

ЛАБОРАТОРНАЯ РАБОТА №2. OPENGL

Стандарт OpenGL (Open Graphics Library - открытая графическая библиотека) был создан и утвержден как эффективный аппаратно-независимый интерфейс, пригодный для реализации на различных платформах. Для использования функций стандарта OpenGL в языке C# будет использоваться библиотека OpenTK, она является открытой и бесплатной.

OPENTK

Open Toolkit позволяет использовать OpenGL, OpenGL|ES, OpenCL и OpenAL APIs из управляемых языков.

OpenTK начинался как экспериментальное ответвление от "Tao framework" в начале лета 2006 года. Оригинальной целью было создать более чистую обертку (wrapper) чем Tao.OpenGL, однако достаточно быстро появилась другая задача - обеспечить доступ к начальной инициализации Khronos и Creative API. В этом смысле Open Toolkit похож на такие проекты, как Tao, SlimDX, SDL или GLFW.

Однако в отличие от этих библиотек OpenTK значительно сосредоточен на формировании удобного интерфейса. Вместо указателей, OpenTK предоставляет обобщенные классы (generic). Вместо простых констант, OpenTK использует строго-типизированные перечисления. Вместо обычного списка функций, OpenTK разделяет функции по категориям. Также в состав OpenTK включена достаточно обширная математическая библиотека, которая может быть использована из любого API.

1. УСТАНОВКА OPENTK И ПОДКЛЮЧЕНИЕ К ПРОЕКТУ.

Скачайте установщик OpenTK с официального сайта [1] и установите. Создайте новый проект Windows Forms Application так же, как в предыдущей лабораторной работе. Откройте SolutionExplorer, нажмите ПКМ по папке References -> Add Reference. В открывшемся окне выберите вкладку Browse и найдите библиотеки OpenTK.dll и OpenTK.GLControl.dll (По умолчанию dll лежат в папке Documents -> OpenTK -> 1.1 -> Binaries -> OpenTK -> Release). Откройте панель ToolBox, Нажмите ПКМ в свободной области -> Choose Items, на вкладке .NET Framework Components включите элемент GLControl, и добавьте его на форму. На форме появится черный прямоугольник.

2. НАСТРОЙКА ОКНА ВЫВОДА OPENGL И ПЕРСПЕКТИВНОЙ ПРОЕКЦИИ.

Создайте новый класс GLGraphics. С помощью директивы using подключите OpenTK.

```
using OpenTK;
```

```
using OpenTK.Graphics.OpenGL;
```

Создайте функцию Init(), которая будет инициализировать начальное состояние OpenGL/

Создайте функцию Resize() с параметрами width и height – размерами окна для OpenGL.

```
public void Resize(int width, int height)
{
```

Вызовите функцию ClearColor, заливающую буфер экрана одним цветом. Функцией ShadeModel установите тип отрисовки полигонов с оттенками (smooth shading). Включите буфер глубины.

```
GL.ClearColor(Color.DarkGray);
```

```
GL.ShadeModel(ShadingModel.SMOOTH);
```

```
GL.Enable(EnableCap.DepthTest);
```

Создайте матрицу проекции, настройте ее согласно по размерам окна, и загрузите в контекст OpenGL. Для дополнительной информации про матрицу проекции прочитайте тут[2].

```
Matrix4 perspectiveMat = Matrix4.CreatePerspectiveFieldOfView(
    MathHelper.PiOver4,
    width / (float)height,
    1,
    64);
GL.MatrixMode(MatrixMode.Projection);
```

```
GL.LoadMatrix(ref perspectiveMat);
```

В классе GLGraphics создайте функцию Update(). Добавьте в нее строку, очищающую буферы.

```
public void Update()  
{GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
```

В классе Form1 создайте экземпляр класса GLGraphics и инициализируйте конструктором по умолчанию.

```
public partial class Form1 : Form  
{GLGraphics GLGraphics = new GLGraphics();
```

