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 "Вертолет""

Выполнил: Стариков Д. С.

Проверил: ст. преподаватель Цагадаева Е. Н

Подпись \_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_

Дата защиты: 17.12.2022

Улан-Удэ

2022

## АННОТАЦИЯ

Приложение написано на языке С# с использованием библиотеки OpenGL и шейдеров. Программа создана в среде Visual Studio 2019. В рамках работы были выполнены все представляемые задачи. Имеется возможность задать позицию камеры, относительно модели, изменить освещение через шейдер и автоматически вращать модель вокруг своей оси.

Содержание

**[АННОТАЦИЯ 3](#_Toc12501)**

**[ВВЕДЕНИЕ](#_Toc11903)** [5](#_Toc11903)

[1.](#_Toc19164) **[ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ OPENGL](#_Toc19164)** [7](#_Toc19164)

[1.1 Основные возможности 7](#_Toc30372)

**[1.2 Работа с матрицами 8](#_Toc28132)**

[1.3 Проекции 9](#_Toc24781)

[1.4 Освещение 10](#_Toc25797)

[1.5 Спецификация материалов 11](#_Toc20150)

**[2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТРОЕНИЯ ДИНАМИЧЕСКОГО ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ОБЪЕКТА "ВЕРТОЛЕТ" 14](#_Toc19336)**

[2.1 Разработка процедуры визуализации трехмерной сцены 14](#_Toc4246)

**[2.3 Разработка подсистемы управления событиями 17](#_Toc3216)**

**[3. ИНФОРМАЦИОННОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 18](#_Toc32290)**

[3.1 Общие сведения о программе 18](#_Toc25626)

[3.2 Функциональное назначение 18](#_Toc156)

[3.3 Логическая структура и функциональная декомпозиция проекта 18](#_Toc1162)

[3.4 Требования к техническому и программному обеспечению 21](#_Toc24149)

[3.5 Руководство пользователя 21](#_Toc2661)

**[4. ЗАКЛЮЧЕНИЕ И ВЫВОДЫ 22](#_Toc23375)**

**[СПИСОК ЛИТЕРАТУРЫ 23](#_Toc9455)**

[ПРИЛОЖЕНИЕ А 24](#_Toc8164)

[ПРИЛОЖЕНИЕ Б. 25](#_Toc16568)

[ПРИЛОЖЕНИЕ В 41](#_Toc7572)

[ПРИЛОЖЕНИЕ Г 42](#_Toc11173)

[ПРИЛОЖЕНИЕ Д 44](#_Toc11122)

## ВВЕДЕНИЕ

OpenGL – это графический стандарт в области компьютерной графики. На данный момент он является одним из самых популярных графических стандартов во всём мире. OpenGL переводится как Открытая Графическая Библиотека (Open Graphics Library), это означает, что OpenGL – это открытый и мобильный стандарт.

Во-первых, имена всех функций, предоставляемых непосредственно OpenGL, начинаются с приставки «gl».

Во-вторых, функции, задающие некоторый параметр, характеризующийся набором чисел (например, координату или цвет), имеют суффикс вида [число параметров + тип параметров + представление параметров].

Можно сказать, что библиотека OpenGL является только воспроизводящей (Rendering), и занимается только отображением 3D обьектов, она не работает с устройствами ввода (клавиатуры, мыши). Также она не поддерживает менеджер окон. OpenGL имеет хорошо продуманную внутреннюю структуру и довольно простой процедурный интерфейс. Несмотря на это с помощью OpenGL можно создавать сложные и мощные программные комплексы, затрачивая при этом минимальное время по сравнению с другими графическими библиотеками.

## 1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ 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.

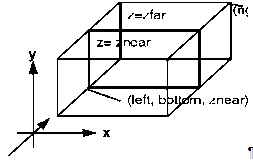


Рисунок 1 – Ортографическая проекция

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]

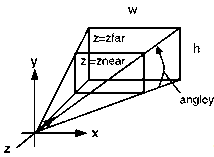


Рисунок 2 – Перспективная проекция

## 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\_SIDEпараметр "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).

1.5 Шейдеры

Шейдер – программа, предназначенная для исполнения процессорами видеокарты (GPU). Шейдеры составляются на одном из специализированных языков программирования и компилируются в инструкции для графического процессора.

Программы, работающие с трёхмерной графикой и видео (игры, GIS, CAD, CAM и др.), используют шейдеры для определения параметров геометрических объектов или изображения, для изменения изображения (для создания эффектов сдвига, отражения, преломления, затемнения с учётом заданных параметров поглощения и рассеяния света, для наложения текстур на геометрические объекты и др.).

Сначала видеокарты оснащали несколькими специализированными процессорами, поддерживающими разные наборы инструкций. Шейдеры делили на три типа в зависимости от того, какой процессор будет их исполнять (в зависимости от того, какие наборы инструкций доступны):

* вершинные (*англ. vertex shader*);
* геометрические (*англ. geometry shader*);
* пиксельные или фрагментные (*англ. pixel shader, fragment shader*).

