

Федеральное агентство по образованию

---

Санкт-Петербургский государственный университет  
информационных технологий, механики и оптики

Г.Г. Хайдаров

**Примеры выполнения самостоятельных работ  
по компьютерной геометрии и графике**

Методические указания к самостоятельным работам



Санкт-Петербург

2006

УДК 681.3

Хайдаров Г.Г. Примеры выполнения самостоятельных работ по компьютерной геометрии и графике,: Метод. указания. СПб., СПбГУ ИТМО, 2005. -52 с.

В данных методических указаниях к самостоятельным работам собраны примеры программирования трехмерных геометрических моделей по дисциплине «Компьютерная геометрия и графика».

Методические указания предназначены для студентов 1 курса специальности 654700- информационные системы и соответствуют рабочей программе дисциплины «Компьютерная геометрия и графика».

Ил. 32, табл.10, библиогр. 17 назв.

Рецензент: А.В. Меженин, ст. преп., кафедра инженерной и компьютерной графики СПбГУ ИТМО

Утверждены на заседании учебно-методической комиссии факультета информационных технологий и программирования 19.04.2005 г.

© Санкт-Петербургский государственный университет информационных технологий, механики и оптики, 2006

© Г.Г. Хайдаров, 2006

# Введение

Под названием дисциплины «Компьютерная геометрия и графика» (КГГ) на практике часто понимают ряд смежных дисциплин, связанных с компьютерным дизайном, компьютерным черчением, геометрическим моделированием, и программированием «быстрой» графики. Мы будем употреблять это название в соответствии с требованиями государственного образовательного стандарта высшего профессионального образования к обязательной дисциплине 2 – го семестра (1 курса) университетов, готовящих студентов по компьютерным специальностям. В первую очередь по специальностям, связанным с информационными технологиями и программированием, а также с автоматизацией.

Прототипом данной дисциплины был курс под названием «Машинная графика» [1], опубликованный в 1991 году. Другим прототипом был курс под названием «Вычислительная и компьютерная геометрия», который появился в ВУЗах страны с 1991 года. Курс дисциплины под названием «Компьютерная графика» был опубликован в России с 1996 года и предназначался для студентов Московского государственного университета имени М.В. Ломоносова [2,3]. В настоящее время только в сети русскоязычного интернета опубликовано около 20 курсов данного направления. В основном это сайты крупных университетов России. Сейчас почти в каждом крупном книжном магазине на полках стоят книги по курсу «Компьютерная графика» [4].

Данные самостоятельные работы основаны на дисциплинах: информатика, программирование, высшая математика, основы инженерной графики. Для применения программирования в данном курсе можно порекомендовать следующие книги по основам языка C++ [5-8] и по C++ Builder [9-12]. Однако в данном курсе от студентов требуется комплексное применение знаний этих дисциплин, что вызывает определенные трудности. Для преодоления этих трудностей написаны данные методические указания с примерами выполнения самостоятельных работ на языке программирования C++. Автор начал работы по созданию данного курса с 1991 года [13-17]. До настоящего времени курс преподавался школьникам в центре компьютерных технологий отдела техники ГДТЮ (Аничков дворец) и для студентов третьего курса факультета информатики и управления в СПбГТИ(ТУ).

Данные методические указания включают минимальный набор примеров собранных вместе по дисциплине «Компьютерная геометрия и графика». Они, несомненно, помогут студентам сориентироваться в типовом программировании трехмерных геометрических моделей.

## 1. Сравнение синтаксиса графических функций языка C++ в зависимости от среды программирования

Для того чтобы студент мог свободно пользоваться литературой с описанием листингов программ на языке C++, проведем сравнение графических функций на языке C++ в средах Borland C++ 3.0 (под DOS), в Borland C++ Builder (под Windows) и в MS Visual C++ на примере функции рисования прямоугольника (rectangle).

В среде DOS названия графических функций пишутся прописными латинскими буквами, что принципиально для языка C++, в отличие от языка Pascal. Заглавная и прописная буквы в языке C++ воспринимаются как два разных символа. Рассмотрим строку кода C++ для рисования прямоугольника (функция `rectangle`) на дисплее компьютера после инициализации графического режима его работы.

```
rectangle(X1, Y1, X2, Y2);
```

Параметры  $X1$ ,  $Y1$  указывают на верхний левый, а  $X2$ ,  $Y2$  — на правый нижний угол прямоугольника.

В 1997 году появился визуальный вариант среды для языка C++, который также был создан компанией Borland и получил название C++ Builder. Позже свой взгляд на визуализацию C++ был предложен компанией Microsoft, которая, вслед за Visual Basic, выпустила среду разработки Visual C++. Таким образом, сейчас распространены два визуальных языка C++. Трудно отдать предпочтение C++ Builder или Visual C++. Достоинствами C++ Builder являются автоматическое создание классов с большим выбором готовых свойств созданного класса и вызов функции (метода) по указателю. Достоинствами Visual C++ являются тесное взаимодействие с операционной системой Windows через функции Win32 API.

Рассмотрим строку кода C++ Builder для рисования прямоугольника (*Rectangle*) на поверхности «холста» (*Canvas*) формы (*Form1*) приложения.

```
Form1->Canvas->Rectangle(X1, Y1, X2, Y2);
```

Здесь функция **Rectangle** рисует прямоугольник, используя выбранное перо, и закрашивает его внутренность с помощью выбранной кисти. Функция возвращает значение: не ноль, если прямоугольник нарисован; ноль — в противном случае.

Рассмотрим фрагмент кода Visual C++ для рисования прямоугольника на форме приложения с функцией Win32 API:

```
Rectangle(HDC, X1, Y1, X2, Y2);
```

Здесь функция **Rectangle** рисует прямоугольник, используя выбранное перо, и закрашивает его внутренность с помощью выбранной кисти. Параметры функции: *HDC* — идентификатор контекста устройства (или дескриптор окна),  $X1$ ,  $Y1$  и  $X2$ ,  $Y2$  — координаты верхнего левого и правого нижнего угла прямоугольника. Функция возвращает не нулевое значение, если прямоугольник нарисован и ноль — в противном случае. Функция находится в файле *gdi32.dll*.

Сравнивая два последних синтаксиса функций можно заметить различие в их описании места вывода прямоугольника. В большинстве случаев достаточно определить местом вывода графических функций рабочую форму приложения (*Form1->Canvas->*) — тогда нет смысла в каждой функции описывать дополнительный параметр *HDC*, как это и реализовано в C++ Builder. Но, если вы хотите создать фокус, например, перевернуть на вашем дисплее изображение «рабочего стола», то для реализации необходимо использовать функции Win32 API. В этом нетипичном для C++ Builder случае необходимо будет подумать, как определять указатель на контекст дисплея. В любом случае обязательно надо получить и использовать идентификатор (дескриптор) контекста устройства вне зависимости, на чем вы пишете программу: на C++ Builder или Visual C++.

Чтобы закончить сравнение C++ Builder и Visual C++ с точки зрения автора отмечу лишь одно положительное свойство C++ Builder. В нем учащемуся приходится постоянно видеть синтаксис написания производных классов.

Наибольшие трудности при освоении дисциплины «Компьютерная геометрия и графика» для студентов первого курса представляет раздел, связанный с методами трехмерного представления, математического описания и программирования трехмерных объектов в среде программирования под Windows. В данный раздел входят самостоятельные работы с номерами 1 - 3 включительно.

Кроме того, в данных методических указаниях подробно рассмотрены лабораторные работы 1 и 2, как подготовительные к работе в среде программирования C++ Builder под Windows. Студент, чувствующий себя

уверенно в программировании C++ под Windows, может не читать следующие два раздела.

## 2. Функции базовой графики языка C++

Для того чтобы студент мог свободно читать тексты программ на языке C++ желательно ознакомиться с тем положительным опытом программирования графики, накопленным в DOS средах. Все лучшее созданное там было перенесено в Windows программирование. Дадим сравнение программирования графики в DOS и Windows средах.

### 2.1 Базовая графика в среде Borland C++ 3.1

В настоящее время во всех современных компьютерах применяется графический режим вывода информации, а операционная система Windows не знает символьного режима. В языке программирования C++ с реализацией под операционную систему DOS графические функции хранятся в библиотеке **graphics.lib**, а прототипы (объявления) этих функций находятся в файле **graphics.h**. Мы будем рассматривать графические функции не в отрыве от практического применения, но наоборот обратим особое внимание на логику составления графических программ. Это понадобится вам при изучении технологии **DirectDraw** в C++ Builder под Windows, так как в технологии **DirectDraw** для обеспечения быстрого вывода изображения на дисплей переняты лучшие воплощения графических функций для операционной системы DOS.

#### 2.1.1 Инициализация графического режима

Самым первым вопросом работы с графикой является проверка работоспособности графического режима в вашей среде программирования. Следующий пример (листинг 4.1) написан именно с данной целью. Скомпилируйте его.

**Листинг 2.1.** Демонстрация работы графического режима

```
// probal.cpp
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* 1. Автоматическое определение наибольшего графического
    режима: gdriver = DETECT */
    int gdriver = DETECT, gmode, errorcode;
    /* Инициализация графического режима */
    initgraph(&gdriver, &gmode, "");
    /* Результат инициализации */
    errorcode = graphresult();
    if (errorcode != grOk) /* Если ошибка */
    {
        printf("Ошибка инициализации: %s\n", grapherrormsg(errorcode));
        printf("Нажмите любую клавишу");
        getch(); exit(1); /* Завершить программу */
    }
}
```

```

}
/* 2. Рисование линии из правого верхнего угла в левый нижний */
line(0, 0, getmaxx(), getmaxy());
/* 3. Заккрытие графического режима */
getch(); closegraph();
return 0;
}

```

Если вы не внесли опечаток в текст программы, тогда файл **proba1.exe** начнет работать. По диагонали экрана должна рисоваться на черном фоне белая линия из правого верхнего угла в левый нижний угол.

### 2.1.2 Типовые ошибки инициализации

Возможны три причины, по которым программа **proba1.exe** с графикой будет работать не так, как мы предположили.

Первая — отсутствие в рабочей папке с исполняемым файлом **proba1.exe** файла драйвера **egavga.bgi**. Результатом работы программы будет сообщение:

Ошибка инициализации: Device driver file not found <EGAVGA.BGI>

Нажми любую клавишу (Press any key to halt)

Это как-то даже неуместно называть проблемой. Из папки **BGI** среды программирования файл **egavga.bgi** переписывается в вашу рабочую папку и все.

Другой вариант. Файл **egavga.bgi** «лежит» в папке рядом с файлом **proba1.exe**, но линия не рисуется. Если программа **proba1.exe** начинает вытворять неизвестно что (или выдает другую ошибку или по экрану начинает «бегать» курсор), тогда смело сотрите старый драйвер **egavga.bgi** и поищите у знакомых или на дисках новую версию. «Ломанные» драйверы встречаются не так редко.

Третья причина — это экзотический «баг». Она встречается в основном в компьютерных сетях. По неизвестной причине файл драйвера **egavga.bgi** отказывается работать до тех пор, пока с этого файла не снять атрибут «только для чтения». Надеюсь, что вы не работаете с удаленного компьютера в сети, поскольку там есть еще ряд особенностей работы среды программирования, связанных с загрузкой среды с другого компьютера и атрибутами «только для чтения».

### 2.1.3 Графические примитивы

Основу мы уже заложили: знаем, как переключиться в графический режим, где применить функции и как правильно завершить работу графической части. Теперь займемся непосредственно разбором примеров с рисованием графических функций. Самые простые функции называются графическими примитивами. Условно примитивы можно разделить на группы: примитивы рисования контуров и площадные фигуры. К примитивам контуров относятся: линии (**line**), прямоугольники (**rectangle**), дуги (**arc**), окружности (**circle**), эллипсы (**ellipse**), многоугольники (**drawpoly**) и прочие не закрашиваемые внутри фигуры. Площадными, закрашиваемыми внутри фигурами являются прямоугольники (**bar**, **bar3d**), круговые и эллиптические секторы (**pieslice**, **sector**). Если у вас фигура замкнута, но не залита, например, многоугольник (**drawpoly**), то его всегда можно закрасить внутри с помощью функций закрашки (**floodfill**, **setfillstyle**). Для выбора цвета рисования применяется функция задания текущего (по умолчанию) цвета графических примитивов (**setcolor**). Для определения цвета точки по ее координатам на экране существует функция **getpixel**, а для

вывода точки заданным цветом на экран — `setpixel`. Вот и весь смысл работы с выводом графических примитивов. Данная логика интуитивно понятна обычному школьнику. Подробное описание и параметры графических функций можно найти в любом справочнике по C++. Если у вас под рукой нет справочника, тогда наберите в редакторе среды программирования интересующее вас имя функции, наведите на него курсор и нажмите комбинацию клавиш `Ctrl+F1`. На экране появится описание функции и внизу подсказки ссылка на пример программы с выбранной графической функцией. Например, если вы введете имя функции `rectangle` и нажмете нужные клавиши, появится подсказка по функции рисования прямоугольника с характерным примером применения. Но кое-что вы можете увидеть и в примере из листинга 2.2, в котором воедино собраны некоторые распространенные функции для работы с примитивами.

