**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 № 4**

**Тема: «Визуализация функционально заданных поверхностей в OpenGL»**

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

**Уфа 2018**

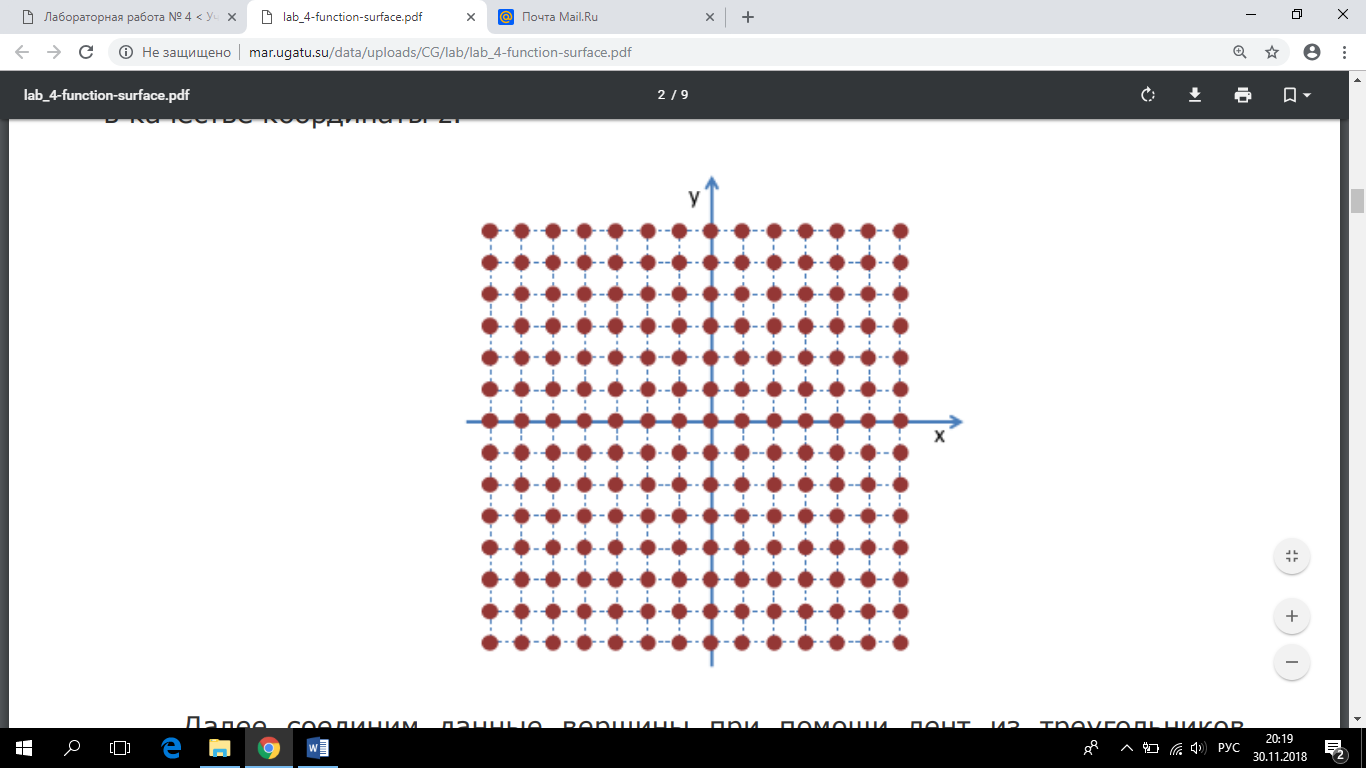
**Цель:** изучив методические указания по построению и визуализации функционально заданных поверхностей в OpenGl, разработать приложение, визуализирующее трехмерную, функционально заданную поверхность, соответствующую одному из заданных вариантов.

