



МИНОБРНАУКИ РОССИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
«Самарский государственный технический университет»

**Кафедра «Вычислительная техника»**

# **ГРАФИЧЕСКИЕ СИСТЕМЫ КОМПЬЮТЕРОВ**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ  
К ЛАБОРАТОРНЫМ РАБОТАМ И КУРСОВОМУ ПРОЕКТИРОВАНИЮ

Самара 2013

Составитель: А.И. Пугачев

УДК 681.3

**ГРАФИЧЕСКИЕ СИСТЕМЫ КОМПЬЮТЕРОВ:** Методические указания к лабораторным работам и курсовому проектированию / Самар. гос. техн. ун-т; Сост. *А.И. Пугачев*. Самара, 2013. – 74 с.: ил.

Изложен теоретический материал, необходимый при подготовке к лабораторным работам и выполнении курсовой работы по дисциплине.

Приведены темы лабораторных работ, порядок их выполнения и сдачи. Изложены требования к выполнению курсовой работы, техническое задание. Даны варианты индивидуальных заданий на выполнение курсовой работы, а также практические рекомендации по программной реализации изучаемых методов.

Методические указания предназначены для студентов, обучающихся по программе бакалавриата по направлениям 230100 – Информатика и вычислительная техника и 231000 – Программная инженерия.

# 1. ЭЛЕМЕНТЫ ТЕОРИИ КОМПЬЮТЕРНОЙ ГРАФИКИ

## 1.1. Визуализация отрезков прямых

Пусть в области вывода с целыми координатами требуется построить отрезок, заданный начальной точкой  $P_1$  с координатами  $(x_1, y_1)$  и конечной  $P_2$  с координатами  $(x_2, y_2)$ , т.е. нужно изобразить его с наименьшей погрешностью в виде непрерывной последовательности пикселей (рис. 1.1). Очевидно, что все промежуточные точки отрезка также должны иметь целые координаты. Далее для определенности будем считать, что целые координаты области вывода соответствуют центрам пикселей.

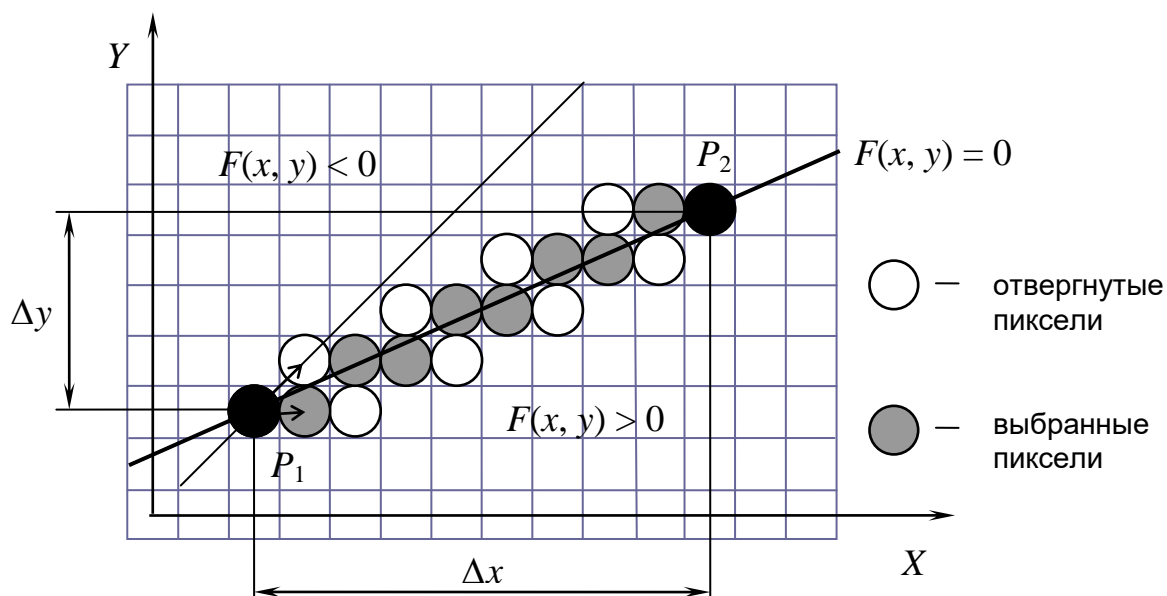


Рис. 1.1

Уравнение прямой, проходящей через точки  $P_1$  и  $P_2$ , имеет вид:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}. \quad (1.1)$$

Обозначив  $x_2 - x_1 = \Delta x$ ,  $y_2 - y_1 = \Delta y$ , представим его следующим образом:

$$\Delta y(x - x_1) - \Delta x(y - y_1) = 0. \quad (1.2)$$

Функцию  $F(x, y) = \Delta y(x - x_1) - \Delta x(y - y_1)$  будем называть оценочной, поскольку ее значение характеризует степень отклонения всякой точки  $(x, y)$  от прямой  $P_1P_2$ . Действительно, в соответствии с уравнением (1.2), для всех точек прямой  $F(x, y) = 0$ , для всех точек справа от вектора  $P_1P_2$   $F(x, y) > 0$ , а для всех точек слева  $F(x, y) < 0$  (рис. 1.1). Причем абсолютная величина  $F(x, y)$  возрастает при удалении точки  $(x, y)$  от прямой прямо пропорционально расстоянию и поэтому может использоваться в качестве меры отклонения оцениваемых точек от строящегося отрезка. Это позволяет вместо непосредственного расчета координат точек отрезка по уравнению прямой линии последовательно выбирать очередные точки из ближайших на основе сравнения значений  $F(x, y)$ .

Начнем построение отрезка, взяв за начальное значение текущей точки  $P$  точку  $P_1$ . Будем двигаться от нее по дискретной плоскости к конечной точке  $P_2$ , стремясь отклоняться от прямой  $P_1P_2$  в наименьшей степени. Направление движения и значения элементарных шагов  $s_x$  и  $s_y$  по осям зависят от знаков приращений  $\Delta x$  и  $\Delta y$ . Если  $\Delta x > 0$ , то  $s_x = 1$ . Если же  $\Delta x < 0$ , то  $s_x = -1$ . Следовательно,  $s_x = \text{sign}(\Delta x)$ . Аналогично, элементарным шагом по координате  $y$  будет  $s_y = \text{sign}(\Delta y)$ .

Любая точка на дискретной плоскости имеет 8 соседних. Если учитывать направление движения от  $P_1$  к  $P_2$ , то для выбора очередной точки вместо 8-ми достаточно рассматривать только 3 соседние точки. Чтобы еще сократить выбор рассмотрим частный случай, когда  $|\Delta x| \geq |\Delta y|$ . Ему соответствуют отрезки с углом наклона к оси  $Ox$ , меньшим или равным  $45^\circ$ . Тогда новым значением текущей точки  $P$

может быть либо точка  $(x + s_x, y)$  либо  $(x + s_x, y + s_y)$ . Для выбора лучшей из них используем оценочную функцию.

Так как за начальное значение текущей точки  $P$  принята точка  $P_1$ , лежащая на прямой  $P_1P_2$ , то начальное значение  $F(x, y) = 0$ . Для точки  $(x + s_x, y)$ :

$$F(x + s_x, y) = \Delta y(x + s_x - x_1) - \Delta x(y - y_1) = F(x, y) + s_x \Delta y; \quad (1.3)$$

Для точки  $(x + s_x, y + s_y)$ :

$$\begin{aligned} F(x + s_x, y + s_y) &= \Delta y(x + s_x - x_1) - \Delta x(y + s_y - y_1) = \\ &= F(x + s_x, y) - s_y \Delta x. \end{aligned} \quad (1.4)$$

Рекурсивный вид формул (1.3) и (1.4) позволяет заметно сократить вычисления по сравнению с непосредственным вычислением  $F(x, y)$ . Поскольку  $\Delta x, \Delta y, s_x, s_y$  постоянные величины, то расчеты по формулам (1.3), (1.4) можно еще сократить, если заранее вычислить  $\Delta F_x = s_x \Delta y$  и  $\Delta F_y = s_y \Delta x$ . Тогда

$$F(x + s_x, y) = F(x, y) + \Delta F_x; \quad (1.5)$$

$$F(x + s_x, y + s_y) = F(x + s_x, y) - \Delta F_y. \quad (1.6)$$

Если  $|F(x + s_x, y)| < |F(x + s_x, y + s_y)|$ , значит точка  $(x + s_x, y)$  лежит ближе к прямой  $P_1P_2$ , чем точка  $(x + s_x, y + s_y)$  и ее следует принять за новое положение текущей точки  $P$ . Иначе новым значением  $P$  должна стать точка  $(x + s_x, y + s_y)$ . Далее процесс построения надо повторять до тех пор, пока текущая точка  $P$  не достигнет конечной  $P_2$ .

В итоге алгоритм визуализации отрезка для случая  $|\Delta x| \geq |\Delta y|$  можно представить следующим образом.

1. Установить цвет рисования *Color*.
2. Вычислить:
  - $\Delta x = x_2 - x_1; \quad \Delta y = y_2 - y_1;$
  - $s_x = \text{sign}(\Delta x); \quad s_y = \text{sign}(\Delta y);$
  - Если  $s_x > 0$ , тогда  $\Delta F_x = \Delta y$ , иначе  $\Delta F_x = -\Delta y$ .
  - Если  $s_y > 0$ , тогда  $\Delta F_y = \Delta x$ , иначе  $\Delta F_y = -\Delta x$ .
  - $x = x_1; \quad y = y_1; \quad //$  начальные координаты текущей точки
  - $F = 0. \quad //$  начальное значение оценочной функции
3. Вывести пиксель с координатами  $(x, y)$  цветом *Color*.

4. Если  $x = x_2$ , то – конец алгоритма.
5. Вычислить:  
 $Fx = F + \Delta Fx$ ;  $F = Fx - \Delta Fy$ ;  $x = x + s_x$ .
6. Если  $|Fx| < |F|$ , то вычислить  $F = Fx$ ,  
 иначе – вычислить  $y = y + s_y$ .
7. Перейти к п. 3.

В алгоритме переменная  $F$  обозначает текущее значение оценочной функции, а  $Fx$  – новое ее значение при шаге по  $x$ . Для сокращения вычислений новое значение оценочной функции при одновременном шаге по  $x$  и  $y$  также обозначается через  $F$ .

Вторую часть алгоритма для построения отрезков с наклоном к оси  $OX$  более  $45^\circ$  легко разработать по аналогии.

## 1.2. Сплайны

Сплайнами называются полиномиальные параметрические кривые  $P(t) = [x(t) \quad y(t)]$ . В качестве базисных функций в сплайнах используют полиномы Лежандра, Ньютона, Эрмита, Бернштейна [3, 5].

Кубическими сплайнами называются параметрические кривые, для которых в качестве базисных функций используются полиномы 3-й степени вида  $at^3 + bt^2 + ct + d$ . На плоскости кубический сплайн задается следующим образом:

$$P(t) = [a_x t^3 + b_x t^2 + c_x t + d_x \quad a_y t^3 + b_y t^2 + c_y t + d_y] \quad (1.7)$$

Параметр  $t$  изменяется в пределах  $0 \leq t \leq 1$ . Следовательно, начальной точкой сплайна будет  $P(0)$ , а конечной – точка  $P(1)$  (рис. 1.2).

Для однозначного определения кубического сплайна в форме Эрмита [1, 4] необходимо задать два вида начальных условий:

- начальную  $T_1$  и конечную  $T_2$  точки;
- касательные векторы  $T'_1$  и  $T'_2$  к кривой в начальной и конечной точках.

Далее для удобства матрицу-строку  $[x \quad y]$  координат  $(x, y)$  какой-либо точки  $T$  или вектора  $T'$  будем обозначать так же, как и саму точ-

ку или вектор. Напомним также, что координатами вектора являются приращения координат по соответствующим осям между его конечной и начальной точками.

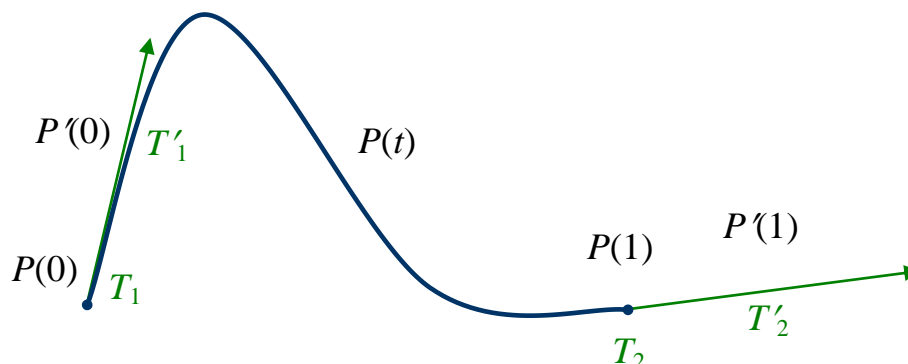


Рис. 1.2

Представим (1.7) в следующей форме:

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \times \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \times L, \quad (1.8)$$

где  $L$  – матрица неизвестных коэффициентов полиномов для  $x(t)$  и  $y(t)$ .

Первое начальное условие дает следующие соотношения:

$$P(0) = T_1; \quad P(1) = T_2. \quad (1.9)$$

Используя (1.8), запишем их в матричной форме:

$$P(0) = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \times L = T_1; \quad (1.10)$$

$$P(1) = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \times L = T_2. \quad (1.11)$$

Второе начальное условие дает дополнительные соотношения:

$$P'(0) = T'_1; \quad P'(1) = T'_2. \quad (1.12)$$

Касательный вектор к кривой  $P(t)$  находится как производная  $P'(t)$ . В данном случае

$$P'(t) = \begin{bmatrix} \frac{dx}{dt} & \frac{dy}{dt} \end{bmatrix} = \begin{bmatrix} 3a_x t^2 + 2b_x t + c_x & 3a_y t^2 + 2b_y t + c_y \end{bmatrix}. \quad (1.13)$$

Запишем  $P'(t)$  в форме, аналогичной (1.8):

$$P'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \times L. \quad (1.14)$$

Перепишем условия (1.12), используя (1.14):

$$P'(0) = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \times L = T'_1; \quad (1.15)$$

$$P'(1) = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} \times L = T'_2. \quad (1.16)$$

Объединим все начальные условия (1.10), (1.11), (1.15), (1.16) в единое матричное уравнение с неизвестной матрицей  $L$

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \times L = \begin{bmatrix} T_1 \\ T_2 \\ T'_1 \\ T'_2 \end{bmatrix}, \quad (1.17)$$

на основании которого  $L$  можно найти так:

$$L = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} T_1 \\ T_2 \\ T'_1 \\ T'_2 \end{bmatrix} = M_h \times G_h. \quad (1.18)$$

Матрица  $M_h$  называется эрмитовой, а вектор  $G_h$  с начальными условиями – геометрическим вектором Эрмита [1]. Несмотря на это общепринятое название  $G_h$ , следует помнить, что каждый элемент  $G_h$  представляет собой строку из двух элементов.

После расчета коэффициентов матрицы  $L$  сплайн можно визуализировать, используя формулу (1.7).

Другим популярным видом сплайнов являются кривые Безье [1, 4]. Параметрическая кривая Безье задается списком точек  $Lp = (T_0, T_1, \dots, T_n)$ . Кривая должна проходить только через начальную



и конечную точки списка. Остальные точки влияют на форму кривой (рис. 1.3). Количество точек в списке может быть любым, большим 1.

Кривая Безье описывается в виде функции полиномиальной интерполяции между начальной  $T_0$  и конечной  $T_n$  точками списка  $Lp$ . При этом в качестве базисных функций используются интерполяционные полиномы Бернштейна:

$$J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad (1.19)$$

где  $\binom{n}{i} = \frac{n!}{i!(n-i)!}$ ,  $0 \leq i \leq n$ ,  $n$  – степень полинома.

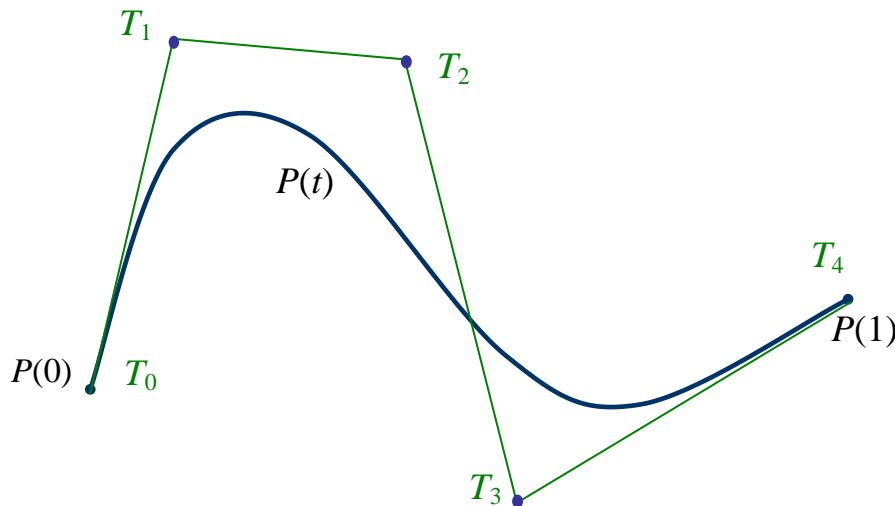


Рис. 1.3

Сама кривая Безье описывается через базисные функции и начальные условия следующим образом:

$$P(t) = \sum_{i=0}^n T_i J_{n,i}(t), \quad 0 \leq t \leq 1 \quad (1.20)$$

Здесь  $T_i = [x_i \ y_i]$  – координаты  $i$ -той точки списка  $Lp$ . Отметим, что степень  $n$  полиномов в  $P(t)$  на единицу меньше числа вершин.

Ниже приведен простейший алгоритм визуализации кривой Безье с постоянным шагом табуляции по  $t$ .