**Листинг 2.2.** Демонстрация работы графических функций

```
// rectangl.cpp
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void main(void)
{int gdriver = DETECT, gmode, errorcode;
int xr=50,yr=20,left, top, right, bottom;
int stangle=0, endangle=360, xradius=40, yradius=25;
int radius=25;
int stangle_arc = 90, endangle_arc = 225;
int xb=0, yb=200, midx, midy;
// 1 Инициализация графического режима
initgraph(&gdriver, &gmode, ""); errorcode = graphresult();
if (errorcode != grOk) {
printf("Ошибка: %s\n", grapherrormsg(errorcode));
printf("Нажмите любую клавишу"); getch(); exit(1); }
// 2 Рисование
setcolor(WHITE);
outtextxy(xr,yr+100,
"rectangle      ellipse      circle      arc");
left = xr +0; top = yr +0;
right = xr+100; bottom = yr+50;
rectangle(left,top,right,bottom);
ellipse(xr+150, yr+25, stangle, endangle, xradius, yradius);
circle(xr+250, yr+25, radius);
arc(xr+350, yr+25, stangle_arc, endangle_arc, radius);
midx = xb; midy = yb;
```

```

outtextxy(xb+100, yb+60, "bar + setfillstyle");
outtextxy(xb+100, yb+200, "sector + setfillstyle");
for (int i=SOLID_FILL; i<USER_FILL; i++)
{ setfillstyle(i, YELLOW);
bar(midx+i*50, midy, midx+40+i*50, midy+40);
sector(midx+i*50+35, midy+150, stangle_arc, endangle_arc,
xradius, yradius);
}
// 3 Закрытие графического режима
getch(); closegraph(); }

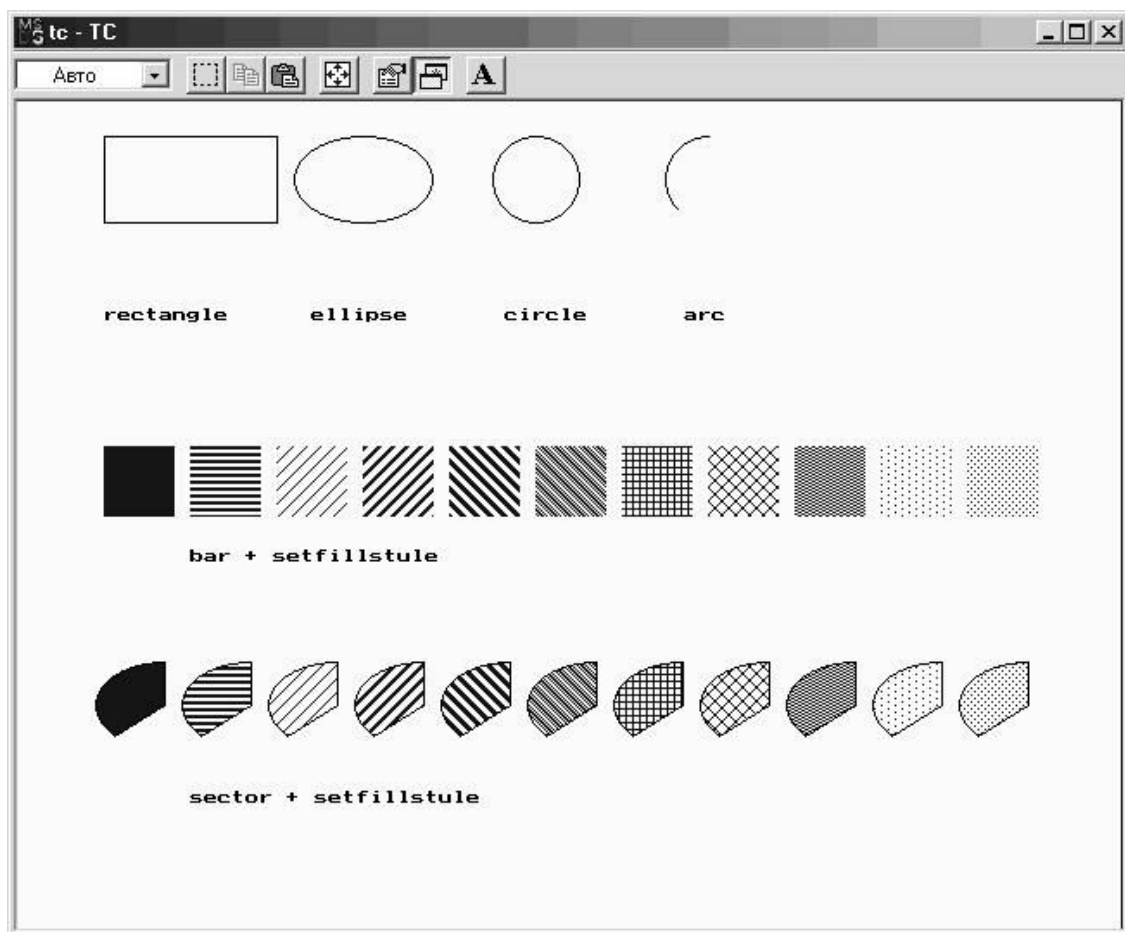
```

В этой простой программе обратите внимание на перечисление типов закрашки площадных фигур в цикле `for (int i=SOLID_FILL; i<USER_FILL; i++)` от `i=1` до `i=11`. Всего существует 13 типов стандартных заливок замкнутых областей, назначение которых перечислено в табл. 4.1.

**Таблица 2.1.** Типы стандартных заливок замкнутых областей

I=0	ENUM_FILL	Заливка цветом фона
I=1	SOLID_FILL	Заливка текущим цветом (установленным <code>setcolor()</code> )
I=2	LINE_FILL	Горизонтальная штриховка
I=3	LTSLASH_FILL	Штриховка тонкими линиями под углом 45° влево
I=4	SLASH_FILL	Штриховка толстыми линиями под углом 45° влево
I=5	BKSLASH_FILL	Штриховка толстыми линиями под углом 45° вправо
I=6	LTBKSLASH_FILL	Штриховка линиями под углом 45° вправо
I=7	HATCH_FILL	Штриховка в клетку
I=8	XHATCH_FILL	Штриховка в клетку под углом 45°
I=9	INTERLEAVE_FILL	Частая штриховка в клетку под 45°
I=10	WIDEDOT_FILL	Заполнение редкими точками
I=11	CLOSEDOT_FILL	Заполнение частыми точками
I=12	USER_FILL	По шаблону программиста

На рис. 2.2 показана экранная копия работы программы `rectang1.cpp`, иллюстрирующая перечисленные способы заполнения внутренних областей.



**Рисунок 2.1.** Демонстрация работы графических функций

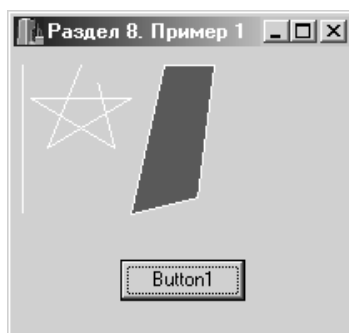
По вопросам анимации графики, применения других графических драйверов, инициализации графического режима через BIOS читателям следует обратиться к литературе [5-9].

## **2.2 Базовая графика в среде C++ Builder**

Базовая графика холста (Canvas) C++ Builder, предназначенная для рисования графических примитивов на поверхности «холста» формы, вобрала в себя все лучшие достижения графики DOS. Плюс качественно упростилась работа с цветными графическими режимами. Например, с 24-битовой глубиной цвета. Однако это не означает, что, взмахнув волшебной палочкой, вы можете добиться любого желаемого графического эффекта — ускорение разработки вы получите только за счет стандартных средств среды C++ Builder. Именно эти стандартные варианты работы и будут рассмотрены в данной главе.

### **2.2.1 Функции Canvas-графики**

В Windows насчитывается несколько десятков графических функций. Их названия частично совпадают с DOS-именами аналогичных средств. Особое внимание надо обратить на заглавные первые буквы в названиях этих функций, что принципиально при наборе программ в C++. Для обзора предлагаемых возможностей для работы с графикой войдите в меню **Help** (справка) главного меню C++ Builder и выберите пункт **Index**. Наберите слово **Canvas** и в результате поиска укажите слово **Methods**, то есть, говоря в строгом соответствии с терминологией C++ — функции для работы с графикой. Вам будет предложен в алфавитном порядке набор функций для работы с графикой: **Arc**, **BrushCopy**, **Chord**, **CopyRect**, **Draw**, **DrawFocusRect**, **Ellipse**, **FillRect**, **FloodFill**, **FrameRect**, **LineTo**, **Lock**, **MoveTo**, **Pie**, **PolyBezier**, **PolyBezierTo**, **Polygon**, **Polyline**, **Rectangle**, **Refresh**, **RoundRect**, **StretchDraw**, **TextExtent**, **TextHeight**, **TextOut**, **TextRect**, **TextWidth**, **TryLock**, **Unlock**. Видите, часть этих названий такие же, что и в DOS.



**Рисунок 2.2.** Экранная копия работы программы из примера 2.3

На рис. 2.2. показана экранная копия работы программы, воспроизведением которой мы сейчас займемся. В визуальную часть программу входит, кроме формы `Form1`, один объект — кнопка `Button1`. Вывод графических примитивов происходит на поверхность формы. Эту поверхность принято называть холстом, то есть `Canvas`. В данном примере рисуется линия, полилиния по массиву структур точек (`Point`) и многоугольник (`Polygon`). Цвет задает свойство «пера» для рисования — `Pen->Color`. Заливка определяется свойством кисти для рисования `Brush->Color`. Во время работы готовой программы при нажатии кнопки `Button1` происходят:

- рисование линии. Невидимое перо перемещается в точку с координатами  $X=5$ ,  $Y=10$ . Далее опущенное на холст перо движется в точку с координатами  $X=5$ ,  $Y=100$ ;
- рисование полилинии. Невидимое перо перемещается в точку с координатами начальной (нулевой) точки `points[0].x = 40` и `points[0].y = 10`. Далее опущенное на холст перо движется по координатам массива точек от 0 до 5;
- рисование замкнутого многоугольника. Перо перемещается в точку с координатами начальной (нулевой) точки `points2[0].x = 40` и `points2[0].y = 10`, затем движется по координатам массива точек от 0 до 3 и далее до нулевой точки, замыкая многоугольник. Для многоугольника, как площадной фигуры, возможна внутренняя заливка кистью `Brush`.

### Листинг 2.3. Простейшие графические функции

```
// -- Uch8_n01.cpp --
#include <vcl.h>
#pragma hdrstop
#include "Uch8_n01.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner){ }
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{ // Линия
Canvas->Pen->Color = clWhite;
```

```

Canvas->MoveTo(5,10); Canvas->LineTo(5,100);
// Полилиния
TPoint points[6]; Canvas->Pen->Color = clWhite;
points[0].x = 40; points[0].y = 10;
points[1].x = 20; points[1].y = 60;
points[2].x = 70; points[2].y = 30;
points[3].x = 10; points[3].y = 30;
points[4].x = 60; points[4].y = 60;
points[5].x = 50; points[5].y = 20;
Form1->Canvas->Polyline(points,5);
// Многоугольник
TPoint points2[4];
points2[0] = Point(90,10);
points2[1] = Point(120,10);
points2[2] = Point(110,90);
points2[3] = Point(70,100);
Form1->Canvas->Brush->Color = clTeal;
Form1->Canvas->Polygon(points2, 3);
}

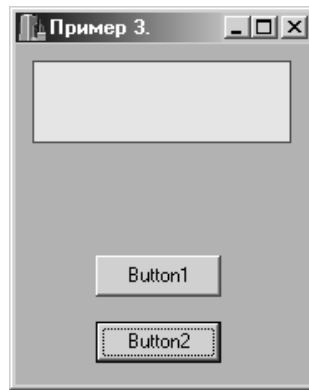
```

Чтобы вам не было скучно перечитывать текст программы, дополним ее такими свойствами пера **Pen**, как ширина линии **Width** и стиль прорисовки линии **Style**. По умолчанию эти свойства принимают значения: **Width=1**, **Style=psSolid**.

### **2.2.2 Заливка областей**

После разбора программы с прорисовкой линий займемся изучением примеров рисования графических примитивов с закраской замкнутых площадей. Графические примитивы принято разделять на две группы: рисование контуров и площадные (заливаемые) фигуры. К примитивам контуров можно отнести: линии (**MoveTo**, **LineTo**), прямоугольники (**Rectangle**), дуги (**Arc**), окружности и эллипсы (**Ellipse**), многоугольники (**Poligon**) и другие, не закрашиваемые внутри фигуры. К площадным, заполняемым внутри фигурам, относятся закрашиваемые прямоугольники (**FillRect**), круговые и эллиптические секторы (**Pie**). Если у вас рисуемая фигура замкнута, но не закрашиваемая, например, многоугольник (**Poligon**), то его всегда можно заштриховать с помощью функций закрашки (**FloodFill**, **FillStyle**). Для выбора цвета рисования применяется функция задания цвета графических примитивов (**Pen->Color**). Вот и вся логика работы с выводом графических примитивов в Windows. Данная логика вобрала в себя опыт, накопленный в DOS. Подробное описание и параметры графических функций можно найти в справочнике по C++ Builder. Если у вас нет справочника, тогда наберите в окне редактора кода интересующее вас слово, наведите на него курсор мыши и нажмите клавишу F1. На экране появится описание функции, как правило, с примером программы для выбранной графической функции.

Рассмотрим пример (рис. 2.3 и листинг 2.4) с заливкой замкнутой фигуры цветом.



**Рисунок 2.3** Экранная копия работы программы из примера 2.4

**Листинг 2.4** Заливка замкнутой области

```
// -- UCh8_n03.cpp --
#include <vcl.h>
#pragma hdrstop
#include "UCh8_n03.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//-----
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner){ }
//-----

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Canvas->Pen->Color = clRed;
    Form1->Canvas->Brush->Color = clTeal;
    Canvas->Rectangle(10, 10, 165, 60);
}

//-----

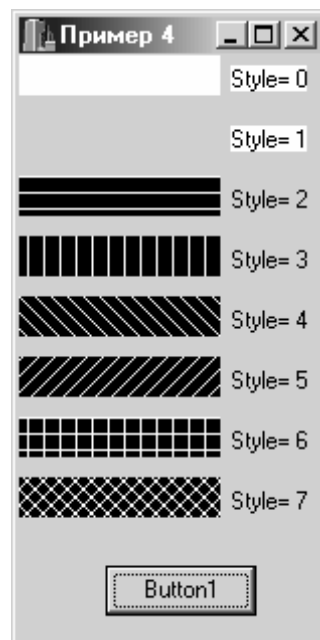
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Form1->Canvas->Brush->Color = clYellow;
    Canvas->FloodFill(30, 30, clRed, fsBorder);
    Form1->Canvas->Brush->Color = clAqua;
    Canvas->FloodFill(3, 30, clRed, fsBorder);
}

//-----
```

В программе при нажатии первой кнопки рисуется покрашенный прямоугольник. Если в ходе выполнения программы необходимо его залить другим цветом, можно использовать функцию заливки **FloodFill**. В данной функции указываются координаты точки начала

заливки, признак заливки до границы **fsBorder** с указанным цветом. В примере при нажатии первой кнопки происходит рисование закрашенного прямоугольника с границей красного цвета (**clRed**). При нажатии второй кнопки внутренняя область заливается желтым (**clYellow**) цветом до границы прямоугольника **fsBorder**. Начинается заливка с координаты (30, 30). И далее происходит заливка, начиная с координаты (3, 30), голубым (**clAqua**) цветом до границы **fsBorder** с цветом **clRed**, то есть области, внешней по отношению к прямоугольнику.

Теперь разберем конкретный пример (рис. 2.4 и листинг 2.5) со сменой стилей закрашки прямоугольной области.



**Рисунок 2.4.** Экранная копия работы программы из примера 2.5

#### **Листинг 2.5** Типы заливок областей

```
// -- UCh8_n04.cpp --
#include <vcl.h>
#include <stdio.h>
#pragma hdrstop
#include "UCh8_n04.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner){ }
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{ char ss[10];
  for(int S=0;S<8;S++){
```

```

sprintf(ss, "Style= %d", S);
Canvas->TextOut(105, 5+S*30, ss);
if (S == bsSolid)
Canvas->Brush->Style = bsSolid;
else if (S == bsClear)
Canvas->Brush->Style = bsClear;
else if (S == bsHorizontal)
Canvas->Brush->Style = bsHorizontal;
else if (S == bsVertical)
Canvas->Brush->Style = bsVertical;
else if (S == bsFDiagonal)
Canvas->Brush->Style = bsFDiagonal;
else if (S == bsBDiagonal)
Canvas->Brush->Style = bsBDiagonal;
else if (S == bsCross)
Canvas->Brush->Style = bsCross;
else if (S == bsDiagCross)
Canvas->Brush->Style = bsDiagCross;
Form1->Canvas->FillRect(Rect(0, 0+S*30, 100, 20+S*30));
};
}

```

В этой простой программе обратите внимание на перечисление типов заливки площадных фигур в цикле `for(int S=0; S<8; S++)` от `S=0` до `S=7`. Всего существует 9 стилей стандартных заливок (см. табл. 2.2).