**Индивидуальное задание:**

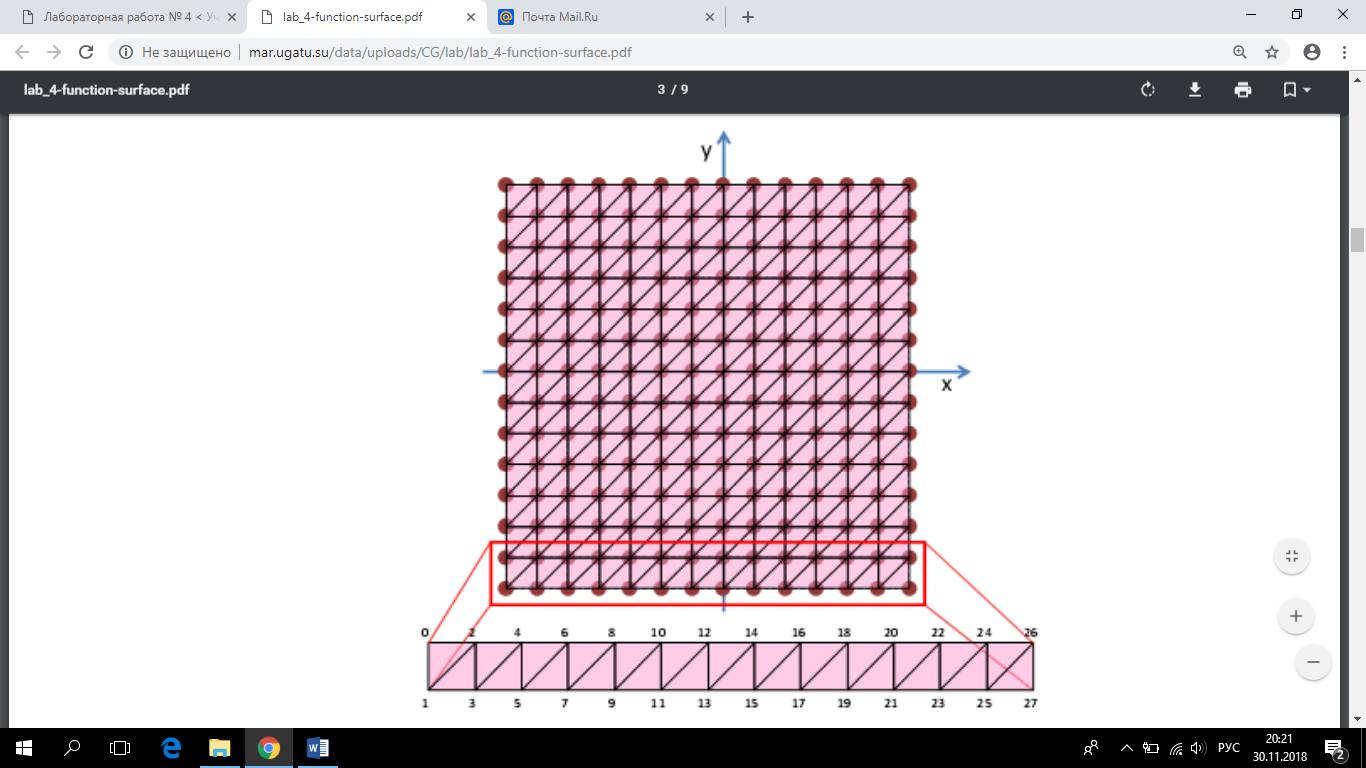
Разработать приложение, визуализирующее трехмерную, функционально заданную поверхность - эллиптический параболоид. Поверхность должна быть визуализирована в каркасном виде и с помощью сплошной закраски аппроксимирующих поверхность граней. Для поверхностей, имеющих коэффициенты (a,b,c) необходимо предусмотреть возможность их изменения в интерфейсе программы.

