Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА ПО ПРОГРАММЕ БАКАЛАВРИАТА

09.03.04 Программная инженерия

(код, наименование ОПОП ВО: направление подготовки, направленность (профиль))

«Разработка программно-информационных систем»

Программная система визуализации трехмерных данных,

использующая библиотеку OpenGL

(название темы)

Дипломный проект

(вид ВКР: дипломная работа или дипломный проект)

Автор ВКР _____ Н.И. Снатенков

(подпись, дата) (инициалы, фамилия)

Группа ПО-02б

Руководитель ВКР _____ В. В. Серебровский

(подпись, дата) (инициалы, фамилия)

Нормоконтроль _____ А. А. Чаплыгин

(подпись, дата) (инициалы, фамилия)

ВКР допущена к защите:

Заведующий кафедрой _____ А. В. Малышев

(подпись, дата) (инициалы, фамилия)

Курск 2024 г.

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

УТВЕРЖДАЮ:

Заведующий кафедрой

_____

(подпись, инициалы, фамилия)

«____» _____ 20____ г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ ПО ПРОГРАММЕ БАКАЛАВРИАТА

Студента Снатенкова Н.И., шифр 20-06-0086, группа ПО-02б

1. Тема «Программная система визуализации трехмерных данных, использующая библиотеку OpenGL» утверждена приказом ректора ЮЗГУ от «04» апреля 2024 г. № 1616-с.

2. Срок предоставления работы к защите «11» июня 2024 г.

3. Исходные данные для создания программной системы:

3.1. Перечень решаемых задач:

1) изучить основы и принципы работы библиотеки OpenGL;

2) разработать концептуальную модель графического движка на основе библиотеки OpenGL;

3) спроектировать программную систему визуализации трёхмерных данных;

4) сконструировать и протестировать программную систему визуализации трёхмерных данных.

3.2. Входные данные и требуемые результаты для программы:

1) Входными данными для программной системы являются: файлы трёхмерных моделей, стандартизированные под классический формат фай-

лов геометрии объектов OBJ, с триангулированными вершинами; файлы изображений текстур.

2) Выходными данными для программной системы являются: файлы данных, содержащие в себе информацию о сохраненных и загруженных файлов в проект; вывод визуализированного изображения трёхмерной виртуальной сцены на экран.

4. Содержание работы (по разделам):

4.1. Введение

4.1. Анализ предметной области

4.2. Техническое задание: основание для разработки, назначение разработки, требования к программной системе, требования к оформлению документации.

4.3. Технический проект: общие сведения о программной системе, проект данных программной системы, проектирование архитектуры программной системы, проектирование пользовательского интерфейса программной системы.

4.4. Рабочий проект: спецификация компонентов и классов программной системы, тестирование программной системы, сборка компонентов программной системы.

4.5. Заключение

4.6. Список использованных источников

5. Перечень графического материала:

Лист 1. Сведения о ВКРБ.

Лист 2. Цель и задачи разработки.

Лист 3. Диаграмма конвеера данных.

Лист 4. Диаграмма вариантов использования.

Лист 5. Диаграмма классов программы.

Лист 6. Прототип компонентов программы.

Лист 7. Прототип интерфейса программы.

Лист 8. Интерфейс программы.

Лист 9. Результат визуализации трёхмерных данных.

Руководитель ВКР	_____	В. В. Серебровский
	(подпись, дата)	(инициалы, фамилия)
Задание принял к исполнению	_____	Н.И. Снатенков
	(подпись, дата)	(инициалы, фамилия)

РЕФЕРАТ

Объем работы равен 110 страницам. Работа содержит 39 иллюстраций, 7 таблиц, 13 библиографических источников и 9 листов графического материала. Количество приложений – 2. Графический материал представлен в приложении А. Пример содержания файла геометрии объектов формата OBJ представлен в приложении Б. Пример содержания файлов Resources.data и Objects.data представлены в приложении В. Фрагмент части исходного кода программы парсера представлен в приложении Г.

Перечень ключевых слов: OpenGL, трёхмерный движок, виртуальная сцена, трёхмерная модель, текстура, вершины, грани, полигоны, триангулярность, диаграмма, объекты, файлы геометрии, парсер, трёхмерное пространство, освещение, координаты вершин, координаты текстур, направления нормалей, рендер, визуализация, проекция, камера, угол обзора, матрицы, матрица преобразований, матрица проекции камеры, аффинные преобразования.

Объектом разработки является программа, представляющая собой трёхмерный графический движок, работающий под управлением графической библиотеки и спецификации OpenGL, способный визуализировать загруженные в него данные трёхмерных объектов.

Целью выпускной квалификационной работы является создание программной основы в виде графического трёхмерного движка с открытым кодом для дальнейших разработок в сфере трёхмерного моделирования и геймдизайна.

В процессе создания программного обеспечения были выделены основные сущности путем создания информационных блоков, использованы классы и методы модулей, обеспечивающие работу с сущностями предметной области, а также корректную работу программной системы, разработана программа для визуализации трёхмерных данных.

При разработке программной системы использовалась графическая библиотека и спецификация «OpenGL».

ABSTRACT

The volume of work is 110 pages. The work contains 39 illustrations, 7 tables, 13 bibliographic sources and 9 sheets of graphic material. The number of applications is 2. Graphic material is presented in Appendix A. An example of the contents of an OBJ format object geometry file is presented in Appendix Б. An example of the contents of the Resources.data and Objects.data files is presented in Appendix B. A fragment of part of the source code of the parser program is presented in Appendix Г.

List of keywords: OpenGL, three-dimensional engine, virtual scene, three-dimensional model, texture, vertices, faces, polygons, triangularity, diagram, objects, geometry files, parser, three-dimensional space, lighting, vertex coordinates, texture coordinates, normal directions, render, visualization, projection, camera, viewing angle, matrices, transformation matrix, camera projection matrix, affine transformations.

The object of development is a program that is a three-dimensional graphics engine running under the control of a graphics library and the OpenGL specification, capable of visualizing the data of three-dimensional objects loaded into it.

The goal of the final qualifying work is to create a software basis in the form of an open-source 3D graphic engine for further developments in the field of 3D modeling and game design.

In the process of creating the software, the main entities were identified by creating information blocks, classes and methods of modules were used to ensure work with entities of the subject area, as well as the correct operation of the software system, and a program for visualizing three-dimensional data was developed.

When developing the software system, the graphics library and specification «OpenGL» were used.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 12
1 Анализ предметной области 15
1.1 История создания трёхмерной компьютерной графики 15
1.2 Основные принципы работы трёхмерной графики 15
1.3 Работа с OpenGL 16
1.3.1 Функции и операторы OpenGL 17
1.3.2 Интерфейс OpenGL 18
1.3.3 Архитектура OpenGL 20
1.3.4 Работа с OpenTK 22
1.4 Математика освещения в трёхмерной графике 23
1.4.1 Рассеянный свет 24
1.4.2 Общий свет 25
1.4.3 Отраженный свет 26
1.4.4 Испускаемый свет 27
1.5 Аффинные преобразования трёхмерного пространства 27
1.5.1 Параллельный перенос 29
1.5.2 Поворот вокруг оси 29
1.5.3 Масштабирование 30
2 Техническое задание 32
2.1 Основание для разработки 32
2.2 Цель и назначение разработки 32
2.3 Требования к программной системе 32
2.3.1 Требования к данным программно-информационной системы 32
2.3.2 Функциональные требования к программной системе 33
2.3.3 Требования к графическому интерфейсу программы 35
2.4 Моделирование вариантов использования 36
2.4.1 Вариант использования: Визуализация данных на экране 37
2.4.2 Вариант использования: Преобразование модели объекта 38
2.4.3 Вариант использования: Создание нового объекта 39

2.4.4 Вариант использования: Загрузка данных из файловой системы 41
2.4.5 Вариант использования: Изменение параметров света 41
2.4.6 Вариант использования: Управление виртуальной камерой 42
2.5 Нефункциональные требования к программной системе 43
2.6 Требования к оформлению документации 43
3 Технический проект 44
3.1 Общая характеристика организации решения задачи 44
3.2 Обоснование выбора технологии проектирования 44
3.3 Описание используемых технологий и языков программирования 45
3.3.1 Язык программирования C# 45
3.3.1.1 Особенности языка C# 45
3.3.2 Спецификация OpenGL 4.6 45
3.3.2.1 Особенности спецификации OpenGL 46
3.4 Диаграмма классов и компонентов программы 47
3.4.1 Диаграмма классов 47
3.4.1.1 Класс Model 49
3.4.1.2 Класс Texture 49
3.4.1.3 Класс Scene 49
3.4.1.4 Класс Shader 49
3.4.1.5 Класс Camera 50
3.4.1.6 Класс Model 50
3.4.1.7 Класс Inspector 50
3.4.1.8 Класс GameWindow 50
3.4.2 Диаграмма компонентов 50
3.4.2.1 Компонент Файл OBJ 52
3.4.2.2 Компонент Парсер 52
3.4.2.3 Компонент Модель 52
3.4.2.4 Компонент Файл изображения текстуры 52
3.4.2.5 Компонент Текстура 53
3.4.2.6 Компонент Файл scene.data 53
3.4.2.7 Компонент Сцена 53

3.4.2.8 Компонент Инспектор 53
3.4.2.9 Компонент Камера 54
3.4.2.10 Компонент Освещение 54
3.4.2.11 Компонент Шейдеров 54
3.4.2.12 Модуль Графический визуализатор 54
3.4.2.13 Компонент Программа 55
3.5 Описание файлов трёхмерных данных формата OBJ 55
3.6 Описание работы парсера 56
3.7 Система хранения данных внутри проекта 58
3.7.1 Файл Resource.data 58
3.7.2 Файл Objects.data 59
3.8 Проектирование пользовательского интерфейса 59
3.8.1 Главное окно программы 60
3.8.2 Окно инспектора 60
3.8.3 Вкладки окон инспектора 61
3.8.4 Активное меню вкладки инспектора 61
3.8.4.1 Вкладка «Объект» 61
3.8.4.2 Вкладка «Инспектор» 61
3.8.4.3 Вкладка «Модели и текстуры» 61
3.8.4.4 Вкладка «Свет» 62
4 Рабочий проект 63
4.1 Классы, использованные при разработке программной системы 63
4.2 Системное тестирование разработанной программной системы 69
4.2.1 Тестовый случай: Инициализация ресурсов программы 69
4.2.2 Тестовый случай: Визуализация объекта на сцене 73
4.2.3 Тестовый случай: Настройка освещения 74
4.2.4 Тестовый случай: Трансформация объекта 77
4.2.5 Тестовый случай: Импортирование ресурсов в проект 79
4.2.6 Тестовый случай: Удаление ресурсов из проекта 82
4.2.7 Тестовый случай: Создание нового объекта 85
ЗАКЛЮЧЕНИЕ 88

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 88
ПРИЛОЖЕНИЕ А Представление графического материала 91
ПРИЛОЖЕНИЕ Б Пример содержания OBJ-файла 101
ПРИЛОЖЕНИЕ В Пример содержания файлов, описывающие хранимые ресурсы проекта 103
ПРИЛОЖЕНИЕ Г Листинг фрагмента программы парсера 104
На отдельных листах (CD-RW в прикрепленном конверте) 110
Сведения о ВКРБ (Графический материал / Сведения о ВКРБ.png) Лист 1
Цель и задачи разработки (Графический материал / Цель и задачи разработки.png) Лист 2
Диаграмма конвеера данных (Графический материал / Диаграмма конвеера данных.png) Лист 3
Диаграмма вариантов использования (Графический материал / Диаграмма вариантов использования.png) Лист 4
Диаграмма классов программы (Графический материал / Диаграмма классов программы.png) Лист 5
Прототип компонентов программы (Графический материал / Прототип компонентов программы.png) Лист 6
Прототип интерфейса программы (Графический материал / Прототип интерфейса программы.png) Лист 7
Интерфейс программы (Графический материал / Интерфейс программы.png) Лист 8
Результат визуализации трёхмерных данных (Графический материал / Результат визуализации трёхмерных данных.png) Лист 9

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ИС – информационная система.

ИТ – информационные технологии.

КТС – комплекс технических средств.

ОМТС – отдел материально-технического снабжения.

ПО – программное обеспечение.

РП – рабочий проект.

ТЗ – техническое задание.

ТП – технический проект.

UML (Unified Modelling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

OBJ – формат файлов описания геометрии, разработанный в Wavefront Technologies для их анимационного пакета Advanced Visualizer. Формат файла является открытым и был принят другими разработчиками приложений 3D-графики.

3D (от англ. 3-dimensional) графика – раздел компьютерной графики, посвящённый методам создания изображений или видео путём моделирования объектов в трёх измерениях.

GL (Graphic Library) – это программная библиотека, предназначенная для оказания помощи в рендеринге компьютерной графики на мониторе.

ВВЕДЕНИЕ

Трёхмерная компьютерная графика появилась в 1960-х годах. Первые векторные изображения состояли из множества точек и кривых, заданных математической формулой. Айван Сазерленд и Дэвид Эванс основали первую в мире кафедру компьютерной графики в университете Юты, США. Именно в те времена появилась программа Sketchpad (от англ. «альбом для рисования») — предок всех современных 3D-редакторов.

На первых компьютерах можно было работать только с векторными изображениями. Затем появилась растровая графика, которая позволила изображать объекты в виде сетки пикселей. Кроме сплошных цветов и фигур, начали использовать текстуры и тени, расширились возможности рендеринга — превращения кода в финальные изображения.

Трёхмерные объекты математически представляются в виде множества точек - вершин, и в тоже время они, соединенные отрезками, образуют рёбра фигиры, а множество рёбер на одной плоскости образуют поверхности - грани фигуры. Данный массив точек трансформируется с учётом матрицы перспективы камеры, далее - накладываются текстуры на поверхности граней фигуры - изображения также растягивают и трансформируют с учётом перспективы наклона грани, затем - обрабатывают свет, падающий на модель, путём изменения яркости текстуры граней, в зависимости от источников света, и наконец - добавляют тень, которая отбрасывает данная трёхмерная модель.

3D-движок, рендерер или визуализатор является программной системой графического движка (название произошло от английского graphics engine) — промежуточное программное обеспечение, программный движок, основной задачей которого является визуализация двухмерной и трёхмерной компьютерной графики. Может существовать как отдельный продукт или в составе игрового движка. Может использоваться для визуализации отдельных изображений или компьютерного видео. Графические движки, использующееся в программах по работе с компьютерной графикой обычно назы-

ваются «рендерерами», «отрисовщиками» или «визуализаторами». Само название «графический движок» используется, как правило, в компьютерных играх.

Основное и важнейшее отличие «игровых» графических движков от неигровых состоит в том, что первые должны обязательно работать в режиме реального времени, тогда как вторые могут тратить по несколько десятков часов на вывод одного изображения. Вторым существенным отличием является то, что начиная приблизительно с 1995-1997 года, графические движки производят визуализацию с помощью графических процессоров, которые установлены на отдельных платах — видеокартах. Программные графические движки используют только центральные процессоры.

Как правило, графические движки не распространяются отдельно от игровых. Единственного графического движка без дополнительных компонентов и инструментария недостаточно для создания игры, поэтому разработчики движков продают лишь игровые движки с полным набором инструментов и компонентов. Однако это правило не относится к свободному программному обеспечению. Энтузиасты создают свободные графические движки и свободно их распространяют. Впоследствии разработчики игр могут объединить свободный графический движок с физическим, звуковым и другими компонентами и создать на основе их полноценный игровой движок.

Цель настоящей работы – разработка графического движка - программной системы реализующей визуализацию трёхмерных данных, на основе графической библиотеки OpenGL. Для достижения поставленной цели необходимо решить *следующие задачи:*

– провести анализ предметной области;

– разработать концептуальную модель программы;

– спроектировать программную систему;

– реализовать программную среду используя графическую библиотеку OpenGL.

Структура и объем работы. Отчет состоит из введения, 4 разделов основной части, заключения, списка использованных источников, 2 приложений. Текст выпускной квалификационной работы равен 110 страницам.

Во введении сформулирована цель работы, поставлены задачи разработки, описана структура работы, приведено краткое содержание каждого из разделов.

В первом разделе на стадии описания технической характеристики предметной области приводится сбор информации о деятельности компании, для которой осуществляется разработка сайта.

Во втором разделе на стадии технического задания приводятся требования к разрабатываемому приложению.

В третьем разделе на стадии технического проектирования представлены проектные решения для программы.

В четвертом разделе приводится список классов и их методов, использованных при разработке программной системы, производится тестирование разработанного сайта.

В заключении излагаются основные результаты работы, полученные в ходе разработки.

В приложении А представлены графические материалы.

В приложении Б представлен пример содержания файла описании геометрии трёхмерных объектов в формате OBJ.

В приложении В представлен пример содержания файлов Resources.data и Objects.data, хранящих данные о ресурсах и объектах в проекте.

В приложении Г представлены фрагменты исходного кода.

1 Анализ предметной области

1.1 История создания трёхмерной компьютерной графики

Впервые трёхмерная компьютерная графика была реализована в 1960-х годах, когда Айван Сазерленд и Дэвид Эванс основали первую в мире кафедру компьютерной графики в университете Юты, США. Основой трёхмерной компьютерной графики стала Евклидова геометрия, а также все математические открытия, которые были сделаны до XX века. Первые трёхмерные изображения состояли из множества точек и кривых, определяемых математическим уравнением. Именно в то время появился прародитель для всех современных 3D-редакторов - программа Sketchpad.

Развитие трёхмерной графики шло быстрым ходом, и уже совсем скоро вместо векторной графики появилась возможность создавать и трёхмерные растровые изображения, а также использовать различные текстуры для объектов, обрабатывать их тени, и наконец - проводить компиляцию всего кода в готовое изображение.

1.2 Основные принципы работы трёхмерной графики

Главным основоположником систематизации математических аксиом и теорем в области геометрии был Евклид. Именно его труды способствовали разработке и технологическому прорыву трёхмерной графики в XX веке. Но за её развитием стоит не только сам Евклид, а также и другие великие умы и открытия, как например, формулы Виета для нахождения корней квадратичного уравнения, благодаря чему в символьном анализе алгебры неизвестные переменные обозначаются как x, y и z, а коэффициенты - a, b и c. А основой отсчёта пространства стала система трёхмерных координат Декарта.

Также незаменимый влкад в разработку трёхмерной графики и геометрии внесли Российские учёные XX века - Борис Делоне и Георгий Вороной. Борис Делоне предложил метод «Триангуляции Делоне», которая стала основой формирования граней трёхмерных моделей, а Георгий Вороной создал

«Диаграмму Вороного», которая используется до сих пор в картографических софтах, дизайне и трёхмерной графике.

Любая трёхмерная модель объекта математически находится в трёх измерениях. И если, чтобы отрисовать изображение объекта в двух измерениях компьютерной мощи требовалось не так много, то с добавлением третьего измерения - оси Z, требовательность к производительности компьютерного железа резко возрасла.

Трёхмерные объекты математически представляются в виде множества точек - вершин, и в тоже время они, соединенные отрезками, образуют рёбра фигиры, а множество рёбер на одной плоскости образуют поверхности - грани фигуры. Данный массив точек трансформируется с учётом матрицы перспективы камеры, далее - накладываются текстуры на поверхности граней фигуры - изображения также растягивают и трансформируют с учётом перспективы наклона грани, затем - обрабатывают свет, падающий на модель, путём изменения яркости текстуры граней, в зависимости от источников света, и наконец - добавляют тень, которая отбрасывает данная трёхмерная модель.

1.3 Работа с OpenGL

”OpenGL”(Open Graphics Library) - это кроссплатформенная спецификация, независимая от языка программирования, которая определяет программный интерфейс для написания приложений для двумерной и трёхмерной компьютерной графики. По своей сути, OpenGL - это низкоуровневый API, который позволяет напрямую работать с коммандами и буфферами видеокарты, так что для его использования необходимо иметь хотя бы основные понимания работы трёхмерной графики и линейной алгебры. Для простого начала знакомства и использования спецификации OpenGL существует множество официальных готовых эффективных реализаций для Windows, Unix-платформ и MacOS.

Спецификация OpenGL была создана в эпоху активного распространения компьютерных 3D игр и приложений, когда взникла сложность с сов-

местимостью устройств во время реализации компьютерного кода при использовании различных аппаратных устройств - процессоров, видеоадаптеров, устройств хранения информации и материнской платы. Это вызывало сильные затраты, трудности и замедляло разработку программного обеспечения. Поэтому, компания Silicon Graphics - лидирующая в то время в сфере производства оборудования для обработки трёхмерной графики разработала программный интерфейс, целью которого было систематизировать доступ и обработку трёхмерных моделей на аппаратном уровне. Плодами их трудов стала спецификация OpenGL, который стандартизировал обработку различных функций 3D систем, которые выполняли единые команды из списка доступных, согласно установленной программой спецификации, что позволило создавать программное обеспечение, которое одинаково корректно работает на всех видах графического оборудования.

Принцип работы OpenGL заключается в получении геометрических векторных примитивов и построении растровой картинки в памяти видеокарты и выводе её на экран. Из-за своей низкоуровневости, данная спецификация требует диктовать программе точный порядок действий от программиста и использовать императивный подход в разработке приложений. Но несмотря на эти сложности, это даёт большой простор, гибкость и свободу в создании программ.

Последняя вышедшая версия OpenGL 4.6, в данный момент уже считается устаревшей, и на смену ей пришёл современный и оптимизированный для новых видеокарт API Vulcan, который стал прямым преемником OpenGL.

1.3.1 Функции и операторы OpenGL

Все функции OpenGL 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 позволяют 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.

1.3.2 Интерфейс OpenGL

OpenGL состоит из набора библиотек. Все базовые функции хранятся в основной библиотеке, для обозначения которой в дальнейшем буден использоваться аббревиатура GL. Помимо основной, OpenGL включает в себя несколько дополнительных библиотек.

Первая из них – библиотека утилит GL(GLU – GL Utility). Все функции этой библиотеки определены через базовые функции GL. В состав GLU вошла реализация более сложных функций, таких как набор популярных геометрических примитивов (куб, шар, цилиндр, диск), функции построения сплайнов, реализация дополнительных операций над матрицами и т.п.

OpenGL не включает в себя никаких специальных команд для работы с окнами или ввода информации от пользователя. Поэтому были созданы специальные переносимые библиотеки для обеспечения часто используемых функций взаимодействия с пользователем и для отображения информации с помощью оконной подсистемы. Наиболее популярной является библиотека GLUT (GL Utility Toolkit). Формально GLUT не входит в OpenGL, но включается почти во все его дистрибутивы и имеет реализации для различных платформ. GLUT предоставляет только минимально необходимый набор функций для создания OpenGL-приложения. Функционально аналогичная библиотека GLX менее популярна. В дальнейшем в этом пособии в качестве основной будет рассматриваться GLUT.

