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 "Воздушный шар""

Выполнил: студент гр. Б720 Дашидондуков Э.Ц.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель: ст. преподаватель Цагадаева Е. Н.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Улан-Удэ

2022

# **АННОТАЦИЯ**

Приложение написано на языке С# в интерфейсе программирования Windows Forms с использованием библиотеки OpenGL, Tao Framework и GLUT. Программа создана в среде Visual Studio 2022. В рамках работы были выполнены все представляемые задачи. Имеется возможность производить различные настройки, такие как: перемещение объекта, выбор перспективы и приближение. Программа имеет интуитивно понятный интерфейс, который схож с другими Windows – приложениям, что указывает на массовость программного продукта.

# ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc122106111)

[1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OPENGL, TAO FRAMEWORK И GLUT 5](#_Toc122106112)

[1.1 Спецификация 5](#_Toc122106113)

1.2 Архитектура OpenGL………...…………………………………………………………...5

[1.3 Геометрические преобразования 7](#_Toc122106115)

[1.4 Формирование тел вращения 9](#_Toc122106116)

[1.5 Библиотека DEVIL 10](#_Toc122106117)

[2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТОРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА "Воздушный шар" 12](#_Toc122106118)

[2.1 Разработка окно программы 12](#_Toc122106119)

[2.2 Описание кода (классов и функций) 13](#_Toc122106120)

[3. ЗАКЛЮЧЕНИЕ И ВЫВОДЫ 16](#_Toc122106121)

[СПИСОК ЛИТЕРАТУРЫ 17](#_Toc122106122)

[ПРИЛОЖЕНИЕ А. Исходный код 18](#_Toc122106123)

[ПРИЛОЖЕНИЕ Б Интерфейс программы 35](#_Toc122106124)

# ВВЕДЕНИЕ

В нашей жизни есть много вещей, на которые изменения не влияют часто. Но это явно не включает передовые технологии. Постоянно обновляясь, они падают на жизнь человека, заставляя все больше и больше удивляться и восхищаться.

Современный мир невозможно представить без информационных технологий. Они все глубже и глубже проникают в нашу жизнь, захватывая все больше и больше наук - информатики и информационных технологий, математики, физики. Совершенствуются широко используемые информационные технологии - в образовании, бизнесе, развлечениях.

Информационному обществу нужны новейшие разработки, альтернатива прошлому веку. На помощь приходят 3D-технологии. Все чаще их можно встретить в печати, телевизорах, принтерах.

Сфера развлечений не обходится без новинок 3D - большинство премьер фильмов показывают в формате 3D.

# 1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OPENGL, TAO FRAMEWORK И GLUT

OpenGL (Open Graphics Library – открытая графическая библиотека) был разработан и утвержден в 1992 году ведущими фирмами в области разработки программного обеспечения как эффективный аппаратно-независимый интерфейс, пригодный для реализации на различных платформах. Основой стандарта стала библиотека IRIS GL, разработанная фирмой Silicon Graphics Inc.

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

## *Стабильность.*Дополнения и изменения в стандарте реализуются таким образом, чтобы сохранить совместимость с разработанным ранее программным обеспечением.

## *Надежность и переносимость.*Приложения, использующие OpenGL, гарантируют одинаковый визуальный результат вне зависимости от типа используемой операционной системы и организации отображения информации. Кроме того, эти приложения могут выполняться как на персональных компьютерах, так и на рабочих станциях и суперкомпьютерах.

## *Легкость применения.*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. Необходимые функции для обеспечения совместимости с различным оборудованием реализованы на уровне библиотеки и значительно упрощают разработку приложений.

## 1.2 Архитектура OpenGL

## Функции OpenGL реализованы в модели клиент-сервер. Приложение выступает в роли клиента – оно вырабатывает команды, а сервер OpenGL интерпретирует и выполняет их. Сам сервер может находиться как на том же компьютере, на котором находится клиент (например, в виде динамически загружаемой библиотеки – DLL), так и на другом (при этом может быть использован специальный протокол передачи данных между машинами).

