# СОДЕРЖАНИЕ

**Введите название главы (уровень 1)1**

Введите название главы (уровень 2)2

Введите название главы (уровень 3)3

**Введите название главы (уровень 1)4**

Введите название главы (уровень 2)5

Введите название главы (уровень 3)6

# Введение

## Тема работы

Разработка 3d графического движка на c++ с использованием набора

библиотек openGL.

## Актуальность работы

Около трёх лет назад, когда я изучал стереометрию в школе, у меня возникла потребность в моделировании некоторых тел в трёхмерном пространстве. С тех пор эта потребность не только не исчезла, но и усилилась – со временем математика становилась лишь труднее в понимании. Вместе с тем у меня возникло желание научиться визуализировать и анимировать трёхмерную графику, но в те времена у меня не было достаточного количества теоретических и практических знаний, чтобы продвинуться в решении этого вопроса.

В современном мире технологии трёхмерного моделирования находят множество применений – на производствах, при проектировании различных моделей для их дальнейшей печати на принтере, могут применяться для обучения студентов и школьников, создании компьютерных игр и фильмов. В условиях ухода многих компаний из России можно задуматься о создании отечественного графического движка.

Для автора этой работы польза заключается в получении опыта по разработке программного продукта, закреплении некоторых принципов линейной алгебры и аналитической геометрии.

## Цель работы

Цель работы – в рамках курсовой работы за 2 семестр создать программное обеспечение, с использованием которого можно будет моделировать те или иные трёхмерные модели, визуализация которых нужна самому автору работы.

## Задачи работы

1. Провести поиск информации о графических движках.
2. Найти как минимум три графических движка высокого уровня, которые уже существуют, отметить их достоинства и недостатки.
3. Составить алгоритм работы движка в виде диаграммы процессов.
4. Составить иерархию объектов системы в виде диаграммы классов.
5. Разработать программный продукт на языке C++.
6. Протестировать работу продукта и сделать выводы о его степени совершенства.
7. Составить отчёт по проделанной работе и защитить её.

## Критерии достижения успеха

Таблица 1.4 – Критерии к продукту

|  |  |  |
| --- | --- | --- |
| № | Критерий | Значение |
| 1 | Поддерживаемые форматы 3d моделей | .obj |
| 2 | Поддерживаемые форматы текстур | .jpg |
| 3 | Частота кадров на компьютере с процессором Intel Core i5 11 поколения и видеокартой NVIDIA RTX 3050 (или лучшие аналоги) при работе тестовых моделей, представленных самим автором курсовой работы | >= 60 |
| 4 | Количество тестовых моделей, пригодных для визуализации на движке | >=3 |
| 5 | Возможность приближённо визуализировать математические поверхности вида z = f(x,y) | Да |
| 6 | Предельное количество вершин, одновременно отображаемых движком, при условиях из критерия 3 | >= 100000 |
| 7 | Возможность двигать и вращать модель | Да |
| 8 | Возможность менять положение камеры, угол её тангажа и крена | Да |
| 9 | Максимально допустимое программой количество дочерних элементов одного объекта, не учитывая условия из критерия 3 | >= 2 147 483 647 |
| 10 | Поддерживаемые разрешения экрана | До 1920x1080 (FullHD) |

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Обзор области разработки

Перефразируя определение понятным языком, компьютерная графика – это изображение некоторых объектов на экране компьютера, при этом компьютерное изображение – это изображение, которое получено вычислительными средствами компьютера, посредством какой-либо компьютерной программы. (Васильев В. Е., Морозов А. В. Компьютерная графика: Учеб. Пособие. – СПб.: С3ТУ, 2005. – страница 4.)

