

УО «Полоцкий государственный университет»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ №5

**к выполнению лабораторной работы
по курсу «Базы данных» для специальности
Программное обеспечение информационных технологий 1-40 01 01**

ТЕМА: Технология создания меню на экранной форме. Работа с функциями базовой графики на языке программирования C# в среде MS Visual Studio 2010.

ЦЕЛЬ: Научить пользователя технологии создания меню, при помощи **Стандартных диалоговых панелей инструментов** системы Windows.

Новополоцк 2011

Методические указания разработала:

Ст. препод. кафедры технологий программирования Бураченко Ирина Брониславовна

ЛАБОРАТОРНАЯ РАБОТА № 5

ТЕМА: Технология создания меню на экранной форме. Работа с функциями базовой графики на языке программирования C# в среде MS Visual Studio 2010.

ЦЕЛЬ: Научить пользователя технологии создания меню и панели инструментов.

Результат обучения:

После успешного завершения занятия пользователь должен:

- Изучить технологию создания меню и панели инструментов в среде MS Visual Studio 2010.
- Уметь рисовать различные графические объекты используя функции базовой графики на языке программирования C#.

Используемая программа: MICROSOFT Visual Studio 2010 язык программирования C#

План занятия:

1. Изучить графические методы работы на языке программирования C# в среде MS Visual Studio 2010.
2. Выполнение самостоятельного задания по созданию меню.

1. ФУНКЦИИ БАЗОВОЙ ГРАФИКИ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C# И РАБОТА В СРЕДЕ MS VISUAL STUDIO 2010

Для того чтобы студент мог прочесть листинг (текст программы), содержащий графические функции на языках C++, Java, C# проведем небольшое сравнение на примере функции рисования прямоугольника (Rectangle). Функция Rectangle рисует прямоугольник, используя выбранное перо, и может закрасить его внутренность с помощью выбранной кисти.

Строка кода на языке C++ в среде C++ Builder для рисования прямоугольника (Rectangle) на поверхности «холста» (Canvas) формы (Form1) приложения имеет вид:

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

Здесь X1, Y1 и X2, Y2 — координаты верхнего левого и правого нижнего угла прямоугольника.

Строка кода на языке C++ в среде Visual C++ для рисования прямоугольника на форме приложения с функцией Win32 API имеет вид:

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

Здесь hdc — идентификатор контекста устройства HDC (или дескриптор окна).

Другим вариантом Visual C++ в MS Visual Studio является создание приложения Windows Forms Application. В этом случае фрагмент кода C++ для рисования прямоугольника на форме приложения может выглядеть так:

```
e->Graphics->DrawRectangle(pen, X1, Y1, W, H);
```

Здесь pen — имя объекта класса Pen с параметрами линии, W — ширина и H — высота рисуемого прямоугольника.

На языке Java графические функции используют не только для приложений. Их широко используют для апплетов и мидлетов.

Рассмотрим фрагмент кода на Java для рисования прямоугольника:

```
g.drawRect(X1, Y1, W, H);
```

Здесь g — любое имя для указателя на класс Graphics,

Рассмотрим фрагмент кода на C# для рисования прямоугольника:

```
g.DrawRectangle(pen, X1, Y1, W, H);
```

Здесь g — любое имя для указателя на класс Graphics.

Обобщая данные функции, заметим, что в каждой из них указывается место рисования и задаются размеры прямоугольника, параметры линии, как правило, задаются по умолчанию. Как видно из примера с функцией Rectangle мы имеем большую схожесть графических классов для разных языков программирования. Поэтому нетрудно прочитать и перевести код с графическими функциями с одного языка на другой. Особенно большое структурное сходство классов графических функций между языками Java и C#. В современной литературе графические функции более часто называются методами, что не меняет их назначения.


Рассмотрим последовательность создания приложения WindowsFormsApplication (WindowsApplication) с графическими функциями.

2. СПОСОБЫ ВЫВОДА ГРАФИКИ НА ФОРМУ ПРИЛОЖЕНИЯ

Для вывода графических методов можно применять три метода инициализации графики, что связано с получением указателя на область вывода графических функций. В самом простом случае в качестве области вывода выберем форму приложения (по умолчанию Form1)

2.1. Инициализация графики методом Paint

Включим в свою программу событие формы Paint:

1. После создания приложения WindowsApplication откроем окно свойств формы (Свойства - Properties).
2. В данном окне перейдем на вкладку методов, нажав на пиктограмме «молния»  с всплывающей подсказкой (события – Events).
3. Из таблицы методов выберем метод Paint.