Затем видеокарты стали оснащать универсальными процессорами (GPU), поддерживающими наборы инструкций всех трёх типов шейдеров (унифицировали шейдерную архитектуру). Деление шейдеров на типы сохранилось для описания назначения шейдера. Появилась возможность выполнения на GPU вычислений общего назначения (не связанных только с компьютерной графикой), например майнинг, нейронные сети.

**Вершинные шейдеры**

Вершинный шейдер оперирует данными, связанными с вершинами многогранников, например, с координатами вершины (точки) в пространстве, с текстурными координатами, с цветом вершины, с вектором касательной, с вектором бинормали, с вектором нормали. Вершинный шейдер может использоваться для видового и перспективного преобразования вершин, для генерации текстурных координат, для расчёта освещения и т. д.

**Геометрические шейдеры**

Геометрический шейдер, в отличие от вершинного, способен обработать не только одну вершину, но и целый примитив. Примитивом может быть отрезок (две вершины) и треугольник (три вершины), а при наличии информации о смежных вершинах (*англ. adjacency*) для треугольного примитива может быть обработано до шести вершин. Геометрический шейдер способен генерировать примитивы «на лету» (не задействуя при этом центральный процессор).

*Геометрические шейдеры впервые стали использоваться на видеокартах Nvidia серии 8.*

**Пиксельные (фрагментные) шейдеры**

Пиксельный шейдер работает с фрагментами растрового изображения и с текстурами — обрабатывает данные, связанные с пикселями (например, цвет, глубина, текстурные координаты). Пиксельный шейдер используется на последней стадии графического конвейера для формирования фрагмента изображения.

**Достоинства:**

* возможность составления любых алгоритмов (гибкость, упрощение и удешевление цикла разработки программы, повышение сложности и реалистичности визуализируемых сцен);
* повышение скорости выполнения (по сравнению со скоростью выполнения того же алгоритма, исполняемого на центральном процессоре).

**Недостатки:**

* необходимость изучения нового языка программирования;
* существование различных наборов инструкций для GPU разных производителей.

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

## 2.1 Разработка процедуры визуализации трехмерной сцены

В качестве модели был взят советский многоцелевой вертолет Ми-8.

Ми-8, и его модификации, самый массовый двухдвигательный вертолёт в мире (в общей сложности построено более 17 тыс. экземпляров всех модификаций); является самым массовым вертолётом в истории авиации.