Соответственно, графическим движком можно считать некоторое обобщение над процессом создания изображений на экране компьютера. Само это обобщение зависит от конкретных целей, которые преследует создатель движка. Например, если нужно выводить на экран файлы формата .jpg – то графический движок будет представлять из себя программу, которая декодирует байты из файла в памяти компьютера, затем переводит их в формат, понятный графическому процессору, и отправляет их в оперативную память, чтобы компьютер закрасил определённые пиксели экрана в нужный цвет. Я же хочу моделировать трёхмерную графику. Поэтому в контексте этой работы, под графическим движком понимается некоторая программа, которая из некоторого набора вершин и их цветов формирует на экране компьютера (закрашивает пиксели по определённому правилу) изображение, которое уже мы, люди, воспринимаем как некоторый трёхмерный объект.

Понятное дело, что само изображение плоское, а мир, в котором мы живём, объёмный, но тут следует вспомнить, что само наше зрение – это тоже иллюзия, создаваемая органами чувств, пусть и довольно точная. Предполагается, что программа будет использовать некоторые правила перспективы и оптики, чтобы нам казалось, что мы видим объёмный объект. Например, можно не рисовать те части объекта, которые находятся за другими. А те части, которые находятся ближе к точке обзора, будут больше, чем те, которые находятся на некотором удалении. (<https://habr.com/ru/articles/324968/>)

Рассмотрим также некоторые области применения графических движков. (Васильев В. Е., Морозов А. В. Компьютерная графика: Учеб. Пособие. – СПб.: С3ТУ, 2005. – страница 7)

1. Научная графика – для изображения процессов самых различных наук наглядным образом.
2. Деловая графика – для изображения схем, графиков, таблиц. Используется преимущественно в бизнес-аналитике.
3. Конструкторская графика – для визуализации чертежей.
4. Полиграфия – для множественного копирования какого-либо изображения.
5. Web-разработка – для вёрстки страниц html.
6. Мультимедиа – для создания интерактивных приложений, в том числе и игр.

Для вычислений позиции и цвета пикселя используют, как правило, вычислительные мощности графического процессора. В отличие от обычного процессора, графический процессор способен выполнять операции одновременно для большого количества входных данных, что очень полезно, когда нужно изображать объект в режиме реального времени. Такая необходимость возникает в случае интерактивности движка, т.е. когда пользователь имеет возможность изменять изображение прямо во время работы программы (например, вращать модель, двигать её вдоль какой-либо оси и т.д.).

Итак, в зависимости от наших нужд, движок может быть двух типов:

1. Движок, который выводит одно изображение, которое и является результатом его работы. Таким движком является, например, программа «просмотр фотографий Windows», или какая-нибудь программа, которая очень точно изображает силовые линии магнитного поля по определённым заданным параметрам этого поля и масштабу изображения.
2. Движок, который оперирует меньшим числом входных данных, но при этом постоянно обновляет изображение на экране, до тех пор, пока его работа не будет завершена пользователем. Например, к таким движкам относятся видеопроигрыватели, современные компьютерные игры.

## Обзор существующих продуктов

Выделим некоторые уже существующие графические движки, которые используют С++ и написаны в парадигме ООП. Данные требования выдвигаются в соответствии с требованиями к курсовой работе, иначе сравнение будет некорректным (у других решений будет доступ ко всем технологиям, а у автора – только к вышеобозначенным).

Наиболее известными графическими движками (<https://tproger.ru/digest/cpp-best-gui/>) являются следующие решения:

1. SFML (Simple and Fast Multimedia Library) – движок для рисования, в основном, двухмерной графики. В своей основе использует набор библиотек openGL. Из преимуществ – возможность нативно использовать функции более низкого уровня, выделять память и передавать буферы с данными на видеокарту. Также большим плюсом является наличие хорошей документации и множества обучающих статей и книг на русском языке, что позволяет сократить время обучения и быстрее перейти к разработке. К недостаткам можно отнести плохую совместимость с 3д (для этого проще использовать нативный openGL) и отсутствие поддержки directX.
2. SDL – движок для создания двумерных игр. К плюсам: простое создание окна, наличие функций для работы со звуком и широкий круг применения благодаря тому, что она написана на С. Тем не менее, программистам приходится реализовывать множество собственных обёрток и абстрагироваться на более высокий уровень разработки, поскольку движок сохраняет близость к «железу». Для программирования 3д графики всё также выгоднее использовать тот же openGL или directX. Документации также много, как и книг по этой библиотеке, что ускоряет разработку продукта.
3. Ogre 13.6 – высокоуровневый объектно-ориентированный движок для создания трёхмерной графики, имеет поддержку openGL и directX. Является наиболее высокоуровневым программируемым движком на чистом C++. Сложен в освоении, поскольку имеет множество уже реализованных функций. К сожалению, документации не так много, как и книг и статей на русском языке. Автор курсовой работы столкнулся с большими трудностями при попытке внедрения этой технологии для создания своего игрового движка, поскольку пакет разработки для этого решения требует ручной установки.
4. wxWidgets – библиотека для отрисовки GUI. Она очень производительна по сравнению с такими аналогами, как Qt. Её отличительной особенностью является использование встроенных в конкретную систему элементов внешнего вида. К таким относятся всякого рода кнопки, полосы прокрутки и другие элементы. Ещё к плюсам этой библиотеки можно отнести её кроссплатформенность на уровне исходного кода.
5. Qt – набор библиотек и фреймворков для создания игр, GUI и различной графики. Имеет платную лицензию. Специально под эту платформу созданы некоторые IDE: QtDesigner, QtCreator, QDevelop, Edyuk. При этом, несмотря на платную лицензию, Qt явялется open-source проектом.

# технологическая часть

## 2.1 Схема работы системы

Для отрисовки низкоуровневой графики (треугольник, вершина) я предпочёл использовать openGL. Конечно, можно отметить распространённость этой технологии и хорошую совместимость как с видеокартами, так и с процессорами, но этими качествами обладает и DirectX. Основным плюсом openGL для меня, как разработчика, является наличие обширной документации и уроков на русском языке, чем не может похвастаться DirectX.

Итак, мы имеем низкоуровневую библиотеку openGL, которой мы можем передать набор вершин. Также имеется окно приложения, которое может отобразить эти вершины на экране, но изначально это только двумерная графика (треугольники в плоскости нашего экрана).

Роль моего движка – упростить жизнь человека, который хочет изобразить какой-либо **трёхмерный** объект на **двумерном** экране компьютера. Здесь мы сталкиваемся с некоторыми трудностями:

1. Как правило, людям нужно изобразить что-то более сложное, чем просто двумерный примитив (точка, треугольник, прямоугольник), иначе они бы просто воспользовались встроенным приложением MS Paint. Проблема формулируется следующим образом: как хранить и визуализировать большое (большее 10000) число вершин? Эта проблема имеет известное решение – использовать какой-либо формат файла для хранения таких вершин, например, формат OBJ.
2. Как изобразить на двумерном экране трёхмерный объект? На самом деле, можно сказать, что наше восприятие является одним из ответов на этот вопрос. Видим мы только двумерное изображение, но при этом воспринимаем мир объёмным. Всё благодаря тому, что наш мозг научился понимать законы оптики и перспективы. Поэтому мы поступим аналогичным образом – создадим точку обзора - иллюзию того, что мы смотрим на объект со стороны. При этом будем называть эту точку обзора Камерой. У камеры будет некоторая область обзора (FOV, Field Of View). Подобно нашей голове, камера может вращаться вокруг и вниз-наверх, поэтому она будет иметь переменные тангажа и рыскания. Итак, точки объекта в трёхмерном пространстве преобразовались в точки в пространстве Камеры. В аналитической геометрии для таких преобразований используют матрицы преобразований – мы будем действовать похожим образом.

