Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования "Восточно-Сибирский государственный университет технологий и управления"

Факультет компьютерных наук и технологий

Кафедра «Программная инженерия и искусственный интеллект»

**Курсовая работа**

по дисциплине "Геометрия и топология"

на тему "Разработка приложения с использованием OpenGL для построения динамического изображения трехмерной модели объекта "Корабль"

Выполнил: студент гр. Б720 Югаев В.Е.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель: ст. преподаватель Цагадаева Е. Н.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_ Дата зашиты \_\_\_\_\_\_\_\_\_\_\_

Улан-Удэ

2022

**АННОТАЦИЯ**

Приложение написано на языке С# с использованием библиотеки  
OpenGL. Программа создана в среде Visual Studio 2022. Имеется возможность производить различные настройки, такие как: настройка и выбор перспективы, загрузка 3D объектов имеющих расширение “.ase”, отображение их текстур. Программа имеет интуитивно понятный интерфейс, который схож с другими Windows – приложениям, что указывает на массовость программного продукта.   
 3D модель корабля, сцена и некоторая часть текстур были созданы в приложении Autodesk 3ds Max.

**ОГЛАВЛЕНИЕ**

[АННОТАЦИЯ 2](#_Toc122106111)

[ВВЕДЕНИЕ 4](#_Toc122106111)

[1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕК OPENGL 6](#_Toc122106112)

[1.1 Основные возможности 6](#_Toc122106113)

[1.2 Работа с матрицами 7](#_Toc122106114)

[1.3 Модельные преобразования с помощтю библиотеки GLUT 8](#_Toc122106115)

[1.4 Текстурирование модели с помощью библиотеки Devil 9](#_Toc122106116)

[2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТОРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА "Корабль" 10](#_Toc122106118)

[2.1 Создание 3D модели и сцены с использованием Autodesk 3ds Max 10](#_Toc122106119)

[2.2 Разработка интерфейса пользователя 12](#_Toc122106120)

[2.3 Разработка подсистем управления событиями 12](#_Toc122106116)

[3. ИНФОРМАЦИОННОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 13](#_Toc122106121)

[3.1 Общие сведения о программе 13](#_Toc122106116)

[3.2 Методы и функции проекта 13](#_Toc122106116)

[ЗАКЛЮЧЕНИЕ 16](#_Toc122106122)

[СПИСОК ИСТОЧНИКОВ 17](#_Toc122106122)

[ПРИЛОЖЕНИЕ А 18](#_Toc122106123)

[ПРИЛОЖЕНИЕ Б 19](#_Toc122106124)

## **ВВЕДЕНИЕ**

Компьютерная графика в настоящее время сформировалась как наука об аппаратном и программном обеспечении для разнообразных изображений от простых чертежей до реалистичных образов естественных объектов. Компьютерная графика используется почти во всех научных и инженерных дисциплинах для наглядности и восприятия, передачи информации. Конечным продуктом компьютерной графики является изображение. Это изображение может использоваться в различных сферах, например, оно может быть техническим чертежом, иллюстрацией с изображением детали в руководстве по эксплуатации, простой диаграммой, архитектурным видом предполагаемой конструкции или проектным заданием, рекламной иллюстрацией или кадром из мультфильма.

Одним из важных вопросов в данной области является выбор инструментов или ПО, которое отвечает запросам удобства использования и универсальности применения? Такими инструментами являются программный интерфейс OpenGl и программное обеспечение Autodesk 3Ds Max.

OpenGL является на данный момент одним из самых популярных программных интерфейсов для разработки приложений в области двумерной и трехмерной графики. На данный момент реализация OpenGL включает в себя несколько библиотек (описание базовых функций OpenGL, GLU,GLUT,GLAUX и другие).

Характерными особенностями OpenGL, которые обеспечили распространение и развитие этого графического стандарта, являются:

1. Дополнения и изменения в стандарте реализуются таким образом, чтобы сохранить совместимость с разработанным ранее программным обеспечением.
2. Приложения, использующие OpenGL, гарантируют одинаковый визуальный результат вне зависимости от типа используемой операционной системы и организации отображения информации.
3. Стандарт OpenGL имеет продуманную структуру и интуитивно понятный интерфейс, что позволяет с меньшими затратами создавать эффективные приложения, содержащие меньше строк кода, чем с использованием других графических библиотек.

3Ds Max — это программный пакет от компании Autodesk которая используется для работы по созданию трехмерных изображений с нуля. Программа постоянно совершенствуется, отдельные компании направляют свою деятельность только на то, чтобы создавать модули и плагины для Autodesk 3Ds Max.

Результаты работы редактора 3Ds Max можно видеть во многих мультфильмах и фильмах.

**1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕК OPENGL**

## 

## **1.1 Основные возможности**

Рассмотрим основные возможности, которые предоставляет библиотека OpenGL в:

1. Геометрические и растровые примитивы. На основе геометрических и растровых примитивов строятся все объекты. Из геометрических примитивов библиотека предоставляет: точки, линии, полигоны. Из растровых: битовый массив (bitmap) и образ (image)
2. Использование В-сплайнов. B-сплайны используются для рисования кривых по опорным точкам.
3. Видовые и модельные преобразования. С помощью этих преобразований можно располагать объекты в пространстве, вращать их, изменять форму, а также изменять положение камеры из которой ведётся наблюдение.
4. Работа с цветом. OpenGL предоставляет программисту возможность работы с цветом в режиме RGBA (красный-зелёный-синий-альфа) или используя индексный режим, где цвет выбирается из палитры.
5. Наложение текстуры. Позволяет придавать объектам реалистичность. На объект, например шар, накладывается текстура (изображение), в результате чего наш объект теперь выглядит не просто как шар, а как разноцветный мячик.
6. Сглаживание. Сглаживание позволяет скрыть ступенчатость, свойственную растровым дисплеям. Сглаживание изменяет интенсивность и цвет пикселей около линии, при этом линия смотрится на экране без всяких зигзагов.
7. Освещение. Позволяет задавать источники света, их расположение, интенсивность, и т.д.