**Описание программы:**

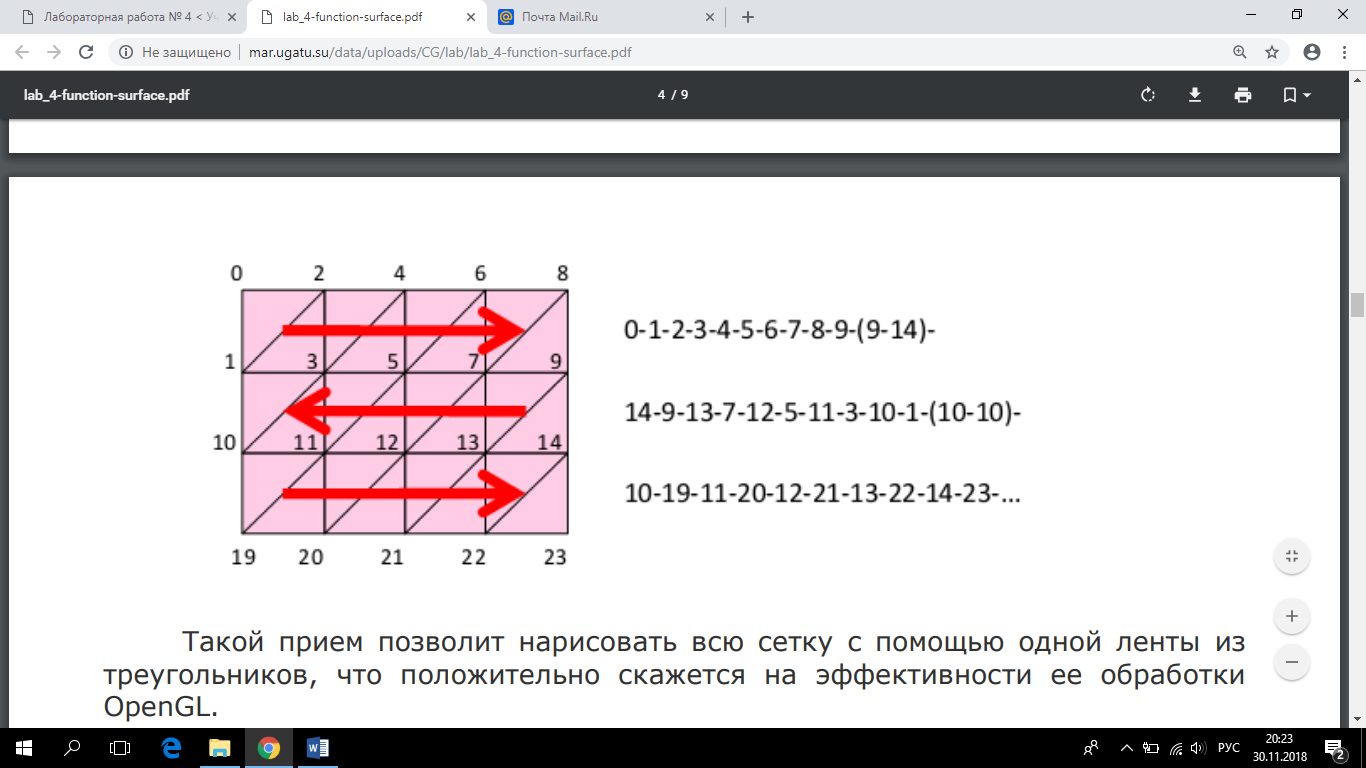
В OpenGL отсутствуют средства визуализации кривых линий и поверхностей, поэтому нам придется аппроксимировать их с помощью отрезков прямых линий и треугольников. Для начала, равномерно разобьем отображаемую область функции на область N\*M ячеек и вычислим значение функции в узлах сетки и используем ее в качестве координаты z.



Далее соединим данные вершины при помощи лент из треугольников (TRIANGLE\_STRIP):



Для сокращения количества групп примитивов, можно «сшивать» соседние ленты треугольников, добавляя пару дополнительных вершин в конце каждой ленты для образования пары вырожденных треугольных граней для смены направления обхода рядов ленты:



Такой прием позволит нарисовать всю сетку с помощью одной ленты из треугольников, что положительно скажется на эффективности ее обработки OpenGL.

Каждая вершина сетки будет характеризоваться координатами и нормалью. Для хранения данных параметров создадим структуру Vertex:

struct Vertex

{

public float x, y, z;

public float nx,ny,nz;

}

Создадим класс CSincSurface, который будет визуализировать трехмерный вариант функции.

Рассмотрим метод CalculateVertex. Сначала мы вычисляем координату z по формуле из задания. На этом можно было бы остановиться, но нам необходимо вычислить вектор нормали к поверхности в точке (x, y, z). Для поверхностей, заданных в виде направление вектора нормали будет совпадать с вектором антиградиента данной функции в точке (x, y, z). Вектор антиградиента – вектор, обратный вектору частных производных функции F. Вычислить вектор частных производных можно было бы аналитически, выполнив дифференцирование функции F по dx, dy и dz, однако вычисления получились бы достаточно громоздкими. В нашем случае нас устроит приблизительное (хотя и достаточно близкое) вычисление производной по ее определению: «Производная равна отношению приращения функции к приращению аргумента, при приращении аргумента стремящемуся к нулю». В нашем случае «стремящееся к нулю» число можно представить как просто «достаточно маленькое число», например, 10-6. Здесь нам пригодятся числа с плавающей запятой двойной точности. Метод рисования полигональной сетки, аппроксимирующей функционально заданную поверхность, воплощает ранее описанный подход к ее построению с помощью лент из треугольников. Для повышения быстродействия, все же, воспользуемся дисплейными списками OpenGL для записи и последующего воспроизведения команд рисования поверхности.

