Реферат

Выпускная квалификационная работа 47 с., 8 рис., 8 табл., 4 ист.

OPENGL, С++, GLSL, ШЕЙДЕРЫ, ВЕРШИННЫЙ ШЕЙДЕР, ФРАГМЕНТНЫЙ ШЕЙДЕР.

Разработка редактора GLSL программ.

Объектом исследования является графический стандарт в области компьютерной графики – OpenGL.

Цель выпускной квалификационной работы - разработка программного продукта, основная возможность которого - редактирование исходного кода GLSL программ.

Результатом выпускной квалификационной работы является приложение, позволяющее управлять графическим конвейером OpenGL, с возможностью моментальной компиляции вершинного и фрагментного шейдеров, написанных на языке программирования GLSL.

Разработанный продукт был создан на языке программирования C++. Основным инструментом разработки был выбран программный продукт от компании Microsoft - Visual Studio 2017.

В соответствии с требованиями к выпускным квалификационным работам бакалавров были рассмотрены вопросы экономической эффективности данного проекта.

Содержание

[Реферат 3](#_Toc10482220)

[Введение 5](#_Toc10482221)

[1 Технология OpenGL 6](#_Toc10482222)

[1.1 Основные положения 6](#_Toc10482223)

[1.2 Графический конвейер OpenGL 7](#_Toc10482224)

[1.3 Язык GLSL 9](#_Toc10482225)

[1.4 Вершинный и фрагментный шейдер 10](#_Toc10482226)

[2 Разработка приложения 13](#_Toc10482227)

[2.1 Требования к программному продукту 13](#_Toc10482228)

[2.2 Выбор поддерживаемых форматов 3D моделей 13](#_Toc10482229)

[2.3 Выбор технологий для разработки 15](#_Toc10482230)

[2.3.1 Выбор языка программирования 15](#_Toc10482231)

[2.3.2 Выбор зависимых библиотек 16](#_Toc10482232)

[2.3.3 Выбор среды разработки 17](#_Toc10482233)

[3 Описание работы с программным продуктом 18](#_Toc10482234)

[4 Основные задачи, которые были решены 20](#_Toc10482235)

[5 Экономическая часть 23](#_Toc10482236)

[5.1 Введение 23](#_Toc10482237)

[5.2 Построение дерева задач проекта 25](#_Toc10482238)

[5.3 Диаграмма Ганта 27](#_Toc10482239)

[5.4 Сетевой график проекта 29](#_Toc10482240)

[5.5 Расчёт стоимости проекта 33](#_Toc10482241)

[5.5.1 Трудовые ресурсы 33](#_Toc10482242)

[5.5.2 Расчёт почасовой ставки команды проекта 34](#_Toc10482243)

[5.5.3 Распределение ресурсов 34](#_Toc10482244)

[5.5.4 Расчёт заработной платы 37](#_Toc10482245)

[5.5.5 Материальные ресурсы 38](#_Toc10482246)

[5.5.6 Расчёт амортизации 38](#_Toc10482247)

[5.5.7 Затраты на ресурсы компьютерной сети «Интернет» 39](#_Toc10482248)

[5.5.8 Общие затраты на проект 39](#_Toc10482249)

[5.6 Оценка рисков 39](#_Toc10482250)

[5.7 Вывод 45](#_Toc10482251)

[Заключение 46](#_Toc10482252)

[Список используемых источников 47](#_Toc10482253)

Введение

В данной выпускной квалификационной работе представлена разработка и реализация редактора GLSL программ, которые управляют одним из этапов графического конвейера технологии OpenGL, Данный программный продукт нацелен на упрощение трудовой деятельности программистов, специализирующихся на данной технологии и нуждающихся в удобном инструменте для написания шейдерных программ.

1 Технология OpenGL

1.1 Основные положения

На сегодняшний день трёхмерную компьютерную графику можно увидеть во многих областях нашей жизни, начиная от кинематографа и заканчивая системами симуляций в реальном времени. Однако это было не всегда. Когда производительные мощности компьютеров были не так высоки, не существовало единого открытого стандарта в данной области. Каждая программа, нуждающаяся в 3D графике, использовала свои графические функции или методы. Спустя время с появлением графических ускорителей и более быстрых процессоров производители программного обеспечения столкнулись с проблемой отсутствия стандартов, позволявших писать программы, которые были бы независимы от операционной системы и оборудования компьютера. Спустя некоторые время такой стандарт появился и называется он OpenGL.

OpenGL – один из самых популярных графических стандартов в области компьютерной графики во всём мире. В 1982 году в Стенфордском университете была разработана идея графической машины. Данная идея послужила толчком для создания конвейера рендеринга компанией Silicon Graphics. Таким образом появилась библиотека IRIS GL, на основе которой в 1992 году была утверждена и опубликована спецификация OpenGL.

OpenGL (Open Graphics Library) переводится как открытая графическая библиотека. Сама библиотека является кроссплатформенной. Это означает, что программы, которые используют данную технологию, можно переносить на различные платформы, и результат получится везде один и тот же. Также она является аппаратно-независимой. Программисту не нужно писать программы для определённого оборудования. Если устройство не поддерживает кукую-либо функцию, то она будет выполнено программно, в противном случае – аппаратно.

Для программиста OpenGL это программный интерфейс, управляющий графическими устройствами, такими как графические ускорители. Он включает более 300 функций для рисования сложных трёхмерных сцен, состоящих из простых примитивов, которые определяет программист.

OpenGL является только воспроизводящей графику библиотекой, она не занимается обработкой ввода информации с различных устройств (мышь, клавиатура и т. д.). Созданием окон операционной систем данная технология также не занимается.

1.2 Графический конвейер OpenGL

Основная задача технологии OpenGL – это поэтапное преобразование координат из 3D пространства в цветные 2D пиксели на мониторе компьютера пользователя. Данный процесс перехода одного в друге управляется **графическим конвейером OpenGL**, названный по аналогии с конвейерами в машиностроении, который состоит из различных этапов, требующих на свой вход результат работы прошлого.

До версии 2.0 в OpenGL графический конвейер был с фиксированной функциональностью (fixed-function pipeline). Это значит, что программист имел ограниченные возможности для настройки алгоритмов отображения графики. Если ему нужно было разработать более реалистичный или более сложный эффект, то он использовал различные трюки, для того чтобы повысить гибкость фиксированного конвейера. На сегодняшний день такой конвейер объявлен устаревшим, и заменой ему служит конвейер с программированной функциональностью.

У программистов появился такой замечательный инструмент как **шейдерные программы (shader programs)**, которые предназначены для управления прежде фиксированными алгоритмами. Любая программа, которая написана с помощью OpenGL, использует внутри себя одну или несколько шейдерных программ. Обычно они состоят из компонентов, которые названы **шейдерами (shaders)**. Каждый шейдер выполняется на графическом процессоре (GPU) и предназначен для воспроизведения алгоритма, который так или иначе связан с эффектами затенения и освещения в изображении. Однако, они могут и намного большее! Например, воспроизводить анимацию, производить универсальные математические вычисления и прочее.

На рисунке ниже (Рисунок 1) представлено примерное представление этапов графического конвейера OpenGL. Подписанными обозначены этапы, которые программист может заменить собственными.