На основе референсов создается 3д модель



Рисунок 4.1 Модель Ми-8, вид спереди

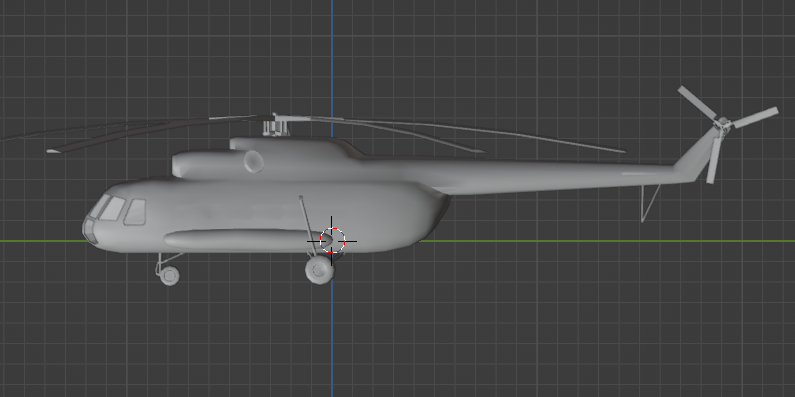


Рисунок 4.2 Модель Ми-8, вид сбоку

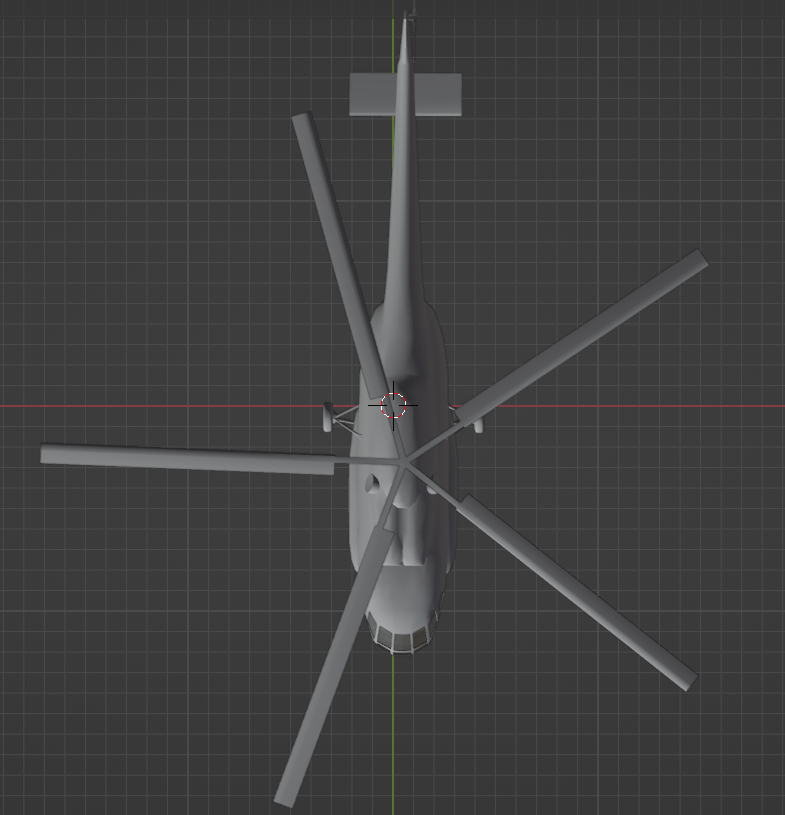


Рисунок 4.3.Модель Ми-8, вид сверху

Весь код метода представлен в Приложении Б, а алгоритм построения трехмерного объекта "Вертолет" средствами OPENGL, в Приложении В.

Все необходимые операции производились теми же функциями glTranslated, glRotated.

2.3 Разработка подсистемы управления событиями

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

* WM\_DESTROY - освобождение занятых ресурсов;
* WM\_SIZE - изменения сцены относительно размеров окна;
* WM\_ERASEBKGND - предотвращения мерцания;
* WM\_TIMER – используется для создания таймера;

3. ИНФОРМАЦИОННОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

## 3.1 Общие сведения о программе

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

Вращать ее автоматически, приближать и удалять сцену с помощью изменения кода.

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

## 3.2 Функциональное назначение

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

* наблюдать модель
* автоматически вращать вертолет;

## 3.3 Логическая структура и функциональная декомпозиция проекта

Инициализация OpenGL происходит в несколько этапов.

1. Выбираем и устанавливаем формат пикселей. В информации о формате пикселей указывается такая информация как глубина цвета, различные флаги поверхности. Вся эта структура представляется в специальной структуре PIXELFORMATDESCRIPTOR. Далее передаем на рассмотрение операционной системе, выбранный формат пикселей. После того, как система просмотрит его, она выберет наиболее совпадающий формат с тем, который поддерживается в контексте устройства. Функцией, осуществляющей такого рода проверку, является ChoosePixelFormat(). После выбора формата пикселей функция SetPixelFormat() устанавливает его в контексте устройства.
2. Создаем контекст вывода для библиотеки OpenGL. Данный контекст создается с помощью функции wglCreateContext(), далее функция wglMakeCurrent() устанавливает текущий контекст. Функция wglGetCurrentDC() необходима для корректного завершения приложения, а wglGetCurrentDC() – для удаления контекста воспроизведения.

Одним из важных методов является usr\_ReSize(), который устанавливает перспективу и область вывода. За отображение сцены отвечает метод usr\_RenderScene(), который вызывает в свою очередь функции рисования компьютера. Функции, вызываемые методом usr\_RenderScene() были подробно рассмотрены в разделе "2.1 Разработка процедуры визуализации трехмерной сцены", а важные для логического понимания структуры события программы рассмотрены в разделе "2.3 Разработка подсистемы управления событиями". Для наглядности приведем таблицу наиболее важных методов - таблица 3.1.

Таблица 3.1 – Основные методы и функции

|  |  |  |  |
| --- | --- | --- | --- |
| № | Метод | Назначение | Назначение параметров |
| 1 | PreCreateWindow (CREATESTRUCT& cs) | Инициализация окна | CS – объект структуры CREATESTRUCT.  Производится изменение значений, присвоенных переменным-членам структуры CREATESTRUCT для изменения режима открытия окна и его параметров. |
| 2 | usr\_bSetupPixelFormat () | Установка формата пикселей |  |
| 3 | usr\_bInitOpenGL() | Инициализация OpenGL |  |
| 4 | user\_DestoryOpenGL() | Освобождение ресурсов (из-под OpenGL) |  |
| 5 | usr\_ReSize (int x, int y, int width, int height) | Корректирует вывод сцены на экран при изменении размера окна | x и y определяют координаты левого нижнего угла вывода, width и height – ширину и высоту области вывода |
| 6 | usr\_RenderScene() | Рисует Вертолет целиком |  |
|  | ObjLoader() | Загружает все шейдеры и модели |  |
|  | ModelLoad() | Проверка наличия файлов и загрузка | Загружает, основываясь на вершинах, текстурных вершинах, нормалях и гранях |
|  | Vert.shader | Настройки вершинного шейдера | Настраивает позицию камеры |
|  | Frag.shader | Настройка пиксельного шейдера, идет после вершинного | Настраивает освешение |

Таблица 3.2 – Спецификации классов

|  |  |
| --- | --- |
| Название | Назначение |
| СMainframe | Класс главного окна приложения. Используется для управления главным окном |
| CKarkasApp | Главный класс приложения. Управляет работой всего приложения. Методы этого класса выполняют инициализацию приложения, обработку цикла сообщений и вызываются при завершении приложения. |
| CKarkasDoc | Класс документа приложения. |
| CKarkasView | Класс окна просмотра документа. Служит для отображения в клиентской области класса документа приложения в нашем случае нашей 3D модели. |
| CAboutDlg | Класс справочной информации о программе |
| DlgPers | Класс диалогового окна. Служит для настройки и смены перспективы |
| DlgOptions | Класс диалогового окна. Служит для включения различных настроек сцены. |
| ShaderHelper | Загружает шейдеры и программу |

## 3.4 Требования к техническому и программному обеспечению

Для успешной эксплуатации программного продукта необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 200 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов – 256). Программное обеспечение: операционная система WINDOWS 2000/XP и выше.