**Таблица 2.2.** Способы заливки замкнутых областей

Номер	Название	Типы заливки области
S=0	bsSolid	Заливка цветом фона
S=1	bsClear	Прозрачная заливка
S=2	bsHorizontal	Горизонтальная штриховка
S=3	bsVertical	Вертикальная штриховка
S=4	bsFDiagonal	Прямая штриховка линиями под 45°
S=5	bsBDiagonal	Обратная штриховка линиями под 45°
S=6	bsCross	Штриховка в клетку
S=7	bsDiagCross	Штриховка в клетку под 45°
S=8	brushBmp	По шаблону программиста

На этом краткой обзор графических функций C++ Builder можно считать оконченным, а любопытных отошлем к литературе [10-13].

### 3. Лабораторная работа 1. Основные функции базовой графики DOS.

Данная лабораторная работа нужна студенту, как подготовительная работа, для приобретения навыков чтения программ на языке C++.

#### 3.1 Цели работы

- Познакомиться с конкретным примером выполнения программы на языке программирования C++.
- Познакомиться с правилами работы в локальной компьютерной сети по работе и сдаче файлов выполненных заданий.
- Познакомиться с правилами оформления файлов отчетов по лабораторным и самостоятельным работам.
- Выполнить задание по указанию преподавателя.

#### 3.2 Задание

По номеру варианта нарисовать указанные графические примитивы для контурных и площадных (с заливкой) фигур. Объяснить назначение функции и параметров функции. Написать текст программы. Привести рисунок копии экрана для своего варианта.

**Таблица 3.1** Пример варианта задания

Номер варианта	Имя функции	Численное значение аргумента номер:						Тип заливки
		a1	a2	a3	a4	a5	a6	
N	line(a1,a2,a3,a4);	400	100	500	200			нет
	---- “ ----							
	bar(a1,a2,a3,a4);	200	400	270	450			Сплошная
	---- “ ----							

#### 3.3 Решение

Функция line вычерчивает отрезок прямой линии. Функция имеет четыре параметра. Все параметры имеют единицы измерения в пикселях. Первый и второй аргументы определяют координаты X и Y начальной точки отрезка линии, третий и четвертый аргументы определяют координаты X и Y конечной точки отрезка линии.

Функция bar вычерчивает закрашенный прямоугольник. Первый и второй аргументы определяют координаты X и Y начальной вершины прямоугольника, третий и четвертый аргументы определяют координаты X и Y конечной вершины прямоугольника (диагональной к ней). Шаблон и цвет закрашивания прямоугольника может быть определен с помощью функций setfillstyle.

#### 3.4 Текст программы

**Листинг 3.1** Текст программы выполнения задания.

```
/* Выполнил студент 1511 группы Петров С.Ю., Номер по списку 03.  
Дата 04.03.2004 */  
  
// файл g151103L.cpp  
  
#include <graphics.h>
```

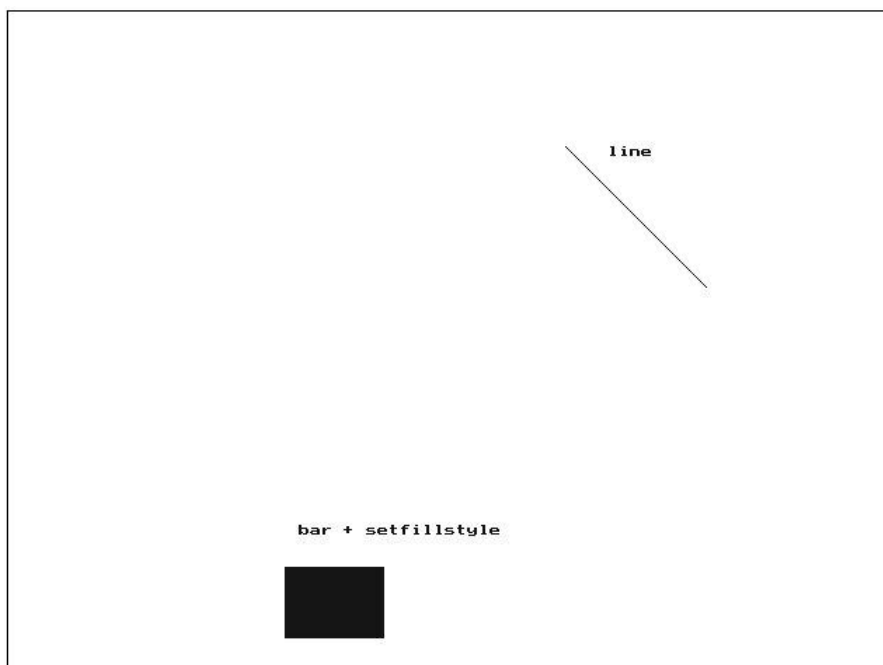
```

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void main(void)
{int gdriver = DETECT, gmode, errorcode;
int a1=400, a2=100, a3=500, a4=200;   int x1=200, y1=400,
x2=270, y2=450;
// 1
initgraph(&gdriver, &gmode, ""); errorcode = graphresult();
if (errorcode != grOk) {printf("Ошибка: %s\n",
grapherrormsg(errorcode));
printf("Нажмите любую клавишу"); getch(); exit(1); }
// 2
setcolor(YELLOW); outtextxy(a1+30,a2,"line"); line(a1,a2,a3,a4);
outtextxy(x1+10, y1-30, "bar + setfillstyle");
setfillstyle(SOLID_FILL, YELLOW); bar(x1, y1, x2, y2);
// 3
getch(); closegraph(); }

```

### 3.5 Копия экрана

Копия экрана приведена на рисунке 3.1.



**Рисунок 3.1.** Экранная копия работы программы из примера 3.1

### 3.6 Оформление результатов работы

Результаты работы оформляются в виде отчета по выдаваемому в электронном виде образцу. В отчет входят следующие разделы: задание, решение, текст программы, копия экрана, литература и интернет- ссылки. К отчету прилагается исходные и исполняемые файлы

программы. Нумерация файлов и папки осуществляется по личному коду, выдаваемому преподавателем.. Все материалы сдаются в электронном виде. Допускается распечатка отчетных материалов.

Например, имя папки с файлами: g151103L1, где g- студенческая группа, 1511- номер группы, 03- номер студента по списку, L- лабораторная работа, 1- порядковый номер работы. В папке находятся следующие файлы: g151103L1.doc- файл отчета по работе (образец выдается преподавателем), исходный файл g151103L.cpp, исполняемый файл g151103L.exe, egavga.bgi- файл графического драйвера.

### 3.7 Дополнительное задание

Кроме общетипового задания по согласованию с преподавателем выдается самостоятельное задание, в котором студент может показать свои знания и умения работы. Например, дополнить графику анимацией.

## 4. Лабораторная работа 2. Основные функции базовой графики Windows.

Данная лабораторная работа нужна студенту, как подготовительная работа, для приобретения навыков работы в среде программирования C++ Builder.

### 4.1 Цели работы

Познакомиться с конкретным примером выполнения программы в среде программирования C++ Builder.

Познакомиться с набором исходных файлов в среде программирования C++ Builder, системой обозначения файлов, компиляцией исполняемого файла, правилами работы в локальной компьютерной сети и сдаче файлов выполненных заданий.

Познакомиться с правилами оформления файлов отчетов по лабораторным и самостоятельным работам.

Выполнить задание по указанию преподавателя.

### 4.2 Задание

По номеру варианта нарисовать указанные графические Windows примитивы для контурных и площадных (с заливкой) фигур. Объяснить назначение функции и параметров функции. Объяснить отличие количества аргументов в графических функциях Windows API и в функциях Borland. Написать текст программы. Привести рисунок копии экрана для своего варианта.

**Таблица 4.1** Пример варианта задания

Номер вариан та	Имя функции	Численное значение аргумента						Тип заливки	Примеч.
		а1	а2	а3	а4	а5	а6		
N	MoveTo(a1,a2);	10	30					нет	
	LineTo(a1,a2);	100	200					нет	Ширина=5
	FillRect(Rect(a1,a2,a3,a4));	300	30	500	100			Сплошная	
	---- “ ----								

### 4.3 Решение

Функция MoveTo изменяет текущую позицию пера на заданную с координатами X и Y. Функция LineTo рисует отрезок прямой линии, начиная с текущей позиции пера, до указанных координат X и Y конечной точки отрезка. Функция FillRect вычерчивает закрашенный прямоугольник. Первый и второй аргументы определяют координаты X и Y начальной вершины прямоугольника, третий и четвертый аргументы определяют координаты X и Y конечной вершины прямоугольника (диагональной к ней). Шаблон и цвет закрашивания прямоугольника может быть определен с помощью свойств Canvas->Brush->Style (по умолчанию сплошная заливка) и Canvas->Brush->Color. Например, Canvas->Brush->Style= bsSolid; Canvas->Brush->Color = clRed;

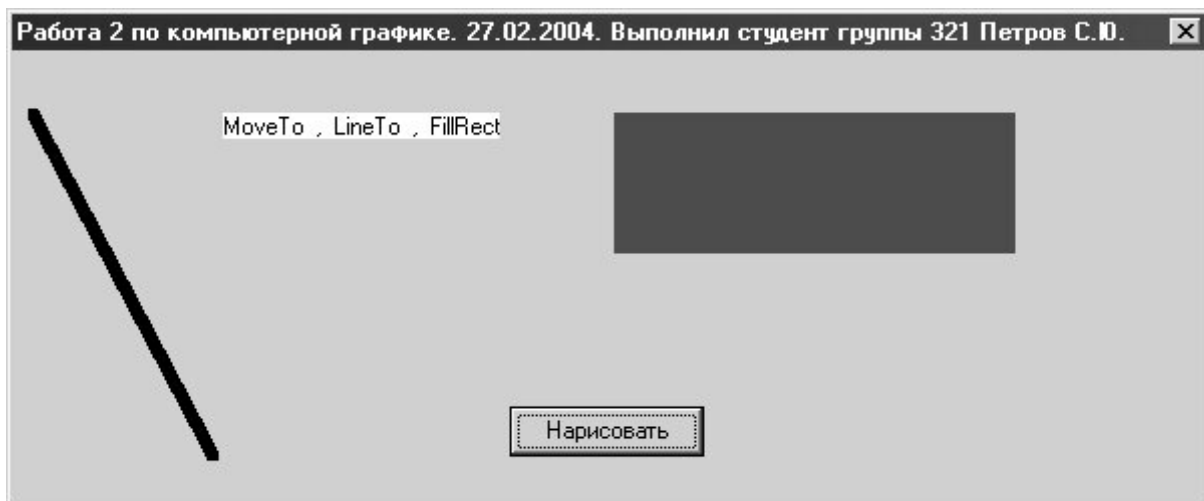
### 4.4 Текст программы

**Листинг 4.1** Текст программы выполнения задания.

```
// Выполнил студент 1511 группы Вариант 03 Петров С.Ю., 04.03.2004
// файл Ug151103L2.cpp
#include <vcl.h>
#pragma hdrstop
#include "Ug151103L2.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner){ }
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{ //Вручную вводится только пять строк, выделенных жирным шрифтом!
Canvas->TextOut(105, 30, "MoveTo , LineTo , FillRect");
Form1->Canvas->Pen->Width = 5;
Form1->Canvas->MoveTo(10,30); Form1->Canvas->LineTo(100,200);
Form1->Canvas->Brush->Color = clRed;
Form1->Canvas->FillRect(Rect(300, 30, 500, 100));
}
//-----
```

### 4.5 Копия экрана

Копия экрана приведена на рис. 4.1.



**Рисунок 4.1.** Экранная копия работы программы из примера 4.1

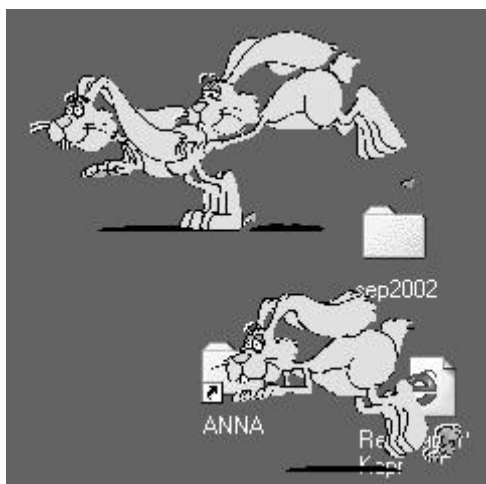
#### **4.6 Оформление результатов работы**

Результаты работы оформляются в виде отчета по выдаваемому в электронном виде образцу. В отчет входят следующие разделы: задание, решение, текст программы, копия экрана, литература и интернет- ссылки. К отчету прилагается исходные и исполняемые файлы программы. Нумерация файлов и папки осуществляется по личному коду, выдаваемому преподавателем. Все материалы сдаются в электронном виде. Допускается распечатка отчетных материалов.

Например, имя папки с файлами: g151103L2, где g- студенческая группа, 1511- номер группы, 03- номер студента по списку, L- лабораторная работа, 2- порядковый номер работы. В папке находятся следующие файлы: g151103L2.doc- файл отчета по работе (образец выдается преподавателем), исходные файлы (Unit) Ug151103L2.cpp, Ug151103L2.h, Ug151103L2.dfm, файл проекта (Project) g151103L2.bpr, g151103L2.cpp, g151103L2.res, исполняемый файл g151103L2.exe.

#### **4.7 Дополнительные задания**

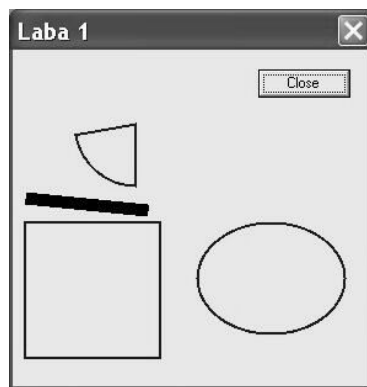
Кроме общетипового задания по согласованию с преподавателем выдается самостоятельное задание, в котором студент может показать свои знания и умения работы. Например, дополнить графику анимацией (по рабочему столу прыгают фигурные формы в виде зайцев).



**Рисунок 4.2-** Экранная копия работы программы

Возможно выполнение задания в другой С- язычной среде программирования. Например, в С# функция рисования будет выглядеть следующим образом:

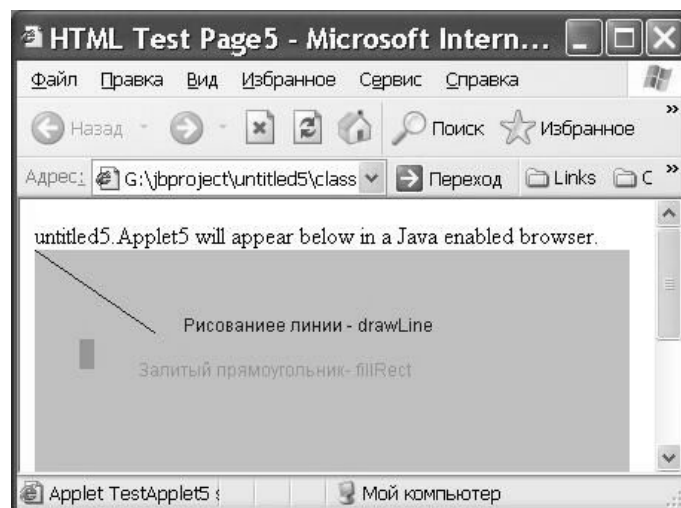
```
public void mainForm_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Pen bluePen = new Pen(Color.Blue, 2);
    Pen barPen = new Pen(Color.Black, 10);
    g.DrawLine(barPen, 10, 120, 110, 130);
    g.DrawRectangle(bluePen, 10, 140, 110, 110);
    g.DrawEllipse(bluePen, 150, 140, 120, 90);
    g.DrawPie(bluePen, 50, 10, 100, 100, 90, 80);
}
```



**Рисунок 4.3-** Экранная копия работы программы

Другим вариантом выполнения задания служит пример функции рисования в Java:

```
public void paint(Graphics g) {
    int xcenter, ycenter;      int x=30, y=60, width=10, height=20;
    xcenter=80;                ycenter=55;
    g.setColor(Color.blue);
    g.drawString("Рисованиее линии - drawLine",xcenter+20,ycenter);
    g.drawLine(xcenter, ycenter, 0, 0);
    g.setColor(Color.green);
    g.drawString("Залитый прямоугольник- fillRect",70,ycenter+30);
    g.fillRect( x, y, width, height);
}
```



**Рисунок 4.4-** Экранная копия работы Java-апплета

## **5. Самостоятельная работа 1. Построение каркасной модели трехмерного объекта**

Самостоятельные работы предназначены для знакомства студента с методами трехмерного геометрического моделирования: построения каркасной модели, построения поверхностной модели и знакомства с основами твердотельного моделирования.