## GL обрабатывает и рисует в буфере кадра графические *примитивы* с учетом некоторого числа выбранных режимов. Каждый примитив – это точка, отрезок, многоугольник и т.д. Каждый режим может быть изменен независимо от других. Определение примитивов, выбор режимов и другие операции описываются с помощью *команд*в форме вызовов функций прикладной библиотеки.

## Примитивы определяются набором из одной или более *вершин*(vertex). Вершина определяет точку, конец отрезка или угол многоугольника. С каждой вершиной ассоциируются некоторые данные (координаты, цвет, нормаль, текстурные координаты и т.д.), называемые *атрибутами*. В подавляющем большинстве случаев каждая вершина обрабатывается независимо от других.

## С точки зрения архитектуры графическая система OpenGL является конвейером, состоящим из нескольких последовательных этапов обработки графических данных.

## Команды OpenGL всегда обрабатываются в том порядке, в котором они поступают, хотя могут происходить задержки перед тем, как проявится эффект от их выполнения. В большинстве случаев OpenGL предоставляет непосредственный интерфейс, т.е. определение объекта вызывает его визуализацию в буфере кадра.

## С точки зрения разработчиков, OpenGL – это набор команд, которые управляют использованием графической аппаратуры. Если аппаратура состоит только из адресуемого буфера кадра, тогда OpenGL должен быть реализован полностью с использованием ресурсов центрального процессора. Обычно графическая аппаратура предоставляет различные уровни ускорения: от аппаратной реализации вывода линий и многоугольников до изощренных графических процессоров с поддержкой различных операций над геометрическими данными.

## 1.3 Геометрические преобразования

Геометрические преобразования – это такие преобразования графических объектов, при которых изменяется направление, масштаб и положение координатной системы, используемой для определения местонахождения точек в пространстве. При этом само пространство не подвергается каким-либо изменениям, т.е. сама структура графического изображения сохраняется. Следовательно, при геометрических преобразованиях неважен тип объекта, так как в преобразованиях участвуют точки.

Основными операциями, которые относятся к геометрическим преобразованиям двумерных графических объектов, являются:

* Перенос начала координат
* Масштабирование
* Поворот осей координат

Рассмотрим эти операции более подробно.

**Перенос**

При выполнении данной операции происходит перенос начала координат на некоторую величину. Обозначим старую систему координат СК1, новую – СК2. При переносе новое начало координат в системе координат СК1 лежит в точке (Tx, Ty), в новой же системе СК2 старое начало координат является точкой (-Tx, -Ty). Начало координат (x,y) системы СК1 в системе СК2 станет точкой (x-Tx, y-Ty).

Стоит упомянуть, что при переносе сохраняются как направление осей координат, так и масштаб.

Матрица, необходимая для выполнения операции переноса имеет следующий вид:

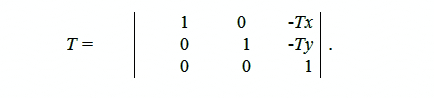


Рисунок 1. Геометрическое преобразование "Перенос".

**Масштабирование**

При выполнении операции масштабирования отрезок единичной длины на оси X становится отрезком длиной Sx на оси X’; отрезок единичной длины на оси Y становится отрезком длиной Sy на оси Y’. Получается, что в новой системе точка с координатой (1,1) преобразуется в точку (Sx, Sy), точка (x,y) будет иметь координаты (x\*Sx, y\*Sy).

Стоит отметить, что при масштабировании начало координат и направление осей новой и старой систем координат совпадают, меняется лишь масштаб по осям.

Матрица, необходимая для выполнения операции масштабирования имеет следующий вид:

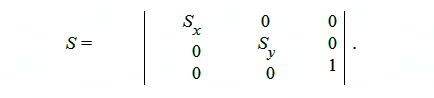


Рисунок 2. Геометрическое преобразование "Масштабирование".

Поворот Рассмотрим частный случай поворота в пространстве XYZ – поворот относительно оси Z, проходящей перпендикулярно плоскости XY.