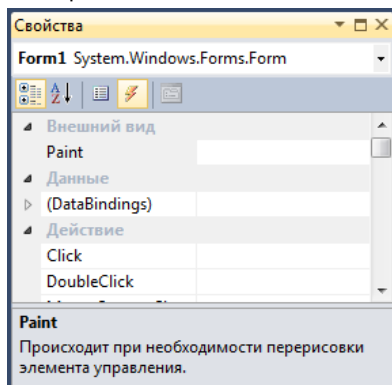


Рис. 1. Окно свойств формы Form1

4. Выполним двойной щелчок в пустом поле таблицы справа от (выделенной на рисунке) ячейки со словом «Paint». При правильной работе автоматически должен сгенерироваться метод Form1_Paint и в файл Form1.cs вписывается фрагмент кода.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
}
```

5. Впишем в метод Form1_Paint программный код:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics; // «Бесплатная» инициализация графики
    Pen bluePen = new Pen(Color.Blue, 2);
    Pen blackPen = new Pen(Color.Black, 10);
    g.DrawLine(blackPen, 10, 20, 110, 30);
    g.DrawRectangle(bluePen, 10, 40, 100, 50);
}
```

6. Получим работающее приложение:

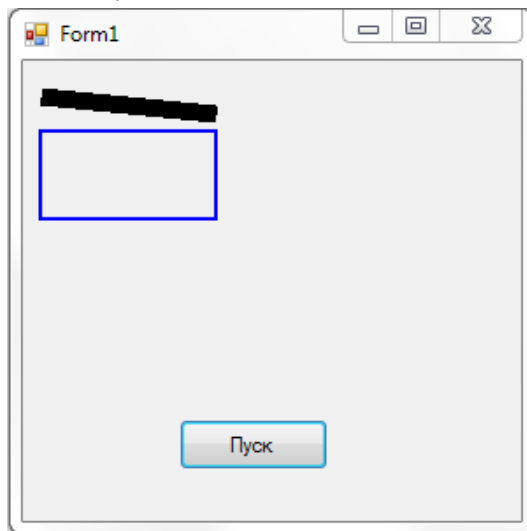


Рис. 2. Работа приложения с рисованием на Form1

Достоинством данной инициализации является «бесплатная» инициализация графики. Недостатком такого вида инициализации графики является отсутствие управляющей последовательности вывода графики на форму.

Обратим особое внимание на то, что на данном примере выбора события Paint для формы приложения Form1, было показано, как вставлять в код программы стандартные события. По такой последовательности выбора события из таблицы событий возможно включать в свою программу любые доступные события для данного компонента.

2.2. Инициализация графики методом Create

Теперь впишем инициализацию графики в управляющую кнопку через событие: «нажать кнопку» - button1_Click. В этом случае удастся избежать предыдущего

недостатка, то есть графика появляется на форме Form1 только после нажатия кнопки button1. Последовательность действий аналогична пункту 2.1.

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = CreateGraphics(); // инициализация графики
    Pen bluePen = new Pen(Color.Blue, 2);
    Pen blackPen = new Pen(Color.Black, 10);
    g.DrawLine(blackPen, 10, 10, 110, 20);
    g.DrawRectangle(bluePen, 20, 30, 100, 50);
}
```

2.3. Инициализация графики методом FromHwnd

Теперь для инициализации графики применим метод FromHwnd, который позволяет указывать на место вывода графики.

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = Graphics.FromHwnd(this.Handle);
    // инициализация графики
    Pen bluePen = new Pen(Color.Blue, 2);
    Pen blackPen = new Pen(Color.Black, 10);
    g.DrawLine(blackPen, 10, 10, 110, 20);
    g.DrawRectangle(bluePen, 20, 30, 100, 50);
}
```

В данном конкретном случае местом вывода является указатель (Handle) на созданный нами объект (this) приложения. В общем случае с использованием метода FromHwnd() мы можем применять указатель не только на форму приложения, но и на любой визуальный объект, поддерживающий класс Graphics.

3. ФУНКЦИИ ГРАФИКИ SYSTEM.DRAWING

Графика в C# основана на графических интерфейсах (Graphics Device Interface) – в подсистеме Windows для вывода графических изображений. Пространства имен в C# для работы с графикой следующие:

- System.Drawing; System.Drawing.Drawing2D; System.Drawing.Imaging;
- System.Drawing.Printing; System.Drawing.Text.