Данная лабораторная работа нужна студенту для приобретения навыков работы в среде программирования C++ Builder.

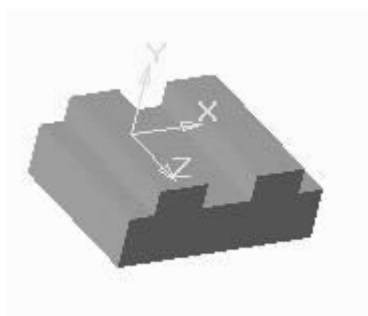
### **5.1 Цели работы**

Познакомиться с конкретным примером выполнения программы (листинги 5.1 и 5.2) и переписать программу в среде программирования C++ Builder.

Выполнить задание по указанию преподавателя, оформить файл отчета, сдать исходные и исполняемый файл.

### **5.2 Задание**

По заданию своего номера варианта создать трехмерную каркасную модель детали средствами графики C++ Builder. Написать текст программы. Предусмотреть параметризацию размеров детали. Привести рисунок копии экрана для своего варианта.

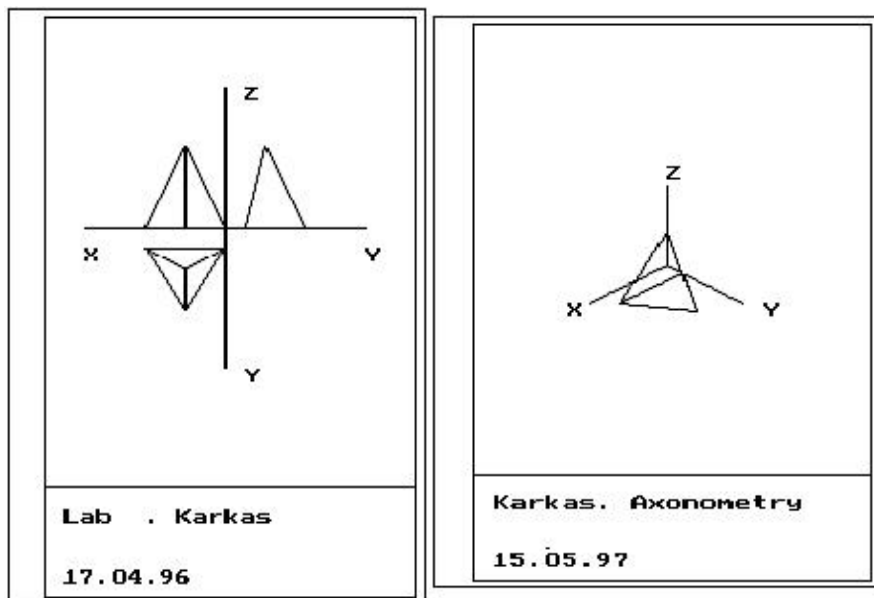


**Рисунок 5.1** Рисунок задания

### 5.3 Пояснение

Прежде чем начать выполнять задание обратимся к принципам проецирования на примере трехгранной пирамидки и разберем решение данной задачи для нее. Кроме того, чтобы студент не только подставил свои числа в готовую программу, а немного подумал о Windows программировании, приведем текст решения на языке C++ с использованием DOS функций (**line**, **moveto**, **lineto**).

Для ускорения работы над заданием желательно нарисовать эскизы ортогональной (рис 5.2) и аксонометрической (рис 5.3) проекций детали, а затем составить таблицы для координат вершин (таблица 5.1) и линий (таблица 5.2). Далее в тексте программы по указанному образцу вписываются свои значения координат из таблицы линий.



Рисунки 5.2 и 5.3 Ортогональные и аксонометрическая проекции каркасной модели.

Таблица 5.1 Значения координат вершин пирамиды

Номер вершины	X	Y	Z
1	40	10	0
2	0	10	0
3	20	40	0
4	20	20	40

Таблица 5.2 Значения координат начала и конца линии ребер каркасной модели

Номер линии (между вершинами)	Значения координат начала линии			Значения координат конца линии		
	Xn	Yn	Zn	Xk	Yk	Zk
0 (1-2)	40	10	0	0	10	0
1 (2-3)	0	10	0	20	40	0
2 (3-1)	20	40	0	40	10	0
3 (1-4)	40	10	0	20	20	40
4 (2-4)	0	10	0	20	20	40
5 (3-4)	20	40	0	20	20	40

**Листинг 5.1** Текст программы вычерчивания ортогональных проекции.

```
// Выполнил студент 321 группы Вариант 01 Петров С.Ю., 12.03.2004
// файл g32101S1.cpp

/*Задание: Написать программу на языке Си для вычерчивания
ортогональных проекций пирамиды.*/

#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

main(void){int i,Driver,Mode,errorcode;      int x0, y0;
int  xn[20]={40, 0,20,40, 0,20},
      yn[20]={10,10,40,10,10,40},
      zn[20]={ 0, 0, 0, 0, 0, 0},
      xk[20]={ 0,20,40,20,20,20},
      yk[20]={10,40,10,20,20,20},
      zk[20]={ 0, 0, 0,40,40,40};

/* Установка параметров */
x0=110.;y0=110.;

/* Инициализация графики */
Driver=DETECT; initgraph(&Driver,&Mode," ");
errorcode=graphresult( );      if(errorcode !=grOk)
{printf(" Ошибка графики: %s Код =%d", grapherrormsg(errorcode),
errorcode);      exit(1);}

/* рамка */ setbkcolor(0); rectangle(2,1,210,297);
rectangle(20,5,205,294); line(20,239,205,239);

outtextxy(30,250,"Tast S1. Karkas "); outtextxy(30,260,"Petrov");
outtextxy(30,270,"gr.321 number 01");outtextxy(30,280,"17.04.96");

/* Оси координат и их обозначение */
line(x0-70.,y0,x0+70,y0 ); line(x0,y0-70.,x0,y0+70);
outtextxy(x0-70.,y0+10.,"X"); outtextxy(x0+70.,y0+10.,"Y");
outtextxy(x0+10.,y0-70.,"Z"); outtextxy(x0+10.,y0+70.,"Y");

/* Основные циклы вычислений проекции XOY*/
for(i=0;i<11;i++){ line(x0-xn[i],y0+yn[i],x0-xk[i],y0+yk[i] );};

/* Основные циклы вычислений проекции XOZ*/
for(i=1;i<11;i++){ line(x0-xn[i],y0-zn[i],x0-xk[i],y0-zk[i] );};

/* Основные циклы вычислений проекции YOZ*/
for(i=1;i<11;i++){ line(x0+yn[i],y0-zn[i],x0+yk[i],y0-zk[i] );};
```

```

/* закрытие графического режима*/
while(!kbhit( )); closegraph( );return(0);}

```

**Листинг 5.2** Текст программы вычерчивания прямоугольной изометрической проекции.

```

// Выполнил студент 321 группы Вариант 01 Петров С.Ю., 12.03.2004
// файл g32101S1.cpp
/*Задание: По ортогональным проекциям каркасной модели написать
программу на языке Си для вычерчивания аксонометрической проекции
пирамиды.*/
#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
main(void){int i,Driver,Mode,errorcode;int sx0, sy0;
float kx,ky,kz,ax,ay,az,sxn,syn,sxk,syk; double sin30,cos30;
/* 1. Ввод значений исходных данных */
int n=6;
int xn[20]={40, 0,20,40, 0,20},
yn[20]={10,10,40,10,10,40},
zn[20]={ 0, 0, 0, 0, 0, 0},
xk[20]={ 0,20,40,20,20,20},
yk[20]={10,40,10,20,20,20},
zk[20]={ 0, 0, 0,40,40,40};
sx0=110;sy0=130;sin30=sin(30./180.*3.14);cos30=cos(30./180.*3.14);
kx=0.82;ky=0.82;kz=0.82;
/* 2. Инициализация графики */
Driver=DETECT; initgraph(&Driver,&Mode," ");
errorcode=graphresult(); if(errorcode !=grOk)
{printf("Ошибка графики:%s код=%d",
grapherrormsg(errorcode), errorcode); exit(1);}
/* вычерчивание рамки формата А4*/
rectangle(2,1,210,297); rectangle(20,5,205,294);
line(20,239,205,239);
outtextxy(30,250,"Tast s1. Karkas "); outtextxy(30,260,"Petrov");
outtextxy(30,270,"gr. 321 num 01"); outtextxy(30,280,"15.05.97");
/* 3. Организация основных циклов вычислений линии ребер каркасной
модели */

```

```

for(i=0;i<=n ;i++){
ax=xn[i];ay=yn[i];az=zn[i];
sxn=sx0-ax*kx*cos30+ ay*ky*cos30;syn=sy0- (az*kz-ax*kx*sin30-
ay*ky*sin30);
ax=xk[i];ay=yk[i];az=z[k][i];
sxx=sx0-ax*kx*cos30+ ay*ky*cos30;syk=sy0- (az*kz-ax*kx*sin30-
ay*ky*sin30);
line(sxn,syn,sxx,syk); }/*next i*/
/* оси координат и их обозначение */
ax=50;ay=0;az=0;
sxx=sx0-ax*kx*cos30+ ay*ky*cos30;syk=sy0- (az*kz-ax*kx*sin30-
ay*ky*sin30);
line(sx0,sy0,sxx,syk);outtextxy(sxx-10 ,syk,"X");
ax=0;ay=50;az=0;
sxx=sx0-ax*kx*cos30+ ay*ky*cos30;syk=sy0- (az*kz-ax*kx*sin30-
ay*ky*sin30);
line(sx0,sy0,sxx,syk);outtextxy(sxx+10,syk,"Y");
ax=0;ay=0;az=50;
sxx=sx0-ax*kx*cos30+ ay*ky*cos30;syk=sy0- (az*kz-ax*kx*sin30-
ay*ky*sin30);
line(sx0,sy0,sxx,syk);outtextxy(sxx,syk-10,"Z");
/* 4. Закрытие графического режима*/
while(!kbhit()); closegraph();
/* окончание программы */
return(0);}

```

## 5.4 Решение

При программировании под Windows функция MoveTo изменяет текущую позицию пера на заданную с координатами X и Y, а функция LineTo рисует отрезок прямой линии. Таким образом, можно заменить функцию line(X1,Y1,X2,Y2) в листингах 5.1 и 5.2 на две функции: Canvas->MoveTo(X1,Y1) и Canvas->LineTo(X2,Y2). Другим вариантом решения является создание своей функции. Для удобства работы сразу сделаем ее производной от класса TForm1, тогда функция вычерчивания линии примет вид:

```

void TForm1::line (int X1,int Y1,int X2,int Y2)
{Canvas->MoveTo (X1,Y1); Canvas->LineTo (X2,Y2);}

```

## 5.5 Текст программы

Подробного решения и текста программы под Windows не приводится, так как примеры создания графических программ разобраны в пункте 2.2 и в лабораторной работе номер 2. Приведем только значения координат для подстановки их в текст программы. Для указанной в задании детали (на рисунке 5.1) массивы значений координат для начала и конца линий приведены в таблице 5.3 (№ xn yn zn xk yk zk).



- Ug151103S1.cpp, Ug151103L2.h, Ug151103S1.dfm, файл проекта (Project)-g151103S1.bpr, g151103S1.cpp, g151103S1.res, исполняемый файл g151103S1.exe.

### 5.8 Дополнительное задание

Кроме общетипового задания по согласованию с преподавателем выдается самостоятельное задание, в котором студент может показать свои знания и умения работы. Например, дополнить программу редактором линий.

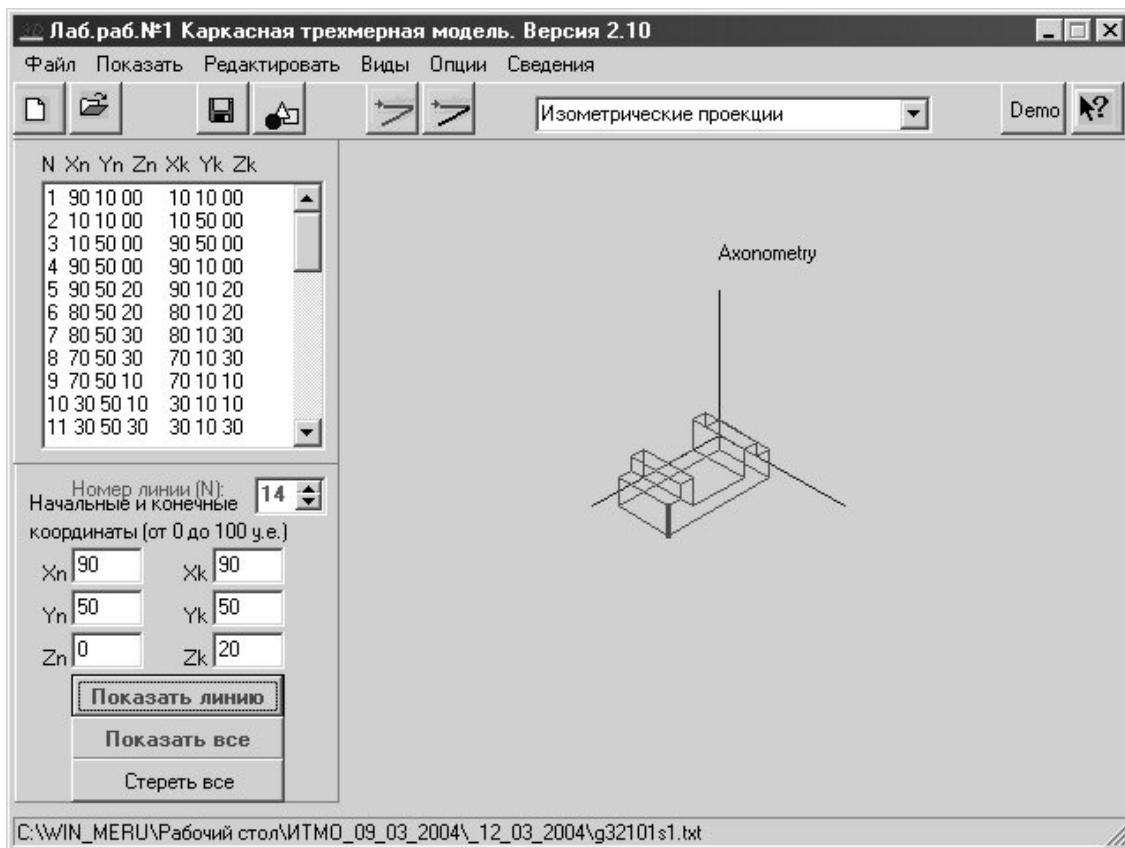


Рисунок 5.5- Экранная копия работы программы

## 6. Самостоятельная работа 2. Построение трехмерной модели с использованием библиотеки OpenGL

Данная лабораторная работа нужна студенту для приобретения навыков работы в среде программирования C++ Builder с использованием библиотеки OpenGL.