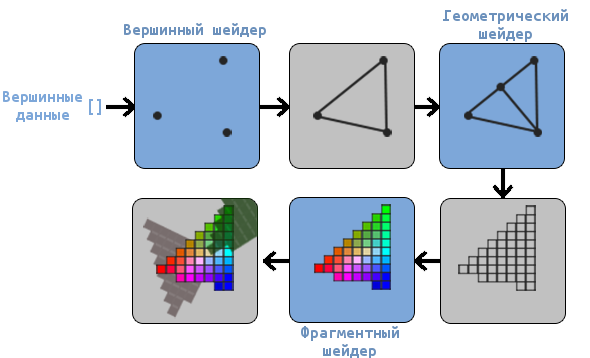


Рисунок 1 – Этапы графического конвейера OpenGL

Шейдеры действуют параллельно. К примеру, один из видов шейдеров – фрагментный, предназначенный для обработки пикселей, вызывается непосредственно для каждого пикселя и все они обрабатываются одновременно в отдельных потоках выполнения на графическом процессоре (GPU).

Шейдеры будучи программами создаются на специальном C-подобном языке GLSL, речь о котором пойдёт в следующем разделе.

1.3 Язык GLSL

GLSL (OpenGL Shading Language) – язык шейдеров OpenGL в данный момент представляет собой фундаментальную базу и неотделяемую часть программирования с использованием данной технологии. Именного он помогает программисту получить всю мощь графических процессоров для создания улучшенных приёмов отображения графики.

Относится к языкам высокого уровня. Его синтаксис схож с языком программирования C, однако, некоторые возможности были из него исключены для увеличения производительности и из-за того, что он применяется для особых целей.

В таблице ниже (Таблица 1) представлены версии языка и версии OpenGL, в которых они поддерживаются.

Таблица 1 – Версии языка GLSL

|  |  |  |
| --- | --- | --- |
| **GLSL версия** | **Версия OpenGL** | **Дата** |
| 1.10 | 2.0 | 30 Апреля 2004 |
| 1.20 | 2.1 | 7 Сентября 2006 |
| 1.30 | 3.0 | 22 Ноября 2009 |
| 1.40 | 3.1 | 22 Ноября 2009 |
| 1.50 | 3.2 | 4 Декабря 2009 |
| 3.30 | 3.3 | 11 Марта 2010 |
| 4.00 | 4.0 | 24 Июля 2010 |
| 4.10 | 4.1 | 24 Июля 2010 |

Окончание таблицы 1

|  |  |  |
| --- | --- | --- |
| 4.20 | 4.2 | 12 Декабря 2011 |
| 4.30 | 4.3 | 7 Февраля 2013 |
| 4.40 | 4.4 | 16 Июня 2014 |
| 4.50 | 4.5 | 9 Мая 2017 |
| 4.60 | 4.6 | 14 Июня 2018 |

Ниже представлен пример простого фрагментного шейдера на GLSL:

#version 330

void main(void)

{

gl\_FragColor = vec4 (1.0, 0.0, 0.0, 1.0);

}

1.4 Вершинный и фрагментный шейдер

В последних версиях технологии OpenGL имеется шесть различных типов шейдеров: вершинные, геометрические, шейдеры управления тесселяцией, шейдеры вычисления тесселяции, фрагментные и вычислительные. В данной выпускной работе будут рассмотрены только вершинные и фрагментные шейдеры, так как они являются наиболее важными для визуализации различных эффектов.

Шейдеры представляют собой заменяемые части графического конвейера OpenGL. Если быть абсолютно точными, то эти части являются программируемыми. На рисунке (Рисунок 2) представлена более простая схема конвейера OpenGL, которая состоит из вершинного и фрагментного шейдеров.

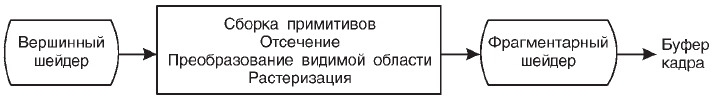


Рисунок 2 – Упрощённая схема конвейера OpenGL

Информация о вершинах передается вниз по конвейеру и попадает в вершинный шейдер через его входные переменные. В общем случае шейдеры получают исходные данные через входные переменные, определяемые программно, а данные в эти переменные записываются основной программой OpenGL или предыдущими шейдерами. Например, входные переменные фрагментного шейдера могут получать данные из выходных переменных вершинного шейдера. Данные могут также передаваться любым шейдерам через uniform-переменные. Эти переменные применяются для передачи информации, которая изменяется гораздо реже, чем атрибуты вершин (например, для передачи матриц, координат источника света и других параметров сцены). На рисунке 3 представлена диаграмма, изображающая отношения между входными и выходными переменными при наличии двух активных шейдеров (вершинного и фрагментного).

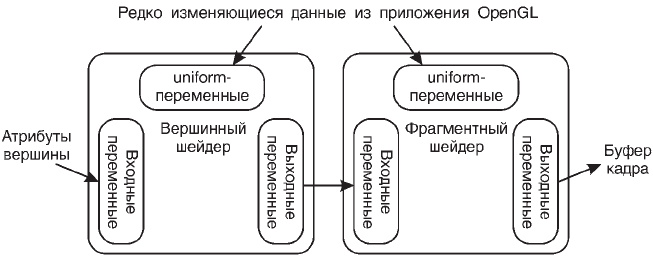


Рисунок 3 – Диаграмма отношений между входными и выходными переменными

Вершинный шейдер выполняется один раз для каждой вершины, причем, как правило, вершины обрабатываются параллельно. Координаты, соответствующие вершине, должны быть преобразованы в усеченные координаты (clip coordinates) и записаны в выходную переменную gl\_Position до того, как вершинный шейдер закончит выполнение. Вершинный шейдер может также передавать вниз по конвейеру (через выходные переменные) другую информацию. Например, в вершинном шейдере может вычисляться цвет вершины. Этот цвет можно передать дальше через соответствующую выходную переменную.

Между вершинным и фрагментным шейдером выполняются сборка вершин в примитивы, усечение видимой области и ее преобразование (наряду с другими операциями). После этого выполняются процедура растеризации и заливка полигона (если необходимо). Фрагментный шейдер выполняется один раз для каждого фрагмента (пикселя) отображаемого полигона (как правило, фрагменты обрабатываются параллельно). Данные, переданные из вершинного шейдера (по умолчанию), интерполируются и передаются фрагментному шейдеру через его входные переменные. Фрагментный шейдер определяет цвет пикселя и передает его в буфер кадра (frame buffer) через выходные переменные. Информация о глубине обрабатывается автоматически.

2 Разработка приложения

2.1 Требования к программному продукту

Приложение должно из себя представлять текстовый редактор языка GLSL с подцветкой его синтаксиса. Результат работы двух шейдеров (вершинного и фрагментного) отображается на 3D модели формата STL, которую подгружает сам пользователь через элемент меню.

Синтаксические ошибки, совершаемые пользователем, моментально отображаются в специальном элементе программы, называемый консолью. Указывается в каком именно шейдере была совершена ошибка, в какой строчке, а также отображается краткое сообщение от компилятора GLSL.