Разберем текст кода программы внутри события button1_Click с использованием базовой графики на языке C#. Для желающих более подробно познакомиться с возможностями данного класса System.Drawing на языке C# предлагается выполнить данную работу, используя help online или учебник по C#. К функциям базовой графики System.Drawing относятся функции данной программы. Пример программного кода приводится.

Листинг 1. Фрагмент кода графики System.Drawing

```
int x0 = 200,y0=100;float xn = 20,xk=140; float yn=10, yk = 20;
//this.BackColor=Color.FromArgb(10,0,0,10);
SolidBrush b=new SolidBrush(Color.FromArgb(150,0,0,255));
Font f=new Font("Areal",15);
Graphics canvas= Graphics.FromHwnd(this.Handle);
// где canvas - любое имя для указателя
Color c3=Color.Red; Pen pen=new Pen(c3,7.0f);
canvas.DrawRectangle(pen,5,5,280,150);
canvas.DrawString("using System.Drawing", f, b, 25, 70);
Pen pen2 = new Pen(Color.Green, 14.0f);
canvas.DrawLine(pen2,x0-xn,y0+yn,x0-xk,y0+yk);
```

```
canvas.DrawEllipse(pen2, 100, 170, 80, 40);
```

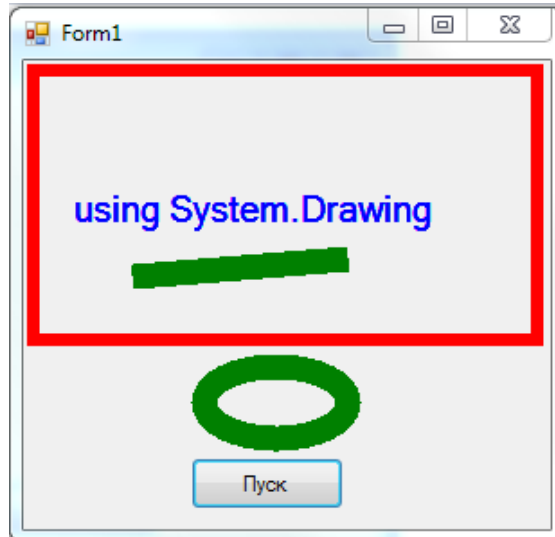


Рис. 3. Работа приложения с рисованием базовой графики

4. ФУНКЦИИ ГРАФИКИ SYSTEM.DRAWING.DRAWING2D

В данном программном коде разберем расширенные возможности класса на конкретных примерах System.Drawing.Drawing2D. Для желающих более подробно познакомиться с возможностями данного класса в C# предлагается выполнить данную работу, используя help online или учебник по C#.

Обратим особое внимание на то, что класс System.Drawing автоматически включается в программу при создании приложения, а для класса System.Drawing.Drawing2D программисту необходимо обязательно вписать объявление этого класса вручную.

Рассмотрим пример в C# с графикой System.Drawing и графикой System.Drawing.Drawing2D

Листинг 2. Фрагмент кода графики System.Drawing.Drawing2D

```
Graphics canvas=Graphics.FromHwnd(this.Handle);  
Pen pen=new Pen(Color.Green,2);  
canvas.DrawRectangle(pen,10,20,50,50);  
canvas.DrawLine(pen,100,20,200,40);  
Point p1=new Point(100,40);Point p2=new Point(200,60);  
// вписать вручную до начала использования функций класса  
using System.Drawing.Drawing2D;  
pen.Color=Color.Red;  
pen.Width=8;  
pen.DashStyle=DashStyle.DashDotDot;  
//  
Array obj=Enum.GetValues(typeof(LineCap));  
LineCap t1=(LineCap)obj.GetValue(6); pen.StartCap=t1;  
LineCap t2=(LineCap)obj.GetValue(8); pen.EndCap=t2;
```

```
//
canvas.DrawLine(pen,p1,p2);
//
SolidBrush b=new SolidBrush(Color.Blue);
canvas.FillPie(b,10,150,50,60,0,270);
//
Array obj2=Enum.GetValues(typeof(HatchStyle));
HatchStyle t3=(HatchStyle)obj2.GetValue(8);
HatchBrush b3=new HatchBrush(t3,Color.Blue,Color.Gold);
canvas.FillEllipse(b3,100,150,50,60);
//
Font font=new Font("Arial",20,FontStyle.Bold|FontStyle.Underline);
float f1=font.GetHeight(); String s1="";s1=s1+f1.ToString();
int f2=(int)f1; String s2="";s2=s2+f2.ToString();
SolidBrush br=new SolidBrush(Color.Gray);
canvas.DrawString("Текст Text",font,br,10,110);
float f3=font.Size; int f4; f4=(int)f3;s2=s2+" "+f3.ToString();
canvas.DrawString(s1,font,br,160,110);
canvas.DrawString(s2,font,br,160,150);
```