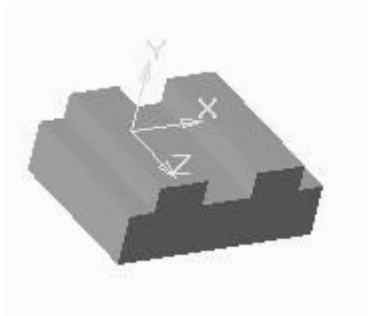
### 6.1 Цели работы

Познакомиться с конкретным примером выполнения программы в среде программирования C++ Builder с использованием библиотеки OpenGL.

Выполнить задание по указанию преподавателя, оформить файл отчета, сдать исходные и исполняемый файл.

### 6.2 Задание

По заданию своего номера варианта создать трехмерную каркасную модель детали средствами графики C++ Builder. Написать текст программы. Предусмотреть параметризацию размеров детали. Привести рисунок копии экрана для своего варианта.



**Рисунок 6.1** Рисунок задания

### 6.3 Пояснение

Прежде чем начать выполнять задание рассмотрим такие графические понятия, как вершины и примитивы библиотеки OpenGL.

Под вершиной понимается точка в трехмерном пространстве, координаты которой можно задавать следующим образом:

```
void glVertex[2 3 4][s i f d](type coords)
```

```
void glVertex[2 3 4][s i f d]v(type *coords)
```

Координаты точки задаются максимум четырьмя значениями:  $x$ ,  $y$ ,  $z$ ,  $w$ , при этом можно указывать два ( $x, y$ ) или три ( $x, y, z$ ) значения, а для остальных переменных в этих случаях используются значения по умолчанию:  $z=0$ ,  $w=1$ . Как уже было сказано выше, число в названии команды соответствует числу явно задаваемых значений, а последующий символ – их типу.

Координатные оси расположены так, что точка  $(0,0)$  находится в левом нижнем углу экрана, ось  $x$  направлена влево, ось  $y$  – вверх, а ось  $z$  – из экрана. Это расположение осей мировой системы координат, в которой задаются координаты вершин объекта, другие системы координат будут рассмотрены ниже.

Однако чтобы задать какую-нибудь фигуру одних координат вершин недостаточно, и эти вершины надо объединить в одно целое, определив необходимые свойства. Для этого в OpenGL используется понятие примитивов, к которым относятся точки, линии, связанные или замкнутые линии, треугольники и так далее. Задание примитива происходит внутри командных скобок:

```
void glBegin(GLenum mode)
```

```
void glEnd(void)
```

Параметр `mode` определяет тип примитива, который задается внутри и может принимать следующие значения:

**GL\_POINTS** каждая вершина задает координаты некоторой точки.

**GL\_LINES** каждая отдельная пара вершин определяет отрезок; если задано нечетное число вершин, то последняя вершина игнорируется.

**GL\_LINE\_STRIP** каждая следующая вершина задает отрезок вместе с предыдущей.

**GL\_LINE\_LOOP** отличие от предыдущего примитива только в том, что последний отрезок определяется последней и первой вершиной, образуя замкнутую ломаную.

**GL\_TRIANGLES** каждая отдельная тройка вершин определяет треугольник; если задано не кратное трем число вершин, то последние вершины игнорируются.

**GL\_TRIANGLE\_STRIP** каждая следующая вершина задает треугольник вместе с двумя предыдущими.

**GL\_TRIANGLE\_FAN** треугольники задаются первой и каждой следующей парой вершин (пары не пересекаются).

**GL\_QUADS** каждая отдельная четверка вершин определяет четырехугольник; если задано не кратное четырем число вершин, то последние вершины игнорируются.

**GL\_QUAD\_STRIP** четырехугольник с номером  $n$  определяется вершинами с номерами  $2n-1$ ,  $2n$ ,  $2n+2$ ,  $2n+1$ .

**GL\_POLYGON** последовательно задаются вершины выпуклого многоугольника.

Для задания текущего цвета вершины используются команды

```
void glColor[3 4][b s i f](GLtype components)
```

```
void glColor[3 4][b s i f]v(GLtype components)
```

Первые три параметра задают R, G, B компоненты цвета, а последний параметр определяет alpha-компоненту, которая задает уровень прозрачности объекта. Если в названии команды указан тип 'f' (float), то значения всех параметров должны принадлежать отрезку  $[0,1]$ , при этом по умолчанию значение alpha-компоненты устанавливается равным 1.0, что соответствует полной непрозрачности. Если указан тип 'ub' (unsigned byte), то значения должны лежать в отрезке  $[0,255]$ .

Разным вершинам можно назначать различные цвета и тогда будет проводиться линейная интерполяция цветов по поверхности примитива.

Для управления режимом интерполяции цветов используется команда `void glShadeModel(GLenum mode)` вызов которой с параметром **GL\_SMOOTH** включает интерполяцию (установка по умолчанию), а с **GL\_FLAT** отключает.

Например, чтобы нарисовать треугольник с разными цветами в вершинах, достаточно написать:

```
GLfloat BlueCol[3]={0,0,1};
glBegin(GL_TRIANGLE);
glColor3f(1.0, 0.0, 0.0); //красный
glVertex3f(0.0, 0.0, 0.0);
glColor3ub(0,255,0); //зеленый
glVertex3f(1.0, 0.0, 0.0);
glColor3fv(BlueCol); //синий
glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Для задания цвета фона используется команда `void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)`. Значения должны находиться в отрезке  $[0,1]$  и по умолчанию равны нулю. После этого вызов команды `void glClear(GLbitfield mask)` с параметром **GL\_COLOR\_BUFFER\_BIT** устанавливает цвет фона во все буфера, доступные для записи цвета (иногда удобно использовать несколько буферов цвета).

## 6.4 Решение

Для решения данной задачи достаточно использовать только один примитив – многоугольник (**GL\_POLYGON**). Особое внимание необходимо обратить на соблюдение обязательного условия выпуклости многоугольника. Для создания многоугольника необходимы трехмерные значения координат вершин, заданных в примитиве вершина (например, `glVertex3f`). При необходимости используется задание цвета многоугольника (например, `glColor3f(0.9,0.4,0.1)`).

**Таблица 6.1** Значения координат вершин граней детали

Номер грани	Координаты вершин для грани			
	Вершина	X	Y	Z
1	1	0.0	0.2	0.1
	2	0.0	0.6	0.1
	3	0.9	0.6	0.1
	4	0.9	0.2	0.1
...				
13bok_3	1	0.2	0.6	0.3
	2	0.2	0.6	0.5
	3	0.3	0.6	0.5
	4	0.3	0.6	0.3

### 6.5 Текст программы

Для упрощения работы над заданием студентам выдается электронный вариант выполненного задания для кубика (с координатами для восьми граней). От студента требуется почистить задание и вставить в функцию `OpenGLPanel1Paint` свои координаты.

**Листинг 6.1** Текст программы создания трехмерной модели.

```
/* Выполнил студент группы 1511 Петров С.Ю., вариант 03, дата
09.03.2004, файл Ug151103S2.cpp */
#include <vcl.h>
#pragma hdrstop
#include " Ug151103S2.h"
//-----
#pragma package(smart_init)
#pragma link "OpenGLPanel"
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{ // Повороты детали вокруг осей X и Y
  RotX=(float)TrackBar1->Position;
  RotY=(float)TrackBar2->Position;}
//-----
void __fastcall TForm1::OpenGLPanel1Init(TObject *Sender)
{ // Инициализация OpenGL графики в среде C++ Builder
  glViewport(0,0,(GLsizei)OpenGLPanel1->Width,
  (GLsizei)OpenGLPanel1->Height);
```

```

glMatrixMode(GL_PROJECTION); glLoadIdentity();
    if ( OpenGLPanel1->Height==0)
gluPerspective(45, (GLdouble)OpenGLPanel1->Width, 1.0, 2000.0);
    else
gluPerspective(45, (GLdouble)OpenGLPanel1->Width/
(GLdouble)OpenGLPanel1->Height,1.0, 2000.0);
glMatrixMode(GL_MODELVIEW); glLoadIdentity();
glEnable(GL_DEPTH_TEST); glClearColor(0.0,0.0,0.0,1.0);}
//-----
void __fastcall TForm1::OpenGLPanel1Resize(TObject *Sender)
{ /* Перерисовка изображения при изменении размеров окна
приложения */
    glViewport(0,0,(GLsizei)OpenGLPanel1->Width,
(GLsizei)OpenGLPanel1->Height);
    glMatrixMode(GL_PROJECTION); glLoadIdentity();
    if ( OpenGLPanel1->Height==0)
gluPerspective(45, (GLdouble)OpenGLPanel1->Width, 1.0, 2000.0);
    else
gluPerspective(45, (GLdouble)OpenGLPanel1->Width/
(GLdouble)OpenGLPanel1->Height,1.0, 2000.0);
    glMatrixMode(GL_MODELVIEW); glLoadIdentity();}
//-----
void __fastcall TForm1::OpenGLPanel1Paint(TObject *Sender)
{ /* Создание поверхностной модели детали из многоугольников
GL_POLYGON */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix(); glTranslated(0.0, 0.0, -5.0);
    glRotatef(RotX, 1.0, 0.0, 0.0); glRotatef(RotY, 0.0, 1.0, 0.0);
    //////////////////////////////////////-1-
    glBegin(GL_POLYGON); glColor3f(0.9,0.4,0.1);
glVertex3f( 0.0, 0.2, 0.1); glVertex3f( 0.0, 0.6, 0.1);
glVertex3f( 0.9, 0.6, 0.1); glVertex3f( 0.9, 0.2, 0.1);
    glEnd();
    //////////////////////////////////////-2-
    glBegin(GL_POLYGON); glColor3f(0.8,0.8,0.8);
glVertex3f( 0.9, 0.2, 0.1); glVertex3f( 0.9, 0.6, 0.1);
glVertex3f( 0.9, 0.6, 0.3); glVertex3f( 0.9, 0.2, 0.3);
    glEnd();

```

```

////////////////////////-3-
glBegin(GL_POLYGON); glColor3f(1.0,0.8,1.0);
glVertex3f( 0.9, 0.2, 0.3); glVertex3f( 0.9, 0.6, 0.3);
glVertex3f( 0.7, 0.6, 0.3); glVertex3f( 0.7, 0.2, 0.3);
    glEnd();
////////////////////////-4-
glBegin(GL_POLYGON); glColor3f(0.5,0.5,0.5);
glVertex3f( 0.7, 0.2, 0.5); glVertex3f( 0.7, 0.6, 0.5);
glVertex3f( 0.7, 0.6, 0.3); glVertex3f( 0.7, 0.2, 0.3);
    glEnd();
////////////////////////-5-
glBegin(GL_POLYGON); glColor3f(1.0,0.5,1.0);
glVertex3f( 0.7, 0.2, 0.5); glVertex3f( 0.7, 0.6, 0.5);
glVertex3f( 0.6, 0.6, 0.5); glVertex3f( 0.6, 0.2, 0.5);
    glEnd();
////////////////////////-6-
glBegin(GL_POLYGON); glColor3f(0.0,1.0,1.0);
glVertex3f( 0.6, 0.2, 0.5); glVertex3f( 0.6, 0.6, 0.5);
glVertex3f( 0.6, 0.6, 0.3); glVertex3f( 0.6, 0.2, 0.3);
    glEnd();
////////////////////////-7-
glBegin(GL_POLYGON); glColor3f(1.0,0.8,1.0);
glVertex3f( 0.6, 0.2, 0.3); glVertex3f( 0.6, 0.6, 0.3);
glVertex3f( 0.3, 0.6, 0.3); glVertex3f( 0.3, 0.2, 0.3);
    glEnd();
////////////////////////-8-
glBegin(GL_POLYGON); glColor3f(0.0,1.0,1.0);
glVertex3f( 0.3, 0.2, 0.3); glVertex3f( 0.3, 0.6, 0.3);
glVertex3f( 0.3, 0.6, 0.5); glVertex3f( 0.3, 0.2, 0.5);
    glEnd();
////////////////////////-9-
glBegin(GL_POLYGON); glColor3f(1.0,0.5,1.0);
glVertex3f( 0.2, 0.2, 0.5); glVertex3f( 0.2, 0.6, 0.5);
glVertex3f( 0.3, 0.6, 0.5); glVertex3f( 0.3, 0.2, 0.5);
    glEnd();
////////////////////////-10-
glBegin(GL_POLYGON); glColor3f(0.2,0.2,1.0);

```