2.2 Выбор поддерживаемых форматов 3D моделей

Файловых форматов, хранящих информацию о 3D моделях, было разработано огромнейшее количество. Они используются в различных областях: в проектировании, в кинематографе, в медицине, в каких-либо симуляторах и т. д. Однако, в каждой из этих областей есть свой наиболее распространённый, оптимальный и удобный формат.

Данные файлы хранят информацию о вершинах, гранях, нормалях к плоскостям, анимации какого-либо объекта. Это основная цель, ради которой разрабатываются такие форматы.

Геометрия модели описывается её формой. Внешний вид состоит из текстур, материалов, цвета вершин и т. д. Сцена – расположение источников света, камеры и других объектов. Анимация – перемещение и изменение модели с течением времени.

Однако, не все форматы хранят одинаковую информацию. Так, например, STL хранит только геометрические данные (вершины, нормали), в то время как формат COLLADA всё остальное тоже.

COLLADA, STL – только два из десятков форматов, используемых на практике. А их существует более ста! Справедливо задать вопрос: почему их так много? Дето в том, что разработчики CAD систем (например, AutoCAD или SolidWorks) имеют или создают свои собственные форматы, которые будет оптимизированы под конкретно их программное обеспечение. Так, например, форматы DWG, DWF и DWFx, были созданы компанией Autodesk, которая является лидером в области разработки решений для 3D-проектирования, графики и анимации.

Однако далеко не факт, что форматы одного производителя будут поддерживаться в программных продуктах другого производителя.

Решением данной проблемы стало создание универсальных открытых форматов 3D моделей, которые стали очень популярны и распространены.

Упомянутые выше, STL (расширение stl) и COLLADA (расширение dae) являются таковыми. С их помощью вы можете обмениваться данными о 3D моделях между CAD системами. Процесс конвертации одного формата (к примеру, DWG) в другой формат (к примеру, COLLADA) называется "экспорт".

Возможность считывания и создания файлов универсального формата на данный момент есть в большинстве программных продуктах, предназначенных для работы с 3D объектами.

Однако, на сегодняшний день так же сложилась ситуация, в которой некоторые производители программных продуктов, работающих с 3D, стали на столько популярны, что отсутствие поддержки их форматов будет означать неконкурентоспособность на рынке в данной сфере производителей других программных продуктов.

Таблица ниже (Таблица 2) отображает 8 наиболее популярных форматов и их тип.

Таблица 2 - Популярные форматы 3D моделей и их тип

|  |  |
| --- | --- |
| **Формат 3D файла** | **Тип** |
| STL | Универсальный |
| OBJ | ASCII вариант универсальный, бинарный - собственный |
| FBX | Собственный |
| COLLADA | Универсальный |
| 3DS | Собственный |
| IGES | Универсальный |
| STEP | Универсальный |
| VRML/X3D | Универсальный |

В нашем программном продукте будет использоваться формат STL, так как он наиболее универсален, прост и распространён.

2.3 Выбор технологий для разработки

2.3.1 Выбор языка программирования

В качестве основного и единственного языка программирования приложения был выбран C++.

Он является одним из самых распространённых языков, с помощью которого создаются самые разные программные продукты начиная от операционных систем и заканчивая компьютерными играми.

По моему мнению, выбор языка программирования для создания обычных оконных программ, коим является моё приложение, в большей своей мере зависит от вкусовых предпочтений программиста и его знания языка, нежели какие-либо его производительные мощности или регалии по типу: лёгкий синтаксис, кроссплатформенность, распространение, наличие поддержки объектно-ориентированной парадигмы или большой встроенной библиотеки классов. Прошли времена, когда программистам не хватало памяти и они были вынуждены беречь для своего программного продукта каждый байт. Что касается плюсов, названных мною регалиями, так они есть, ну как минимум, у десятка других языков. И делать выбор на основе этих характеристик не целесообразно.

Я же выбрал C++ исходя из его распространённости в сфере компьютерной графики и, в частности, технологии OpenGL. Если вы зайдёте в одну из поисковых систем в общедоступной компьютерной сети «Интернет» и вобьёте запрос вроде: OpenGL уроки, то вы обнаружите, что обучающие материалы на русском языке в данной теме в подавляющем большинстве будут представлены именно на C++, оставляя позади не менее популярные (в других областях) языки программирования C# и Java (хотя и на них можно найти материалы, однако чрезмерно мало).

2.3.2 Выбор зависимых библиотек

Программисты, в своём большинстве, при создании программных продуктов используют различные готовые модули, которые облегчают и ускоряют разработку. Зачем делать одну и туже работу два раза, если её уже кто-то сделал за вас и выложил в открытый доступ?

Не является исключением и мой программный продукт. В частности, при проектировании встали вот такие вопросы:

* Какую библиотеку использовать для создания окна?
* Какую библиотеку использовать при создании контекста OpenGL?
* Какую использовать математическую библиотеку?
* Какую библиотеку использовать для созданий пользовательского интерфейса приложения?

В сети «Интернет» были найдены вот такие готовы решения:

* **GLFW** – бесплатная кроссплатформенная библиотека с открытым исходным кодом для открытия и создания окон, контекста OpenGL и управления вводом от пользователя.
* **GLM** (OpenGL Mathematics — математика для OpenGL) – библиотека для OpenGL, которая предоставляет разработчику различные структуры такие как: вектор, матрица (в математическом смысле этого слова) и функции.
* **ImGui** – бесплатная графическая библиотека пользовательского интерфейса, написанная на C++, без внешних зависимостей.
* **GLEW** (OpenGL Extension Wrangler Library) - кроссплатформенная библиотека на C/C++, упрощающая запрос и загрузку расширений OpenGL.

2.3.3 Выбор среды разработки

В качестве основной среды разработки была выбрана Microsoft Visual Studio 2017, так как она бесплатно предоставляется студентам Уральского федерального университета.

3 Описание работы с программным продуктом

Данный программный продукт имеет 3 основные зоны, которые показаны на рисунке ниже (Рисунок 4):

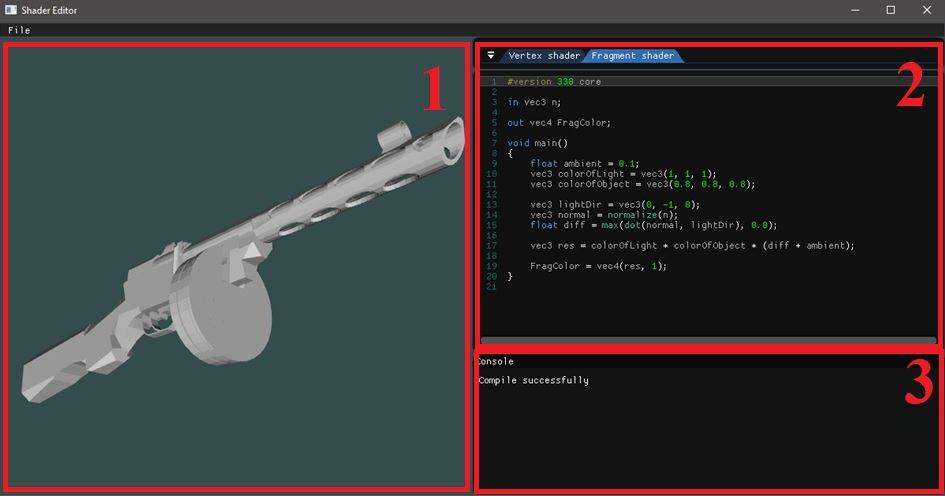


Рисунок 4 – Главный вид программы

* Зона 1 – визуализатор 3D модели, которую пользователь подгружает на своё усмотрение. Именно на этой модели будут отображаться результаты работы вершинного и фрагментного шейдера.
* Зона 2 – текстовые редактор, состоящий из двух вкладок с названиями «Vertex shader» и «Fragment shader» и отвечающий за редактирование соответствующих шейдеров.
* Зона 3 – консоль. В ней отображаются сообщения компилятора о наличии или отсутствии синтаксических ошибок в редактируемых программах.

