**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский государственный авиационный технический университет"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина: Компьютерная графика**

**Отчет по лабораторной работе № 3**

**Тема: «Формирование моделей объемных объектов в виде тел вращения в OpenGL»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМИ-248 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Фаизова А.Н. |  |  |  |
| Принял | Мухтаров А.Р. |  |  |  |

**Уфа 2018**

**Цель:** изучив методические указания по построению и визуализации тел вращения в OpenGl, разработать приложение, визуализирующее одно из тел вращения для своего варианта задания, реализовать управление движением объекта в пространстве при помощи мыши.

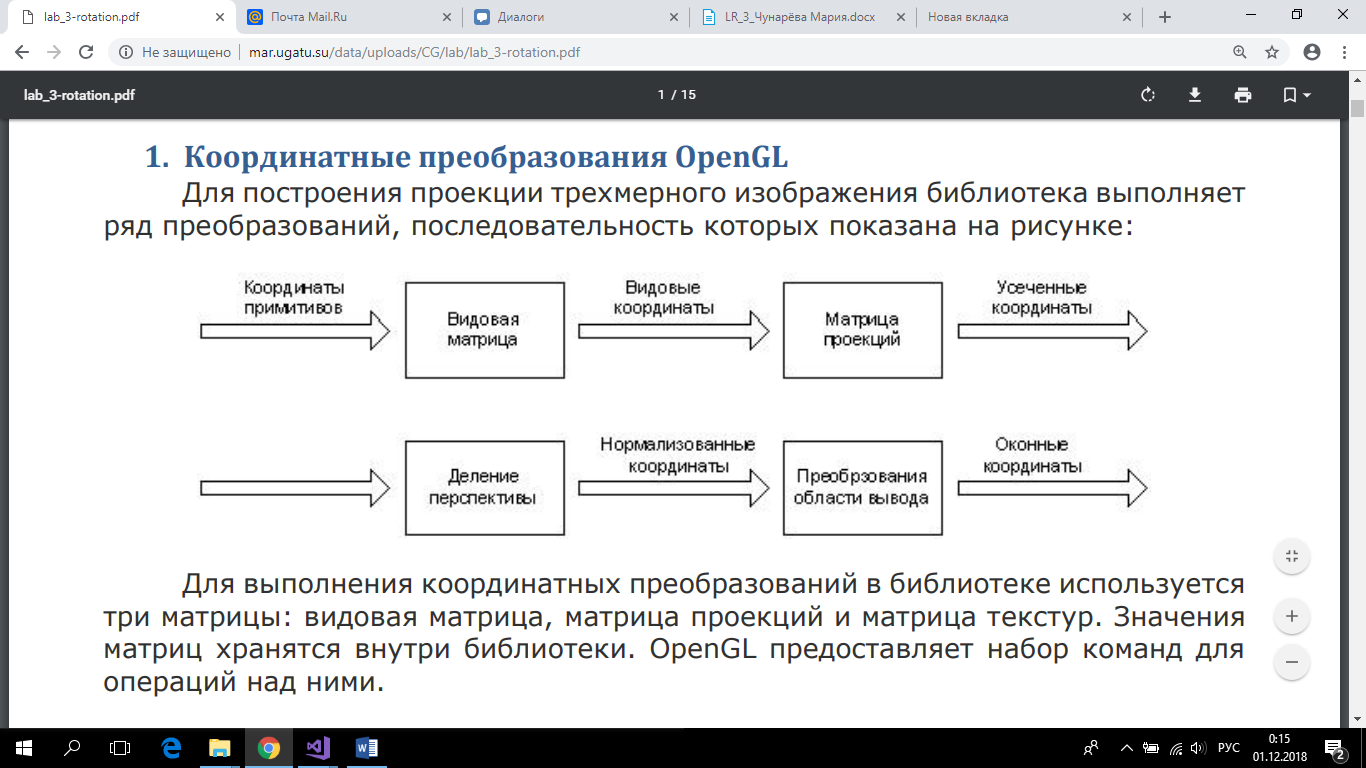
**Индивидуальное задание**

Разработайте приложение, визуализирующее тело вращения - шар, реализуйте управление движением объекта в пространстве при помощи мыши следующим образом:

* вращение фигуры производится только при наведении указателя мыши на окно, при уводе указателя за пределы окна движение прекращается;
* по клику мыши в области окна вращение фигуры переключается на противоположное;
* добавьте в форму ползунок, отвечающий за изменение скорости вращения фигуры.