```

glVertex3f( 0.2, 0.2, 0.5); glVertex3f( 0.2, 0.6, 0.5);
glVertex3f( 0.2, 0.6, 0.3); glVertex3f( 0.2, 0.2, 0.3);
glEnd();
////////////////////////////////////-11-
glBegin(GL_POLYGON); glColor3f(1.0,0.8,1.0);
glVertex3f( 0.0, 0.2, 0.3); glVertex3f( 0.0, 0.6, 0.3);
glVertex3f( 0.2, 0.6, 0.3); glVertex3f( 0.2, 0.2, 0.3);
glEnd();
////////////////////////////////////-12-
glBegin(GL_POLYGON); glColor3f(0.2,0.2,0.8);
glVertex3f( 0.0, 0.2, 0.3); glVertex3f( 0.0, 0.6, 0.3);
glVertex3f( 0.0, 0.6, 0.1); glVertex3f( 0.0, 0.2, 0.1);
glEnd();
////////////////////////////////////-12- bok_1
glBegin(GL_POLYGON); glColor3f(0.2,1.0,0.1);
glVertex3f( 0.6, 0.2, 0.1); glVertex3f( 0.9, 0.2, 0.1);
glVertex3f( 0.9, 0.2, 0.3); glVertex3f( 0.7, 0.2, 0.3);
glVertex3f( 0.7, 0.2, 0.5); glVertex3f( 0.6, 0.2, 0.5);
glEnd();
////////////////////////////////////-12- bok_1
glBegin(GL_POLYGON); glColor3f(0.2,1.0,0.1);
glVertex3f( 0.0, 0.2, 0.1); glVertex3f( 0.9, 0.2, 0.1);
glVertex3f( 0.9, 0.2, 0.3); glVertex3f( 0.0, 0.2, 0.3);
glEnd();
////////////////////////////////////-12- bok_2
glBegin(GL_POLYGON); glColor3f(0.2,1.0,0.1);
glVertex3f( 0.6, 0.2, 0.3); glVertex3f( 0.6, 0.2, 0.5);
glVertex3f( 0.7, 0.2, 0.5); glVertex3f( 0.7, 0.2, 0.3);
glEnd();
////////////////////////////////////-12- bok_3
glBegin(GL_POLYGON); glColor3f(0.2,1.0,0.1);
glVertex3f( 0.2, 0.2, 0.3); glVertex3f( 0.2, 0.2, 0.5);
glVertex3f( 0.3, 0.2, 0.5); glVertex3f( 0.3, 0.2, 0.3);
glEnd();
////////////////////////////////////-13- bok_1
glBegin(GL_POLYGON); glColor3f(1.0,1.0,0.0);
glVertex3f( 0.0, 0.6, 0.1); glVertex3f( 0.9, 0.6, 0.1);

```

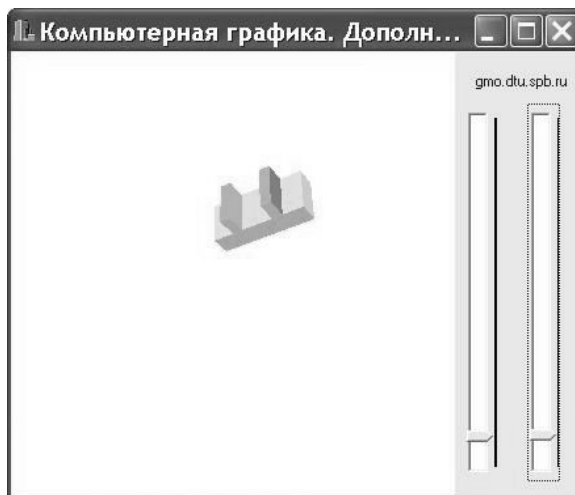
```

glVertex3f( 0.9, 0.6, 0.3); glVertex3f( 0.0, 0.6, 0.3);
glEnd();
////////////////////////////////////-13- bok_2
glBegin(GL_POLYGON); glColor3f(1.0,1.0,0.0);
glVertex3f( 0.6, 0.6, 0.3); glVertex3f( 0.6, 0.6, 0.5);
glVertex3f( 0.7, 0.6, 0.5); glVertex3f( 0.7, 0.6, 0.3);
glEnd();
////////////////////////////////////-13- bok_3
glBegin(GL_POLYGON); glColor3f(1.0,1.0,0.0);
glVertex3f( 0.2, 0.6, 0.3); glVertex3f( 0.2, 0.6, 0.5);
glVertex3f( 0.3, 0.6, 0.5); glVertex3f( 0.3, 0.6, 0.3);
glEnd();
glPopMatrix();}
//-----
void __fastcall TForm1::TrackBar1Change(TObject *Sender)
{ // Перерисовка детали после поворота вокруг оси X
RotX=(float)TrackBar1->Position;
OpenGLPanel1->Repaint();}
//-----
void __fastcall TForm1::TrackBar2Change(TObject *Sender)
{// Перерисовка детали после поворота вокруг оси Y
RotY=(float)TrackBar2->Position;
OpenGLPanel1->Repaint();}
//-----

```

## 6.6 Копия экрана

После подстановки своих массивов координат в программу получаем решение.



**Рисунок 6.2** - Экранная копия работы программы

## 6.7 Оформление результатов работы

Результаты работы оформляются в виде отчета по выдаваемому в электронном виде образцу. В отчет входят следующие разделы: задание, решение, текст программы, копия экрана, литература и интернет- ссылки. К отчету прилагается исходные и исполняемые файлы программы. Нумерация файлов и папки осуществляется по личному коду, выдаваемому преподавателем. Все материалы сдаются в электронном виде. Допускается распечатка отчетных материалов.

Например, имя папки с файлами: g151103S2, где g- студенческая группа, 1511- номер группы, 03- номер студента по списку, S- самостоятельная работа, 2- порядковый номер работы. В папке находятся следующие файлы: g151103L2.doc- файл отчета по работе (образец выдается преподавателем), исходные файлы (Unit) - Ug151103S2.cpp, Ug151103S2.h, Ug151103S2.dfm, файл проекта (Project)- g151103S2.bpr, g151103S2.cpp, g151103S2.res, исполняемый файл g151103S2.exe.

## 6.8 Дополнительное задание

Кроме общетипового задания по согласованию с преподавателем выдается самостоятельное задание, в котором студент может показать свои знания и умения работы.

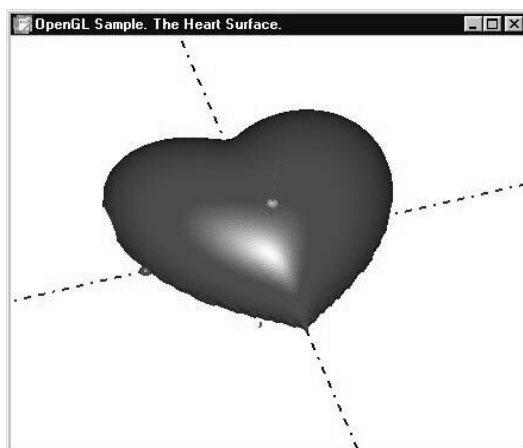


Рисунок6.3- Экранная копия работы программы

# 7. Самостоятельная работа 3. Построение трехмерной модели средствами VRML

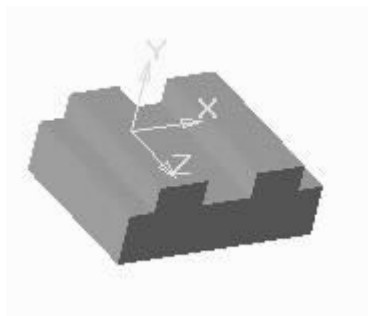
Данная лабораторная работа нужна студенту для приобретения навыков работы с трехмерной графикой.

## 7.1 Цели работы

- Познакомиться с конкретным примером выполнения программы средствами VRML.
- Выполнить задание по указанию преподавателя, оформить файл отчета, сдать исходные и исполняемый файл.

## 7.2 Задание

По заданию своего номера варианта создать параметрическую трехмерную модель детали средствами языка VRML. Написать текст программы. Привести рисунок копии экрана для своего варианта.



**Рисунок 7.1** Рисунок задания

### 7.3 Пояснение

Прежде чем начать выполнять задание рассмотрим основные понятия языка программирования VRML.

Язык VRML (Virtual Realty Modelling Language) предназначен для описания трехмерных изображений и оперирует объектами, описывающими геометрические фигуры и их расположение в пространстве. Vmrl-файл представляет собой обычный текстовый файл, интерпретируемый браузером. Поскольку большинство браузеров не имеет встроенных средств поддержки vmrl, для просмотра Vmrl-документов необходимо подключить вспомогательную программу - Vmrl-браузер, например, Live3D или Cosmo Player. Как и в случае с HTML, один и тот же vmrl-документ может выглядеть по-разному в разных VRML-браузерах. Кроме того, многие разработчики VRML-браузеров добавляют нестандартные расширения VRML в свой браузер.

Существует немало VRML-редакторов, делающих удобней и быстрее процесс создания VRML-документов, однако несложные модели можно создать при помощи самого простого текстового редактора.

В VRML приняты следующие единицы измерения: расстояние и размер: метры, углы: радианы, остальные значения: выражаются, как часть от 1. Координаты берутся в трехмерной декартовой системе координат.

Для того, чтобы VRML-браузер распознал файл с VRML-кодом, в начале файла ставится специальный заголовок - file header:

```
#VRML V1.0 ascii
```

Такой заголовок обязательно должен находиться в первой строке файла, кроме того, перед знаком диеза не должно быть пробелов.

В VRML определены четыре базовые примитивные фигуры: куб (верней не куб, а прямоугольный параллелепипед), сфера, цилиндр и конус. Эти фигуры называются примитивами (primitives). Набор примитивов невелик, однако комбинируя их, можно строить достаточно сложные трехмерные изображения. Рассмотрим поподробней каждый из примитивов.

**Куб.** Возможные параметры: width - ширина, height - высота, depth - глубина.

```
Cube {
    width    2    # ширина
    height   3    # высота
    depth    1    # глубина
}
```

**Сфер.** Параметр у сферы только один, это radius.

```
Sphere {
    radius   1    # радиус
```

```
}
```

**Конус.** Возможные параметры: `bottomRadius` - радиус основания, `height` - высота, `parts` - определяет, какие части конуса будут видны. Параметр `parts` может принимать значения `ALL`, `SIDES` или `BOTTOM`.

```
Cone {
    parts          ALL    #видны и основание, и боковая поверхность
конуса
    bottomRadius  1      #радиус основания
    height        2      #высота
}
```

**Цилиндр.** Для цилиндра можно задать параметры `radius` и `height`. Кроме того, с помощью параметра `parts` для цилиндра можно определить будут ли отображаться основания цилиндра и его боковая поверхность. Параметр `parts` может принимать значения `ALL`, `SIDES`, `BOTTOM` или `TOP`.

```
Cylinder {
    parts  ALL    #видны все части цилиндра
    radius 1      #радиус основания
    height 2      #высота цилиндра
}
```

Кроме описанных выше примитивных фигур формы строятся с помощью точек, линий и граней. Использование точек (`Points`), линий (`Lines`) и граней (`Faces`) позволяет создавать более сложные формы, чем примитивы и генерировать более реальные VRML миры. Формы, созданные с помощью точек, линий и граней имеют больше функциональных возможностей, чем примитивы.

Рассматриваемый принцип построения форм характеризуется тем, что координаты точек и связь между ними задаются отдельно. Данный метод описания форм характеризуется большой гибкостью. Описание геометрии форм происходит в два этапа:

1.Описание координат точек

2.Соединение точек

Координаты точек описываются с помощью узла `Coordinate`:

```
Coordinate {
    point [ 1.0 2.0 3.0 ,
            4.0 1.5 5.3 ,
            . . . ]
}
```

Три типа узлов описывают геометрию форм, основанную на точках и связях между ними:

```
PointSet
IndexedLineSet
IndexedFaceSet
```

Все они имеют поле `coord`, значением которого является узел `Coordinate`. Рассмотрим эти типы узлов более подробно.

**Узел `PointSet`.** Этот узел определяет массив точек в трехмерном пространстве в локальной системе координат :

```
PointSet {
    coord Coordinate { point [ . . . ] }
    color . . .
}
```

Каждой точке может быть присвоен свой цвет . Он задается в поле `color`, которое мы рассмотрим позже. Если поле `color` не определено, то используется цвет, заданный в поле `emissiveColor` узла `Material`. Размер точек постоянный и его нельзя изменять.

**Узел `IndexedLineSet`.** Этот узел определяет множество ломанных линий в трехмерном пространстве, соединяющих точки, описанные в поле `coord` :

```
IndexedLineSet {
  coord Coordinate { point [ . . . ] }
  coordIndex [ 1 , 0 , 2 , -1 , . . . ]
  . . .
}
```

Поле `coordIndex` описывает соединения между точками, описанными в поле `coord` . Числа в поле `coordIndex` являются индексами точек в поле `coord` :

```
coordIndex [ 1 , 0 , 3 , -1 , . . . ]
```

При этом справедливы следующие условия:

порядок точек произвольный; точки индексируются, начиная с нуля; ломанная линия может содержать несколько точек; конец ломанной определяется числом `-1`; можно описывать множество ломанных.

**Узел `IndexedFaceSet`.** Этот узел определяет форму в трехмерном пространстве, созданную из граней, полученных путем соединения по периметру грани точек, описанных в поле `coord`.

```
IndexedFaceSet {
  coord Coordinate { point [ . . . ] }
  coordIndex [ 1 , 0 , 2 , -1 , . . . ]
  . . .
}
```

Описание форм с помощью точек и связей между ними позволяет создавать сложные формы. Узлы `PointSet`, `IndexedLineSet` и `IndexedFaceSet` описывают геометрию форм с помощью точек.

## 7.4 Решение

Для решения данной задачи достаточно использовать только два примитива: массив точек и массив граней. Особое внимание необходимо обратить на соблюдение обязательного условия выпуклости многоугольника. Для создания многоугольника необходимы трехмерные значения массива координат вершин, заданных в примитиве вершина ( `point [ ]` ) и массива индексов точек, из которых состоят грани( `coordIndex [ ]` ).

**Таблица 7.1** Значения координат вершин детали

Номер вершины	X	Y	Z
0	0.0	0.0	0.0
1	0.0	20.0	10.0
2	90.0	20.0	10.0
3	90.0	20.0	30.0
4	70.0	20.0	30.0
...			
24	0.0	60.0	30.0

**Таблица 7.2 Индексы граней**

Номер вершины, входящий в грань				
Номер грани	№1	№2	№3	№4
1	1	2	14	13
2	2	3	15	14
3	3	4	16	15
4	4	5	17	16
5	5	6	18	17
...				
18	20	21	22	23

## 7.5 Текст программы

Для упрощения работы над заданием студентам выдается электронный вариант выполненного задания для кубика (с координатами для восьми граней). От студента требуется почистить задание и вставить свои координаты.

**Листинг 7.1** Текст программы создания трехмерной модели.

