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 "Мотоцикл"

Выполнил: студент гр. Б720 Матхеева С.О.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель: ст. преподаватель Цагадаева Е. Н.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Улан-Удэ

2022

# **АННОТАЦИЯ**

Приложение написано на C# с использованием библиотеки OpenGL. Программа создана в среде Visual Studio 2022. В ней возможны различные настройки, такие как настройка перспективы и бликов, загрузка различных 3D-объектов с расширением «.ase», отображение их текстур. Программа имеет интуитивно понятный интерфейс, аналогичный другим Windows-приложениям, демонстрирующий общую функциональность программных продуктов.

Сама 3D-модель мотоцикла была создана в приложении 3Ds Max 2023, а все текстуры, включая сцену объекта, созданы и экспортированы в формате «.ase» с помощью этой программы.

# ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc122106111)

[1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OPENGL, TAO FRAMEWORK И GLUT 6](#_Toc122106112)

[1.1 Спецификация 6](#_Toc122106113)

[1.2 Дополнительные библиотеки 7](#_Toc122106114)

[1.3 Геометрические преобразования 9](#_Toc122106115)

[1.4 Формирование тел вращения 11](#_Toc122106116)

[1.5 Библиотека DEVIL 12](#_Toc122106117)

[2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТОРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА "Мотоцикл" 14](#_Toc122106118)

[2.1 Разработка окно программы 14](#_Toc122106119)

[2.2 Описание кода (классов и функций) 15](#_Toc122106120)

[3. ЗАКЛЮЧЕНИЕ И ВЫВОДЫ 18](#_Toc122106121)

[СПИСОК ЛИТЕРАТУРЫ 19](#_Toc122106122)

[ПРИЛОЖЕНИЕ А. Исходный код 20](#_Toc122106123)

[ПРИЛОЖЕНИЕ Б Интерфейс программы 43](#_Toc122106124)

# ВВЕДЕНИЕ

Сегодня 3D-изображения можно найти повсюду, от компьютерных игр до систем моделирования в реальном времени. До того, как 3D-графика существовала только в суперкомпьютерах, единого стандарта в области графики не существовало. Все программы создавались с нуля или с использованием накопленного опыта, но у каждой программы свой способ отображения графической информации.

С появлением мощных процессоров и графических ускорителей 3D-графика стала реальностью на персональных компьютерах. Однако в то же время производители программного обеспечения столкнулись с серьезными проблемами. Не существует стандарта , который бы делал программирование независимым от оборудования и операционной системы. Одним из первых стандартов, который существует до сих пор, является OpenGL.

OpenGL — это открытая библиотека. Это означает, что вы можете создавать и использовать программы, которые работают на любом компьютере под управлением Windows, Mac OS X или Linux. Вы также можете разрешить пользователям создавать свои собственные инструменты для создания графики с нуля. OpenGL это графический пользовательский интерфейс, предлагающий множество функций, которые делают его отличным инструментом для создания 3D-графики.

В настоящее время OpenGL является одним из самых популярных программных интерфейсов для разработки приложений с 2D- и 3D-графикой. В настоящее время реализация OpenGL включает в себя несколько библиотек (объясняющих базовую функциональность OpenGL, GLU, GLUT, GLAUX и т. д.).

OpenGL — это полнофункциональная открытая графическая библиотека для моделирования и рендеринга 3D-сцен. Создаваемый с 1992 года по настоящее время, он остается востребованным и по сей день. Основным конкурентом OpenGl был DirectX для компьютерных игр. Обе платформы хороши сами по себе, не лучше и не хуже. Мультиплатформенный характер этой библиотеки облегчает перенос программного обеспечения с одной операционной системы на другую.

3Ds Max — это программный пакет Autodesk, используемый для создания 3D-изображений с нуля. Эта программа сочетает в себе функционал сразу нескольких приложений, что может значительно ускорить процесс создания3D-моделей. В первую очередь, в 3Ds Max есть возможность создания трехмерной анимации, которая может использоваться в фильмах, мультфильмах, играх, а также в рекламе

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

OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый (независимый от языка программирования) программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

## 1.1 Спецификация

Основные возможности OpenGL:

1. Функции описания примитивов определяют нижнюю иерархию объектов (примитивов), которые могут быть отображены графической подсистемой. В OpenGL примитивами являются точки, линии, многоугольники и т. д. Примитив может быть как геометрическим объектом, так и изображением, состоящим из нескольких примитивов
2. Функции описания света используются для описания положения и параметров источников света в 3D-сцене. Они используются в сценариях освещения для указания типа освещения, используемого в сцене. Кроме того, они используются в параметрическом освещении для создания и редактирования источников света в вашей сцене.
3. Функция, которая устанавливает атрибуты. Программисты определяют, как отображаемый объект выглядит на экране, устанавливая атрибуты. Другими словами, если примитивы определяют, что отображается на экране, то атрибуты определяют, как отображается экран. В качестве атрибутов OpenGL позволяет задавать цвета, свойства материалов, текстуры и параметры освещения.
4. Функция визуализации позволяет задать положение наблюдателя в виртуальном пространстве, параметры объектива камеры. Зная эти параметры, система может не только правильно построить изображение, но и вырезать невидимые объекты. Для этого используется функция просмотра, она позволяет вам установить координаты наблюдателя и указать, как наблюдатель смотрит на камеру.
5. Набор функций геометрического преобразования позволяет программистам выполнять различные преобразования (поворот, перемещение, масштабирование) объектов. Эти функции позволяют создавать сложные 3D-модели, которые можно использовать для создания анимации, визуализаций и многого другого.

## 1.2 Дополнительные библиотеки

Существует ряд библиотек, созданных поверх или в дополнение к OpenGL. Например, библиотека GLU, являющаяся практически стандартным дополнением OpenGL и всегда её сопровождающая, построена поверх последней, то есть использует её функции для реализации своих возможностей.

Матричные операции используются в OpenGL для установки различных преобразований объектов сцены. Существует три типа матриц: матрицы вида модели, матрицы проекции и матрицы текстуры. Все они 4х4. Матрица вида определяет преобразования объекта в мировых координатах, такие как параллельный перенос, изменение масштаба и поворот. Матрица проекции определяет, как 3D-объект проецируется на плоскость экрана (в координатах окна), а матрица текстуры определяет, как текстура применяется к объекту.

Умножение координат на матрицу происходит путем вызова соответствующей команды OpenGL (обычно команды glVertex\*), определяющей координаты.

Чтобы выбрать матрицу для изменения, используйте следующие команды: void glMatrixMode(GLenum mode), GL\_MODELVIEW, GL\_PROJECTION или GL\_TEXTURE. Вызывается со значением параметра "mode", равным режиму, в котором выполняется манипулирование моделью. Отображать соответственно модельно-видовой матрицей, матрицей проекций, или матрицей текстуры. Для вызова команд, определяющих матрицы любого типа, необходимо сначала установить соответствующий режим.

Команда void glLoadMatrix[f d](GLtype \*m) вызывается для определения элементов матрицы текущего типа, где 'm' относится к массиву из 16 чисел с плавающей запятой или удвоений, в зависимости от имени команды. Сначала запишите первый столбец матрицы, затем второй, третий и четвертый столбцы. Опять же, обратите внимание, что в массиве ‘m’ матрица записывается столбцами. Команда void glLoadIdentity(void) заменяет текущую матрицу идентификационной матрицей.

Библиотека набора утилит OpenGL (GLUT, от GL Utility Toolkit) - это интерфейс для программистов, позволяющий создавать независимые от окон приложения ANSI C и FORTRAN OpenGL. Эта библиотека была создана Марком Дж. Килгардом и заполняет большой пробел, оставленный спецификацией OpenGL. Благодаря разработчикам GLUT теперь мы можем использовать единый интерфейс для работы с окнами независимо от платформы. Приложения OpenGL, использующие GLUT, можно легко портировать между платформами без существенных изменений исходного кода. GLUT значительно упрощает написание кода приложения и дополняет библиотеку OpenGL.

GLUT API, как и сам OpenGL, является конечным автоматом (state machine). Это означает, что GLUT содержит множество переменных состояния, которые изменяются во время работы приложения. Начальное состояние машины GLUT выбирается для большинства приложений. Программы могут изменять значения этих переменных состояния в соответствии со своими конкретными требованиями. Каждый раз, когда вызывается функция GLUT, она ведет себя по-разному в зависимости от значения переменной состояния. Функции GLUT просты и требуют минимальных параметров. Они не возвращают указатели. Единственными указателями в параметрах функции являются указатели на текстовые строки и дескрипторы шрифтов.