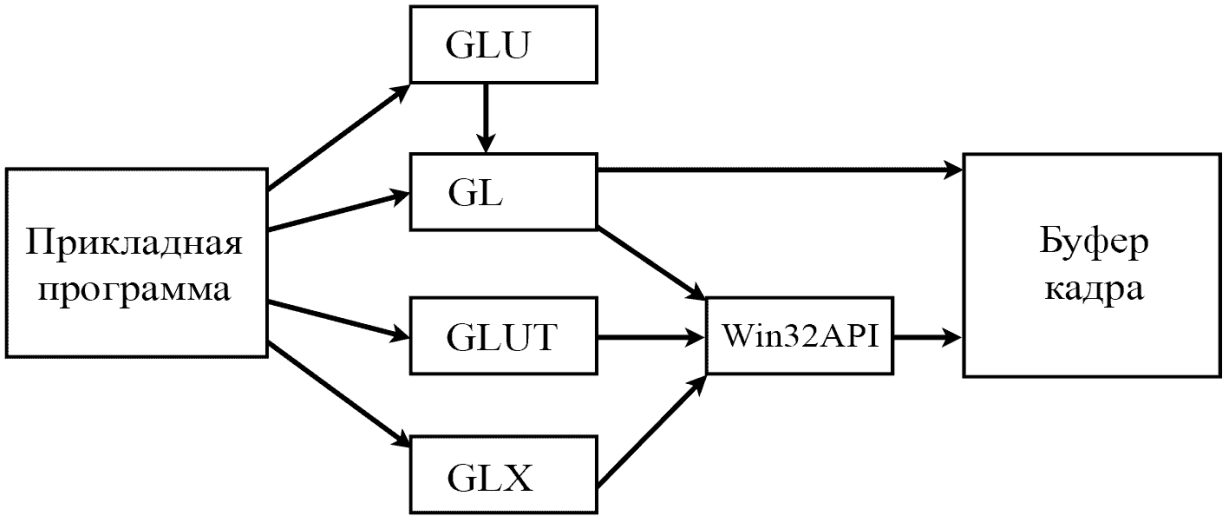


Рисунок 1.1 – Организация библиотеки OpenGL

Функции, специфичные для конкретной оконной подсистемы, обычно входят в ее прикладной программный интерфейс. Так, функции, поддерживающие выполнение OpenGL, есть в составе Win32 API и X Window. На рисунке схематически представлена организация системы библиотек в версии, работающей под управлением системы Windows. Аналогичная организация используется и в других версиях OpenGL.

1.3.3 Архитектура OpenGL

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

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

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

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

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

С точки зрения разработчиков, OpenGL – это набор команд, которые управляют использованием графической аппаратуры. Если аппаратура состоит только из адресуемого буфера кадра, тогда OpenGL должен быть реализо-

ван полностью с использованием ресурсов центрального процессора. Обычно графическая аппаратура предоставляет различные уровни ускорения: от аппаратной реализации вывода линий и многоугольников до нестандартных и непривычных графических процессоров с поддержкой множеств операций над геометрическими данными.

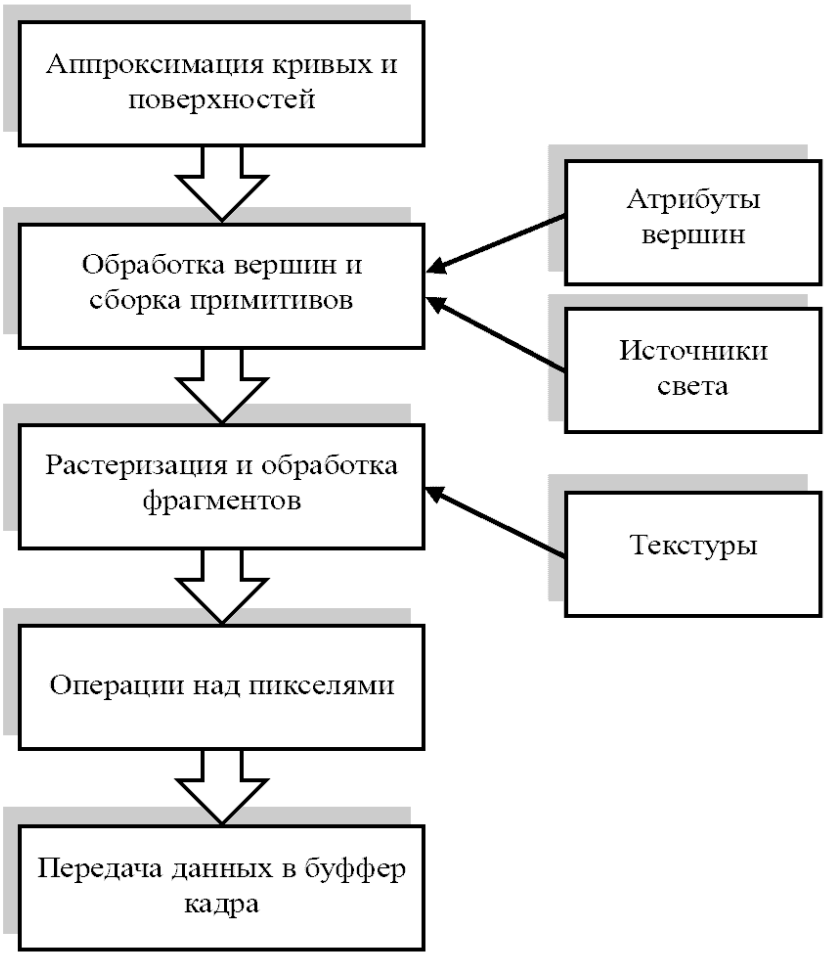


Рисунок 1.2 – Функционирование конвейера OpenGL

OpenGL является соединяющим звеном между аппаратурой и пользовательским уровнем, что позволяет предоставлять единый интерфейс на разных платформах, используя возможности аппаратной поддержки.

Кроме того, OpenGL можно рассматривать как конечный автомат, состояние которого определяется множеством значений специальных переменных и значениями текущей нормали, цвета, координат текстуры и других атрибутов и признаков. Вся эта информация будет использована при поступле-

нии в графическую систему координат вершины для построения фигуры, в которую она входит. Смена состояний происходит с помощью команд, которые оформляются как вызовы функций.

1.3.4 Работа с OpenTK

OpenTK (Open набор средств) — это расширенная низкоуровневая библиотека C#, которая упрощает работу с OpenGL, OpenCL и OpenAL. OpenTK можно использовать для игр, научных приложений или других проектов, требующих трехмерной графики, аудио или вычислительной функциональности.

Использование OpenTK предоставляет следующие возможности:

– Быстрая разработка — OpenTK предоставляет надежные типы данных и встроенную документацию для улучшения рабочего процесса программирования и перехвата ошибок проще и быстрее;

– простая интеграция — OpenTK была разработана для легкой интеграции с приложениями .NET;

– разрешительная лицензия — OpenTK распространяется в соответствии с лицензиями MIT/X11 и полностью бесплатно;

– расширенные привязки type-Сейф — OpenTK поддерживает последние версии OpenGL, OpenGL|ES, OpenAL и OpenCL с автоматической загрузкой расширений, проверка проверка и встроенной документацией;

– гибкие параметры графического интерфейса — OpenTK предоставляет собственное, высокопроизводительные окно игры и приложения;

– полностью управляемый, CLS-совместимый код . OpenTK поддерживает 32-разрядные и 64-разрядные версии macOS без неуправляемых библиотек;

– трехмерные математические набор средств поставки Quaternion Vector Matrix OpenTK и Bezier структуры с помощью 3D-математических набор средств.

1.4 Математика освещения в трёхмерной графике

Для того, чтобы воссоздать искусственное освещение внутри моделируемого виртуального пространства, нам необходимо смоделировать поведение света при взаимодействии с различными поверхностями. Интересно, что эту задачу начал решать ещё в далеком в 18-м веке человек по имени Иоганн Генрих Ламберт.

В 1760 году швейцарский учёный выпустил книгу под названием Photometria. В ней он изложил фундаментальные правила поведения света; самым примечательным из них стало следующее — поверхность испускает свет (отражением или как источник освещения) таким образом, что яркость испускаемого света меняется в зависимости от косинуса угла между поверхностью нормали и наблюдателем.

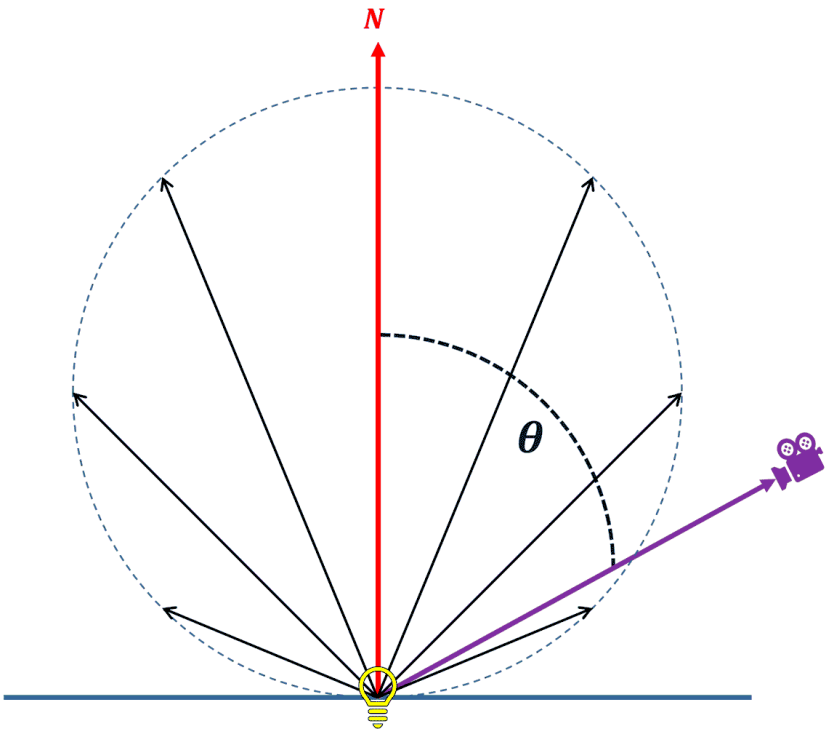


Рисунок 1.3 – Зависимость яркости отражаемого света поверхности от косинуса угла между нормалью и наблюдателем

1.4.1 Рассеянный свет

Это простое правило заложило основание определению рассеянного (diffuse light) освещения. Это математическая модель, используемая для вычисления цвета поверхности в зависимости от её физических свойств (например, её цвета и степени отражения света) и расположения источника освещения.

При 3D-визуализации для этого требуется некоторое множество параметров, что можно представить в виде такой схемы на рисунке 1.4.

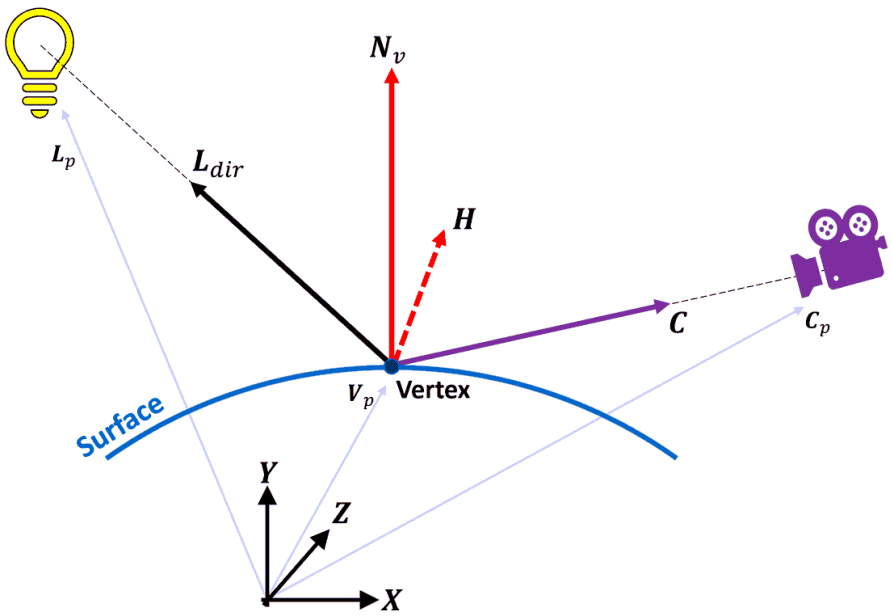


Рисунок 1.4 – Схема используемых параметров при 3D-рендеринге освещения

На изображении показано много указателей, это векторы, и для вычисления цвета отражения требуются следующие векторы:

– 3 вектора для позиции вершины, источника освещения и камеры, смотрящей на сцену;

– 2 вектора для направлений источника освещения и камеры с точки зрения вершины;

– 1 вектор нормали (изображён красным);

– 1 полувектор (он всегда посередине между векторами направлений освещения и камеры).

Они вычисляются на этапе обработки вершин процесса рендеринга, а объединяющее их всех уравнение (называемое ламбертовой моделью) (формула 1):

$$L_D = \sum_{0}^{n} (C_S \times C_L \times (\hat{N}_\nu \cdot \hat{L}_{dir}) \times A \times S) \tag{1}$$

То есть, цвет вершины при рассеянном освещении вычисляется перемножением цвета поверхности, цвета источника освещения и скалярного произведения векторов нормали вершины и направления света с коэффициентами затухания и прожекторного освещения. Эта операция выполняется для каждого источника освещения в сцене, отсюда и символ суммы в начале уравнения.

Всё это нужно для вычисления значения рассеянного освещения и все эти операции требуется выполнять для каждого источника освещения в виртуальной сцене, или, по крайней мере, для каждого источника, который существует на сцене и обрабатывается программой. Многие из этих уравнений выполняются графическими API, но их можно выполнять и вручную, если необходимо реализовывать более сложные операции над изображением.

1.4.2 Общий свет

Но помимо рассеянного света существует бесконечное множество других видов освещений, и каждая поверхность отражает свет, поэтому все они влияют на общее освещение сцены. Даже ночью присутствует фоновое освещение, будь то звёзды и планеты или свет, рассеивающийся в атмосфере.

Для моделирования реалистичного света, вычисляется ещё одно значение освещения: ambient lighting (освещение окружающей среды) (формула 2), или же общий свет.

$$L_A = C_{SA} \times \left(C_{GA} + \sum_{0}^{n} (A \times S \times C_{LA}) \right) \quad (2)$$

Это уравнение проще, чем для рассеянного освещения, потому что не требуются направления. Здесь выполняется простое перемножение различных коэффициентов:

– C_{SA} — цвет подсветки поверхности
– C_{GA} — цвет подсветки глобальной 3D-сцены
– C_{LA} — цвет подсветки всех источников освещения в сцене

1.4.3 Отраженный свет

Определив фоновое освещение и введя в расчет рассеянный свет источников освещения от различных поверхностей 3D-мира, виртуальное освещение начинает выглядеть намного более реалистично. Но модель Ламберта работает только для материалов, которые отражают освещение от своей поверхности во всех направлениях. Объекты, изготовленные из стекла или металла, создают другой тип отражения, называемый зеркальным (specular). И для него тоже необходимо определить уравнение (3).

$$L_S = C_S \times \sum_{0}^{n} \left(C_{LS} \times (\hat{N}_\nu \cdot \hat{H})^p \times A \times S \right) \quad (3)$$

Отдельные части этой формулы уже были определены раннее: имеется два значения зеркального цвета (одно для поверхности — C_S, другое для света — C_{LS}), а также привычные коэффициенты затухания и прожекторности.

Так как зеркальное отражение очень сфокусировано и направлено, для определения яркости зеркального освещения используются два вектора: нормаль вершины и полувектор. Коэффициент p называется мощностью зеркального отражения, это число, определяющее яркость отражения в зависимости от свойств материала поверхности. При увеличении p зеркальный эффект становится ярче, но более сфокусированным и меньшим по размерам.

1.4.4 Испускаемый свет

Последний учитываемый элемент — самый простой, потому что математически представляется лишь одним числом. Оно называется испускаемым (emissive) освещением, и применяется к объектам, являющимся непосредственным источником освещения, то есть к пламени, фонарику или Солнцу.

Это означает, что на данный момент определены одно число и три набора уравнений для вычисления цвета вершины поверхности, учёта фонового освещения (окружающей среды), а также взаимодействия между различными источниками освещения и свойствами материала поверхности (diffuse и specular). В программной среде можно использовать либо только один или же скомбинировать все четыре источника света, просто сложив их вместе (формула 4):

$$L_O = L_D + L_A + L_S + L_E \tag{4}$$

Визуально сочетание различных видов света выглядит так:

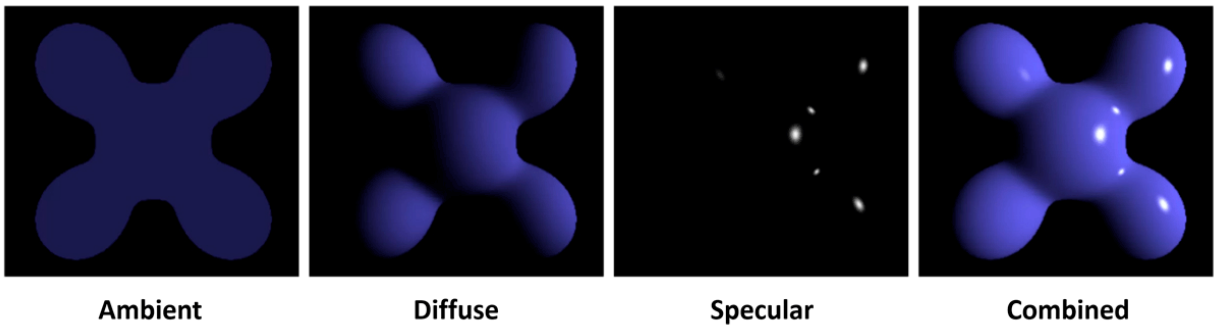


Рисунок 1.5 – Визуальное представление результата наложения нескольких типов освещений на одном объекте

1.5 Аффинные преобразования трёхмерного пространства

При работе с трёхмерными объектами, часто требуется совершать по отношению к ним различные преобразования: двигать, поворачивать, сжимать, растягивать, скашивать и т.д. При этом в большинстве случаев требу-

ется, чтобы после применения этих преобразований сохранялись определенные свойства.

Преобразование плоскости называется аффинным (от англ. affinity – родство), если

– оно взаимно однозначно;

– образом любой прямой является прямая.

Преобразование называется взаимно однозначным, если

– разные точки переходят в разные;

– в каждую точку переходит какая-то точка.

Свойства аффинного преобразования в трехмерном пространстве:

– отображает n-мерный объект в n-мерный: точку в точку, линию в линию, поверхность в поверхность;

– сохраняет параллельность линий и плоскостей;

– сохраняет пропорции параллельных объектов – длин отрезков на параллельных прямых и площадей на параллельных плоскостях.

Любое аффинное преобразование задается матрицей 3x3 с ненулевым определителем (формула 6) и вектором переноса (формула 5):

$$\vec{p'} = R\vec{p} + \vec{t} \tag{5}$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{6}$$

1.5.1 Параллельный перенос

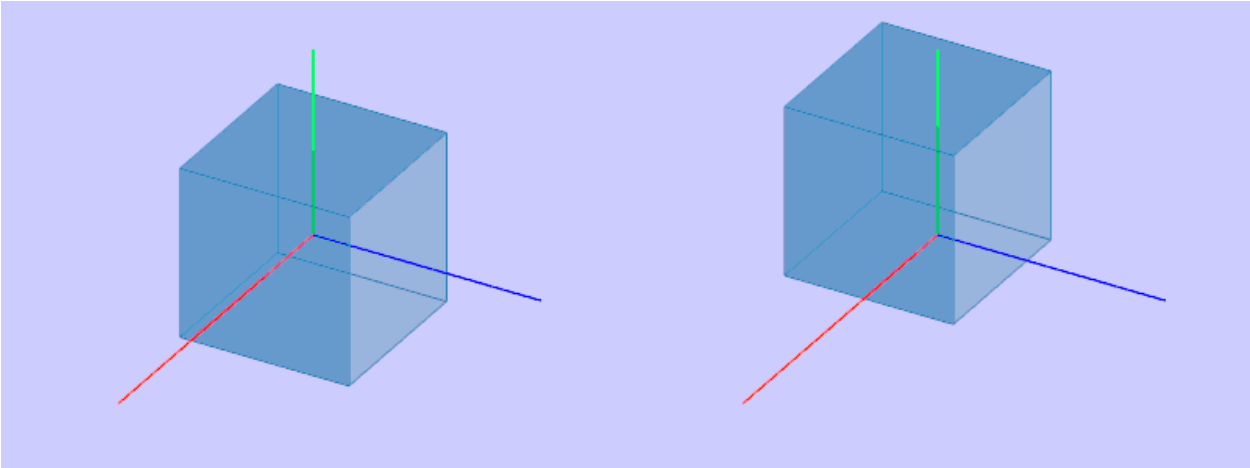


Рисунок 1.6 – Преобразование вида параллельный перенос

Матрица этого преобразования выглядит следующим образом:

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

В данном случае матрица R = E, единичной матрице.

1.5.2 Поворот вокруг оси

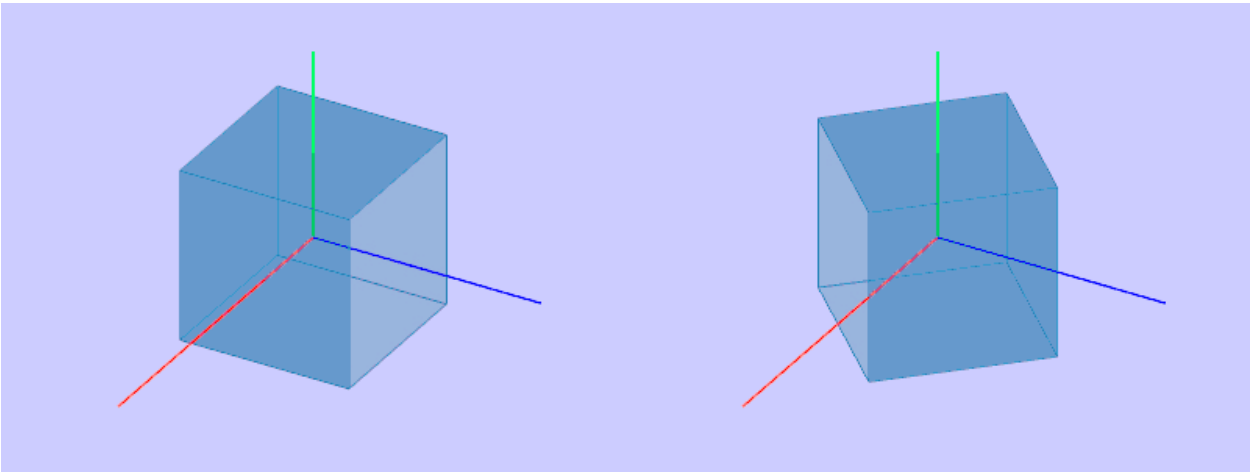


Рисунок 1.7 – Преобразование вида поворот вокруг оси

Заметим, что при повороте вокруг оси y ординаты точек (y-координаты) не меняются. Также стоит отметить, что координаты x и z точки преобразуются независимо от y-координаты. Это означает, что любая точка p(x, y, z) перейдет в точку p'(x'(x, z), y, z'(x, y)). Теперь осталось понять, как преобразуются координаты x и z: в плоскости Oxz это будет поворот вокруг начала координат по часовой стрелке (т.к. x z y - левая тройка), т.е. в отрицательном направлении. Матрица такого преобразования известна:

$$\begin{pmatrix} \cos(-\phi_y) & -\sin(-\phi_y) \\ \sin(-\phi_y) & \cos(-\phi_y) \end{pmatrix} \tag{8}$$

Матрица преобразования Ry(φy):

$$\begin{pmatrix} \cos(-\phi_y) & -\sin(-\phi_y) & \\ 0 & 1 & 0 \\ \sin(-\phi_y) & \cos(-\phi_y) & \end{pmatrix} \tag{9}$$

1.5.3 Масштабирование

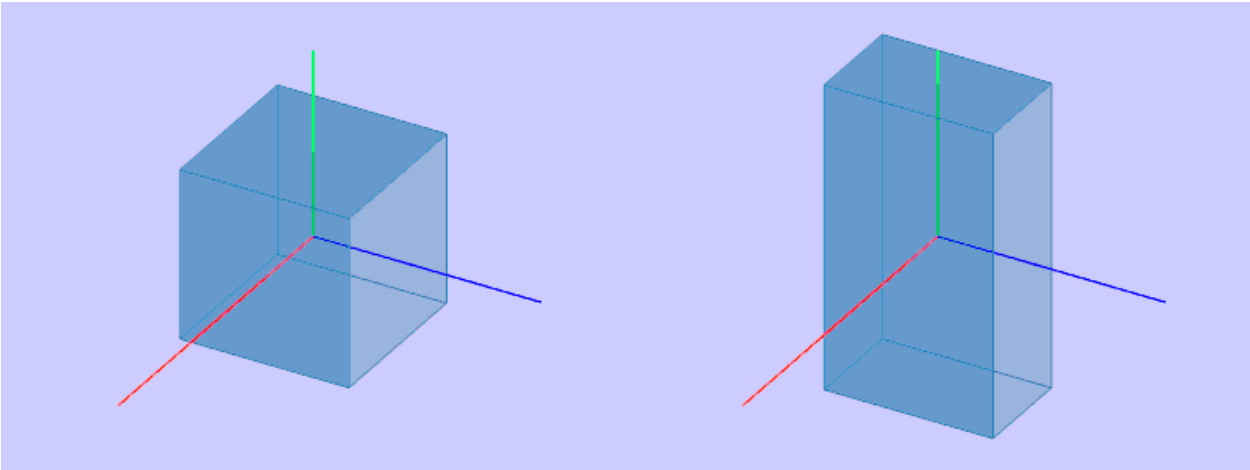


Рисунок 1.8 – Преобразование вида масштабирование

Коэффициенты сжатия/растяжения, по аналогии с двухмерным пространством, определяются диагональными членами матрицы R (формула 10).

$$\begin{pmatrix} S_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix} \tag{10}$$

Результат:

$$\begin{aligned} x^{'} &= s_x x \\ y^{'} &= s_y y \\ z^{'} &= s_z z \end{aligned} \tag{11}$$

Комбинация коэффициентов s_x = -1, s_y = 1, s_z = 1 будет задавать отражение от плоскости Oyz (x = 0). При s_x = s_y = s_z = -1 получим центральную симметрию относительно начала координат.