Предположим, что центр вращения совпадает с началом координат. Тогда при повороте системы координат XY в декартовой плоскости на некоторый произвольный угол w относительно центра вращения, координаты произвольной точки С в системе координат (x’,y’) можно выразить через координаты этой точки в изначальной системе координат (x, y):

x' = x cos w + y sin w,

y' = -x sin w + y cos w

Для выполнения операции поворота используется следующая матрица:

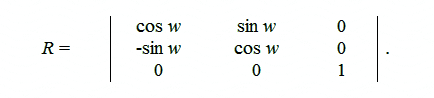


Рисунок 3. Геометрическое преобразование "Поворот".

## 1.4 Формирование тел вращения

Один из способов формирования сложных объемных изображений на экране - применение скелетных тел вращения. При этом выбирается ось вращения, в одной плоскости с ней формируется кривая, аппроксимируемая обычно последовательностью отрезков прямых линий. В результате вращения кривой вокруг выбранной оси и фиксации положения этой кривой через определенные отсчеты угла вращения создается скелетное тело вращения.

Имею некоторую геометрию кривой, заданную рядом точек, мы можем построить объект, основываясь на повороте геометрии данной кривой. Разбив объект таким образом на N данных кривых и соединив их вершины, образовывая полигоны между кривыми, мы получаем оболочку тела вращения.

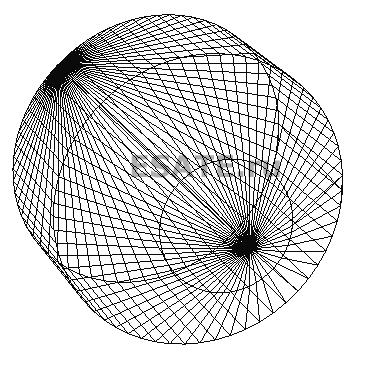


Рисунок 4. Пример объекта, построенного как тело вращения.

Количество отрезков исходной кривой и количество разбиений определяют гладкость объекта. Соответственно, большое количество полигонов увеличивают нагрузку на графический адаптер.

## 1.5 Библиотека DEVIL

DevIL — кроссплатформенная программная библиотека, целью которой является создание общего интерфейса программирования приложений для разных файловых форматов графических изображений.

Для работы с текстурами мы будем использовать библиотеку DevIL. Как мы уже говорили, описывая компоненты библиотеки Tao, DevIL 1.6.8.3, включенная в Tao 2.1.0 (она же OpenIL) – 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.

Чтобы использовать наложение текстуры, мы должны выполнить следующие шаги:

* Создать текстурный объект и задать текстуру для него.
* Задать, как текстура должна воздействовать на каждый пиксель.
* Активизировать механизм текстурирования.
* Нарисовать сцену, передавая на конвейер визуализации, геометрические координаты и координаты текстуры.

# 2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТОРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА "Воздушный шар"

## 2.1 Разработка окно программы

Окно программы (рис. 5) создано с помощью Windows Forms. Для отображения 3д модели использовался инструмент SimpleOpenGlControl (7).