## 3.5 Руководство пользователя

После запуска программы пользователь может вращать модель вертолета. Также имеется возможность приближать и удалять модель, это можно сделать, изменив часть кода.

4. 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;
* получение практических навыков использования средств OpenGL;
* получение навыков программирования динамических трехмерных анимационных сцен;
* получение навыков программирования интерактивных трехмерных приложений.

Также была проведена работа с такими возможностями библиотеки как:

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

Электронный ресурс

* 1. Все о C# | Полный курс C Sharp от Code Blog
  2. Программирование GLUT: окна и анимация. Miguel Angel Sepulveda, LinuxFocus.
  3. Уроки OpenGL и C# на сайте esate.ru
  4. OpenGL – Материал из Википедии
  5. Вертолёт Ми-8. Техническое описание. Материал из Википедии
  6. Вертолет Ми-8. Материал из Википедии
  7. 3Д модель Mi-8 Russian Military Helicopter 3D от HuNtEr\_3DdD
  8. Шейдер – Материал из Википедии
  9. LearnOpenGL. Урок 1.5 — Shaders – Habr.com
  10. ПРИЛОЖЕНИЕ А. Алгоритм построения трёхмерной сцены

Начало

glColor3dv

glTranslatef

glScalef

glRotatef

auxSolidSphere

glColor3dv

glTranslatef

gluCylinder

glTranslatef

gluCylinder

glColor3dv

auxSolidBox

glTranslatef

glTranslatef

auxSolidBox

glColor3dv

glColor3dv

glEnd

glBegin

auxSolidBox

glColor3dv

glColor3dv

glTranslatef

gluCylinder

auxSolidSphere

glTranslatef

glColor3dv

glTranslatef

glTranslatef

auxSolidBox

gluCylinder

glTranslatef

gluCylinder

glScalef

glRotatef

glColor3dv

glTranslatef

glTranslatef

auxSolidSphere

auxSolidSphere

auxSolidSphere

glTranslatef

auxSolidSphere

glTranslatef

glColor3dv

glTranslatef

auxSolidBox

glTranslatef

auxSolidSphere

glTranslatef

auxSolidSphere

gluCylinder

glTranslatef

gluCylinder

glTranslatef

glTranslatef

glTranslatef

auxSolidSphere

glTranslatef

auxSolidSphere

glTranslatef

gluCylinder

gluCylinder

glTranslatef

glFinish

glEnd

glTranslatef

gluPartialDisk

gluPartialDisk

glRotatef

glColor3f

glRotatef

glRotatef

glTranslatef

Конец

Рисунок А.1 – Алгоритм построения объекта "Вертолет"