**1.2 Работа с матрицами**

Для задания различных преобразований объектов сцены в OpenGL используются операции над матрицами, при этом различают три типа матриц: модельно-видовая, матрица проекций и матрица текстуры. Все они имеют размер 4x4. Видовая матрица определяет преобразования объекта в мировых координатах, такие как параллельный перенос, изменение масштаба и поворот. Матрица проекций определяет, как будут проецироваться трехмерные объекты на плоскость экрана (в оконные координаты), а матрица текстуры определяет наложение текстуры на объект.

Умножение координат на матрицы происходит в момент вызова соответствующей команды OpenGL, определяющей координату (как правило, это команда glVertex.

Для того чтобы выбрать, какую матрицу надо изменить, используется команда: void glMatrixMode(GLenum mode), вызов которой, со значением параметра "mode" равным GL\_MODELVIEW, GL\_PROJECTION, или GL\_TEXTURE включает режим работы с модельно-видовой матрицей, матрицей проекций, или матрицей текстуры соответственно. Для вызова команд, задающих матрицы того или иного типа, необходимо сначала установить соответствующий режим.

## **1.3 Взаимодействие с моделью**

OpenGL Utility Toolkit (GLUT) – это библиотека различных утилит, предназначенная для приложений, разрабатываемых с использованием OpenGL API. По большей части данная библиотека отвечает за операции ввода-вывода, необходимые для обмена с операционной системой.

В OpenGL API существуют команды для модельных преобразований:

1. glTranslate\*()
2. glRotate\*()
3. glScale\*()

Перенос реализуется с помощью команды void glTranslate{fd} (TYPE x, TYPE y, TYPE z); - данная команда позволяет произвести перенос объекта по осям X, Y, Z, которые указываются в качестве параметров.

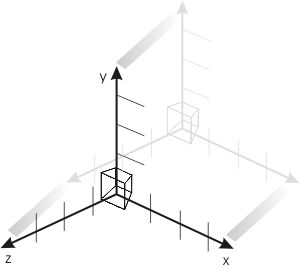


Рисунок 1 - Перенос объекта.

Поворот реализуется с помощью команды void glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z); - данная функция умножает текущую матрицу на матрицу, производящую поворот объекта (операция поворота происходит в направлении против часовой стрелки, вокруг луча из начала координат, проходящего через точку (x, y, z)).

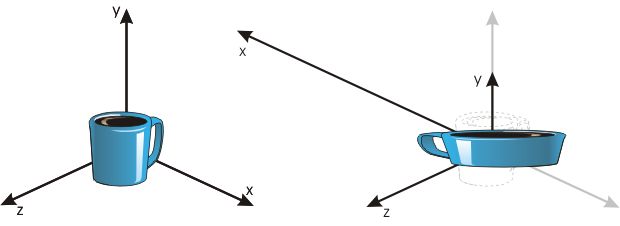
Масштабирование реализуется с помощью команды void glScale{fd} (TYPE x, TYPE y, TYPE z); - данная команда позволяет сжимать или отражать объект вдоль координатных осей.   


Рисунок 2 - Масштабирование и отражение объекта.

Данные команды являются эквивалентом соответствующих матриц с вызовом функции glTranslate\*() с нужной матрицей в качестве аргумента.

**1.4 Текстурирование модели с помощью библиотеки Devil.**

Tao.Devil– кроссплатформенная библиотека, реализующая программный интерфейс для работы с изображениями. На данный момент библиотека поддерживает работу с изображениями 43-х форматов для чтения и 17-ти форматов для записи.   
 Текстурирование модели с помощью данной библиотеки происходит в данном порядке:

1. Создание текстурного объекта и наложение текстуры для него.
2. Определение, как текстура должна воздействовать на каждый пиксель.
3. Активизация механизма текстурирования.
4. Рисование сцены, передавая на конвейер визуализации, геометрические координаты и координаты текстуры.

**2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТОРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА "Корабль"**

**2.1 Создание 3D модели и сцены с использованием Autodesk 3ds Max.**

Для создания 3D модели и сцены для неё было использовано ПО Autodesk 3ds Max. Выбор данного ПО для реализации модели был условлен простотой работы в этом интерфейсе. В отличии от создания модели в C# с помощью OpenGl, где результат построения не будет виден сразу времени, а только после запуска кода программы, в Autodesk 3ds Max можно наблюдать результат в реальном времени, что в разы упрощает работу и уменьшает затрачиваемое время на разработку.

Модель корабля состоит из 27 разных деталей. Пять деталей созданы из объекта “Cylinder”, Три детали из объекта “Cube”, Одна деталь из объекта “Circle”, Восемнадцать деталей из объекта “Plane”.



Рисунок 3 – Модель корабля без текстур.

Сцена созданы следующим образом. Вокруг главной модели (корабля) поставлены 4 стены, вода, на которой “стоит” корабль представляет собой ровную поверхность, на которую нужно добавить неровности для более точно имитации воды.