**Описание программы:**

Для построения проекции трехмерного изображения библиотека выполняет ряд преобразований, последовательность которых показана на рисунке:



Для выполнения координатных преобразований в библиотеке используется три матрицы: видовая матрица, матрица проекций и матрица текстур. Значения матриц хранятся внутри библиотеки. OpenGL предоставляет набор команд для операций над ними.

Команды работы с матрицами

*Выбор текущей матрицы*

OpenGL реализует несколько матриц, содержание которых хранится в самой библиотеке. Получить доступ в каждый конкретный момент времени можно только к одной из этих матриц. Матрица, с которой будут выполняться те или иные преобразования, должна быть предварительно выбрана в качестве текущей с помощью команды MatrixMode. Выбранная матрица остается текущей до следующего вызова команды MatrixMode

void MatrixMode(MatrixMode mode);

*Изменение текущей матрицы на единичную матрицу*

void LoadIdentity();

Для замены текущей матрицы (которая выбрана с помощью команды MatrixMode) на единичную матрицу используется команда LoadIdentity. Команда не имеет параметров.

*Замена текущей матрицы на произвольную*

Для замены текущей матрицы OpenGL (которая выбрана с помощью команды MatrixMode) на произвольную матрицу используется команда LoadMatrix[fd]. Команда существует в двух вариантах: для матрицы вещественных значений одинарной точности и двойной точности.

*Параллельный перенос*

Для выполнения параллельного переноса библиотека реализует команду Translate[f d]. Команда имеет три параметра, определяющие координаты вектора параллельного переноса. Команда Translate формирует матрицу параллельного переноса и умножает её на текущую матрицу OpenGL.

*Масштабирование относительно начала координат*

Для выполнения операции масштабирования относительно начала координат используется команда Scale[f d]. Команда имеет три параметра, которые определяют коэффициенты масштабирования для каждой из осей координат.

*Поворот вокруг произвольной оси в пространстве*

Преобразование поворота вокруг произвольной оси в пространстве может быть выполнено с помощью команды Rotate[f d].

*Проекционные преобразования*

Проекционные преобразования определяются матрицей проекции и выполняются после видовых преобразований. Матрица проекции выполняет несколько функций:

1. Ограничивает область видимости объемной сцены. Для этого в командах Ortho и Frustum в качестве параметров передаются три пары координат для трех осей координат. С ограничением области вывода неразрывно связана операция усечения координат. Результатом этого преобразования является преобразование координат вершин видимой области сцены для каждой из осей координат, таким образом, чтобы они находились в интервале от –1 до 1.

2. Определяет перспективные преобразования.

3. Выполняет преобразование мировой (правосторонней) системы координат в видовую (левостороннюю) систему координат. Преобразование заключается в изменении направления оси OZ на противоположное.

*Параллельная проекция*

Для формирования матрицы параллельной проекции используется команда

Ortho(l, r, b, t, n, f: GLdouble)

Команда имеет шесть параметров. Каждая пара параметров определяет координаты границ области видимости для каждой из осей координат:

l, r – левая и правая границы области видимости вдоль оси OX;

b, t – нижняя и верхняя границы области видимости вдоль оси OY;

n, f – ближняя и дальняя границы области видимости вдоль оси OZ.

Фактически параметры команды определяют расположение граничных плоскостей, ограничивающих видимую область пространства.