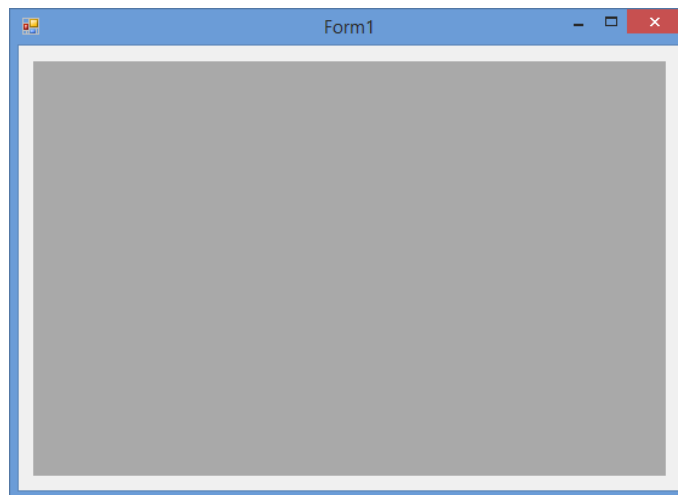
Откройте свойства вашего glControl, откройте список доступных Events и двойным щелчком создайте метод Load, и в нем вызовите функцию Setup с размерами вашего glcontrol она в качестве параметров.

```
private void glControl1_Load(object sender, EventArgs e)  
{glGraphics.Resize(glControl1.Width, glControl1.Height);
```

Аналогичным способом создайте метод Paint, и в нем вызовите метод Update.

```
private void glControl1_Paint(object sender, PaintEventArgs e)  
{  
    GLGraphics.Update();  
    glControl1.SwapBuffers();  
}
```

Запустите программу. На данном этапе glControl должен окраситься в серый цвет.



3. НАСТРОЙКА ПОЗИЦИИ КАМЕРЫ И РИСОВАНИЕ ТЕСТОВОГО КВАДРАТА.

В классе GLGraphics создайте функцию drawTestQuad. Рисование примитивов в OpenGL начинается функцией glBegin(), за которой идут координаты вершин (вместе с координатами вершин могут идти цвет этих вершин, нормали, текстурные координаты), и заканчивается рисование функцией glEnd. OpenTK оборачивает вызовы функций на языке C в свои C# функции.

```
private void drawTestQuad()  
{  
    GL.Begin(PrimitiveType.Quads);  
    GL.Color3(Color.Blue);  
    GL.Vertex3(-1.0f, -1.0f, -1.0f);  
    GL.Color3(Color.Red);  
    GL.Vertex3(-1.0f, 1.0f, -1.0f);  
    GL.Color3(Color.White);  
    GL.Vertex3(1.0f, 1.0f, -1.0f);  
    GL.Color3(Color.Green);  
    GL.Vertex3(1.0f, -1.0f, -1.0f);  
    GL.End();  
}
```

Создайте функцию Render, в которой вызовите только что созданный метод drawTestQuad.

```
public void Render()  
{  
    drawTestQuad();
```

```
}
```

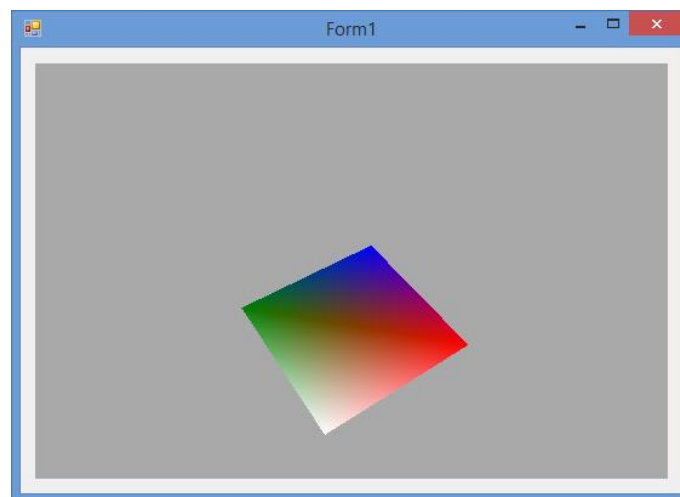
Создайте переменные Vector3, в которых будет храниться параметры положения камеры.

```
class GLGraphics
{
    Vector3 cameraPosition = new Vector3(2, 3, 4);
    Vector3 cameraDirecton = new Vector3(0, 0, 0);
    Vector3 cameraUp = new Vector3(0, 0, 1);
```

В уже созданную функцию Update добавьте код, который создает матрицу преобразования вида с помощью координат камеры и загружает матрицу в контекст OpenGL. Вызовите функцию Render для отрисовки тестового квадрата.

```
public void Update()
{
    GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
    Matrix4 viewMat = Matrix4.LookAt(cameraPosition, cameraDirecton, cameraUp);
    GL.MatrixMode(MatrixMode.Modelview);
    GL.LoadMatrix(ref viewMat);
    Render();
}
```

Запустите программу, на экране должен появиться квадрат, раскрашенный градиентом.