Представим, что пользователь движка хочет изобразить какой-либо объект на экране. Ему необходимо выполнить для этого следующую последовательность действий:

1. Предоставить файл формата OBJ в качестве трёхмерной модели и текстуру в виде картинки JPG. Движок считает файл и преобразует все данные модели в набор вершин, а картинку – в набор байт, которые уже можно передать в функции openGL.
2. Написать (или позаимствовать) шейдеры – вершинный (который преобразует позиции) и фрагментный (который преобразует цвета). В простейшем случае вершинный шейдер будет создавать изображение на Камере (умножать матрицу преобразований на исходный вектор), а фрагментный – просто оставлять исходный цвет.

Отметим, что это довольно небольшой список действий, чего мы как раз и добиваемся – упрощаем жизнь пользователю программы. Мы инкапсулируем низкоуровневые операции с памятью видеокарты. При этом движок проделывает следующие операции:

1. Построчно читает файл OBJ как текст, разбивая его на части, где указаны координаты вершин, текстурные координаты и многоугольники (как правило, трёх- и четырёхугольники), которые представляют из себя индексы позиций вершин и текстурных координат (до этого мы собирали координаты вершин и текстурные координаты в отдельный массив). Например, если у нас есть некоторый многоугольник вида *1/2 2/1 3/1*, то эта запись означает треугольник, первая вершина которого имеет позицию с номером 1 (надо посмотреть в тот массив с вершинами и подставить вместо этого номера конкретные позиции), вторая – 2, третья – 3, при этом первая вершина окрашена в текстурную позицию с номером 2, а другие две – в позицию с номером 1. В конечном итоге движок преобразует всю эту информацию к вектору чисел с плавающей точкой, который построен по следующему виду: первые три числа – позиции вершины, 4, 5, 6 число – соответственно нормализованные значения r, g, b каналов, 7 и 8 число – текстурные координаты вершины (то есть, какая точка из двумерной текстуры будет подставлена в качестве цвета). На этом информация о первой вершине заканчивается, и начинается информация о следующей вершине. При этом, для всех объектов мы используем метод триангуляции, поэтому каждые три вершины (24 индекса в векторе) – это информация об одном треугольнике.
2. Движок компилирует предоставленные ему шейдеры, при ошибке компиляции движок выведет соответствующее сообщение об ошибке. Скомпилированные шейдеры преобразуются в шейдерную программу, которая будет передана видеокарте.
3. Представленную картинку движок преобразует в массив байт и предоставляет движку openGL.
4. Вершины объекта загружаются в память видеокарты.
5. Движок сообщает openGL, сколько позиций уходит на одну вершину в массиве вершин, на каких позициях находятся координаты вершин, текстурные координаты и цвета.
6. После успешной загрузки начинается главный цикл отрисовки. В каждом цикле движок обновляет значений матриц поворота и предоставляет их видеокарте, чтобы она умножила **каждую** точку объекта на эту матрицу. Мы очищаем экран каким-то цветом, затем вызываем функцию отрисовки и указываем в ней, какой буфер вершин, уже загруженный в память видеокарты, нам рисовать. На этом цикл заканчивается.