*Перспективная проекция*

Для задания центральной перспективной проекции в OpenGL используется команда

Frustum: Frustum (l, r, b, t, n, f: GLdouble);

Команда Frustum имеет шесть параметров, которые определяют граничные координаты области видимости:

• l, r – левая и правая границы области видимости вдоль оси OX;

• b, t – нижняя и верхняя границы области видимости вдоль оси OY;

• n, f – ближняя и дальняя границы области видимости вдоль оси OZ.

Ближняя граница оси OZ (параметр n) так же определяет положение плоскости проекции, которая располагается перпендикулярно оси OZ. Точка наблюдения находится в начале координат. Видимый объем представляет собой усеченную пирамиду, проекция видимой области формируется на плоскости усечения, перпендикулярной оси OZ. Значения l, r, b, t задаются для ближней к наблюдателю плоскости отсечения. Значения n и f должны быть положительными.

*Определение области вывода*

На последнем этапе координатных преобразований вычисляются координаты области вывода. Координаты области вывода определяются с помощью команды:

Viewport (x,y: GLint; w, h: GLsizei);

Команда имеет четыре параметра:

• x, y – координаты верхнего левого угла области вывода;

• w, h – ширина и высота области вывода.

По ходу работы производим построение массива геометрии тела, построенного вращением на основе заданного заранее массива GeometricArray.

Когда геометрия объекта построена, обрабатываем сообщение таймера для вызова функции отрисовки, а также реализовываем непосредственно функцию Draw. В функции Draw производим 3 вида визуализации, которые будут использованы в зависимости от установленного режима в элементе comboBox: с помощью точек, линий или полигонов.

Исходный код программы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing.Imaging;

using OpenTK;

using OpenTK.Graphics;

using OpenTK.Graphics.OpenGL;

namespace LabWork\_3

{

public partial class Form1 : Form

{

private float rot\_1, rot\_2;

private double[,] GeometricArray = new double[64, 3];

private double[,,] ResaultGeometric = new double[64, 64, 3];

private int count\_elements = 0;

private double Angle = 2 \* Math.PI / 64;

private int Inter = 64;

private int R = 10;

bool invert = false, rotate =true;

private void AnT\_Load(object sender, EventArgs e)

{

//очистка окна

GL.ClearColor(Color.Aqua);

//очистка буферов цвета и глубины

GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

//установка порта вывода в соответствии с размерами элемнта AnT

GL.Viewport(0, 0, AnT.Width, AnT.Height);

//настройка проекции

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

GL.Frustum(-0.1 \* AnT.Width / AnT.Height, 0.1 \* AnT.Width / AnT.Height, -0.1, 0.1, 0.1, 200);

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadIdentity();

//настройка параметров OpenGL для визуализации

GL.Enable(EnableCap.DepthTest);

GL.Enable(EnableCap.Lighting);

GL.Enable(EnableCap.Light0);

count\_elements = 21;

for (int i = 0; i < count\_elements; i++)

{

GeometricArray[i, 0] = R \* Math.Sin(i \* 9 \* Math.PI / 180);

GeometricArray[i, 1] = 0;

GeometricArray[i, 2] = R \* Math.Cos(i \* 9 \* Math.PI / 180);

}

//по умолчанию мы будем отрисовывать фигуру в режиме GL.POINTS

comboBox1.SelectedIndex = 0;

//построение геометрии тел вращения

//принцип сводится к двум циклам: на основе первого перебираются вершины в геометрической последовательности,

//а второй использует параметр Iter и производит поворот последней линии геометри вокруг центра тела вращения

//при этом используется заранее опрееделнный угол angle, который определяется как 2\*Pi / количество медиан объекта

//за счёт выполнения этого алгоритма получается набор вершин, описывающих оболочку тела вращения

//остаётся только соединить эти точки в режиме рисования примитивов для получения

//визуализированного объекта

//цикл по последовательности точек кривой, на основе которой будет построено тело вращения

for (int ax = 0; ax < count\_elements; ax++)

{

for (int bx = 0; bx < Inter; bx++)

{

if (bx>0)