4. ДОБАВЛЕНИЕ ИНТЕРАКТИВНОСТИ

Для того, чтобы лучше почувствовать объем, нужно реализовать передвижение в пространстве. В данном пособии перемещение в пространстве будет реализовано следующим образом: Камера устремлена в центр координат и движется по окружающей сфере подобно спутнику.

Создайте в классе переменные для радиуса окружающей сферы и угла по широте и долготе.

```
public float latitude = 47.98f;
public float longitude = 60.41f;
public float radius = 5.385f;
```

В функции Update создайте вычисление текущей позиции камеры по радиусу и углам широты и долготы, для вычислений используйте формулы перехода от сферических координат к декартовым.

```
cameraPosition = new Vector3(
    (float)(radius*Math.Cos(Math.PI/180.0f*latitude)*Math.Cos(Math.PI/180.0f*longitude)),
    (float)(radius*Math.Cos(Math.PI/180.0f*latitude)*Math.Sin(Math.PI/180.0f*longitude)),
    (float)(radius * Math.Sin(Math.PI / 180.0f * latitude)));
```

Откройте методы Events у расположенного на форме glControl и двойным щелчком создайте метод MouseMove. В методе пропишите преобразование координат текущего положения мыши в расстояние от центра изображения до текущего положения мыши, нормализуйте от -0.5 до 0.5, умножьте

```
private void glControl1_MouseMove(object sender, MouseEventArgs e)
```

```

{
    float widthCoef = (e.X - glControl1.Width * 0.5f) / (float)glControl1.Width;
    float heightCoef = (-e.Y + glControl1.Height * 0.5f) / (float)glControl1.Height;
    GLGraphics.latitude = heightCoef * 180;
    GLGraphics.longitude = widthCoef * 360;
}

```

5. ТРАНСФОРМАЦИЯ ОБЪЕКТА

Трансформировать объекты принято в следующем порядке:

1. Масштабирование (функция scale);
2. Поворот (функция rotate);
3. Перемещение (функция translate).

Чтобы трансформация затрагивала только один объект, до трансформаций вызывается функция `pushMatrix`, а после отрисовки трансформированного объекта функция `popMatrix`. Для объектов, рисуемых после данного вызова, трансформации нужно применять заново. Отдельно можно уточнить функцию `Rotate`: в качестве параметров для нее используются угол, на который нужно повернуть рисуемый объект, и вектор, вокруг которого поворот происходит.

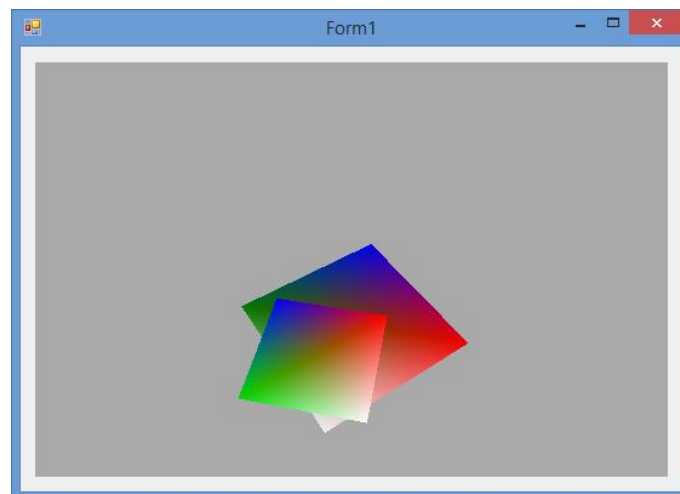
В функцию `Render` добавьте код, который рисует копию уже нарисованного квадрата с примененными к нему трансформациями.

```

drawTestQuad();
GL.PushMatrix();
GL.Translate(1, 1, 1);
GL.Rotate(45, Vector3.UnitZ);
GL.Scale(0.5f, 0.5f, 0.5f);
drawTestQuad();
GL.PopMatrix();

```

Результатом работы вашей программы должно быть подобное изображение.