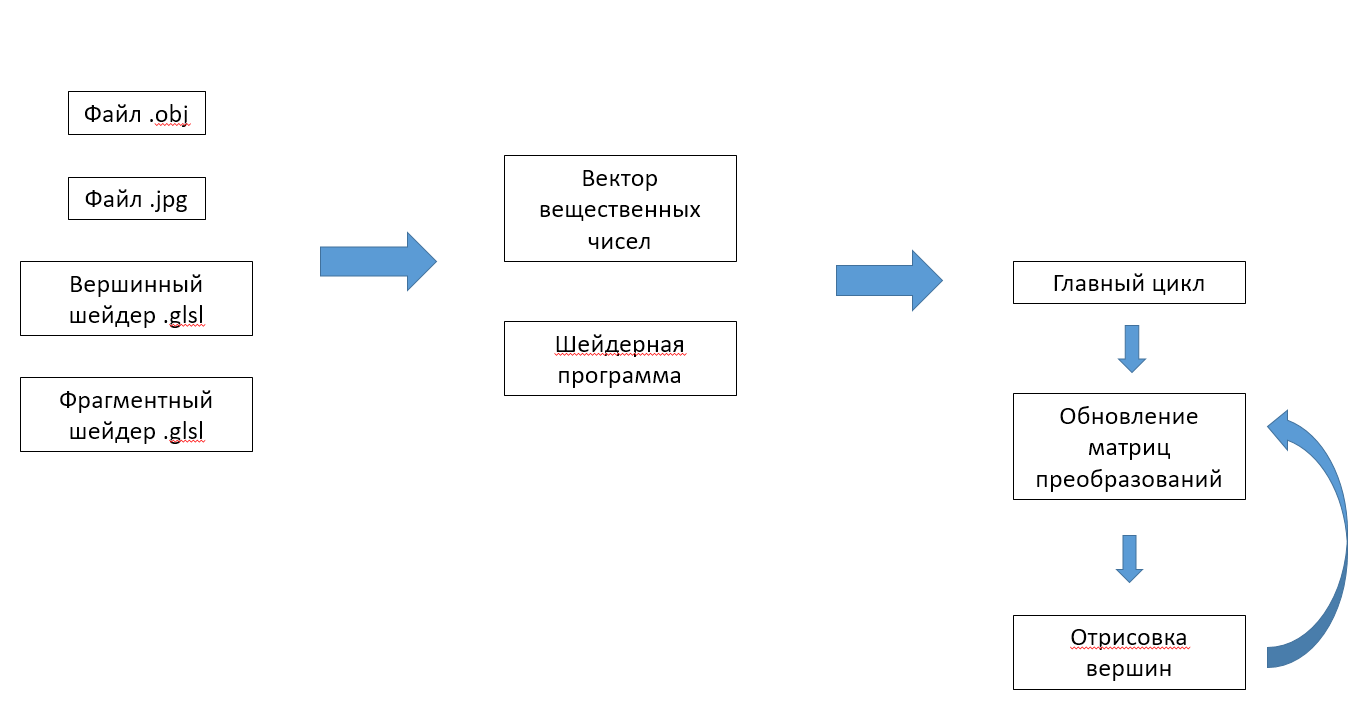


Рисунок 2.1.1 – схема работы движка

## Архитектура классов

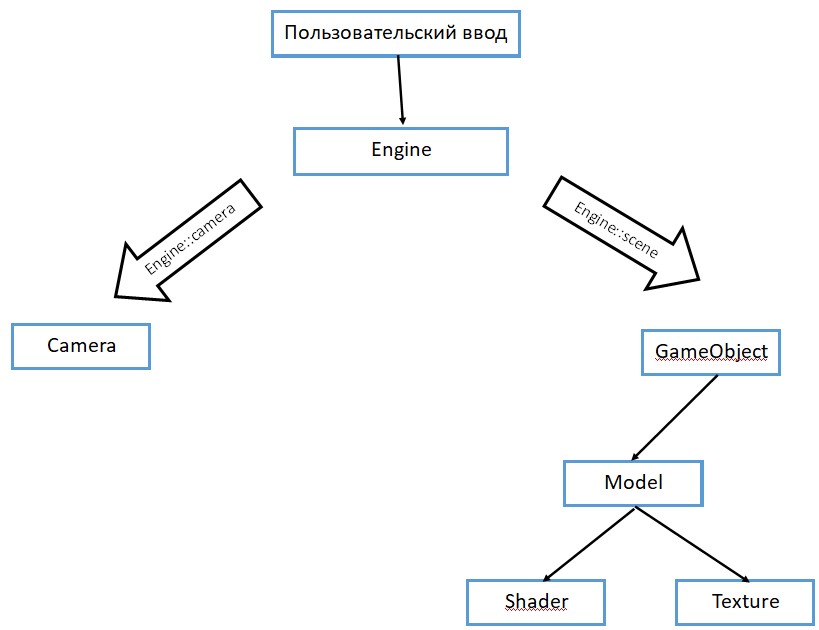


Рисунок 2.2.1 – схема классов проекта

Большинство классов представляют собой обёртки для низкоуровневых функций openGL, предоставляя возможность группировать часто используемые участки кода.