Для управления свойствами материала используем функцию SetupMaterial().

Для анимации в виде вращения поверхности используем функцию SetupCamera().

DrawSurface() -функция рисования поверхности.

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 OpenTK;

using OpenTK.Graphics;

using OpenTK.Graphics.OpenGL;

namespace WindowsFormsApp5

{

public partial class Form1 : Form

{

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

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

GL.ClearColor(Color.Yellow);

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();

}

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

{

// настройка параметров OpenGL для визуализации

GL.Enable(EnableCap.CullFace);

GL.CullFace(CullFaceMode.Back);// тип отображаемых граней

GL.FrontFace(FrontFaceDirection.Ccw);//значение по умолчанию

// запуск таймера

RenderTimer.Start();

comboBox1.SelectedIndex = 0;

}

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

{

DrawSurface();

AnT.Invalidate();

}

struct Vertex

{

public float x, y, z;

public float nx,ny,nz;

}

/\* Вычисляем z. Потом вычисляем нормаль к повехности в точке.

\* Направление вектора нормали будет сопадать с вектором антиградиента данной ф-ии в точке хуz.

\* Вычислить ветор антиградиента-вектор, обратный вктору частных производных функции-мб бы аналитически

\* В нашем случае-приблизительное вычисление производной по определению:

\* Произ-я равна отн-ю приращения ф-ии к приращению аргумента, при приращении аргумента -> 0

\* \*/

class CSinSurface

{

public double F(double x, double y,double z,double a,double b)

{

double f = (x \* x) / (a \* a) + (y \* y) / (b \* b);

return f - z;

}

public Vertex CalculateVertex(double x, double y, double a, double b)

{

Vertex resault = new Vertex();

// вычисляем значение координаты z

double r = (x \* x)/(a\*a) + (y \* y)/(b\*b);

double z = r;

double delta = 1e-6;

float f = (float)F(x, y, z, a, b);

// вычисляем приблизительное частные производные функции по dx, dy, dz

// их значения прибличительно равны координатам вектора

// нормали к поверхности в точке (x,y,z)

double dfdx = -(F(x + delta, y, z, a, b) - f) / delta;

double dfdy = -(F(x , y+delta, z, a, b) - f) / delta;

double dfdz = 1;

// личина обратная длине вектора антиградиента

double invLen = 1 / Math.Sqrt(dfdx \* dfdx + dfdy \* dfdy + dfdz \* dfdz);

// координаты вершины

resault.x = (float)x; resault.y = (float)y; resault.z = (float)z;

// приводим вектор нормали к единичной длине

resault.nx = (float)(dfdx \* invLen);

resault.ny = (float)(dfdy \* invLen);

resault.nz = (float)(dfdz \* invLen);

return resault;

}

}

private void SetupMaterial()

{

// задаем характеристики материала

// коэффициент диффузного отражения

float[] m\_diffuse = new float[] { 0.0f, 0.5f, 0.0f, 1 };

// коэффициент фонового отражения

float[] m\_ambient = new float[] { 0.0f, 0.2f, 0.0f, 1 };

// коэффициент зеркального отражения

float[] m\_specular = new float[] { 0.3f, 0.2f, 0.3f, 1 };

// степень зеркального отражения

float m\_shininess = 1;

// face- определяет тип граней, для к-х задается материал

// Diffuse параметр определяет диффузный цвет материала

GL.Material(MaterialFace.Front, MaterialParameter.Diffuse, m\_diffuse);

// Ambient параметр определяет рассеянный цвет материала (цвет материала в тени)

GL.Material(MaterialFace.Front, MaterialParameter.Ambient, m\_ambient);

// Specular параметр определяет зеркальный цвет материала

GL.Material(MaterialFace.Front, MaterialParameter.Specular, m\_specular);