```
#VRML V2.0 utf8 #####
# Выполнил студент 1511 группы Петров С.Ю., , вариант 03
# дата 09.03.2004, файл g151103S3.wrl
Group {      children [
    Shape {
        appearance Appearance {
            material DEF _DefMat Material {
            }
        }
        geometry IndexedFaceSet {
            coord Coordinate {
                point [
# Массив точек #####
0.0  0.0 0.0,
    0.0  20.0 10.0,
    90.0  20.0 10.0,
    90.0  20.0 30.0,
    70.0  20.0 30.0,
    70.0  20.0 50.0,
    60.0  20.0 50.0,
    60.0  20.0 30.0,
    30.0  20.0 30.0,
```

```

30.0  20.0  50.0,
20.0  20.0  50.0,
20.0  20.0  30.0,
0.0   20.0  30.0,
#####
0.0   60.0  10.0,
90.0  60.0  10.0,
90.0  60.0  30.0,
70.0  60.0  30.0,
70.0  60.0  50.0,
60.0  60.0  50.0,
60.0  60.0  30.0,
30.0  60.0  30.0,
30.0  60.0  50.0,
20.0  60.0  50.0,
20.0  60.0  30.0,
0.0   60.0  30.0
#####
]

    }
    solid FALSE
    creaseAngle      0.5
    coordIndex [
# Массив граней #####
1, 2, 14, 13, -1,
2, 3, 15, 14, -1,
3, 4, 16, 15, -1,
4, 5, 17, 16, -1,
5, 6, 18, 17, -1,
6, 7, 19, 18, -1,
7, 8, 20, 19, -1,
8, 9, 21, 20, -1,
9,10, 22, 21, -1,
10,11, 23, 22, -1,
11,12, 24, 23, -1,
12, 1, 13, 24, -1,
#####

```