1. Установить цвет рисования *Color*.
2. Вычислить:
 
$$nFact = Factorial(n);$$

$$dt = Const; \quad t = dt;$$

$$xPred = Lp[0].x; \quad yPred = Lp[0].y.$$
3. Пока  $t < 1 + dt/2$  выполнить п.п. 3.1. – 3.4.
  - 3.1. Вычислить:
 
$$xt = 0; \quad yt = 0;$$

$$i = 0.$$
  - 3.2. Пока  $i \leq n$  выполнить п. 3.2.1.
    - 3.2.1. Вычислить
 
$$J = t^i * (1 - t)^{n-i} * nFact / (Factorial(i) * Factorial(n-i));$$

$$xt = xt + Lp[i].x * J;$$

$$yt = yt + Lp[i].y * J;$$

$$i = i + 1.$$
  - 3.3. Вывести отрезок (*xPred*, *yPred*, *xt*, *yt*) цветом *Color*.
  - 3.4. Вычислить:
 
$$t = t + dt; \quad xPred = xt; \quad yPred = yt.$$

В алгоритме предполагается, что список *Lp* организован как массив, а для вычисления факториала используется ранее определенная функция *Factorial*.

### 1.3. Алгоритмы закрашивания многоугольников

Всякая ограниченная область задается описанием своей границы. Если граница содержит криволинейные участки, то при визуализации их предварительно аппроксимируют ломаными линиями. Поэтому можно ограничиться рассмотрением методов закрашивания многоугольных областей.

Граничный многоугольник области задается упорядоченным списком вершин  $Pg = (P_1, P_2, \dots, P_n)$ , который описывает замкнутую ломаную линию. Для полноты формального описания многоугольной области важно также задать правило, определяющее то, с какой стороны от граничной линии располагаются ближайшие внутренние точки области.

Одно из наиболее простых и понятных правил определения внутренних точек области можно сформулировать следующим образом: всякая точка плоскости является внутренней точкой многоугольника,

если луч, проведенный из нее в любом направлении, пересекает границу этого многоугольника нечетное число раз (рис. 1.4). Число же точек пересечения всякой прямой линии с границей будет четным. Причем при движении вдоль прямой будут чередоваться точки входа в многоугольник и выхода из него.

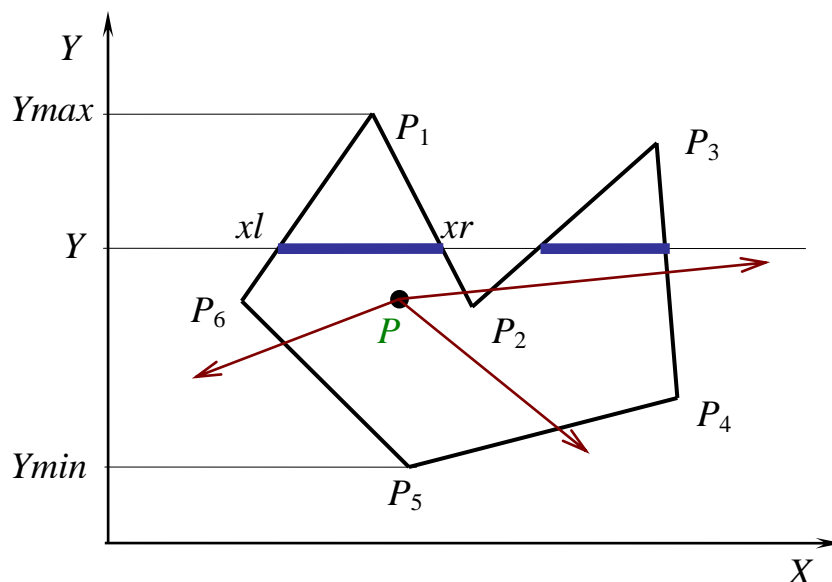


Рис. 1.4

Сказанное справедливо и для горизонтальных строк растра ( $Y = \text{Const}$ ). Поэтому закрашивание многоугольника можно выполнять построчно в пределах от ординаты  $Y_{\min}$  самой нижней вершины многоугольника до ординаты  $Y_{\max}$  самой верхней его вершины. Если список точек пересечения сторон многоугольника со строкой  $Y$  упорядочить по возрастанию или убыванию координаты  $x$ , то каждая очередная пара точек в списке будет определять границы  $(xl, Y)$  и  $(xr, Y)$  закрашиваемого сегмента строки (рис. 1.4).

Расчет границ  $Y_{\min}$  и  $Y_{\max}$  закрашивания по  $Y$  позволяет сократить число рассматриваемых строк. Однако, по разным причинам, например, за счет масштабирования многоугольник может частично или полностью выйти по  $Y$  за пределы интервала  $[0..Y_{\max}]$ , где  $Y_{\max}$  – максимальное значение координаты  $y$  области вывода. Чтобы

исключить безрезультатные вычисления в этих случаях, найденные значения следует скорректировать так:

$$Y_{min} = \max(Y_{min}, 0); \quad (1.21)$$

$$Y_{max} = \min(Y_{max}, Y_{etax}). \quad (1.22)$$

При реализации метода следует учесть случаи, требующие особой обработки. Во-первых, это вершины многоугольника. В каждой вершине сопрягаются две смежные стороны. Если при закрашивании учитывать все точки пересечения сторон со строками, не исключая, начальную и конечную, то каждой вершине будут соответствовать две совпадающие границы сегментов, что во многих случаях, может привести к неправильному закрашиванию. Вторым особым случаем закрашивания являются горизонтальные стороны ( $\Delta y = 0$ ) граничного многоугольника, поскольку они не пересекают строки раstra, а лежат на них.

На рис. 1.5 приведен пример граничного контура многоугольника, на котором цифрами отмечены особые случаи: 1 – это вершины, которые должны обрабатываться как одна граница сегмента; 2 – вершины, которые следует рассматривать как две совпадающие границы; 3 – граничные вершины горизонтальных сторон.

Если для всякой стороны  $P_i P_{i+1}$  приращение по  $y$  рассчитывать как

$$\Delta y = y_{i+1} - y_i, \quad (1.23)$$

то случаи 1 и 2 легко различать между собой по знаку приращений  $\Delta y$  смежных сторон, примыкающих к вершинам. В случае 1 они совпадают, а в случае 2 – противоположны.

Для корректной обработки особых случаев примем следующее правило: если сторона многоугольника направлена вверх ( $\Delta y > 0$ ), то из расчета точек пересечения со строками будем исключать ее начальную вершину, и наоборот, если сторона многоугольника направлена вниз ( $\Delta y < 0$ ), то из расчета точек пересечения со строками будем исключать ее конечную вершину.

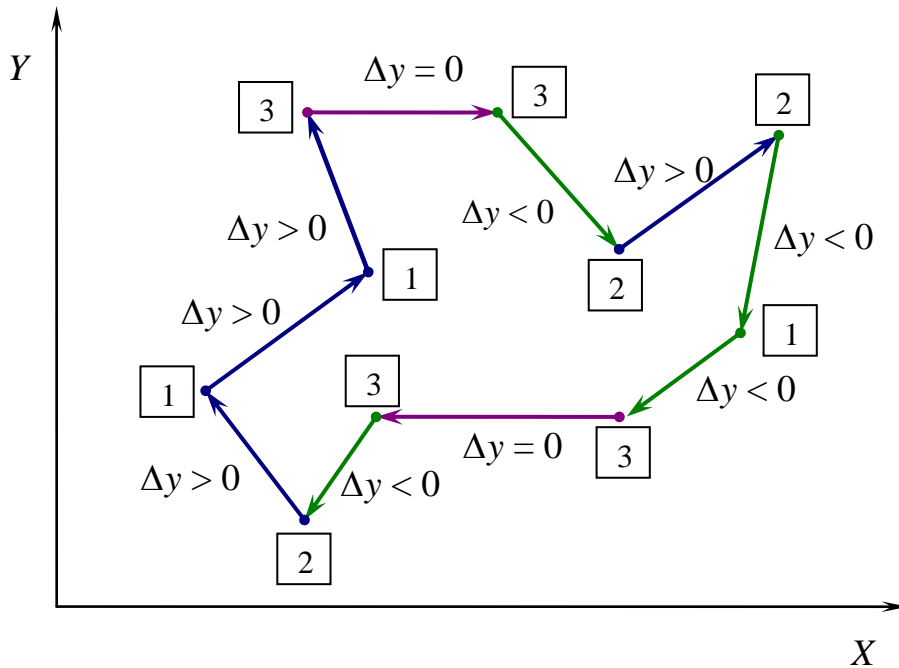


Рис. 1.5

В остальных случаях для каждой стороны нужно установить пересекается ли она с текущей строкой и если – да, то вычислить координату  $x$  точки пересечения. Пусть для очередной стороны многоугольника  $y_i$  – координата  $y$  начальной вершины, а  $y_k$  – координата  $y$  ее конечной вершины. Тогда критерием пересечения стороны со строкой  $Y$  будет выполнение следующего условия:

$$((y_i < Y) \text{ и } (y_k \geq Y)) \text{ или } ((y_i \geq Y) \text{ и } (y_k < Y)). \quad (1.24)$$

Легко установить, что помимо сторон, не пересекающихся со строкой  $Y$ , данное условие блокирует и обработку горизонтальных сторон, а также, начальных вершин для сторон, направленных вверх, и конечных вершин для сторон, направленных вниз, т. е. обеспечивает корректное вычисление границ сегментов в особых случаях.

С учетом сделанных выводов алгоритм закрашивания будет иметь вид, приведенный ниже.

1. Последовательно просматривая список вершин  $(P_1, P_2, \dots, P_n)$ , найти границы сканирования  $Y_{min}$  и  $Y_{max}$ .
2. Вычислить  $Y_{min} = \max(Y_{min}, 0)$ ;  $Y_{max} = \min(Y_{max}, Y_{max})$ .

3. Для каждой строки  $Y$  из  $[Y_{min} . . Y_{max}]$  выполнить п.п. 3.1. – 3.4.
  - 3.1. Очистить список  $Xb$ .
  - 3.2. Для всех  $i$  из  $[1 . . n]$  выполнить п. 3.2.1. – 3.2.2.
    - 3.2.1. Если  $i < n$ , то  $k = i + 1$ , иначе  $k = 1$ .
    - 3.2.2. Если  $((y_i < Y) \text{ и } (y_k \geq Y))$  или  $((y_i \geq Y) \text{ и } (y_k < Y))$ , то вычислить координату  $x$  точки пересечения стороны  $P_i P_k$  со строкой  $Y$  и записать в  $Xb$ .
  - 3.3. Отсортировать список  $Xb$  по возрастанию.
  - 3.4. Последовательно беря из списка  $Xb$  пары  $x_l, x_r$ , закрасить соответствующие им сегменты строки  $Y$ .

В алгоритме для каждой строки  $Y$  сначала очищается список  $Xb$  границ сегментов, а затем циклически рассматриваются все стороны  $P_i P_k$  многоугольника,  $i$  – индекс текущей, а  $k$  – индекс последующей вершины. Учитывается, что за вершиной  $P_n$  следует вершина  $P_1$ . В цикле производится тестирование сторон  $P_i P_k$  многоугольника по критерию (1.24). Для сторон, удовлетворяющих этому условию, вычисляется координата  $x$  точки пересечения стороны со строкой и записывается в  $Xb$ .

Когда обработка списка вершин заканчивается, список  $Xb$  сортируется по возрастанию. Сортировка гарантирует, что после этого левые и правые границы сегментов при движении от начала к концу списка будут чередоваться. Поэтому на заключительном этапе для закрашивания сегментов строки из  $Xb$  последовательно берутся пары элементов.

У данного алгоритма есть некоторые особенности, показанные на рис. 1.6. Во-первых, вершины локального максимума по  $Y$  будут всегда закрашиваться, а вершины локального минимума – нет. Это – следствие принятого критерия (1.24) к обработке сторон. На рис. 1.6 квадратиками отмечены места расположения вершин многоугольника. Вершины заданы в центрах квадратиков.

Во-вторых, как видно из того же рисунка, тонкие слабо наклонные к оси  $OX$  выступы многоугольника могут закрашиваться с разрывами. Здесь причина в построчном способе закрашивания. Максимальная погрешность в расчете границ сегментов из-за их округления составляет 0,5 пикселя, но погрешность в изображении сторон по длине в таких случаях может стать значительно большей. Чтобы ви-

The figure shows a scatter plot of data points (black dots) and a fitted regression line (solid red line). A dashed red rectangle represents the confidence interval for the regression line. The axes are labeled X and Y. The data points are clustered around the origin, and the regression line passes through the center of the cluster. The confidence interval is wider at the ends of the line and narrower in the middle.

Для определения внутренних точек закрашиваемой многоугольной области можно применить другое, более формальное и более универсальное правило, основанное на использовании понятия ориентации граничного контура [4]. Если исходить из того, что порядок вершин в списке  $Pg = (P_1, P_2, \dots, P_n)$  определяет и порядок построения сторон многоугольника, то каждую сторону  $P_i P_{i+1}$  можно рассматривать как вектор, направленный из  $P_i$  в  $P_{i+1}$ . Тогда правило определения внутренних точек области сформулируем следующим образом: при движении от начала всякого вектора граничного конту-

ра к его концу ближайšie внутренние точки области расположены слева от него (рис. 1.7).

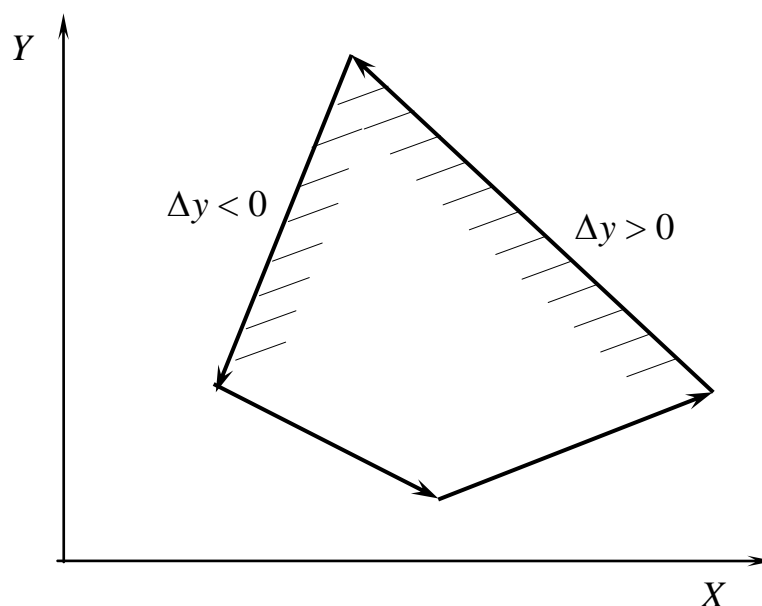


Рис. 1.7

Из этого правила следует, что при обходе вершин против часовой стрелки, внутренние точки области лежат внутри многоугольника, а при обходе по часовой стрелке – вне многоугольника. В последнем случае должна закрашиваться вся область вывода, кроме точек внутри многоугольника.

В соответствии со сказанным граничный многоугольник будем считать ориентированным по часовой стрелке или против, в зависимости от заданного в списке  $Pg = (P_1, P_2, \dots, P_n)$  направления обхода его вершин. Использование понятия ориентации для границы многоугольника открывает новые возможности.

Как видно из рис. 1.7, стороны многоугольника, направленные вверх (приращение  $\Delta y > 0$ ), являются правыми границами, а стороны, направленные вниз ( $\Delta y < 0$ ), – левыми границами закрашиваемых сегментов строки. Поэтому точки пересечения сторон с очередной строкой по знаку  $\Delta y$  можно сразу классифицировать как левые или правые границы сегментов. В связи с этим вместо общего списка  $Xb$  границ сегментов строки здесь удобнее формировать отдельно список



$Xl$  левых и список  $Xr$  правых границ. Очевидно, что очередную координату  $x$  следует заносить в список  $Xr$ , когда  $\Delta y > 0$ , и в список  $Xl$  – в противном случае. Когда все точки пересечения сторон многоугольника со строкой будут найдены, количество элементов в списках  $Xl$  и  $Xr$  должно быть одинаковым.

Если граничный многоугольник ориентирован так, что должна закрашиваться внешняя область (по часовой стрелке), то все строки, не имеющие точек пересечения с многоугольником, должны закрашиваться полностью (рис. 1.8). Для остальных строк список  $Xl$  не будет содержать левой границы первого закрашиваемого сегмента, а список  $Xr$  не будет содержать правой границы последнего сегмента строки. В этих случаях для закрашивания в пределах области вывода в качестве недостающей левой границы первого сегмента следует взять левую границу области вывода, а в качестве недостающей правой границы последнего сегмента строки – ее правую границу.

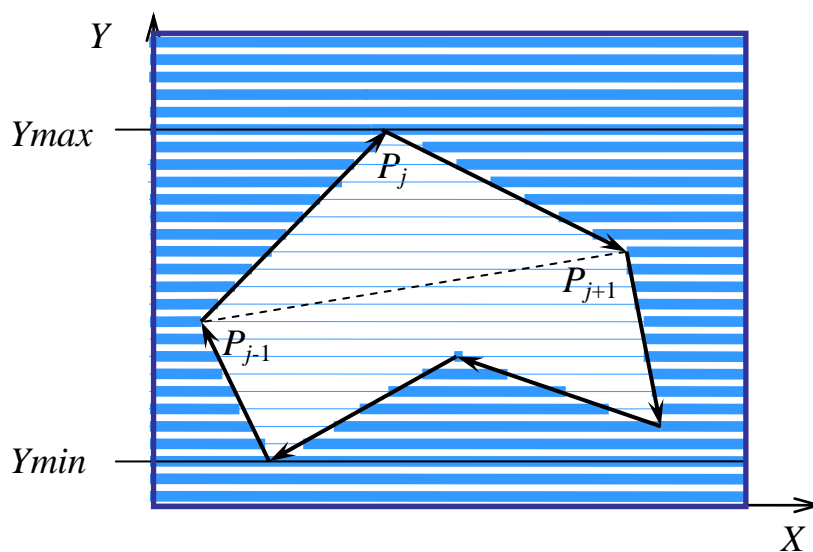


Рис. 1.8

Ориентацию (по часовой стрелке или против) граничного многоугольника можно определить аналитически. Для этого не требуется анализ взаимного расположения всех вершин. Ориентация граничного многоугольника совпадает с ориентацией невырожденного треугольника, заданного самой высшей или самой низшей вершиной

многоугольника и двумя смежными с ней вершинами. Для определенности найдем в списке  $Pg$  наивысшую вершину  $P_j$  и две смежные с ней вершины  $P_{j-1}$  и  $P_{j+1}$  (рис. 1.8).