Самый простой способ создания анимации вращения – хранить значение угла в переменной, которую изменять по определенному закону, и перерисовывать изображение.

В классе `GLGraphics` создайте переменную `rotateAngle` типа `float`. В функции `Update` ее увеличьте значение на константу, например, на `0.1`. В методе `Render` используйте эту переменную в качестве значения угла поворота. Создайте функцию `Application_Idle`, которая будет заставлять вашу форму перерисовываться.

```

void Application_Idle(object sender, EventArgs e)
{
    while (glControl1.IsIdle)

```

```

        glControl1.Refresh();
    }

```

В методе `glcontrol_load` подключите созданную нами функцию к событию формы `idle`.

```
Application.Idle += Application_Idle;
```

6. НАЛОЖЕНИЕ ТЕКСТУРЫ

Для наложения текстур на объекты используются текстурные координаты.

В классе `GLGraphics` создайте функцию `loadTexture`, которая загружает текстуру из изображения в память видеокарты. Результатом работы этой функции будет являться ID текстуры, который будет использоваться для того, чтобы указать, какую текстуру применять.

```

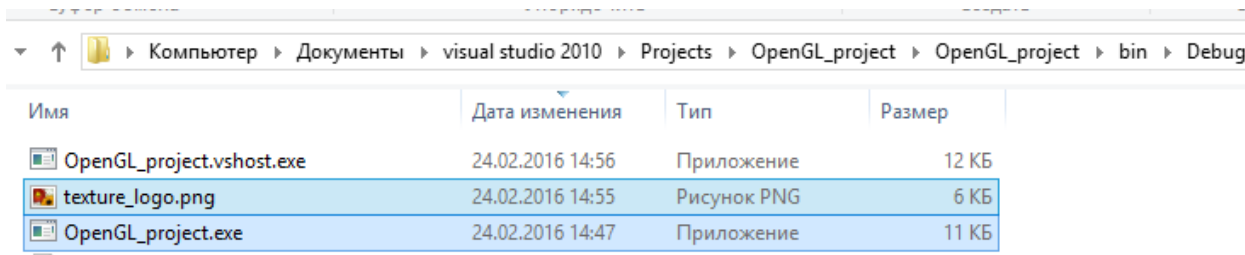
public int LoadTexture(String filePath)
{
    try
    {
        Bitmap image = new Bitmap(filePath);
        int texID = GL.GenTexture();
        GL.BindTexture(TextureTarget.Texture2D, texID);
        BitmapData data = image.LockBits(
            new System.Drawing.Rectangle(0, 0, image.Width, image.Height),
            ImageLockMode.ReadOnly, System.Drawing.Imaging.PixelFormat.Format32bppArgb);
        GL.TexImage2D(TextureTarget.Texture2D, 0,
            PixelInternalFormat.Rgba, data.Width, data.Height, 0,
            OpenTK.Graphics.OpenGL.PixelFormat.Bgra, PixelType.UnsignedByte, data.Scan0);
        image.UnlockBits(data);
        GL.GenerateMipmap(GenerateMipmapTarget.Texture2D);
        return texID;
    }
    catch (System.IO.FileNotFoundException e)
    {
        return -1;
    }
}

```

Создайте список номеров текстур и инициализируйте конструктором по умолчанию.

```
public List<int> texturesIDs = new List<int>();
```

Скопируйте файл, который будете использовать в качестве текстуры, в папку с исполняемым файлом (.exe). Чтобы открыть папку с исполняемым файлом, откройте `Solution Explorer`, нажмите по проекту ПКМ -> `Open Folder in Windows Explorer`. В открывшемся окне перейдете в папку `bin -> Debug` (или `Release`, если вы собираете `Release` версию). Скопируйте вашу текстуру в данную папку.