ПРИЛОЖЕНИЕ Б. Исходный код

**//App.xaml.cs//**

*using System;*

*using System.Collections.Generic;*

*using System.Configuration;*

*using System.Data;*

*using System.Linq;*

*using System.Windows;*

*namespace ObjLoader\_OpenGL33WPF*

*{*

*/// <summary>*

*/// Interaction logic for App.xaml*

*/// </summary>*

*public partial class App : Application*

*{*

*}*

*}*

**//MainWindow.xaml.cs//**

*using System;*

*using System.Windows;*

*using OpenTK.Graphics.OpenGL;*

*using System.Drawing;*

*using System.Drawing.Imaging;*

*using OpenTK;*

*using System.Windows.Threading;*

*namespace ObjLoader\_OpenGL33WPF*

*{*

*/// <summary>*

*/// Interaction logic for MainWindow.xaml*

*/// </summary>*

*public partial class MainWindow : Window*

*{*

*private bool \_canDraw = false;*

*private int \_numOfVertices;*

*private ObjLoader \_obj;*

*private float \_angle = 0.0f;*

*private int \_program;*

*private int \_uMvpMatrix;*

*private int \_uModelMatrix;*

*private Matrix4 \_mvpMatrix;*

*private Matrix4 \_modelMatrix;*

*private Matrix4 \_viewMatrix;*

*private Matrix4 \_projMatrix;*

*private DispatcherTimer \_dispatcherTimer;*

*public MainWindow()*

*{*

*InitializeComponent();*

*}*

*private void WindowsFormsHost\_Initialized(object sender, EventArgs e)*

*{*

*glControl.MakeCurrent();*

*}*

*private void GLControl\_Load(object sender, EventArgs e)*

*{*

*GL.ClearColor(0.2f, 0.3f, 0.2f, 1.0f);*

*GL.Enable(EnableCap.DepthTest);*

*if (!ShaderHelpers.InitShaders(*

*"Shaders/vert.shader.glsl",*

*"Shaders/frag.shader.glsl", out \_program))*

*{*

*return;*

*}*

*if (!InitVertexBuffers(\_program))*

*{*

*return;*

*}*

*InitTextures();*

*\_uMvpMatrix = GL.GetUniformLocation(\_program, "uMvpMatrix");*

*if (\_uMvpMatrix < 0)*

*{*

*MessageBox.Show("Failed to get uMvpMatrix variable.");*

*return;*

*}*

*\_uModelMatrix = GL.GetUniformLocation(\_program, "uModelMatrix");*

*if (\_uModelMatrix < 0)*

*{*

*MessageBox.Show("Failed to get uModelMatrix variable.");*

*return;*

*}*

*\_viewMatrix = Matrix4.LookAt(*

*new Vector3(0.0f, 3.0f, 10.0f),*

*new Vector3(0.0f, 0.0f, 0.0f),*

*new Vector3(0.0f, 1.0f, 0.0f));*

*\_dispatcherTimer = new DispatcherTimer();*

*\_dispatcherTimer.Tick += OnUpdate;*

*\_dispatcherTimer.Interval = TimeSpan.FromMilliseconds(16.0);*

*\_dispatcherTimer.Start();*

*\_canDraw = true;*

*}*

*private void OnUpdate(object sender, EventArgs e)*

*{*

*\_angle += 2.0f;*

*\_modelMatrix =*

*Matrix4.CreateScale(3.0f) \**

*Matrix4.CreateRotationY(MathHelper.DegreesToRadians(\_angle));*

*\_mvpMatrix = \_modelMatrix \* \_viewMatrix \* \_projMatrix;*

*GL.UniformMatrix4(\_uMvpMatrix, false, ref \_mvpMatrix);*

*GL.UniformMatrix4(\_uModelMatrix, false, ref \_modelMatrix);*

*glControl.Invalidate();*

*}*

*private void GLControl\_Paint(object sender, System.Windows.Forms.PaintEventArgs e)*

*{*

*GL.Viewport(0, 0, glControl.Width, glControl.Height);*

*GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);*

*if (\_canDraw)*

*{*

*GL.DrawArrays(PrimitiveType.Triangles, 0, \_numOfVertices);*

*}*

*glControl.SwapBuffers();*

*}*

*private void GLControl\_Resize(object sender, EventArgs e)*

*{*

*SetProjMatrix();*

*glControl.Invalidate();*

*}*

*private bool InitVertexBuffers(int program)*

*{*

*\_obj = new ObjLoader();*

*\_obj.LoadModel("Models/monkey/monkey.obj");*

*\_numOfVertices = \_obj.vertIndex.Count;*

*int vbo;*

*GL.GenBuffers(1, out vbo);*

*GL.BindBuffer(BufferTarget.ArrayBuffer, vbo);*

*GL.BufferData(BufferTarget.ArrayBuffer,*

*\_obj.model.Length \* sizeof(float),*

*\_obj.model, BufferUsageHint.StaticDraw);*

*int textureOffset = \_obj.vertIndex.Count \* 3 \* sizeof(float);*

*int normalOffset = textureOffset + \_obj.texIndex.Count \* 2 \* sizeof(float);*

*GL.VertexAttribPointer(0, 3, VertexAttribPointerType.Float, false, 0, 0);*

*GL.EnableVertexAttribArray(0);*

*GL.VertexAttribPointer(1, 2, VertexAttribPointerType.Float, false, 0, textureOffset);*

*GL.EnableVertexAttribArray(1);*

*GL.VertexAttribPointer(2, 3, VertexAttribPointerType.Float, false, 0, normalOffset);*

*GL.EnableVertexAttribArray(2);*

*return true;*

*}*

*private void InitTextures()*

*{*

*int texture;*

*GL.GenTextures(1, out texture);*

*GL.BindTexture(TextureTarget.Texture2D, texture);*

*GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapS, (int)All.Repeat);*

*GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapT, (int)All.Repeat);*

*GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter, (int)All.Linear);*

*GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter, (int)All.Linear);*

*string imageFileName = "Models/monkey/DefTexture.png";*

*Bitmap image;*

*try*

*{*

*image = new Bitmap(imageFileName);*

*}*

*catch (Exception)*

*{*

*MessageBox.Show("Failed to load the texture: " + imageFileName);*

*return;*

*}*

*Rectangle rect = new Rectangle(0, 0, image.Width, image.Height);*

*BitmapData data = image.LockBits(rect, ImageLockMode.ReadOnly, System.Drawing.Imaging.PixelFormat.Format32bppRgb);*