Очевидно, что для расположения осей координат как на рис. 2.13 ориентация треугольника  $P_{j-1} P_j P_{j+1}$  будет по часовой стрелке (*clockwise direction*), если вершина  $P_{j+1}$  правее вершины  $P_{j-1}$ , то есть  $x_{j+1} > x_{j-1}$ . Иначе ориентация треугольника будет против часовой стрелки.

Если рассматривается многоугольник без самопересечений, то его ориентация совпадает с ориентацией данного треугольника. Если же контур многоугольника имеет самопересечения, то само понятие ориентации становится неприменимым.

Ниже представлен алгоритм закрашивания ориентированных многоугольников, построенный на основании изложенного.

1. Последовательно просматривая список вершин ( $P_1, P_2, \dots, P_n$ ), найти индекс  $j$  наивысшей вершины, а также границы сканирования  $Ymin$  и  $Ymax$ .
2. Вычислить  $Ymin = \max(Ymin, 0)$ ;  $Ymax = \min(Ymax, Y_{emax})$ .
3. Если  $P_{j+1}.x > P_{j-1}.x$ , то  $CW = \text{True}$ , иначе  $CW = \text{False}$ .
4. Если  $CW$ , то строки области вывода от 0 до  $Ymin$  закрасить целиком.
5. Для каждой строки  $Y$  из  $[Ymin \dots Ymax]$  выполнить п.п. 5.1. - 5.5.
  - 5.1. Очистить списки  $Xl$  и  $Xr$ .
  - 5.2. Для всех  $i$  из  $[1 \dots n]$  выполнить п. 5.2.1. – 5.2.2.
    - 5.2.1. Если  $i < n$ , то  $k = i + 1$ , иначе  $k = 1$ .
    - 5.2.2. Если  $((y_i < Y) \text{ и } (y_k \geq Y))$  или  $((y_i \geq Y) \text{ и } (y_k < Y))$ , то выполнить п.п. 5.2.2.1. – 5.2.2.2.
      - 5.2.2.1. Вычислить координату  $x$  точки пересечения стороны  $P_i P_k$  со строкой  $Y$ .
      - 5.2.2.2. Если  $P_k.y - P_i.y > 0$ , то записать  $x$  в  $Xr$ , иначе - записать  $x$  в  $Xl$ .
  - 5.3. Если  $CW$ , то дополнить список  $Xl$  левой границей области вывода, а список  $Xr$  - ее правой границей.
  - 5.4. Отсортировать списки  $Xl$  и  $Xr$  по возрастанию.
  - 5.5. Последовательно беря из списков  $Xl$  и  $Xr$  по одному элементу в качестве границ  $xl$ ,  $xr$  сегмента, закрасить все сегменты строки  $Y$ . Причем, если  $xl > xr$ , то закрашивание производить не следует.
6. Если  $S < 0$ , то строки области вывода от  $Ymax$  до  $Y_{emax}$  закрасить целиком.

Для обработки особых случаев в данном алгоритме используется тот же критерий, что и в предыдущем алгоритме. Особенностью является учет ориентации многоугольника, а также использование отдельных списков левых  $Xl$  и правых  $Xr$  границ сегментов строки.

Следует отметить, что два рассмотренных алгоритма закрашивания отличаются между собой не только правилами выбора внутренних точек. Они дают разные результаты и в том случае, когда граничный контур области имеет самопересечения. Кроме того, второй алгоритм без существенных переделок пригоден и для закрашивания многосвязных областей, т.е. областей, ограниченных несколькими граничными многоугольниками.

#### 1.4. Теоретико-множественные операции над двумерными областями

Сложные фигуры можно синтезировать из более простых, в том числе и из примитивов, с помощью теоретико-множественных операций (ТМО). Примеры операций объединения, пересечения, разности и симметрической разности приведены на рис 1.9.

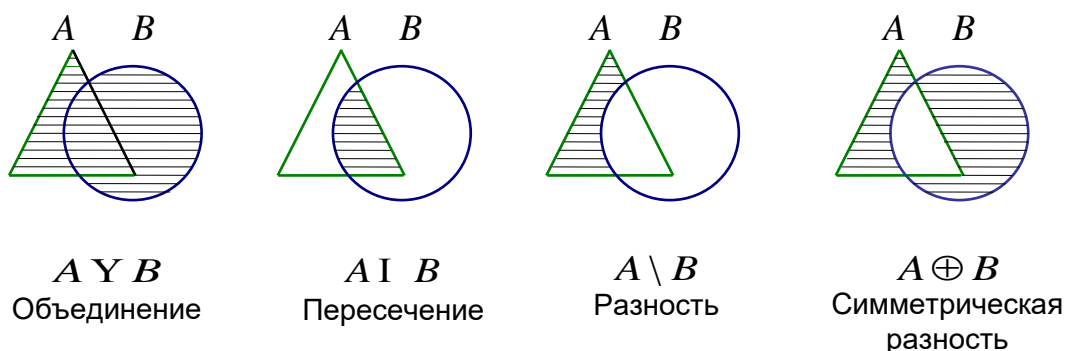


Рис. 1.9

Если имеются примитивы с криволинейными границами, то такие границы аппроксимируются ломаными линиями из отрезков прямых, поэтому в качестве примитивов будем иметь в виду многоугольники. Результатом ТМО над многоугольниками может быть как односвязная, так и многосвязная область плоскости. Поиск точного описания границ результирующей области требует сложных вычислений и тонкого анализа возможных вариантов взаимного располо-

жения границ областей [1, 2]. Если же учесть, что изображение формируется построчно, то ТМО проще выполнять над горизонтальными сечениями исходных областей непосредственно во время визуализации [4].

На рис. 1.10 показана реализация построчного выполнения ТМО над сечениями  $S_A$  и  $S_B$  исходных фигур  $A$  и  $B$  строкой  $Y$ . Переменная  $S$  обозначает сечение результирующей области, полученное с помощью ТМО над  $S_A$  и  $S_B$ . Сплошные участки сечений фигур будем называть сегментами,  $x_{Al}$ ,  $x_{Bl}$ ,  $x_{Ar}$ ,  $x_{Br}$  – левые и правые границы сегментов для  $S_A$  и  $S_B$ .

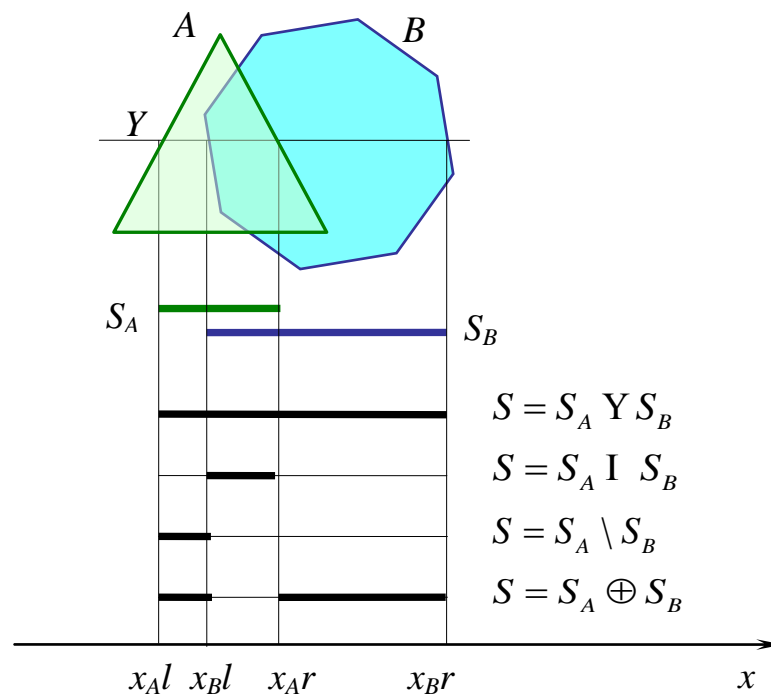


Рис. 1.10

Поставим в соответствие внутренним точкам сегментов сечения  $S$  многоугольной области функцию  $L(x)$ , которую будем называть пороговой. Определим ее следующим образом:

$$L(x) = \begin{cases} 0, & \text{если } x \notin S; \\ 1, & \text{если } x \in S. \end{cases} \quad (1.26)$$

Поведение пороговой функции иллюстрирует рис. 1.11. Как видно, внутренним точкам области соответствует значение  $L(x) = 1$ .

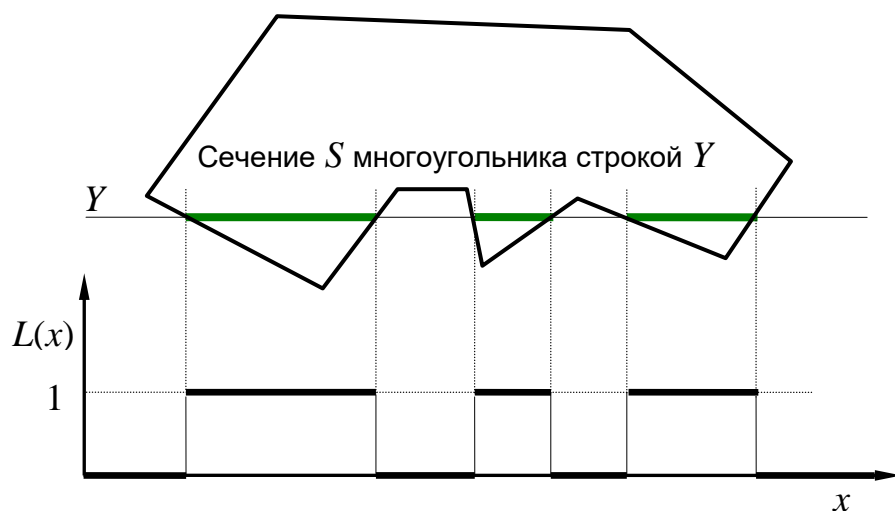


Рис. 1.11

Пусть теперь имеются сечения  $S_A$  и  $S_B$  строкой  $Y$  двух фигур  $A$  и  $B$ . Обозначим через  $L_a(x)$  и  $L_b(x)$  пороговые функции для  $S_A$  и  $S_B$ . Рассмотрим взвешенную сумму  $Q(x) = gL_a(x) + hL_b(x)$  пороговых функций (рис. 1.12) для  $g \neq h$  и  $g, h \neq 0$ .

Анализ свойств суммы  $Q(x)$  позволяет выделить области значений, соответствующие различным видам ТМО. Результат анализа приведен в табл. 1.1.

Таблица 1.1

$Q(x)$	ТМО
$\geq \min(g, h)$	Объединение
$= g + h$	Пересечение
$= g, h$	Симметрическая разность
$= g$	Разность $A \setminus B$
$= h$	Разность $B \setminus A$

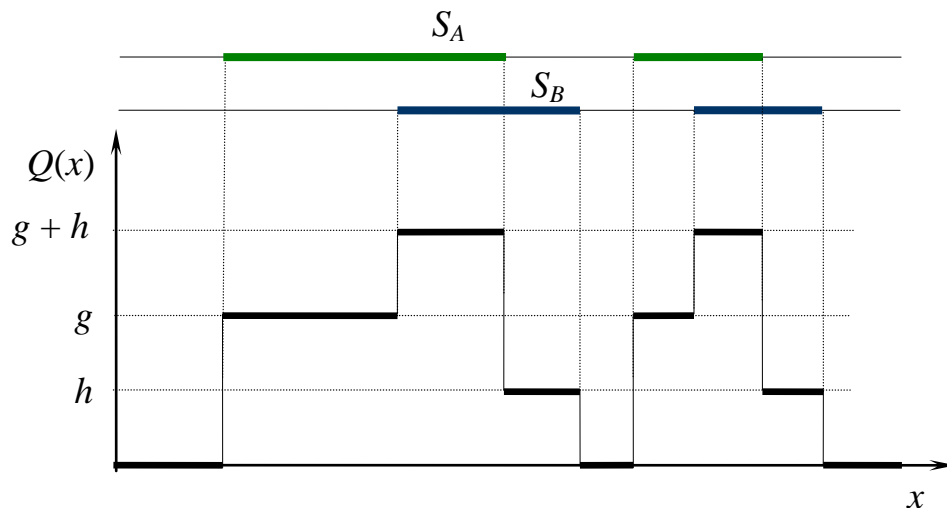


Рис. 1.12

Из табл. 1.1 видно, что рассмотренный метод применим для реализации всех основных видов ТМО.

Перейдем к разработке алгоритма выполнения ТМО над сечениями двух плоских фигур  $A$  и  $B$ . Будем считать, что в качестве исходных данных для очередной строки  $Y$  используются массивы левых  $X_{al}$  и правых  $X_{ar}$  границ сегментов сечения  $S_A$  фигуры  $A$ , и массивы левых  $X_{bl}$  и правых  $X_{br}$  границ сегментов сечения  $S_B$  фигуры  $B$ .

Закодируем все ТМО, а также примем конкретные значения весов для расчета  $Q(x)$ :  $g = 2$ ;  $h = 1$ . Тогда табл. 1.1 примет вид табл. 1.2.

Таблица 1.2

Код ТМО	$Q(x)$	ТМО
1	$\geq 1$	Объединение
2	$= 3$	Пересечение
3	$= 1, 2$	Симметрическая разность
4	$= 2$	Разность $A \setminus B$
5	$= 1$	Разность $B \setminus A$

Код заданной ТМО также будет параметром алгоритма.

Для совместной обработки исходных границ сегментов используем рабочий массив  $M$ . Элементы массива  $M$  представляют собой записи с полями  $M[i].x$  и  $M[i].dQ$ . Поле  $M[i].x$  служит для записи координаты  $x$  границы сегмента, а поле  $M[i].dQ$  – для записи соответствующего приращения пороговой функции с учетом веса операнда. Поле

$dQ$  необходимо для того, чтобы при последовательном просмотре  $M$  можно было однозначно восстановить вид функции  $Q(x)$ . Результатом работы алгоритма будут массивы  $Xrl$  и  $Xrr$  левых и правых границ сегментов сечения результирующей области  $R$  строкой  $Y$ .

Кроме перечисленных выше, в алгоритме использованы также следующие переменные:

$SetQ$  – множество значений суммы  $Q$  пороговых функций  $L$  операндов, соответствующее заданной ТМО;

$k$  и  $m$  – счетчики элементов соответственно в массивах  $Xrl$  и  $Xrr$ ;

$Xemin$  и  $Xemax$  – соответственно левая и правая границы области вывода.

Для удобства представления алгоритма используется функция  $Length(X)$ , возвращающая заполненное количество элементов в массиве  $X$ .

Алгоритм выполнения любой из перечисленных в табл. 1.2 ТМО представлен ниже.

1. Если ТМО = 1, то  $SetQ = [1, 3]$  // объединение  
 иначе если ТМО = 2, то  $SetQ = [3, 3]$  // пересечение  
 иначе если ТМО = 3, то  $SetQ = [1, 2]$  // сим. разность  
 иначе если ТМО = 4, то  $SetQ = [2, 2]$  // разность A - B  
 иначе если ТМО = 5, то  $SetQ = [1, 1]$  // разность B - A
2.  $n = Length(Xa)$ .
3. Для всех  $i$  из  $[1..n]$  выполнить  $M[i].x = Xa[i]$ ;  $M[i].dQ = 2$ .
4.  $nM = n$ ;  $n = Length(Xar)$ .
5. Для всех  $i$  из  $[1..n]$  выполнить  $M[nM + i].x = Xar[i]$ ;  $M[nM + i].dQ = -2$ .
6.  $nM = nM + n$ ;  $n = Length(Xb)$ .
7. Для всех  $i$  из  $[1..n]$  выполнить  $M[nM + i].x = Xb[i]$ ;  $M[nM + i].dQ = 1$ .
8.  $nM = nM + n$ ;  $n = Length(Xbr)$ .
9. Для всех  $i$  из  $[1..n]$  выполнить  $M[nM + i].x = Xbr[i]$ ;  $M[nM + i].dQ = -1$ ,
10.  $nM = nM + n$ ; // общее число элементов в массиве  $M$
11. Отсортировать массив  $M$  по возрастанию  $M[i].x$ .
12.  $k = 1$ ;  $m = 1$ ;  $Q = 0$ .
13. Если  $(M[1].x \geq Xemin)$  и  $(M[1].dQ < 0)$ , то  
 $Xrl[1] = Xemin$ ;  $Q = -M[1].dQ$ ;  $k = 2$ .
14. Для всех  $i$  из  $[1..nM]$  выполнить п.п. 14.1. - 14.4.
  - 14.1.  $x = M[i].x$ ;  $Qnew = Q + M[i].dQ$ .
  - 14.2. Если  $(Q \notin SetQ)$  и  $(Qnew \in SetQ)$ , то  $Xrl[k] = x$ ;  $k = k + 1$ .
  - 14.3. Если  $(Q \in SetQ)$  и  $(Qnew \notin SetQ)$ , то  $Xrr[m] = x$ ;  $m = m + 1$ .
  - 14.4.  $Q = Qnew$ .
15. Если  $Q \in SetQ$ , то  $Xrr[m] = Xemax$ .

В первой части алгоритма границы сегментов из исходных массивов  $Xal$ ,  $Xbl$ ,  $Xar$ ,  $Xbr$  переписываются в рабочий массив  $M$ . При этом в поле  $dQ$  записываются приращения пороговых функций с учетом веса операндов. Цель сортировки массива  $M$  по возрастанию – упорядочивание значений аргумента функции  $Q(x)$ . После сортировки начинается обработка массива  $M$ . Особым случаем является такой, когда первым элементом массива  $M$  оказывается правая граница сегмента, т.е.  $M[1].x \geq Xemin$  и  $M[1].dQ < 0$ . В этой ситуации список  $Xrl$  левых границ сегментов необходимо дополнить значением  $Xemin$ .

В циклической части алгоритма последовательно просматриваются элементы массива  $M$ . В качестве текущего значения переменной  $x$  всегда выбирается очередное значение  $M[i].x$ . При этом функция  $Q$  изменяется на величину  $M[i].dQ$ .

Если в очередном цикле предыдущее значение  $Q$  не входило в  $SetQ$ , а новое ее значение  $Qnew$  в  $SetQ$  входит, значит  $x$  – координата левой границы сегмента результирующей области. Это значение записывается в  $Xrl$ . Аналогичным образом устанавливаются случаи, в которых  $x$  является координатой правой границы. Такие значения  $x$  записываются в массив  $Xrr$ .