Имя	Дата изменения	Тип	Размер
OpenGL_project.vshost.exe	24.02.2016 14:56	Приложение	12 КБ
texture_logo.png	24.02.2016 14:55	Рисунок PNG	6 КБ
OpenGL_project.exe	24.02.2016 14:47	Приложение	11 КБ

Загрузите текстуру с помощью написанной вами функции. Загрузить текстуру нужно один раз при запуске программы. Нет разницы в том, вызывать загрузку текстуры из класса `Form1` или из класса `GLGraphics`. Например, при загрузке текстур внутри функции `glControl_load` код будет выглядеть так:

```

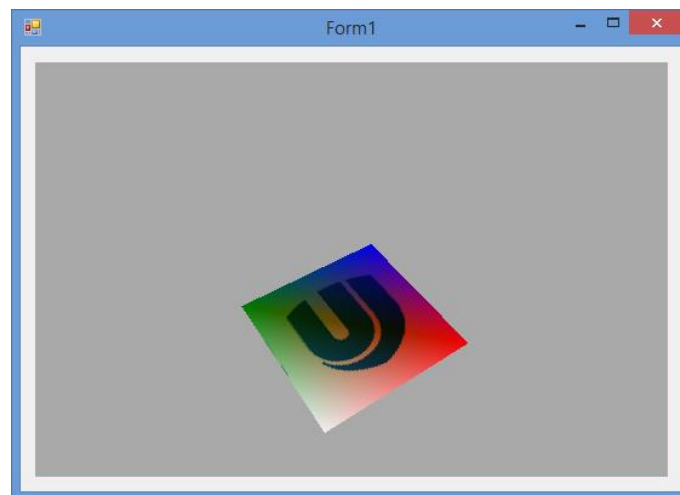
int texID = GLGraphics.LoadTexture("texture_logo.png");
GLGraphics.texturesIDs.Add(texID);

```

Создайте метод `drawTextureQuad` по аналогии с методом `drawtextquad`. Главным отличием является то, что в новой функции каждой вершине ставится в соответствие текстурная координата.

```
private void drawTexturedQuad()
{
    GL.Enable(EnableCap.Texture2D);
    GL.BindTexture(TextureTarget.Texture2D, texturesIDs[0]);
    GL.Begin(PrimitiveType.Quads);
    GL.Color3(Color.Blue);
    GL.TexCoord2(0.0, 0.0);
    GL.Vertex3(-1.0f, -1.0f, -1.0f);
    GL.Color3(Color.Red);
    GL.TexCoord2(0.0, 1.0);
    GL.Vertex3(-1.0f, 1.0f, -1.0f);
    GL.Color3(Color.White);
    GL.TexCoord2(1.0, 1.0);
    GL.Vertex3(1.0f, 1.0f, -1.0f);
    GL.Color3(Color.Green);
    GL.TexCoord2(1.0, 0.0);
    GL.Vertex3(1.0f, -1.0f, -1.0f);
    GL.End();
    GL.Disable(EnableCap.Texture2D);
}
```

Вызовите функцию, рисующую текстурированный квадрат, в функции Render. Проверьте правильность отрисовки.



7. ОСВЕЩЕНИЕ В OPENGL

Настройку освещения в OpenGL можно условно разделить на две части: настройка источников света и настройка материала освещаемых объектов. В данной лабораторной работе для всех объектов будет использоваться один и тот же материал для упрощения работы.

В OpenGL реализовано моделирование освещения точечными источниками света (до 8 источников света).

В классе GLGraphics создайте метод SetupLightning. Внутри метода включите расчет освещения, включите нулевой источник света, включите освещение, включите освещение освещения цветных вершин.

```
GL.Enable(EnableCap.Lighting);
GL.Enable(EnableCap.Light0);
GL.Enable(EnableCap.ColorMaterial);
```

Установите положение источника света.

```
Vector4 lightPosition = new Vector4(1.0f, 1.0f, 4.0f, 0.0f);
GL.Light(LightName.Light0, LightParameter.Position, lightPosition);
```

Установите ambient цвет источника – цвет, который будет иметь объект, не освещенный источником.

```
Vector4 ambientColor = new Vector4(0.2f, 0.2f, 0.2f, 1.0f);
GL.Light(LightName.Light0, LightParameter.Ambient, ambientColor);
```

Установите diffuse цвет источника – цвет, который будет иметь объект, освещенный источником.