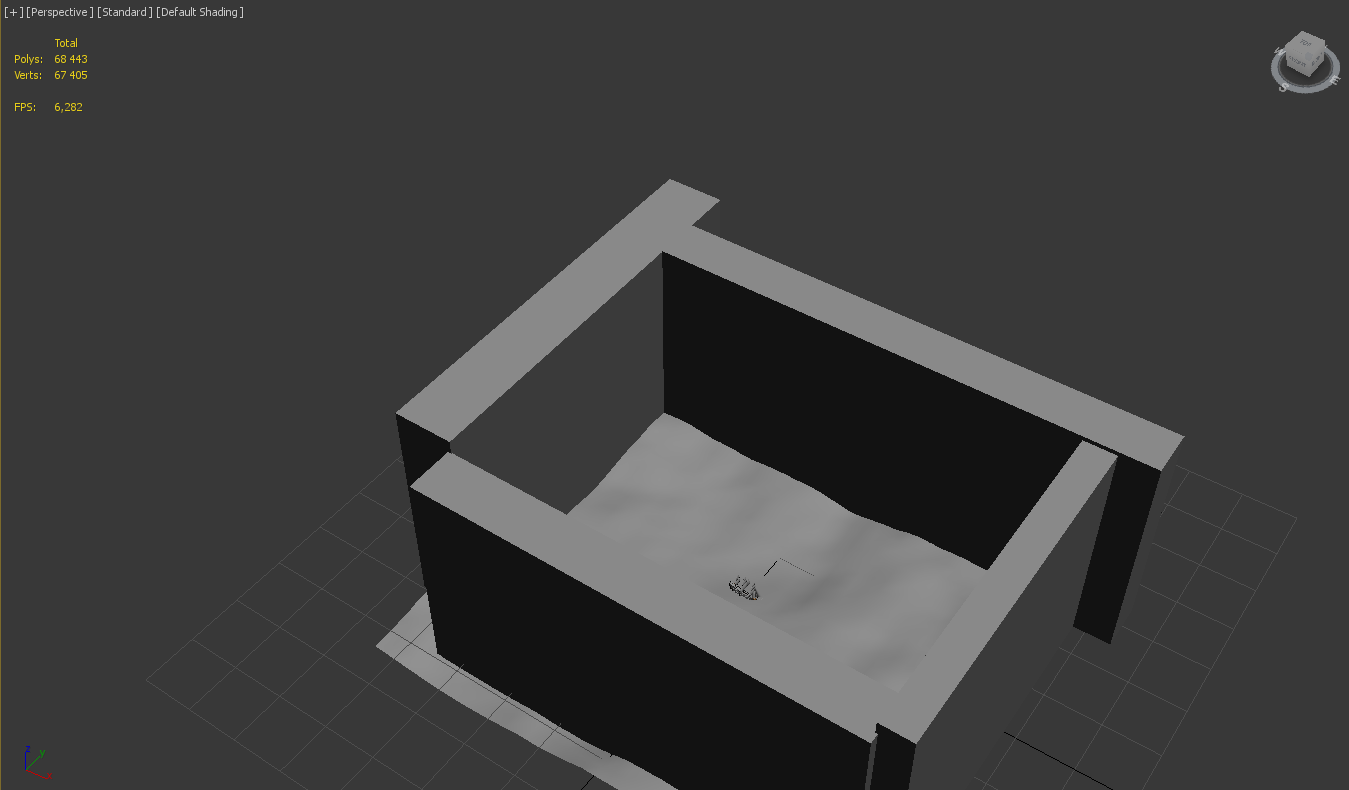


Рисунок 4 – Сцена.

После реализации вышеописанного нужно “наложить” текстуры на все 27 деталей корабля и 5 деталей сцены. Все текстуры представляют собой изображения в формате “.jpg”. В итоге имеем:



Рисунок 5 – Конечный результат.

**2.2 Разработка интерфейса пользователя**

Для данной программы разработан интерфейс, позволяющий:

* Загрузка файла формата “.ase”
* Вращение вдоль оси X.
* Вращение вдоль оси Y.
* Вращение вдоль оси Z.

**2.3 Разработка подсистемы управления событиями**

Любое windows-положение основано на событиях и их обработке, другими словами поведение программы управляется событиями. Данный проект тоже является windows приложением, и, следовательно, обработка событий занимает важное место. К основным событиям, играющим важную, для корректной работы программы, роль относятся следующие:

* private void trackBar1\_Scroll(object sender, EventArgs e) – ползунок для перемещения по оси X.
* private void trackBar2\_Scroll(object sender, EventArgs e) – ползунок для перемещения по оси Y.
* private void trackBar3\_Scroll(object sender, EventArgs e) – ползунок для перемещения по оси Z.
* private void trackBar4\_Scroll(object sender, EventArgs e) – ползунок для изменения угла просмотра.
* private void загрузитьМодельToolStripMenuItem\_Click(object sender, EventArgs e) – загрузка модели формата “.ase”.
* private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e) – выбор оси.

**3. ИНФОРМАЦИОННОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ**

* 1. **Общие сведения о программе**

Данный проект работает по следующему принципу: в программу загружается файл формата “.ase”, который в дальнейшем “открывается” и вся информация о модели сделанной в Autodesk 3ds Max, а именно: пути к текстурам и их координаты, координаты модели и всех её вершин,  последовательности из номеров вершин, которые мы должны соединять. Данная информация переносится в созданные для этого переменные в C#.

При работе с данной программой у пользователя есть возможность работать с визуальной моделью данного объекта (вращать ее относительно осей).

Программное обеспечение, на котором разработана приложение – Microsoft Visual 2022 С# .

**3.2 Методы и функции проекта**

Одним из важных методов является public int LoadModel(string FileName), с его помощью мы считываем из файла информацию о текстуре (файл, который должен быть загружен в качестве текстуры, а также текстурные координаты), после чего загружается и визуализируется трехмерная модель с текстурой. Таким образом, любая сцена, созданная в 3D Studio Max или другом пакете трехмерного моделирования может быть загружена в программу. После этого” геометрия” модели будет загружена в массив объектов limb, каждый из элементов которого будет отвечать за один из логических элементов сцены. После загрузки производится визуализация данной модели в дисплейный список. После этого нам достаточно вызывать данный дисплейный список для отрисовки “геометрии”.

Алгоритм чтения файла и организация файла модели в памяти программы выглядит следующим образом:

1. Начинается построчное чтение файла (вся необходимая информация для любой управляющей конструкции будет находиться в рамках одной строки, поэтому построчное чтение нам подходит)
2. Отсечение первого слова в строке.
3. Проверка первого символа в полученном первом слове – если оно равно «\*», то мы имеем управляющее слово.
4. На основе управляющего слова проверяем, что это за управляющее слово, и если оно относится к тем, которые необходимы (описывают геометрию, текстурные координаты), то обрабатываем данную строку, отрезая первые слова и затем переводя их из строкового представления к числовому.

В таблицах 3.1 и 3.2 – описаны основные методы и функции данного проекта с их назначением.