По завершению цикла обработки  $M$  число левых границ в массиве  $Xrl$  должно быть равно числу правых границ в массиве  $Xrr$ . Если же в этот момент  $Q \in SetQ$ , значит не найдена правая граница последнего сегмента. Тогда за нее принимается правая граница  $Xemax$  области вывода.

## 1.5. Непрерывные геометрические преобразования

Непрерывные геометрические преобразования – важный инструмент в интерактивных графических программах, с помощью которого производится построение и размещение фигур на экране дисплея. Другим применением непрерывных преобразований является анимация, т. е. моделирование различных видов движения объектов по за-



ранее разработанным траекториям или других непрерывных изменений их геометрии.

В силу дискретного характера формирования изображения на экране дисплея плавные, непрерывные преобразования получают как последовательность малых преобразований. Результат каждого преобразования, как кадр фильма, выводится на экран. Реализация непрерывных преобразований возможна в двух формах – в интегральной и дифференциальной [4].

В интегральной форме в любой момент времени  $t$  матрица  $C(t)$  текущих координат рассчитывается на основе матрицы  $C_0$  исходных, т.е. первоначально заданных и неизменяемых координат:

$$C(t) = C_0 \times W(t), \quad (1.27)$$

где  $W(t)$  – матрица текущего преобразования на момент времени  $t$ .

В дифференциальной форме для расчета матрицы  $C(t + \Delta t)$  новых координат на очередном шаге используется матрица  $C(t)$  координат, полученных на предыдущем шаге:

$$C(t + \Delta t) = C(t) \times \Delta W, \quad (1.28)$$

где  $\Delta W$  – матрица элементарно малого преобразования за время  $\Delta t$ .

В программах, моделирующих движение объектов в реальных условиях, параметры  $t$  и  $\Delta t$  соответствуют времени. В большинстве же других графических приложений более важным бывает сам факт движения объектов. Тогда в качестве величины, сопоставимой со временем, удобнее использовать программные циклы по расчету результатов элементарных преобразований и визуализации, т.е.  $t$  в этом случае будет соответствовать номеру цикла, а  $\Delta t = 1$ .

Ниже показано как в интегральной форме выглядят частные виды непрерывных преобразований.

Преобразование плоскопараллельного перемещения:

$$C(t) = C_0 \times M(t) = C_0 \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m_x(t) & m_y(t) & 1 \end{bmatrix}. \quad (1.29)$$

Для моделирования равномерного прямолинейного перемещения следует задать  $m_x(t) = v_x t$  и  $m_y(t) = v_y t$ , где  $v_x, v_y$  – составляющие скорости, [1/цикл].

Преобразование вращения относительно начала координат:

$$C(t) = C_0 \times R(t) = C_0 \times \begin{bmatrix} \cos \varphi(t) & \sin \varphi(t) & 0 \\ -\sin \varphi(t) & \cos \varphi(t) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (1.30)$$

Для моделирования равномерного вращения  $\varphi(t) = \omega t$  следует задать угловую скорость  $\omega$ , [рад/цикл].

Преобразование масштабирования относительно начала координат:

$$C(t) = C_0 \times S(t) = C_0 \times \begin{bmatrix} s_x(t) & 0 & 0 \\ 0 & s_y(t) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (1.31)$$

Для моделирования линейного изменения масштабов по осям со скоростями  $v_x$  и  $v_y$ :

$$\begin{aligned} s_x(t) &= 1 + v_x t; \\ s_y(t) &= 1 + v_y t; \\ 0 &< |v_x|, |v_y| < 1. \end{aligned} \quad (1.32)$$

При программной реализации этого закона следует контролировать и не допускать использование нулевых значений  $s_x(t)$  и  $s_y(t)$ , приводящих к вырождению преобразуемых объектов.

Эффекту приближения и удаления фигур лучше соответствует показательный закон изменения масштаба:

$$\begin{aligned}
s_x(t) &= (1 + \delta_x)^t; \\
s_y(t) &= (1 + \delta_y)^t; \\
0 < |\delta_x|, |\delta_y| < 1.
\end{aligned} \tag{1.33}$$

Этот закон удобен и в реализации, так как нулевые значения  $s_x(t)$  и  $s_y(t)$  здесь исключаются.

Простейшие законы изменения параметров в приведенных преобразованиях даны в качестве примеров.

Теперь приведем вид матриц элементарно-малых преобразований для дифференциальной формы.

Плоскопараллельное перемещение:

$$\Delta M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta m_x & \Delta m_y & 1 \end{bmatrix}. \tag{1.34}$$

Вращение:

$$\Delta R = \begin{bmatrix} \cos \Delta \varphi & \sin \Delta \varphi & 0 \\ -\sin \Delta \varphi & \cos \Delta \varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1.35}$$

Масштабирование:

$$\Delta S = \begin{bmatrix} 1 + \delta_x & 0 & 0 \\ 0 & 1 + \delta_y & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1.36}$$

Заметим, что моделирование масштабирования в дифференциальной форме по результату соответствует масштабированию в интегральной форме на основе показательного закона изменения масштабных коэффициентов.

## 1.6. OpenGL и библиотека Tao Framework

**OpenGL** (Open Graphics Library) – открытая графическая библиотека, включающая в себя несколько сотен функций. Она определяет независимый от языка программирования кросс-платформенный программный интерфейс для разработки приложений, использующих двухмерную и трехмерную компьютерную графику [6].

Первая базовая версия **OpenGL** была разработана компанией Silicon Graphics Inc. в 1992 году. В последующие годы библиотека дорабатывалась и усовершенствовалась. Последняя на сегодняшний день версия библиотеки **OpenGL 4.3** [8]. Ее создатели заложили в нее механизм расширений, благодаря которому производители аппаратного обеспечения (к примеру, производители видеокарт) могут выпускать расширения **OpenGL**, для поддержки новых специфических возможностей, не включенных в текущую версию библиотеки.

Поскольку нет прямой поддержки данной библиотеки в **.NET Framework** для работы с **OpenGL** целесообразно использовать библиотеку **Tao Framework** [7]. Это свободно распространяемая библиотека с открытым исходным кодом, предназначенная для быстрой и удобной разработки кросс-платформенного мультимедийного программного обеспечения в среде **.NET Framework** и **Mono**. На сегодняшний день **Tao Framework** - это лучший путь для использования библиотеки **OpenGL** при разработке в среде **.NET** на языке C#. В состав библиотеки входят все современные средства, которые могут понадобиться в ходе разработки мультимедиа программного обеспечения: реализация библиотеки **OpenGL**; реализация библиотеки **FreeGlut**, содержащей все самые новые функции этой библиотеки; библиотека **DevIL** (лежшая в основу стандарта **OpenIL** – Open Image Library), и многие другие.

Основные библиотеки, включенные в **Tao Framework** версии **2.1.0**, перечислены в табл. 1.3.

Таблица 1.3

Библиотека	Характеристики
<b>OpenGL 2.1.0.12</b>	свободно распространяемый аппаратно-программный интерфейс для визуализации 2D и 3D графики
<b>FreeGLUT 2.4.0.2</b>	библиотека с открытым исходным кодом, являющаяся альтернативой библиотеке GLUT (OpenGL Utility Toolkit)
<b>DevIL 1.6.8.3</b> (OpenIL)	кросс-платформенная библиотека, реализующая программный интерфейс для работы с изображениями, поддерживает работу с изображениями 43-х форматов для чтения и 17-ти форматов для записи
<b>Cg 2.0.0.0</b>	язык высокого уровня, созданный для программирования текстурных и вершинных шейдеров
<b>OpenAL; 1.1.0.1</b>	свободно распространяемый аппаратно-программный интерфейс для обработки аудиоданных
<b>PhysFS 1.0.1.2</b>	библиотека для работы с вводом / выводом файловой системы, а так же различного вида архивами, на основе собственного API
<b>SDL 1.2.13.0</b>	кросс-платформенная мультимедийная библиотека, активно используемая для написания мультимедийных приложений в операционной системе
<b>GNU/Linux ODE 0.9.0.0</b>	свободно распространяемый физический программный интерфейс, главной особенностью которого является реализация системы динамики абсолютно твердого тела и система обнаружения столкновений

В **OpenGL** многоугольник (полигон) – это некая область в пространстве, которая ограничивается одной замкнутой ломаной линией. В качестве примитивов могут использоваться только выпуклые многоугольники без самопересечений и без ограничения числа вершин.

Для задания вершин многоугольников используется специальная команда: **glVertex\*()**. Она имеет следующий формат:

```
void glVertex {234 } {ifd }(TYPE coords);
```

Здесь 2, 3, 4 – это количество координат для задания вершин. В **OpenGL** точки в трехмерном пространстве могут задаваться в однородных координатах, т.е. четырьмя координатами  $x$ ,  $y$ ,  $z$ ,  $r$ . Таким образом, положение точки в обычной системе координат описывается,

как  $x/r$ ,  $y/r$ ,  $z/r$ . По умолчанию (когда  $r$  не указывается)  $r = 1$ . Если указываются точки в двухмерной системе координат,  $z$  считается равным нулю. В рассматриваемой команде  $i, f, d$  – это возможный тип принимаемых значений. Например, если написать команду **glVertex3f(,,)** то в скобках должны быть указаны три переменные типа **float**. Для  $d$  – **double**, для  $i$  – **int**.

Вызов функции **glVertex** может иметь результат только между вызовами функций **glBegin()** и **glEnd()**. Для построения фигур, задаваемых с помощью вершин, используются разные режимы, которые описываются различными правилами объединения вершин в многоугольники.

Такие режимы рисования возможны лишь с помощью перечисления точек между командами **glBegin()** и **glEnd()**, причем при вызове функции **glBegin** указывается, в каком режиме будут соединяться полигоны. Значения параметров, которые может принимать **glBegin**, приведены в табл. 1.4.

Таблица 1.4

Библиотека	Характеристики
<b>GL_POINTS</b>	На месте каждой указанной с помощью команды <b>glVertex</b> вершины рисуется точка
<b>GL_LINES</b>	Каждые 2 вершины объединяются в отрезок
<b>GL_LINE_STRIP</b>	Вершины соединяются последовательно, образуя ломаную линию
<b>GL_LINE_LOOP</b>	То же, что и <b>GL_LINE_STRIP</b> , но последняя вершина автоматически будет соединена отрезком с начальной вершиной
<b>GL_TRIANGLES</b>	Каждые три вершины объединяются в треугольник
<b>GL_TRIANGLE_STRIP</b>	Лента треугольников. Треугольники рисуются таким образом, что последняя сторона треугольника - является начальной стороной следующего
<b>GL_TRIANGLE_FAN</b>	То же, что и <b>GL_TRIANGLE_STRIP</b> , только изменен порядок следования вершин. В итоге получается «веер» из треугольников

Библиотека	Характеристики
<b>GL_QUADS</b>	Каждые четыре вершины объединяются в квадрат
<b>GL_QUAD_STRIP</b>	Вершины объединяются в квадраты, причем последняя сторона квадрата является первой стороной следующего
<b>GL_POLYGON</b>	Рисуется многоугольник на основе вершин. Вершин не должно быть менее 3 и должны отсутствовать самопересечения, а так же должны сохраняться условия выпуклости

В **OpenGL API** существуют команды для модельных преобразований **glTranslate\*()**, **glRotate\*()**, **glScale\*()**. С помощью них можно выполнять преобразования объекта или координатной системы (в зависимости от точки зрения на преобразования). Данные команды являются эквивалентом соответствующих матриц с вызовом функции **glMultMatrix\*()** с нужной матрицей преобразования в качестве аргумента. Но команды частных случаев преобразования **glTranslate**, **glRotate** и **glScale** выполняются быстрее. Рассмотрим их немного подробнее.

Плоскопараллельное перемещение (перенос) определено как

**void glTranslate{fd} (TYPE x, TYPE y, TYPE z);**

Параметры **x**, **y**, **z** задают перемещения по соответствующим осям.

Преобразование поворота

**void glRotate{fd} (TYPE Angle, TYPE x, TYPE y, TYPE z);**

Данная функция умножает текущую матрицу преобразования на матрицу, производящую поворот объекта на угол **Angle**. Для положительного значения **Angle** поворот происходит против часовой стрелки, вокруг луча из начала координат к точке (**x**, **y**, **z**). Угол поворота задается в градусах.

Преобразование масштабирования выполняется с помощью функции

**void glScale{fd} (TYPE x, TYPE y, TYPE z);**

Параметры  $x$ ,  $y$ ,  $z$  задают масштабные коэффициенты по соответствующим осям. Отрицательные значения параметров кроме масштабирования приводят к отражению относительно одноименных осей.



## **2. ЛАБОРАТОРНЫЕ РАБОТЫ**

### **2.1. Общие требования к выполнению лабораторных работ**

Целью проведения лабораторных работ является закрепление теоретических знаний и приобретение практических навыков в применении базовых методов и алгоритмов машинной графики при разработке программ.

Программы, разрабатываемые на лабораторных занятиях, должны обладать общепринятыми элементами интерфейса: системным и контекстным меню, инструментальными панелями, другими элементами диалога. В программах должны быть предусмотрены средства для выбора цвета рисования.

Для задания и выделения геометрических объектов на экране, а также для работы с ними необходимо использовать известные или разработать собственные интерактивные средства. В частности, отдельные точки, отрезки прямых (векторы), многоугольники общего вида должны задаваться на экране с помощью мыши. Желательно использовать мышь для выделения отдельных точек, концов отрезков, фигур и изменения их положения.

Каждый исследуемый метод должен быть реализован в виде процедуры или метода программного объекта.

При разработке программ необходимо принимать все меры для того, чтобы добиться высокого быстродействия процедур визуализации.

При сдаче лабораторной работы студент должен продемонстрировать преподавателю работу действующей программы, произвести ее тестирование, дать необходимые пояснения теоретического характера. Затем студент должен прокомментировать разработку текста

программы в той части, которая касается исследуемого метода или алгоритма. Следует приводить теоретические обоснования, подтверждающие правильность выбранных решений.

## **2.2. Лабораторная работа № 1**

**Тема:** «Визуализация отрезков прямых»

**Цель работы** – практическая реализация алгоритма визуализации отрезков и оценка его быстродействия.

**Программное обеспечение.** Для выполнения лабораторной работы на компьютере должен быть установлен пакет программ **Microsoft Visual Studio**.

### ***Порядок проведения работы***

Используя материал из разд. 1.1, необходимо разработать алгоритм визуализации отрезков прямых для общего случая, когда начальная и конечная точки отрезка располагаются на плоскости произвольным образом. На основании алгоритма нужно разработать программу, обеспечивающую рисование на экране любого количества отрезков. Начальная и конечная точки отрезков должны задаваться интерактивно с помощью мыши.

Быстродействие алгоритма следует оценивать по числу операций, приходящихся на вывод пикселя в наилучшем и наихудшем случае.

После завершения разработки программа должна быть протестирована на всех возможных вариантах взаимного расположения начальной и конечной точек отрезка.

### ***Содержание отчета***

Отчет должен включать

- цель работы;
- текст задания;

- исходные данные;
- описание выполняемых в соответствии с заданием действий;
- промежуточные результаты;
- конечные результаты;
- выводы.

Результаты визуализации отрезков должны быть представлены в отчете в виде скриншотов.

### **Контрольные вопросы**

1. Каковы причины погрешности визуализации геометрических объектов на дискретной области вывода?
2. Почему оценочная функция получила такое название?
3. Почему в алгоритме визуализации отрезков для выбора очередной точки рассматриваются только два варианта?
4. Почему в алгоритме визуализации отрезков сравниваются между собой абсолютные величины значений оценочной функции?

## **2.3. Лабораторная работа № 2**

### **Тема: «Сплаины»**

**Цель работы** – практическое знакомство со способами задания сплайнов и методами их визуализации.

**Программное обеспечение.** Для выполнения лабораторной работы на компьютере должен быть установлен пакет программ **Microsoft Visual Studio**.

### **Порядок проведения работы**

В данной лабораторной работе требуется разработать программу для интерактивного задания и визуализации кубических сплайнов и кривых Безье. При подготовке к лабораторной работе и разработке программы следует руководствоваться материалами разд. 1.2.

Для контроля правильности построения кубического сплайна должны быть предусмотрены два режима вывода: визуализация сплайна с касательными векторами в начальной и конечной точках и визуализация только сплайна.

Кривую Безье также необходимо выводить в режимах с визуализацией и без визуализации начальных условий. При визуализации опорных точек кривой желательно соединять их друг с другом отрезками в порядке построения.

После завершения разработки программы произвести ее тестирование, а также исследовать влияние начальных условий на геометрию сплайна.

### ***Содержание отчета***

Отчет должен включать

- цель работы;
- текст задания;
- исходные данные;
- описание выполняемых в соответствии с заданием действий;
- промежуточные результаты;
- конечные результаты;
- выводы.

Результаты задания и визуализации сплайнов должны быть представлены в отчете в виде скриншотов.

### **Контрольные вопросы**

1. Чем отличаются между собой алгебраические и параметрические линии?
2. Что является начальными условиями для кубических сплайнов в базисе Эрмита?
3. От чего зависит степень полиномов, используемых для описания кривых Безье?
4. Какова минимальная степень полиномов для кривых Безье?
5. Какие проблемы, связанные с точностью, возникают при машинном построении сплайнов?

## 2.4. Лабораторная работа № 3

**Тема:** «Закрашивания многоугольников»

**Цель работы** – программная реализация двух методов закрашивания многоугольников.

**Программное обеспечение.** Для выполнения лабораторной работы на компьютере должен быть установлен пакет программ **Microsoft Visual Studio**.

### ***Порядок проведения работы***

В данной лабораторной работе требуется разработать программу, которая обеспечивает интерактивный ввод вершин произвольного многоугольника и автоматическое его закрашивание двумя методами, изложенными в разд. 1.3.

Для контроля правильности закрашивания нужно предусмотреть два режима вывода: с прорисовкой граничного многоугольника и без прорисовки.

При тестировании программы обязательно должна быть проверена правильность закрашивания невыпуклых многоугольников, а также многоугольников с отдельными горизонтальными сторонами и цепочками горизонтальных сторон. При тестировании процедуры закрашивания ориентированных многоугольников в качестве тестов необходимо задавать многоугольники с ориентацией, как по часовой стрелке, так и против часовой стрелки.

Результаты данной лабораторной работы могут быть использованы в следующей лабораторной работе.