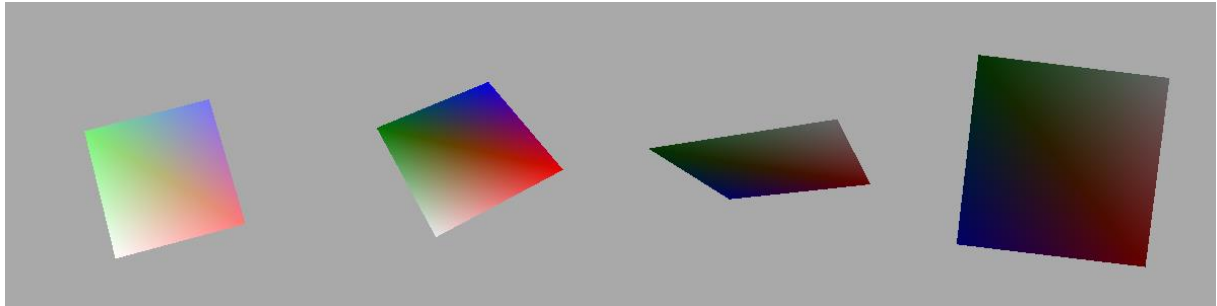
```
Vector4 diffuseColor = new Vector4(0.6f, 0.6f, 0.6f, 1.0f);
```

```
GL.Light(LightName.Light0, LightParameter.Diffuse, diffuseColor);
```

Для создания бликов на поверхностях установите всем материалам зеркальную составляющую. Из-за большого значения диффузной составляющей зеркальную не будет видно до тех пор, пока она не примет очень большое значение. В данном пособии Specular компонента будет включена для всех объектов, но в настоящих проектах specular составляющая устанавливается для каждого материала отдельно.

```
Vector4 materialSpecular = new Vector4(1.0f, 1.0f, 1.0f, 1.0f);
GL.Material(MaterialFace.Front, MaterialParameter.Specular, materialSpecular);
float materialShininess = 100;
GL.Material(MaterialFace.Front, MaterialParameter.Shininess, materialShininess);
```

Вызовите созданную функцию SetupLightning в методе Resize, посмотрите на цветной квадрат под разными углами. Квадрат с освещенной стороны должен изменять свой цвет в зависимости от угла просмотра, а с неосвещенной стороны быть темным и не менять свой цвет в зависимости от угла.



8. РИСОВАНИЕ ОБЪЕКТА С БОЛЬШИМ ЧИСЛОМ ПОЛИГОНОВ

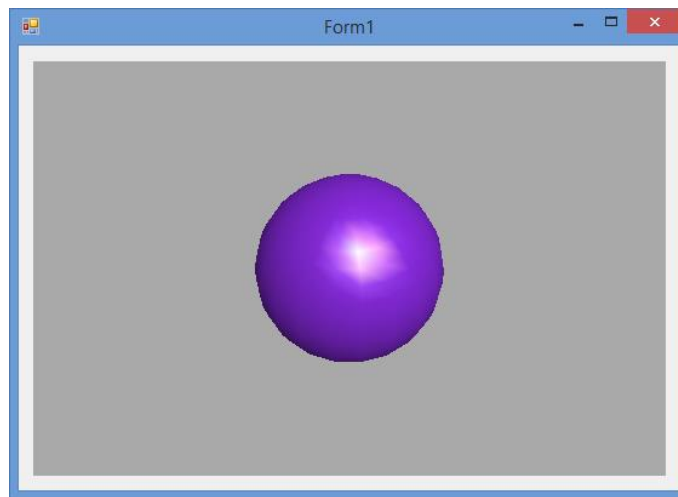
Для того, чтобы нарисовать сферу, создайте функцию DrawSphere. В приведенном ниже коде рассматривается создание сферы из четырехугольников подобно тому, как поверхность глобуса разбита на четырехугольники линиями широты и долготы. Количество четырехугольников по ширине и высоте задается параметрами nx и ny. На сфере единичного радиуса значение нормали в точке совпадает с координатой точки, что очень удобно.

```
private void DrawSphere(double r, int nx, int ny)
{
    int ix, iy;
    double x, y, z;
    for (iy = 0; iy < ny; ++iy)
    {
        GL.Begin(BeginMode.QuadStrip);
        for (ix = 0; ix <= nx; ++ix)
        {
            x = r * Math.Sin(iy * Math.PI / ny) * Math.Cos(2 * ix * Math.PI / nx);
            y = r * Math.Sin(iy * Math.PI / ny) * Math.Sin(2 * ix * Math.PI / nx);
            z = r * Math.Cos(iy * Math.PI / ny);
            GL.Normal3(x, y, z);
            GL.Vertex3(x, y, z);

            x = r * Math.Sin((iy + 1) * Math.PI / ny) * Math.Cos(2 * ix * Math.PI / nx);
            y = r * Math.Sin((iy + 1) * Math.PI / ny) * Math.Sin(2 * ix * Math.PI / nx);
            z = r * Math.Cos((iy + 1) * Math.PI / ny);
            GL.Normal3(x, y, z);
            GL.Vertex3(x, y, z);
        }
        GL.End();
    }
}
```

Вызовите созданную функцию в методе Render, предварительно установив сфере цвет. Посмотрите на сферу под разными углами с включенным освещением и бликами.

```
GL.Color3(Color.BlueViolet);
drawSphere(1.0f, 20, 20);
```



9. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

Применив полученные знания, выполните следующие задания.