```

1, 2, 3, 12, -1,
4, 5, 6, 7, -1,
8, 9, 10, 11, -1,
13,14, 15, 24, -1,
16,17, 18, 19, -1,
20,21, 22, 23, -1
#####
]
    }
}
]
}

```

## 7.6 Копия экрана

После подстановки своих массивов координат в программу получаем решение.

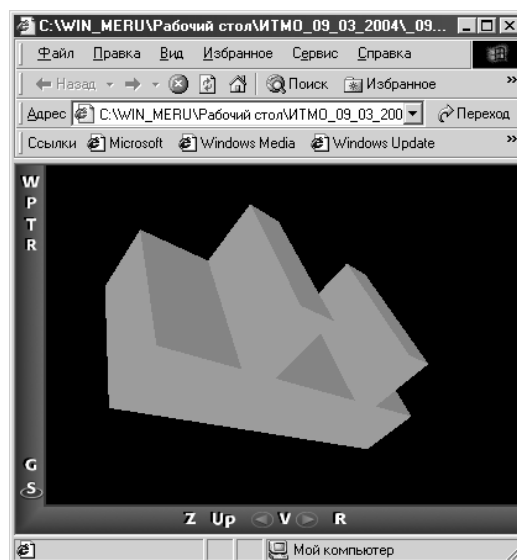


Рисунок 7.2 - Экранная копия работы программы

## 7.7 Оформление результатов работы

Результаты работы оформляются в виде отчета по выдаваемому в электронном виде образцу. В отчет входят следующие разделы: задание, решение, текст программы, копия экрана, литература и интернет- ссылки. К отчету прилагается исходные и исполняемые файлы программы. Нумерация файлов и папки осуществляется по личному коду выдаваемому преподавателем. Все материалы сдаются в электронном виде. Допускается распечатка отчетных материалов.

Например, имя папки с файлами: g151103S3, где g- студенческая группа, 1511- номер группы, 03- номер студента по списку, S- самостоятельная работа, 3- порядковый номер работы. В папке находятся следующие файлы: g151103L3.doc- файл отчета по работе (образец выдается преподавателем), исходный файл - Ug151103S3.wrl.

## 8. Самостоятельная работа 4. Построение трехмерной модели средствами графического редактора «Компас 3D»

Данная лабораторная работа нужна студенту для приобретения навыков работы с трехмерной графикой.

### 8.1 Цели работы

- Познакомиться с конкретным примером построения трехмерной модели детали по указанным размерам в графическом редакторе «Компас 3D».
- Выполнить задание по указанию преподавателя, оформить файл отчета, сдать исходные и исполняемый файл.

### 8.2 Задание

По заданию своего номера варианта создать трехмерную модель детали средствами графического редактора «Компас». Привести рисунок копии экрана для своего варианта. Привести рисунок копии экрана для своего варианта.

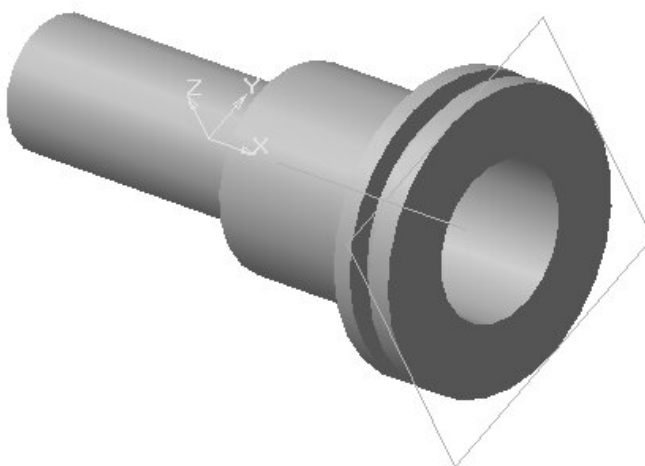


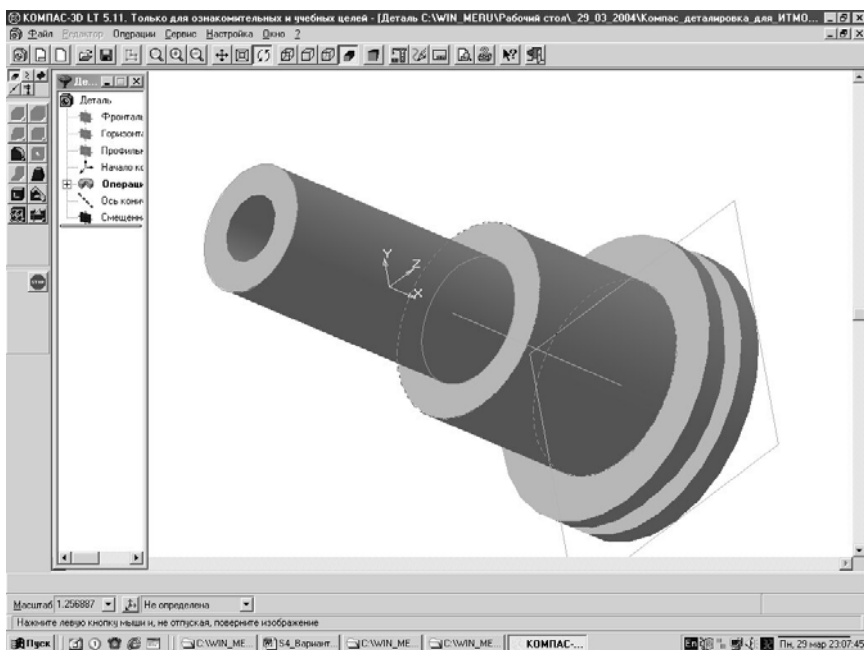
Рисунок 8.1 Рисунок задания

### 8.3 Пояснение

Для решения данной задачи:

1. выполняется необходимое количество проекций детали.
2. выполняются необходимые разрезы и сечения.
3. осуществляется простановка размеров.





**Рисунок 8.4** Копия экрана трехмерной модели детали

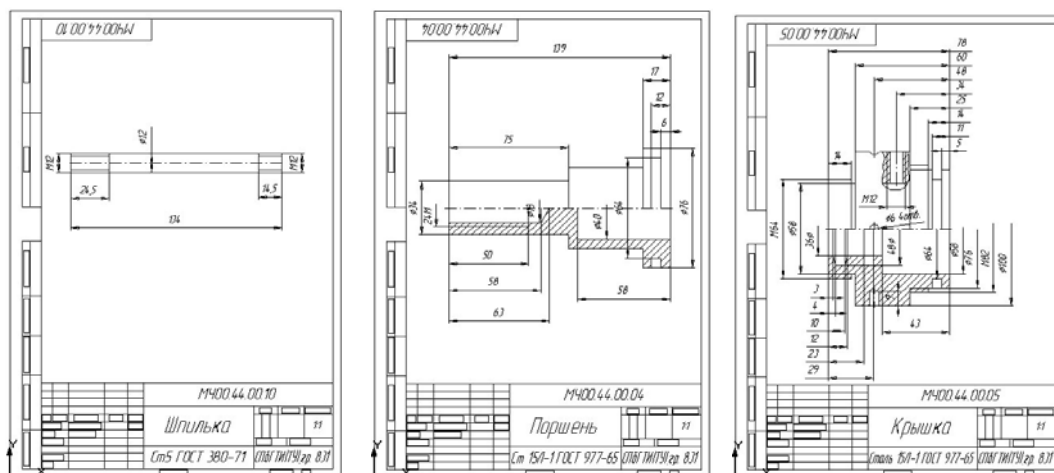
### 8.5 Оформление результатов работы

Результаты работы оформляются в виде отчета по выдаваемому в электронном виде образцу. В отчет входят следующие разделы: задание, решение, копия экрана, литература и интернет-ссылки. К отчету прилагается исходные. Нумерация файлов и папки осуществляется по личному коду, выдаваемому преподавателем. Все материалы сдаются в электронном виде. Допускается распечатка отчетных материалов.

Например, имя папки с файлами: g151103S4, где g- студенческая группа, 1511- номер группы, 03- номер студента по списку, S- самостоятельная работа, 4- порядковый номер работы. В папке находятся следующие файлы: g151103L4.doc- файл отчета по работе (образец выдается преподавателем), файлы - Ug151103S4.cdw, Ug151103S4.m3d.

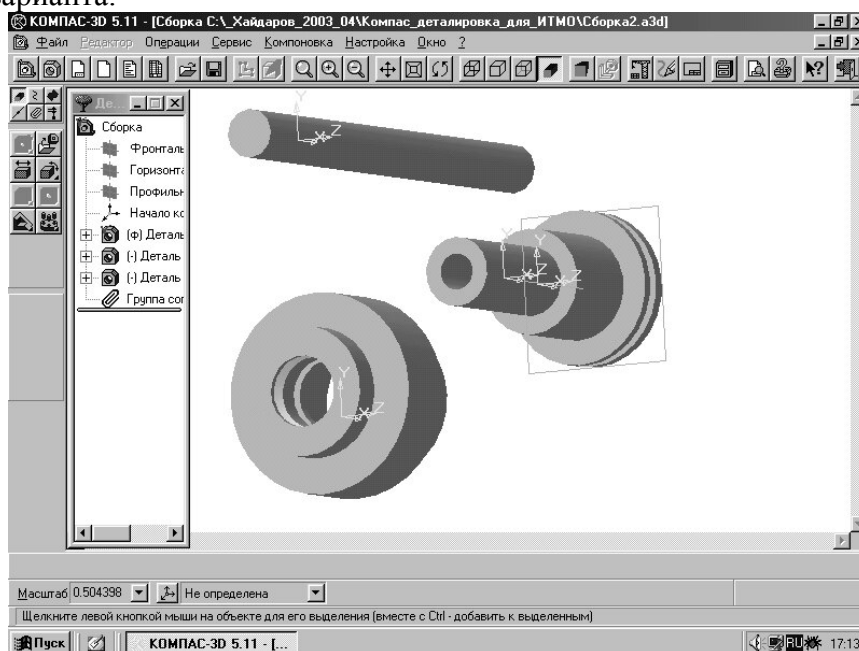
### 8.6 Дополнительное задание

Кроме общетипового задания по согласованию с преподавателем выдается самостоятельное задание, в котором студент может показать свои знания и умения работы. Заданы три детали – тела вращения. Необходимо создать трехмерную модель сборки деталей средствами графического редактора «Компас».



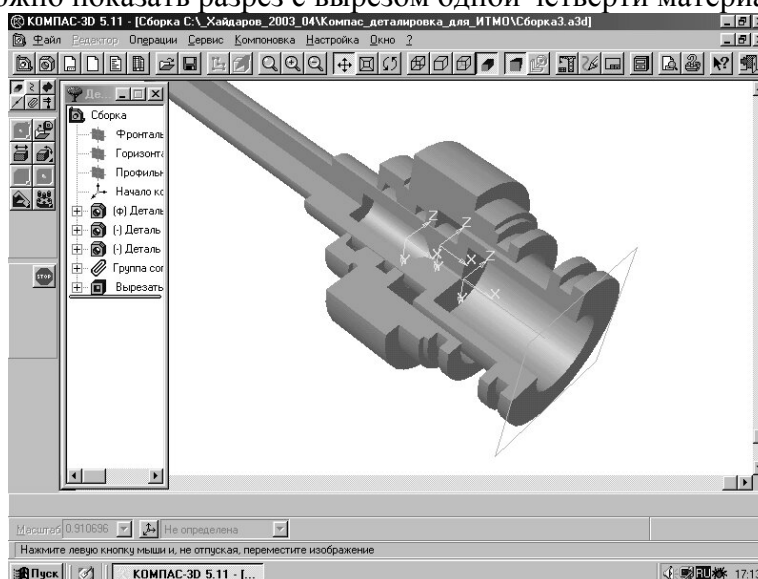
**Рисунки 8.5, 8.6, 8.7-** Три чертежа деталей для сборки

Для каждой детали выполняется операция прорисовки контура вращения вокруг осевой линии. Выполняются операция вращения вокруг осевой линии и создаются трехмерные модели деталей. Далее следует загрузить три детали в файл сборки: g151103s4A.a3d, где 03- номер варианта.



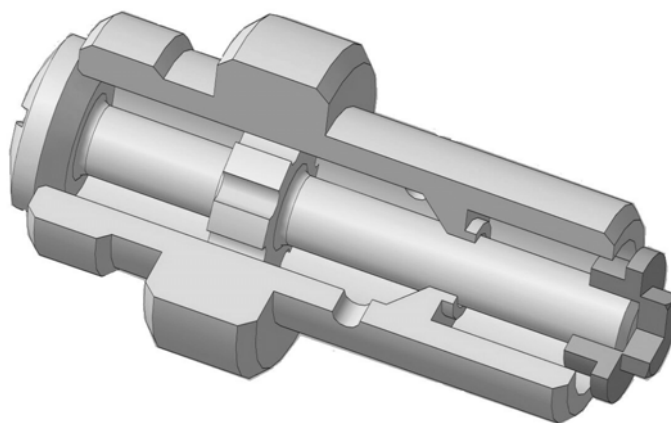
**Рисунок 8.8-** Три детали для сборки

Произвести центровку деталей по одной общей оси в файле сборки g321NNs3a.a3d. Для наглядности можно показать разрез с вырезом одной четверти материала деталей сборки.



**Рисунок 8.9-** Сборка трех деталей

Рассмотрим еще один пример сборки трех деталей. В данном примере кроме операции вращения тела имеют технологические прорезы и отверстия.



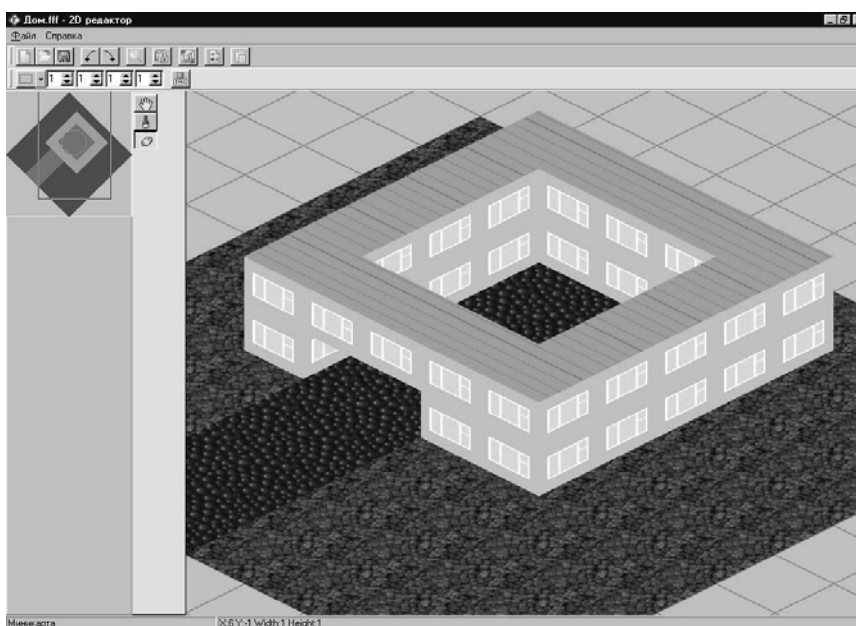
**Рисунок 8.10-** Сборка трех деталей

## 9. Примеры применение методов компьютерной графики

Комплексное использование методов компьютерной графики и их модификаций в современном программировании позволяет получить интересные результаты. Рассмотрим ряд программ выполненных с применением данных методов.

### 9.1 Двухмерный конструктор

В данной программе наряду с использованием «быстрой» графики (оконный режим DirectDraw) для визуализации текстур использовались модифицированные алгоритмы рисования линии и заливки. Данная программа позволяет создавать аксонометрическое изображение домов, парков, строительных участков и других объектов. Данные изображения собираются, как бы из кубиков. Можно строить объекты разной этажности. Текстуры для кубиков можно рисовать и дополнять библиотеку текстур самим. Единственным ограничением данного конструктора является одинаковость размера кубиков, как блоков строительства.



**Рисунок 9.1-** Экранная копия работы программы

## 9.2 Редактор трехмерной полигональной модели

В данной программе наряду с использованием «быстрой» графики (полноэкранный режим DirectDraw) использовались модификации всех перечисленных ранее алгоритмов компьютерной графики (линии, заливки, отсечения, плавающего горизонта, Z буфера), а также алгоритм освещения по методу Фонга. Кроме того, в данной программе вручную (manual) были применены матрицы преобразования трехмерных объектов, редактирования вершин трехмерной полигональной модели, определены нормали к граням. На примере данной работе показано, как с нуля создать трехмерный редактор. Исходный код насчитывает около 5000 строк. Для первоначальной загрузки использовался формат файлов 3DS, в дальнейшем для редактирования и применения трехмерной модели использовался более простой формат, достоинством которого была высокая скорость загрузки в следующей программе.

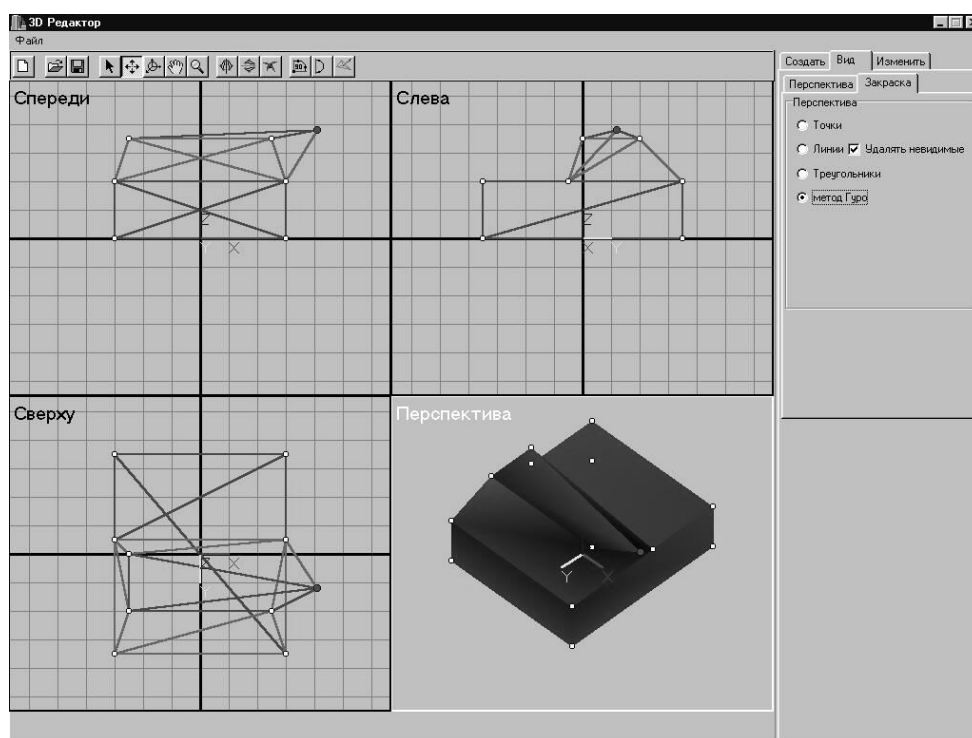


Рисунок 9.2- Экранная копия работы программы

## 9.3 Движок для игровой программы

Используя две предыдущие программы, был создан движок для игровой программы (полноэкранный режим DirectDraw). В данной программе создавался трехмерный город из «кубиков» с текстурами. В отличие от первой программы «кубики» трехмерные и рассматривать их можно с пяти сторон (кроме вида снизу). Использование второй программы - редактора трехмерной полигональной модели позволяло создать движущиеся объекты (машинки). Главным достоинством данной программы является ее быстродействие до 70 кадров в секунду, что для трехмерных игр проблематично. Такое высокое быстродействие достигается за счет программирования алгоритмов компьютерной графики «с нуля» в DirectDraw, применения своего формата для визуализации движущихся объектов, исключения из рассмотрения в матрице трехмерного преобразования возможности взгляда снизу на сцену игры. Сравнение скорости работы программы (путем установки счетчика кадров) с аналогичными игровыми программами показывает, что изучение алгоритмов компьютерной графики и их применение по сравнению с использованием готовых универсальных библиотек (например, OpenGL) позволяет увеличить скорость работы конкретной программы в 2 – 2,5 раза. Данное увеличение скорости работы достигается только за счет вдумчивой модификации общих алгоритмов и исходных кодов трехмерной

графики к конкретной трехмерной модели. При этом все исходные коды алгоритмов компьютерной графики надо писать самому. Например, исходный код данной программы насчитывает больше тысячи строк.



Рисунок 9.3- Экранная копия работы программы

## 10. Контрольные вопросы

1. Для чего применяется функция `initgraph` и какие она имеет параметры?
2. Для чего применяется файл `egavga.bgi`?
3. Для чего применяется функция `setfillstyle` и какие она имеет параметры?
4. Для чего применяется метод `Canvas` и какие графические функции он содержит?
5. Для чего применяется функция `FloodFill` и какие она имеет параметры?
6. Какие вы знаете способы преобразования трехмерной координаты точки к двумерным экранным координатам?
7. Какие ортогональные проекции вы знаете? Напишите уравнения и объясните геометрический смысл слагаемых для преобразования трехмерной модели в ортогональную проекцию.
8. Какие аксонометрические проекции вы знаете? Напишите уравнения и объясните геометрический смысл слагаемых для преобразования трехмерной модели в аксонометрическую проекцию.
9. Какая структура функций (команд) в OpenGL? Объясните ее на конкретном примере функции `void glVertex[2 3 4][s i f d](type coords)`.
10. Для чего применяются в OpenGL следующая группа функций: `glBegin`, `glColor3f`, `glVertex3f`, `glEnd`, `glTranslated`, `glRotatef` и какие они имеют параметры? Дополните список функций этой группы.
11. Для чего применяются в OpenGL следующая группа функций: `glViewport`, `glMatrixMode`, `glLoadIdentity`, `gluPerspective` и какие они имеют параметры? Дополните список функций этой группы.
12. Какова последовательность создания трехмерной детали вращения в «Компас 3D»?

# Литература

## Основная литература

1. Л. Аммерал. Принципы программирования в машинной графике: Пер. с англ. –М.: «Сол-Систем», 1992. 224с.
2. Шикин А.В., Боресков А.В. Компьютерная графика. Динамика, реалистические изображения. – М.: ДИАЛОГ-МИФИ, 1996. – 288 с.
3. Шикин А.В., Боресков А.В., Компьютерная графика. Полигональные модели. –М.: ДИАЛОГ-МИФИ, 2001. -464с.
4. Порев В.Н. Компьютерная графика.- СПб.: БХВ- Петербург, 2002. -432с.

## Дополнительная литература

5. Касаткин А.И., Вальвачев А.Н. Профессиональное программирование на языке Си: От TurboC к Borland C++: Справ. пособие; Под общ. ред. А.И. Касаткина.- Мн.: Выш. шк., 1992. 240с.
6. Березин Б.И., Березин С.Б. Начальный курс C и C++. –М.: ДИАЛОГ-МИФИ, 2001. - 288с.
7. Белецкий Я. Энциклопедия языка Си: Пер. с польск. –М.: Мир, 1992. -687с.
8. Ла Мот А., Ратклифф Д., Семинаторе М., Тайлер Д. Секреты программирования игр: Пер. с англ. –СПб.: Питер, 1995. -720с.
9. Кальвет Чарльз и др. Borland C++Builder3. Энциклопедия пользователя: Пер. с англ. – К.: Издательство «ДиаСофт», 1998. -804с.
10. Кент Рейсдорф, Кен Хендерсон. Borland C++Builder. Освой самостоятельно. Пер. с англ. –М.: «Издательство БИНОМ», 1998. -704с.
11. М. Теллес. Borland C++Builder: библиотека программиста: Пер. с англ. –СПб.: Питер Ком, 1998. -512с.
12. А.Я. Архангельский. Программирование в C++Builder5. –М.: ЗАО «Издательство БИНОМ», 2000. -1152с.
13. Геометрическое моделирование трехмерных объектов и создание баз данных для них на ЭВМ : Метод. указание (Сост.: М.В. Александров, Г.Г. Хайдаров) ЛТИ им.Ленсовета, -Л., 1991. -21с.
14. Выбор типа аксонометрической проекции пересекающихся поверхностей с помощью ЭВМ: Метод. указания (Сост.: М.В. Александров, Г.Г. Хайдаров, И.В. Можерук) ЛТИ им.Ленсовета. -Л., 1991, -12с.
15. Построение графика функции  $z=f(x,y)$  для заданной поверхности трехмерного объекта: Метод. указание (Сост.: Г.Г. Хайдаров, В.Н. Федоров) С-Пб. гос. технолог. ин-т. (технич. универ.) –СПб., 1996, -18с.
16. Вычислительная и компьютерная геометрия : Рабочая программа учебной дисциплины для школьников. (Сост.: Г.Г. Хайдаров, С.Ю. Алексеев) С-Пб. гос. технолог. ин-т. (технич. универ.) -СПб., 1997. -7с.
17. Примеры выполнения лабораторных работ по алгоритмам компьютерной графики,: Метод. указания. (Сост.: Хайдаров Г.Г., Алексеев С.Ю.) СПб., СПбГТИ(ТУ), 2005. -30 с.

# Содержание

Введение .....	3
1. Сравнение синтаксиса графических функций языка C++ в зависимости от среды программирования .....	3
2. Функции базовой графики языка C++ .....	5
2.1 Базовая графика в среде Borland C++ 3.1 .....	5
2.1.1 Инициализация графического режима .....	5
2.1.2 Типовые ошибки инициализации .....	6
2.1.3 Графические примитивы .....	6
2.2 Базовая графика в среде C++ Builder .....	9
2.2.1 Функции Canvas-графики .....	9
2.2.2 Заливка областей .....	11
3. Лабораторная работа 1. Основные функции базовой графики DOS. ....	15
3.1 Цели работы .....	15
3.2 Задание .....	15
3.3 Решение .....	15
3.4 Текст программы .....	15
3.5 Копия экрана .....	16
3.6 Оформление результатов работы .....	16
3.7 Дополнительное задание .....	17
4. Лабораторная работа 2. Основные функции базовой графики Windows. ....	17
4.1 Цели работы .....	17
4.2 Задание .....	17
4.3 Решение .....	18
4.4 Текст программы .....	18
4.5 Копия экрана .....	18
4.6 Оформление результатов работы .....	19
4.7 Дополнительные задания .....	19
5. Самостоятельная работа 1. Построение каркасной модели трехмерного объекта .....	21
5.1 Цели работы .....	21
5.2 Задание .....	21
5.3 Пояснение .....	22
5.4 Решение .....	25
5.5 Текст программы .....	25
5.6 Копия экрана .....	26
5.7 Оформление результатов работы .....	26
5.8 Дополнительное задание .....	27
6. Самостоятельная работа 2. Построение трехмерной модели с использованием библиотеки OpenGL .....	27
6.1 Цели работы .....	27
6.2 Задание .....	27
6.3 Пояснение .....	28
6.4 Решение .....	29
6.5 Текст программы .....	30
6.6 Копия экрана .....	34
6.7 Оформление результатов работы .....	35
6.8 Дополнительное задание .....	35
7. Самостоятельная работа 3. Построение трехмерной модели средствами VRML .....	35

7.1 Цели работы.....	35
7.2 Задание .....	35
7.3 Пояснение .....	36
7.4 Решение.....	38
7.5 Текст программы.....	39
7.6 Копия экрана .....	41
7.7 Оформление результатов работы .....	41
8. Самостоятельная работа 4. Построение трехмерной модели средствами графического редактора «Компас 3D».....	42
8.1 Цели работы.....	42
8.2 Задание .....	42
8.3 Пояснение .....	42
8.4 Решение.....	43
8.5 Оформление результатов работы .....	44
8.6 Дополнительное задание .....	44
9. Примеры применение методов компьютерной графики.....	46
9.1 Двухмерный конструктор .....	46
9.2 Редактор трехмерной полигональной модели.....	47
9.3 Движок для игровой программы .....	47
10. Контрольные вопросы .....	48
Литература.....	49
Основная литература .....	49
Дополнительная литература .....	49
Содержание.....	50

Геннадий Гасимович Хайдаров

Примеры выполнения самостоятельных работ по компьютерной геометрии и графике

Методические указания к самостоятельным работам

В авторской редакции

Дизайн И.И. Иванов

Зав РИО Н.Ф. Гусарова

**Редакционно-издательский отдел**

Санкт-Петербургского государственного  
университета информационных  
технологий, механики и оптики

197101, Санкт-Петербург, Кронверкский пр., 49



Лицензия № ИД 00408 от 05.11.99

Отпечатано с оригинал-макета. Формат 60х90.  $\frac{1}{16}$

Печ. л. 3,5. Тираж 100 экз

Заказ № 944

*Г.Г. Хайдаров*

# ***Примеры выполнения самостоятельных работ по компьютерной геометрии и графике***

***Методические указания к самостоятельным работам***



***Санкт-Петербург  
2006***



## **Факультет информационных технологий и программирования**

В настоящее время факультет информационных технологий и программирования сосредоточил свои усилия на трех наиболее важных для нашей страны направлениях, связанных, во-первых, с глобальными информационными сетями, во-вторых, с разработкой и эксплуатацией программного обеспечения широкого назначения и корпоративных информационно-управляющих систем и, в третьих, с компьютерными образовательными технологиями.

Своеобразной визитной карточкой факультета является кафедра компьютерных технологий, занимающая одну из первых строчек в списке сильнейших компьютерных кафедр России. Кафедра компьютерных технологий была основана в 1991 году и предназначена для отбора и обучения одаренных в области точных наук студентов и школьников по направлению 5102 «прикладная математика и информатика». Большинство из 200 обучающихся на кафедре студентов являются дипломантами городских и региональных олимпиад по математике, информатике, программированию и физике, а около 70 - победителями Всероссийских и Международных олимпиад. На кафедре сосредоточено более трети от общего числа дипломантов Всероссийских олимпиад школьников по информатике. Студенты кафедры четыре раза выигрывали полуфинальные соревнования чемпионата мира по программированию, причем в 1995 году первыми из российских вузов пробившись в финал чемпионата. В течение десяти лет, начиная с 1996 года, студенты кафедры неизменно выходили в финал чемпионата мира, в 2000 году завоевали серебряные медали чемпионата, а в 1999, 2001, 2003 и 2005 гг. – золотые, в 2004 году стали абсолютными чемпионами мира и Европы по программированию.

В 2005 году сотрудникам факультета заведующему кафедрой компьютерных технологий ректору профессору В.Н.Васильеву, декану факультета заведующему кафедрой информационных систем профессору В.Г. Парфенову, ассистентам кафедры компьютерных технологий Р.А. Елизарову и А.С. Станкевичу была присуждена Премия Президента Российской Федерации в области образования за разработку концепции и создание организационной структуры, учебно-методического и программного обеспечения инновационной системы подготовки высококвалифицированных кадров в области информационных технологий