2 Техническое задание

2.1 Основание для разработки

Основанием для разработки является задание на выпускную квалификационную работу бакалавра «Программная система визуализации трехмерных данных, использующая библиотеку OpenGL».

2.2 Цель и назначение разработки

Основной задачей выпускной квалификационной работы является разработка программной системы для визуализации трёхмерных данных, использующая библиотеку OpenGL.

Используя библиотеку OpenGL последней версии, планируется разработать приложение на языке C#, способное импортировать и обрабатывать массивы трёхмерных данных и в дальнейшем визуализировать их в виде трёхмерных моделей.

Задачами данной разработки являются:

– разработка и проектирование основы графического движка;

– реализация программы парсеров, способных обрабатывать, преобразовывать и загружать в программу трёхмерные данные;

– разработка и реализация функции хранения трёхмерных данных внутри программы и их загрузки из внешней файловой среды;

– разработка интерфейса для взаимодействия с программой;

– создание готовой программы, реализующей графику, на основе спецификации и библиотеки OpenGL.

2.3 Требования к программной системе

2.3.1 Требования к данным программно-информационной системы

Входными данными для программной системы являются файлы с расширением *.obj, внутри которых содержится информация в виде массивов

трёхмерных данных; файлы текстур - изображения с расширением *.png и *.jpg; данные ввода с клавиатуры и мыши, которые служат для управления перемещением и вращением камеры.

Выходными данными для программной системы являются файлы данных, содержащие информацию о загруженных объектах и ресурсах в программную среду, а также выводимое на экран изображение двумерной проекции трёхмерных объектов на виртуальной сцене.

На рисунке 2.1 представлена диаграмма описания потоков данных в программе.

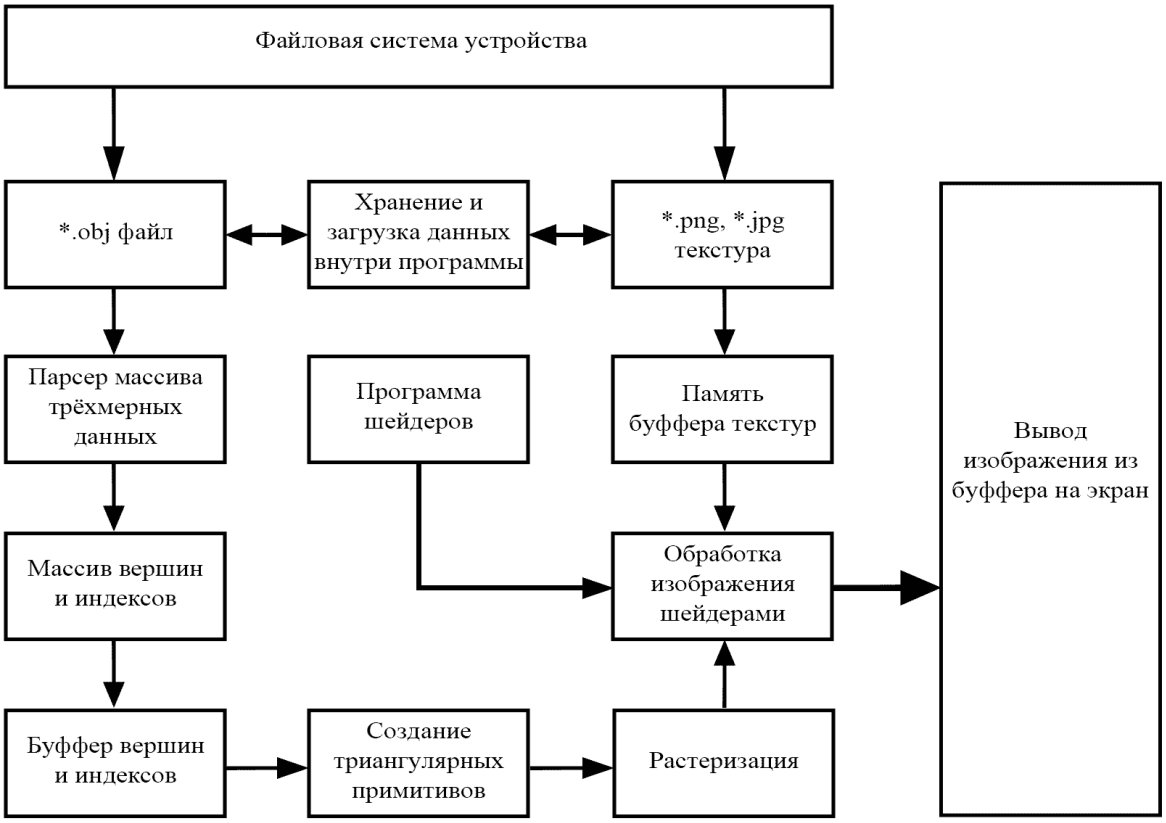


Рисунок 2.1 – Диаграмма потоков данных в программе

2.3.2 Функциональные требования к программной системе

Программа должна реализовывать следующие функции:

– позволять пользователю импортировать любые массивы трёхмерных данных (включая текстуры для трёхмерных моделей);

– выполнять отрисовку трёхмерной сцены в реальном времени (каждый кадр);

– пользволять пользователю свободно управлять перспективой виртуальной камеры;

– предоставлять возможность преобразовывать массив трёхмерных данных (производить аффинные преобразования) в рамках: сдвига, вращения и растяжения (сжатия);

– предоставлять возможность управлять настройками освещения - источником света (прямой свет, отраженный свет и рассеянный свет);

– импортировать, хранить и загружать массив трёхмерных данных внутри самой программы.

Виды преобразований массива трёхмерных данных, предоставляемых программой, представлены на рисунке 2.2.

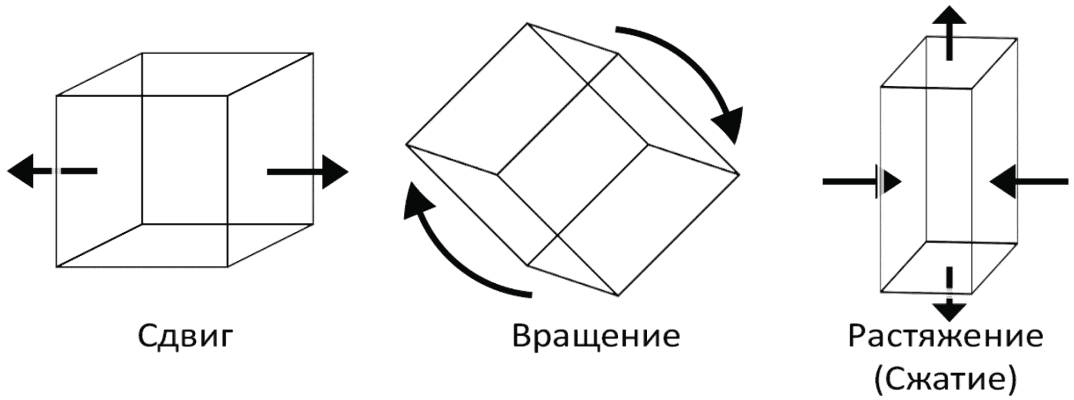


Рисунок 2.2 – Виды преобразований массива трёхмерных данных

Виды света, реализуемые программной системой, чьи настройки атрибутов предоставляются программной системой пользователю, представлены на рисунке 2.3.

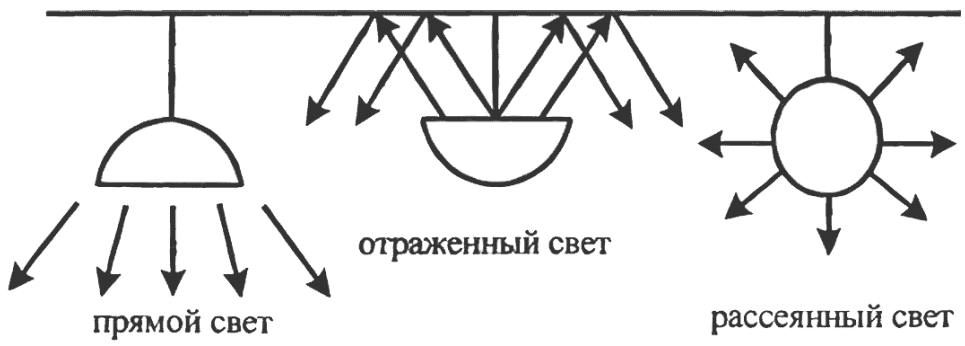


Рисунок 2.3 – Виды света, излучаемые источником освещения

2.3.3 Требования к графическому интерфейсу программы

Программное обеспечение должно иметь содержательный и интуитивно понятный интерфейс. Большую часть основного окна приложения должно занимать само окно вывода растрового изображения проекции виртуальной сцены и всех трёхмерных данных, загруженных в программу. Сбоку, поверх основного окна приложения должно находиться специально выведенное отдельное окно «инспектора», с помощью которого пользователь сможет управлять загрузкой и сохранением массивов трёхмерных данных, а также взаимодействовать с уже существующими данными на виртуальной сцене.

Макет пользовательского интерфейса, составленный по данным требованиям представлен на рисунке 2.4

Рисунок 2.4 – Макет пользовательского интерфейса

2.4 Моделирование вариантов использования

Для разрабатываемого программного обеспечения была реализована модель, которая демонстрирует наглядное представление вариантов использования программы.

На основании анализа предметной области в программе должны быть реализованы следующие прецеденты:

1. Импортирование массивов трёхмерных данных из файловой системы пользователя.
2. Просмотр визуализированного массива трёхмерных данных в виде отрисованных объектов на экране.
3. Осуществление трансформаций над массивом трёхмерных данных.
4. Настройка параметров источника света и общего освещения.
5. Хранение, загрузка и удаление массивов трёхмерных данных из программной системы.

На рисунке 2.5 представлена диаграмма вариантов использования

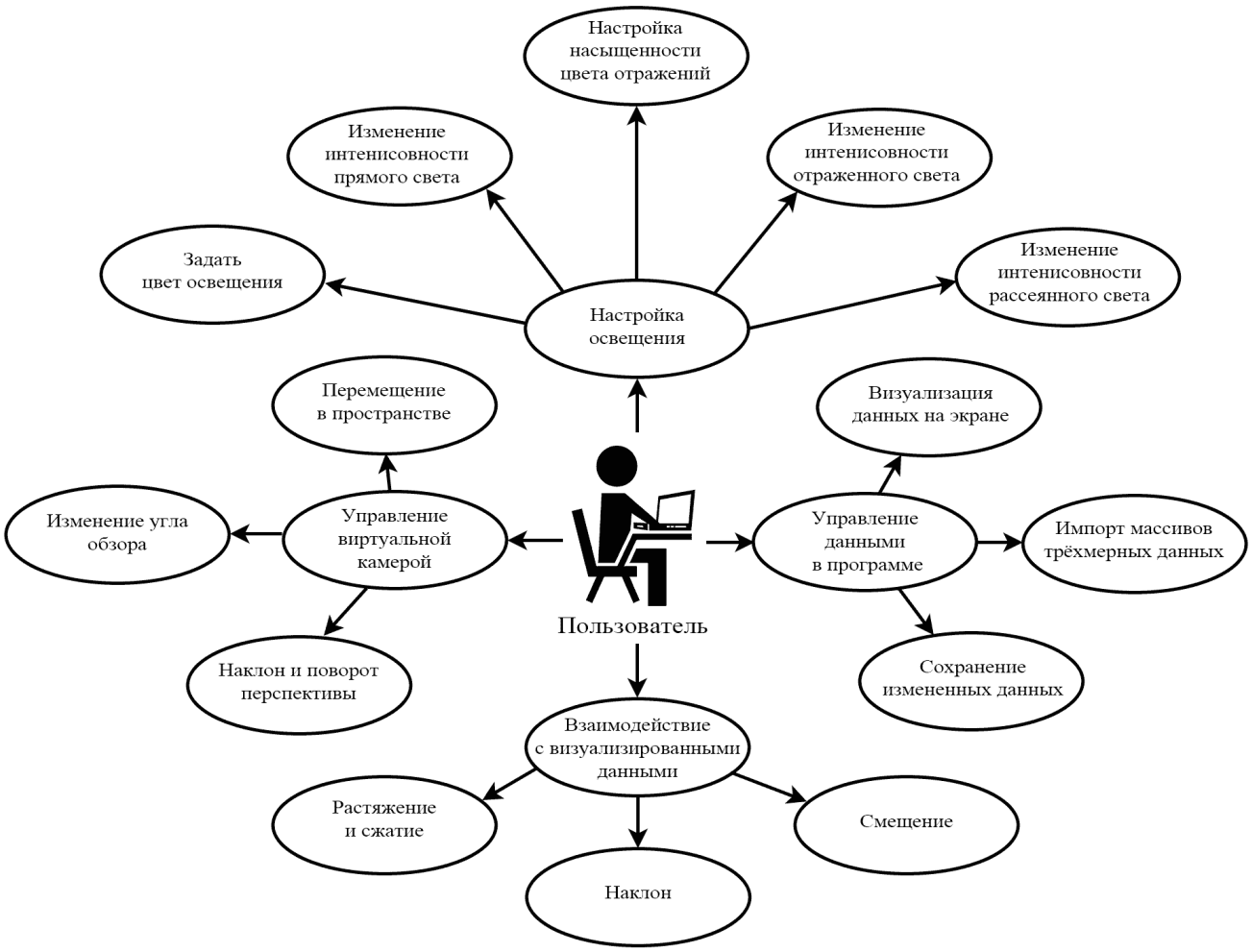


Рисунок 2.5 – Диаграмма вариантов использования

На данном рисунке представлена диаграмма, которая в общих чертах описывает модель вариантов использования программы. При взаимодействии с программой, пользователь будет осуществлять одно или сразу несколько действий из представленной модели вариантов использования. Для моделирования более конкретных случаев работы программы, было написано несколько сценариев вариантов использования.

2.4.1 Вариант использования: Визуализация данных на экране

Результат варианта использования: пользователь, выбрав из списка необходимый объект, загружает его на виртуальную сцену и наблюдает скомпилированную композицию трёхмерной сцены с объектом на ней в основном окне программы.

Сценарий использования описан в таблице 2.5

Таблица 2.1 – Сценарий варианта использования: Визуализация данных на экране.

Номер шага	Действующее лицо	Действие
1	Пользователь	Пользователь, находясь во вкладке инспектора «Объект», нажимает на выпадающее меню напротив надписи «Объект».
2	Система	Появляется список выпадающего меню, содержащий наименования всех хранящихся трёхмерных объектов в программе.
3	Пользователь	Пользователь кликает на строку с желаемым именем объекта в списке
4	Пользователь	Пользователь нажимает на кнопку «Загрузить».
5	Система	На экране пользователя отображается композиция трёхмерной сцены, на которой будет отображен выбранный раннее пользователем объект, а также источник освещения.

2.4.2 Вариант использования: Преобразование модели объекта

Результат варианта использования: пользователь, введя коэффициенты в соответствующие поля и подтвердив действия нажатием на кнопку, применяет аффиные преобразования к трёхмерной модели объекта всех трёх видов: смещение, наклон и сжатие, в результате чего трёхмерная модель объекта, находящегося на сцене, видоизменяется в соответствии с введёными значениями преобразования.

Сценарий использования описан в таблице 2.2.

Таблица 2.2 – Сценарий варианта использования: Преобразование модели объекта.

Номер шага	Действующее лицо	Действие
1	Пользователь	Пользователь, находясь во вкладке инспектора «Объект», нажимает на выпадающее меню напротив надписи «Объект».
2	Система	Появляется список выпадающего меню, содержащий наименования всех хранящихся трёхмерных объектов в программе.
3	Пользователь	Пользователь кликает на строку с желаемым именем объекта в списке
4	Пользователь	Пользователь нажимает на кнопку «Загрузить».
5	Система	На экране пользователя отображается композиция трёхмерной сцены, на которой будет отображен выбранный раннее пользователем объект, а также источник освещения.
6	Пользователь	Пользователь переключается на соседнюю вкладку инспектора с названием «Инспектор» нажатием на соответствующую кнопку на вкладке инспектора. Далее пользователь вводит желаемые значения коэффициентов аффинных преобразований в группы ввода строк под соответстующими метками «Позиция», «”Наклон», «Сдвиг» и нажимает на кнопку «Применить»
7	Система	К объекту, визуализированному на экране применились аффинные преобразования, с указанными пользователем коэффициентами

2.4.3 Вариант использования: Создание нового объекта

Результат варианта использования: пользователь, введя имя нового объекта, и выбрав из двух выпадающих списков необходимые модель и текстуру для объекта, создаёт новый объект, имеющий указанное имя, модель и

текстуру, который теперь хранится в программе и может будет в дальнейшем загружен на виртуальную сцену.

Сценарий использования описан в таблице 2.5

Таблица 2.3 – Сценарий варианта использования: Визуализация данных на экране.

Номер шага	Действующее лицо	Действие
1	Пользователь	Пользователь, находясь во вкладке инспектора «Объект», вводит желаемое имя для объекта в строку ввода, напротив надписи «Имя объекта».
2	Пользователь	Пользователь, нажимает на выпадающее меню напротив надписи «Модель объекта».
3	Система	Появляется список выпадающего меню, содержащий названия всех файлов трёхмерных моделей формата .obj.
4	Пользователь	Пользователь кликает на строку с желаемым файлом трёхмерной модели.
5	Пользователь	Пользователь, нажимает на выпадающее меню напротив надписи «Текстура объекта».
6	Система	Появляется список выпадающего меню, содержащий названия всех файлов изображений текстур формата .png и .jpg.
7	Пользователь	Пользователь кликает на строку с желаемым файлом изображения текстуры.
8	Пользователь	Пользователь нажимает на кнопку «Создать».
9	Система	На экране пользователя отображается всплывающее сообщение, информирующее пользователя об успешном создании нового объекта с указанным названием, моделью и текстурой. В программу записываются данные о новом объекте.

2.4.4 Вариант использования: Загрузка данных из файловой системы

Результат варианта использования: пользователь, выбрав необходимый файл трёхмерных данных из файловой системы устройства, загружает его в проект.

Сценарий использования описан в таблице 2.5

Таблица 2.4 – Сценарий варианта использования: Визуализация данных на экране.

Номер шага	Действующее лицо	Действие
1	Пользователь	Пользователь, перейдя во вкладку инспектора «Модели и текстуры», нажимает на кнопку «Выбрать файл модели...» внутри контекстной группы «Работа с моделями».
2	Система	Появляется диалоговое окно со средством выбора файла.
3	Пользователь	Пользователь в диалоговом окне файловой системы выбирает необходимый файл в указанном формате .obj.
4	Пользователь	Пользователь нажимает на кнопку «Загрузить модель».
5	Система	На экране пользователя отображается всплывающее сообщение, информирующее пользователя об успешной загрузки файла в проект.

2.4.5 Вариант использования: Изменение параметров света

Результат варианта использования: пользователь, введя необходимые значения интенсивности света применяет изменения к параметрам освещения источника света.

Сценарий использования описан в таблице 2.5

Таблица 2.5 – Сценарий варианта использования: Визуализация данных на экране.

Номер шага	Действующее лицо	Действие
1	Пользователь	Пользователь, перейдя во вкладку инспектора «Свет», вводит необходимые значения интенсивности определенного типа света в текстовые поля напротив надписей «Общий свет», «Рассеянный свет» и «Отраженный свет» соответственно.
2	Пользователь	Нажимает на кнопку «Применить».
3	Система	На всей виртуальной сцене меняется освещение, в соответствии с веденными параметрами света.

2.4.6 Вариант использования: Управление виртуальной камерой

Результат варианта использования: пользователь, используя ввод мыши и клавиатуры управляет камерой: перемещает камеру и осматривается в виртуальном пространстве.

Сценарий использования описан в таблице 2.6.

Таблица 2.6 – Сценарий варианта использования: Управление виртуальной камерой

Номер шага	Действующее лицо	Действие
1	Пользователь	Пользователь, наведя указателем мыши на любую область в пределах главного окна программы зажимает правую кнопку мыши и перемещает указатель
2	Система	Указатель становится неактивным, а виртуальная камера перемещает свой взгляд вслед движению мыши
3	Пользователь	Пользователь зажимает клавишу «W»
4	Система	Камера начинает двигаться в сторону направления взгляда

Продолжение таблицы 2.6

5	Пользователь	Пользователь зажимает клавишу «A»
6	Система	Камера начинает двигаться влево, относительно направления взгляда
7	Пользователь	Пользователь зажимает клавишу «S»
8	Система	Камера начинает двигаться в противоположную сторону от направления взгляда
9	Пользователь	Пользователь зажимает клавишу «D»
10	Система	Камера начинает двигаться вправо, относительно направления взгляда
11	Пользователь	Пользователь зажимает клавишу «ПРОБЕЛ»
12	Система	Камера начинает двигаться вверх, по координате +Y
13	Пользователь	Пользователь зажимает клавишу «CTRL»
14	Система	Камера начинает двигаться вниз, по координате -Y

2.5 Нефункциональные требования к программной системе

Требования к аппаратной совместимости:

– видеоадаптер с поддержкой OpenGL версии не ниже 4.6;

– минимальное разрешение экрана - 800x600 пикселей.

Требования к программной совместимости:

– операционная система x86 Windows 7 и выше;

– система, с установленными компонентами Microsoft Visual C++ версии 2015 года и выше.

2.6 Требования к оформлению документации

Разработка программной документации и программного изделия должна производиться согласно ГОСТ 19.102-77 и ГОСТ 34.601-90. Единая система программной документации.

3 Технический проект

3.1 Общая характеристика организации решения задачи

Поставлена цель разработать программу, работающую на спецификации OpenGL, способную загрузить массив трёхмерных данных и графически осуществить их визуализацию на экране для пользователя.

Трёхмерный графический движок на высоком уровне является программным движком, основной задачей которого является визуализация трёхмерной компьютерной графики, представляет из себя структуру основных классов - примитивов, являющихся основой трёхмерной графики: массив координат точек, массив нумерации граней, матрица проекциий, объект виртуальной камеры, набор текстур и программ шейдеров, которые все вместе осуществляют отрисовку объекта на трёхмерной сцене.

