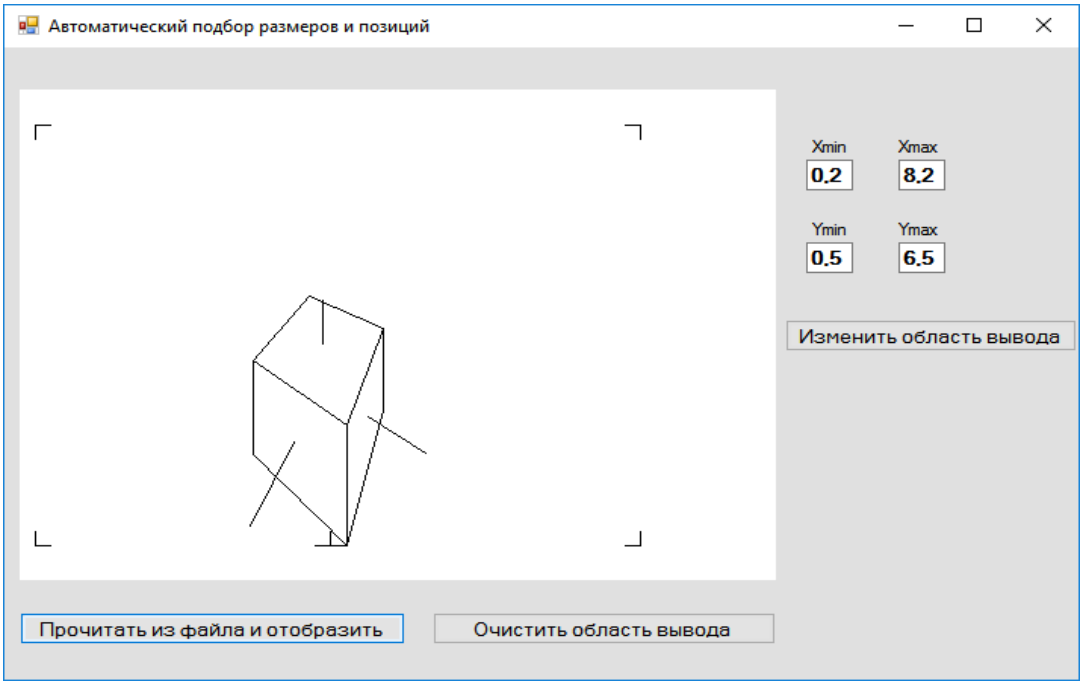


Рис. 24.14 – Объектная точка O очень высоко над кубиком

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Аммерал Л. Принципы программирования в машинной графике / Л. Аммерал. – М .: Сол Систем, 1992. – 224 с.

2. Сиденко Л.А. Компьютерная графика и геометрическое моделирование [Текст]: Учебное пособие / Л.А. Сиденко. – СПб.: Питер, 2009 г. – 224 с.

3. Аммерал Л. Интерактивная трехмерная машинная графика / Л. Аммерал. – М.: Сол Систем, 1992. – 317 с.

4. Фролов А.В. Визуальное проектирование приложений C# / А. В. Фролов, Г. В. Фролов. М.: КУДИЦ-ОБРАЗ, 2003. – 512 с.

СПИСОК ЭЛЕКТРОННЫХ РЕСУРСОВ

1. Л. Аммерал. Принципы программирования в машинной графике [Электрон. ресурс] – Режим доступа: http://bsuir-helper.ru/sites/default/files/2010/07/03/met/Ammeral_Principy_programmirovaniya_v_mashinnoy_grafike.djvu – Загл. с экрана.

2. А. В. Фролов, Г. В. Фролов Визуальное проектирование приложений C# [Электрон. ресурс] – Режим доступа: http://www.frolov-lib.ru/books/msnet/c_sharp2/ch10.html. – Загл. с экрана.

3. П. В. Вельтмандер. Учебное пособие «Основные алгоритмы компьютерной графики» [Электронный ресурс] – Режим доступа: http://ermak.cs.nstu.ru/kg_rivs/. – Загл. с экрана.