*GL.TexImage2D(TextureTarget.Texture2D, 0,*

*PixelInternalFormat.Rgb, image.Width, image.Height,*

*0, OpenTK.Graphics.OpenGL.PixelFormat.Rgba,*

*PixelType.UnsignedByte, data.Scan0);*

*image.UnlockBits(data);*

*}*

*private void SetProjMatrix()*

*{*

*\_projMatrix = Matrix4.CreatePerspectiveFieldOfView(*

*MathHelper.DegreesToRadians(45.0f),*

*(float)glControl.Width / glControl.Height, 0.1f, 1000.0f);*

*}*

*}*

*}*

**//ObjLoader.cs//**

*using System.Collections.Generic;*

*using System.IO;*

*using System.Text.RegularExpressions;*

*using System.Windows.Forms;*

*using System.Globalization;*

*namespace ObjLoader\_OpenGL33WPF*

*{*

*class ObjLoader*

*{*

*public float[] model;*

*public List<float> vertCoord = new List<float>();*

*public List<float> texCoord = new List<float>();*

*public List<int> vertIndex = new List<int>();*

*public List<int> texIndex = new List<int>();*

*public List<int> normIndex = new List<int>();*

*private List<float> \_model = new List<float>();*

*private List<List<int>> \_vertIndex = new List<List<int>>();*

*private List<List<int>> \_texIndex = new List<List<int>>();*

*private List<List<int>> \_normIndex = new List<List<int>>();*

*private List<List<float>> \_vertCoords = new List<List<float>>();*

*private List<List<float>> \_texCoords = new List<List<float>>();*

*private List<List<float>> \_normCoords = new List<List<float>>();*

*private const string WHITESPACE\_RE = @"\s+";*

*public void LoadModel(string filePath)*

*{*

*if (!File.Exists(filePath))*

*{*

*MessageBox.Show("Failed to open the file: " + filePath);*

*return;*

*}*

*using (var sr = new StreamReader(filePath))*

*{*

*string data = sr.ReadToEnd();*

*string[] lines = data.Split(new char[] { '\n' });*

*CultureInfo culture = new CultureInfo("en-US");*

*for (int i = 0; i < lines.Length; i++)*

*{*

*if (lines[i].StartsWith("#")) continue;*

*string line = lines[i].Trim();*

*List<string> values = new List<string>(Regex.Split(line, WHITESPACE\_RE));*

*if (values.Count == 0) continue;*

*if (values[0] == "v")*

*{*

*float x = float.Parse(values[1], culture);*

*float y = float.Parse(values[2], culture);*

*float z = float.Parse(values[3], culture);*

*\_vertCoords.Add(new List<float>() { x, y, z });*

*}*

*if (values[0] == "vt")*

*{*

*float u = float.Parse(values[1], culture);*

*float v = float.Parse(values[2], culture);*

*\_texCoords.Add(new List<float>() { u, v });*

*}*

*if (values[0] == "vn")*

*{*

*float x = float.Parse(values[1], culture);*

*float y = float.Parse(values[2], culture);*

*float z = float.Parse(values[3], culture);*

*\_normCoords.Add(new List<float>() { x, y, z });*

*}*

*if (values[0] == "f")*

*{*

*List<int> face\_i = new List<int>();*

*List<int> tex\_i = new List<int>();*

*List<int> norm\_i = new List<int>();*

*for (int j = 1; j < 4; j++)*

*{*

*string[] w = values[j].Split(new char[] { '/' });*

*face\_i.Add(int.Parse(w[0]) - 1);*

*tex\_i.Add(int.Parse(w[1]) - 1);*

*norm\_i.Add(int.Parse(w[2]) - 1);*

*}*

*\_vertIndex.Add(face\_i);*

*\_texIndex.Add(tex\_i);*

*\_normIndex.Add(norm\_i);*

*}*

*}*

*for (int i = 0; i < \_vertIndex.Count; i++)*

*{*

*for (int j = 0; j < \_vertIndex[i].Count; j++)*

*{*

*vertIndex.Add(\_vertIndex[i][j]);*

*}*

*}*

*for (int i = 0; i < \_texIndex.Count; i++)*

*{*

*for (int j = 0; j < \_texIndex[i].Count; j++)*

*{*

*texIndex.Add(\_texIndex[i][j]);*

*}*

*}*

*for (int i = 0; i < \_normIndex.Count; i++)*

*{*

*for (int j = 0; j < \_normIndex[i].Count; j++)*

*{*

*normIndex.Add(\_normIndex[i][j]);*

*}*

*}*

*for (int i = 0; i < vertIndex.Count; i++)*

*{*

*int index = vertIndex[i];*

*List<float> coords = \_vertCoords[index];*

*for (int j = 0; j < coords.Count; j++)*

*{*

*\_model.Add(coords[j]);*

*vertCoord.Add(coords[j]);*

*}*

*}*

*for (int i = 0; i < texIndex.Count; i++)*

*{*

*int index = texIndex[i];*

*List<float> coords = \_texCoords[index];*

*for (int j = 0; j < coords.Count; j++)*

*{*

*\_model.Add(coords[j]);*

*texCoord.Add(coords[j]);*

*}*

*}*

*for (int i = 0; i < normIndex.Count; i++)*

*{*

*int index = normIndex[i];*

*List<float> coords = \_normCoords[index];*

*for (int j = 0; j < coords.Count; j++)*

*{*

*\_model.Add(coords[j]);*

*}*

*}*

*model = \_model.ToArray();*

*}*

*}*

*}*

*}*

**//ShaderHelper.cs//**

*using System;*

*using OpenTK.Graphics.OpenGL;*

*using System.Windows.Forms;*

*using System.IO;*

*namespace ObjLoader\_OpenGL33WPF*

*{*

*class ShaderHelpers*

*{*

*///<summary>*

*///Create a program object and make current*

*///</summary>*

*///<param name="vShader">a vertex shader program</param>*

*///<param name="fShader">a fragment shader program</param>*

*///<param name="program">created program</param>*

*///<returns>*

*///return true, if the program object was created and successfully made current*