На более низком уровне трёхмерный движок представляет из себя набор комманд и программ для работы с видеоадаптерами и буфферами памяти системы. Из данного набора команд и программ была создана основа разрабатываемой программы, работающей под спецификацией OpenGL.

В движке используется разработанная программа парсера. Парсер считывает структуру файлов трёхмерных объектов. Массив трёхмерных данных считывается последовательно и обрабатываются синтаксическим анализатором. На основе данных считанных объектов формируются массивы координат вершин, индексов и координат текстур.

3.2 Обоснование выбора технологии проектирования

Обоснованием выбора технологии проектирования послужило задание на разработку, целью которой являлось создание программы, работающей на спецификации OpenGL. Соответственно, была выбрана последняя выпущенная версия OpenGL 4.6. А выбором среды и языка проектирования интерфейса и создания программы являлось наличие необходимых функций для проектирования интерфейса и программы графического движка у языка C# в среде Visual Studio.

3.3 Описание используемых технологий и языков программирования

В процессе разработки программного обеспечения было использовано программное средство IDE Visual Studio, а также использованы языки программирования C# - при создании интерфейса программы, C - при работе со спецификацией OpenGL, графическая библиотека OpenGL и библиотека OpenTK для упрощения работы и адаптации OpenGL на платформы Mono и .NET на языке C#.

3.3.1 Язык программирования C#

3.3.1.1 Особенности языка C#

C# - многоцелевой объектно-ориентированный язык программирования. Относится к семье с C-подобным синтаксисом, наиболее идентичен C++ и Java. Имеет более расширенный спект функций, чем у предшественника - C++, чем является намного проще в использовании, но также из-за своих удобств является более высокоуровневым, что делает его ориентированным в основном на разработку десктопных приложений.

Язык C# является немного более высокоуровневым, чем C++, поэтому менее тесно взаимодействует с аппаратной частью вычислительных систем, что в какой-то степени ограничивает его функционал в случаях разработки низкоуровневых приложений, где работа с памятью системы и прямое взаимодействие с аппаратной частью необходимо.

3.3.2 Спецификация OpenGL 4.6

OpenGL ориентирован на две задачи:

– Скрыть сложности адаптации различных 3D-ускорителей, предоставляя разработчику единый API;

– скрыть различия в возможностях аппаратных платформ, требуя реализации недостающей функциональности с помощью программной эмуляции.

3.3.2.1 Особенности спецификации OpenGL

– Гибкость, открытый код и низкие требования к ресурсам устройства;

– возможности данной спецификации позволяют разработчику создавать полностью уникальные и подстроенные под особую специфику задач приложения;

– устройство данной графической библиотеки позволяет запускать и поддерживать приложения с максимально возможной производительностью;

– это низкоуровневая библиотека, которая позволяет напрямую работать с аппаратным железом - управлять памятью системы и буфером видеоадаптера;

– самодостаточность спецификации даёт возможность не использовать дополнительные плагины и библиотеки в реализации визуализации графики;

– полная мультиплатформенность - OpenGL работает на всех платформах, языках и устройствах;

– низкоуровневость библиотеки - для неопытного, или же начинающего разработчика это может стать главной проблемой в работе с OpenGL, потому как для создания программного обеспечения, использующего спецификацию OpenGL, необходимы глубокие знания об основах трёхмерной графики и линейной алгебры, а также иметь минимальное представление об обмене данными в видеоадаптерах на аппаратном уровне;

– данная технология в настоящее время считается уже устаревшей, и была заменена более современным API Vulcan;

– с выходом новых видеокарт, их спектр возможностей, как и реализуемых функций расширился, так что новые функции, такие как DLSS и RTX не вошли в стандарт спецификации OpenGL.

3.4 Диаграмма классов и компонентов программы

3.4.1 Диаграмма классов

Диаграмма классов описывает виды классов программы и различного рода связи, которые существуют между элементами. На диаграммах изображаются также атрибуты классов, их функции, принимаемые значения, а также ограничения описывающие взаимодействие между классами. Вид и представление диаграммы классов напрямую зависит от уровня абстракции: классы могут быть показаны в качестве сущностей предметной области или же как элементы частей программного обеспечения. В нашем случае, на рисунке 3.1 показана диаграмма классов, которые являются элементами программной среды.

Атрибуты в диаграмме описывают свойства объектов класса. Элементы класса обладают своей индивидуальностью из-за различий в их параметрах, а также связях с другими объектами.

В большинстве случаев, в данном программном обеспечении большинство классов существуют в единственном экземпляре, из-за их уникальной сущности. Как пример, можно привести класс, определяющий виртуальную камеру, которая должна существовать в единственном экземпляре, ведь на один экран должно выводиться одновременно лишь одно изображение.

Но несмотря на это, структура классов в данном программном обеспечении всё равно является объектно ориентированной, благодаря чему внутри неё был реализован механизм управления объектными моделями, текстурами и другими элементами, которые все связаны между собой наследуемостью классов, а также существует возможность дальнейшего развития и расширения общей системы классов, путём добавлением в неё новых элементов.

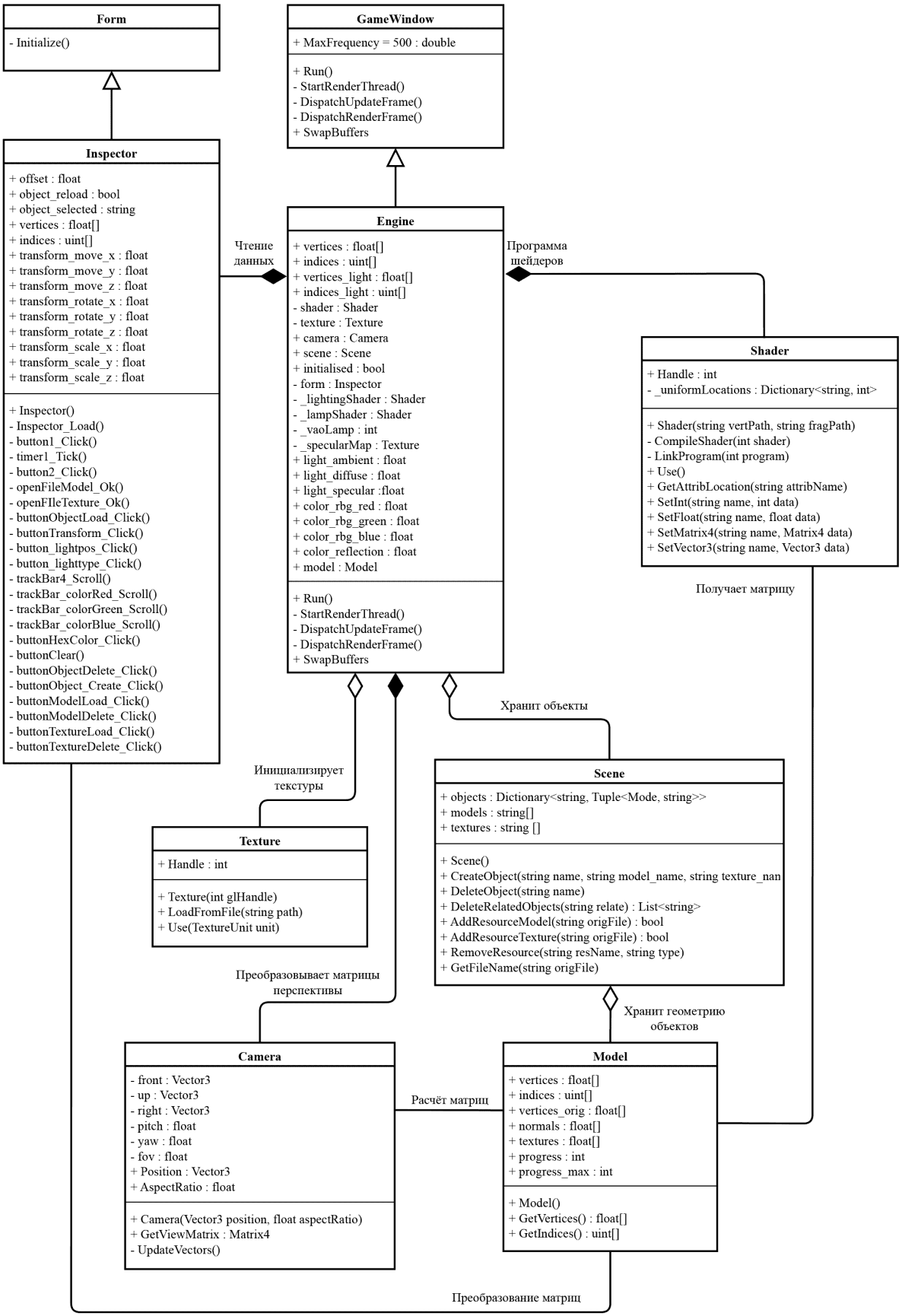


Рисунок 3.1 – Диаграмма классов программы

Все элементы программы имеют общий родительский элемент в виде главного класса - обработчика событий, который является главным элементом и центром всей программы.

3.4.1.1 Класс Model

Является каркасом трёхмерной модели. Выполняет роль приёма, хранения и передачи массива трёхмерных данных для обработчика триангулярных примитивов. При создании выполняет чтение и преобразование в массивы вершин и индексов, загруженного файла с данными трёхмерного объекта.

3.4.1.2 Класс Texture

При инициализации выделяет место в буффере памяти видеоадаптера для загруженной в программу текстуры, а также обрабатывает настройки параметров её отображения;

3.4.1.3 Класс Scene

Выполняет роль хранилища моделей и изображений, а также выступает виртуальной сценой, на которой находятся все трёхмерные объекты и остальные элементы, учавствующие в визуализации. При инициализации выполняет загрузку данных массивов моделей и привязанных к ним файлов текстур, которые уже были раннее загружены в проект;

3.4.1.4 Класс Shader

Программа настройки конечной визуализации, и при инициализации создаёт подпрограмму в общей графике, для преобразования растеризированного изображения внутри буффера памяти видеоадаптера, для отображения текстур и освещения;

3.4.1.5 Класс Camera

Является матрицей преобразования, для корректного отображения перспективы виртуального трёхмерного вида сцены, и для того, чтобы конечный вид проекции сцены был перспективным, а не ортографическим;

3.4.1.6 Класс Model

Представляет из себя основу проекта - графический движок, который инициализирует всю графику, принимает события ввода пользователя, отвечает за обновление кадров и управление всеми настройками, а также связывает все классы между собой;

3.4.1.7 Класс Inspector

Часть пользовательского интерфейса, представляющего из себя окно взаимодействия между пользователем и программой. Служит для того, чтобы пользователь осуществлял загрузку собственных массивов трёхмерных данных моделей и текстур в проект, а также осуществлял указанные аффинные преобразования над загруженными моделями.

3.4.1.8 Класс GameWindow

Также отдельно стоит указать класс GameWindow - это родительский класс для класса Engine. По своей сути он является измененным элементом класса Form от WindowsForms и не принимает прямого участия в работе проекта, но содержит формальные данные, необходимые для корректной инициализации главного окна проекта.

3.4.2 Диаграмма компонентов

Диаграмма компонентов - это структурная диаграмма языка унифицированного моделирования, она описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитек-

туру разрабатываемой системы, установив зависимости между программными компонентами.

Диаграмма компонентов предоставляет общую картину архитектуры системы, помогает разработчикам и архитекторам лучше понять ее структуру и взаимосвязи, а также является полезным инструментом для коммуникации и документирования архитектурных решений.

Диаграмма компонентов разрабатывается для следующих целей:

– визуализация общей структуры исходного кода программной системы;

– спецификация исполнимого варианта программной системы;

– обеспечение многократного использования отдельных фрагментов программного кода;

– представление концептуальной и физической схем баз данных.

На рисунке 3.2 изображена диаграмма компонентов программной системы.

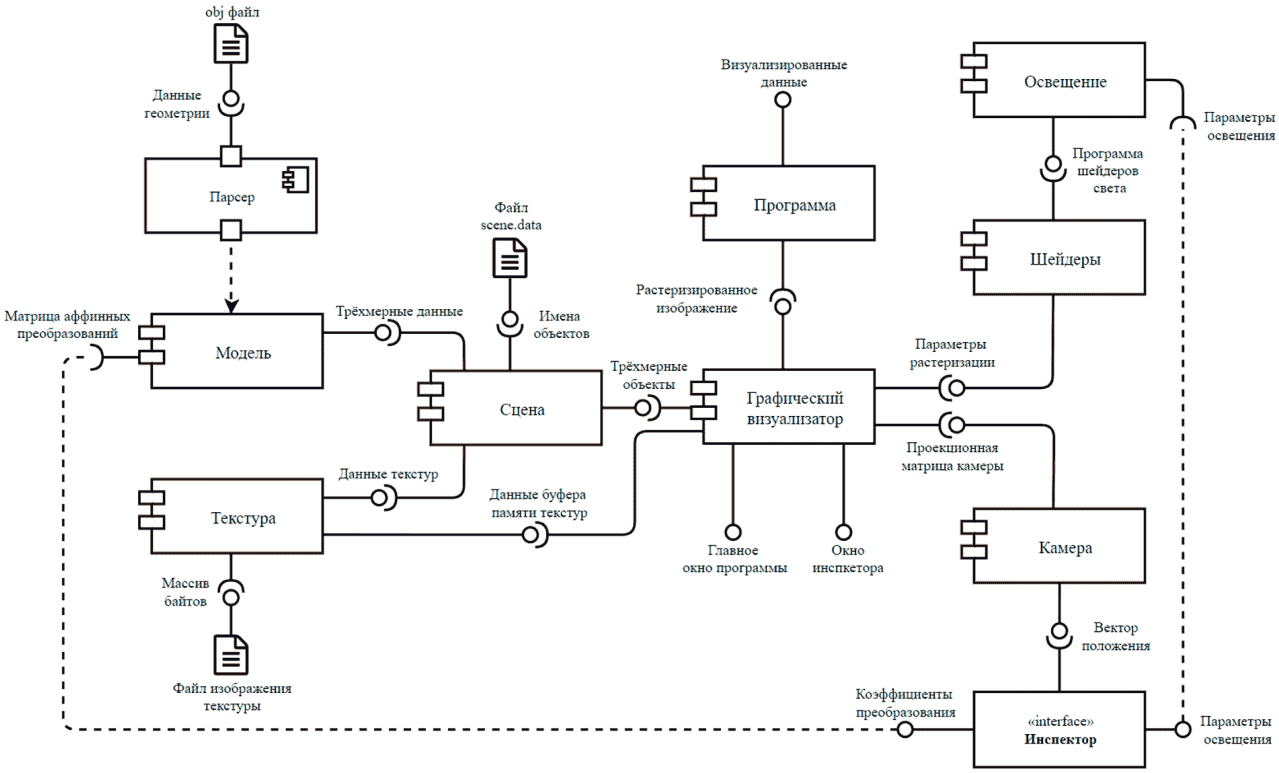


Рисунок 3.2 – Диаграмма компонентов программы

Данная диаграмма определяет общую архитектуру приложения, все его компоненты, а также устанавливает зависимости между ними в программной системе. Далее мы подробно опишем работу каждого компонента по отдельности.

3.4.2.1 Компонент Файл OBJ

Является файлом с данными, расширения *.obj, содержащим данные трёхмерной геометрии объекта.

Предоставляет общий набор данных геометрии трёхмерной модели.

3.4.2.2 Компонент Парсер

Синтаксический анализатор, преобразующий общий набор данных геометрии в отдельные массивы, подходящие по стандарту программной системы для корректной визуализации объектов.

Принимает общие данные геометрии трёхмерных моделей.

Является компонентом, зависимым от модуля Модель.

3.4.2.3 Компонент Модель

Выполняет функцию обработки и хранения массивов трёхмерных данных. Преобразовывает, хранит и передаёт данные файла геометрии трёхмерного объекта.

Предоставляет массивы трёхмерных данных, понятных в интерпретации для программной среды.

3.4.2.4 Компонент Файл изображения текстуры

Является файлом растрового изображения в форматах *.png или *.jpg.

Предоставляет закодированное изображение текстуры в виде массива байтов.

3.4.2.5 Компонент Текстура

Обработчик изображения, который генерирует текстуру в системе и задаёт параметры для отрисовки и фильтрации изображения. Выделяет место в буффере текстур, а также привязывает ей определенный идентификатор в памяти буфера видеоадаптера.

Принимает массив байтов изображения текстуры.

Предоставляет общие данные текстуры в системе - её идентификатор и адрес в буфере памяти.

3.4.2.6 Компонент Файл scene.data

Файл в который программа записывает и хранит данные идентификаторов, имён и ссылок на модели и текстуры.

Передаёт набор синтаксически размеченных строк, определяющих имена моделей и текстур.

3.4.2.7 Компонент Сцена

Виртуальная сцена, которая представляет из себя словарь моделей и текстур, связывающий их идентификаторы в общую структуру объекта.

Принимает трёхмерные данные моделей, системы данные и параметры текстур, а также имён - идентификаторов моделей и текстур.

Передаёт обработанные и структурированные данные трёхмерных объектов, готовых к визуализации.

3.4.2.8 Компонент Инспектор

Часть графического интерфейса программы, через который пользователь может производить взаимодействия над параметрами виртуальной среды и трёхмерными объектами, загруженными на виртуальную сцену.

Принимает данные вектора положения камеры.

Предоставляет коэффициенты для осуществления трансформации матриц аффинных преобразований. Также предоставляет параметры настройки освещения.

3.4.2.9 Компонент Камера

Описывает матрицу перспективы, по которой происходит создание перспективы виртуальной среды.

Предоставляет проекционную матрицу перспективы.

3.4.2.10 Компонент Освещение

Программа шейдера, отвечающая за обработку отражения света, а также уровня яркости грани модели, в зависимости от положения источника света.

Принимает параметры для настроек аттрибутов освещения.

Предоставляет программу шейдеров света.

3.4.2.11 Компонент Шейдеров

Общая программа шейдеров, отвечающая за обработку отображения цвета и текстур вершин и граней моделей.

Принимает программу шейдеров света.

Предоставляет параметры шейдинга для финальной растеризации визуализированной сцены.

3.4.2.12 Модуль Графический визуализатор

Главный блок программы, представляющий собой графический визуализатор, связывающий все модули программной среды, обрабатывая все необходимые данные для создания композиции и визуализации сцены трёхмерной среды.

Принимает обработанные массивы данных трёхмерных объектов, данные буфера памяти текстур, параметры программ шейдеров, а также матрицу проекции камеры.

Предоставляет графический интерфейс в виде главного окна программы и окна инспектора. Также передаёт конечное растрированное изображение в главный класс программы.

3.4.2.13 Компонент Программа

Является начальным родительским компонентом всей программной среды, который осуществляет запуск приложения с определенными параметрами. Осуществляет запуск графического визуализатора с заданными аттрибутами.

Принимает данные графического визуализатора в виде растеризированного кадра.

Визуализирует растеризированную композицию кадра виртуальной сцены у пользователя на экране в главном окне программы.

3.5 Описание файлов трёхмерных данных формата OBJ

Формат файлов OBJ, или же Wavefront .obj файл - это формат файлов описания геометрии, разработанный в 1992 году компанией компьютерной графики Wavefront Technologies. Он содержит только 3D геометрию, а именно: позицию каждой вершины, связь координат текстуры с вершиной, направления нормалей, а также параметры, которые создают триангулярные примитивы. На основе файлов данного формата, завязан главный принцип работы разрабатываемой программной среды.

Данный формат файла имеет следующие особенности:

– он позволяет пользователям использовать файловое представление объектов сложной или неевклидовой формы, разделяя поверхность на треугольные грани, которые называются триангулярными примитивами. Данный процесс трасселяции упрощает процесс манипуляции, моделирования и визуализации, поскольку позволяет изменять каждую грань независимо от остальных;

– ещё одной важной особенностью является способность определять свойства поверхности трёхмерной геометрии объектов, такие как определение координат текстур и карты нормалей для шейдинга;

– OBJ поддерживает данные высокого разрешения, в сравнении со схожими форматами трёхмерных данных, к примеру, такими как STL;

– и в заключении, ещё одной особенностью данного формата является возможность хранения сразу нескольких текстур и цветов в одном объекте, в отличие от файлов, как в том же формате STL.

Несмотря на то, что данный формат расширения файлов используется уже более тридцати лет, он не является устаревшим, а даже более того - почти основным и универсальным форматом хранения данных трёхмерной геометрии.

Существует множество способов открыть OBJ-файл и преобразовать его в различные другие форматы. Одним из вариантов является использование такого программного обеспечения, как 3DS Max, Solidworks, Cinema 4D или Blender, которое позволяет легко импортировать 3D-модели в формате OBJ, а затем преобразовывать и экспортировать их в любой необходимый формат.

Также данные объекта в OBJ-файле можно редактировать даже не запуская ни одно из программных обеспечений для трёхмерной графики, а просто открыв его в блокноте. Данный формат не закодирован какой-либо особой системой, и представлен в виде простого незашифрованного набора текста - данных трёхмерной геометрии. Пример содержания OBJ-файла представлен в приложении Б.

3.6 Описание работы парсера

В результате необходимости импортирования и использования в программной среде файлов трёхмерных данных с расширением *.obj из внешней файловой системы устройства, возникла необходимость автоматизировать загрузку и интерпретацию этих данных для программы. Для решения данной задачи был разработан специальный алгоритм программы парсера.

Парсер, или же синтаксический анализатор - часть программы, которая преобразовывает текстовые входные данные в структурированный формат.

Исходя из этого, необходимо, чтобы парсер мог разделить все эти данные на отдельные массивы, которые будет использовать программа. В заключение вышесказанного, можно построить схему, по которой должен будет работать парсер, как показано на рис. 3.3, где изображена схема работы подпрограммы парсера.

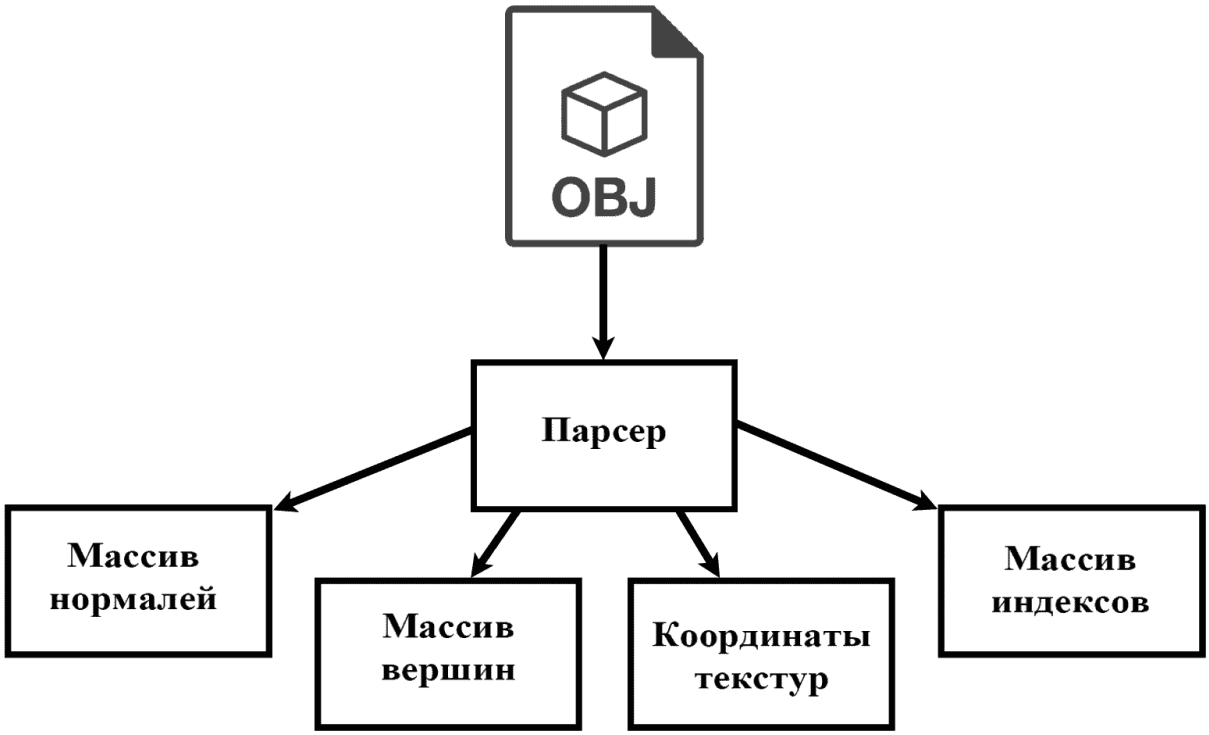


Рисунок 3.3 – Схема работы программы парсера

На данной схеме показано, на какие массивы данных разбивается OBJ-файл в результате синтаксического анализа парсера. Данные массивы необходимо будет использовать по отдельности, чтобы каждый был задействован в соответствующих модулях и подпрограммах программной среды.