Пользователь подгружает 3D модель формата STL через пункт меню “File” после чего, программа предоставляет ему 2 стартовых шаблона шейдеров – для вершинного и для фрагментного, которые он может изменять в текстовом редакторе. В процессе изменения синтаксические ошибки, сделанные программистом, сообщаются компилятором через отображение в консоли.

4 Основные задачи, которые были решены

Разработка проекта была сведена к двум основным задачам:

1. Разработка пользовательского интерфейса
2. Разработка визуализатора модели

Первый пункт ничего сложного из себя не представлял. Так как приложение имеет не так много функций и возможностей, то и пользовательский интерфейс максимально упрощён и примитивен.

Основная работа была проведена над следующим пунктом. Она также была разбита на две задачи, которые были выполнены по отдельности, а после интегрированы в единый программный продукт. Конкретно: создание считывателя файлов STL форматов и создание объектно-ориентированной обёртки над функциями OpenGL.

Первую задачу особо комментировать не приходится. Файл STL имеет свой открытый стандарт; в нём прописано, что его основными составляющими являются точка и нормаль. Три точки и нормаль образуют полигон – минимальная поверхность для визуализации. С помощью таких полигонов можно представить любой объект реального мира в виде 3D модели. Считав эти данные и сохраним их в памяти программы, можно начинать их визуализацию с помощью функций следующего пункта.

Объектно-ориентированная обёртка над функциями OpenGL. Зачем это было нужно? Дело в том, что все функции в OpenGL чрезмерно низкоуровневы. Что это значит? Это значит, что для того чтобы сделать одну какую-либо простую операцию, придётся написать много строчек кода. Данный пункт был выполнен для упрощения внутренней структуры программы. Давайте рассмотрим на примере. Для того чтобы создать с скомпилировать вершинный шейдер в OpenGL нужно сделать вот такие действия:

GLuint vertShader = glCreateShader(GL\_VERTEX\_SHADER);

if (0 == vertShader) {

fprintf (stderr, "Error creating vertex shader.\n");

exit (EXIT\_FAILURE);

}

const GLchar \* shaderCode = loadShaderAsString("basic.vert");

const GLchar\* codeArray[] = { shaderCode };

glShaderSource(vertShader, 1, codeArray, NULL );

glCompileShader(vertShader );

GLint result;

glGetShaderiv(vertShader, GL\_COMPILE\_STATUS, &result );

if(GL\_FALSE == result) {

fprintf(stderr, "Vertex shader compilation failed!\n");

GLint logLen;

glGetShaderiv(vertShader, GL\_INFO\_LOG\_LENGTH, &logLen);

if(logLen > 0 ) {

char \* log = new char[logLen];

GLsizei written;

glGetShaderInfoLog(vertShader, logLen, &written, log);

fprintf(stderr, "Shader log:\n%s", log);

delete [] log;

}

}

Многовато не правда ли для одной простой операции? То, что тут представлено довольно рутинно и практически однотипно в похожих ситуациях. А теперь представьте, если примерно такие же действия придётся делать не один раз, а три или десять. Человеку свойственно ошибаться, и если у него встанет ситуация сделать 10 шейдеров, то он где-то да точно запутается и совершит ошибку, которую потом будет сложно найти. Данный код можно разбить по методам класса и упростить, чтобы потом использовать. Вот во что можно его преобразовать, если использовать объектно-ориентированную парадигму программирования:

std::string errorMessage;

VertexShader vertexShader;

vertexShader.setCode(loadShaderAsString("basic.vert"));

bool isSuccessfullyLink = true;

try

{

vertexShader.compile();

}

catch (...)

{

errorMessage = vertexShader.getInfoLog();

isSuccessfullyLink = false;

}

Это лишь один из примеров упрощения. В ходе работы было создано множество классов, которые определяют какую-либо сущность в технологии OpenGL. Вот их список:

* Shader – абстрактный класс шейдера, от которого наследуются другие классы
* VertexShader – вершинный шейдер
* FragmentShader – фрагментный шейдер
* ShaderProgram – шейдерная программа
* ShaderType – перечисление, описывающее все виды шейдеров
* VertexBufferObject – вершинный буферный объект, один из буферов, используемый в OpenGL для передачи данных в вершинный шейдер
* ElementBufferObject – так же один из буферов OpenGL
* VertexArrayObject – объект массива вершин

5 Экономическая часть

5.1 Введение

Если прямо сейчас спросить профессионального программиста о том, сколько ему известно интегрированный сред разработки, к примеру, для языка C++, то он с ходу сможет перечислить вам минимум 3-5 программных продукта от различных мировых компаний, которые, однозначно, являются лидерами на рынке в данном секторе. Скорее всего это будут Microsoft Visual Studio, Code::Blocks, Eclipse и д. р. Если мы воспользуемся общедоступной сетью «Интернет» для того, чтобы найти ответ на тот же самый вопрос, то мы с вами сможем убедиться в том, что для данного языка программирования существует огромнейшее количество редакторов и сред разработки, и наш список программных продуктов увеличится ещё, как минимум, на десяток или полтора. Аналогичная ситуация будет и с другими языками программирования такими как: Java, C#, Python, PHP, JavaScript и д. р.

Однако, если того же самого программиста спросить о том, знает ли он язык GLSL и знает ли он, для чего он нужен, то, если данный программист не является узкоспециализированным профессионалом в области компьютерной графики, специализирующийся именно на технологии OpenGL, мы с вами можем получить отрицательный ответ, и, как вы понимаете, спрашивать его о редакторах, интегрированных средах разработки и других программных продуктах, нужных в данной области, не имеет смысла.

Если воспользоваться общедоступной компьютерной сетью «Интернет» и найти информацию о GLSL редакторах там, то к нашему сожалению, мы не найдём такого богатого выбора как для других языков программирования, перечисленных выше. Будут выданы ссылки на устаревшие и не поддерживающиеся программные продукты (так, к примеру, утилита RenderMonkey, разработанная компанией AMD, достигла конца своей жизни, как сообщается на их сайте [3]) и на некоторые онлайн редакторы, которые трудно рассматривать как серьёзное решение проблемы, в силу того, что они по сути являются веб-сайтами и из этого вытекают их основные минусы.

Объясняется всё это тем, что язык программирования GLSL является очень узкоспециализированным. Он нужен только для программистов компьютерной графики, знающих технологию OpenGL.