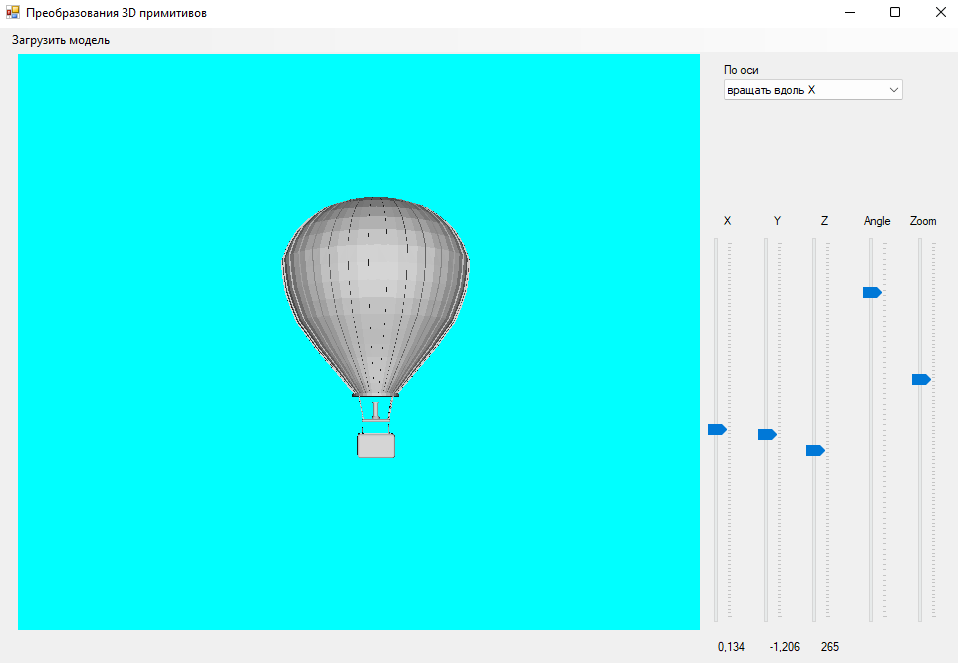


Рисунок 5. Окно программы

Данный интерфейс имеет настройки:

1. Перемещение модели по оси X (влево, вправо)
2. Перемещение модели по оси Y (вверх, вниз)
3. Перемещение модели по оси Z (вперед, назад)
4. Вращение модели по осям XYZ
5. Увеличения или уменьшение модели
6. Выбор оси вращения
7. Так же модель можно вращать мышью по осям XZ

## 2.2 Описание кода (классов и функций)

Главный файл Form1.cs используется для описания инструментов окна

Главная функция. С нее начинается загрузка программы и формы.

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

Функция для определения координат и угла перспективы модели

private void Draw()

Функция таймера. через определенное время запускает функцию Draw

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

Функция изменения координаты Х для модели

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

Функция изменения координаты Y для модели

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

Функция изменения координаты Z для модели

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

Функция определяет нажатие кнопки мыши и ее координат в этот момент

private void AnT\_MouseDown(object sender, MouseEventArgs e)

Функция определяет опускание кнопки мыши

private void AnT\_MouseUp(object sender, MouseEventArgs e)

Функция определяет передвижение мыши. С помощью нее вращается модель

private void AnT\_MouseMove(object sender, MouseEventArgs e)

Функция изменения угла перспективы

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

Функция зума

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

Функция определения оси вращения

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

Файл anModelLoader.cs нужен для отрисовки 3д модели, сцены и считывания файла формата ASE.

Класс LIMB отвечает за логические единицы 3D объектов в загружаемой сцене.

class LIMB

Функция для записи количества вершин и полигонов, которые описывают геометрию подобъекта.

public LIMB(int a, int b)

Функции для определения наличии текстуры

public bool NeedTexture()

public void SetMaterialNom(int new\_nom)

Функция для создания массивов для текстурных координат.

public void createTextureVertexMem(int a)

Функция для привязки значений текстурных координат к полигонам

public void createTextureFaceMem(int b)

Функция для запоминания геометрии

private void memcompl()

Функция для возврата номера текстуры

public int GetTextureNom()

Класс для работы с текстурами

public class TexturesForObjects

Функция для возврата идентификатора текстуры в памяти OpenGL

public uint GetTextureObj()

Функция загрузки текстуры

public void LoadTextureForModel(string FileName)

Функция создания текстуры в памяти OpenGL

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

Небольшой класс для описания ориентации модели в 3D пространстве сцены

public class Model\_Prop

Класс, выполняющий загрузку 3D модели

public class anModelLoader

Функции установки минимумов и максимумов для размещения модели

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

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

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

Функция вращение 3D модели

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

Функция перемещение модели

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

Функция загрузки модели. Чтение файла ASE

public int LoadModel(string FileName)

Функция отрисовки

private void CreateList()

Функция получения первого слова строки

private string GetFirstWord(string word, int from)

Функция отрисовки 3д модели

public void DrawModel()

# 3. ЗАКЛЮЧЕНИЕ И ВЫВОДЫ

В процессе выполнения курсовой работы были получены практические навыки по разработке приложения для отображения 3д модели с использованием Windows Forms, библиотек OpenGL и Tao Framework.