Парсер не представляет собой отдельный независимый класс. Он является компонентом модуля, и его подпрограмма находится в классе Model. Это создаёт удобство в использовании данного класса, так как в следствии, каждый его элемент будет иметь возможность автоматически интерпретировать и структурировать данные геометрии своей модели в массивы трёхмерных

данных, в результате чего, они сразу смогут структурироваться под стандарты программы, при инициализации элемента класса Model.

Фрагмент листинга кода парсера представлен в ПРИЛОЖЕНИИ Г.

3.7 Система хранения данных внутри проекта

Программная система имеет начальный набор данных - текстуры и файлы геометрии, чтобы пользователь мог начать ознакомление с программой со встроенных данных, без необходимости подгружать дополнительные файлы из системы для полноценной работы программы. В ситуации, в которой количество хранимых файлов приложением больше одного, возникла необходимость спроектировать структурированную систему индексирования и хранения всех данных, чтобы программа понимала, какие данные в текущий момент пользователь планирует использовать. Для того, чтобы приложение могло работать с большим количеством данных и файлов, была разработана система хранения данных, которая имеет свои принципы и особенности. К файлам проекта программная система обращается через два файла данных - «Objects.data» и «Resources.data».

3.7.1 Файл Resource.data

В файле «Resources.data» программа записывает и считывает массив всех хранимых программой ресурсов - файлов геометрии и текстур. Каждая строка представляет собой отдельный ресурс - либо файл геометрии, либо текстуры, в которой записано их имя - относительный путь к файлам внутри проекта. Хранение ресурсов осуществляется по принципу листа, который проще говоря, является динамически расширяемым массивом данных, который можно расширять с конца. Данные ресурсы используются для дальнейшего представления объектов. Пример содержания и синтаксиса файла приведён в ПРИЛОЖЕНИИ В.

3.7.2 Файл Objects.data

В файле «Objects.data» программа записывает и считывает структуру объекта. Каждая строка файла представляет собой отдельный объект, который состоит из трёх частей - имени объекта, файла геометрии OBJ и файла изображения текстуры. Хранение объектов осуществляется по принципу словаря, где к каждому имени объекта привязаны свои определенные модели и текстуры, что собственно и является самими представлением объекта. Пример содержания и синтаксиса файла приведён в ПРИЛОЖЕНИИ В.

3.8 Проектирование пользовательского интерфейса

Графический прототип интерфейса, показанный на рис. 3.4 демонстрирует, какие элементы пользовательского интерфейса будут включены в конечную реализацию программы.

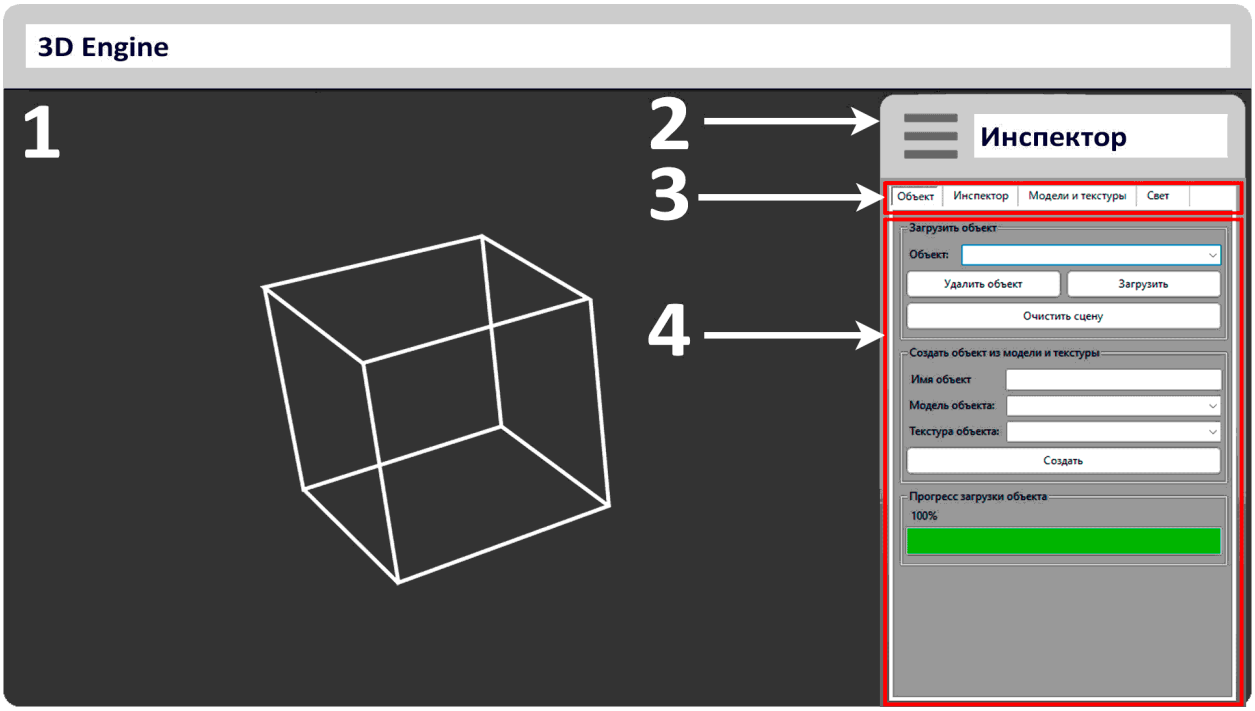


Рисунок 3.4 – Прототип пользовательского интерфейса

Описание элементов интерфейса, показанного на рис. 3.4:

1. Главное окно программы. Выводит визуализированные данные на экран.

2. Окно инспектора.

3. Вкладки окон инспектора.
4. Активное меню вкладки инспектора.

3.8.1 Главное окно программы

Главное окно программы представляет из себя виртуальную трёхмерную среду с одноцветным фоном. Оно служит для отображения визуализации трёхмерных моделей, которые пользователь загрузит в программу. Также данное окно является частью пользовательского интерфейса с интуитивной функцией ввода - пользователь может управлять направлением перспективы взгляда камеры, зажав правую кнопку мыши и направляя курсор в нужную сторону, чтобы управлять камерой и осматриваться в виртуальном пространстве. А используя колесо прокрутки, пользователь может отдалять и приближать угол обзора.

Управление перемещением камеры в пространстве также осуществляется в главном окне программы, с помощью клавиатуры:

– клавиша «W» – переместить камеру вперёд по направлению взгляда;

– клавиша «A» – переместить камеру вправо относительно направления взгляда;

– клавиша «S» – переместить камеру назад по направлению взгляда;

– клавиша «D» – переместить камеру влево относительно направления взгляда;

– клавиша «левый Shift» – поднять камеру вверх, по координате +Y;

– клавиша «левый Ctrl» – опустить камеру вниз, по координате -Y;

3.8.2 Окно инспектора

Инспектор - это дополнительное окно, расположенное по краю справа, поверх главного окна. Оно служит вспомогательным элементом интерфейса, чтобы пользователь мог взаимодействовать и управлять элементами внутри виртуальной сцены. Например, выполнять преобразование массива данных трёхмерной модели - перемещать её в пространстве, наклонять под опреде-

лённым углом и прозводить растяжение или сжатие, и всё это по всем трём координатам.

3.8.3 Вкладки окон инспектора

Меню со вкладками окон инспектора, где пользователь может выбрать одну трёх вкладок, чтобы переключать и менять содержание активного меню инспектора.

3.8.4 Активное меню вкладки инспектора

Показывает основной интерфейс меню инспектора. Инспектор имеет три разных активных меню, и содержание окна инспектора будет меняться, в зависимости от выбранной вкладки.

3.8.4.1 Вкладка «Объект»

Вкладка «Объект» - в данной вкладке пользователь может выбрать из выпадающего списка объект, который он хочет загрузить и визуализировать на виртуальной сцене. Также данная вкладка имеет ещё один важный раздел, в котором пользователь может создать новый объект внутри проекта - связав выбранный файл из списка моделей и файл из списка текстур, имеющихся внутри проекта, в новую сущность, дав имя получившемуся объекту.

3.8.4.2 Вкладка «Инспектор»

Вкладка «Инспектор» - это информационная вкладка, в которой показаны точные координаты и наклон камеры, а также в которой пользователь может осуществить аффиинные преобразования над загруженной моделью на сцену в данный момент и узнать данные об уже раннее произведенных преобразованиях над моделью: смещении, наклона и масштабировании.

3.8.4.3 Вкладка «Модели и текстуры»

Вкладка «Модели и текстуры» содержит весь список доступных моделей и текстур в программе, загруженных пользователем. Также в ней пользо-

ватель может осуществлять непосредственно саму загрузку массивов трёхмерных данных с расширением *.obj в программу и изображений текстур с расширением *.png и *.jpg. Также в данной вкладке пользователь может удалить уже существующие файлы моделей и текстур из проекта.

3.8.4.4 Вкладка «Свет»

Во вкладке «Свет» пользователю представлены текущие настройки параметров всех типов освещений, излучаемых источником света, которые пользователь может свободно изменять, чтобы получить желаемый результат отрисовки освещения отраженного и рассеянного света то поверхности объекта.

4 Рабочий проект

4.1 Классы, использованные при разработке программной системы

В результате разработке программной системы было спроектировано множество классов и методов, которые продемонстрированы в таблице 4.1).

Таблица 4.1 – Описание классов, используемых в приложении

Название класса	Описание класса	Методы
1	2	3
Engine	Engine – класс программы, представляющий собой графический визуализатор, связывающий между собой основные модули и классы программной среды, обрабатывая все необходимые данные для создания композиции и визуализации сцены трёхмерной среды.	**void Run_Inspector()** Запускает окно Инспектора **void OnLoad()** Событие, вызываемое при полной загрузке программы, определяющее первоначальную визуализацию пустой сцены **void Initialize()** Метод инициализации и отрисовки объектов на виртуальной сцене **void OnRenderFrame(Frame EventArgs e)** Событие, вызываемое каждый раз при создании нового кадра приложением **void OnUpdateFrame(FrameEventArgs e)** Событие, вызываемое каждый раз при изменении или обновлении кадра пользователем

Продолжение таблицы 4.1

1	3	4
		void ReloadScene() Метод, перезагружающий виртуальную сцену, перерисовывая заново все объекты на сцене **void OnMouseWheel(MouseWheelEventArgs e)** Событие, вызываемое при прокрутке колеса мыши **void OnResize(ResizeEventArgs e)** Событие, вызываемое при изменении размера окна приложения **void OnClosing(CancelEventArgs e)** Событие, вызываемое при закрытии главного окна приложения **OnInspectorClose(object sender, EventArgs e)** Событие, вызываемое при закрытии окна инспектора
Program	Program – главный класс программы, который задаёт настройки и запускает класс графического визуализатора	**void Main()** Главный метод программы

Продолжение таблицы 4.1

1	3	4
Scene	Класс Scene выполняет роль хранилища моделей и изображений, а также выполняет роль виртуальной сцены, на которой находятся все трёхмерные объекты и остальные элементы, участвующие в визуализации. При инициализации выполняет загрузку объектов и привязанных к ним данных массивов моделей и файлов текстур, которые уже были раннее загружены в проект	**public Scene()** Метод инициализации класса Scene **void CreateObject(string name, string model_name, string texture_name)** Метод создания нового объекта с указанным именем и выбранными данными трёхмерной модели и изображением текстуры **void DeleteObject(string name)** Метод удаления объекта с указанным именем **List<string> DeleteRelatedObjects(string relate)** Метод удаления объекта, который имеет содержит в себе вхождение файла relate. Возвращает лист имён удаленных объектов **bool AddResourceModel(string origFile)** Метод добавления ресурса нового файла модели в проект. Возвращает bool переменную, отображающую успешность выполнения добавления нового или перезаписи существующего файла в проекте

Продолжение таблицы 4.1

1	3	4
		bool AddResourceTexture(string origFile) Метод добавления ресурса нового файла изображения текстуры в проект. Возвращает bool переменную, отображающую успешность выполнения операции добавления нового или перезаписи существующего файла в проекте **void RemoveResource(string resName, string type)** Метод удаления указанного файла ресурса из проекта **string GetFileName(string origFile)** Метод, возвращающий имя файла из указанного полного пути
Model	Model - класс, описывающий трёхмерную модель объектов. Выполняет роль приёма, хранения и передачи массива трёхмерных данных для обработчика триангулярных примитивов. При создании выполняет чтение и преобразование в массивы вершин и индексов, загруженного файла с данными трёхмерного объекта	**public Model(string path)** Метод, инициализирующий новый класс Model, используя файл трёхмерных данных по указанному пути **float[] GetVertices()** Метод, возвращающий массив вершин объекта, в данный момент загруженного и обработанного классом Model **uint[] GetIndices()** Метод, возвращающий массив индексов объекта, в данный момент загруженного и обработанного классом Model

Продолжение таблицы 4.1

1	3	4
Shader	Shader - класс, представляющий программу настройки конечной визуализации, и при инициализации создаёт подпрограмму в общей графике, для преобразования растеризированного изображения внутри буффера памяти видеоадаптера, для отображения текстур и освещения	**Shader(string vertPath, string fragPath)** Метод инициализации нового элемента класса Shader, используя программы шейдеров вершин и фрагментов по указанным путям **void CompileShader(int shader)** Метод компиляции указанного шейдера **void LinkProgram(int program)** Внутренний метод, связывающий объект программы шейдеров с указанной переменной **void Use()** Метод активации раннее связанной программы шейдера **int GetAttribLocation(string attribName)** Метод, возвращающий местонахождения указанного атрибута в шаблоне программы шейдера
Texture	Texture - класс описывающий данные, параметры и атрибуты текстуры изображения. При инициализации выделяет место в буффере памяти видеоадаптера для загруженной в программу текстуры, а также обрабатывает настройки параметров её отображения	**Texture LoadFromFile(string path)** Метод, инициализирующий текстуру из файла **void Use(TextureUnit unit)** Метод, указывающий, какая текстура будет использоваться в проекте

Продолжение таблицы 4.1

1	3	4
Inspector	Класс Inspector - является частью пользовательского интерфейса, представляющего из себя окно взаимодействия между пользователем и программой. Служит для того, чтобы пользователь осуществлял загрузку собственных массивов трёхмерных данных моделей и текстур в проект, а также осуществлял указанные аффинные преобразования над загруженными моделями.	**public Inspector()** Метод инициализации инспектора **void Inspector_Load(object sender, EventArgs e)** Событие, срабатывающее по окончании загрузки инспектора **void timer1_Tick(object sender, EventArgs e)** Событие, срабатывающее каждый тик таймера **void openFileModel_Ok(object sender, CancelEventArgs e)** Событие, срабатывающее при подтверждении выбора файла модели **void openFileTexture_Ok(object sender, CancelEventArgs e)** Событие, срабатывающее при подтверждении выбора файла текстуры **void trackBar_colorRed_Scroll (object sender, EventArgs e)** Метод обработчика скролл-бара красного цвета **void trackBar_colorGreen_Scroll (object sender, EventArgs e)** Метод обработчика скролл-бара зеленого цвета **void trackBar_colorBlue_Scroll (object sender, EventArgs e)** Метод обработчика скролл-бара синего цвета

Продолжение таблицы 4.1

1	3	4
Camera	Класс Camera представляет собой матрицу преобразования, которая служит для корректного отображения перспективы виртуального трёхмерного вида сцены, и для того, чтобы конечный вид проекции сцены был перспективным, а не ортографическим	**Camera(Vector3 position, float aspectRatio)** Метод инициализации камеры **Matrix4 GetViewMatrix()** Метод, возвращающий текущую матрицу взгляда камеры **Matrix4 GetProjectionMatrix()** Метод, возвращающий текущую матрицу проекции камеры **void UpdateVectors()** Метод, обновляющий координаты и векторы камеры

4.2 Системное тестирование разработанной программной системы

В соответствии с реализованными функциями программы, было разработано специализированное системное тестирование, подходящее для тестирования данного программного продукта, которое включает себя набор тестовых случаев, чтобы проверить работоспособность всех осуществляемых функций программы.

4.2.1 Тестовый случай: Инициализация ресурсов программы

Действия пользователя: Пользователь запускает программу.

Ожидаемый результат: Открывается два окна программы. Окно инспектора должно быть поверх всех окон. Программа загружает ресурсы из указанного во внутренних файлах данных пути проекта, и все загруженные ресурсы должны корректно отображаться в проекте.

Ход выполнения:

При запуске программы на экране пользователя отображаются два окна: главное окно вывода и окно инспектора. Окно инспектора находится поверх всех окон, как показано на рисунке 4.1.

Рисунок 4.1 – Интерфейс программы с двумя окнами

Далее необходимо убедиться, что все объекты и данные ресурсов, которые указаны в примере файлов Resources.data и Objects.data в ПРИЛОЖЕНИИ В, были интерпретированы и загружены в программу.

При нажатии на контекстное меню напротив надписи «Объект», появляется выпадающий список, как показано на рисунке 4.2, содержащий все загруженные программой объекты, указанные в файле Objects.data.

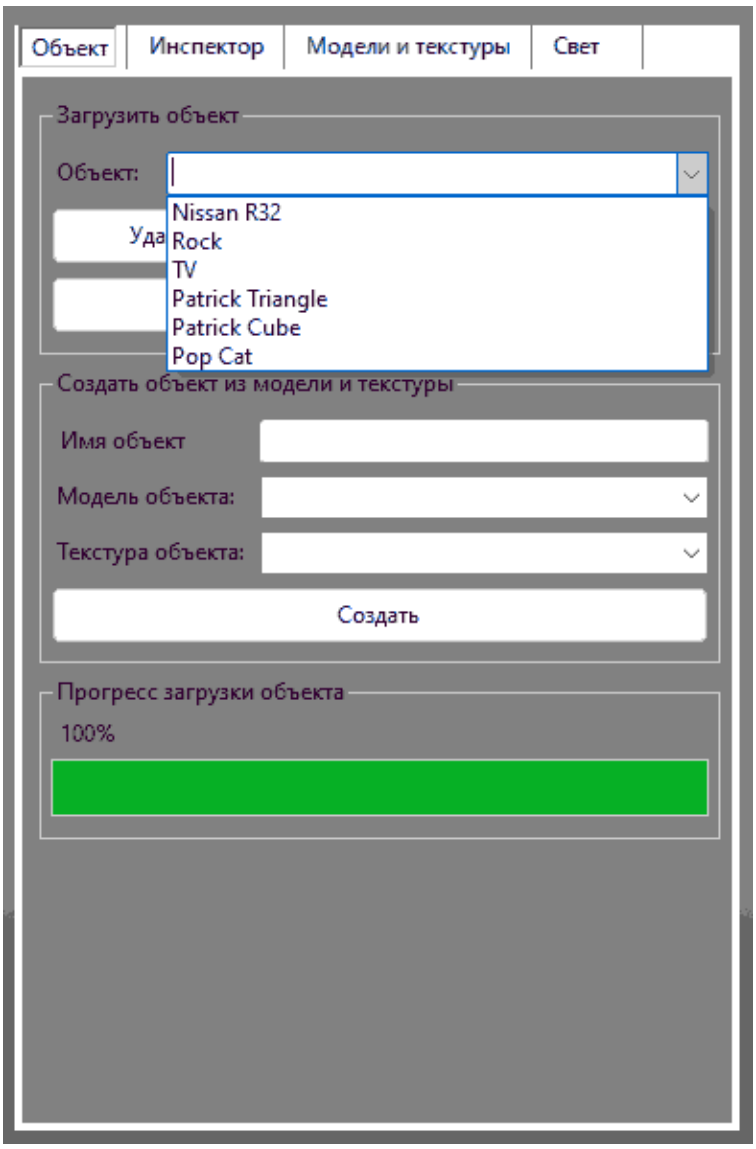


Рисунок 4.2 – Объекты, инициализированные программой

При нажатии на контекстное меню напротив надписей «Модель объекта» и «Текстура объектов», появляются выпадающие списки, содержащие ресурсы, загруженные программой при запуске, как показано на рисунках 4.3 и 4.4.

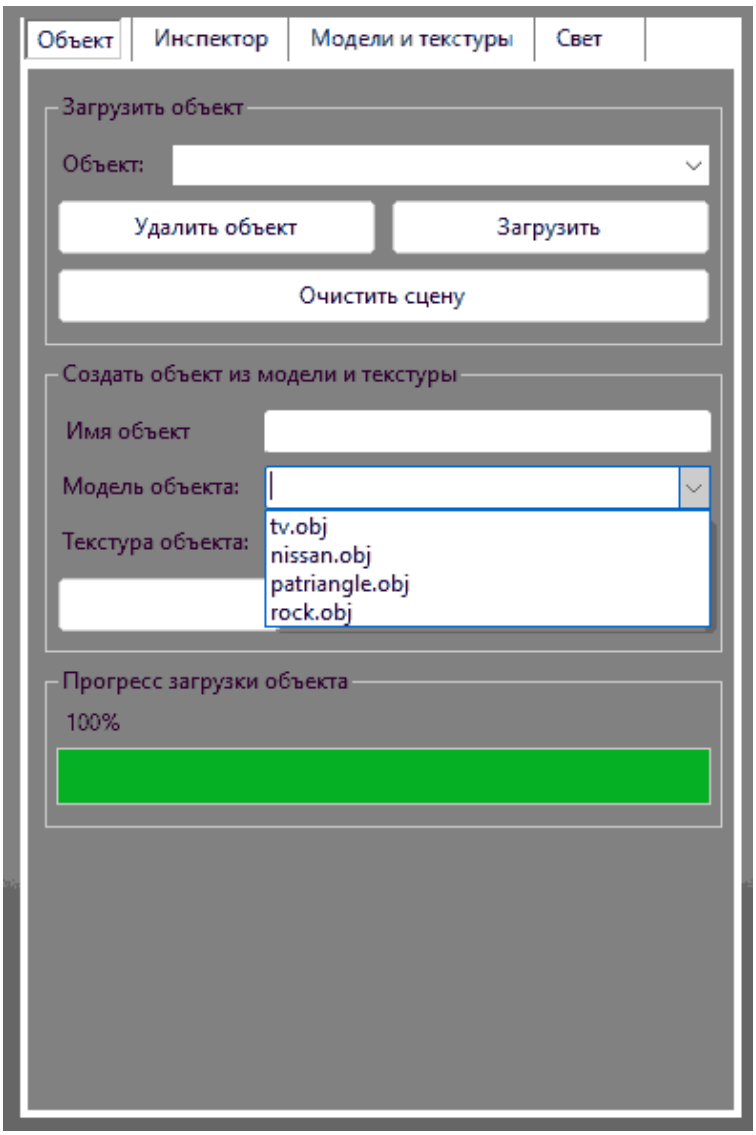


Рисунок 4.3 – Список моделей, загруженных в проект

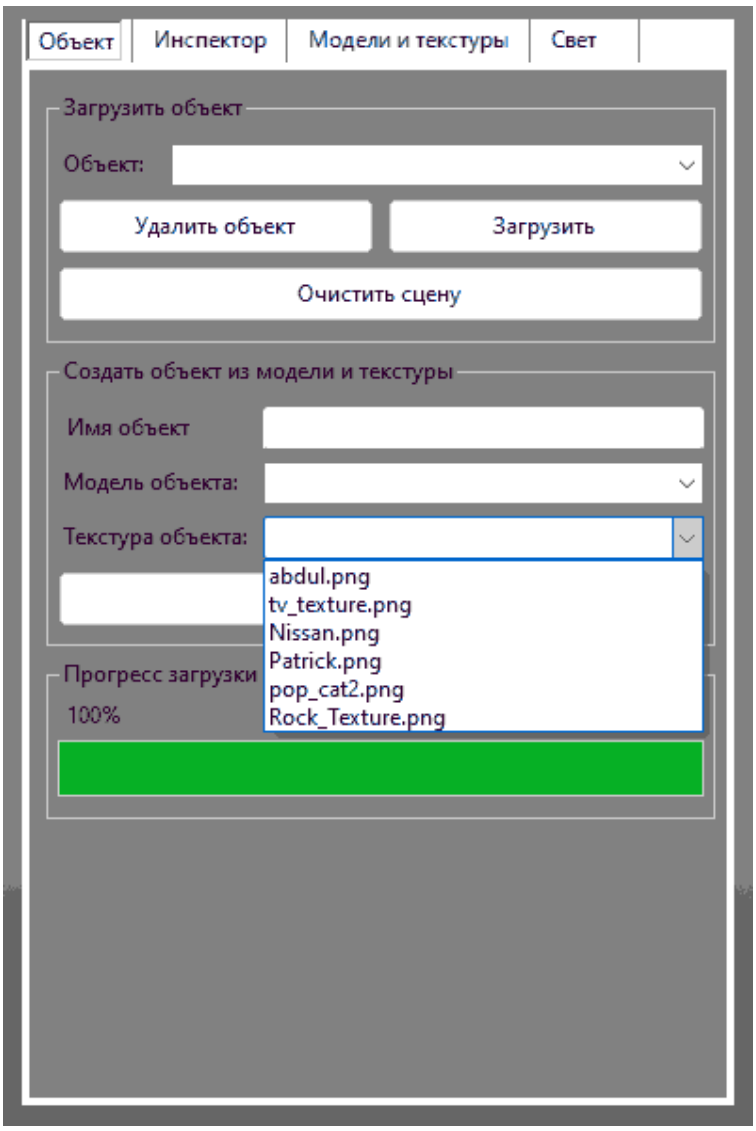


Рисунок 4.4 – Список текстур, загруженных в проект

На данных изображениях видно, что программа успешно загрузила все объекты и ресурсы, которые были указаны в файлах хранения данных.