// Shininess параметр определяет степень зеркального отражения материала

GL.Material(MaterialFace.Front, MaterialParameter.Shininess, m\_shininess);

}

// используем ф-ю SetupLight для управления источниками света

private void SetupLight()

{

//задаем характристики источника света

float[] lightDirection = new float[] { 2, 2, 2, 0 };

//устанавливаем характеристики источника света

float[] l\_diffuse = new float[] { 0.4f, 0, 3f, 1f, 1 };

float[] l\_ambient = new float[] { 0.2f, 0, 9f, 0f, 1 };

float[] l\_specular = new float[] { 0f, 0, 5f, 0.5f, 1 };

// Position параметр определяет положение источника света

GL.Light(LightName.Light0, LightParameter.Position, lightDirection);

// параметр определяет цвет диффузного освещения

GL.Light(LightName.Light0, LightParameter.Diffuse, l\_diffuse);

// Ambient параметр определяет цвет фонового освещения

GL.Light(LightName.Light0, LightParameter.Ambient, l\_ambient);

// Specular параметр определяет цвет зеркального отражения

GL.Light(LightName.Light0, LightParameter.Specular, l\_specular);

//включение освещения

GL.Enable(EnableCap.Lighting);

GL.Enable(EnableCap.Light0);

}

float rotationAngle = 0;

// используем ф-ю SetupCamera() для анимации в виде вращения поверхности

private void SetupCamera()

{

//скорость вращения

float rotationSpeed = 0.2f;

GL.LoadIdentity();

float cameraPositionX = 15;

float cameraPositionY = 15;

float cameraPositionZ = 13;

//вычисляем угол вращения

rotationAngle = (rotationAngle + rotationSpeed) % 360;

//устанавливаем камеру

Matrix4 lookat = Matrix4.LookAt(cameraPositionX, cameraPositionY, cameraPositionZ, 1, 1, 1, 0, 0, 1);

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadMatrix(ref lookat);

GL.Rotate(rotationAngle, 0, 0, 1);

}

private void DrawSurface()

{

//переменные для определения поверхности

float a = (float)numericUpDown1.Value;

float b = (float)numericUpDown1.Value;

int s\_displayList = 0;

int s\_columns = 80;

int s\_rows = 80;

float s\_xMin = -10;

float s\_xMax = 10;

float s\_yMin = -10;

float s\_yMax = 10;

float ScaleKof = 0.8f;

const float ZNEAR = 1f;

const float ZFAR = 40;

const float FIELD\_OF\_VIEW = 60;

float aspect = (float)AnT.Width/(float)AnT.Height;

CSinSurface pv = new CSinSurface();

//очистка окна

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

GL.ClearColor(0, 0, 0, 1);

SetupLight(); //настройки параметров освещения

SetupMaterial();// настройки свойства материала

SetupCamera(); // настройка камеры

//задания типа примитивов, используемых для отображения поверхности

switch ( comboBox1.SelectedIndex)

{

case 0:

GL.PolygonMode(MaterialFace.Front, PolygonMode.Point);

GL.PolygonMode(MaterialFace.Back, PolygonMode.Point);

GL.PointSize(1);

break;

case 1:

GL.PolygonMode(MaterialFace.Front, PolygonMode.Line);

GL.PolygonMode(MaterialFace.Back, PolygonMode.Line);

break;

case 2:

GL.PolygonMode(MaterialFace.Front, PolygonMode.Fill);

GL.PolygonMode(MaterialFace.Back, PolygonMode.Fill);

break;

}

//настройка проекция

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

// Матрица проекций отвечает за то, какой объём пространства будет визуализироваться,

// каким образом вершины графических примитивов будут спроецированы на двумерную поверхность экрана монитора.

// Преобразования матрицы проекций ведут к тому, что все изображение будет изменяться ( вращаться).

// При перспективной проекции используется тот факт, что человеческий глаз работает с предметом дальнего типа, размеры которого имеют угловые размеры.

// Чем дальше объект, тем меньше он нам кажется.Таким образом, объём пространства, который визуализируется представляет собой пирамиду

Matrix4 perspectiveMatrix = Matrix4.CreatePerspectiveFieldOfView(MathHelper.DegreesToRadians(FIELD\_OF\_VIEW), aspect, ZNEAR, ZFAR);

GL.LoadMatrix(ref perspectiveMatrix);

//FIELD\_OF\_VIEW - определяет поле зрения

//aspect -отношение ширины отсекающей рамки к высоте.