### ***Содержание отчета***

Отчет должен включать

- цель работы;
- текст задания;
- исходные данные;

- описание выполняемых в соответствии с заданием действий;
- промежуточные результаты;
- конечные результаты;
- выводы.

Результаты задания и визуализации закрашенных многоугольников должны быть представлены в отчете в виде скриншотов.

### **Контрольные вопросы**

1. Чем принципиально отличаются алгоритмы закрашивания неориентированных и ориентированных многоугольников?
2. Какие особые случаи существуют в алгоритмах закрашивания?
3. Какие проблемы возникают при закрашивании, если многоугольник содержит горизонтальные стороны?
4. Как определить пресекается ли сторона многоугольника с текущей строкой?
5. Почему важно упорядочивать границы сегментов в строке по возрастанию или убыванию?

## **2.5. Лабораторная работа № 4**

**Тема:** «Теоретико-множественные операции»

**Цель работы** – программная реализация ТМО над двумя произвольными многоугольниками.

**Программное обеспечение.** Для выполнения лабораторной работы на компьютере должен быть установлен пакет программ **Microsoft Visual Studio**.

### **Порядок проведения работы**

В данной лабораторной работе требуется разработать программу, в которой предусмотрено интерактивное построение двух произвольных многоугольников – операндов ТМО и выполнение над ними

ТМО заданного вида с использованием методики, изложенной в разд. 1.4.

### ***Содержание отчета***

Отчет должен включать

- цель работы;
- текст задания;
- исходные данные;
- описание выполняемых в соответствии с заданием действий;
- промежуточные результаты;
- конечные результаты;
- выводы.

Результаты задания исходных данных и синтеза новых фигур должны быть представлены в отчете в виде скриншотов.

### **Контрольные вопросы**

1. Что характеризует пороговая функция сечения какой-либо фигуры горизонтальной строкой?
2. С какой целью выполняется взвешенное суммирование пороговых функций сечений операндов ТМО?
3. В каком формате должны быть представлены сечения операндов для алгоритма ТМО?
4. В каких случаях список границ сегментов сечения результирующей области приходится дополнять левой или правой границами области вывода?

## **2.6. Лабораторная работа № 5**

**Тема:** «Геометрические преобразования на плоскости»

**Цель работы** – программная реализация геометрических преобразований над двумерными объектами.

**Программное обеспечение.** Для выполнения лабораторной работы на компьютере должен быть установлен пакет программ **Microsoft Visual Studio**.

**Порядок проведения работы.** В данной лабораторной работе требуется разработать программу, обеспечивающую интерактивное задание какого-либо объекта, например, многоугольника, а также его анимацию. При подготовке к лабораторной работе и разработке программы следует руководствоваться материалами, изложенными в разд. 1.5.

Геометрические преобразования должны выполняться в программе в интегральной форме с помощью матричных операций в однородных координатах.

Моделирование непрерывных геометрических преобразований должно быть организовано либо в виде анимации с ручным изменением параметра преобразования с помощью мыши, либо как анимация с заранее запрограммированным законом изменения параметров. В любом случае нужно предусмотреть возможность моделирования всех основных видов преобразований.

### ***Содержание отчета***

Отчет должен включать

- цель работы;
- текст задания;
- исходные данные;
- описание выполняемых в соответствии с заданием действий;
- промежуточные результаты;
- конечные результаты;
- выводы.

Результаты задания исходных данных и геометрических преобразований над ними должны быть представлены в отчете в виде скриншотов.



### Контрольные вопросы

1. На чем основано машинное моделирование непрерывных преобразований?
2. Чем отличаются между собой интегральная и дифференциальная формы реализации непрерывных геометрических преобразований?
3. В чем может проявиться погрешность вычислений при моделировании непрерывных преобразований в дифференциальной форме?

## 2.7. Лабораторная работа № 6

### Тема: «Знакомство с OpenGL»

**Цель работы** – Инициализация библиотеки **Tao Framework** в среде **.NET** для **C#** и практическое знакомство с библиотекой **OpenGL**.

**Программное обеспечение.** Для выполнения лабораторной работы на компьютере должен быть установлен пакет программ **Microsoft Visual Studio**, библиотека **Tao Framework**.

### *Порядок проведения работы*

При подготовке к лабораторной работе следует руководствоваться материалами, изложенными в разд. 1.6, а также в [7].

Запустив **Micrisoft Visual Studio**, нужно создать новый проект на языке **C#** вида Приложение Windows Forms. Имя проекта, например, **TaoInit**.

После завершения генерации проекта нужно открыть Обзорщик решений (рис. 2.1), открыть список ссылок (References). В контекстном меню для этого списка выберите добавление ссылок, как показано на рис. 2.1.

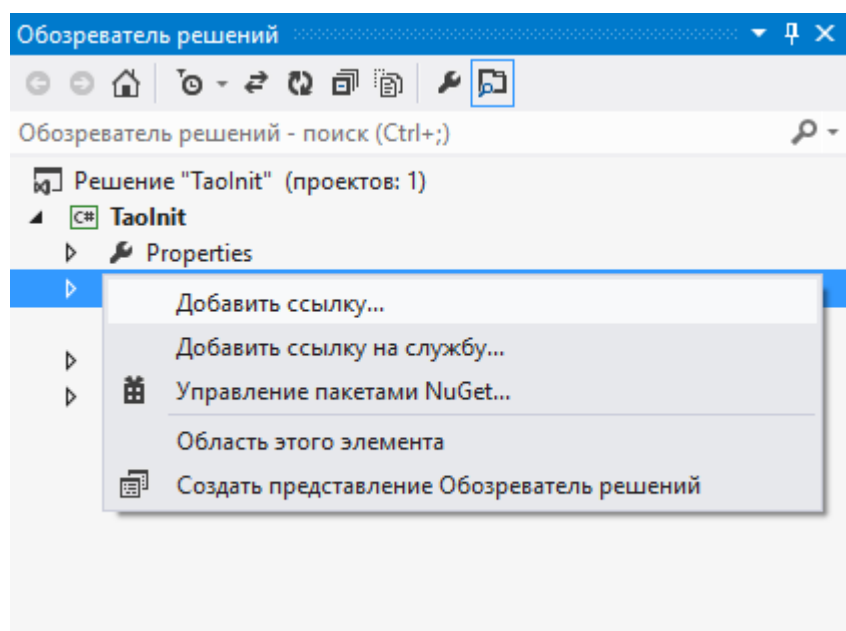


Рис. 2.1

В открывшемся окне «Добавить ссылку», перейдите к закладке Обзор. После этого перейдите к директории, в которую была установлена библиотека **Tao Framework** (по умолчанию – «C:\Program Files\Tao Framework»). Необходимые библиотеки хранятся в папке **bin**. Выделите в папке **bin** три библиотеки, как показано на рис. 2.2:

- **Tao.OpenGL.dll** - отвечает за реализацию библиотеки **OpenGL**
- **Tao.FreeGlut.dll** - отвечает за реализацию функций библиотеки **Glut**. Мы будем ее использовать для инициализации рендера, а так же для различных других целей.
- **Tao.Platform.Windows.dll** - отвечает за поддержку элементов для визуализации непосредственно на платформе **Windows**.

Ссылки на перечисленные библиотеки должны добавиться в список **References** обозревателя решений.

Теперь перейдите к исходному коду окна. Для работы с выбранными библиотеками, нужно подключить соответствующие пространства имен, как показано ниже.

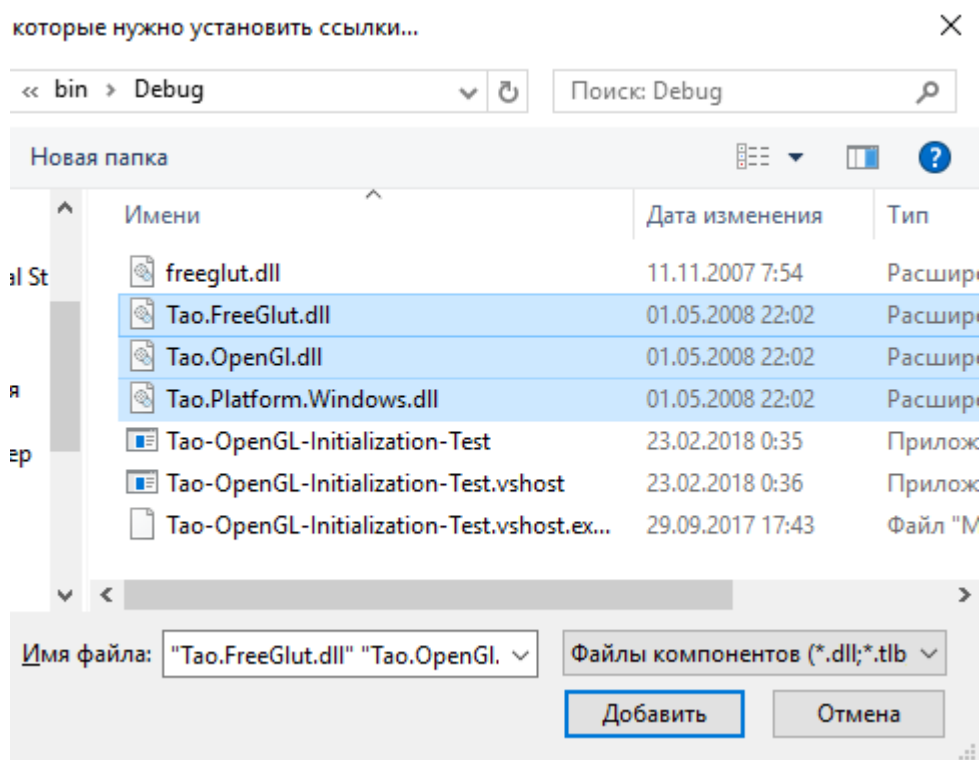


Рис. 2.2.

```
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 Tao.OpenGl;           // для работы с библиотекой OpenGL
using Tao.FreeGlut;         // для работы с библиотекой FreeGLUT
// для работы с элементом управления SimpleOpenGLControl
using Tao.Platform.Windows;
```

Теперь вернитесь к конструктору диалогового окна и перейдите к окну **Toolbox** (панель элементов). Щелкнув правой кнопкой на вкладке **Общие (General)**, в раскрывшемся контекстном меню выберите пункт «**Выбрать элементы**» (**Choose Items**), рис. 2.3.

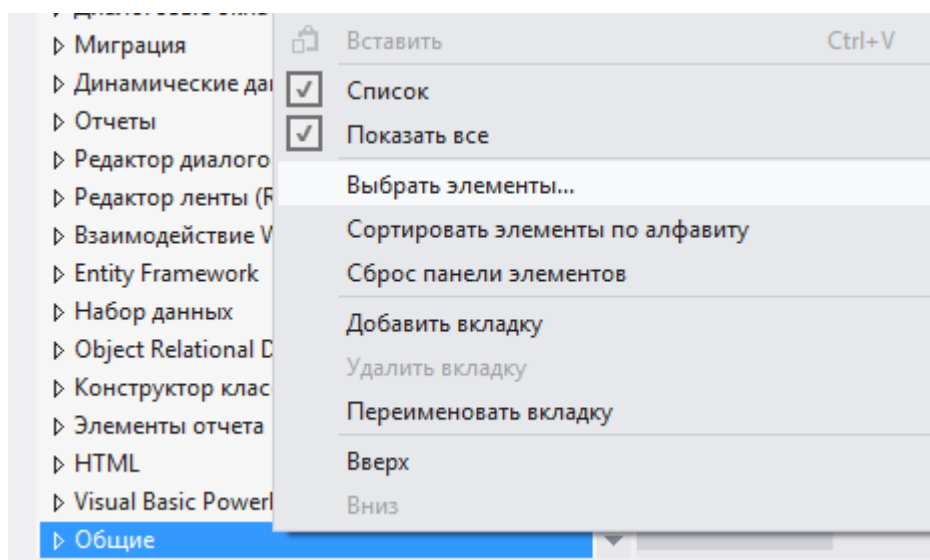


Рис. 2.3.

В открывшемся окне найдите элемент **SimpleOpenGLControl** и установите возле него галочку, как показано на рис. 2.4. Затем нажмите кнопку **ОК**.

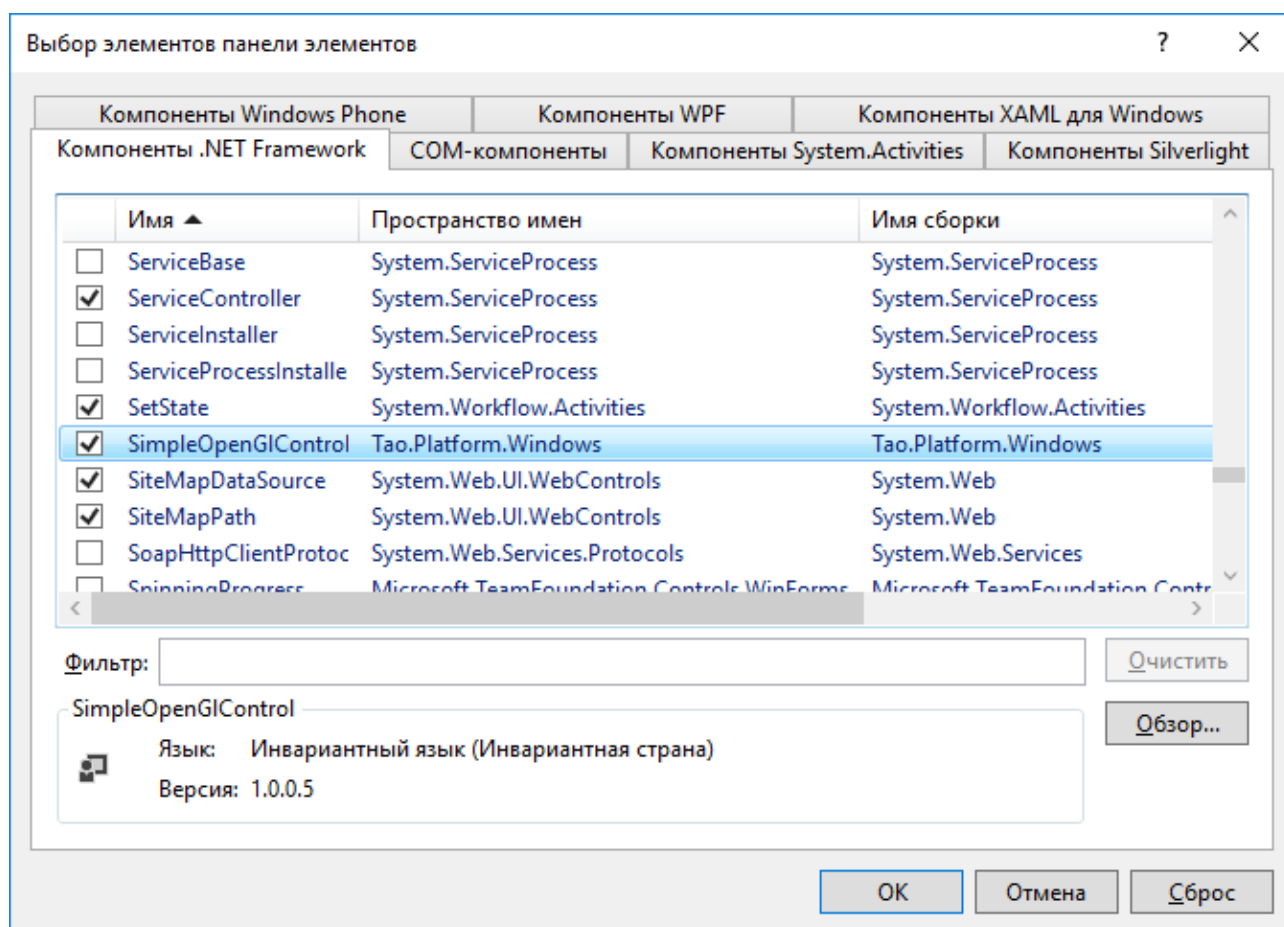


Рис. 2.4

Теперь данный элемент станет доступным для размещения на форме приложения. Перетащите его на форму, и разместите так, как показано на рис. 2.5. Справа от размещенного элемента установите также две кнопки – **Визуализация** и **Выход**.

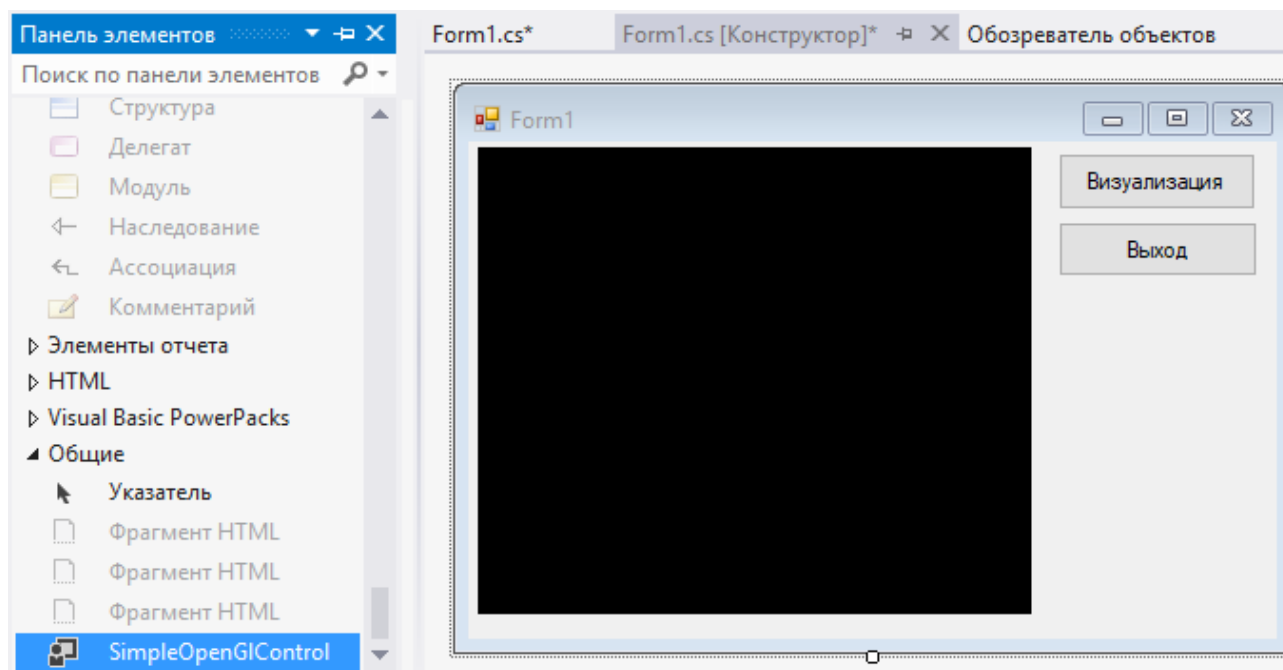


Рис. 2.5

Теперь выделите элемент **SimpleOpenGLControl1**, расположенный на форме и перейдите к его свойствам. Измените параметр **name** на значение «**AnT**». Далее элемент **SimpleOpenGLControl** будем называть **AnT** (рис. 2.6).