{

double new\_x = ResaultGeometric[ax, bx - 1, 0] \* Math.Cos(Angle) - ResaultGeometric[ax, bx - 1, 1] \* Math.Sin(Angle);

double new\_y = ResaultGeometric[ax, bx - 1, 0] \* Math.Sin(Angle) + ResaultGeometric[ax, bx - 1, 1] \* Math.Cos(Angle);

ResaultGeometric[ax, bx, 0] = new\_x;

ResaultGeometric[ax, bx, 1] = new\_y;

ResaultGeometric[ax, bx, 2] = GeometricArray[ax, 2];

}

else

//для построения первой медианы мы используем начальную кривую, описывая ее нулевым значение угла поворота

{

double new\_x = GeometricArray[ax, 0] \* Math.Cos(0) - GeometricArray[ax, 1] \* Math.Sin(0);

double new\_y = GeometricArray[ax, 1] \* Math.Sin(0) + GeometricArray[ax, 1] \* Math.Cos(0);

ResaultGeometric[ax, bx, 0] = new\_x;

ResaultGeometric[ax, bx, 1] = new\_y;

ResaultGeometric[ax, bx, 2] = GeometricArray[ax, 2];

}

}

}

RenderTimer.Start();

}

private void Draw()

{

if (invert)

{

if(rotate)

{//два параметра, которые будем использовать для непрерывного вращения сцены вокруг 2 координратных осей

rot\_1 += trackBar2.Value / 2;

rot\_2 += trackBar2.Value / 2;

}

else

{

rot\_1 -= trackBar2.Value / 2;

rot\_2 -= trackBar2.Value / 2;

}

}

//очистка буфера цвета и буфера глубины

GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

GL.ClearColor(Color.White);

//очищение текущей мтарицы

GL.LoadIdentity();

//установка положения камеры (наблюдателя)

//при перемещении ползунка, наблюдатель будет отдаляться или приближаться к объекту наблюдения

GL.Translate(0, 0, -1 - trackBar1.Value);

// RenderTimer.Interval = trackBar2.Value;

//2 поворота

if (k%2==0 )

{

GL.Rotate(rot\_1, 1, 0, 0);

GL.Rotate(rot\_2, 0, 1, 0);

}else

{

GL.Rotate(-rot\_1, 1, 0, 0);

GL.Rotate(-rot\_2, 0, 1, 0);

}

GL.PointSize(5.0f);

switch (comboBox1.SelectedIndex)

{

case 0:

{

GL.Begin(PrimitiveType.Points);

for (int ax=0; ax<count\_elements;ax++)

{

for (int bx=0; bx<Inter; bx++)

{

//отрисовка точки

GL.Vertex3(ResaultGeometric[ax, bx, 0], ResaultGeometric[ax, bx, 1], ResaultGeometric[ax, bx, 2]);

}

}

GL.End();

break;

}

case 1:

//отображение объекта в сеточном режиме, используя режим GL\_LINES\_STRIP

{

//устанавливаем режим отрисовки линиями (последовательность линий)

GL.Begin(PrimitiveType.LineStrip);

for (int ax = 0; ax < count\_elements; ax++)

{

for (int bx = 0; bx < Inter; bx++)

{

GL.Vertex3(ResaultGeometric[ax, bx, 0], ResaultGeometric[ax, bx, 1], ResaultGeometric[ax, bx, 2]);

GL.Vertex3(ResaultGeometric[ax+1, bx, 0], ResaultGeometric[ax+1, bx, 1], ResaultGeometric[ax+1, bx, 2]);

if (bx+1<Inter-1)

{

GL.Vertex3(ResaultGeometric[ax + 1, bx + 1, 0], ResaultGeometric[ax + 1, bx + 1, 1], ResaultGeometric[ax + 1, bx + 1, 2]);

}

else

{

GL.Vertex3(ResaultGeometric[ax + 1, 0, 0], ResaultGeometric[ax + 1, 0, 1], ResaultGeometric[ax + 1, 0, 2]);