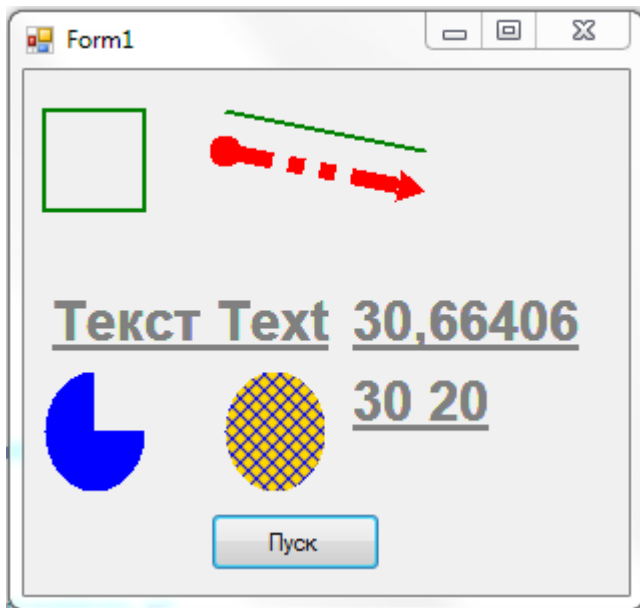


Рис. 4. Работа приложения с рисованием базовой графики и графики *Drawing2D*

5. ОСНОВЫ ПРОГРАММИРОВАНИЯ ТРЕХМЕРНОЙ ГРАФИКИ В C# С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OPENGL

Открытая графическая библиотека OpenGL хороша тем, что имеет международный стандарт и разработана для всех платформ и основных языков программирования, в том числе для языков си-язычного направления: C++, Java, C#.

5.1. Основы OpenGL

OpenGL является популярным прикладным программным интерфейсом (API – Application Programming Interface) для разработки приложений в области двумерной и трехмерной графики. Стандарт OpenGL (Open Graphics Library – открытая графическая библиотека) был разработан и утвержден в 1992 году как эффективный аппаратно-независимый интерфейс, пригодный для реализации на различных платформах.

Основой стандарта стала библиотека IRIS GL, разработанная фирмой Silicon Graphics Inc. OpenGL состоит из набора библиотек. Все базовые функции хранятся в основной библиотеке, для обозначения которой в дальнейшем мы будем использовать аббревиатуру GL. Первая из них – библиотека утилит GL.

Все функции этой библиотеки вошли графические примитивы, например: точки, линии, полигоны. Все функции имеют префикс gl. Вторая GLU – GL Utility. В состав GLU вошла реализация более сложных функций, таких как набор популярных геометрических примитивов (куб, шар, цилиндр, диск), функции построения сплайнов, реализация дополнительных операций над матрицами и т.п.

OpenGL не включает в себя никаких специальных команд для работы с окнами или ввода информации от пользователя. Поэтому были созданы специальные переносимые библиотеки для обеспечения часто используемых функций взаимодействия с пользователем и для отображения информации с помощью оконной подсистемы. Наиболее популярной является библиотека GLUT (GL Utility Toolkit). Формально GLUT не входит в OpenGL, но de facto включается почти во все его дистрибутивы и имеет реализации для различных платформ. GLUT предоставляет только минимально необходимый набор функций для создания OpenGL-приложения.

5.2. Графические команды (функции) OpenGL

Все команды (функции) библиотеки GL начинаются с префикса gl, все константы – с префикса GL_. Соответствующие команды и константы библиотек GLU и GLUT аналогично имеют префиксы glu (GLU_) и glut (GLUT_)

5.3. Синтаксис команд

Кроме того, в имена команд входят суффиксы, несущие информацию о числе и типе передаваемых параметров. В OpenGL полное имя команды имеет вид:

```
type glCommand_name[1 2 3 4][b s i f d ub us ui][v](type1 arg1, ..., typeN argN)
```

Имя состоит из нескольких частей:

имя библиотеки, в которой описана эта функция: для базовых функций OpenGL, функций из библиотек GL, GLU, GLUT это gl, glu, glut соответственно.