4.2.2 Тестовый случай: Визуализация объекта на сцене

Действия пользователя: Пользователь пытается загрузить выбранный объект из выпадающего списка объектов на виртуальную сцену.

Ожидаемый результат: Выбранный пользователем объект корректно отображается на виртуальной сцене в главном окне программы.

Ход выполнения:

В открывшемся окне инспектора, пользователь открывает выпадающий список объектов, и выбирает строку с желаемым именем объектом. По-

сле чего, нажимает на кнопку «Загрузить» и запускается процесс отрисовки трёхмерного объекта. Прогресс визуализации можно наблюдать по шкале прогресса в окне инспектора чуть ниже, в разделе «Прогресс загрузки».

Рисунок 4.5 – Главное окно программы с корректно отрисованным объектом на сцене

После окончания визуализации объекта, пользователь увидит его в главном окне программы и сможет управлять перемещением камеры, настройкой света и выполнять трансформации над загруженным объектом.

4.2.3 Тестовый случай: Настройка освещения

Действия пользователя: Пользователь изменяет параметры источника света: цвет освещения - пытается ввести некорректные значения цвета в HEX формате, интенсивность общего света, интенсивность рассеянного света, интенсивность отраженного света и интенсивность отражения цвета.

Ожидаемый результат: Некорректные значения цвета в HEX формате преобразовываются в нейтрально белый цвет по умолчанию, источник света изменяет свои параметры в соответствии с веденными пользователем значениями.

Ход работы:

Для того, чтобы редактировать параметры света, необходимо, чтобы на сцене был загружен какой-либо объект, в противном случае изменение параметров света не будет иметь значения, так как не будет поверхностей, от которых свет мог бы отражаться.

В запущенной программе, пользователь переходит на вкладку инспектора «Свет». В открытой вкладке пользователь пытается ввести некорректные значения света в HEX-формате, как показано на рисунке 4.6.

Рисунок 4.6 – Пример некорректно введённого значения цвета в формате HEX

После попытки применить данный некорректно введенный цвет к освещению, нажав на кнопку «Выбрать цвет», программа преобразует его в нейтрально белый цвет по умолчанию, как показано на рисунке 4.7.

Рисунок 4.7 – Результат преобразования некорректного значения цвета HEX в нейтрально белый цвет

Далее пользователь вводит остальные значения света, как в примере на рисунке 4.8.

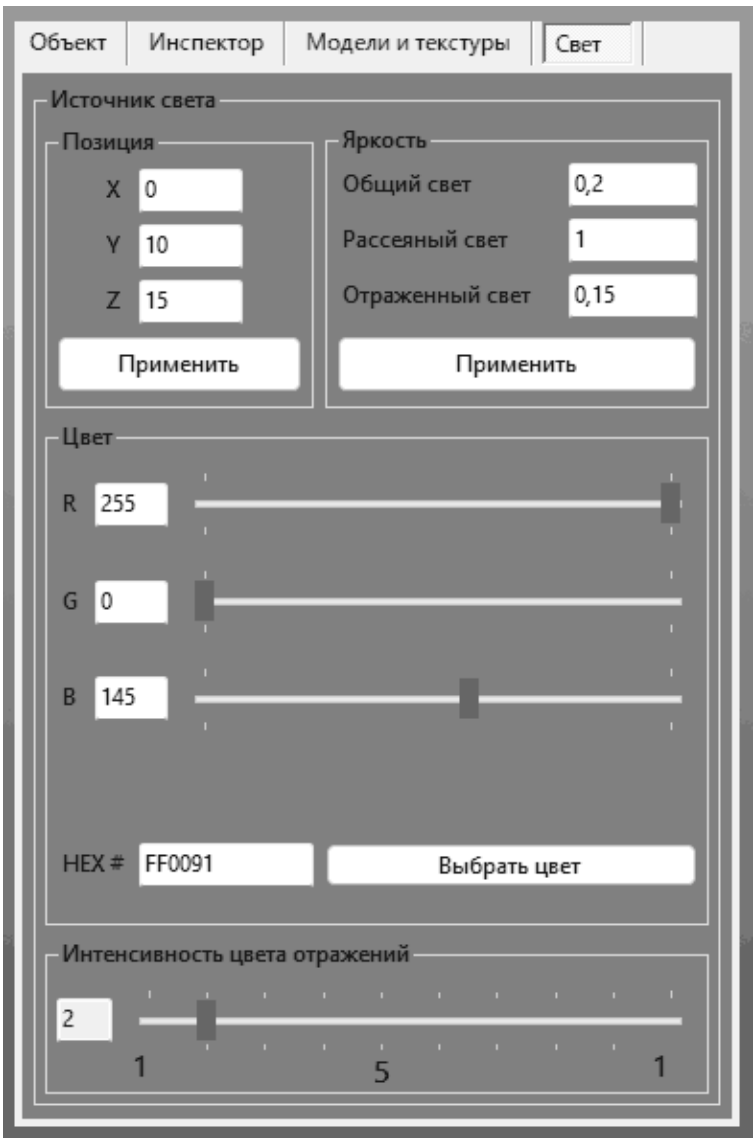


Рисунок 4.8 – Пример введёных значений параметров света

По мере введения данных значений, пользователь уже начнёт замечать изменения освещения. Ползунки, отвечающие за интенсивность одного из трёх цветов, применяют изменения к освещению сразу же, как только пользователь начал взаимодействовать с данным элементом интерфейса. Также моментально применяет свои значения и ползунок интенсивности отражения цвета.

Для применения изменения позиции и интенсивности света, необходимо нажать на кнопки «Применить» в соответствующих разделах поля.

Результат конечных изменений параметров света показан на рисунке 4.9.

Рисунок 4.9 – Результат изменений параметров света на виртуальной сцене

4.2.4 Тестовый случай: Трансформация объекта

Действия пользователя: Пользователь загружает объект и пытается применить к нему трансформацию, вводя различные значения, включая отрицательные и значения, равные нулю.

Ожидаемый результат: Модель объекта, загруженного на сцену будет трансформирована, соответственно введенным пользователем значениям.

Ход выполнения:

Пользователь загружает объект на сцену, затем перейдя во вкладку «Инспектор», вводит значения трансформации модели объекта в соответствующие строки, как показано в примере на рисунке 4.10.

Рисунок 4.10 – Пример введенных значений трансформации модели

Чтобы применить данные трансформации, пользователь нажимает на кнопку «Применить». И результат становится виден на экране, как показано на рисунке 4.11.

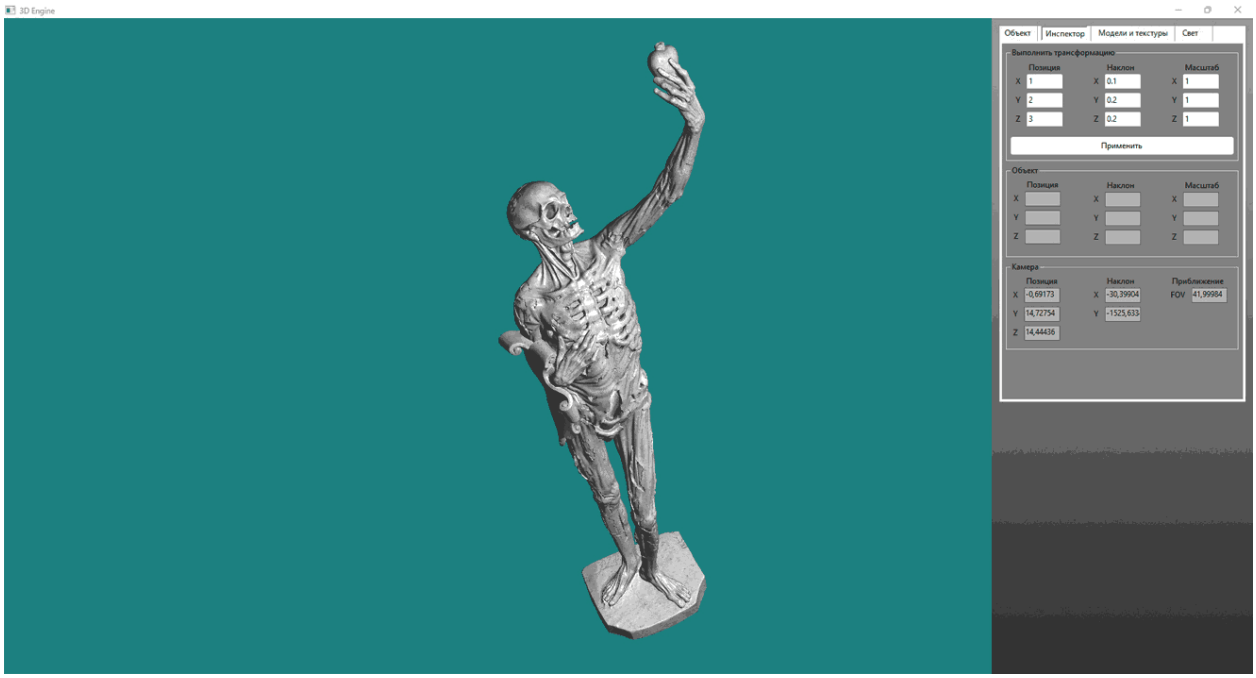


Рисунок 4.11 – Результат трансформации трёхмерной модели объекта

Как видно на снимке экрана, модель наклонилась набок, по осям в определенных значениях, указанным пользователем.

Все значения трансформаций, кроме значений масштаба модели могут быть равны нулю и отрицательными. Но в данном случае пользователь пытается задать нулевые значения для трансформации масштаба модели, как показано на рисунке 4.12

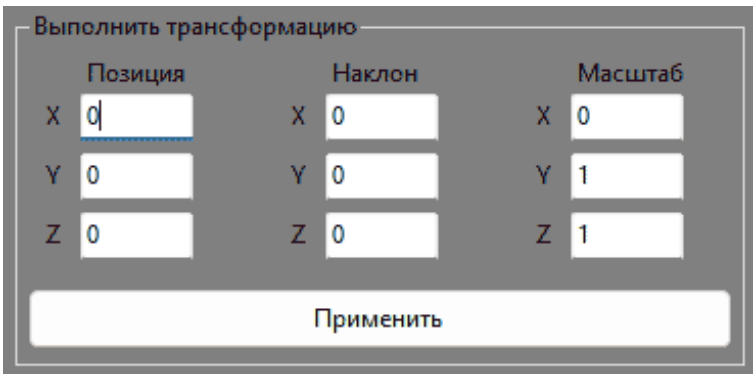


Рисунок 4.12 – Трансформация масштаба, равная нулю в одной из координат

Трансформация масштаба - это коэффициент матрицы, проще говоря число, на которое будет умножена матрица трёхмерной модели. Соответственно, при умножении матрицы на ноль, модель объекта по своей сути умножается на ноль и перестаёт существовать, так что пользователь будет наблюдать пустую виртуальную сцену.

4.2.5 Тестовый случай: Импортирование ресурсов в проект

Действия пользователя: Пользователь пытается загрузить собственные файлы моделей или текстур в проект со своего устройства, для дальнейшей работы с ними.

Ожидаемый результат: Файлы будут успешно загружены в проект, а пользователь сможет с ними взаимодействовать внутри программы.

Ход выполнения:

В открытой программе пользователь, переключившись на окно Инспектора, переходит во вкладку «Модели и текстуры», как показано на рисунке 4.13.

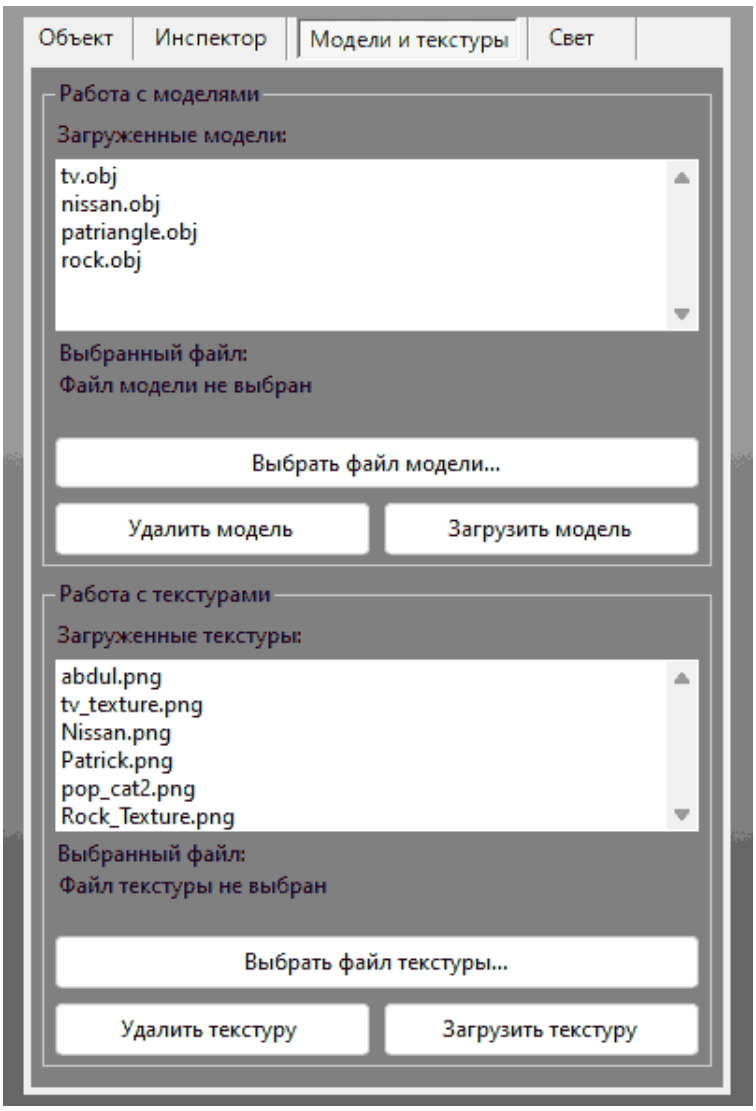


Рисунок 4.13 – Открытая вкладка инспектора «Модели и текстуры»

На активном окне вкладки «Модели и текстуры» пользователь может видеть списки моделей и текстур, уже загруженные в проект. В зависимости от типа загружаемого файла в проект, пользователь нажимает на соответствующую кнопку. В данном случае пользователь желает загрузить файл трёхмерной модели объекта, и нажимает на кнопку «Загрузить модель». На экране появляется новое диалоговое окно с функцией выбора файла из системы устройства пользователя. Данное окно показано на рисунке 4.14.

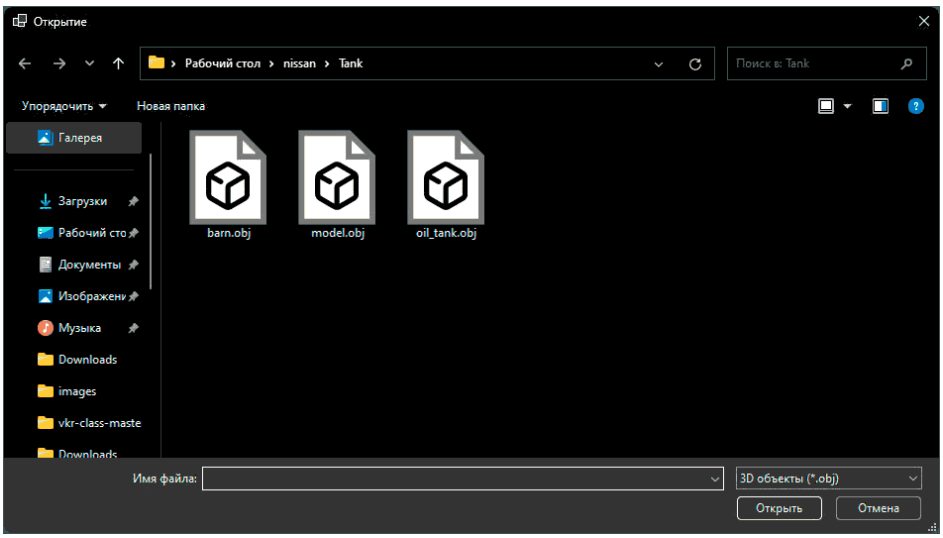


Рисунок 4.14 – Диалоговое окно выбора файла

В данном окне пользователь выбирает желаемый файл трёхмерной модели и нажимает на кнопку «Открыть». Если данный файл уже был загружен в проект, то на экране появится диалоговое окно, информирующее пользователя о перезаписи существующего файла. После окончания загрузки файла в проект, то появится диалоговое окно, информирующее пользователя об успешном импорте файла в проект (рисунок 4.15).

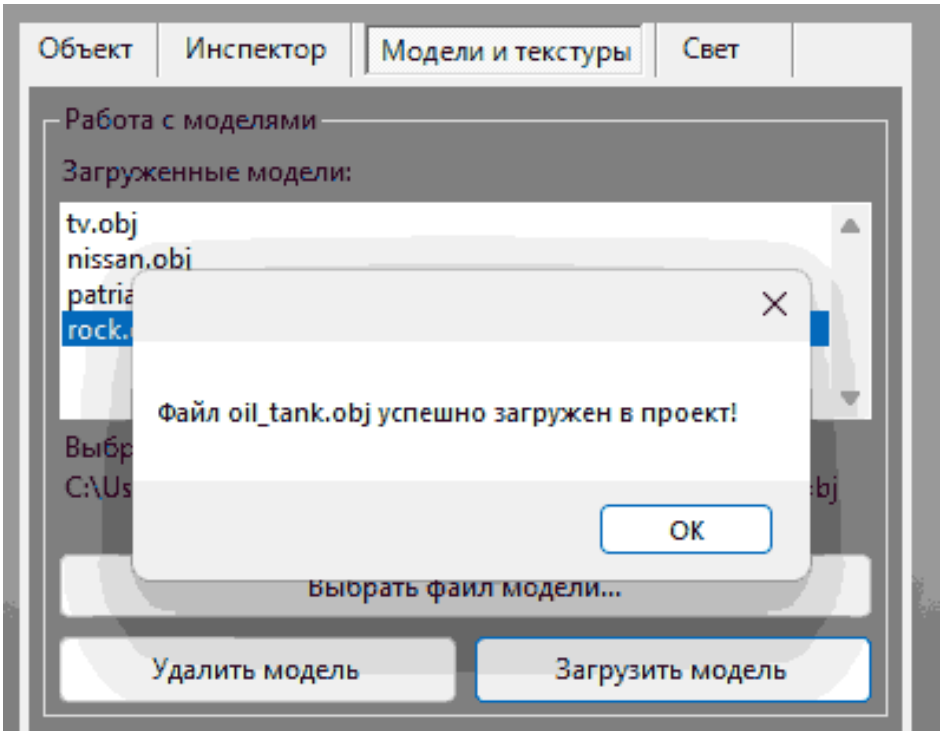


Рисунок 4.15 – Диалоговое окно успешного импорта файла

После загрузки файла, необходимо проверить корректно ли прошёл импорт. Пользователь переходит во вкладку «Объект» и нажимает на строку выпадающего списка того же типа, которого и был раннее загружен файл в проект. Так как пользователь загружал модель, то и нажмёт он на список, содержащий трёхмерные модели в проекте. Результат показан на рисунке 4.16.

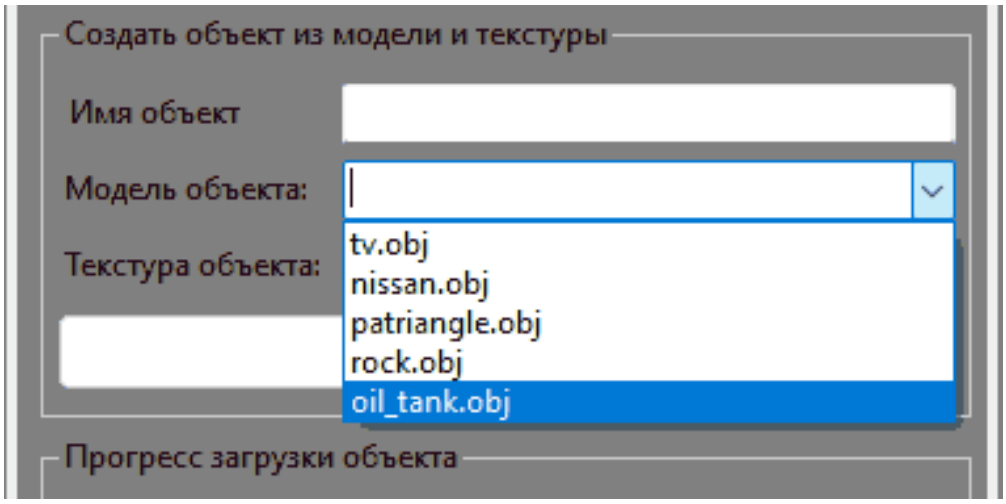


Рисунок 4.16 – Список загруженных моделей в проект

Как видно на снимке экрана, в конец выпадающего списка моделей добавилась новая позиция с наименованием файла модели oil_tank.obj, который и загрузил пользователь немного раннее.

4.2.6 Тестовый случай: Удаление ресурсов из проекта

Действия пользователя: Пользователь выполняет действия для удаления соответствующего ресурса из проекта.