Функции GLUT можно разделить на несколько групп в зависимости от их функциональности:

* Инициализация
* Начало обработки событий
* Управление окнами
* Управление перекрытием
* Управление меню
* Регистрация вызываемых (callback) функций
* Управление индексированной палитрой цветов
* Чтение состояния
* Отображение шрифтов
* Отображение геометрических фигур

Все функции GLUT начинаются с префикса переполнения (например, glutPostRedisplay помечает текущее окно как требующее перерисовки). В OpenGL API есть команды для перевода модели: glTranslate\*(), glRotate\*(), glScale\*(). Эти команды можно использовать для выполнения преобразований объекта или системы координат. Эти команды являются эквивалентами матриц, которые вызывают функции glMultMatrix\*() с требуемой матрицей в качестве аргумента. Однако в этом случае быстрее использовать команды glTranslate, glRotate и glScale. GlScale\*() — это всего лишь одна из трех команд преобразования модели, которые изменяют размеры объектов.

## 1.3 Геометрические преобразования

Геометрическое преобразование — это преобразование графического объекта, которое изменяет ориентацию, масштаб и положение системы координат, используемой для размещения точек в пространстве. При этом само пространство не меняется. Структура самого графического изображения сохраняется. Таким образом, для геометрических преобразований тип объекта не имеет значения, поскольку в преобразовании участвуют точки.

Основные операции, связанные с геометрическими преобразованиями 2D графических объектов:

* Перенос начала координат
* Масштабирование
* Поворот осей координат

Рассмотрим эти операции более подробно.

**Перенос**

Эта операция перемещает начало координат на фиксированную величину. Пусть старая система координат будет CK1, а новая система координат CK2. При передаче новое начало системы координат CK1 находится в точке (Tх, Tу), а старое начало новой системы координат CK2 — в точке (-Tх, - Tу). Началом (x, y) системы CK1 в системе CK2 является точка (x- Tх, y- Tу).

Обратим внимание, что и ориентация, и масштаб гномона сохраняются во время перемещения.

Вот матрица, необходимая для выполнения операции переноса:

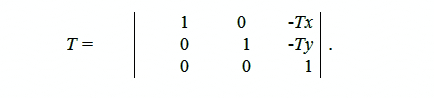


Рисунок 1 - Геометрическое преобразование "Перенос"

**Масштабирование**

После выполнения операции масштабирования отрезок единичной длины по оси X становится отрезком длины Sх по оси X, а отрезок единичной длины по оси Y становится отрезком по оси Y Это будет отрезок длины Sу. В этой системе точка с координатами (1,1) преобразуется в точку (Sх, Sу), а точка (x, y) становится с координатами (x\* Sх, y\* Sу).

Стоит обратить внимание, что если старая и новая системы координат имеют одинаковый масштаб, начало координат и оси, изменится только масштаб по осям.

Матрицы, необходимые для выполнения операции масштабирования:

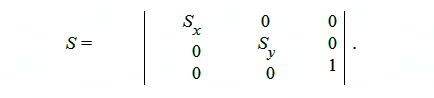


Рисунок 2 - Геометрическое преобразование "Масштабирование"

**Поворот**

Рассмотрим конкретный случай вращения в пространстве XYZ. То есть вращение вокруг оси Z, проходящей перпендикулярно плоскости XY.

Предположим, что центр вращения совпадает с началом координат. Тогда, если систему координат XY повернуть в декартовой плоскости на произвольный угол w вокруг центра вращения, координаты любой точки C в системе координат (x', y') будут в исходной системе координат (x, y) координаты этой точки:

x' = x cos w + y sin w,

y' = -x sin w + y cos w

Для выполнения операции поворота используется следующая матрица:

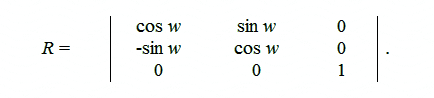


Рисунок 3. Геометрическое преобразование "Поворот".

## 1.4 Формирование тел вращения

Одним из способов формирования сложных трехмерных изображений на экране является вращение скелетных тел. В этом случае выбирается ось вращения и в той же плоскости, что и эта ось, формируется кривая, обычно аппроксимируемая серией отрезков прямой линии. Вращение кривой вокруг выбранной оси, считывание определенного угла поворота и фиксация положения этой кривой создает скелетное вращение.

С некоторой криволинейной геометрией, определяемой серией точек, вы можете построить объект на основе вращения указанной криволинейной геометрии. Разбивая, таким образом, объект на заданные N кривых и соединяя их вершины многоугольниками между кривыми, мы получаем оболочку тела вращения.

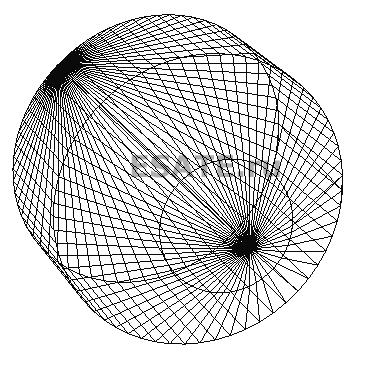


Рисунок 4. Пример объекта, построенного как тело вращения.