Command_name имя команды (функции)

[1 2 3 4] число аргументов команды

[b s i f d ub us ui] тип аргумента: символ b – GLbyte (аналог char в C/C++), символ s – short, символ i – GLint (аналог int), символ f – GLfloat (аналог float), символ d – double и так далее.

[v] наличие этого символа показывает, что в качестве параметров функции используется указатель на массив значений. Символы в квадратных скобках в некоторых названиях не используются.

Например, команда glVertex2i() описана в библиотеке GL, и использует в качестве параметров два целых числа, а команда glColor3fv() использует в качестве параметра указатель на массив из трех вещественных чисел.

5.4. Вершины и примитивы

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 в OpenGL мы здесь рассматривать не будем.

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 α -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, можно написать так:

```
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.

5.5. Скелет программы OpenGL

Существует два варианта использования библиотеки OpenGL для языка программирования C#. Первый вариант – это импорт системных библиотек Microsoft Windows. Второй вариант – это применение библиотек OpenGL, разработанных под язык программирования C#.

6. ПРИМЕНЕНИЕ БИБЛИОТЕК OPENGL ДЛЯ ЯЗЫКА C#

Библиотечные файлы OpenGL для C# можно скачать с сайта <http://csgl.sourceforge.net>. Создадим код трехмерных объектов с использованием библиотеки OpenGL с сайта <http://nehe.gamedev.net>. Разберем код исходного файла. Обратите особое внимание на строчку кода `using CsGL.Basecode;` и метод `private override void Draw()` с командами (функциями) OpenGL.

Преимуществом данного варианта является простота инициализации базового кода при работе с командами OpenGL. Особое внимание следует обратить на наличие рядом с исполняемым файлом динамических библиотек `CsGL.Basecode.dll`, `csgl.dll`, `csgl.native.dll`. Кроме того, мы должны в явном виде подключить в тексте программы ссылку, записав строку `using CsGL.Basecode;`

Листинг 3. Текст программы применения библиотеки OpenGL

```
////////
using System;
using System.Windows.Forms;
using System.Drawing;
using CsGL.OpenGL;
namespace lesson05
```

```

{
public class OurView : OpenGLControl
{
public float rtri; // rtri is for rotating the pyramid
public float rquad; // rquad is for rotating the quad
public bool finished;
public OurView() : base()
{
this.KeyDown += new KeyEventHandler(OurView_OnKeyDown);
finished = false;
}
protected void OurView_OnKeyDown(object Sender, KeyEventArgs kea)
{
//if escape was pressed exit the application
if (kea.KeyCode == Keys.Escape)
{
finished = true;
}
}

public override void glDraw()
{
GL.glClear(GL.GL_COLOR_BUFFER_BIT |
GL.GL_DEPTH_BUFFER_BIT);
// Clear the Screen and the Depth Buffer
GL.glMatrixMode(GL.GL_MODELVIEW);
// Modelview Matrix
GL.glLoadIdentity();
// reset the current modelview matrix
GL.glTranslatef(-1.5f,0.0f,-6.0f);
// move 1.5 Units left and 6 Units into the screen
GL.glRotatef(rtri,0.0f,1.0f,0.0f);
// rotate the Pyramid on it's Y-axis
rtri+=0.2f;
// rotation angle
GL.glBegin(GL.GL_TRIANGLES);
//start drawing a triangle, always counterclockside (top-left-right)
GL.glColor3f(1.0f,0.0f,0.0f); // Red
GL.glVertex3f(0.0f,1.0f,0.0f); // Top of Triangle (Front)
GL.glColor3f(0.0f,1.0f,0.0f); // green
GL.glVertex3f(-1.0f,-1.0f,1.0f); // left of Triangle (front)
GL.glColor3f(0.0f,0.0f,1.0f); // blue
GL.glVertex3f(1.0f,-1.0f,1.0f); // right of triangle (front)
GL.glColor3f(1.0f,0.0f,0.0f); // red
GL.glVertex3f(0.0f,0.0f,1.0f); // top of triangle (right)
GL.glColor3f(0.0f,0.0f,1.0f); // blue
GL.glVertex3f(1.0f,-1.0f,1.0f); // left of triangle (right)
GL.glColor3f(0.0f,1.0f,0.0f); // green
GL.glVertex3f(1.0f,-1.0f,-1.0f); // right of triangle (right)
GL.glColor3f(1.0f,0.0f,0.0f); // red
GL.glVertex3f(0.0f,1.0f,0.0f); // top of triangle (back)
GL.glColor3f(0.0f,1.0f,0.0f); // green
GL.glVertex3f(1.0f,-1.0f,-1.0f); // left of triangle (back)
GL.glColor3f(0.0f,0.0f,1.0f); // blue
GL.glVertex3f(-1.0f,-1.0f,-1.0f); // right of triangle (back)
GL.glColor3f(1.0f,0.0f,0.0f); // red
GL.glVertex3f(0.0f,1.0f,0.0f); // top of triangle (left)

```