Ожидаемый результат: Файлы ресурсов, их ассоциации и объекты, связанные с данными ресурсами будут удалены из проекта, без возможности дальнейшего взаимодействия с ними.

Ход выполнения:

В открытой программе пользователь, переключившись на окно Инспектора, переходит во вкладку «Модели и текстуры».

На активном окне вкладки «Модели и текстуры» пользователю предоставляются списки моделей и текстур, находящиеся в проекте. В данных

списках пользователь выбирает желаемый ресурс для удаления, нажав на его имя. После того, как желаемый ресурс будет выделен, пользователь нажимает на кнопку «Удалить модель». На экране высвечивается сообщение об успешном удалении модели из проекта.

Но в случаях, когда данная модель являлась составляющей какого-либо существующего объекта в программе, то пользователю будет высвечено сообщение со списком объектов, которые были уничтожены, в связи с удалением их трёхмерной модели из проекта, как показано на рисунке 4.17.

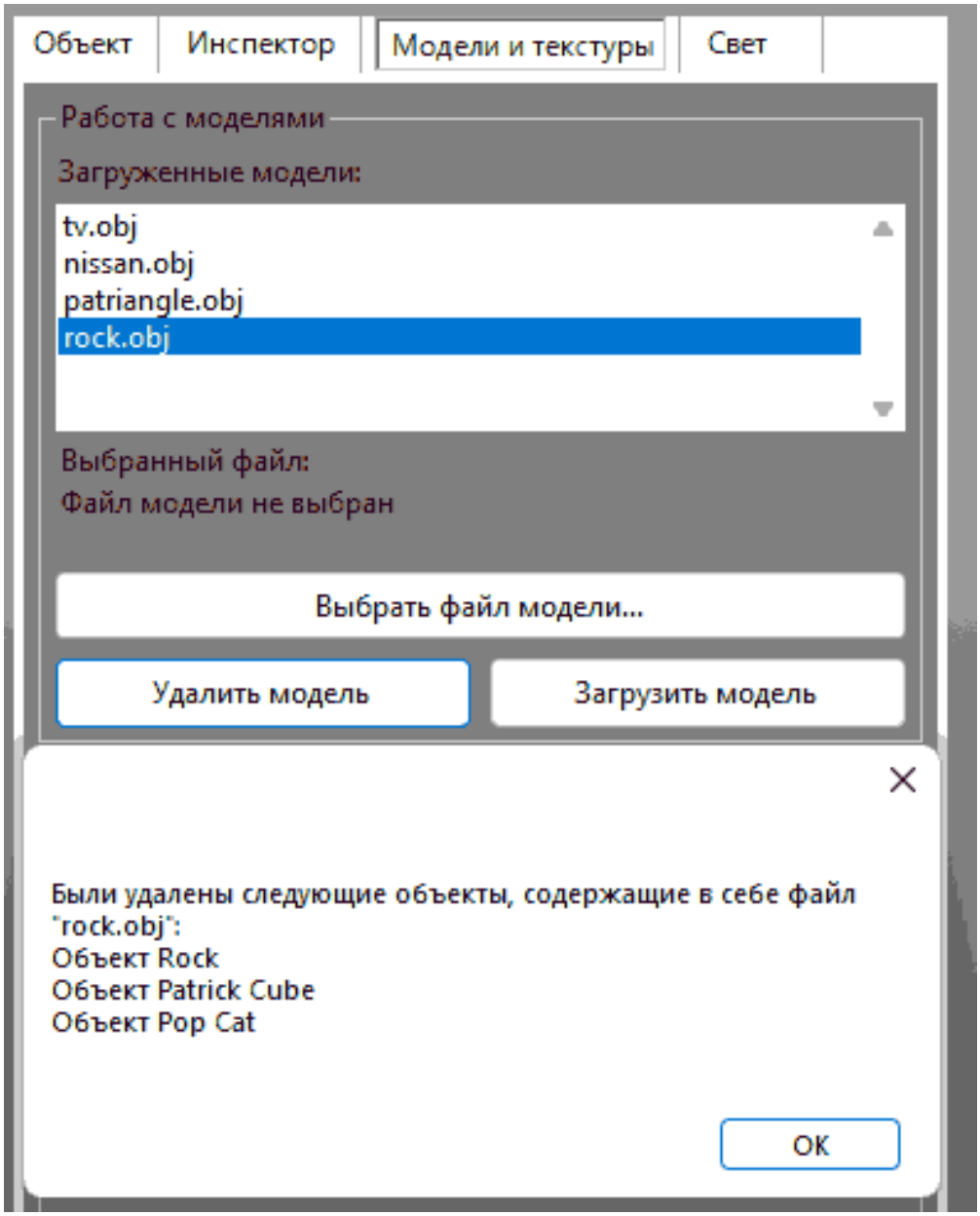


Рисунок 4.17 – Вывод сообщения со списком удаленных объектов в проекте

Перейдя во вкладку «Объект», и открыв выпадающий список моделей, можно увидеть на рисунке 4.18, что модель удаленного файла, также была удалена из списка прогружаемых моделей в проекте.

Далее, открыв выпадающий список объектов, можно увидеть на рисунке 4.19, что объекты, содержащие удаленную модель, также были удалены из проекта.

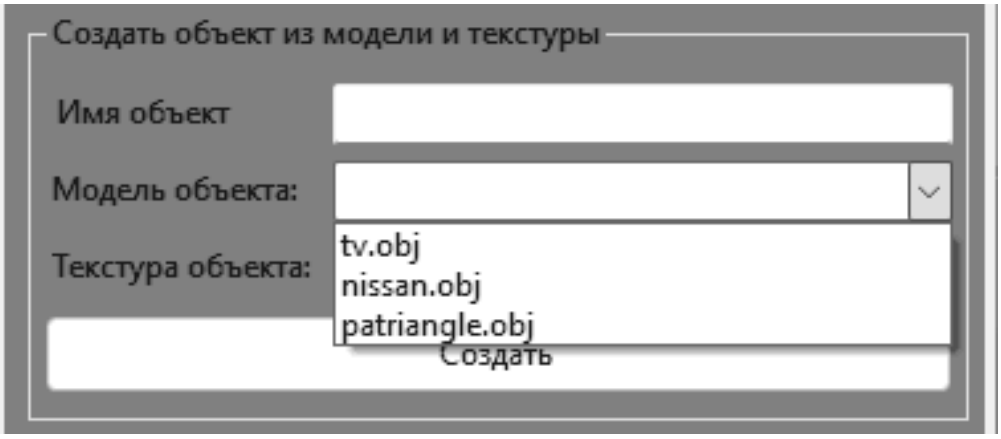


Рисунок 4.18 – Список, демонстрирующий оставшиеся модели проекта, после удаления файла rock.obj

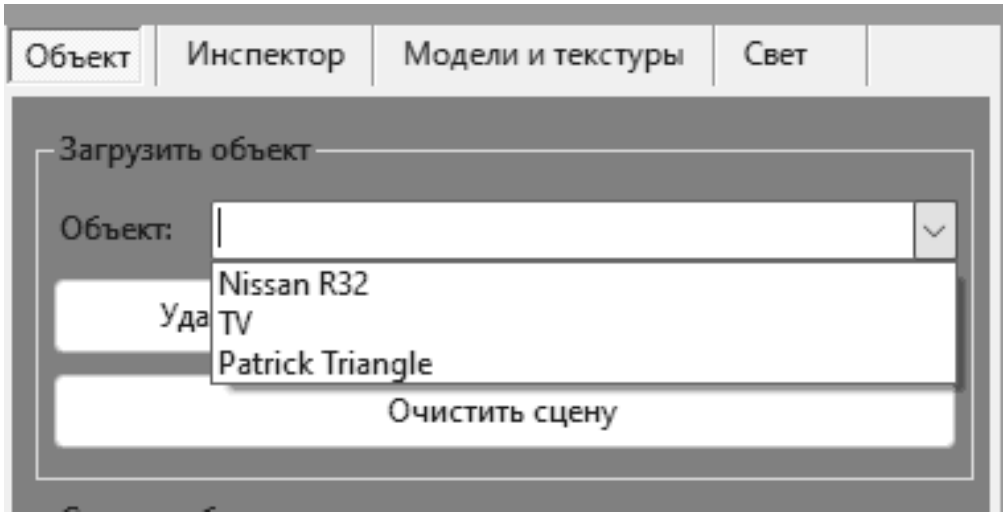


Рисунок 4.19 – Список, демонстрирующий оставшиеся объекты в проекте, после удаления файла rock.obj

4.2.7 Тестовый случай: Создание нового объекта

Действия пользователя: Пользователь создаёт новый объект, введя его имя и выбрав модель с текстурой.

Ожидаемый результат: В проекте создаётся новый объект, с указанным пользователем названием.

Ход работы:

В открывшемся окне инспектора, пользователь вводит желаемое имя для объекта напротив надписи «Имя объекта», а также выбирает модель и текстуру для данного объекта из выпадающих списков чуть ниже. Корректное и полное заполнение полей для создания объекта показано на рисунке 4.20.

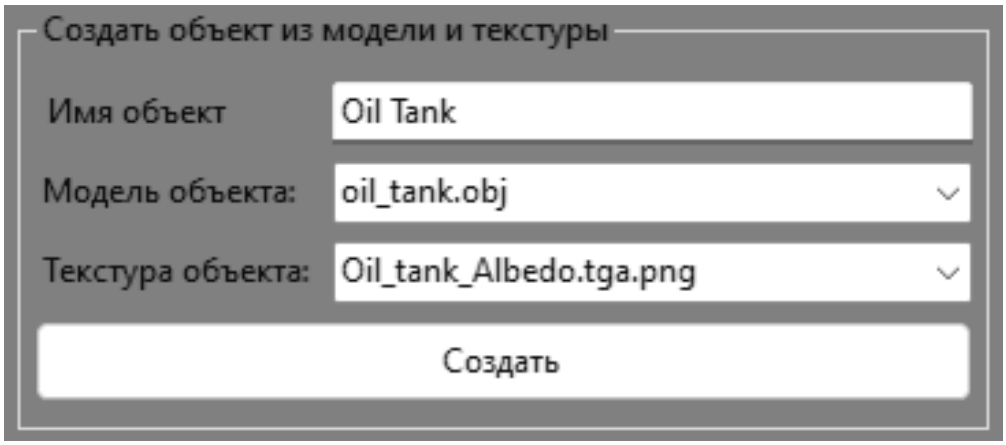


Рисунок 4.20 – Пример корректного заполнения полей для создания нового объекта

Нажав на кнопку «Создать», система уведомит об успешном создании объекта с указанным именем.

Теперь пользователь может загрузить на сцену только что созданный объект. На той же вкладке, открыв выпадающий список объектов, в его конце появится имя нового объекта (рисунок 4.21).

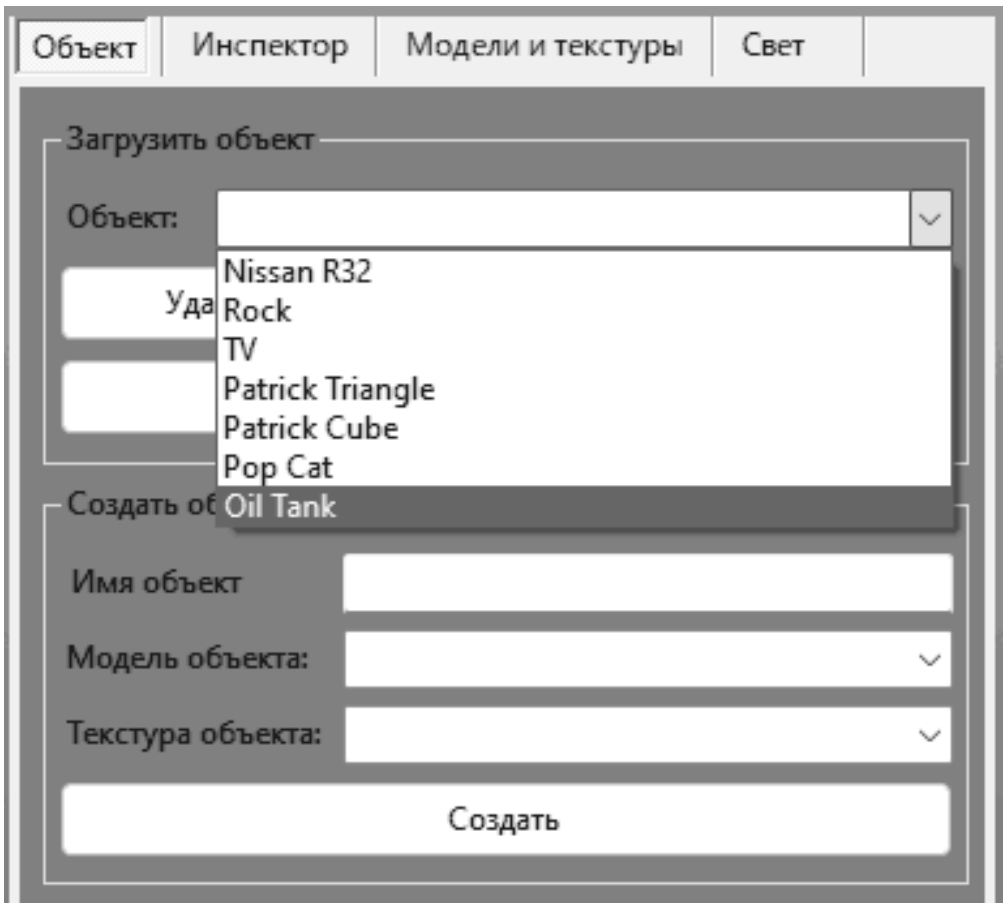


Рисунок 4.21 – Обновленный список объектов, в который был добавлен объект Oil Tank

После того, как в проект был добавлен объект с указанным именем и связанными с ним моделью и текстурой, можно проверить корректность его отображения на виртуальной сцене. Выбрав имя объекта в выпадающем списке, и нажав на кнопку «Загрузить», в главном окне программы будет отрисован текущий объект. Результат визуализации нового объекта показан на рисунке 4.22.

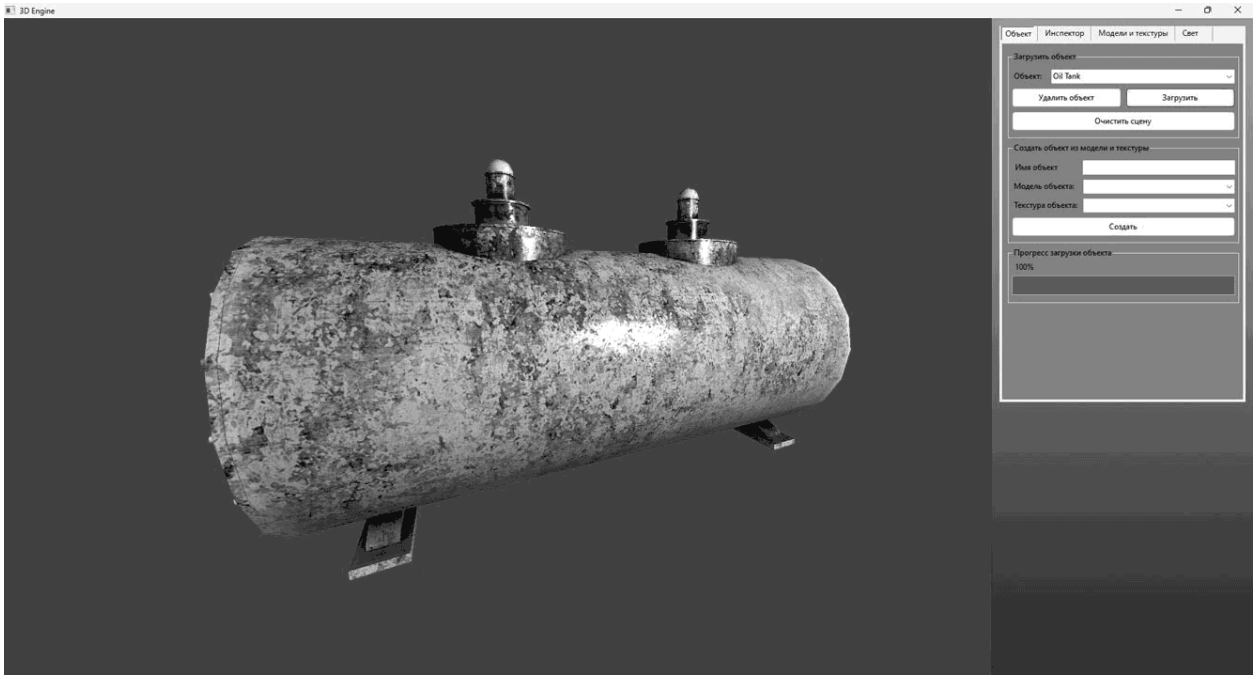


Рисунок 4.22 – Результат визуализации объекта на сцене, созданного пользователем

На данном снимке экрана можно увидеть, что модель и текстура объекта на виртуальной сцене отображена корректно.

ЗАКЛЮЧЕНИЕ

Задача трехмерной графики — презентовать объект, явление или пространство в стилизованной или реалистичной визуальной форме. Дизайнеры рисуют предметы почти с нуля, настраивают освещение, работают над композицией кадра и в целом выполняют большую работу, поэтому с каждым годом трёхмерная компьютерная графика становится реалистичнее.

Постоянное совершенствование компьютерного оборудования и программного обеспечения сделало 3D-технологии доступными. Сегодня 3D-модели повсеместно используют вместо обычных макетов в проектировании для проработки крупных или миниатюрных деталей, а «объемная» визуализация становится одним из инструментов маркетинговых мероприятий, интерактивных тренингов, презентаций.

Основные результаты работы:

1. Проведен анализ предметной области. Выявлена необходимость использовать библиотеку OpenGL.

2. Разработана концептуальная модель программы. Разработана модель данных системы. Определены требования к системе.

3. Осуществлено проектирование программной системы. Разработана основа графического движка. Разработан пользовательский интерфейс приложения.

4. Реализована и протестирована программная система. Проведено системное тестирование.

Все требования, объявленные в техническом задании, были полностью реализованы, все задачи, поставленные в начале разработки проекта, были также решены.

Готовый рабочий проект представлен в виде десктопного приложения. Приложение записано на внешний носитель информации, который приложен к отчёту.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дэвид Вольф: OpenGL 4. Язык шейдеров. Книга рецептов / Д. Вольф – Москва : ДМК-Пресс, 2015. – 368 с. – ISBN 978-5-97060-255-3. – Текст : непосредственный.

2. C# 8.0 и .NET Core 3.0 – Модернизация кросс-платформенной разработки / М. Дж. Прайс – Packt Publishing, 2019. – 818 с. – ISBN 978-1-78847-812-0. - Текст : непосредственный.

3. Компьютерная графика и 3D-моделирование: учебное пособие для СПО/ Л. Ю. Забелин, О. Л. Штейнбах, О. В. Диль / Профобразование, 2023. ISBN 978-5-4488-1594-2 - Текст : непосредственный.

4. Обработка и визуализация пространственных данных на гибридном вычислительном кластере / В. А. Бобков, А. С. Черкашин - Синергия, 2014. - ISBN 978-5-04-015250-6. - Текст : непосредственный.

5. C#. Программирование 2D и 3D векторной графики /Н. А. Тюкачев, В. Г. Хлебостроев – Санкт-Петербург : Лань, 2022. – 320 с. – ISBN 978-5-8114-8988-6. – Текст : непосредственный.

6. Объектно-ориентированное программирование с примерами на C#. Учебное пособие. Студентам ВУЗов. / П.Б. Хорев – Москва : Форум, 2023. – 200 с. – ISBN 978-5-00091-680-3. – Текст : непосредственный.

7. 3D Graphics Rendering Cookbook / С. Косаревский, В. Латыпов – Packt Publishing, 2021. – ISBN 978-1-83898-619-3. - Текст : непосредственный.

8. OpenGL ES 3.0. Руководство разработчика / Д. Гинсбург – БХВ-Петербург, 2014. – ISBN 978-5-97060-256-0. - Текст : непосредственный.

9. Введение в трехмерную компьютерную графику с использованием библиотеки OpenGL / Ю. Кошкина, М. Персова - Новосибирский государственный технический университет, 2019. - ISBN 978-5-7782-3744-5. - Текст : непосредственный.

10. Программирование на C# для начинающих. Особенности языка / А.Н. Васильев – Москва : Бомбора, 2022. – 528 с. – ISBN 978-5-04-092520-9. – Текст : непосредственный.

11. Модели освещения и алгоритмы затенения в компьютерной графике / А. Г. Задорожный - Новосибирский государственный технический университет, 2023. - ISBN 978-5-7782-4308-8. - Текст : непосредственный.

12. OpenGL ES 3.0. Руководство разработчика / Д. Гинсбург, Б. Пурномо - ДМК Пресс, 2014 - ISBN 978-5-97060-256-0. - Текст : непосредственный.

13. C# 5.0. Карманный справочник / Д. Албахари, Б. Албахари - Диалектика-Вильямс, 2015 - ISBN 978-5-8459-1820-8. - Текст : непосредственный.

ПРИЛОЖЕНИЕ А

Представление графического материала

Графический материал, выполненный на отдельных листах, изображен на рисунках А.1–А.9.

Сведения о ВКРБ

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ПО ПРОГРАММЕ БАКАЛАВРИАТА

«Программная система визуализации трехмерных данных,
использующая библиотеку OpenGL»

Руководитель ВКРБ
д.т.н, профессор
Серебровский Вадим Владимирович

Автор ВКРБ
студент группы ПО-02б
Снатенков Николай Иванович

				ВКРБ 20060086.09.03.04.24.009			
	Фамилия И. О.	Подпись	Дата	Сведения о ВКРБ	Лит.	Масса	Масштаб
Автор работы	Снатенков Н.И.						
Руководитель	Серебровский В.В.				Лист 1	Листов 9	
Нормоконтроль	Чаплыгин А.А.			Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-02б		

Рисунок А.1 – Сведения о ВКРБ

Цель и задачи разработки

Цель настоящей работы - проектирование и разработка графического движка - программной системы визуализации трёхмерных данных, использующей библиотеку OpenGL.

Для достижения поставленной цели необходимо решить следующие задачи:

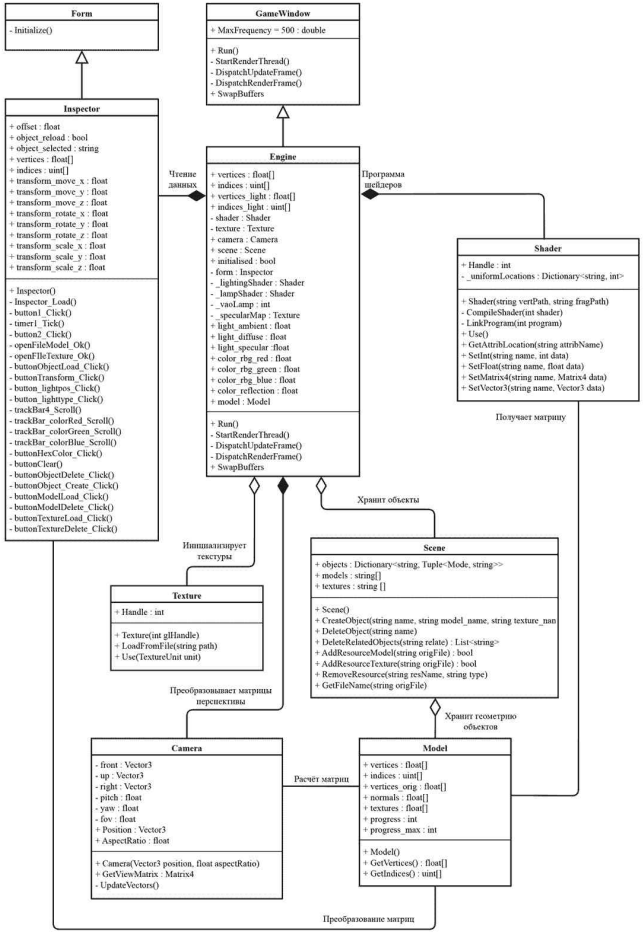
1. Изучить, провести анализ основ и принципов работы спецификации и библиотеки OpenGL.
2. Разработать концептуальную модель графического движка на основе библиотеки OpenGL.
3. Спроектировать программную систему визуализации трёхмерных данных.
4. Сконструировать и протестировать программную систему визуализации трёхмерных данных.