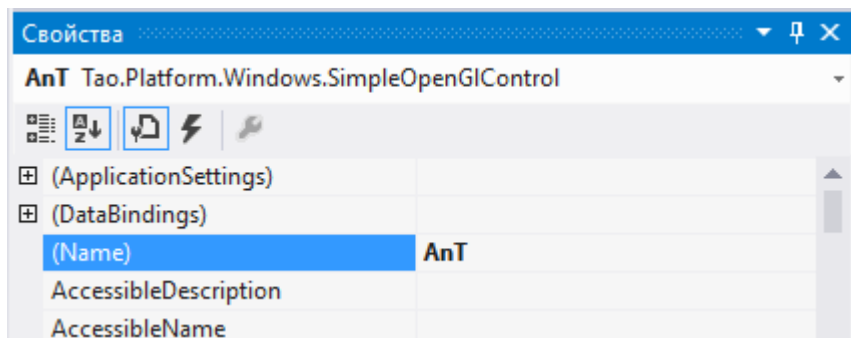


Рис. 2.6

Теперь необходимо инициализировать работу **OpenGL**. Сначала в конструкторе класса нужно инициализировать работу элемента **AnT**:

```
public Form1()
{
    InitializeComponent();
    AnT.InitializeContexts();
}
```

Снова перейдите к конструктору и сделайте двойной щелчок левой клавишей мыши на форме. Создастся функция-обработчик события загрузки формы. В ней нужно поместить код инициализации **OpenGL**, приведенный ниже.

```
private void Form1_Load(object sender, EventArgs e)
{
    // инициализация Glut
    Glut.glutInit();
    Glut.glutInitDisplayMode(Glut.GLUT_RGB | Glut.GLUT_DOUBLE | Glut.GLUT_DEPTH);

    // очистка окна
    Gl.glClearColor(1, 1, 1, 1); // белый

    // установка порта вывода в соответствии с размерами элемента AnT
    Gl.glViewport(0, 0, AnT.Width, AnT.Height);

    // настройка проекции
    Gl.glMatrixMode(Gl.GL_PROJECTION);
    Gl.glLoadIdentity();
```

```

Glu.gluPerspective(45, (float)AnT.Width / (float)AnT.Height, 0.1, 200);
Gl.glMatrixMode(Gl.GL_MODELVIEW);
Gl.glLoadIdentity();

// настройка параметров OpenGL для визуализации
Gl.glEnable(Gl.GL_DEPTH_TEST);
}

```

В первую очередь в процедуре проходит инициализация библиотеки **Glut**. Для работы с функциями библиотеки **OpenGL** используется класс **Gl**, находящийся в пространстве имен **Tao.OpenGL**. Для работы с функциями библиотеки **Glut** используется класс **Glut**.

Функция **Glut.glutInitDisplayMode** устанавливает режим отображения. В нашем случае устанавливается **RGB** режим для визуализации (**GLUT\_RGB** — это псевдоним **GLUT\_RGBA**, он устанавливает режим **RGBA** битовой маски окна). Значение **Glut.GLUT\_DOUBLE** устанавливает двойную буферизацию окна. Двойная буферизация, как правило, используется для устранения мерцания, возникающего в процессе быстрой перерисовки кадров несколько раз подряд. **GLUT\_DEPTH** указывает при инициализации окна, если в приложении будет использоваться буфер глубины.

На окне мы создали две кнопки. Обработчик кнопки **Выход** будет выглядеть следующим образом

```

//обработчик кнопки «Выход»
private void button2_Click(object sender, EventArgs e)
{
    Application.Exit();
}

```

Обработчик кнопки **Визуализация** будет содержать код, реализующий каркасную визуализацию сферы, (для полутоновой визуализации используется библиотека **FreeGLUT**). Код, который нужно разметить в данной функции, отвечает и за разные технические аспекты визуализации.

```

// обработчик кнопки «Визуализация»
private void button1_Click(object sender, EventArgs e)
{
    Gl.glClear(Gl.GL_COLOR_BUFFER_BIT | Gl.GL_DEPTH_BUFFER_BIT);

    Gl.glLoadIdentity();
    Gl.glColor3f(1.0f, 0, 0);

    Gl.glPushMatrix();
    Gl.glTranslated(0,0,-6);
    Gl.glRotated(45, 1, 1, 0);

    // рисуем сферу с помощью библиотеки FreeGLUT
    Glut.glutWireSphere(2, 32, 32);

    Gl.glPopMatrix();
    Gl.glFlush();
    AnT.Invalidate(); // перерисовка кадра изображения
}

```

Откомпилируйте и запустите приложение. Если оно не содержит ошибок, то после нажатия на кнопку **Визуализация** результат должен быть аналогичен показанному на рис. 2.7.

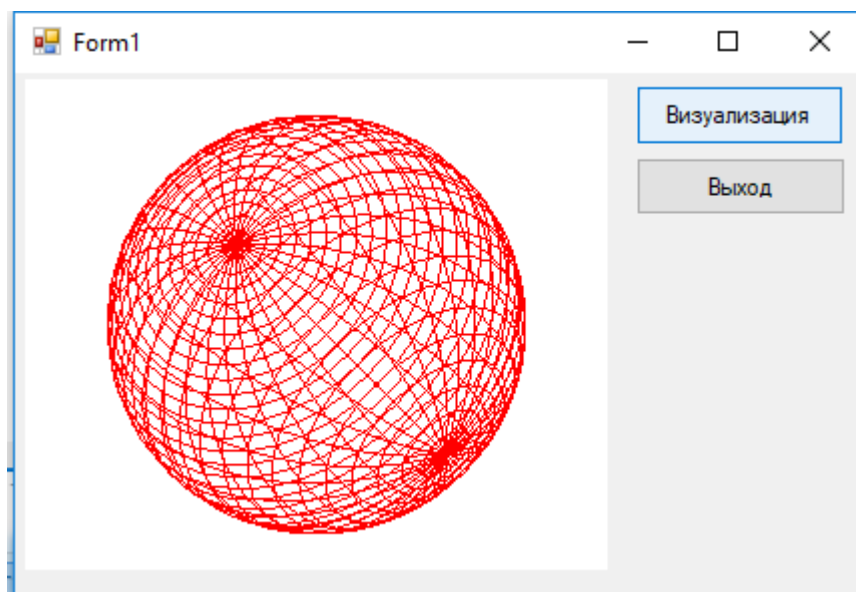


Рис. 2.7.



## *Содержание отчета*

Отчет должен включать

- цель работы;
- текст задания;
- описание выполняемых в соответствии с заданием действий;
- конечные результаты;
- выводы.

Результаты визуализации объекта с помощью библиотеки **Tao Framework** представить в виде скриншота.

## **Контрольные вопросы**

1. Какие основные библиотеки содержатся в **Tao Framework**?
2. Какова роль **Tao.Platform.Windows.dll** в созданном приложении?
3. Какие настройки выполняются в обработчике события загрузки формы?

## **2.8. Лабораторная работа № 7**

**Тема:** «Двумерная графика в OpenGL»

**Цель работы** – визуализация примитивов и других двумерных объектов средствами **OpenGL**.

**Программное обеспечение.** Для выполнения лабораторной работы на компьютере должен быть установлен пакет программ **Microsoft Visual Studio**, библиотека **Tao Framework**.

### ***Порядок проведения работы***

При подготовке к лабораторной работе следует руководствоваться материалами, изложенными в разд. 1.6, а также в [7].

Запустив **Micrisoft Visual Studio**, нужно создать новый проект на языке **C#** так же, как и в предыдущей лабораторной работе. Как и там нужно инициализировать **Tao Framework**, расширить пространство имен, расположить основные элементы на форме. Далее также сле-

дует назначить необходимые обработчики событий и инициализировать необходимые элементы.

Перейдя к свойствам формы, измените название окна на «**Визуализация 2D примитивов**».

Рассмотрим код функции **Form1\_Load**. Теперь он будет выглядеть следующим образом:

```
private void Form1_Load(object sender, EventArgs e)
{
    // инициализация Glut
    Glut.glutInit();
    Glut.glutInitDisplayMode(Glut.GLUT_RGB | Glut.GLUT_DOUBLE | Glut.GLUT_DEPTH);

    // очистка окна
    Gl.glClearColor(1, 1, 1, 1); // белый цвет

    // установка порта вывода в соответствии с размерами элемента AnT
    Gl.glViewport(0, 0, AnT.Width, AnT.Height);

    // настройка проекции
    Gl.glMatrixMode(Gl.GL_PROJECTION);
    Gl.glLoadIdentity();

    // настройка 2D ортогональной проекции
    // в зависимости от того, какая сторона больше
    if ((float)AnT.Width <= (float)AnT.Height)
    {
        Glu.gluOrtho2D(0.0, 30.0 * (float)AnT.Height / (float)AnT.Width, 0.0, 30.0);
    }
    else
    {
        Glu.gluOrtho2D(0.0, 30.0 * (float)AnT.Width / (float)AnT.Height, 0.0, 30.0);
    }

    Gl.glMatrixMode(Gl.GL_MODELVIEW);
    Gl.glLoadIdentity();
}
```

Как видно из кода, за настройку 2D ортогональной проекции отвечает функция `gluOrtho2D`, реализация которой предоставлена библиотекой **Glu**. Эта функция как бы помещает начало координат в самый левый нижний квадрат, а камера (наблюдатель) в таком случае находится на оси **Z**, и таким образом визуализируется графика в 2D режиме.

По сути, нужно передать в качестве параметров координаты области видимости окном проекции: **left**, **right**, **bottom** и **top**. Чтобы исключить искажения, связанные с тем, что область вывода не квадратная, параметр **top** рассчитывается как произведение 30 и отношения высоты элемента **AnT** (в котором будет идти визуализация) к его ширине. В том же случае, если высота больше ширины, то наоборот: отношением ширины к высоте.

Теперь нужно написать код функции **button1\_Click**. В ней будет производиться очистка буфера цвета кадра и очистка текущей объектно-видовая матрица. Затем в этой же функции в режиме рисования линий нужно задать координаты вершин контуров для буквы «А». В конце должна следовать перерисовка содержимого элемента **AnT**.

Но перед тем как нарисовать букву «А», для тренировки просто проведите линию из начала координат в противоположенный угол окна. Тогда код функции будет выглядеть следующим образом:

```
private void button1_Click(object sender, EventArgs e)
{
    // очищаем буфер цвета
    Gl.glClear(Gl.GL_COLOR_BUFFER_BIT);

    // очищаем текущую матрицу
    Gl.glLoadIdentity();
    // устанавливаем текущий цвет - красный (от 0 до 1)
    Gl.glColor3f(1, 0, 0);
    // активируем режим ломаной линий для рисования диагонали
    Gl.glBegin(Gl.GL_LINE_STRIP);

    // первая вершина будет находиться в начале координат
    Gl.glVertex2d(0, 0);
```

```

// рисуем вторую вершину в противоположенном углу
// в зависимости от того, как была определена проекция
if ((float)AnT.Width <= (float)AnT.Height)
{
    Gl.glVertex2d(30.0f * (float)AnT.Height / (float)AnT.Width, 30);
}
else
{
    Gl.glVertex2d(30.0f * (float)AnT.Width / (float)AnT.Height, 30);
}
// завершаем режим рисования
Gl.glEnd();

// ожидаем конца визуализации кадра
Gl.glFlush();

AnT.Invalidate(); // перерисовка элемента AnT.
}

```

На рис. 2.9 показан примерный вид окна приложения после его компиляции и запуска приложения.

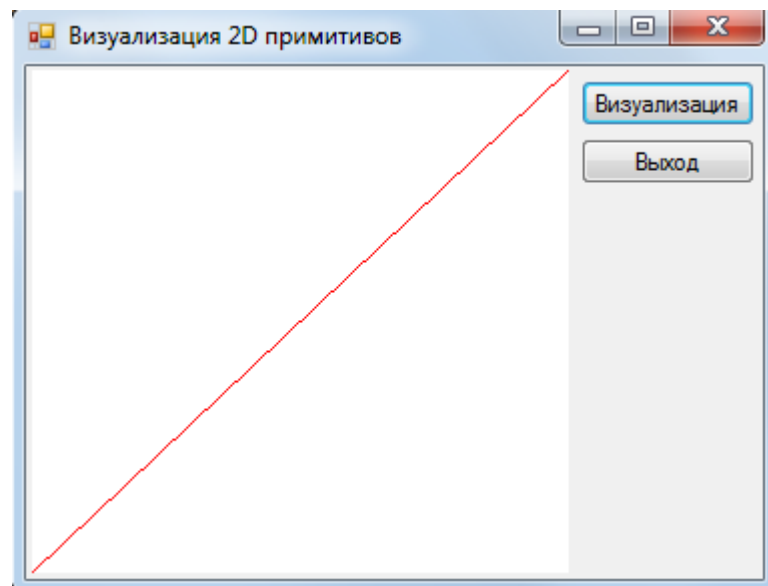


Рис. 2.9

Далее программу необходимо переработать для визуализации изображения, в виде буквы «А». Изображение буквы состоит из двух замкнутых ломаных линий (рис. 2.10).

На рис. 2.10 отмечены начальные точки обеих линий и значения координат всех вершин.

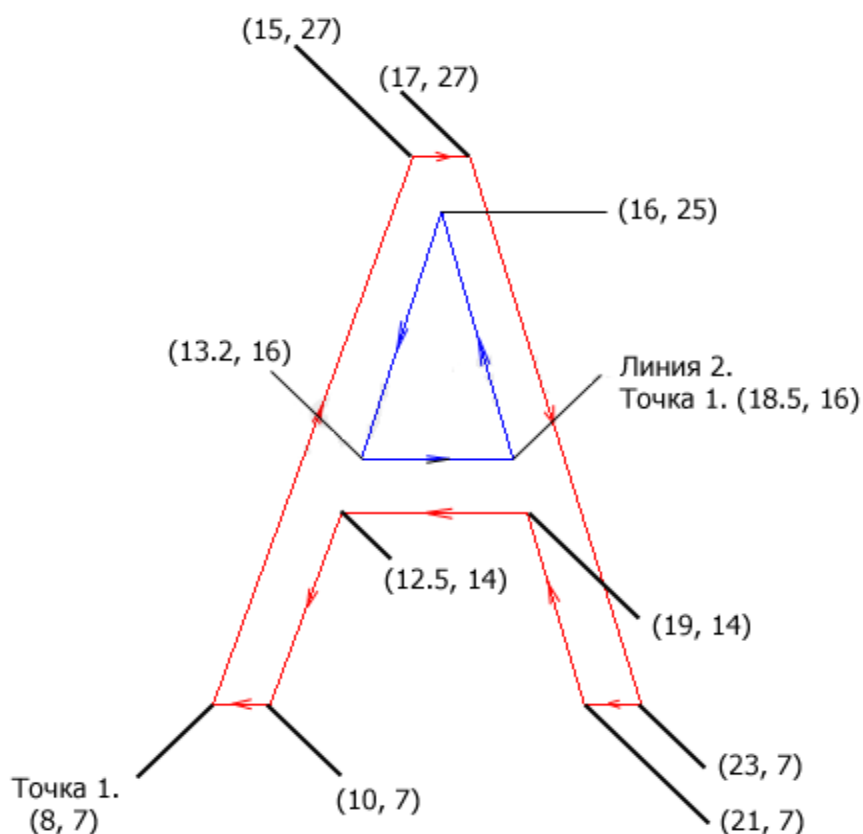


Рис. 2.10

Теперь код, написанный для рисования диагональной линии, надо заменить на новый, приведенный ниже:

// активируем режим рисования первой линии

```
Gl.glBegin(Gl.GL_LINE_LOOP);
```

```
Gl.glVertex2d(8, 7);
```

```
Gl.glVertex2d(15, 27);
```

```
Gl.glVertex2d(17, 27);
```

```
Gl.glVertex2d(23, 7);
```

```
Gl.glVertex2d(21, 7);
```

```
Gl.glVertex2d(19, 14);
```

```

Gl.glVertex2d(12.5, 14);
Gl.glVertex2d(10, 7);

Gl.glEnd(); // завершаем режим рисования

// вторая линия
Gl.glBegin(Gl.GL_LINE_LOOP);

Gl.glVertex2d(18.5, 16);
Gl.glVertex2d(16, 25);
Gl.glVertex2d(13.2, 16);

Gl.glEnd(); // завершаем режим рисования

```

Если снова откомпилировать проект и запустить приложение, а затем нажать на кнопку «Визуализировать», должен быть следующий результат (рис. 2.11).

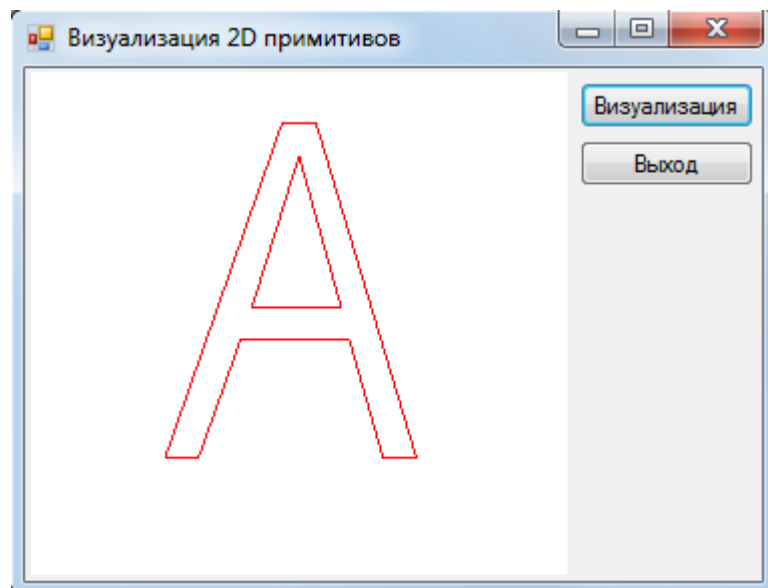


Рис. 2.11.