```

GL.glColor3f(0.0f,0.0f,1.0f); // blue
GL.glVertex3f(-1.0f,-1.0f,-1.0f); // left of triangle (left)
GL.glColor3f(0.0f,1.0f,0.0f); // green
GL.glVertex3f(-1.0f,-1.0f,1.0f); // right of triangle (left)
GL.glEnd();
GL.glLoadIdentity(); // reset the current modelview matrix
GL.glTranslatef(1.5f,0.0f,-7.0f);
// move 1.5 Units right and 7 into the screen
GL.glRotatef(rquad,1.0f,1.0f,1.0f);
// rotate the quad on the X,Y and Z-axis
rquad=0.15f; // rotation angle
GL.glBegin(GL.GL_QUADS); // start drawing a quad
GL.glColor3f(0.0f,1.0f,0.0f); // green top
GL.glVertex3f(1.0f,1.0f,-1.0f); // top right (top)
GL.glVertex3f(-1.0f,1.0f,-1.0f); // top left (top)
GL.glVertex3f(-1.0f,1.0f,1.0f); // bottom left (top)
GL.glVertex3f(1.0f,1.0f,1.0f); // bottom right (top)
GL.glColor3f(1.0f,0.5f,0.0f); // orange
GL.glVertex3f(1.0f,-1.0f,1.0f); // top right (bottom)
GL.glVertex3f(-1.0f,-1.0f,1.0f); // top left (bottom)
GL.glVertex3f(-1.0f,-1.0f,-1.0f); // bottom left (bottom)
GL.glVertex3f(1.0f,-1.0f,-1.0f); // bottom right (bottom)
GL.glColor3f(1.0f,0.0f,0.0f); // red
GL.glVertex3f(1.0f,1.0f,1.0f); // top right (front)
GL.glVertex3f(-1.0f,1.0f,1.0f); // top left (front)
GL.glVertex3f(-1.0f,-1.0f,1.0f); // bottom left (front)
GL.glVertex3f(1.0f,-1.0f,1.0f); // bottom right (front)
GL.glColor3f(1.0f,1.0f,0.0f); // yellow
GL.glVertex3f(-1.0f,1.0f,-1.0f); // top right (back)
GL.glVertex3f(1.0f,1.0f,-1.0f); // top left (back)
GL.glVertex3f(1.0f,-1.0f,-1.0f); // bottom left (back)
GL.glVertex3f(-1.0f,-1.0f,-1.0f); // bottom right (back)
GL.glColor3f(0.0f,0.0f,1.0f); // blue
GL.glVertex3f(-1.0f,1.0f,1.0f); // top right (left)
GL.glVertex3f(-1.0f,1.0f,-1.0f); // top left (left)
GL.glVertex3f(-1.0f,-1.0f,-1.0f); // bottom left (left)
GL.glVertex3f(-1.0f,-1.0f,1.0f); // bottom right (left)
GL.glColor3f(1.0f,0.0f,1.0f); // violett
GL.glVertex3f(1.0f,1.0f,-1.0f); // top right (right)
GL.glVertex3f(1.0f,1.0f,1.0f); // top left (right)
GL.glVertex3f(1.0f,-1.0f,1.0f); // bottom left (right)
GL.glVertex3f(1.0f,-1.0f,-1.0f); // bottom right (right)
GL.glEnd();
}

protected override void InitGLContext()
{
    GL.glShadeModel(GL.GL_SMOOTH); // enable smooth shading
    GL.glClearColor(0.90f, 0.90f, 0.90f, 0.5f); // background
    GL.glClearDepth(1.0f); // depth buffer setup
    GL.glEnable(GL.GL_DEPTH_TEST); // enables depth testing
    GL.glDepthFunc(GL.GL_LEQUAL); // type of depth test
    GL.glHint(GL.GL_PERSPECTIVE_CORRECTION_HINT,
    GL.GL_NICEST);
    // nice perspective calculations
    //rtri = 30.0f; define the rotation angle in the start position of
    the

```