}

}

}

GL.End();

break;

}

case 2:

//отрисовка оболочки с расчетом нормалей для корретного затенения граней объекта

{

GL.Begin(PrimitiveType.Quads);

//режим отрисовки полигонов, состоящих из 4 вершин

for (int ax = 0; ax < count\_elements; ax++)

{

for (int bx = 0; bx < Inter; bx++)

{

double x1 = 0, x2 = 0, x3 = 0, x4 = 0,

y1 = 0, y2 = 0, y3 = 0, y4 = 0, z1 = 0, z2 = 0, z3 = 0, z4 = 0;

x1 = ResaultGeometric[ax, bx, 0];

y1= ResaultGeometric[ax, bx, 1];

z1= ResaultGeometric[ax, bx, 2];

if (ax+1<count\_elements)

//если текущий ax не последний

{

x2 = ResaultGeometric[ax + 1, bx, 0];

y2 = ResaultGeometric[ax + 1, bx, 1];

z2 = ResaultGeometric[ax + 1, bx, 2];

if (bx+1<Inter-1)

//если текущий bx не последний

{

x3 = ResaultGeometric[ax + 1, bx+1, 0];

y3 = ResaultGeometric[ax + 1, bx + 1, 1];

z3 = ResaultGeometric[ax + 1, bx + 1, 2];

x4 = ResaultGeometric[ax, bx + 1, 0];

y4 = ResaultGeometric[ax, bx + 1, 1];

z4 = ResaultGeometric[ax, bx + 1, 2];

}

else

{

//если это последний медиан, то в качестве следующего мы берем начальный

x3 = ResaultGeometric[ax + 1, 0, 0];

y3 = ResaultGeometric[ax + 1, 0, 1];

z3 = ResaultGeometric[ax + 1, 0, 2];

x4 = ResaultGeometric[ax, 0, 0];

y4 = ResaultGeometric[ax, 0, 1];

z4 = ResaultGeometric[ax, 0, 2];

}

}

else

{

//если данный элемент ax последний, будем использовтаь начальный вместо ах

x2 = ResaultGeometric[0, bx, 0];

y2 = ResaultGeometric[0, bx, 1];

z2 = ResaultGeometric[0, bx, 2];

if (bx + 1 < Inter - 1)

//если текущий bx не последний

{

x3 = ResaultGeometric[0, bx + 1, 0];

y3 = ResaultGeometric[0, bx + 1, 1];

z3 = ResaultGeometric[0, bx + 1, 2];

x4 = ResaultGeometric[ax, bx + 1, 0];

y4 = ResaultGeometric[ax, bx + 1, 1];

z4 = ResaultGeometric[ax, bx + 1, 2];

}

else

{

x3 = ResaultGeometric[0, 0, 0];

y3 = ResaultGeometric[0, 0, 1];

z3 = ResaultGeometric[0, 0, 2];

x4 = ResaultGeometric[ax, 0, 0];

y4 = ResaultGeometric[ax, 0, 1];

z4 = ResaultGeometric[ax, 0, 2];

}

}

//перменные для расчёта нормали

double n1 = 0, n2 = 0, n3 = 0;

//нормаль будем рассчитывать как векторное произведение граней полигона

//для нулевого элемента нормаль мы будем считать немного по-другому

//на самом деле разница в расчета нормали актуальна только для первого и последнего полигона на медиане

if (ax==0)

//при расчете нормали для ах мы будем использовать точки 1, 2, 3

{

n1 = (y2 - y1) \* (z3 - z1) - (y3 - y1) \* (z2 - z1);

n2 = (z2 - z1) \* (x3 - x1) - (z3 - z1) \* (x2 - x1);

n3 = (x2 - x1) \* (y3 - y1) - (x3 - x1) \* (y2 - y1);

}

else

//для остальных - 1, 2, 3

{

n1 = (y4 - y3) \* (z1 - z3) - (y1 - y3) \* (z4 - z3);

n2 = (z4 - z3) \* (x1 - x3) - (z1 - z3) \* (x4 - x3);

n3 = (x4 - x3) \* (y1 - y3) - (x1 - x3) \* (y4 - y3);