*///</returns>*

*public static bool InitShaders(string vShaderPath, string fShaderPath, out int program)*

*{*

*string vShaderSource, fShaderSource;*

*LoadShaderFromFile(vShaderPath, out vShaderSource);*

*LoadShaderFromFile(fShaderPath, out fShaderSource);*

*program = CreateProgram(vShaderSource, fShaderSource);*

*if (program == 0)*

*{*

*MessageBox.Show("Failed to create program");*

*return false;*

*}*

*GL.UseProgram(program);*

*return true;*

*}*

*private static int CreateProgram(string vShader, string fShader)*

*{*

*// Create shader object*

*int vertexShader = CreateShader(ShaderType.VertexShader, vShader);*

*int fragmentShader = CreateShader(ShaderType.FragmentShader, fShader);*

*if (vertexShader == 0 || fragmentShader == 0)*

*{*

*return 0;*

*}*

*// Create a program object*

*int program = GL.CreateProgram();*

*if (program == 0)*

*{*

*return 0;*

*}*

*// Attach the shader objects*

*GL.AttachShader(program, vertexShader);*

*GL.AttachShader(program, fragmentShader);*

*// Link the program object*

*GL.LinkProgram(program);*

*// Check the result of linking*

*int status;*

*GL.GetProgram(program, GetProgramParameterName.LinkStatus, out status);*

*if (status == 0)*

*{*

*string errorString = string.Format("Failed to link program: {0}" + Environment.NewLine, GL.GetProgramInfoLog(program));*

*MessageBox.Show(errorString);*

*GL.DeleteProgram(program);*

*GL.DeleteShader(vertexShader);*

*GL.DeleteShader(fragmentShader);*

*return 0;*

*}*

*return program;*

*}*

*///<summary>*

*///Load a shader from a file*

*///</summary>*

*///<param name="errorOutputFileName">a file name for error messages</param>*

*///<param name="fileName">a file name to a shader</param>*

*///<param name="shaderSource">a shader source string</param>*

*public static void LoadShaderFromFile(string shaderFileName, out string shaderSource)*

*{*

*shaderSource = null;*

*using (StreamReader sr = new StreamReader(shaderFileName))*

*{*

*shaderSource = sr.ReadToEnd();*

*}*

*}*

*private static int CreateShader(ShaderType shaderType, string shaderSource)*

*{*

*// Create shader object*

*int shader = GL.CreateShader(shaderType);*

*if (shader == 0)*

*{*

*MessageBox.Show("Unable to create shader");*

*return 0;*

*}*

*// Set the shader program*

*GL.ShaderSource(shader, shaderSource);*

*// Compile the shader*

*GL.CompileShader(shader);*

*// Check the result of compilation*

*int status;*

*GL.GetShader(shader, ShaderParameter.CompileStatus, out status);*

*if (status == 0)*

*{*

*string errorString = string.Format("Failed to compile {0} shader: {1}", shaderType.ToString(), GL.GetShaderInfoLog(shader));*

*MessageBox.Show(errorString);*

*GL.DeleteShader(shader);*

*return 0;*

*}*

*return shader;*

*}*

*}*

*}*

**//frag.shader.glsl//**

*#version 330*

*precision mediump float;*

*uniform sampler2D uSampler;*

*in vec2 vTexCoord;*

*in vec3 vNormal;*

*out vec4 fragColor;*

*void main()*

*{*

*vec3 ambientLightIntensity = vec3(0.2, 0.2, 0.2);*

*vec3 sunLightIntensity = vec3(0.7, 0.7, 0.7);*

*vec3 sunLightDirection = normalize(vec3(-20.0, 20.0, 20.0));*

*vec2 flipped = vec2(vTexCoord.x, 1 - vTexCoord.y);*

*vec4 texel = texture(uSampler, flipped);*

*vec3 lightIntensity = ambientLightIntensity + sunLightIntensity \* max(dot(vNormal, sunLightDirection), 0.0f);*