Количество сегментов и подразделений исходной кривой определяет гладкость объекта. Поэтому большое количество полигонов сильно нагружает графический адаптер.

## 1.5 Библиотека DEVIL

DevIL (библиотека изображений для разработчиков) — это кроссплатформенная программная библиотека, целью которой является предоставление общего интерфейса прикладного программирования для различных форматов файлов изображений. Первоначально эта библиотека называлась «OpenIL». DevIL состоит из трех частей: основной библиотеки, сервисной библиотеки и сервисного инструментария.

Работа с текстурами с использованием библиотеки DevIL. В настоящее время библиотека поддерживает работу с изображениями в 43 форматах чтения и 17 форматах записи.

Чтобы использовать наложения текстур, необходимо выполнить следующие шаги:

* Создайте объект текстуры и установите текстуру.
* Установить, как текстура влияет на каждый пиксель.
* Включить текстурный движок.
* Рендеринг сцены и передача конвейера рендеринга, координаты геометрии и координаты текстуры.

# 2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТОРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА "Водный скутер"

## 2.1 Разработка окно программы

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

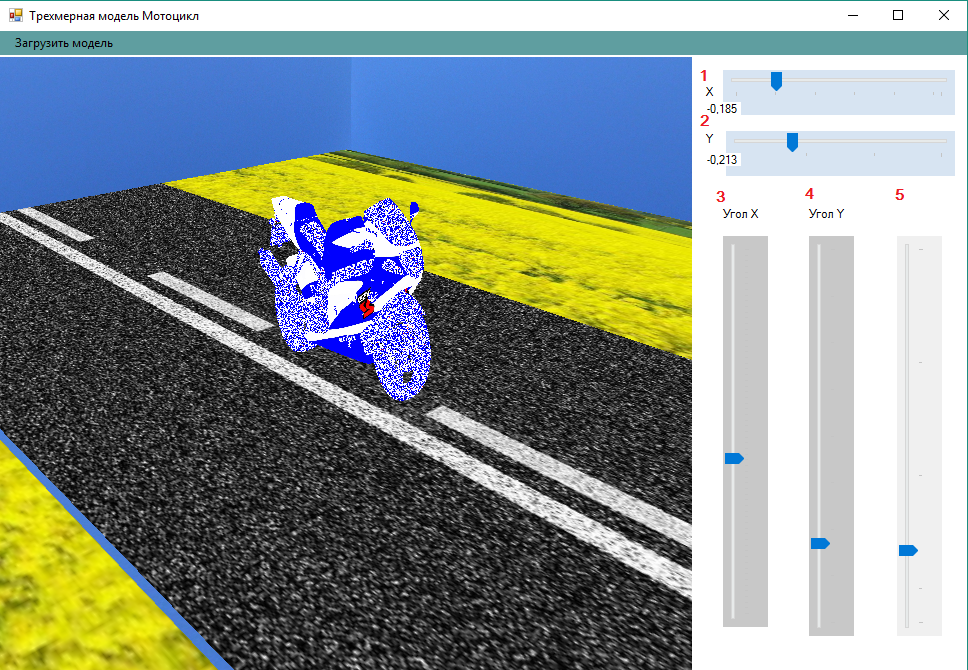


Рисунок 5 - Окно программы

Данный интерфейс имеет настройки:

1. Перемещение модели по оси X (влево, вправо)
2. Перемещение модели по оси Y (вверх, вниз)
3. Перемещение модели по двум осям (угол X)
4. Перемещение модели по двум осям (угол Y)
5. Увеличения или уменьшение модели

## 2.2 Описание кода (классов и функций)

Главный файл Form1.cs используется для описания инструментов окна

Главная функция. С неё 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 trackBar1\_Scroll(object sender, EventArgs e)

Функция изменения координаты Y для модели

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

Функция поворота камеры по Х для модели

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

Функция поворота камеры по Y для модели

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

Функция зума

private void trackBar5\_Scroll(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()

Файлы, загруженные в программу, являются управляющими структурами. Ключевые слова, указывающие на новые структуры управления, отмечены знаком «\*» в начале слова.

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

1. Прочитайте файл построчно (желательно построчно, так как вся информация, необходимая для структуры управления, помещается на одной строке).

2. Сократите первое слово строки.

3. Проверьте первую букву первого полученного слова. Если оно равно '\*', то есть управляющее слово.