Таблица 3.1 – Основные методы и функции

|  |  |  |
| --- | --- | --- |
| № | Метод | Назначение |
| 1 | public void LoadTextureForModel(string FileName) | загрузка текстуры |
| 2 | private static uint MakeGlTexture(int Format, IntPtr pixels, int w, int h) | создание текстуры в памяти OpenGL |
| 3 | public int RotateModel(int os, float target, float step) | вращение 3D модели |
| 4 | public int LoadModel(string FileName) | загрузка модели |
| 5 | private void CreateList() | функция отрисовки |
| 6 | private string GetFirstWord(string word, int from) | чтение загружаемого файла |
| 7 | public void DrawModel() | функция отрисовки 3D модели |
| 8 | public void createTextureVertexMem(int a) | массивы для текстурных координат |
| 9 | public void createTextureFaceMem(int b) | привязка значений текстурных координат к полигонам |

Таблица 3.2 – Спецификации классов

|  |  |
| --- | --- |
| Название | Назначение |
| Limb | отвечает за логические единицы 3D объектов в загружаемой сцене |
| TexturesForObjects | класс для работы с текстурами |
| anModelLoader | класс, выполняющий загрузку 3D модели |

**ЗАКЛЮЧЕНИЕ**

В ходе разработки данного приложения были получены практические навыки по разработке программ для операционных систем семейства Windows с применением технологий трехмерной графики с использованием библиотек OpenGL, GLUT, Tao.Devil, а также Autodesk 3ds Max.

Таким образом, можно выделить следующие решенные в рамках данной курсовой работы задачи:

* изучение принципов работы OpenGL в оконной среде Windows;
* получение практических навыков использования средств OpenGL и Autodesk 3ds Max;
* получение навыков программирования статичных трехмерных сцен;

Также была проведена работа с такими возможностями библиотеки как:

* использование графических примитивов;
* загрузка текстур;
* применение проекции.

**СПИСОК ИСТОЧНИКОВ**

1. Уроки OpenGL [Электронный ресурс]//esate – URL <http://esate.ru/uroki/OpenGL/>

2. Графика, решаемые ею задачи и направления [Электронный ресурс]//elib – URL <http://elib.kstu.kz/fulltext/books/Prikladn@ie_graficheskie_sistem@i/teory/лекция%20_1ban.htm>

3. Введение в OpenGL [Электронный ресурс]//codenet – URL [http://www.codenet.ru/progr/opengl/3d.php](http://www.codenet.ru/progr/opengl/3d.php%20)

4. Видео уроки 3ds Max [Электронный ресурс]//3dmaster – URL [https://3dmaster.ru/uroki/](https://3dmaster.ru/uroki/%20) **ПРИЛОЖЕНИЕ А**

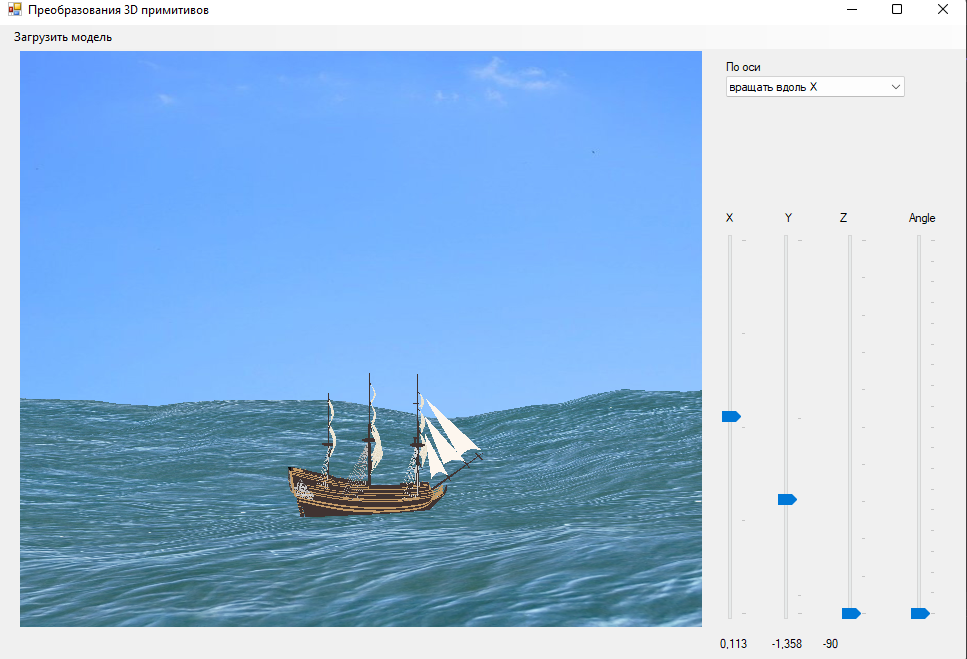
****

Рисунок – Трехмерная модель объекта "Корабль".



Рисунок – Прототип модели.

**ПРИЛОЖЕНИЕ Б**

Листинг программы:

Form1.cs\*

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using Tao.DevIl;

using Tao.FreeGlut;

using Tao.OpenGl;

using Tao.Platform.Windows;

namespace Model

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

AnT.InitializeContexts();

}

double a = 0, b = 0, c = -5, d =-80, zoom = 4;

int os\_x = 0, os\_y = 0, os\_z = 500;

bool Wire = false;

anModelLoader Model = null;

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

{

Glut.glutInit();

Glut.glutInitDisplayMode(Glut.GLUT\_RGB | Glut.GLUT\_DOUBLE | Glut.GLUT\_DEPTH);

Il.ilInit();

Il.ilEnable(Il.IL\_CONV\_PAL);

Gl.glClearColor(0, 0, 0, 1);

Gl.glViewport(0, 0, AnT.Width, AnT.Height);

Gl.glMatrixMode(Gl.GL\_PROJECTION);

Gl.glLoadIdentity();

Glu.gluPerspective(45, (float)AnT.Width / (float)AnT.Height, 0.1, 200);

Gl.glMatrixMode(Gl.GL\_MODELVIEW);

Gl.glLoadIdentity();

Gl.glEnable(Gl.GL\_DEPTH\_TEST);

Gl.glEnable(Gl.GL\_LIGHTING);

Gl.glEnable(Gl.GL\_LIGHT0);

Gl.glBlendFunc(Gl.GL\_SRC\_ALPHA, Gl.GL\_ONE\_MINUS\_SRC\_ALPHA);

Gl.glEnable(Gl.GL\_BLEND);

Gl.glLineWidth(1.0f);

comboBox1.SelectedIndex = 0;

// опиции для загрузки файла

openFileDialog1.Filter = "ase files (\*.ase)|\*.ase|All files (\*.\*)|\*.\*";

}

private void Draw()

{

Gl.glClear(Gl.GL\_COLOR\_BUFFER\_BIT | Gl.GL\_DEPTH\_BUFFER\_BIT);

Gl.glLoadIdentity();