//znear и zfar - имеют всегда положительные значения и определяют расстояние от точки зрения до ближней и дальней отсекающих рамок

GL.MatrixMode(MatrixMode.Modelview);

GL.PushMatrix();

GL.Scale(ScaleKof, ScaleKof, ScaleKof);

//при первом обращении к данной функции запишем команды рисования поверхности в дисплейный список

if (s\_displayList == 0)

{

s\_displayList = GL.GenLists(1);

GL.NewList(s\_displayList, ListMode.Compile);

GL.Color3(255, 0, 0);

//вычисляем шаг узлов сетки

float dx = (s\_xMax - s\_xMin) / (s\_columns - 1);

float dy = (s\_yMax - s\_yMin) / (s\_rows - 1);

float y = s\_yMin;

//пробегаем по строкам сетки

for (int row = 0; row < s\_rows - 1; ++row, y += dy)

{

//каждой строке будет соответствовать своя лента из треугольников

GL.Begin(PrimitiveType.TriangleStrip);

float x = s\_xMin;

//пробегаем по столбцам текущей строки

for (int column = 0; column < s\_columns - 1; ++column, x += dx)

{

//вычисляем параметры вершины в узлах пары соседних вершин ленты из треугольников

Vertex v0 = pv.CalculateVertex(x, y + dy,a,b);

Vertex v1 = pv.CalculateVertex(x, y,a,b);

//задаем нормаль и координаты вершины на четной позиции

GL.Normal3(v0.nx, v0.ny, v0.nz);

GL.Vertex3(v0.x, v0.y, v0.z);

//задаем нормаль и координаты вершины на нечетной позиции

GL.Normal3(v1.nx, v1.ny, v1.nz);

GL.Vertex3(v1.x, v1.y, v1.z);

}

GL.End();

}

GL.EndList();

GL.CallList(s\_displayList);

}

//завершаем рисование

GL.Flush();