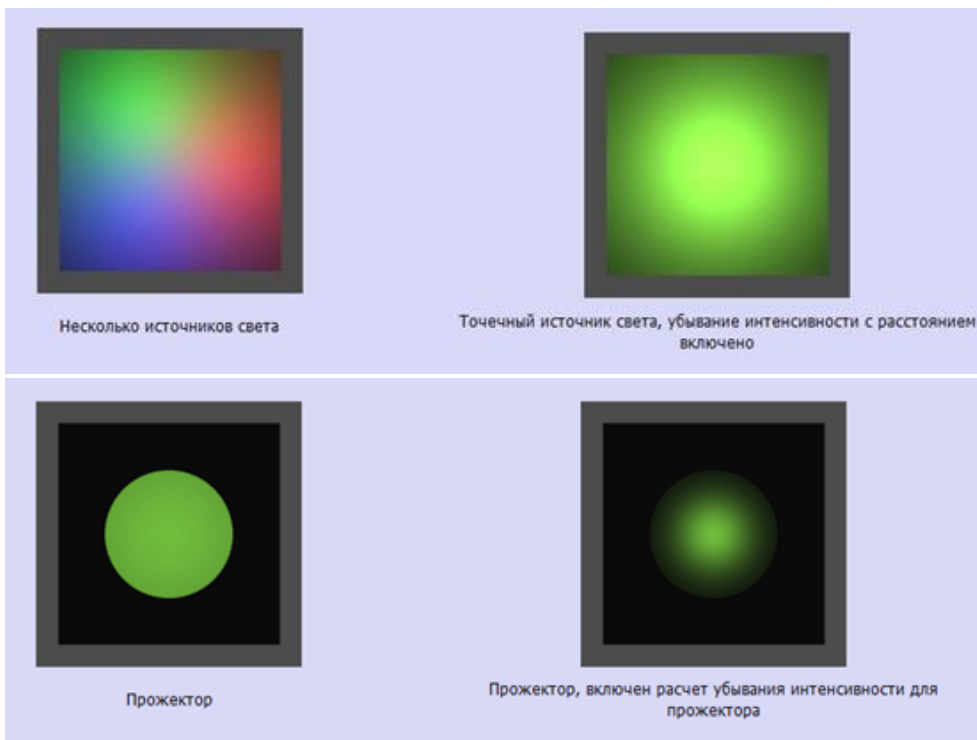
- Создайте остальные графические примитивы: Point, Line, Triangle, TriangleStrip, TriangleFan и др.
- Создайте функцию DrawCube, которая будет рисовать куб с 6 гранями, наложите на разные грани разные текстуры.
- Измените код так, чтобы объект двигался по круговой траектории.
- Создайте два источника света разных цветов, рядом с какой-либо поверхностью, чтобы увидеть, как происходит смешивание освещений.

10. ДОПОЛНИТЕЛЬНЫЕ ЗАДАНИЯ

Для дополнительного развития навыков программирования, выполните следующие задания.

- Добавьте возможность изменения дальности камеры от начала координат при помощи клавиш клавиатуры или колесика мыши.
- Измените код так, чтобы объект двигался по закону плоского математического маятника, либо пружинного маятника.
- Добавьте в код сферы создание текстурных координат у вершин, и наложите на сферу текстуру.
- Реализуйте различные типы освещения: реализация различных типов освещения на языке C++ и C# можно изучить по ссылкам[3,4].





11. ССЫЛКИ

1. <http://www.opentk.com/>
2. <http://opengl-tutorial.blogspot.ru/p/3.html>
3. http://esate.ru/uroki/OpenGL/uroki_opengl_p4077/
4. <http://www.intuit.ru/studies/courses/2313/613/lecture/13305>