Gl.glColor3i(255, 0, 0);

Gl.glPushMatrix();

Gl.glTranslated(a, b, c);

Gl.glRotated(d, os\_x, os\_y, os\_z);

Gl.glScaled(zoom, zoom, zoom);

if(Model != null)

Model.DrawModel();

Gl.glPopMatrix();

Gl.glFlush();

AnT.Invalidate();

}

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

{

Draw();

}

private void trackBar1\_Scroll(object sender, EventArgs e)

{

a = (double)trackBar1.Value / 1000.0;

label4.Text = a.ToString();

Draw();

}

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

{

}

private void trackBar2\_Scroll(object sender, EventArgs e)

{

b = (double)trackBar2.Value / 1000.0;

label5.Text = b.ToString();

Draw();

}

private void trackBar3\_Scroll(object sender, EventArgs e)

{

c = (double)trackBar3.Value / 1000.0;

label6.Text = c.ToString();

Draw();

}

private void trackBar4\_Scroll(object sender, EventArgs e)

{

d = (double)trackBar4.Value;

label6.Text = d.ToString();

Draw();

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

switch (comboBox1.SelectedIndex)

{

case 0:

{

os\_x = 1;

os\_y = 0;

os\_z = 0;

break;

}

case 1:

{

os\_x = 0;

os\_y = 1;

os\_z = 0;

break;

}

case 2:

{

os\_x = 0;

os\_y = 0;

os\_z = 1;

break;

}

}

Draw();

}

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

Draw();

}

// загрузка модели

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

{

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

Model = new anModelLoader();

Model.LoadModel(openFileDialog1.FileName);

RenderTimer.Start();

}

}

}

}

anModelLoader.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Windows.Forms;

using Tao.OpenGl;

using Tao.FreeGlut;

using Tao.Platform.Windows;

using Tao.DevIl;

namespace Model

{

// класс LIMB отвечает за логические единицы 3D объектов в загружаемой сцене

class LIMB

{

// при инициализации мы должны указать количество вершин (vertex) и полигонов (face) которые описывают геометри под-объекта

public LIMB(int a, int b)

{

if (temp[0] == 0)

temp[0] = 1;

// записываем количество вершин и полигонов

VandF[0] = a;

VandF[1] = b;

// выделяем память

memcompl();

}

public int Itog; // флаг успешности

// массивы для хранения данных (геометрии и текстурных координат)

public float[,] vert;

public int[,] face;

public float[,] t\_vert;

public int[,] t\_face;

// номер материала (текстуры) данного под-объекта

private int MaterialNom = -1;

// временное хранение информации

public int[] VandF = new int[4];

private int[] temp = new int[2];

// флаг , говорящий о том, что модель использует текстуру

private bool ModelHasTexture = false;

// функция для определения значения флага (о наличии текстуры)

public bool NeedTexture()

{

// возвращаем значение флага

return ModelHasTexture;

}

public void SetMaterialNom(int new\_nom)

{

MaterialNom = new\_nom;

if(MaterialNom > -1)

// отмечаем флаг о наличии текстуры

ModelHasTexture = true;

}

// массивы для текстурных координат

public void createTextureVertexMem(int a)

{

VandF[2] = a;

t\_vert = new float[3, VandF[2]];

}

// привязка значений текстурных координат к полигонам

public void createTextureFaceMem(int b)

{

VandF[3] = b;

t\_face = new int[3, VandF[3]];

}

// память для геометрии

private void memcompl()

{

vert = new float[3, VandF[0]];

face = new int[3, VandF[1]];

}

// номер текстуры

public int GetTextureNom()

{

return MaterialNom;

}

};

// класс для работы с текстурами

public class TexturesForObjects

{

public TexturesForObjects()

{

}

// имя текстуры

private string texture\_name = "";

// ее ID

private int imageId = 0;

// идетификатор текстуры в памяти openGL

private uint mGlTextureObject = 0;

// получение этого идентификатора

public uint GetTextureObj()

{

return mGlTextureObject;

}

// загрузка текстуры

public void LoadTextureForModel(string FileName)

{

// запоминаем имя файла

texture\_name = FileName;

// создаем изображение с индификатором imageId

Il.ilGenImages(1, out imageId);

// делаем изображение текущим

Il.ilBindImage(imageId);

string url = texture\_name;

// если загрузка удалась

if (Il.ilLoadImage(url))

{

// если загрузка прошла успешно

// сохраняем размеры изображения

int width = Il.ilGetInteger(Il.IL\_IMAGE\_WIDTH);

int height = Il.ilGetInteger(Il.IL\_IMAGE\_HEIGHT);

// определяем число бит на пиксель

int bitspp = Il.ilGetInteger(Il.IL\_IMAGE\_BITS\_PER\_PIXEL);

switch (bitspp)// в зависимости оп полученного результата

{

// создаем текстуру используя режим GL\_RGB или GL\_RGBA

case 24:

mGlTextureObject = MakeGlTexture(Gl.GL\_RGB, Il.ilGetData(), width, height);

break;

case 32:

mGlTextureObject = MakeGlTexture(Gl.GL\_RGBA, Il.ilGetData(), width, height);

break;

}

// очищаем память

Il.ilDeleteImages(1, ref imageId);

}

}

// создание текстуры в панями openGL

private static uint MakeGlTexture(int Format, IntPtr pixels, int w, int h)

{

// индетефекатор текстурного объекта

uint texObject;

// генерируем текстурный объект

Gl.glGenTextures(1, out texObject);

// устанавливаем режим упаковки пикселей

Gl.glPixelStorei(Gl.GL\_UNPACK\_ALIGNMENT, 1);

// создаем привязку к только что созданной текстуре

Gl.glBindTexture(Gl.GL\_TEXTURE\_2D, texObject);

// устанавливаем режим фильтрации и повторения текстуры

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_WRAP\_S, Gl.GL\_REPEAT);

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_WRAP\_T, Gl.GL\_REPEAT);

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_MAG\_FILTER, Gl.GL\_LINEAR);

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_MIN\_FILTER, Gl.GL\_LINEAR);

Gl.glTexEnvf(Gl.GL\_TEXTURE\_ENV, Gl.GL\_TEXTURE\_ENV\_MODE, Gl.GL\_REPLACE);