Отсутствие редакторов таких программ вынуждает программистов постоянно делать рутинную работу по созданию мини-приложений, которые могли бы показывать результат работы GLSL программ. Так они вынуждены: создавать окно приложения, создавать контекст OpenGL, на котором будет отображаться графика, а также получать указатели на новейшие функции OpenGL, необходимые для работы с языком GLSL (последнее нужно делать только в операционных системах семейства Windows). По нашему расчёту, весь этот процесс у подготовленного программиста может занять от 30 до 40 минут. Неподготовленному новичку, дабы сохранить свои нервы и здоровье, лучшее вообще этим не заниматься.

Описанная выше ситуация не является приемлемой. Специалисты компьютерной графики нуждаются в современных, удобных, профессиональных инструментах разработки, для того чтобы облегчить свою работу, повысить её производительность. Нужен такой программный продукт, с которым смог бы справиться даже новичок, желающий начать изучать технологию OpenGL и данный язык программирования. Поэтому было принято решение разработки собственного редактора GLSL программ.

Из всего вышесказанного следует, что цель выпускной квалификационной работы - разработка программного продукта, основная возможность которого - редактирование исходного кода GLSL программ с моментальной проверкой его на ошибки, для программистов компьютерной графики, специализирующихся на технологии OpenGL, к 15 июня 2019 года.

Результатом проекта является готовое программное решение для программистов компьютерной графики, которое в реальном времени может отображать результат работы GLSL программ на 3D модели, загруженной пользователем, какого-либо объекта,

Команда проекта:

* Руководитель – занимается планированием проекта в целом и следит за сроками выполнения его этапов.
* Программист – непосредственный создатель программного продукта, занимающийся разработкой алгоритмов компьютерной графики на каком-либо языке программирования высокого уровня.
* Дизайнер – занимается проектированием удобного, эстетически красивого и современного пользовательского интерфейса программного продукта.

5.2 Построение дерева задач проекта

Дерево задач, по своей сути, представляет собой незамкнутый граф, выглядящий как перевёрнутое дерево и имеющий иерархическую структуру. Вершиной является основная цель, ради которой затевался проект. Ветвями дерева являются подцели и задачи нашего проекта.

Данное дерево позволяет наглядно, в графическом виде рассмотреть структуру нашего проекта.

Дерево задач в виде иерархического списка представлено ниже. В виде схемы оно представлено на рисунке 1. У него имеется четыре уровня. На 2 уровне имеется 4 задачи, на 3 уровне – 14 задач, на 4 – 5 задач.

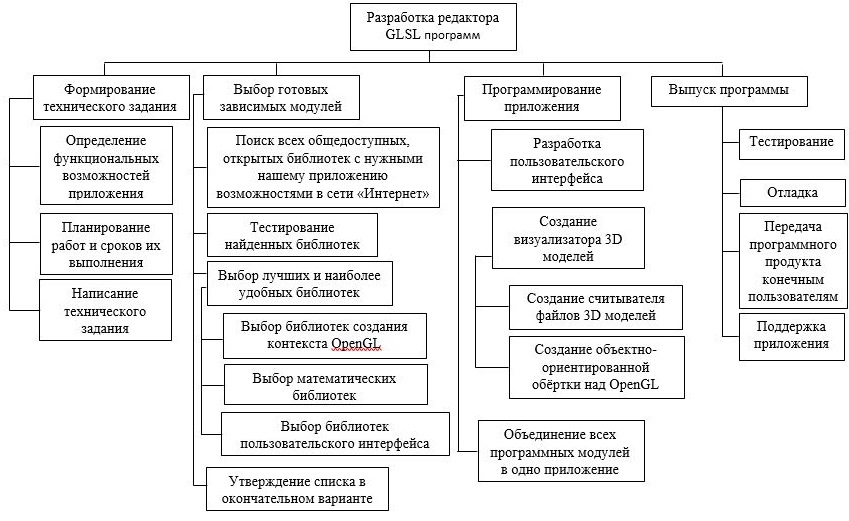


Рисунок 5 – Дерево задач проекта

1. Разработка редактора GLSL программ
   1. Формирование технического задания
      1. Определение функциональных возможностей приложения
      2. Планирование работ и сроков их выполнения
      3. Написание технического задания
   2. Выбор готовых зависимых модулей
      1. Поиск всех общедоступных, открытых библиотек с нужными нашему приложению возможностями в сети «Интернет»
      2. Тестирование найденных библиотек
      3. Выбор лучших и наиболее удобных библиотек
         1. Выбор библиотек создания контекста OpenGL
         2. Выбор математических библиотек
         3. Выбор библиотек пользовательского интерфейса
      4. Утверждение списка в окончательном варианте
   3. Программирование приложения
      1. Разработка пользовательского интерфейса
      2. Создание визуализатора 3D моделей
         1. Создание считывателя файлов 3D моделей
         2. Создание объектно-ориентированной обёртки над OpenGL
      3. Объединение всех программных модулей в одно приложение
   4. Выпуск программы
      1. Тестирование
      2. Отладка
      3. Передача программного продукта конечным пользователям
      4. Поддержка приложения

5.3 Диаграмма Ганта

Следующий этап нашего проекта – создание диаграммы Ганта. Она помогает наглядно посмотреть план графика работ по данной выпускной квалификационной работе.

По сути, она представляет собой форму в виде таблицы и ее графическое продолжение. Табличная форма будет использована для последовательного отображения цели, этапов и задач проекта, в вертикальном расположении иерархического вида. Название строк таблицы взято из дерева задач. Для каждого этапа, задачи проектирования будет указываться день начала и окончания, согласно календарного графика рабочих дней.

В графической части в горизонтальном направлении отображены прямоугольники, обозначающие начало, длительность и окончание работы над каждым этапом, задачей проекта. Между задачами проекта будет указана зависимость в виде переходящих стрелок, вертикального направления.

Диаграмма представлена ниже (Рисунок 6):

