МИНЕСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Восточно-Сибирский государственный университет технологий и управления»

Кафедра «Программная инженерия и искусственный интеллект»

**КУРСОВАЯ РАБОТА**  
на тему «Разработка динамического изображения трехмерной модели объекта “Велосипед”»

по дисциплине «Геометрия топология»

Выполнили: студенты гр. Б720 Давыдов И.С

Руководитель: СТ. Цагадаева Е.Н.

Оценка \_\_\_\_\_\_\_\_\_\_\_

Дата защиты: 17.12.2022 г.

Улан-Удэ

2022

**Аннотация**

Приложение написано на языке C# с использованием библиотек OpenGl и TaoFramework, а также с помощью профессионального программного обеспечения 3ds Max. Программа создана в среде Visual Studio 2019. В рамках работы были выполнены все представляемые задачи. Программа имеет возможность для выбора перспективы, с помощью перемещения модели по осям XYZ. Программа имеет интуитивно понятный интерфейс, который схож с другими Windows.

**Содержание**

АННОТАЦИЯ

ВВЕДЕНИЕ

1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OPENGL

1.1 Основные возможности

1.2 Работа с матрицами

1.3 Проекции

2. Моделирование с использованием 3ds Max

2.1 Наложение текстур на модель

2.2 Создание трехмерной модели «Танк»

3. Разработка интерфейса пользователя для построения динамического

изображения трехмерной модели объекта «ТАНК»

3.1 Разработка интерфейса

3.2 Разработка подсистемы управления событиями

4. Информационное и программное обеспечение

4.1 Общие сведения о программе

4.2 Функциональное назначение

4.3 Логическая структура и функциональная декомпозиция проекта

5. Заключение и выводы

СПИСОК ЛИТЕРАТУРЫ

ПРИЛОЖЕНИЯ

## **ВВЕДЕНИЕ**

В наше время 3D моделирование используется во многих сферах деятельности. В кино, в играх, на производстве и т.д. Потому чтобы научиться работать с 3D объектами, нами был выбран проект реализации модели танка на базе WindowsForms в среде программирования Visual Studio 2019. Также при выполнении задачи были использованы библиотеки OpenGL.

**OpenGl**

OpenGL – это графический стандарт в области компьютерной графики. На данный момент он является одним из самых популярных графических стандартов во всём мире.

Tao Framework – это свободно-распространяемая библиотека с открытым исходным кодом, предназначенная для быстрой и удобной разработки кроссплатформенного мультимедийного программного обеспечения в среде .NET Framewrok и Mono. На сегодняшний день Tao Framework - это лучший путь для использования библиотеки OpenGL при разработке в среде .NET на языке C#.

Что же представляет из себя OpenGL? С точки зрения программиста OpenGL – это программный интерфейс для графических устройств, таких как графические ускорители. Он включает в себя около 150 различных команд, с помощью которых программист может определять различные объекты и производить рендеринг. Поэтому можно сказать, что библиотека OpenGL является только воспроизводящей, и занимается только отображением 3D обьектов, она не работает с устройствами ввода. OpenGL имеет хорошо продуманную внутреннюю структуру и довольно простой процедурный интерфейс.

OpenGL переводится как Открытая Графическая Библиотека (Open Graphics Library), это означает, что OpenGL – это открытый и мобильный стандарт. Программы, написанные с помощью OpenGL можно переносить практически на любые платформы, получая при этом одинаковый результат, будь это графическая станция или суперкомпьютер. OpenGL освобождает программиста от написания программ для конкретного оборудования, поэтому не составило труда перенести и улучшить наш проект в среду 3ds Max.

**3ds Max**

3ds Max – профессиональное программное обеспечение для 3D – моделирования, [анимации](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B8%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) и визуализации при создании [игр](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D1%91%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0) и проектировании. В настоящее время разрабатывается и издается компанией [Autodesk](https://ru.wikipedia.org/wiki/Autodesk" \o "Autodesk).

Первая версия пакета под названием 3D Studio DOS была выпущена в [1990 году](https://ru.wikipedia.org/wiki/1990_%D0%B3%D0%BE%D0%B4). Тогда разработками пакета занималась независимая студия Yost Group, созданная [программистом](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82) Гари Йостом. Autodesk на первых порах занимался только изданием пакета. Существуют сведения, что Гари Йост покинул прежнее место работы после переговоров с Эриком Лайонсом (Eric Lyons), в то время директором по новым проектам Autodesk.

## **1.ПРОГРАММИРОВАНИЕ С ИС ПОЛЬЗОВАНИЕМ OPENGL**

## **1.1 Основные возможности**

Возможности OpenGL описаны через функции его библиотеки. Все функции можно разделить на пять категорий.

Функции описания примитивов определяют объекты нижнего уровня иерархии (примитивы), которые способна отображать графическая подсистема. В OpenGL в качестве примитивов выступают точки, линии, многоугольники и т.д.

Функции описания источников света служат для описания положения и параметров источников света, расположенных в трехмерной сцене.

Функции задания атрибутов. С помощью задания атрибутов программист определяет, как будут выглядеть на экране отображаемые объекты. Другими словами, если с помощью примитивов определяется, что появится на экране, то атрибуты определяют способ вывода на экран. В качестве атрибутов OpenGL позволяет задавать цвет, характеристики материала, текстуры, параметры освещения.

Функции визуализации позволяет задать положение наблюдателя в виртуальном пространстве, параметры объектива камеры. Зная эти параметры, система сможет не только правильно построить изображение, но и отсечь объекты, оказавшиеся вне поля зрения.

Набор функций геометрических преобразований позволяют программисту выполнять различные преобразования объектов - поворот, перенос, масштабирование. При этом OpenGL может выполнять дополнительные операции, такие как использование сплайнов для построения линий и поверхностей, удаление невидимых фрагментов изображений, работа с изображениями на уровне пикселей 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.

Для определения элементов матрицы текущего типа вызывается команда void glLoadMatrix[f d](GLtype \*m), где "m" указывает на массив из 16 элементов типа float или double в соответствии с названием команды, при этом сначала в нем должен быть записан первый столбец матрицы, затем второй, третий и четвертый. Еще раз следует обратить внимание, в массиве "m" матрица записана по столбцам. Команда void glLoadIdentity(void) заменяет текущую матрицу на единичную.

**1.3 Проекции**

В OpenGL существуют стандартные команды для задания ортографической (параллельной) и перспективной проекций. Первый тип проекции может быть задан командами void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far) и void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top).

Первая команда создает матрицу проекции в усеченный объем видимости (параллелепипед видимости) в левосторонней системе координат. Параметры команды задают точки (left, bottom, znear) и (right, top, zfar), которые отвечают левому нижнему и правому верхнему углам окна вывода. Параметры "near" и "far" задают расстояние до ближней и дальней плоскостей отсечения по удалению от точки (0,0,0) и могут быть отрицательными.

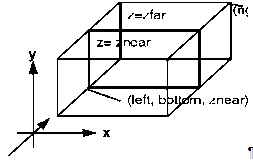


Рис. 1 – Ортографическая проекция

Перспективная проекция определяется командой void gluPerspective(GLdouble angley, GLdouble aspect, GLdouble znear, GLdouble zfar), которая задает усеченный конус видимости в левосторонней системе координат. Параметр "angley" определяет угол видимости в градусах по оси у и должен находиться в диапазоне от 0 до 180. Угол видимости вдоль оси x задается параметром "aspect", который обычно задается как отношение сторон области вывода (как правило, размеров окна). Параметры "zfar" и "znear" задают расстояние от наблюдателя до плоскостей отсечения по глубине и должны быть положительными. Чем больше отношение zfar/znear, тем хуже в буфере глубины будут различаться расположенные рядом поверхности, так как по умолчанию в него будет записываться "сжатая" глубина в диапазоне от 0 до 1.

Прежде чем задавать матрицы проекций, нужно включить режим работы с нужной матрицей командой glMatrixMode(GL\_PROJECTION) и сбросить текущую, вызвав glLoadIdentity().[5]

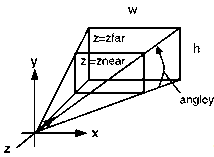


Рис. 2 – Перспективная проекция

**2. МОДЕЛИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ 3DS MAX**

**2.1 Наложение текстур на модель**

Благодаря 3ds Max наложение текстур на модель не составило никакого труда. Перенеся модель в это ПО и выделив желаемые части объекта, нажав на горячую клавишу “M”, мы вызвали редактор текстур. Затем лишь взяли подготовленные заранее текстуры установили их на модель.

Взяв фотографию из сети интернет, мы смогли затекстурировать наш объект, тем самым мы достигли качества, которого не смогли бы достичь, используя лишь библиотеки OpenGl.

**2.2 Создание трехмерной модели «Танк»**

Opengl позволяет создавать множество 3D – объектов, однако мы решили использовать 3ds Max для создание более сложных графических сцен. Таким образом создав трехмерный объект мы можем сохранить его, и посредством экспорта получить файл, который хранит в себе геометрические данные, текстурные координаты и т.д в зависимости от настроек. При создании использовались модифицированные графические примитивы.

Загрузку трехмерной модели мы производим в формате ASE. В данный формат можно легко сохранить трехмерную модель, используя пакет трехмерного моделирования 3D Studio Max. Формат является текстовым.

**3. РАЗРАБОТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ ДЛЯ ПОСТРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА «ТАНК»**

**3.1 Разработка интерфейса**

Для данной программы разработан интерфейс, позволяющий:

* загрузить любую модель формата “ASE”;
* вращать модель;
* передвигать модель по осям XYZ;
* приближать и отдалять модель;

Также добавлен пункт «По оси», позволяющий выбирать ось, по которой будет двигаться модель.

**3.2 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:

* WM\_DESTROY - освобождение занятых ресурсов;
* WM\_SIZE - изменения сцены относительно размеров окна;
* WM\_ERASEBKGND - предотвращения мерцания;
* WM\_TIMER – используется для создания таймера;
* WM\_MOUSEWHEEL – обработка вращения колеса мышки;
* WM\_KEYDOWN – обработка нажатия клавиши;
* COMMAND(ID\_PERSP, CKarkasView::OnPersp ()) – обработка события при вызове окна настройки перспективы;
* COMMAND(ID\_OPTIONS, CKarkasView::OnOptions ()) – обработка события при вызове окна настройки типа вращения и скорости движения объекта;
* COMMAND(ID\_VIEW\_1, CKarkasView::OnView1())– обработка события выбора типа тумана;
* COMMAND(ID\_VIEW\_SBROS, CKarkasView:: OnVewSbros()) – обработка события нажатия кнопки "Убрать туман";
* COMMAND (ID\_VIEW\_MOVE, CKarkasView::OnMove ()) – обработка нажатия кнопки "Погружение".

**4. ИНФОРМАЦИОННОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ**

**4.1 Общие сведения о программе**

Программа называется "Трехмерная модель «Танк»". При работе с данной программой у пользователя есть возможность работать с визуальной моделью данного объекта.

Программное обеспечение, на котором разработано приложение – Microsoft Visual С# 2019.

**4.2 Функциональное назначение**

Данная программа предназначается для представления трехмерной модели батискафа. Приложение дает следующие возможности:

* загружать модель с заранее прописанными текстурами;
* наблюдать модель;
* вращать Танк;
* приближать и удалять объект с ползунка;

**4.3 Логическая структура и функциональная декомпозиция проекта**

Загружаемый в программу файл представляет собой управляющие конструкции. Ключевые слова, указывающие на новую управляющую конструкцию, отмечены знаком «\*» в начале слова.

Алгоритм чтения файла и организация файла модели в памяти программы будут выглядеть 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 мы проверяем, что это за управляющее слово, и если оно относится к тем, которые необходимы нам (описывают геометрию, текстурные координаты), то мы обрабатываем данную строку, отрезая первые слова и затем переводя их из строкового представления к числовому.

Управляющие слова, которые нас интересуют:

\*MATERIAL\_COUNT – количество текстур, используемое в данной 3D-сцене. Если значение этого параметра не равно 0, то дальше будет идти описание каждого материала. Среди параметров ambient, diffuse, specular, которые отражают свойства материала (и обработку которых мы не рассматривали в рамках данной программы) в блоке описания материала будет также указана информация о текстуре.

\*MATERIAL – начало блока описания материала. Далее следует указание номера материала.

\*BITMAP – адрес текстуры для данного материала.

\*MATERIAL\_REF – указание на то, какой номер материала назначен данному объекту.

\*GEOMOBJECT – указывает на новый геометрический подобъект в загружаемой из файла трехмерной сцене. Этот блок также описывает довольно большое количество информации – имя объекта, его начальные координаты, ориентация и т.д.

\*MESH\_NUMVERTEX – указывает на количество вершин, которые будут описывать геометрию данного 3D-объекта (подобъекта). Мы будем использовать это значение (и значение, описывающие количество полигонов) для выделения памяти при создании экземпляра, описывающего геометрию подобъекта.

\*MESH\_NUMFACES – указывает количество полигонов, в данном геометрическом подобъекте.

\*MESH\_NUMTVERTEX, \*MESH\_NUMTVFACES – описание текстурных координат и их привязка к полигонам.

\*MESH\_VERTEX, \*MESH\_FACE, \*MESH\_TVERT, \*MESH\_TFACE – непосредственное описание вершин и полигонов (геометрии и текстурных координат). Самые часто встречаемые ключевые слова.

За данными ключевыми словами следует ряд параметров. В случае вершин - это номер вершины и три ее координаты. В случае описания полигонов – это номер полигона (после которого стоит знак «:», который необходимо отрезать) и далее номера вершин, координаты которых должны быть использованы для построения полигона.

Теперь, разобравшись с данными в описании 3D-модели, мы переходим к алгоритму загрузки 3D-модели, а также ее визуализации. Первым делом, объявляются необходимые для дальнейшей работы переменные, массивы для объектов и т.д.

Функция загрузки 3D модели, как уже говорили, это построчное чтение файла и поиск управляющих слов, а также последующее внесение описываемых ими данных в объекты описания геометрии и создание и загрузка текстурных объектов в память OpenGL.

Последний этап - это генерация дисплейного списка. Мы сгенерировали его номер, затем начали создание дисплейного списка и вызвали функцию CreateList();. В данной функции производилась визуализация полученной геометрии 3D-модели с учетом установленных текстур, текстурных координат.

В цикле отрисовываем массивы вершин, перебирая массив подобъектов, затем массив полигонов. В случае необходимости мы включаем режим текстурирования и указываем текстурные координаты.

**5. ЗАКЛЮЧЕНИЕ И ВЫВОДЫ**

В ходе разработки данного приложения были получены практические навыки по разработке программ для операционных систем семейства Windows с применением технологий трехмерной графики с использованием библиотеки OpenGL.

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

* изучение принципов работы OpenGL в оконной среде Windows;
* изучение принципов работы 3ds max;
* получение практических навыков использования средств OpenGL;
* получение навыков программирования динамических трехмерных сцен;
* получение навыков программирования интерактивных трехмерных приложений.

**СПИСОК ЛИТЕРАТУРЫ**

1. Уроки OpenGL и C# (си шарп) для начинающих (esate.ru)
2. The Tao Framework download | SourceForge.net
3. Уроки 3Ds Max - моделирование – YouTube

**ПРИЛОЖЕНИЕ**

**Приложение А**

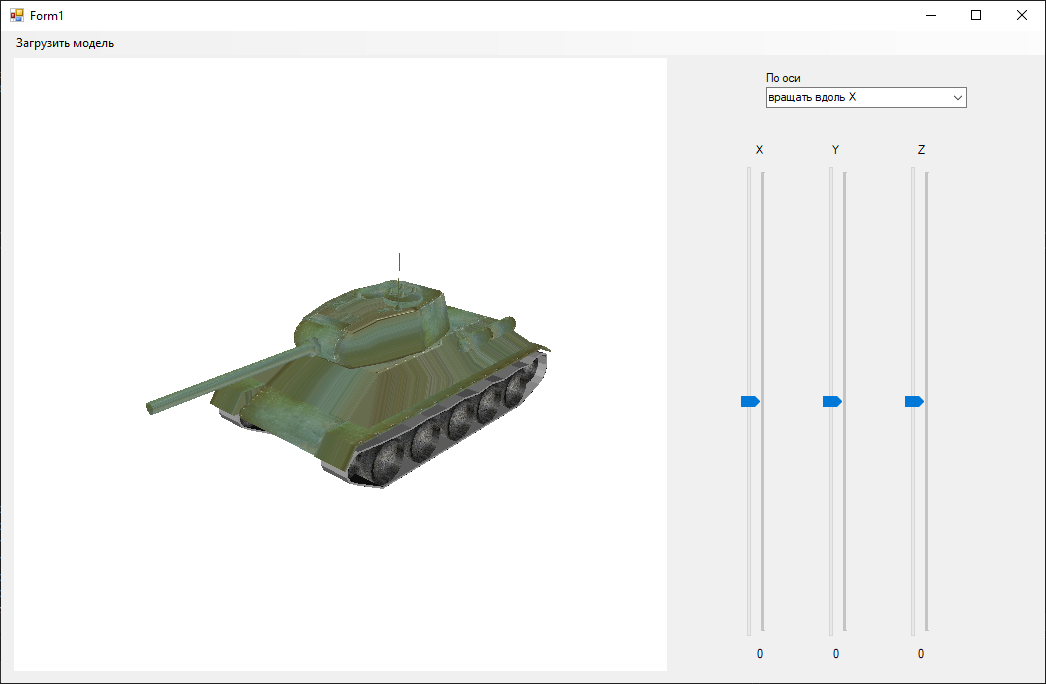


Рис. 3 – Танк Т-34, получившийся в результате нашей работы.



Рис. 4 – Танк Т-34, в среде моделирования 3ds Max.

**Приложение Б**

**Form1:**

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 Курсовой\_2.\_0

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

AnT.InitializeContexts();

}

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();

}

double a = 0, b = 0, c = -5, d = -80, zoom = 1;

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

bool Wire = false;

Model.anModelLoader Model = null;

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();

}

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

{

Draw();

}

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(255, 255, 255, 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);

comboBox1.SelectedIndex = 0;

// опиции для загрузки файла

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

}

private void AnT\_Keydown(object sender, KeyEventArgs e)

{

}

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

{

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

{

Model = new Model.anModelLoader();

Model.LoadModel(openFileDialog1.FileName);

RenderTimer.Start();

}

}

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();

}

}

}

**Class1:**

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 нормаль

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();

}

}

}