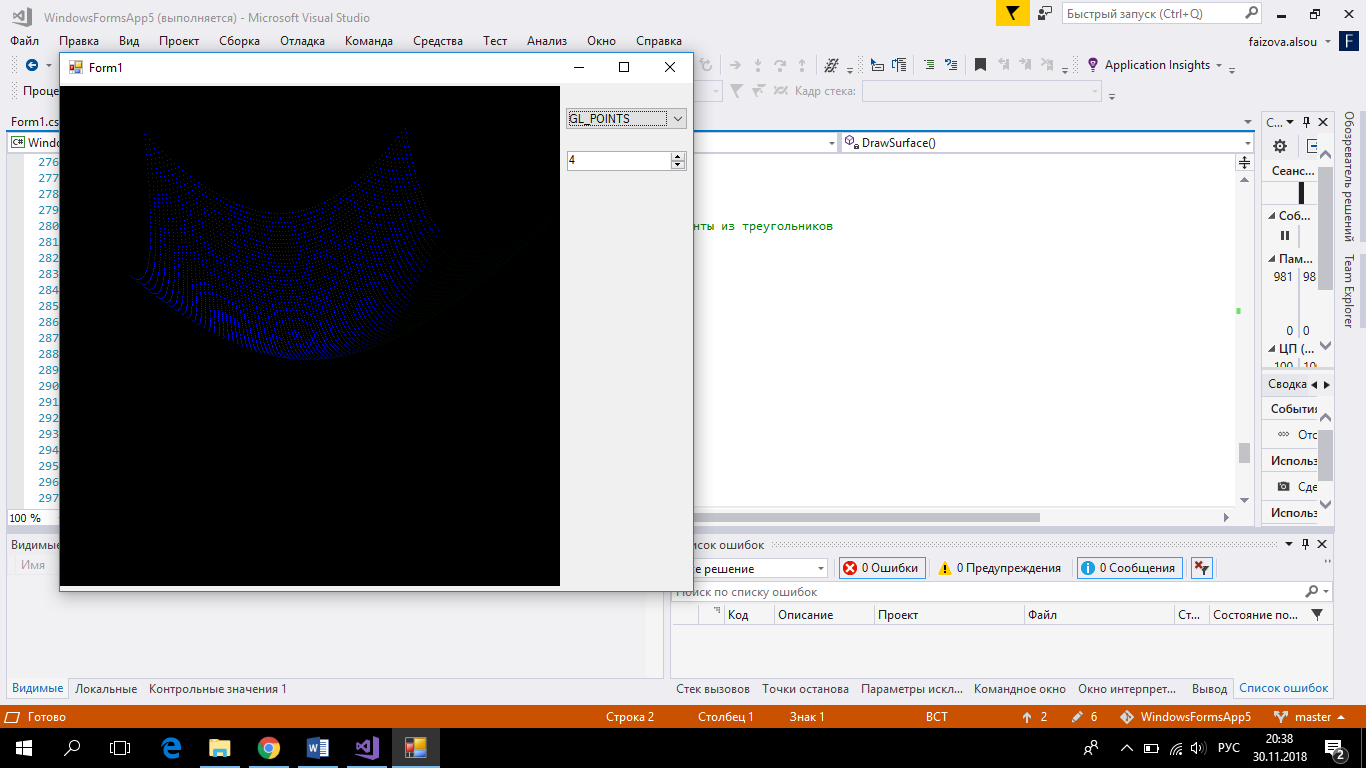
//обновляем элемент AnT

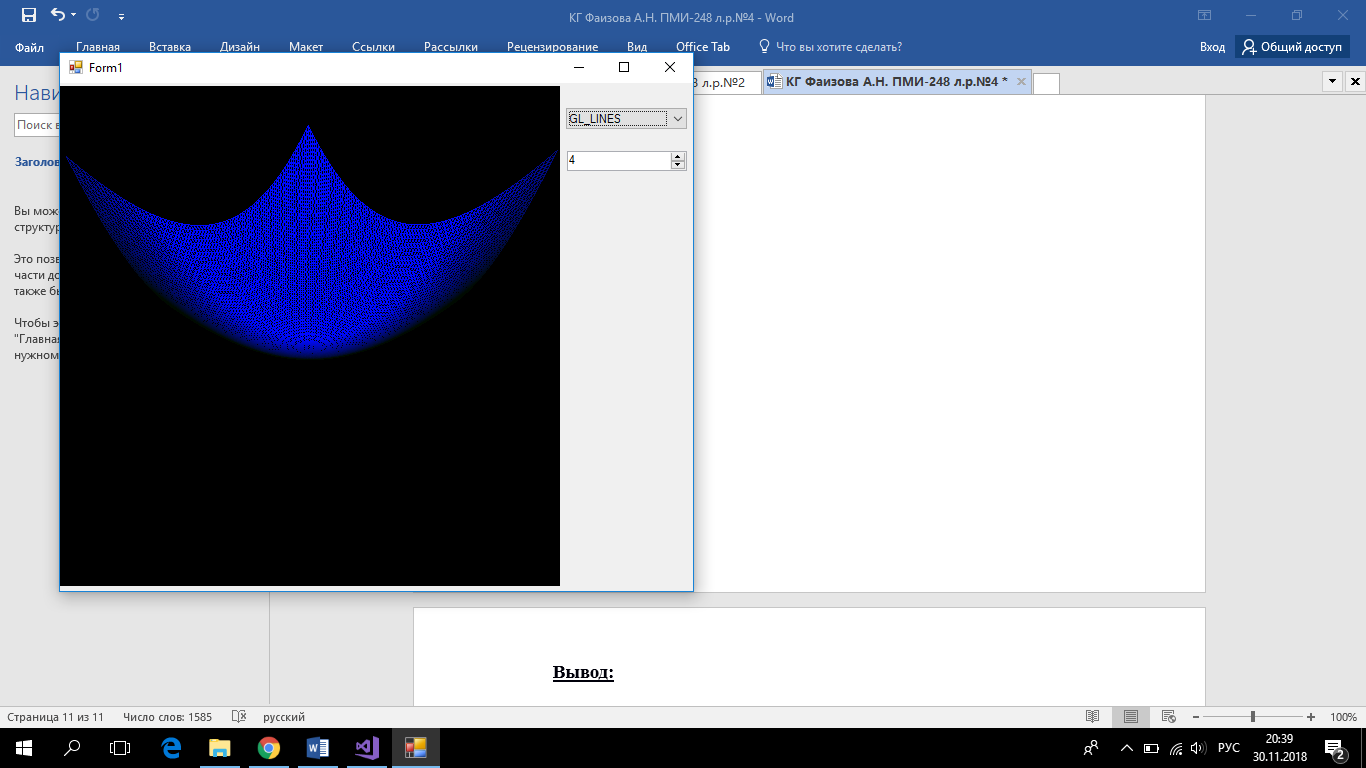
AnT.SwapBuffers();