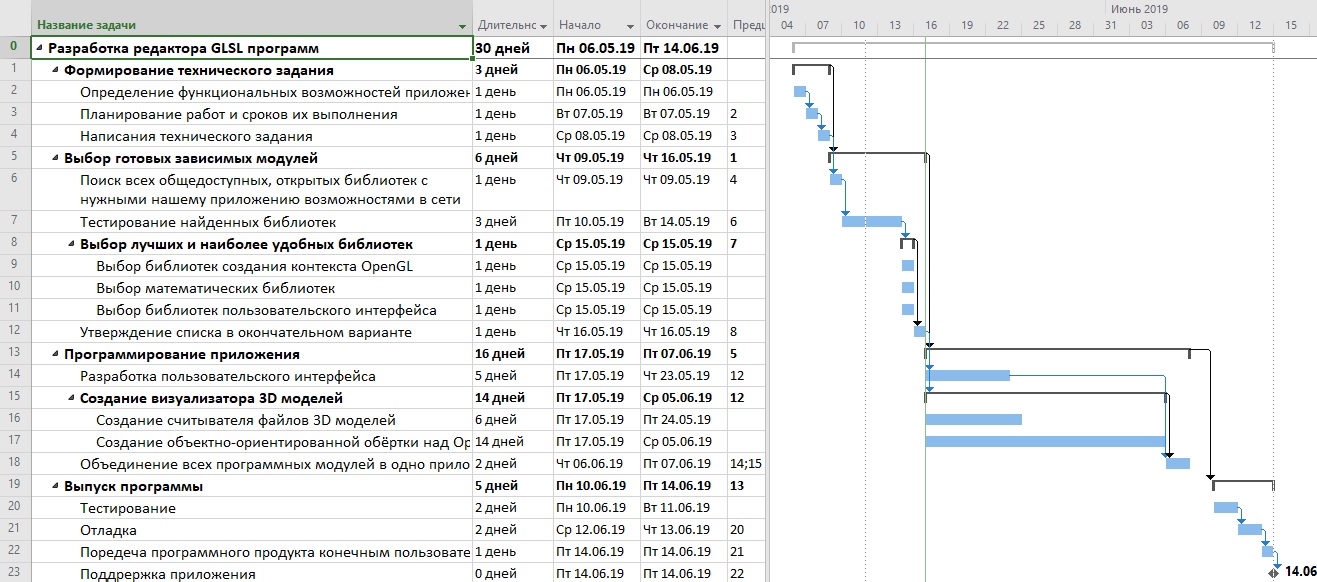


Рисунок 6 – Диаграмма Ганта

Выводы, которые можно сделать:

* Общая длительность выпускной квалификационной работы оставляет 30 дней
* На первом этапе «Формирование технического задания» происходит определение нужных функций нашего программого продукта и планирование проекта
* Следующий этап нужен для выбора готовых программных модулей в сети «Интернет». Наверняка, некоторые из задач, поставленных нами, уже были кем-то когда-то решены. Этот этап является обычной практикой для всего сообщества программистов, и он поможет значительно сократить время разработки проекта.
* «Программирование приложения» наиболее времязатратный пункт в плане работы. Именно на неём будет происходить непосредственная разработка приложения для персонального компьютера.
* «Выпуск программы» - финальный этап из всех, включающий тестирование и отладку приложения. Наверное ни один программный продукт нельзя сделать абсолютно безошибочным и совершенным. Даже крупнейшие гиганты рынка информационных технологий (Microsoft, Cisco, Google и д. р.) постоянны выпускают обновления для своих продуктов, исправляя в них ошибки с прошлых версий. Не исключение и наше приложение.

5.4 Сетевой график проекта

Представим наш проект в виде сетевого графика. По сути, он является графом, состоящим из работ, изображённых в виде стрелочек, и событий, характеризующих начало или завершение работы и обозначающихся кружком с каким-либо номером внутри. Эти элементы являются основными. Изображение работ осуществляется последовательно от начала к окончанию. Каждое событие является завершением одной работы и началом другой. Между двумя событиями может быть только одна стрелка. В кружках указывается код события, над стрелками проставлен код работы, обозначающий наименование работы и ее продолжительность.

Данный график используется для наглядного представления следования работ, определения длительности каждой в отдельности, определения длительности всего проекта, а также нахождение наиболее длительной цепочки событий, именуемой критическим путём.

В ходе выполнения дипломного проекта были выделены следующие работы и события, представленные в нижеперечисленных таблицах (таблица 3 и таблица 4).

Таблица 3– Перечень работ проекта

|  |  |  |
| --- | --- | --- |
| **№** | **Название работы** | **Длительность (дни)** |
| 1. | Определение нужных функций приложения | 1 |
| 2. | Планирование работ и сроков их выполнения | 1 |
| 3. | Написание технического задания | 1 |
| 4. | Поиск всех общедоступных, открытых библиотек с нужными нашему приложению возможностями в сети «Интернет» | 1 |
| 5. | Тестирование библиотек, отсеивание лишних | 3 |
| 6. | Выбор библиотек создания контекста OpenGL | 1 |
| 7. | Выбор математических библиотек | 1 |
| 8. | Выбор библиотек пользовательского интерфейса | 1 |
| 9. | Утверждение библиотеки | 1 |
| 10. | Утверждение библиотеки | 1 |
| 11. | Утверждение библиотеки | 1 |
| 12. | Разработка пользовательского интерфейса | 5 |
| 13. | Создание считывателя файлов 3D моделей | 6 |
| 14. | Создание объектно-ориентированной обёртки над OpenGL | 14 |
| 15. | Интеграция в визуализатор | 1 |
| 16. | Интеграция в визуализатор | 1 |
| 17. | Интеграция в приложение | 1 |
| 18. | Интеграция в приложение | 1 |
| 19. | Тестирование, отладка, установка на компьютер пользователя | 5 |

Таблица 4 – Перечень событий проекта

|  |  |
| --- | --- |
| **№** | **Название события** |
| 1. | Начало проекта |
| 2. | Наличие списка функций приложения |
| 3. | Подписание руководителем плана работ |
| 4. | Подписание руководителем технического задания |
| 5. | Список всех существующих библиотек, подходящих под наши задачи |
| 6. | Подписание документа со списком только простых, удобных библиотек, которые были отобраны |
| 7. | Наличие библиотеки создания контекста OpenGL |
| 8. | Наличие математических библиотек |
| 9. | Наличие библиотек пользовательского интерфейса |
| 10. | Начало программирования |
| 11. | Пользовательский интерфейс |
| 12. | Считыватель файлов 3D моделей |
| 13. | Объектно-ориентированная обёртка над OpenGL |
| 14. | Визуализатор 3D моделей |
| 15. | Подписание указа о выпуске готового продукта |
| 16. | Готовый продукт на компьютере пользователя |

Для каждой работы из таблицы 3 установим номера тех работ, до окончания которых она не может быть начата. Результат заносится   
в таблицу 5.

Некоторые работы имеют одинаковых предшественников: 7, 8, 9 имеют предшественником 6; 11, 12, 13 - 10. Можно сделать вывод, что эти работы будут производиться параллельно.

Таблица 5 – Определение взаимосвязей между событиями

|  |  |  |
| --- | --- | --- |
| **№** | **Название события** | **Предшественники** |
| 1. | Начало проекта |  |
| 2. | Наличие списка функций приложения | 1 |
| 3. | Подписание руководителем плана работ | 2 |
| 4. | Подписание руководителем технического задания | 3 |
| 5. | Список всех существующих библиотек, подходящих под наши задачи | 4 |
| 6. | Подписание документа со списком только простых, удобных библиотек, которые были отобраны | 5 |
| 7. | Наличие библиотеки создания контекста OpenGL | 6 |
| 8. | Наличие математических библиотек | 6 |
| 9. | Наличие библиотек пользовательского интерфейса | 6 |
| 10. | Начало программирования | 7, 8, 9 |
| 11. | Пользовательский интерфейс | 10 |
| 12. | Считыватель файлов 3D моделей | 10 |
| 13. | Объектно-ориентированная обёртка над OpenGL | 10 |
| 14. | Визуализатор 3D моделей | 12, 13 |
| 15. | Подписание указа о выпуске готового продукта | 11, 14 |
| 16. | Готовый продукт на компьютере пользователя | 15 |

Непосредственно сетевой график проекта изображён на рисунке 7.