// создаем RGB или RGBA текстуру

switch (Format)

{

case Gl.GL\_RGB:

Gl.glTexImage2D(Gl.GL\_TEXTURE\_2D, 0, Gl.GL\_RGB, w, h, 0, Gl.GL\_RGB, Gl.GL\_UNSIGNED\_BYTE, pixels);

break;

case Gl.GL\_RGBA:

Gl.glTexImage2D(Gl.GL\_TEXTURE\_2D, 0, Gl.GL\_RGBA, w, h, 0, Gl.GL\_RGBA, Gl.GL\_UNSIGNED\_BYTE, pixels);

break;

}

// возвращаем индетефекатор текстурного объекта

return texObject;

}

}

// небольшой класс для описания ориентации модели в 3D пространстве сцены

public class Model\_Prop

{

public Model\_Prop()

{

pos\_abs[0] = 0;

pos\_abs[1] = 0;

pos\_abs[2] = 0;

maximum[0] = 0;

maximum[1] = 0;

maximum[2] = 0;

minimum[0] = 0;

minimum[1] = 0;

minimum[2] = 0;

rotating\_angles[0] = 0;

rotating\_angles[1] = 0;

rotating\_angles[2] = 0;

}

public float[] pos\_abs = new float[3];

public float[] maximum = new float[3];

public float[] minimum = new float[3];

public float[] rotating\_angles = new float[3];

};

// класс, выполняющий загрузку 3D модели

public class anModelLoader

{

public anModelLoader()

{

}

// имя файла

public string FName = "C:\\Users\\Vlad\\Desktop\\лучший6.ASE";

// загружен ли (флаг)

private bool isLoad = false;

// счетчик по-объектов

private int count\_limbs;

// переменная для зранения номера текстуры

private int mat\_nom = 0;

// номер дисплейног осписка с данной моделью

private int thisList = 0;

// данная переменная будет указывать на количество прочитанных символов в строке при чтении информации из файла

private int GlobalStringFrom = 0;

// массив под-объектов

LIMB[] limbs = null;

// массви для хранения текстур

TexturesForObjects[] text\_objects = null;

// описание ориентации модели

Model\_Prop coord = new Model\_Prop();

// установка минимумов и максимумов для размещения модели

public void SetMinimum(float x, float y, float z)

{

coord.minimum[0] = x;

coord.minimum[1] = y;

coord.minimum[2] = z;

}

public void SetMaximum(float x, float y, float z)

{

coord.maximum[0] = x;

coord.maximum[1] = y;

coord.maximum[2] = z;

}

public void SetAbsCoords(float x, float y, float z)

{

coord.pos\_abs[0] = x;

coord.pos\_abs[1] = y;

coord.pos\_abs[2] = z;

}

// вращение 3D модели

public int RotateModel(int os, float target, float step)

{

if((coord.rotating\_angles[os] - target) > 0)

{

coord.rotating\_angles[os] -= step;

if(coord.rotating\_angles[os] < target)

{

coord.rotating\_angles[os] = target;

return -1;

}

}

else

{

coord.rotating\_angles[os] += step;

if(coord.rotating\_angles[os] > target)

{

coord.rotating\_angles[os] = target;

return -1;

}

}

return 0;

}

// перемещение модели

public int MoveModel(int os, float target, float step)

{

if (step == 0)

return -1;

float real\_target = target;

if ((coord.pos\_abs[os] - real\_target) > 0)

{

if (coord.pos\_abs[os] - step >= coord.minimum[os])

{

coord.pos\_abs[os] -= step;

if (coord.pos\_abs[os] < real\_target)

{

coord.pos\_abs[os] = real\_target;

return -1;

}

return 0;

}

else

{

coord.pos\_abs[os] = coord.minimum[os];

return -1;

}

}

if ((coord.pos\_abs[os] - real\_target) < 0)

{

if (coord.pos\_abs[os] + step <= coord.maximum[os])

{

coord.pos\_abs[os] += step;

if (coord.pos\_abs[os] > real\_target)

{

coord.pos\_abs[os] = real\_target;

return -1;

}

return 0;

}

else

{

coord.pos\_abs[os] = coord.maximum[os];

return -1;

}

}

if ((coord.pos\_abs[os] - real\_target) == 0)

return -1;

return 0;

}

// загрузка модели

public int LoadModel(string FileName)

{

// модель может содержать до 256 под-объектов

limbs = new LIMB[256];

// счетчик скинут

int limb\_ = -1;

// имся файла

FName = FileName;

// начинаем чтение файла

StreamReader sw = File.OpenText(FileName);

// временные буферы

string a\_buff = "";

string b\_buff = "";

string c\_buff = "";

// счетчики вершин и полигонов

int ver = 0, fac = 0;

// если строка успешно прочитана

while ((a\_buff = sw.ReadLine()) != null)

{

// получаем первое слово

b\_buff = GetFirstWord(a\_buff, 0);

if (b\_buff[0] == '\*') // определеям, является ли первый символ звездочкой

{

switch (b\_buff) // если да, то проверяем какое управляющее слово содержится в первом прочитаном слове

{

case "\*MATERIAL\_COUNT": // счетчик материалов

{

// получаем первое слово от символа указанного в GlobalStringFrom

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

int mat = System.Convert.ToInt32(c\_buff);

// создаем объект для текстуры в памяти

text\_objects = new TexturesForObjects[mat];

continue;

}

case "\*MATERIAL\_REF": // номер текстуры

{

// записываем для текущего под-объекта номер текстуры

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

int mat\_ref = System.Convert.ToInt32(c\_buff);

// устанавливаем номер материала, соответствующий данной модели.

limbs[limb\_].SetMaterialNom(mat\_ref);

continue;

}

case "\*MATERIAL": // указание на материал

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

mat\_nom = System.Convert.ToInt32(c\_buff);

continue;

}

case "\*GEOMOBJECT": // начинается описание геметрии под-объекта

{

limb\_++; // записываем в счетчик под-объектов

continue;

}

case "\*MESH\_NUMVERTEX": // количесвто вершин в под-объекте

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

ver = System.Convert.ToInt32(c\_buff);

continue;