```

triangle
//rquad = 30.0f; define the rotation angle in the start position of
the quad
}

protected override void OnSizeChanged(EventArgs e)
{
    base.OnSizeChanged(e);
    Size s = Size;
    GL.glMatrixMode(GL.GL_PROJECTION);
    GL.glLoadIdentity();
    GL.gluPerspective(45.0f, (double)s.Width / (double) s.Height, 0.1f,
    100.0f);
    GL.glMatrixMode(GL.GL_MODELVIEW);
    GL.glLoadIdentity();
}
}

public class MainForm : System.Windows.Forms.Form
{
    private lesson05.OurView view;
    public MainForm()
    {
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.ClientSize = new System.Drawing.Size(640, 480);
        this.Name = "MainForm";
        this.Text = "NeHe lesson 05 in C# (by Sabine Felsinger)";
        this.view = new lesson05.OurView();
        this.view.Parent = this;
        this.view.Dock = DockStyle.Fill; // Will fill whole form
        this.Show();
    }
    static void Main()
    {
        MainForm form = new MainForm();
        while ((!form.view.finished) && (!form.IsDisposed))
            // refreshing the window, so it rotates
        {
            form.view.glDraw();
            form.Refresh();
            Application.DoEvents();
        }
        form.Dispose();
    }
}
}

```

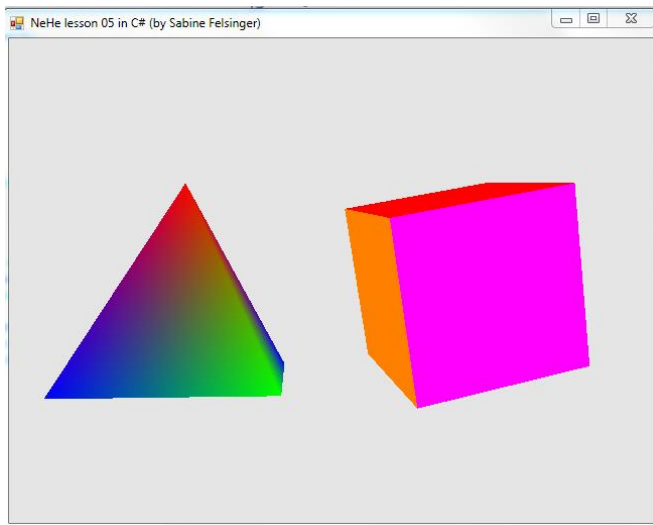


Рис. 5. Пример работы программы

7. САМОСТОЯТЕЛЬНОЕ ЗАДАНИЕ ПО СОЗДАНИЮ МЕНЮ

Построить меню, предназначенное для рисования геометрических фигур:

1. Меню содержит пункты (опции) **Масштаб, Фигура, Заливка**
2. Пункт **Масштаб** должен содержать 2 подпункта: команду **Координаты углов** графического окна, в котором будет изображена фигура, и команду **Установка масштаба**.
3. Пункт **Фигура** должен содержать 4 подпункта: **Квадрат, Прямоугольник, Окружность, Овал**.
4. Каждый из них в свою очередь должен содержать подпункты: команды для ввода координат и размеров данной фигуры в выбранном масштабе и команду **Нарисовать** для изображения данной фигуры.
5. Пункт **Заливка** будет содержать 2 подпункта: **Стиль** (Сплошная заливка, Горизонтальная штриховка, Вертикальная штриховка) и **Цвет** (Красный, Зеленый, Синий).
6. Создайте панель инструментов и продублируйте выполнение основных элементов на панели инструментов.

Продемонстрируйте Вашу работу преподавателю!

КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Перечислите функции базовой графики на языке программирования C# в среде MS Visual Studio 2010.
2. С какими способами вывода графики на форму приложения Вы знакомы?
3. Опишите известные Вам приемы программирования трехмерной графики в C# с использованием библиотеки OpenGL.
4. Как создать меню и панель инструментов в среде MS Visual Studio 2010 C#?