}

//если не включен режим NORMOLIZE, то м ыдолжны в обязательном порядке

//произвести нормализацию вектора перед тем как передать информацию о нормали

double n5 = (double)Math.Sqrt(n1 \* n1 + n2 \* n2 + n3 \* n3);

n1 /= (n5 + 0.01);

n2 /= (n5 + 0.01);

n3 /= (n5 + 0.01);

// передаем информацию о нормали

GL.Normal3(n1, n2, n3);

//передаем 4 вершины для отрисовки полигона

GL.Vertex3(x1, y1, z1);

GL.Vertex3(x2, y2, z2);

GL.Vertex3(x3, y3, z3);

GL.Vertex3(x4, y4, z4);

}

}

GL.End();

break;

}

}

//возвращаем сохраненную матрицу

GL.PopMatrix();

//завершаем рисование

GL.Flush();

//обновляем элемент AnT

AnT.SwapBuffers();

}

private void RenderTimer\_Tick(object sender, EventArgs e)

{

Draw();

}

int k;

private void AnT\_Click(object sender, EventArgs e)

{

k++;

}

private void AnT\_MouseEnter(object sender, EventArgs e)

{

invert = !invert;

}

private void AnT\_MouseLeave(object sender, EventArgs e)

{

invert = !invert;

}

public Form1()

{

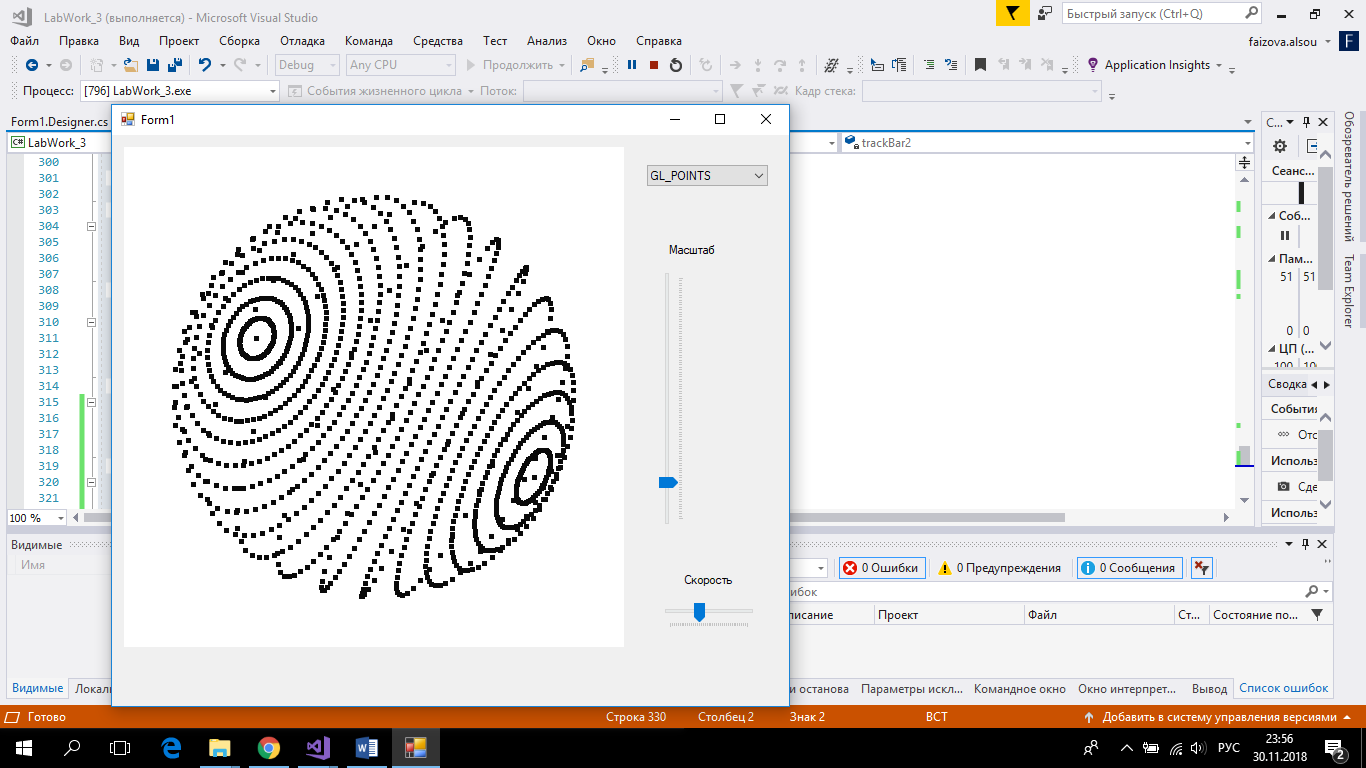
InitializeComponent();