}

case "\*BITMAP": // имя текстуры

{

c\_buff = ""; // обнуляем временный буффер

for (int ax = GlobalStringFrom + 2; ax < a\_buff.Length - 1; ax++)

c\_buff += a\_buff[ax]; // считываем имя текстуры

text\_objects[mat\_nom] = new TexturesForObjects(); // новый объект для текстуры

text\_objects[mat\_nom].LoadTextureForModel(c\_buff); // загружаем текстуру

continue;

}

case "\*MESH\_NUMTVERTEX": // количество текстурных координат, данное слово говорит о наличии текстурных координат - следовательно мы должны выделить память для них

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

if (limbs[limb\_] != null)

{

limbs[limb\_].createTextureVertexMem(System.Convert.ToInt32(c\_buff));

}

continue;

}

case "\*MESH\_NUMTVFACES": // память для текстурных координат (faces)

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

if (limbs[limb\_] != null)

{

// выделяем память для текстурныйх координат

limbs[limb\_].createTextureFaceMem(System.Convert.ToInt32(c\_buff));

}

continue;

}

case "\*MESH\_NUMFACES": // количество полиговов в под-объекте

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

fac = System.Convert.ToInt32(c\_buff);

// если было объвляющее слово \*GEOMOBJECT (гарантия выполнения условия limb\_ > -1) и были указаны количство вершин

if (limb\_ > -1 && ver > -1 && fac > -1)

{

// создаем новый под-объект в памяти

limbs[limb\_] = new LIMB(ver, fac);

}

else

{

// иначе завершаем неудачей

return -1;

}

continue;

}

case "\*MESH\_VERTEX": // информация о вершине

{

// под-объект создан в памяти

if (limb\_ == -1)

return -2;

if (limbs[limb\_] == null)

return -3;

string a1 = "", a2 = "", a3 = "", a4 = "";

// полчучаем информацию о кооринатах и номере вершины

// (получаем все слова в строке)

a1 = GetFirstWord(a\_buff, GlobalStringFrom);

a2 = GetFirstWord(a\_buff, GlobalStringFrom);

a3 = GetFirstWord(a\_buff, GlobalStringFrom);

a4 = GetFirstWord(a\_buff, GlobalStringFrom);

// преобразовываем в целое цисло

int NomVertex = System.Convert.ToInt32(a1);

// заменяем точки в представлении числа с плавающей точкой, на запятые, чтобы правильно выполнилась функция

// преобразования строки в дробное число

a2 = a2.Replace('.', ',');

a3 = a3.Replace('.', ',');

a4 = a4.Replace('.', ',');

// записываем информацию о вершине

limbs[limb\_].vert[0, NomVertex] = (float)System.Convert.ToDouble(a2); // x

limbs[limb\_].vert[1, NomVertex] = (float)System.Convert.ToDouble(a3); // y

limbs[limb\_].vert[2, NomVertex] = (float)System.Convert.ToDouble(a4); // z

continue;

}

case "\*MESH\_FACE": // информация о полигоне

{

// под-объект создан в памяти

if (limb\_ == -1)

return -2;

if (limbs[limb\_] == null)

return -3;

// временные перменные

string a1 = "", a2 = "", a3 = "", a4 = "", a5 = "", a6 = "", a7 = "";

// получаем все слова в строке

a1 = GetFirstWord(a\_buff, GlobalStringFrom);

a2 = GetFirstWord(a\_buff, GlobalStringFrom);

a3 = GetFirstWord(a\_buff, GlobalStringFrom);

a4 = GetFirstWord(a\_buff, GlobalStringFrom);

a5 = GetFirstWord(a\_buff, GlobalStringFrom);

a6 = GetFirstWord(a\_buff, GlobalStringFrom);

a7 = GetFirstWord(a\_buff, GlobalStringFrom);

// получаем нмоер полигона из первого слова в строке, заменив последний символ ":" после номера на флаг окончания строки.

int NomFace = System.Convert.ToInt32(a1.Replace(':', '\0'));

// записываем номера вершин, которые нас интересуют

limbs[limb\_].face[0, NomFace] = System.Convert.ToInt32(a3);

limbs[limb\_].face[1, NomFace] = System.Convert.ToInt32(a5);

limbs[limb\_].face[2, NomFace] = System.Convert.ToInt32(a7);

continue;

}

// текстурые координаты

case "\*MESH\_TVERT":

{

// под-объект создан в памяти

if (limb\_ == -1)

return -2;

if (limbs[limb\_] == null)

return -3;

// временные перменные

string a1 = "", a2 = "", a3 = "", a4 = "";

// получаем все слова в строке

a1 = GetFirstWord(a\_buff, GlobalStringFrom);

a2 = GetFirstWord(a\_buff, GlobalStringFrom);

a3 = GetFirstWord(a\_buff, GlobalStringFrom);

a4 = GetFirstWord(a\_buff, GlobalStringFrom);

// преобразуем первое слово в номер вершины

int NomVertex = System.Convert.ToInt32(a1);

// заменяем точки в представлении числа с плавающей точкой, на запятые, чтобы правильно выполнилась функция

// преобразования строки в дробное число

a2 = a2.Replace('.', ',');

a3 = a3.Replace('.', ',');

a4 = a4.Replace('.', ',');

// записываем значение вершины

limbs[limb\_].t\_vert[0, NomVertex] = (float)System.Convert.ToDouble(a2); // x

limbs[limb\_].t\_vert[1, NomVertex] = (float)System.Convert.ToDouble(a3); // y

limbs[limb\_].t\_vert[2, NomVertex] = (float)System.Convert.ToDouble(a4); // z

continue;

}

// привязка текстурных координат к полигонам

case "\*MESH\_TFACE":

{

// под-объект создан в памяти

if (limb\_ == -1)

return -2;

if (limbs[limb\_] == null)

return -3;

// временные перменные

string a1 = "", a2 = "", a3 = "", a4 = "";

// получаем все слова в строке

a1 = GetFirstWord(a\_buff, GlobalStringFrom);

a2 = GetFirstWord(a\_buff, GlobalStringFrom);

a3 = GetFirstWord(a\_buff, GlobalStringFrom);

a4 = GetFirstWord(a\_buff, GlobalStringFrom);

// преобразуем первое слово в номер полигона

int NomFace = System.Convert.ToInt32(a1);