В частности:

* Отрисовки 3д модели
* Перемещение объекта
* Вращение объекта
* Изменения размеров объекта

# СПИСОК ЛИТЕРАТУРЫ

1. Уроки OpenGL + C# [Интернет ресурс] <http://esate.ru/uroki/OpenGL/uroki-OpenGL-c-sharp/>
2. OpenGL [Интернет ресурс] <https://ru.wikipedia.org/wiki/OpenGL>
3. Основы по 3D-моделированию в 3ds Max. [Интернет ресурс] <https://3dtoday.ru/blogs/cheap3d/the-basics-of-3d-modelling-in-3ds-max-part-1-create-and-print-piggy-ba>

# **ПРИЛОЖЕНИЕ А.**Исходный код

Главный файл 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 = 0, zoom = 1;

int os\_x = 1, os\_y = 0, os\_z = 0;

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\_ORIGIN\_SET);

Gl.glClearColor(0, 255, 255, 255);

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.glEnable(Gl.GL\_LINE\_SMOOTH);

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 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 trackBar5\_Scroll(object sender, EventArgs e)

{

zoom = (double)trackBar5.Value / 1000.0;

label6.Text = zoom.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();

}

}

}

}

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 = "";

// 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

//CreateList();

Gl.glCallList(thisList);

// возврат матрицы

Gl.glPopMatrix();

}

}

}

# ПРИЛОЖЕНИЕ Б Интерфейс программы

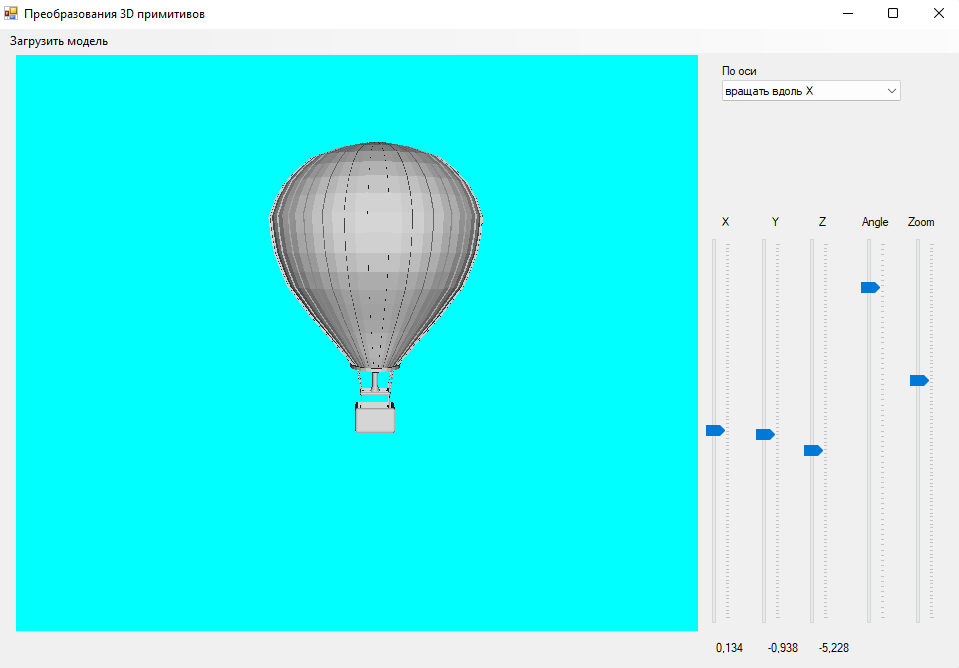


Рисунок Б.1 – Окно программы

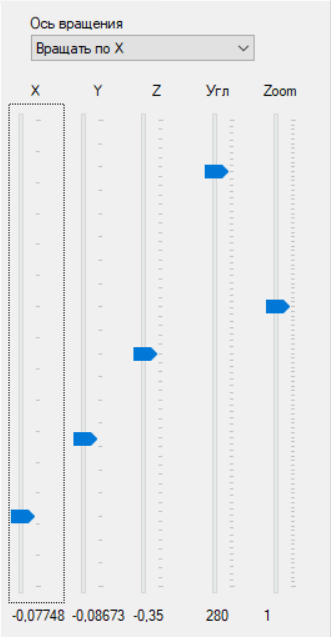


Рисунок Б.2 – Панель настроек

ПРИЛОЖЕНИЕ В Прототип модели



Рисунок В.1 – Прототип модели

Размещено на Allbest.ru