Рисунок А.2 – Цель и задачи разработки

Рисунок А.3 – Диаграмма конвеера данных

Рисунок А.4 – Диаграмма вариантов использования

Диаграмма классов программы



				ВКРБ 20060086.09.03.04.24.009		
	Фомкина Н. О.	Подпись	Дата	Диаграмма классов программы	Лит.	Масса	Масштаб
Автор работы	Сенченко Н.И.						
Руководитель	Серебровский В.В.				Лист 5	Листов 9
Нормоконтроль	Чистягин А.А.			Выпускная квалификационная работа бакалавра	ЮЗГУ П0-026	

Рисунок А.5 – Диаграмма классов программы

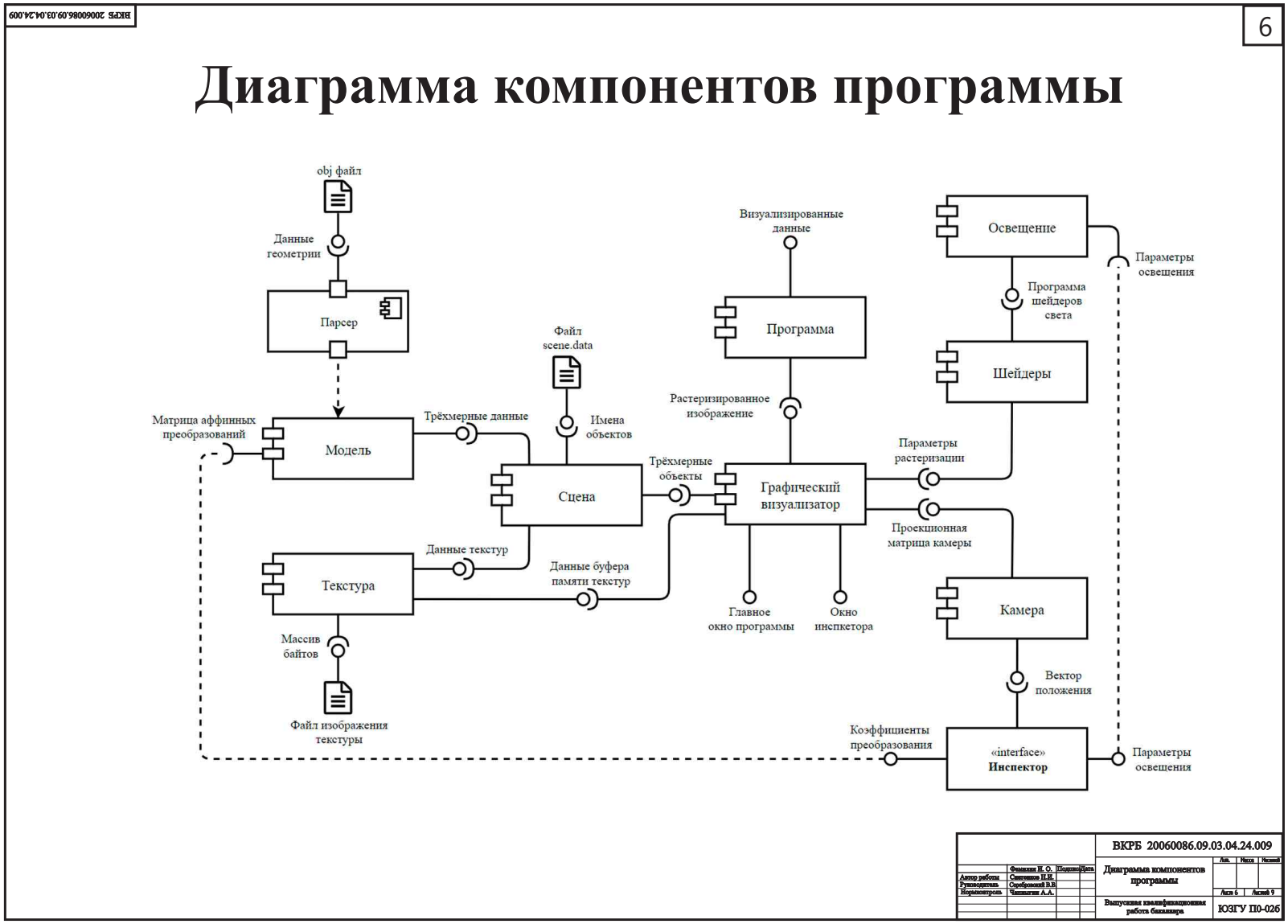
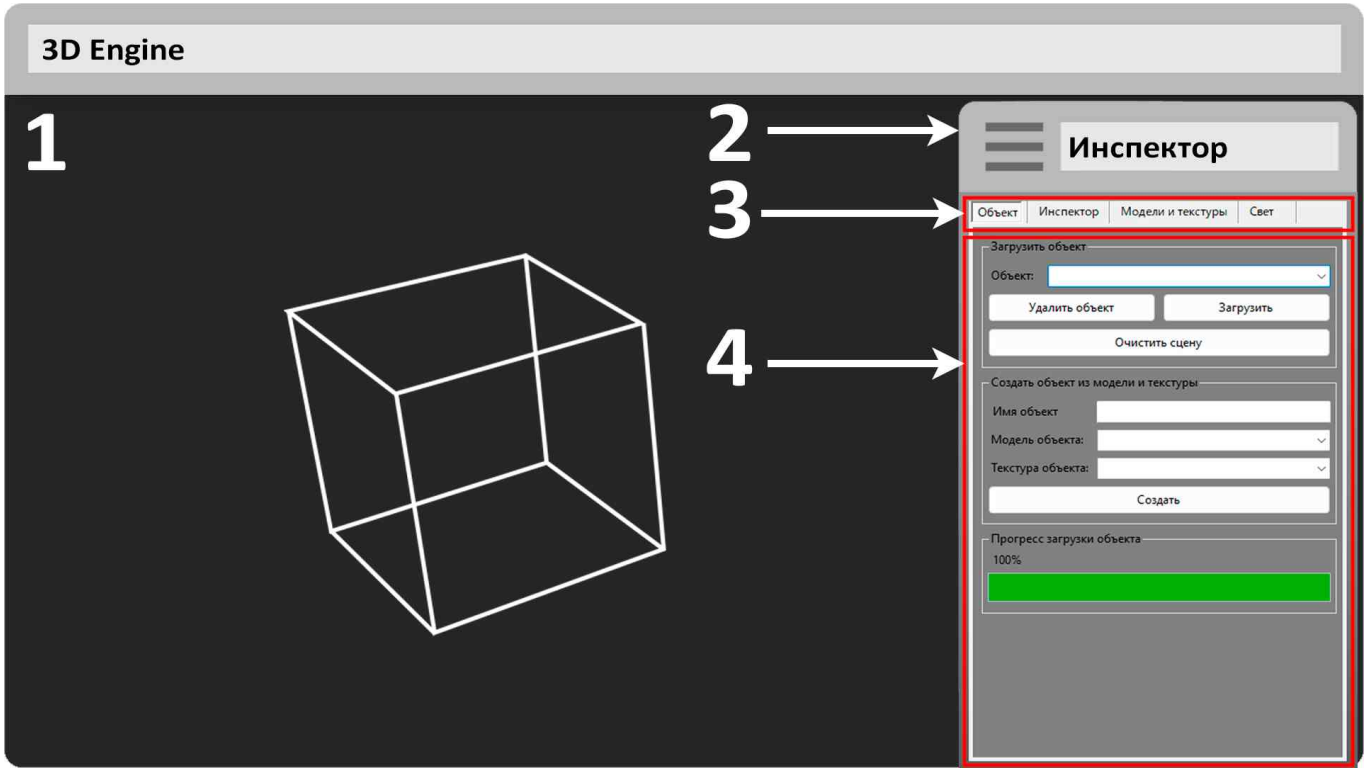


Рисунок А.6 – Прототип компонентов программы

Прототип пользовательского интерфейса



				ВКРБ 20060086.09.03.04.24.009		
	Фомичев Н. О.	Подпись	Дата	Прототип пользовательского интерфейса	Лит.	Масса	Масштаб
Автор работы	Сивушкин Н.И.						
Руководитель	Серебровский В.В.				Лист 7	Листов 9	
Нормоконтроль	Чапыгин А.А.			Выпускная квалификационная работа бакалавра	ЮЗГУ П0-026		

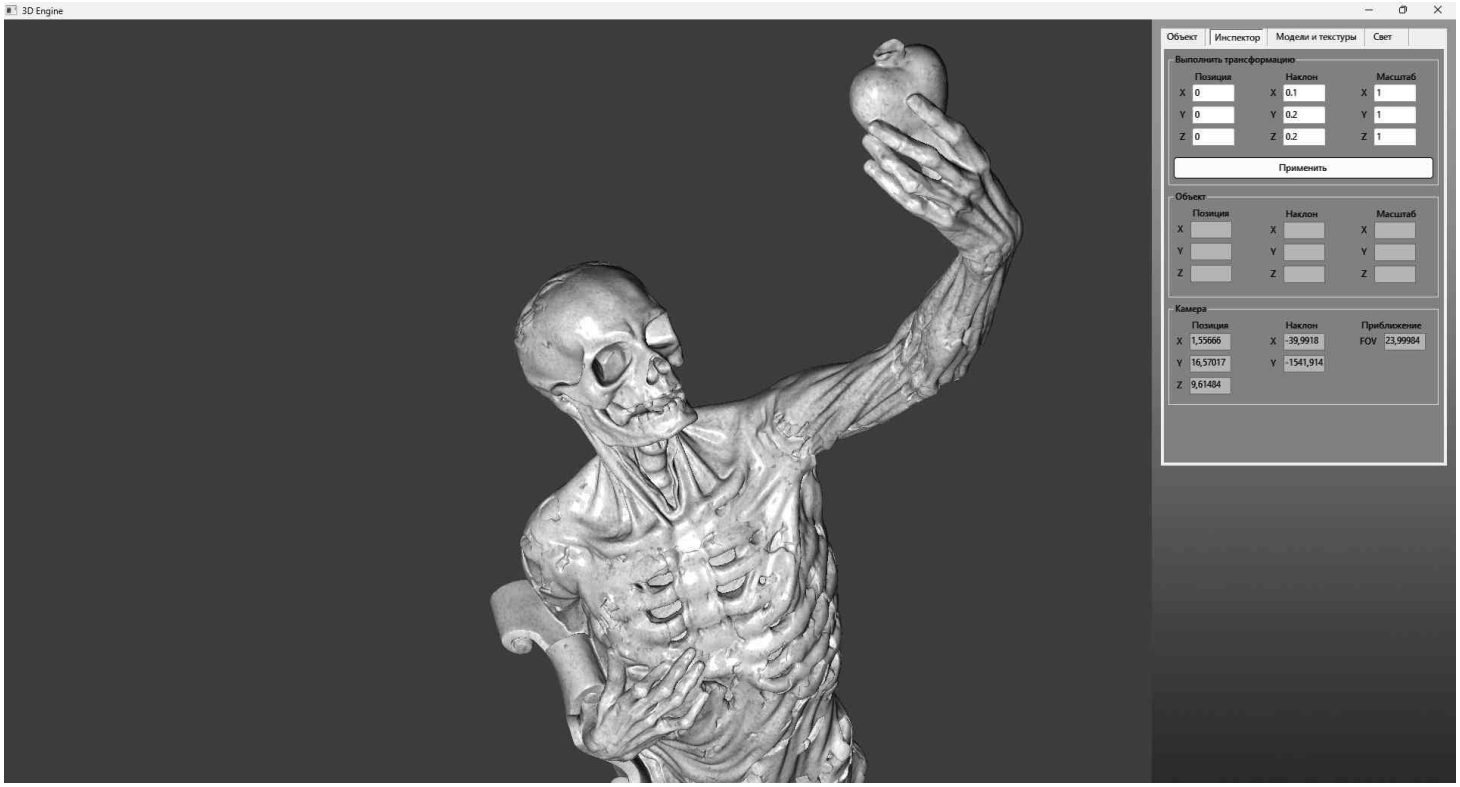
Рисунок А.7 – Прототип интерфейса программы

Интерфейс программы

				ВКРБ 20060086.09.03.04.24.009	
	Фомкина Н. О.	Подпись	Дата	Интерфейс программы	Лит. / Масса / Масштаб
Автор работы	Сивушкин Н.И.				
Руководитель	Серебровский В.В.				Лист 8 / Листов 9
Нормоконтроль	Чистягин А.А.			Выпускная квалификационная работа бакалавра	ЮЗГУ П0-026

Рисунок А.8 – Интерфейс программы

Результат визуализации трёхмерных данных



				ВКРБ 20060086.09.03.04.24.009			
	Фомкина Н. О.	Подпись	Дата	Результат визуализации трёхмерных данных	Лит.	Масса	Масштаб
Автор работы	Сидоренко Н.И.						
Руководитель	Серебровский В.В.				Лист 9	Листов 9	
Нормоконтроль	Чистягин А.А.						
				Выпускная квалификационная работа бакалавра	ЮЗГУ П0-026		

Рисунок А.9 – Результат визуализации трёхмерных данных

ПРИЛОЖЕНИЕ Б

Пример содержания OBJ-файла

cube.obj

```
# Blender v3.3.1 OBJ File: 'Rock.blend'
# www.blender.org
mtllib Rock.mtl
o Cube
v 1.786532 2.008477 -1.563261
v 1.786532 0.008477 -1.563261
v 1.786532 2.008477 0.436739
v 1.786532 0.008477 0.436739
v -0.213468 2.008477 -1.563261
v -0.213468 0.008477 -1.563261
v -0.213468 2.008477 0.436739
v -0.213468 0.008477 0.436739
vt 0.666000 0.000000
vt 0.999715 1.000000
vt 0.666000 1.000000
vt 0.666312 1.000000
vt 0.333000 0.000000
vt 0.666312 0.000000
vt 0.999964 1.000000
vt 0.666000 0.000000
vt 1.000000 0.000000
vt 0.999843 0.000000
vt 0.900000 0.300000
vt 0.900000 0.000000
vt 0.333300 1.000000
vt 0.000000 0.000000
vt 0.333319 0.000000
vt 0.333000 1.000000
vt 0.666000 0.000000
vt 0.333000 0.000000
vt 0.999715 0.000000
vt 0.333000 1.000000
vt 0.666000 1.000000
vt 0.999843 0.300000
vt 0.000000 1.000000
vn 0.0000 1.0000 0.0000
vn 0.0000 0.0000 1.0000
vn -1.0000 0.0000 0.0000
vn 0.0000 -1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
usemtl Material
s off
f 5/1/1 3/2/1 1/3/1
f 3/4/2 8/5/2 4/6/2
f 7/7/3 6/8/3 8/9/3
f 2/10/4 8/11/4 6/12/4
f 1/13/5 4/14/5 2/15/5
f 5/16/6 2/17/6 6/18/6
```

```
f  5/1/1  7/19/1  3/2/1
f  3/4/2  7/20/2  8/5/2
f  7/7/3  5/21/3  6/8/3
f  2/10/4  4/22/4  8/11/4
f  1/13/5  3/23/5  4/14/5
f  5/16/6  1/3/6  2/17/6
```

ПРИЛОЖЕНИЕ В

Пример содержания файлов, описывающие хранимые ресурсы проекта

Resources.data

```
\TEXTURE\abdul.png
\TEXTURE\tv_texture.png
\TEXTURE\Nissan.png
\TEXTURE\Patrick.png
\TEXTURE\pop_cat2.png
\TEXTURE\Rock_Texture.png

\MODEL\tv.obj
\MODEL\nissan.obj
\MODEL\patriangle.obj
\MODEL\rock.obj
```

Objects.data

```
Nissan R32\nissan.obj\Nissan.png
Rock\rock.obj\Rock_Texture.png
TV\tv.obj\tv_texture.png
Patrick Triangle\patriangle.obj\Patrick.png
Patrick Cube\rock.obj\Patrick.png
Pop Cat\rock.obj\pop_cat2.png
```

ПРИЛОЖЕНИЕ Г

Листинг фрагмента программы парсера

```
1 {
2     string[] file = File.ReadAllLines(path);
3     int vertices_size = 0;
4     int normals_size = 0;
5     int textures_size = 0;
6     int indices_size = 0;
7
8     progress_max = file.Length * 3;
9
10    int vertices_orig_size = 0;
11    int textures_orig_size = 0;
12
13    for (int i = 0; i < file.Length; i++)
14    {
15        if (file[i][0] == 'v' && file[i][1] == ' ')
16        {
17            vertices_orig_size += 3;
18        }
19
20        if (file[i][0] == 'v' && file[i][1] == 'n' && file[i][2] == ' ')
21        {
22            normals_size += 3;
23        }
24
25        if (file[i][0] == 'v' && file[i][1] == 't' && file[i][2] == ' ')
26        {
27            textures_size += 2;
28        }
29
30        if (file[i][0] == 'f' && file[i][1] == ' ')
31        {
32            indices_size += 3;
33            vertices_size += 24;
34        }
35
36        progress++;
37    }
38
39    vertices = new float[vertices_size];
40    vertices_orig = new float[vertices_orig_size];
41    normals = new float[normals_size];
42    textures = new float[textures_size];
43    indices = new uint[indices_size];
44
45    int vertices_count = 0;
46    int vertices_orig_count = 0;
47    int normals_count = 0;
48    int indices_count = 0;
49    int textures_count = 0;
50
```

```
51      for (int i = 0; i < file.Length; i++)
52      {
53          if (file[i][0] == 'v' && file[i][1] == ' ')
54          {
55              string digit = "";
56              bool is_digit = false;
57
58              for (int j = 0; j < file[i].Length; j++)
59              {
60                  if (is_digit == true)
61                  {
62                      if (file[i][j] == ' ')
63                      {
64                          vertices_orig[vertices_orig_count] = float.Parse(
                               digit, CultureInfo.InvariantCulture.NumberFormat);
65                          vertices_orig_count++;
66
67                          digit = "";
68                      }
69                      else if (j == file[i].Length - 1)
70                      {
71                          vertices_orig[vertices_orig_count] = float.Parse(
                               digit, CultureInfo.InvariantCulture.NumberFormat);
72                          vertices_orig_count++;
73                          digit = "";
74                      }
75                      else
76                      {
77                          digit += file[i][j];
78                      }
79                  }
80                  else if (file[i][j] == ' ')
81                  {
82                      is_digit = true;
83                  }
84              }
85          }
86          else if (file[i][0] == 'v' && file[i][1] == 'n' && file[i][2] == ' ')
87          {
88              string digit = "";
89              bool is_digit = false;
90
91              for (int j = 0; j < file[i].Length; j++)
92              {
93                  if (is_digit == true)
94                  {
95                      if (file[i][j] == ' ')
96                      {
97                          normals[normals_count] = float.Parse(digit,
                               CultureInfo.InvariantCulture.NumberFormat);
98                          normals_count++;
99
100                         digit = "";
101                     }
```

```
102                     else if (j == file[i].Length - 1)
103                     {
104                         normals[normals_count] = float.Parse(digit,
                                CultureInfo.InvariantCulture.NumberFormat);
105                         normals_count++;
106                         digit = "";
107                     }
108                     else
109                     {
110                         digit += file[i][j];
111                     }
112                 }
113                 else if (file[i][j] == ' ')
114                 {
115                     is_digit = true;
116                 }
117             }
118         }
119         else if (file[i][0] == 'v' && file[i][1] == 't' && file[i][2] == ' ')
120         {
121             string digit = "";
122             bool is_digit = false;
123 
124             for (int j = 0; j < file[i].Length; j++)
125             {
126                 if (is_digit == true)
127                 {
128                     if (file[i][j] == ' ')
129                     {
130                         textures[textures_count] = float.Parse(digit,
                                CultureInfo.InvariantCulture.NumberFormat);
131                         textures_count++;
132 
133                         digit = "";
134                     }
135                     else if (j == file[i].Length - 1)
136                     {
137                         textures[textures_count] = float.Parse(digit,
                                CultureInfo.InvariantCulture.NumberFormat);
138                         textures_count++;
139                         digit = "";
140                     }
141                     else
142                     {
143                         digit += file[i][j];
144                     }
145                 }
146                 else if (file[i][j] == ' ')
147                 {
148                     is_digit = true;
149                 }
150             }
151         }
152 
```

```
153         progress++;
154     }
155 
156     for (int i = 0; i < file.Length; i++)
157     {
158         if (file[i][0] == 'f' && file[i][1] == ' ')
159         {
160             string digit = "";
161             int data_type = 0;
162 
163             for (int j = 0; j < file[i].Length; j++)
164             {
165                 if (data_type == 0)
166                 {
167                     if (file[i][j] == ' ')
168                     {
169                         data_type = 1;
170                     }
171                 }
172                 else if (data_type == 1)
173                 {
174                     if (file[i][j] == '/')
175                     {
176                         vertices[vertices_count] = vertices_orig[(int.Parse(
                                digit) - 1) * 3];
177                         vertices_count++;
178 
179                         vertices[vertices_count] = vertices_orig[(int.Parse(
                                digit) - 1) * 3 + 1];
180                         vertices_count++;
181 
182                         vertices[vertices_count] = vertices_orig[(int.Parse(
                                digit) - 1) * 3 + 2];
183                         vertices_count++;
184 
185 
186                         indices[indices_count] = uint.Parse(digit,
                                CultureInfo.InvariantCulture.NumberFormat);
187                         indices_count++;
188 
189                         digit = "";
190                         data_type++;
191                     }
192                     else
193                     {
194                         digit += file[i][j];
195                     }
196                 }
197                 else if (data_type == 2)
198                 {
199                     if (file[i][j] == '/')
200                     {
201                         vertices[vertices_count] = textures[(int.Parse(digit)
                                - 1) * 2];
```

```
202                         vertices_count++;
203
204                         vertices[vertices_count] = textures[(int.Parse(digit)
                                - 1) * 2 + 1];
205                         vertices_count++;
206
207                         digit = "";
208                         data_type++;
209                     }
210                     else
211                     {
212                         digit += file[i][j];
213                     }
214                 }
215                 else if (data_type == 3)
216                 {
217                     if (file[i][j] == ' ')
218                     {
219                         vertices[vertices_count] = normals[(int.Parse(digit)
                               - 1) * 3];
220                         vertices_count++;
221
222                         vertices[vertices_count] = normals[(int.Parse(digit)
                               - 1) * 3 + 1];
223                         vertices_count++;
224
225                         vertices[vertices_count] = normals[(int.Parse(digit)
                               - 1) * 3 + 2];
226                         vertices_count++;
227
228                         digit = "";
229                         data_type = 1;
230                     }
231                     else if (j == file[i].Length - 1)
232                     {
233                         digit += file[i][j];
234
235                         vertices[vertices_count] = normals[(int.Parse(digit)
                               - 1) * 3];
236                         vertices_count++;
237
238                         vertices[vertices_count] = normals[(int.Parse(digit)
                               - 1) * 3 + 1];
239                         vertices_count++;
240
241                         vertices[vertices_count] = normals[(int.Parse(digit)
                               - 1) * 3 + 2];
242                         vertices_count++;
243
244                         digit = "";
245                         data_type = 1;
246                     }
247                     else
248                     {
```

```
249                         digit += file[i][j];
250                     }
251                 }
252             }
253         }
254
255         progress++;
256     }
257
258     for (int i = 0; i < indices_size; i++)
259     {
260         indices[i] = (uint)i;
261     }
262 }
```

Место для диска