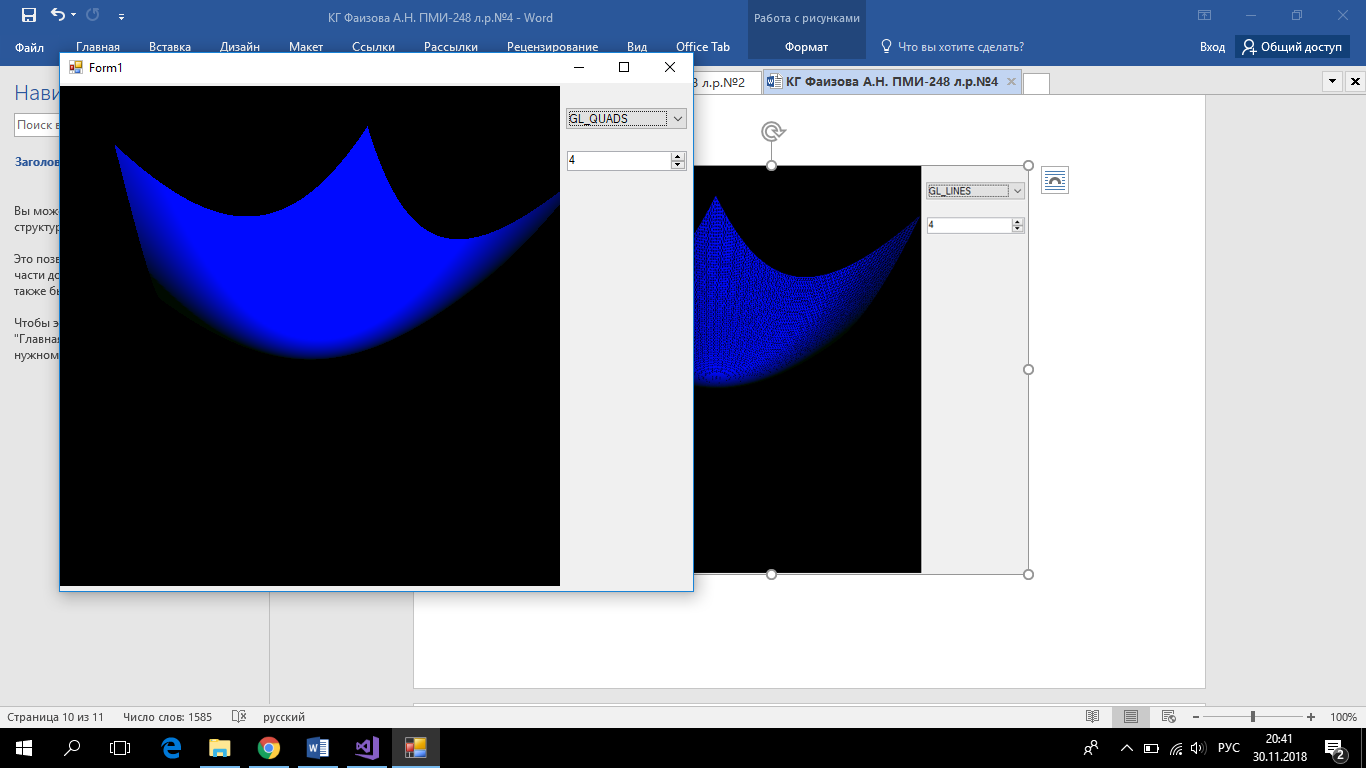
}

}

Пример выполнения программы







**Вывод:** на данной лабораторной работе мы разработали приложение, визуализирующее трехмерную, функционально заданную поверхность, соответствующую одному из заданных вариантов. В итоге, мы получили анимированную визуализацию трехмерной поверхности с возможностью переключения между представлениями.