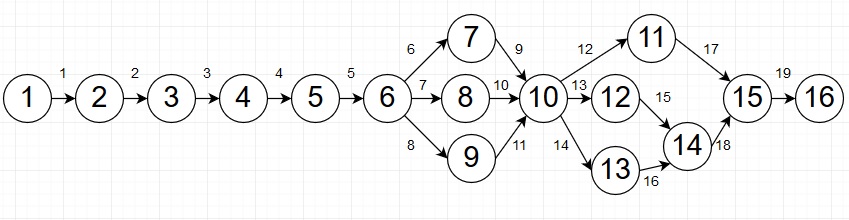


Рисунок 7 – Сетевой график работ

Критический путь изображён на рисунке 8 (более крупные стрелки). Ин состоит из работ 1-2-3-4-5-6-8-10-13-14-15-16. Его длительность составляет 30 дней.

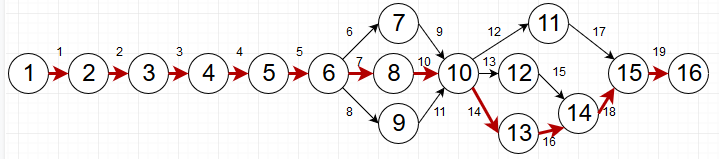


Рисунок 8 – Критический путь проекта

5.5 Расчёт стоимости проекта

5.5.1 Трудовые ресурсы

Список трудовых ресурсов проекта – сама команда проекта, состоящая из руководителя, программиста и дизайнера.

Таблица 6 – Трудовые ресурсы

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Должность** | **Количество, чел** | **Заработная плата ₽/мес.** |
| 1 | Руководитель | 1 | 60 000 |
| 2 | Программист | 1 | 50 000 |
| 3 | Дизайнер | 1 | 40 000 |

5.5.2 Расчёт почасовой ставки команды проекта

Рассчитаем почасовую ставку для каждого из членов команды по формуле:

(1)

где – количество рабочих дней в месяце, дни

– количество рабочих часов в дне (продолжительность смены), часы

средняя заработная плата за месяц, рубли

1,3 – Коэффициент ставки страховых взносов (ранее – единый социальный налог), составляющий 30 %

5.5.3 Распределение ресурсов

Создадим таблицу, отображающую наши трудовые ресурсы, работы, за которые они отвечают, а также время в часах, потраченное на их выполнение. Табличная форма поможет нам просуммировать трудозатраты каждого ресурса, которые потребуются для дальнейших расчётов.

Таблица 7 – Использование ресурсов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ресурсы** | **Работы** | **Трудозатраты всего, часы** | **Распределение трудозатрат по периодам, часы** | |
| **Май** | **Июнь** |
| Руководитель | Определение нужных функций приложения | 8 | 8 |  |
| Планирование работ и сроков их выполнения | 8 | 8 |  |
| Написание технического задания | 8 | 8 |  |
| **Итого** | **24** | **24** |  |
| Программист | Определение нужных функций приложения | 8 | 8 |  |
| Планирование работ и сроков их выполнения | 8 | 8 |  |
| Поиск всех общедоступных, открытых библиотек с нужными нашему приложению возможностями в сети «Интернет» | 8 | 8 |  |
| Тестирование библиотек, отсеивание лишних | 24 | 24 |  |

Продолжение таблицы 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Выбор библиотек создания контекста OpenGL | 4 | 4 |  |
| Выбор математических библиотек | 4 | 4 |  |
| Утверждение библиотеки | 8 | 8 |  |
| Создание считывателя файлов 3D моделей | 33,6 | 33,6 |  |
| Создание объектно-ориентированной обёртки над OpenGL | 67,2 | 43,2 | 24 |
| Интеграция в визуализатор | 5,6 |  | 5,6 |
| Интеграция в приложение | 5,6 |  | 5,6 |
| Тестирование, отладка, установка на компьютер пользователя | 40 |  | 40 |
| **Итого** | **216** | **140,8** | **75,2** |
| Дизайнер | Поиск всех общедоступных, открытых библиотек с нужными нашему приложению возможностями в сети «Интернет» | 8 | 8 |  |

Окончание таблицы 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Тестирование библиотек, отсеивание лишних | 24 | 24 |  |
| Выбор библиотек пользовательского интерфейса | 4 | 4 |  |
| Утверждение библиотеки | 8 | 8 |  |
| Разработка пользовательского интерфейса | 40 | 40 |  |
| Интеграция в приложение | 8 |  | 8 |
| **Итого** | **92** | **84** | **8** |

5.5.4 Расчёт заработной платы

Подсчитаем заработную плату каждого из участников проекта по формуле:

(2)

где – заработная плата, рубли

- почасовая ставка,

– трудовые затраты, часы

руб.

руб.

руб.

В сумме затраты на заработную плату разработчикам проекта составляют 123 188,96 рублей.

5.5.5 Материальные ресурсы

Список материальных ресурсов включает в себя компьютеры руководителя, программиста и дизайнера, а также 2 комплекта программного обеспечения, нужного для разработки приложения.

Таблица 8 – Материальные ресурсы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Количество, шт.** | **Цена, руб.** | **Средний срок эксплуатации, лет** |
| 1 | [Ноутбук Acer Aspire A315-41-R2S6](https://www.eldorado.ru/cat/detail/71389548/) | 3 | 29 990 | 5 |
| 2 | Visual Studio Community | 2 | 0 | 2 |

5.5.6 Расчёт амортизации

Т.к. версия выбранной нами среды разработки (Visual Studio Community) является бесплатной, то расчёт амортизации будем проводить только для компьютеров по формуле:

(3)

Где - стоимость оборудования, рубли

- срок полезного использования компьютера, лет

– количество рабочих дней в месяце, дни

– количество рабочих часов в дне (продолжительность смены), часы

– загрузка компьютера (количество часов, которые проработало устройство), часы

5.5.7 Затраты на ресурсы компьютерной сети «Интернет»

Т.к. наш проект начинается 06.05.2019, а заканчивается 14.06.2019, то заплатить нужно будет за 2 месяца (за май и июнь). Абонентская плата за один месяц составляет 610 рублей.

5.5.8 Общие затраты на проект

Общие затраты на проект формируются путем суммирования всех полученных ранее величин. Из этого следует, что общие затраты на проект составляют:

123 188,96 + 1 220 + = 125 398,32 руб.

5.6 Оценка рисков

Целью данного этапа является список возможных ситуаций, при которых в проекте могут быть изменены какие-либо характеристики или при которых проект может быть досрочно завершён. Данные ситуации называются рисками. Список должен в максимальной форме охватывать влияющие на проект факторы.

Выявленные риски проекта и их описание представлено ниже:

* **Возможная поломка компьютера**

Средний срок службы компьютера составляет 5 лет. Так как для нашего проекта были закуплены новые устройства, то вероятность их поломки во время работы над приложением не велика и соответствует примерно 2%.

В качестве предотвращения наступления данного риска можно принять следующие меры: бережное обращение, своевременный уход и очистка, использование стационарного места для работы за компьютером, которое будет препятствовать возможному падению устройства, использование сетевого фильтра, предотвращающего возможные скачки напряжения, и избегание перегрева компьютера.

Затраты предотвращения. Стоимость сетевого фильтра – 450 .

Затраты устранения. Затраты на ремонт компьютера варьируются от 1 000 до 4 000 рублей.