// записываем номера вершин, которые опиывают полигон

limbs[limb\_].t\_face[0, NomFace] = System.Convert.ToInt32(a2);

limbs[limb\_].t\_face[1, NomFace] = System.Convert.ToInt32(a3);

limbs[limb\_].t\_face[2, NomFace] = System.Convert.ToInt32(a4);

continue;

}

}

}

}

// пересохраняем количесвто полигонов

count\_limbs = limb\_;

// получаем ID для создаваемого дисплейного списка

int nom\_l = Gl.glGenLists(1);

thisList = nom\_l;

// генерируем новый дисплейный список

Gl.glNewList(nom\_l, Gl.GL\_COMPILE);

// отрисовываем геометрию

CreateList();

// завершаем дисплейный список

Gl.glEndList();

// загрузка завершена

isLoad = true;

return 0;

}

// функция отрисовки

private void CreateList()

{

// сохраняем тек матрицу

Gl.glPushMatrix();

// проходим циклом по всем под-объектам

for (int l = 0; l <= count\_limbs; l++)

{

// если текстура необходима

if( limbs[l].NeedTexture() )

if( text\_objects[limbs[l].GetTextureNom()] != null ) // текстурный объект существует

{

Gl.glEnable(Gl.GL\_TEXTURE\_2D); // включаем режим текстурирования

// ID текстуры в памяти

uint nn = text\_objects[limbs[l].GetTextureNom()].GetTextureObj();

// активируем (привязываем) эту текстуру

Gl.glBindTexture(Gl.GL\_TEXTURE\_2D, nn);

}

Gl.glEnable(Gl.GL\_NORMALIZE);

// начинаем отрисовку полигонов

Gl.glBegin(Gl.GL\_TRIANGLES);

// по всем полигонам

for (int i = 0; i < limbs[l].VandF[1]; i++)

{

// временные переменные, чтобы код был более понятен

float x1, x2, x3, y1, y2, y3, z1, z2, z3 = 0;

// вытакскиваем координаты треугольника (полигона)

x1 = limbs[l].vert[0, limbs[l].face[0, i]];

x2 = limbs[l].vert[0, limbs[l].face[1, i]];

x3 = limbs[l].vert[0, limbs[l].face[2, i]];

y1 = limbs[l].vert[1, limbs[l].face[0, i]];

y2 = limbs[l].vert[1, limbs[l].face[1, i]];

y3 = limbs[l].vert[1, limbs[l].face[2, i]];

z1 = limbs[l].vert[2, limbs[l].face[0, i]];

z2 = limbs[l].vert[2, limbs[l].face[1, i]];

z3 = limbs[l].vert[2, limbs[l].face[2, i]];

// рассчитываем номраль

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

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

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

// устанавливаем номраль

Gl.glNormal3f(n1, n2, n3);

// если установлена текстура

if (limbs[l].NeedTexture() && (limbs[l].t\_vert != null) && (limbs[l].t\_face != null))

{

// устанавливаем текстурные координаты для каждой вершины, ну и сами вершины

Gl.glTexCoord2f(limbs[l].t\_vert[0, limbs[l].t\_face[0, i]], limbs[l].t\_vert[1, limbs[l].t\_face[0, i]]);

Gl.glVertex3f(x1, y1, z1);

Gl.glTexCoord2f(limbs[l].t\_vert[0, limbs[l].t\_face[1, i]], limbs[l].t\_vert[1, limbs[l].t\_face[1, i]]);

Gl.glVertex3f(x2, y2, z2);

Gl.glTexCoord2f(limbs[l].t\_vert[0, limbs[l].t\_face[2, i]], limbs[l].t\_vert[1, limbs[l].t\_face[2, i]]);

Gl.glVertex3f(x3, y3, z3);

}

else // иначе - отрисовка только вершин

{

Gl.glVertex3f(x1, y1, z1);

Gl.glVertex3f(x2, y2, z2);

Gl.glVertex3f(x3, y3, z3);

}

}

// завершаем отрисовку

Gl.glEnd();

Gl.glDisable(Gl.GL\_NORMALIZE);

// открлючаем текстурирование

Gl.glDisable(Gl.GL\_TEXTURE\_2D);

}

// возвращаем сохраненную ранее матрицу

Gl.glPopMatrix();

}

// функиц я получения первого слова строки

private string GetFirstWord(string word, int from)

{

// from указывает на позицию, начиная с которой будет выполнятся чтение файла

char a = word[from]; // первый символ

string res\_buff = ""; // временный буффер

int L = word.Length; // длина слова

if (word[from] == ' ' || word[from] == '\t') // если первый символ, с которого предстоит искать слово является пробелом или знаком табуляции

{

// необходимо вычисслить наличие секции проблеов или знаков табуляции и откинуть их

int ax = 0;

// проходим до конца слова

for( ax = from; ax < L; ax++)

{

a = word[ax];

if(a != ' ' && a != '\t') // если встречаем символ пробела или табуляции

break; // выходим из цикла.

// таким образом мы откидываем все последовательности пробелов или знаков табуляции, с которых могла начинатся переданная строка

}

if(ax == L) // если вся представленная строка является набором пробелов или знаков табуляции - возвращаем res\_buff

return res\_buff;

else

from = ax; // иначе сохраняем значение ax

}

int bx = 0;

// теперь, когда пробелы и табуляция откинуты мы непосредственно вычисляем слово

for (bx = from; bx < L; bx++)

{

// если встретили знак пробела или табуляции - завершаем чтение слова

if (word[bx] == ' ' || word[bx] == '\t')

break;

// записываем символ в бременный буффер, постепенно получая таким образом слово

res\_buff += word[bx];

}

// если дошли до конца строки

if (bx == L)

bx--; // убераем посл значение

GlobalStringFrom = bx; // позиция в данной строке, для чтения следующего слова в данной строке

return res\_buff; // возвращаем слово

}

// функция отрисовки 3D модели

public void DrawModel()

{

// если модель не загружена - возврат из функции

if (!isLoad)

return;

// сохраняем матрицу

Gl.glPushMatrix();

// масштабирование по умолчанию

Gl.glScalef(0.05f, 0.05f, 0.05f);

// вызов дисплейного списка

Gl.glCallList(thisList);

// возврат матрицы

Gl.glPopMatrix();

}

}

}