### ***Содержание отчета***

Отчет должен включать

- цель работы;
- описание выполняемых в соответствии с заданием действий;

- конечные результаты;
- выводы.

Результаты визуализации 2D-объектов с помощью библиотеки **Tao Framework** представить в виде скриншотов.

### Контрольные вопросы

1. Какова роль команды **Gl.glBegin()**?
2. Какова область действия команды **Gl.glBegin()**?
3. Какие варианты задания многоугольников и ломаных линий предусмотрены в **OpenGL**?

## 2.9. Лабораторная работа № 8

**Тема:** «Трехмерные примитивы и геометрические преобразования»

**Цель работы** – визуализация примитивов и других трехмерных объектов средствами **OpenGL**.

**Программное обеспечение.** Для выполнения лабораторной работы на компьютере должен быть установлен пакет программ **Microsoft Visual Studio**, библиотека **Tao Framework**.

### *Порядок проведения работы*

При подготовке к лабораторной работе следует руководствоваться материалами, изложенными в разд. 1.6, а также в [7].

Создайте новый проект. После этого создайте окно приложения, разместив на нем элемент **SimpleOpenGLControl1**, переименовав его в **AnT**, а также следующие элементы управления:

- поле выбора типа **combobox** с названием «Объект»;
- флаг «каркасная модель»;
- поле выбора типа **combobox** с названием «Вращение вокруг оси»;

- три ползунковых элемента типа **trackBar** для задания перемещений по осям **X**, **Y** и **Z**;
- элемент типа **trackBar** с названием «Угол» для задания угла поворота относительно заданной оси вращения;
- элемент типа **trackBar** с названием «М» для задания коэффициента пропорционального масштабирования.

Размещение всех перечисленных диалоговых элементов на форме показано на рис. 2.12.

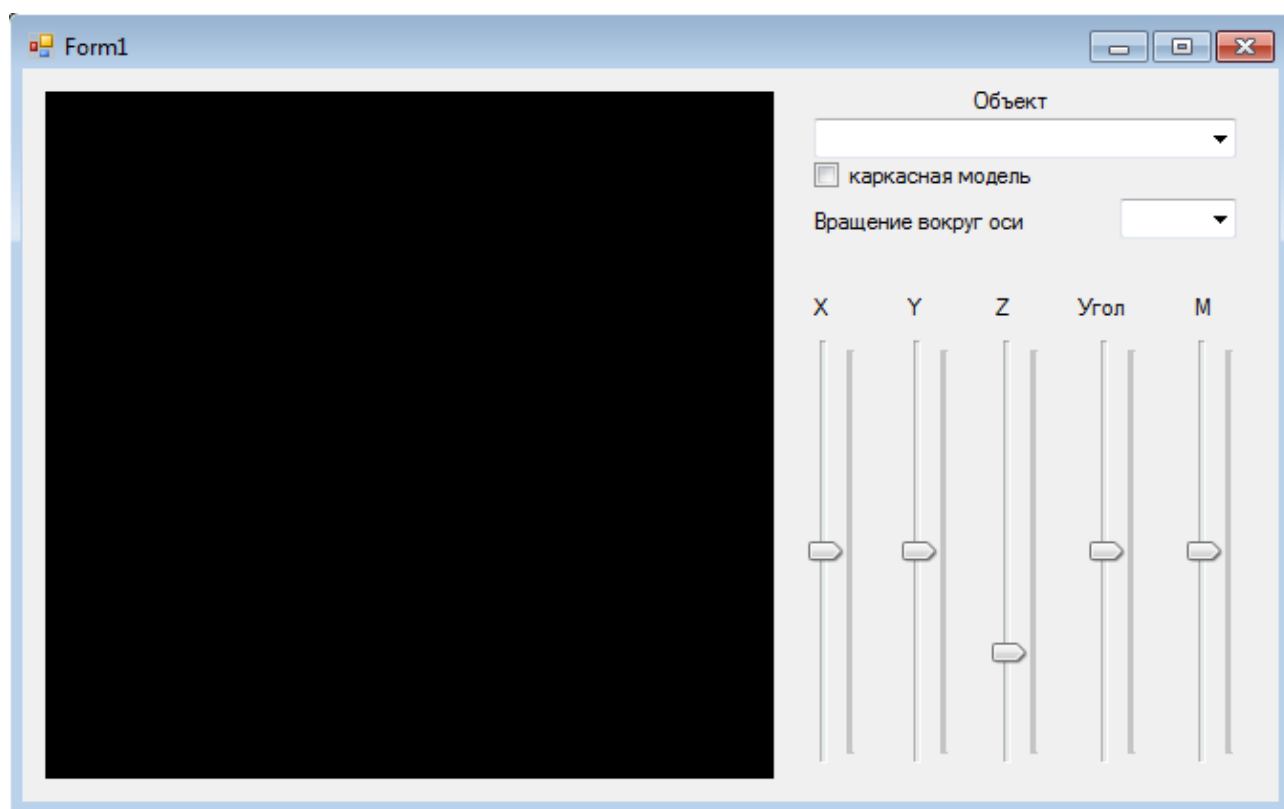


Рис. 2.12

Для элементов **combobox** установите элементы, в соответствии с тем, как показано на рис. 2.13 и 2.14.



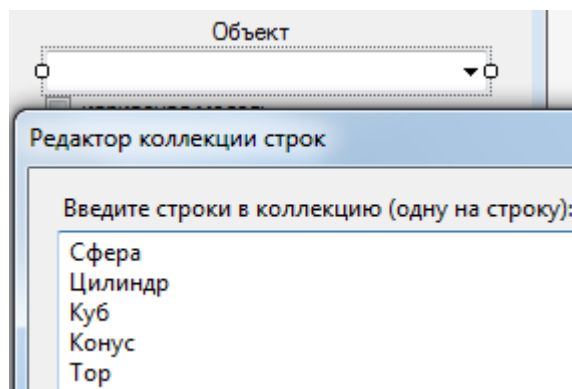


Рис. 2.13

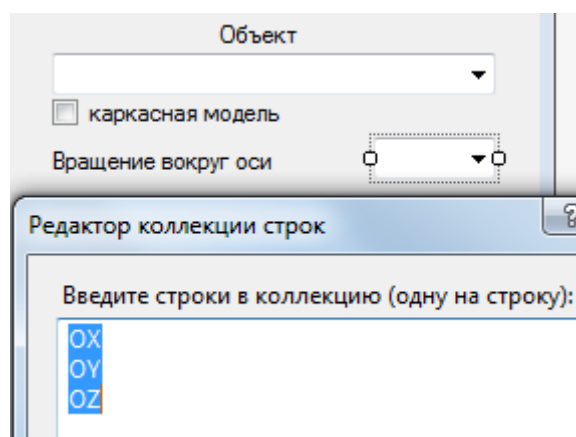


Рис. 2.14

В элементе **trackBar** для перемещений по осям установите диапазон значений от -50000 до 50000. Диапазон угла поворота от -360 до 360, для масштабирования от -5000 до 5000. Так же добавьте таймер, переименовав экземпляр в **RenderTimer**, и установите время отклика равное 30 миллисекундам.

В итоге будет создана оболочка программы со всеми необходимыми элементами управления. Принцип работы программы максимально прост. На форме можно устанавливать вид примитива, способ визуализации, а также параметры преобразований перемещения по осям OX, OY и OX. Визуализация вызывается периодически с помощью таймера, чем обеспечивается интерактивный характер преобразований.

Объявление начальных переменных и инициализация **OpenGL** показаны ниже.

```

// вспомогательные переменные для хранения текущих значений
double deltax = 0, deltax = 0, deltaz = -5;
double gamma = 0, zoom = 1;
int os_x = 1, os_y = 0, os_z = 0; // начальное значение оси вращения

// режим каркасной визуализации
bool Wire = false;

private void Form1_Load( object sender, EventArgs e)
{
// инициализация библиотеки Glut
Glut.glutInit();
// инициализация режима экрана
Glut.glutInitDisplayMode( Glut.GLUT_RGB | Glut.GLUT_DOUBLE);

// установка цвета очистки экрана (RGBA)
Gl.glClearColor(1, 1, 1, 1);

// установка порта вывода
Gl.glViewport(0, 0, AnT.Width, AnT.Height);

// активация матрицы проецирования
Gl.glMatrixMode( Gl.GL_PROJECTION);
// очистка матрицы
Gl.glLoadIdentity();

// установка перспективы
Glu.gluPerspective(45, (float)AnT.Width / (float)AnT.Height, 0.1, 200);

Gl.glMatrixMode( Gl.GL_MODELVIEW);
Gl.glLoadIdentity();

// начальная настройка параметров OpenGL (тест глубины, освещение и
// первый источник света)
Gl.glEnable( Gl.GL_DEPTH_TEST);
Gl.glEnable( Gl.GL_LIGHTING);
Gl.glEnable( Gl.GL_LIGHT0);

// установка первых элементов в списках combobox

```

```
comboBox1.SelectedIndex = 0;  
comboBox2.SelectedIndex = 0;
```

```
// инициализация таймера, вызывающего функцию для визуализации  
RenderTimer.Start();  
}
```

Необходимо добавить обработку событий изменения элементов **trackBar**, событие отклика таймера и обработку изменения значений элементов **checkbox** и **comboBox**.

```
// обрабатываем отклик таймера  
private void RenderTimer_Tick( object sender, EventArgs e)  
{  
    // вызываем функцию отрисовки сцены  
    Draw();  
}
```

```
// событие изменения значения  
private void trackBar1_Scroll(object sender, EventArgs e)  
{  
    // перевод значения, в элементе trackBar в необходимый формат  
    deltax = (double)trackBar1.Value / 1000.0;  
}
```

```
// событие изменения значения  
private void trackBar2_Scroll(object sender, EventArgs e)  
{  
    // перевод значения, в элементе trackBar в необходимый формат  
    deltay = (double)trackBar2.Value / 1000.0;  
}
```

```
// событие изменения значения  
private void trackBar3_Scroll(object sender, EventArgs e)  
{  
    // перевод значения, в элементе trackBar в необходимый формат  
    deltaz = (double)-trackBar3.Value / 1000.0;  
}
```

```
// событие изменения значения  
private void trackBar4_Scroll(object sender, EventArgs e)  
{  
    // угол поворота
```

```

    gamma = (double)trackBar4.Value / 10;
}
// событие изменения значения
private void trackBar5_Scroll(object sender, EventArgs e)
{
    // масштабный коэффициент
    zoom = (double)(trackBar5.Value + 1000.0) / 1000.0;
}
// изменения значения checkBox
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    Wire = checkBox1.Checked;
}

private void comboBox1_SelectedIndexChanged_1(object sender, EventArgs e)
// изменение в элементах comboBox
{
    os_x = 0; os_y = 0; os_z = 0;
    // в зависимости от выбранного режима
    switch (comboBox1.SelectedIndex)
    {
        // устанавливаем ось вращения (для функции glRotate)
        case 0: { os_x = 1; break; }
        case 1: { os_y = 1; break; }
        case 2: { os_z = 1; break; }
    }
}

```

Ниже приведен текст функции визуализации сцены. Обратите внимание на то, что перед геометрическими преобразованиями, значение текущей матрицы необходимо записать в стек матриц, чтобы дальнейшие преобразования затронули только визуализируемые объекты.

```

// метод визуализации
private void Draw()
{
    // очистка буфера цвета и буфера глубины
    Gl.glClear(Gl.GL_COLOR_BUFFER_BIT | Gl.GL_DEPTH_BUFFER_BIT);
}

```

```

Gl.glColor3f(1.0f, 0, 0);
// очистка текущей матрицы преобразования
Gl.glLoadIdentity();

// помещаем текущую матрицу преобразования в стек матриц,
// дальнейшие преобразования будут относиться только к выбранному
объекту
Gl.glPushMatrix();

// преобразование плоско-параллельного перемещения
Gl.glTranslated(deltaX, deltaY, deltaZ);
// поворот на угол gamma вокруг установленной оси
Gl.glRotated(gamma, os_x, os_y, os_z);
// пропорциональное масштабирование объекта
Gl.glScaled(zoom, zoom, zoom);

// в зависимости от установленного типа объекта
switch (comboBox2.SelectedIndex)
{
    // рисуем нужный объект, используя функции библиотеки GLUT
    case 0:
    {
        if (Wire)
            Glut.glutWireSphere(2, 32, 32); // каркасная сфера
        else
            Glut.glutSolidSphere(2, 32, 32); // полигональная сфера
        break;
    }
    case 1:
    {
        if (Wire)
            Glut.glutWireCylinder(1, 2, 32, 32); // цилиндр
        else
            Glut.glutSolidCylinder(1, 2, 32, 32);
        break;
    }
    case 2:
    {
        if (Wire)
            Glut.glutWireCube(2); // куб

```

```

        else
            Glut.glutSolidCube(2);
        break;
    }
case 3:
    {
        if (Wire)
            Glut.glutWireCone(2, 3, 32, 32);    // конус
        else
            Glut.glutSolidCone(2, 3, 32, 32);
        break;
    }
case 4:
    {
        if (Wire)
            Glut.glutWireTorus(0.6, 2.2, 32, 32); // тор
        else
            Glut.glutSolidTorus(0.6, 2.2, 32, 32);
        break;
    }
}

Gl.glPopMatrix();    // возвращаем состояние матрицы
Gl.glFlush();        // завершаем рисование

AnT.Invalidate();    // обновляем элемент AnT
}

```

Результат работы приложения должен быть аналогичен, показанному на рис. 2.15, 2.16.

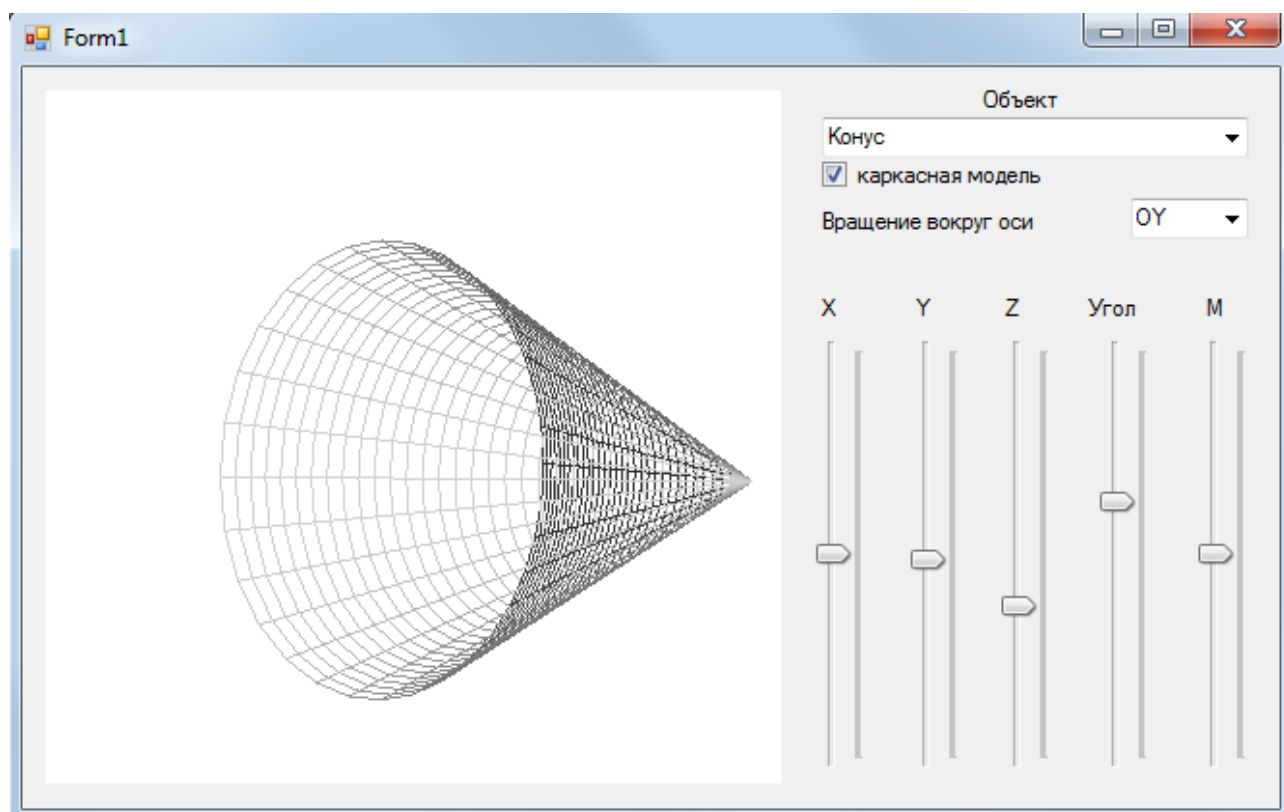


Рис. 2.15

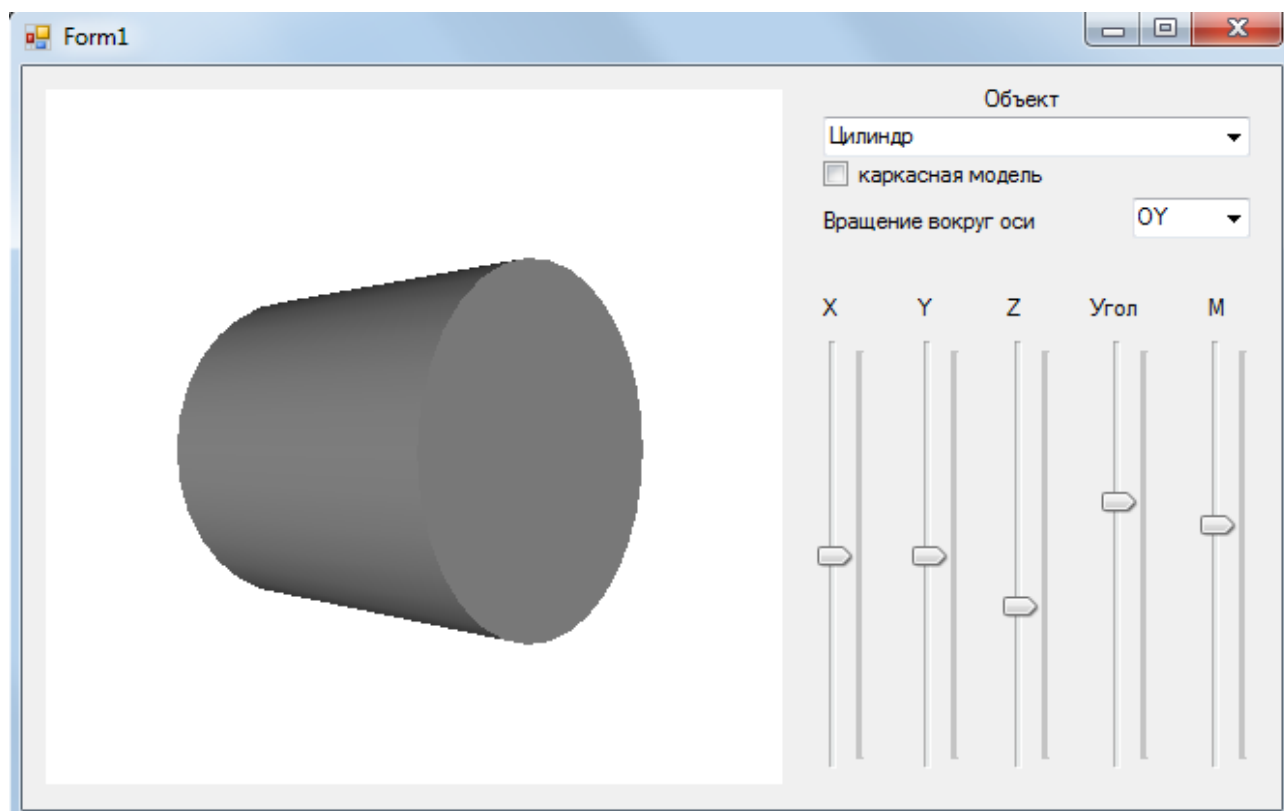


Рис. 2.16

**Содержание отчета**

Отчет должен включать

- цель работы;
- текст задания;
- описание выполняемых в соответствии с заданием действий;
- конечные результаты;
- выводы.