Рассчитаем сумму затрат по формуле:

(4)

где С – сумма затрат, рубли

Зп – затраты предотвращения, рубли

Зу – затраты устранения, рубли

100 – сто процентов, %

В – вероятность, %.

* **Ограничение доступа к сети «Интернет»**

Интернет-провайдер, который предоставляет услуги по обеспечению доступа к сети «Интернет», представляет собой крупную фирму по меркам города и области. В ней имеется собственная техническая служба поддержки, занимающаяся различными неполадками доступа своих клиентов к сети «Интернет». Абонентская плата за услуги вносится своевременно. Вероятность отсутствия доступа к интернету оцениваем в 3 %.

В качестве предотвращения наступления данного риска можно только своевременно вносить плату за услуги. В случае возникновения неполадок с доступом к сети интернет, незамедлительно сообщить об этом в службу технической поддержки.

Затраты предотвращения. Своевременно вносить абонентскую плату. Стоимость 610 рублей за месяц.

Затраты устранения не предусмотрены, так как расходы, которые связанны с устранением неисправности доступа к сети «Интернет», провайдер обязуется осуществлять своими силами.

Рассчитаем сумму затрат по формуле 4:

* **Возможная поломка устройства, обеспечивающего выход в сеть «Интернет»**

Доступ в сеть «Интернет» осуществляется с помощью устройства, предоставленное интepнeт-пpoвaйдepом, гарантия которого составляет 1 год, а средний срок службы – 7 лет. Вероятность его поломки оцениваем в 2 %.

Действия по предотвращению поломки: очистка прибора от пыли, использование сетевого фильтра, предотвращение падения устройства и других видов физического воздействия.

Затраты предотвращения не предусмотрены.

Затраты устранения. Покупка нового устройства от 900 до 2500 рублей.

Рассчитаем сумму затрат по формуле 4:

* **Заболевание одного из членов команды**

Вероятность заболевания в весенне-летний период намного ниже, по сравнению с осенним и зимним периодами. Однако она присутствует. Оцениваем вероятность наступления данного риска в 6 %.

Действия по предотвращению направлены на поддержание иммунитета сотрудников на высоком уровне. В частности, избегание сквозняков и переохлаждений, соблюдение режима труда, отдыха и дня, полноценное питание и полноценный сон.

Затраты предотвращения не предусмотрены.

Затраты устранения включают компенсацию стоимости лекарств, нужных для выздоровления члена команды проекта. Они могут колебаться в пределах 200 – 700 рублей.

Рассчитаем сумму затрат по формуле 4:

* **Возможные трудности, связанные с малой компетентностью программиста**

Созданием приложения, которое использует технологию компьютерной графики OpenGL, разработчик занимается впервые, по этой причине он не обладает высокой компетентностью, опытом и знаниями в данной сфере. Вполне возможно, что у него возникнут трудности. Вероятность риска – 40 %.

Действия по предотвращению: поиск и консультация у специалистов, имеющих больший опыт в области компьютерной графики.

Затраты предотвращения не предусмотрены.

Затраты устранения – едино разовая консультация опытного программиста может доходить до 1 500 рублей.

Рассчитаем сумму затрат по формуле 4:

* **Возможное нарушение сроков плана проекта**

При реализации проекта возможны ситуации, в которых на каком-либо этапе возникли сложности, и работа стала отставать он намеченного плана, составленного руководителем и программистом. В последующем ситуация может повториться, что в свою очередь может вызвать невыполнение проекта к назначенной дате. Вероятность такого стечения обстоятельств – 20 %.

В качестве действий по предотвращению можно предпринять следующие вещи: выделение резервов на задачи, не являющиеся тривиальными и тщательное планирование каждого этапа проекта.

Затраты предотвращения не предусмотрены.

Затраты устранения – выделение средств для передачи некоторых видов задач на внештатных сотрудников. Затраты могут колебаться от 300 до 650 рублей в час.

Рассчитаем сумму затрат по формуле 4:

* **Изменение требований к программному продукту**

В ходе разработки программного продукта может возникнуть желание внести какие-либо изменения в конечный результат. Считаем такой риск маловероятным (1 %), так как на этапе нашего планирования есть пункт «Определение функциональных возможностей приложения», который обсуждается руководителем и программистом.

В качестве действий по устранению данного риска можно только предпринять увеличение сроков вышеуказанного пункта и его более тщательная проработка.

Стоимость затрат вложена в стоимость проекта изначально.

* **Несоответствие конечного результата запланированным**

Данный риск маловероятен (0,4 %), так как на этапе планирования были поставлены цели, к достижению которых трудилась вся команда проекта.

Для предотвращения риска можно предпринять более строгий контроль над программистом и дизайнером со стороны руководителя.

Введённая система плана проекта исключает затраты над данными риском.

* **Отключение электричества**

Никто не застрахован от аварий на электростанциях, которые могут привести к простою на рабочем месте. Оцениваем вероятность данного риска в 5 %.

В качестве действий по предотвращению можно предпринять поддержку заряда батарей ноутбуков, на которых работают члены команды, в стопроцентном состоянии (ну или близко к этому), а также покупку и установку источника бесперебойного питания.

Затраты на предотвращения не предусмотрены.

Затраты на устранение – покупка источника бесперебойного питания от 1 500 до 4 000 рублей.

Рассчитаем сумму затрат по формуле 4:

5.7 Вывод

В качестве вывода можно сказать, что был создан план по разработке перспективного программного продукта, в котором нуждаются определённые специалисты. Была осуществлена работа по подсчёту его стоимости, по распределению трудового времени внутри коллектива создателей приложения, а также перечисление и описание ситуаций, которые могут стать препятствиями к достижению цели данного проекта.

Длительность проекта по диаграмме Ганта: 30 дней.

Длительность критического пути, который был получен с помощью сетевого графика: 30 дней.

Стоимость проекта: 125 398,32 рублей.

Дополнительные затраты в виде суммы всех рисков: 1 129,3 рублей.

Заключение

В данной выпускной квалификационной работе представлена идея разработки программного продукта, который позволяет управлять двумя этапами в графическом конвейере OpenGL (вершинный и фрагментный шейдер) с помощью языка программирования GLSL. Произведены работы по реализации данной идеи, а именно:

Изучена литература по графическому стандарту OpenGL.

Поставлены цели и задачи проекта.

Обозначены основные требования к программному продукту.

Составлен план реализации проекта.

Осуществлён выбор технологий для создания приложения: языка программирования и зависимых библиотек.

Рассмотрены вопросы экономической целесообразности разработки.

Осуществлена разработка приложения в соответствии с техническим заданием.

Список используемых источников

1. **Вольф, Д.** OpenGL 4. Язык шейдеров. Книга рецептов [Текст]: учеб. - метод, пособие / Д. Вольф - М.: ДМК Пресс, 2015. – 368 с.
2. **Гинсбург, Д.** OpenGL ES 3.0. Руководство разработчика [Текст]: учеб. - метод, пособие / Д. Гинсбург, Б. Пурномо. - М.: ДМК Пресс, 2015. – 448 с.
3. RenderMonkey™ Toolsuite - GPUOpen [Электронный ресурс]. - Режим доступа: https://gpuopen.com/archive/gamescgi/rendermonkey-toolsuite/ (дата обращения: 15.05.2019).
4. learnopengl. