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 "Самолёт"

Выполнил: студент гр. Б720 Русаков С.А.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель: ст. преподаватель Цагадаева Е. Н.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Улан-Удэ

2022

## АННОТАЦИЯ

Приложение написано на языке С# с использованием библиотеки OpenGL. Программа создана в среде Visual Studio 2022. В рамках работы были выполнены все представляемые задачи. Имеется возможность увеличивать и уменьшать объект, изменять его угол поворота и вертеть объект по 3 координатам x,y,z. Программа имеет интуитивно понятный интерфейс.

**ОГЛАВЛЕНИЕ**

АННОТАЦИЯ

ВВЕДЕНИЕ

1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OPENGL, FREEGLUT, DEVIL

1.1 Основные возможности

1.2 Работа с матрицами

1.3 Проекции

1.4 Освещение

2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА "САМОЛЕТА"

2.1 Разработка процедуры визуализации трехмерной сцены

2.2 Разработка интерфейса пользователя

2.3 Разработка подсистемы управления событиями

3. ИНФОРМАЦИОННОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

3.1 Общие сведения о программе

3.2 Функциональное назначение

3.3 Логическая структура и функциональная декомпозиция проекта

3.4 Требования к техническому и программному обеспечению

3.5 Руководство пользователя

4. ТЕСТИРОВАНИЕ

5. ЗАКЛЮЧЕНИЕ И ВЫВОДЫ

СПИСОК ЛИТЕРАТУРЫ

ПРИЛОЖЕНИЯ

## ВВЕДЕНИЕ

На сегодняшний день 3D-визуализация играет важную роль в жизни общества. Мы сталкиваемся с ней ежедневно: рекламные ролики, компьютерные игры, фильмы и мультфильмы с применением трёхмерной графики... Этот список можно продолжать очень долго, что наводит на мысль о действительно огромном потенциале использования этого инструмента. OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый (независимый от [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F" \o "Язык программирования)) [программный интерфейс](https://ru.wikipedia.org/wiki/API" \o "API) для написания приложений, использующих двумерную и трёхмерную [компьютерную графику](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0" \o "Компьютерная графика). Разрабатывается в США и Европе, имеет тип лицензий GNU-/EU/. Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%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" \o "Компьютерная игра), [САПР](https://ru.wikipedia.org/wiki/%D0%A1%D0%90%D0%9F%D0%A0" \o "САПР), [виртуальной реальности](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C" \o "Виртуальная реальность), визуализации в научных исследованиях. На платформе [Windows](https://ru.wikipedia.org/wiki/Windows" \o "Windows) конкурирует с [Direct3D](https://ru.wikipedia.org/wiki/Direct3D" \o "Direct3D). Одним из первых таких стандартов, существующий и по сей день является OpenGL. Как известно, OpenGL расшифровывается как Open Graphics Library, что в переводе на русский язык означает «открытая графическая библиотека». Другими словами, OpenGL - это некая спецификация, включающая в себя несколько сотен функций. Она определяет независимый от языка программирования кросс-платформенный программный интерфейс, с помощью которого программист может создавать приложения, использующие двухмерную и трехмерную компьютерную графику. Первая базовая версия OpenGL появилась в 1992 году – она была разработана компанией Silicon Graphics Inc., занимающейся разработками в области трехмерной компьютерной графики.В современном мире, библиотеки OpenGL и DirectX являются конкурентами на платформе MS Windows. Microsoft всеми силами продвигает свою библиотеку DirectX, а также всеми силами стремится замедлить развитие библиотеки OpenGL, что ослабило бы графическую систему конкурирующих ОС, где для реализации вывода всей графики используется исключительно библиотека OpenGL. Мы будем учиться визуализации компьютерной графики именно с применением этой библиотеки. Но, к сожалению, прямой поддержки данной библиотеки в .NET Framework нет. Поэтому мы будем использовать библиотеку Tao Framework.Tao Framework – это свободно-распространяемая библиотека с открытым исходным кодом, предназначенная для быстрой и удобной разработки кросс-платформенного 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#. В состав библиотеки на данный момент входят все современные средства, которые могут понадобиться в ходе разработки мультимедиа программного обеспечения: реализация библиотеки OpenGL, реализация библиотеки FreeGlut, содержащей все самые новые функции этой библиотеки, библиотека DevIL (легшая в основу стандарта OpenIL – Open Image Library) и многое другое.

1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OPENGL, FREEGLUT, DEVIL

1.1 Основные возможности

Описывать возможности OpenGL мы будем 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.

FREEGLUT — позволяет пользователю создавать окна, предоставляющие контекст OpenGL на широком спектре платформ, и управлять ими, а также взаимодействовать с [мышью](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BC%D1%8B%D1%88%D1%8C" \o "Компьютерная мышь), [клавиатурой](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D0%B2%D0%B8%D0%B0%D1%82%D1%83%D1%80%D0%B0" \o "Клавиатура) и [джойстиком](https://ru.wikipedia.org/wiki/%D0%94%D0%B6%D0%BE%D0%B9%D1%81%D1%82%D0%B8%D0%BA" \o "Джойстик).

DevIL  — [кроссплатформенная](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5" \o "Кроссплатформенное программное обеспечение) [программная](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0" \o "Компьютерная программа) [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)" \o "Библиотека (программирование)), целью которой является создание общего [интерфейса программирования приложений](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9" \o "Интерфейс программирования приложений) для разных [файловых форматов](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D1%82_%D1%84%D0%B0%D0%B9%D0%BB%D0%B0" \o "Формат файла) графических изображений.

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.

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().

1.4 Освещение

В OpenGL используется модель освещения, в соответствии с которой цвет точки определяется несколькими факторами: свойствами материала и текстуры, величиной нормали в этой точке, а также положением источника света и наблюдателя. Для корректного расчета освещенности в точке надо использовать единичные нормали, однако команды: типа glScale\*(), могут изменять длину нормалей. Чтобы это учитывать, нужно использовать режим нормализации векторов нормалей, который включается вызовом команды glEnable(GL\_NORMALIZE) .

Для задания глобальных параметров освещения используются команды void glLightModel[i, f](GLenum pname, GLenum param) и void glLightModel[i f]v(GLenum pname, const GLtype \*params).Аргумент "pname" определяет, какой параметр модели освещения будет настраиваться и может принимать следующие значения: GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, параметр "param" должен быть булевым и задает положение наблюдателя. Если он равен GL\_FALSE, то направление обзора считается параллельным оси z, вне зависимости от положения в видовых координатах. Если же он равен GL\_TRUE, то наблюдатель находится в начале видовой системы координат. Это может улучшить качество освещения, но усложняет его расчет. Значение по умолчанию – GL\_FALSE.

GL\_LIGHT\_MODEL\_TWO\_SIDE параметр "param" должен быть булевым и управляет режимом расчета освещенности, как для лицевых, так и для обратных граней. Если он равен GL\_FALSE, то освещенность рассчитывается только для лицевых граней. Если же он равен GL\_TRUE, расчет проводится и для обратных граней. Значение по умолчанию – GL\_FALSE.

GL\_LIGHT\_MODEL\_AMBIENT параметр "params" должен содержать четыре целых или вещественных числа, которые определяют цвет фонового освещения даже в случае отсутствия определенных источников света. Значение по умолчанию – (0.2, 0.2, 0.2,1.0).

1. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА "САМОЛЕТА"
   1. Разработка процедуры визуализации трехмерной сцены

Класс LIMB отвечает за логические единицы 3D объектов в загружаемой сцене, в нем инициализируется количество вершин и полигонов нашего объекта, а также массивы для хранения вершин, полигонов координаты текстуры.

Класс TexturesForObjects работает с загрузкой текстуры, функция LoadTextureForModel загружает текстуру, т.е. запоминает ее имя и создает изображение.

Класс anModelLoader выполняет загрузку 3D модели. В функции LoadModel описана загрузка 3D модели - 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.

Функция CreateList отрисовывает нашу 3D модель по полученным данным из файла модели, после получения данных рисуются вершины и текстурные координаты каждой вершины.

Весь код загрузки текстуры, модели и их отрисовки представлен в приложении A.

* 1. Разработка интерфейса пользователя

Для данной программы разработан интерфейс, позволяющий:

- вращать модель по 3 осям x,y,z;

- изменять угол модели

- увеличивать или уменьшать размер модели

* 1. Разработка подсистемы управления событиями

Любое windows-приложение 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:

- Scroll возникает при каждом перемещении ползунка движения по координатам;

- RenderTimer используется для создания таймера;

- SelectedIndexChanged выбор оси для изменения угла модели;

3. ИНФОРМАЦИОННОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

3.1 Общие сведения о программе

Программа называется "Трехмерная модель самолёта". При работе с данной программой у пользователя есть возможность работать с визуальной моделью данного объекта. Вращать ее относительно осей, приближать и удалять сцену с помощью ползунка, Программное обеспечение, на котором разработана приложение – Microsoft Visual Studio 2022 C#.

3.2 Функциональное назначение

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

- Наблюдать модель;

- Вращать модель;

- Изменять угол наклона модели;

- Увеличивать или уменьшать модель;

3.3 Логическая структура и функциональная декомпозиция проекта

Мы обязательно должны вызвать функцию glutInit() до того, как начнем использовать любые другие функции данной библиотеки, так как эта функция производит инициализацию библиотеки Glut. Далее мы вызываем следующую функцию: Glut.glutInitDisplayMode (Glut.GLUT\_RGB|Glut.GLUT\_DOUBLE|Glut.GLUT\_DEPTH). Эта функция устанавливает режим отображения. В нашем случае устанавливается RGB режим для визуализации. Далее мы устанавливаем двойную буферизацию окна. Двойная буферизация, как правило, используется для устранения мерцания, возникающего в процессе быстрой перерисовки кадров несколько раз подряд. GLUT\_DEPTH указывает при инициализации окна, если в приложении будет использоваться буфер глубины.

Gl.glViewport(0, 0, AnT.Width, AnT.Height); Здесь мы указываем библиотеке OpenGL на то, что вывод будет осуществляться во всей области элемента AnT (наш элемент, расположенный на форме для визуализации в него сцены). Определяем тот самый порт просмотра, куда последним шагом будет визуализироваться модель. После этого происходит настройка проекции. Для этого мы сначала вызываем: Gl.glMatrixMode(Gl.GL\_PROJECTION);

Функция glMatrixMode предназначена для того, чтобы задавать матричный режим: будет определена матрица, над которой мы в дальнейшем будем производить операции. В нашем случае это GL\_PROJECTION – матрица проекций. Следующей командой мы очищаем матрицу с помощью функции glLoadIdentity (функция заменяет текущую матрицу на единичную).

Далее мы устанавливаем тип текущей проекции с помощью функции gluPerspective. Gl.glLoadIdentity(); Glu.gluPerspective(45, (float)AnT.Width / (float)AnT.Height, 0.1, 200);

Функция gluPerspective определена в библиотеке GLU – OpenGL Utility Library (GLU). Эта библиотека является надстройкой над библиотекой OpenGL, реализующей ряд более продвинутых функций. Она также является свободно распространяемой и поставляется вместе с библиотекой OpenGL. Данная функция строит пирамиду охвата видимости, основываясь на угле визуального охвата, отношении сторон порта просмотра и установке ближней и дальней плоскости просмотра.

Теперь, когда проекция определена, мы устанавливаем в качестве текущей матрицы объектно-видовую матрицу и очищаем ее. Gl.glMatrixMode(Gl.GL\_MODELVIEW); Gl.glLoadIdentity();

Теперь нам остается только включить некоторые опции, необходимые для корректной визуализации нашей сцены. Это тест глубины, а также отображение цветов материалов для рисуемой геометрии. Gl.glEnable(Gl.GL\_DEPTH\_TEST); Gl.glEnable(Gl.GL\_COLOR\_MATERIAL);

Вот и все, что касается инициализации OpenGL.

3.4 Требования к техническому и программному обеспечению

Для успешной эксплуатации программного продукта необходим персональный компьютер со следующими характеристиками: процессор Intel Core i3-7700 с тактовой частотой 3600 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 10 Гбайт, Программное обеспечение: операционная система WINDOWS 10 и выше.

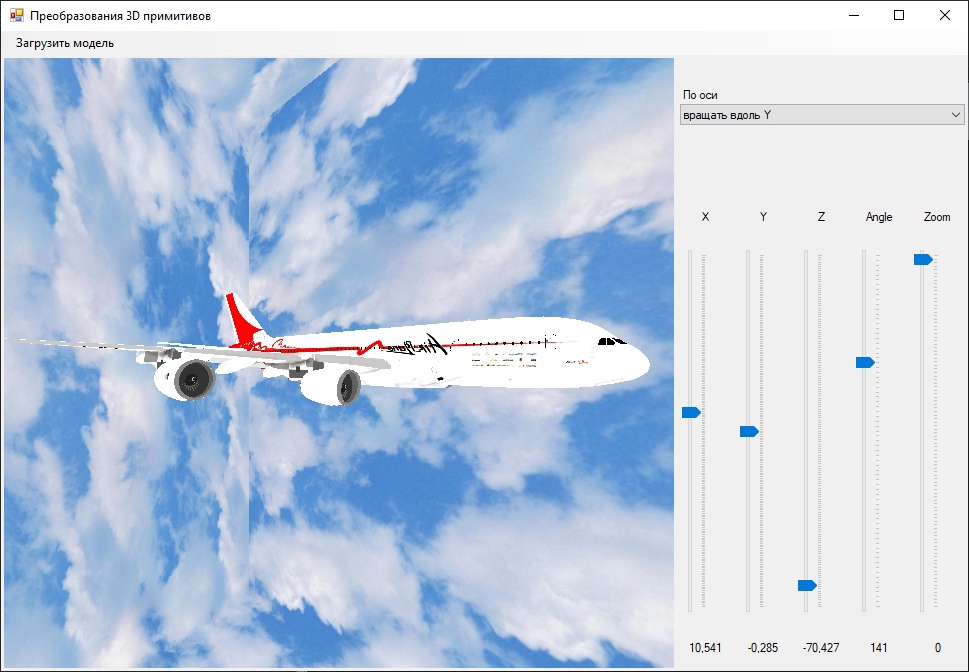
3.5 Руководство пользователя

Для установки приложения требуется разархивировать папку с проектом. Для запуска нужно войти в Microsoft Visual Studio 2022 и открыть проект в папке.

Разработанное приложение имеет интуитивно понятный интерфейс, который схож с другими Windows – приложениями. После запуска программы пользователь, может вращать сцену и поворачивать её с помощью ползунков. Также имеется возможность приближать и удалять модель, это можно сделать, ползунками.

4. ТЕСТИРОВАНИЕ

Программа запускается корректно, весь функционал работает.



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, FREEGLUT и DEVIL.

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

- изучение принципов работы OpenGL, FREEGLUT и DEVIL в оконной среде Windows;

- получение практических навыков использования средств OpenGL, FREEGLUT и DEVIL;

- получение навыков программирования динамических трехмерных сцен;

Также была проведена работа с такими возможностями библиотеки как: загрузка текстур; загрузка модели; использование графических примитивов.

СПИСОК ЛИТЕРАТУРЫ

1. <http://esate.ru/uroki/OpenGL/uroki-OpenGL-c-sharp/>
2. <https://opengl.org.ru/coding/glut/glut2.html>
3. <https://ru.wikipedia.org/wiki/DevIL>
4. <https://ru.wikipedia.org/wiki/Freeglut>
5. <https://rsdn.org/article/opengl/ogltut2.xml>

ПРИЛОЖЕНИЕ А. Исходный код файла anModelLoader.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Windows.Forms;

using Tao.OpenGl;

using Tao.FreeGlut;

using Tao.Platform.Windows;

using Tao.DevIl;

namespace Model

{

// класс LIMB отвечает за логические единицы 3D объектов в 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 памяти 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 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);

Gl.glCallList(thisList);

// возврат матрицы

Gl.glPopMatrix();

}

}

}

Исходный код 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\_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);

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();

}

}

}

}