### Класс Shader

Класс Shader отвечает за жизненный цикл шейдеров. У каждого шейдера есть следующие переменные-члены:

1. Путь к файлу с исходным кодом
2. Исходный код в строковом формате
3. Идентификатор доступа к скомпилированному шейдеру низкого уровня, который представляет из себя некоторое 32-х битное беззнаковое число. По этому идентификатору можно обратиться к openGL, чтобы сделать что-то с шейдером, например, удалить и заменить на другой.

Класс реализует простую загрузку и компиляцию шейдера из файла.

### Класс Texture

Класс реализует загрузку текстуры из файла (на данный момент только формата .jpg). После загрузки текстуру можно прикрепить к какому-либо игровому объекту.

У каждой текстуры есть следующие переменные-члены:

1. Путь к файлу с текстурой
2. Размеры картинки
3. Массив байт с текстурой
4. Идентификатор доступа к текстуре низкого уровня, по которому можно обратиться к openGL

### Класс Model

Класс является обёрткой для хранения основной информации о 3д-модели:

1. Позиции вершин
2. Текстурные координаты вершин
3. Число изображаемых треугольников
4. Индексы и названия тех вершин, с который начинается та или иная дочерняя модель исходной модели. Например, модель «Самолёт» может содержать внутри модель «Винт», «Правое крыло». Затем можно будет использовать эти индексы, чтобы разделить исходную модель на дочерние, создать на их основе другие экземпляры класса GameObject и даже задать для них собственную логику поведения, например, ускорять «Винт» самолёта по нажатию на клавишу «стрелка влево».

### Класс GameObject

Ключевой класс всего движка. Экземпляр класса имеет следующие переменные-члены:

1. Идентификатор доступа к шейдерной программе, которая представляет из себя соединённые скомпилированные вершинный и фрагментный шейдеры.
2. Буферы Vertex Array Object, Vertex Buffer Object, Element Buffer Object, которые необходимы для передачи буфера вершин и индексов на GPU.
3. Способ изображения на экране. Можно интерпретировать буфер вершин как последовательность точек, треугольников, прямоугольников или линий.
4. Указатели на дочерние и родительские игровые объекты.
5. Указатель на буфер вершин.
6. Экземпляр структуры Transform, которая содержит актуальные матрицы перехода – поворота, переноса позиции и др.
7. Указатель на функцию, которая вызывается каждый раз, когда необходимо передать какие-либо данные в шейдерную программу.
8. Масштаб, который применяется на этапе считывания вершин из файла.

Класс GameObject реализует создание игрового объекта из массива вершин, загрузку вершин, текстуры и шейдеров из файла, поворот объекта вокруг указанной оси, изменение положения объекта в пространстве.

### Класс Camera

Объект этого класса хранит информацию о Камере, например, значения рыскания и тангажа. У камеры есть метод, который вызывается при каждой смене кадра и обновляет данные камеры в зависимости от пользовательского ввода.

### Класс Engine

Экземпляр этого класса занимается запуском приложения в несколько этапов:

1. Загрузка библиотеки openGL
2. Запуск окна
3. Запуск библиотеки Glew
4. Создание Viewport`а (отвечает за параметры окна)
5. Назначение функции для обработки пользовательского ввода
6. Установка некоторых параметров openGL, например, сглаживания

Класс Engine находится выше всех других классов в иерархии, имеет доступ к объекту Камеры, а также Сцену – список всех изображаемых объектов. Объекты сцены будут обновлены в главном цикле программы, который активируется в функции main.

На этом можно завершить рассмотрение классов продукта. Отдельные детали исходного кода будут рассмотрены в следующей главе.