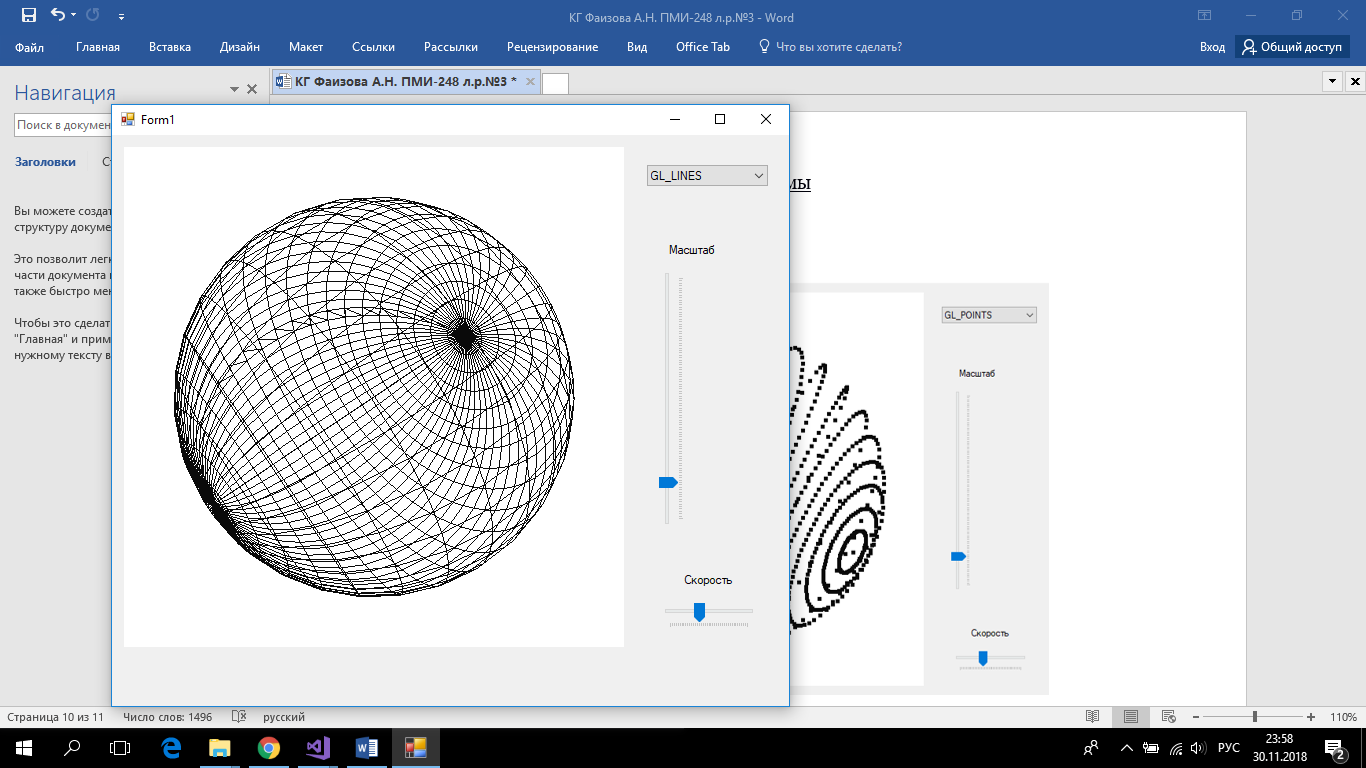
}

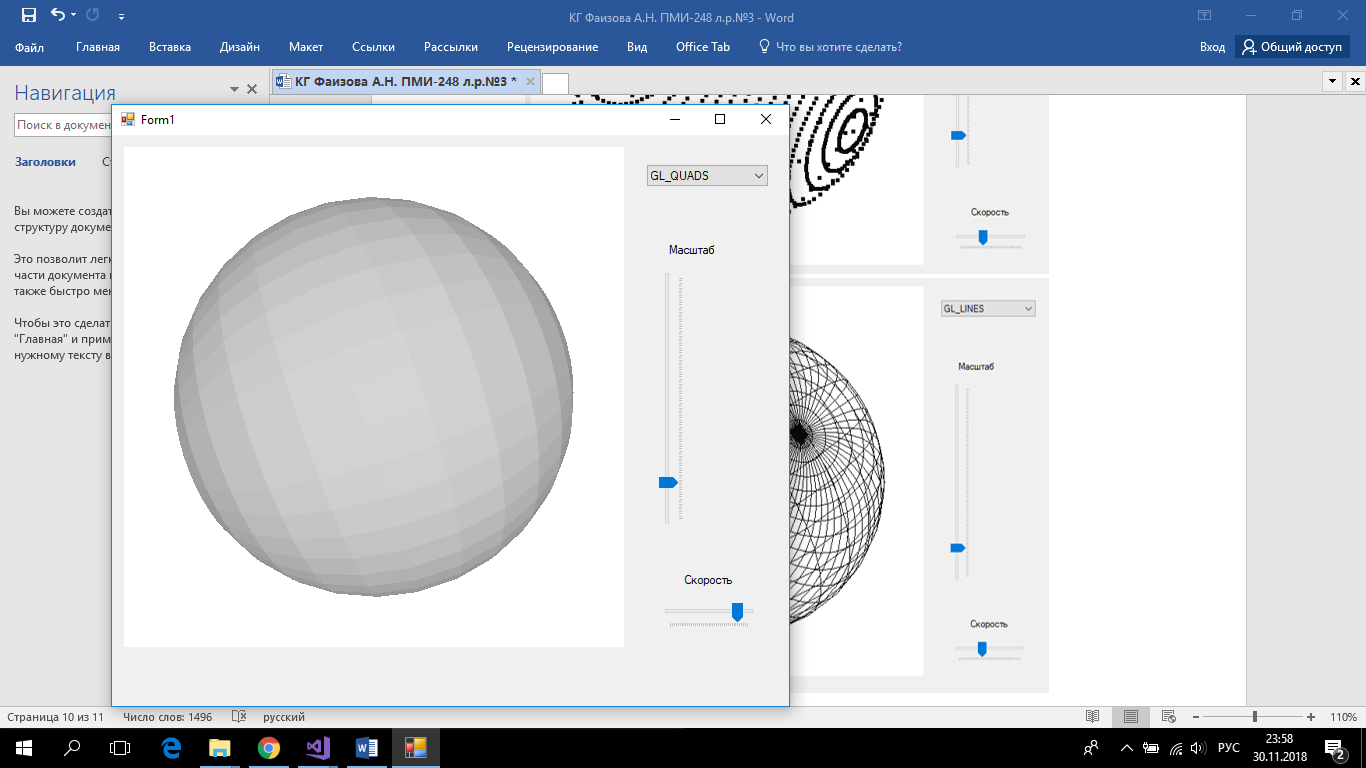
}

}

Пример выполнения программы







**Вывод:**