Результаты визуализации 3D-объектов с помощью библиотеки **Tao Framework** представить в виде скриншотов.

### Контрольные вопросы

1. Чем отличается каркасная визуализация от сплошной?
2. Какие процедуры **OpenGL** используются для геометрических преобразований плоско-параллельного перемещения, вращения, масштабирования?
3. Для каких целей используется буфер цвета и буфер глубины?



### **3. КУРСОВОЕ ПРОЕКТИРОВАНИЕ**

#### **3.1. Общие требования к выполнению курсовой работы**

Приведенные ниже задания для курсовой работы по тематике соответствуют рабочей программе курса «Графические системы компьютеров». Содержание заданий охватывает весь спектр вопросов, внесенных в рабочую программу, и направлено на приобретение навыков по самостоятельному проектированию программ, реализующих базовые алгоритмы компьютерной графики.

Для разработки программы по курсовой работе может быть использован любой язык программирования. Предпочтительнее использовать объектно-ориентированные языки высокого уровня.

Программная реализация методов и алгоритмов компьютерной графики каждым студентом должна проводиться самостоятельно. Для реализации остальных функций программы могут использоваться стандартные средства среды программирования или разработки из любых доступных проектов.

#### **3.2. Техническое задание для курсовой работы**

Разработать программу объектно-ориентированного графического редактора, обеспечивающего выполнение следующих основных функций:

- выбор, размещение на экране и визуализация примитивов из заданного набора;
- синтез более сложных фигур с помощью теоретико-множественных операций (ТМО) как над примитивами, так и над ранее синтезированными фигурами;

- выделение любого объекта, выведенного на экран, и выполнение над ним любой последовательности геометрических преобразований из заданного набора в интерактивном режиме;
- выделение любого объекта на экране и его удаление.

Уровень сложности синтезируемых фигур по числу используемых для этого ТМО не должен ограничиваться. Не должно быть также ограничений на количество примитивов любого вида или сложных фигур, выводимых на экран. При выполнении ТМО объекты – операнды данной операции следует удалять из списка самостоятельных объектов, а их изображения – стирать с экрана.

Любые геометрические преобразования нужно выполнять в однородных координатах в матричной форме и применять ко всем видам объектов.

Геометрические преобразования над каким-либо объектом не должны приводить к стиранию изображения или другим искажениям остальных объектов.

Программа должна располагать общепринятыми элементами интерфейса: системным и контекстными меню, инструментальными панелями, другими элементами диалога, системой помощи и подсказок по всем основным функциям программы. Предусмотреть диалоговые средства для выбора цвета рисования.

Программа должна быть рассчитана на работу в операционной системе Windows 7 и выше.










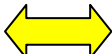






В табл. 3.1 приведен перечень примитивов, их внешнего вида в исходном состоянии и обозначений.

Все приведенные в табл. 3.1 примитивы, кроме кривых, – закрашиваемые многоугольники.

Для расчета координат вершин правильных многоугольников и звезд, а также правильного креста в исходном состоянии в программе необходимо разработать соответствующие процедуры. Координаты вершин таких фигур, как треугольники, прямоугольник, ромб, парал-

лелограмм, равнобедренная трапеция, также следует рассчитывать, задав такие параметры, как высота и ширина. Для расчета вершин параллелограмма и равнобедренной трапеции дополнительно следует задать один из углов.

Таблица 3.1

Примитивы	Вид	Обозначение
Кривая Безье		BZ
Кривая Эрмита		ER
Равнобедренный треугольник		Tgr
Прямоугольный треугольник		Tgp
Ромб		Romb
Параллелограмм		Prlg
Равнобедренная трапеция		Trp
Произвольный $n$ -угольник, $n \leq 20$		FPg
Правильный $n$ -угольник, $n \leq 20$		Pgn
Правильная $n$ -конечная звезда, $n \leq 20$		Zv
Стрелка 1		Str1
Стрелка 2		Str2
Стрелка 3		Str3
Флаг		Flag
Правильный крест		Kr
Уголок 1		Ugl1
Уголок 2		Ugl2
Уголок 3		Ugl3

Вершины таких фигур, как стрелки, флаг, уголки можно либо также рассчитывать на основании высоты и ширины, выбрав какие-либо пропорции, либо их можно задать в константах программы по чертежу фигуры в исходном состоянии.

В табл. 3.2 перечислены виды геометрических преобразований и их обозначения.

Таблица 3.2

Геометрические преобразования	Обозначение
Поворот вокруг заданного центра на произвольный угол	Rc
Поворот вокруг заданного центра на угол $30^\circ$	Rc30
Поворот вокруг центра фигуры на произвольный угол	Rf
Поворот вокруг центра фигуры на угол $60^\circ$	Rf60
Масштабирование по оси X относительно заданного центра	Sxc
Масштабирование по оси X относительно центра фигуры	Sxf
Масштабирование по оси Y относительно заданного центра	Syc
Масштабирование по оси Y относительно центра фигуры	Syf
Пропорциональное масштабирование относительно заданного центра	Sxyc
Пропорциональное масштабирование относительно центра фигуры	Sxyf
Зеркальное отражение относительно заданного центра	SPc
Зеркальное отражение относительно центра фигуры	SPf
Зеркальное отражение относительно вертикальной прямой	SV
Зеркальное отражение относительно горизонтальной прямой	SH
Зеркальное отражение относительно прямой общего положения	SL

Под заданным центром подразумевается любая точка области вывода на экране, указанная с помощью мыши, которую в дальнейшем следует использовать как центр преобразования. До окончания преобразования центр нужно показывать каким-либо условным обозначением, например, в виде перекрестья.

В качестве центра правильных многоугольников и звезд следует использовать центр описанной окружности. За центр остальных фигур может быть взят центр прямоугольника, в который вписывается

преобразуемый объект. Методика расчета центра фигуры должна быть разработана самостоятельно.

Вертикальная, горизонтальная или прямая общего положения для соответствующих видов зеркального отражения должны строиться на экране интерактивно и автоматически стираться с экрана после окончания преобразования.

### 3.3. Варианты индивидуальных заданий для курсовой работы

Варианты индивидуальных заданий для общего технического задания для курсовой работы, изложенного в разд. 3.2., приведены в табл. 3.3. Кроме перечисленных в табл. 3.3, обязательным для всех вариантов примитивом является отрезок прямой, а обязательным видом геометрических преобразований – плоскопараллельное перемещение.

Таблица 3.3

Вариант	Примитивы	Геометрические преобразования	ТМО
1	BZ, Tgr, Zv	Rc, Sxf, SH	$\cap, \oplus$
2	BZ, Prlg, FPg	Rc, SPc, SH	$\oplus, /$
3	ER, Ugl3, FPg	Rf, Rf60, SPc	$\cup, \cap$
4	BZ, Romb, Zv	Rf, Sxc, Syc	$\oplus, /$
5	BZ, Tgp, FPg	Rc, Sxc, Syf	$\cup, \oplus$
6	ER, Trp, Str1	Rf60, Syc, SL	$\cup, \cap$
7	BZ, Str1, Str2	Rc, Rc30, Sxf	$\oplus, /$
8	ER, Pgn, Ugl1	Rc, Sxyc, SPf	$\cap, \oplus$
9	BZ, Tgr, FPg	Rf, Sxyf, SH	$\cap, /$
10	ER, Tgp, Flag	Rc, SPf, SV	$\cup, /$
11	ER, Ugl2, Str3	Rf60, SPc, SH	$\cup, \oplus$
12	BZ, Trp, FPg	Rc30, Sxyf, SL	$\cap, /$
13	ER, Kr, Str2	Rc, Sxc, Syc	$\cap, \oplus$

Таблица 3.3 (Продолжение)

Вариант	Примитивы	Геометрические преобразования	ТМО
14	BZ, Pgn, FPg	Rf, SV, SH	$\cup, \cap$
15	BZ, Zv, Str1	Rf, Sxyf, SPc	$\cup, /$
16	ER, Romb, FPg	Rf60, SH, SL	$\oplus, /$
17	BZ, Pgn, Kr	Rf, SV, Sxc	$\cup, \cap$
18	BZ, Tgr, Str3	Rc30, SV, SL	$\cup, \oplus$
19	ER, Tgp, Ugl2	Rc30, Sxf, Syf	$\oplus, \cap$
20	BZ, Trp, Zv	Syf, SPc, SV	$\cup, \cap$
21	BZ, Ugl3, Str3	Rf60, Syf, SV	$\cap, /$
22	ER, Prlg, Ugl1	Rc, Rc30, Sxyf	$\cap, /$
23	BZ, Kr, Flag	Rf, Rf60, Sxc	$\cap, \oplus$
24	BZ, Tgr, Pgn	Rc30, Sxc, SL	$\oplus, /$
25	ER, FPg, Zv	Rf, Syc, SPc	$\cup, \cap$
26	ER, Trp, Flag	Rf60, SPf, SH	$\cap, /$
27	BZ, Romb, Str3	Rf60, Sxf, SH	$\cap, \oplus$
28	ER, Prlg, Str2	Spf, Sxc, SL	$\oplus, /$
29	ER, Tgp, Pgn	Rc, SV, SH	$\cup, \cap$
30	BZ, Romb, Flag	Rc, Rf60, Syf	$\cup, \oplus$
31	ER, Kr, Ugl1	Rc, Sxyc, Syf	$\oplus, \cap$
32	ER, Tgp, Str3	Rc30, Sxyf, SPf	$\oplus, /$
33	BZ, Flag, Kr	Rc30, Sxyf, SL	$\cup, /$
34	BZ, Pgn, Str1	Rc, SPc, SPf	$/, \cup$
35	ER, Ugl1, Str2	Rf, Rf60, Sxyc	$\cup, \cap$
36	ER, Ugl3, Zv	Rf, SV, SPc	$\cup, /$
37	BZ, Prlg, Ugl2	Rc, Rc30, Sxyf	$\cap, /$
38	BZ, Ugl1, Ugl2	Rc, Syf, SPc	$\oplus, \cap$
39	BZ, Romb, Kr	Rf, SV, SL	$\cup, \cap$
40	ER, Kr, Flag	Rf, Rf60, Sxc	$\cup, /$
41	BZ, Prlg, Ugl3	Rc, Sxc, Syf	$\cup, \oplus$
42	BZ, Zv, Str1	Rf, Sxyf, SPc	$\cup, /$
43	BZ, Kr, Str3	Rc30, SV, SL	$\cup, \oplus$
44	ER, Ugl3, FPg	Rf, Rf60, SPc	$\cup, \cap$

Таблица 3.3 (Продолжение)

Вариант	Примитивы	Геометрические преобразования	ТМО
45	BZ, Tgr, FPg	Rf, Sxyf, SH	$\cap, /$
46	ER, Flag, Str1	Rf60, Syc, SL	$\cup, /$
47	BZ, Tgp, Ugl1	Rf, Sxc, Syf	$\cup, \oplus$
48	BZ, Pgn, Kr	Rc, SPc, SPf	$/, \cup$
49	ER, Tgp, Pgn	Rc, SV, SH	$\cup, \oplus$
50	ER, Trp, Str1	Rf60, Syc, SL	$\cup, \cap$
51	ER, Tgp, Flag	Rc, SPf, SV	$\cup, /$
52	BZ, FPg, Ugl3	Syc, SPc, SH	$\cup, \cap$
53	BZ, Ugl2, Str3	Rf30, Syf, Syc	$\cap, /$
54	ER, Prlg, Str2	Spf, Sxc, SL	$\oplus, /$
55	BZ, Pgn, Tgr	Rc, SPc, SPf	$\oplus, \cup$
56	ER, Romb, Str1	Rc, SPc, SPf	$/, \cup$
57	BZ, Trp, Kr	Rf, SV, Sxc	$\cup, \oplus$
58	ER, Trp, Zv	Rc, Sxyf, SPc	$\cup, \cap$
59	ER, Tgp, Str3	Rc30, Sxyf, SPf	$\oplus, /$
60	BZ, Kr, Flag	Rf, Rf60, Sxc	$\cap, \oplus$

Заданные наборы геометрических преобразований и ТМО должны быть применимы к любому объекту, выведенному на экран.

### 3.4.Порядок защиты курсовой работы

В качестве результата выполнения курсовой работы должны быть представлены:

- пояснительная записка, оформленная в соответствии со стандартами университета;
- действующий вариант программы, разработанной в соответствии с требованиями технического задания;
- комплект файлов на диске или дискете, содержащий все материалы по разработке: тексты программных модулей,

вспомогательные файлы с текстами, пиктограммами и т.п., которые необходимы для компиляции программы и ее работы.

Пояснительная записка должна включать в себя следующие разделы:

- техническое задание;
- разработка типов данных для представления примитивов и сложных объектов, конструируемых с помощью ТМО;
- разработка способов задания параметров примитивов и процедур их визуализации;
- программная реализация ТМО;
- программная реализация непрерывных геометрических преобразований;
- руководство пользователя;
- библиографический список;
- оглавление.

Объем пояснительной записки должен составлять 15-20 страниц.

При представлении курсовой работы студент должен защищать проект программы, реализованной самостоятельно в соответствии с техническим заданием.

Защита курсовой работы включает демонстрацию всех функций программы, предусмотренных индивидуальным заданием, и доклад с обоснованием проектных решений в части программной реализации методов и алгоритмов компьютерной графики. Демонстрацию нужно сопровождать пояснениями, касающимися сути реализованных функций и результатов их применения. Доклад должен содержать конкретные пояснения по выбору основных структур данных, по методике реализации процедур и функций, по назначению различных переменных программы. Обоснование методик должно опираться на теоретический материал курса лекций или материалы из библиографических источников.



При защите оценивается степень выполнения технического задания, качество реализации заданных функций, уровень теоретических знаний в обосновании конкретных проектных решений в программе.

## Библиографический список

1. *Божко А. Н., Жук Д. М., Маничев В. Б.* Компьютерная графика – М: МГТУ им. Н. Э. Баумана, 2007. – 392 с.
2. *Никулин Е.А.* Компьютерная геометрия и алгоритмы машинной графики – СПб.: БХВ-Петербург, 2003. . – 560 с.
3. *Порев В.Н.* Компьютерная графика: Учеб. пособие. – СПб.: БХВ-Петербург, 2004. . – 432 с.
4. *Пугачев А.И.* Основы компьютерной графики. Курс лекций. / Самар. гос. техн. ун-т. – Самара, 2010. – 104 с.
5. *Роджерс Д., Адамс А.* Математические основы машинной графики – М.: Мир, 2001. . – 604 с.
6. *Баяковский Ю.М., Игнатенко А.В.* Начальный курс OpenGL – М.: Планета Знаний, 2007. – 221с.
7. <http://www.esate.ru/page/uroki-OpenGL-c-sharp/>
8. <http://www.opengl.org/registry/#apispecs>

## Оглавление

---

1. Элементы теории компьютерной графики	2
1.1. Визуализация отрезков прямых	2
1.2. Сплаины	5
1.3. Алгоритмы закрашивания многоугольников	9
1.4. Теоретико-множественные операции над двумерными областями	19
1.5. Непрерывные геометрические преобразования	24
1.6. <b>OpenGL</b> и библиотека <b>Tao Framework</b>	28
2. Лабораторные работы	33
1.1. Общие требования к выполнению лабораторных работ	33
2.2. Лабораторная работа № 1	34
2.3. Лабораторная работа № 2	35
2.4. Лабораторная работа № 3	36
2.5. Лабораторная работа № 4	38
2.6. Лабораторная работа № 5	39
2.6. Лабораторная работа № 6	41
2.6. Лабораторная работа № 7	49
2.6. Лабораторная работа № 8	55
3. Курсовое проектирование	64
3.1. Общие требования к выполнению курсовой работы	64
3.2. Техническое задание для курсовой работы	64
3.3. Варианты индивидуальных заданий для курсовой работы	68
3.4. Порядок защиты курсовой работы	70
Библиографический список	73

## **Графические системы компьютеров**

Составитель: *ПУГАЧЕВ Анатолий Иванович*

Подписано в печать 31.05.13  
Формат 60x84  $\frac{1}{16}$ . Бум. офсетная.  
Печать офсетная.  
Усл. п. л. 4,42. Уч.-изд. л. 4,38.  
Тираж 50 экз.

.

---

МИНОБРНАУКИ РОССИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
«Самарский государственный технический университет»  
443100, г. Самара, ул. Молодогвардейская, 244. Главный корпус