4. На основе контрольного слова проверяем, что это за контрольное слово, и если оно относится к чему-то, что вы хотите (геометрия, описание координат текстуры), обработайте эту строку и обрежьте первое слово, а затем переведите. Строковое представление числа.

# 3. ЗАКЛЮЧЕНИЕ И ВЫВОДЫ

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

В частности:

* Получение навыков программирования динамических трехмерных интерактивных сцен
* 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 = -80, zoom = 1, i=-100;

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

Gl.glLineWidth(1.0f);

// 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.glTranslated(-a, -b, -c);

Gl.glRotated(i, 0, 0, 1);

Gl.glTranslated(a, b, c);

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 label8\_Click(object sender, EventArgs e)

{

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void label4\_Click(object sender, EventArgs e)

{

}

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

{

}

private void menuStrip1\_ItemClicked(object sender, ToolStripItemClickedEventArgs e)

{

}

private void openFileDialog1\_FileOk(object sender, CancelEventArgs e)

{

}

private void label6\_Click(object sender, EventArgs e)

{

}

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

{

}

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

{

i = (double)trackBar3.Value;

Draw();

}

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

{

d = (double)trackBar4.Value;

Draw();

}

private void trackBar5\_Scroll(object sender, EventArgs e)

{

zoom = (double)trackBar5.Value / 100.0;

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();

}

}

}

}

Файл для считывания 3д модели из файла .ase расширения

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();

}

}

}

# ПРИЛОЖЕНИЕ Б Интерфейс программы

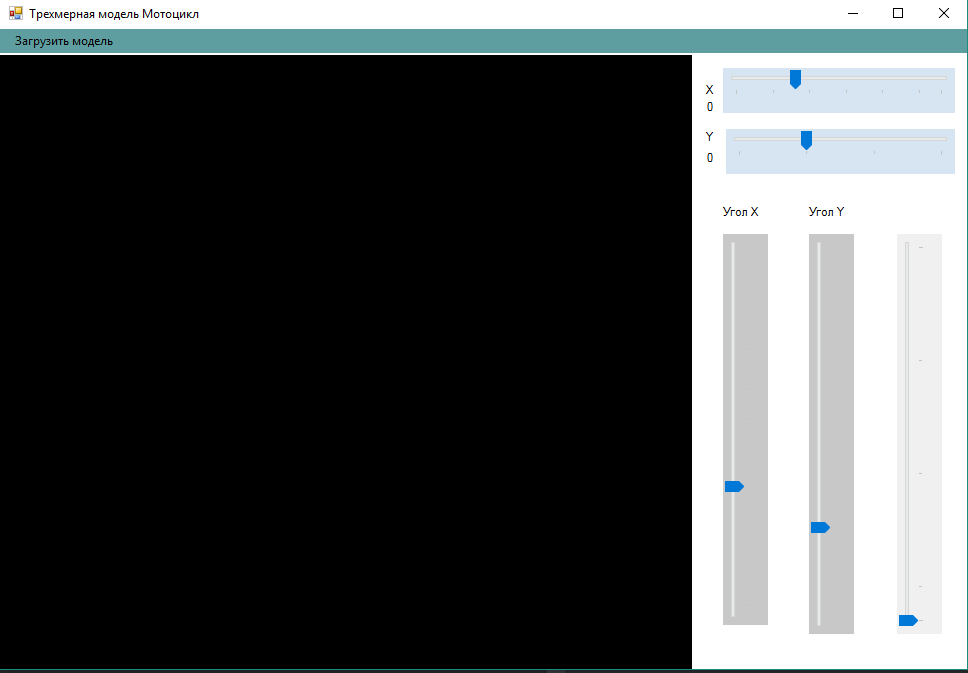


Рисунок 6 – Интерфейс программы

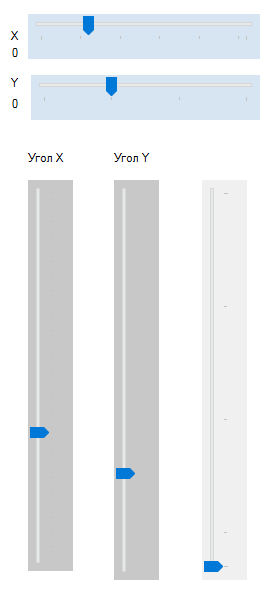


Рисунок 7 – Панель настроек

**ПРИЛОЖЕНИЕ В Прототип модели**



Рисунок 9 – Прототип модели (Suzuki GSXR 750)



Рисунок 8 – отображение мотоцикла в 3Ds max