*fragColor = vec4(texel.rgb \* lightIntensity, texel.a);*

*}*

**//vert.shader.glsl//**

*#version 330*

*layout(location=0) in vec3 aPosition;*

*layout(location=1) in vec2 aTexCoord;*

*layout(location = 2) in vec3 aNormal;*

*uniform mat4 uMvpMatrix;*

*uniform mat4 uModelMatrix;*

*out vec2 vTexCoord;*

*out vec3 vNormal;*

*void main()*

*{*

*gl\_Position = uMvpMatrix \* vec4(aPosition, 0.5);*

*vNormal = (uModelMatrix \* vec4(aNormal, 0.0)).xyz;*

*vTexCoord = aTexCoord;*

*}*

ПРИЛОЖЕНИЕ В

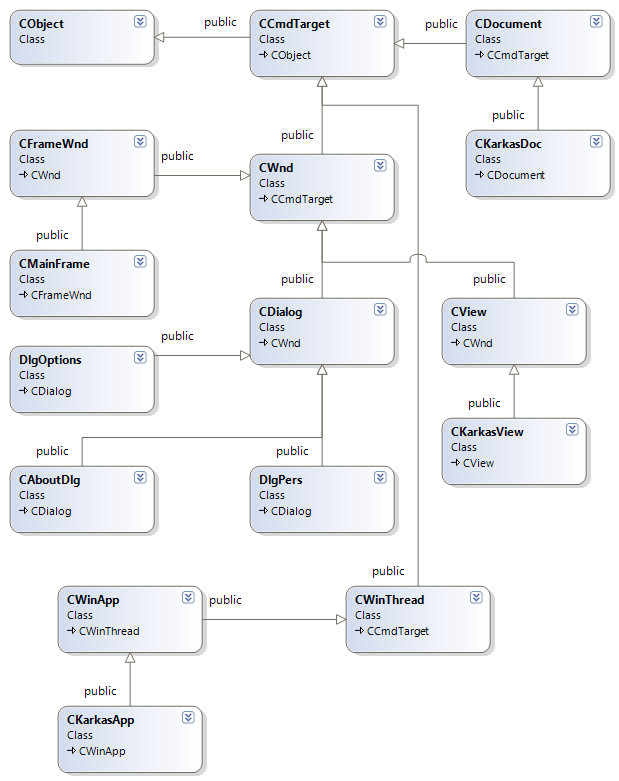


Рисунок В.1 – Диаграмма классов

## ПРИЛОЖЕНИЕ Г



Рисунок Г.1 – Трехмерная модель объекта "Вертолет"

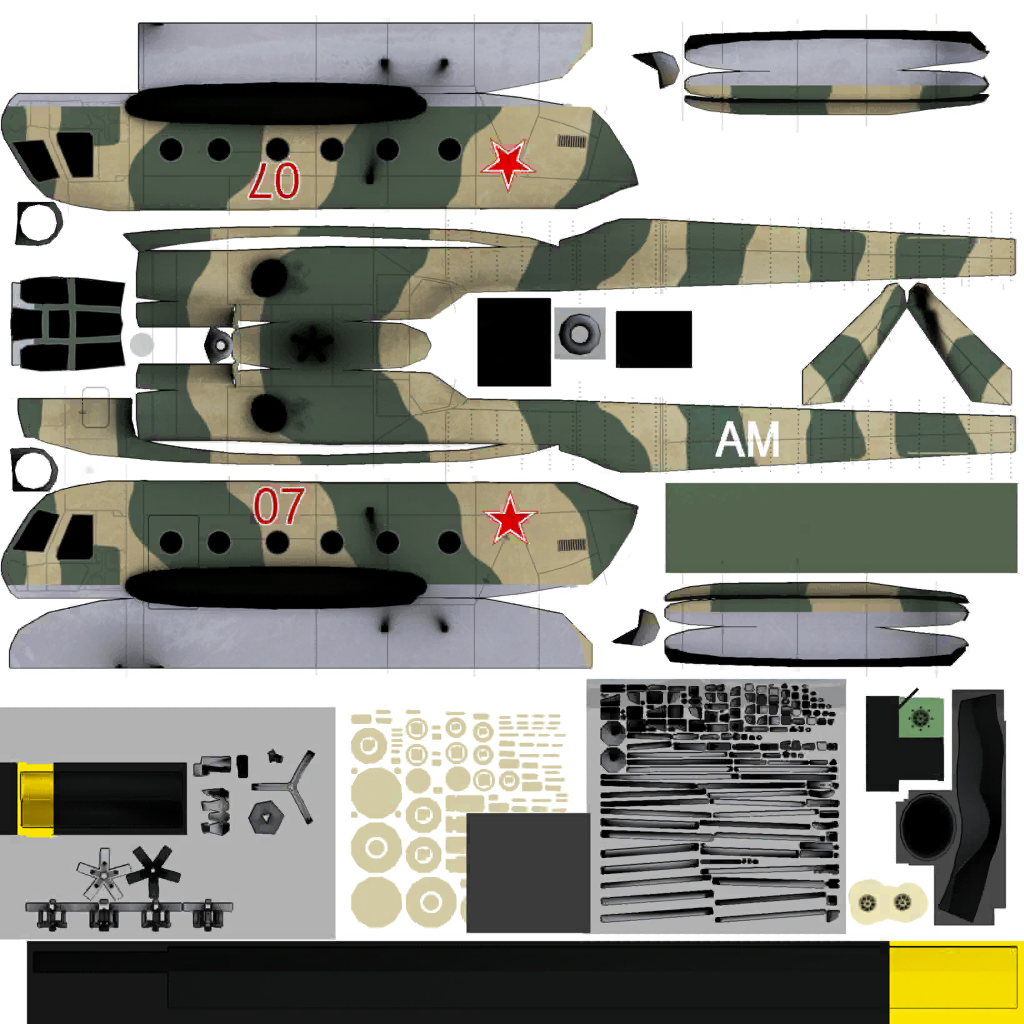


Рисунок Г.2 – Текстура Вертолета

## ПРИЛОЖЕНИЕ Д





Рисунок Д.1-2 – Прототип модели (Вертолет Ми-8)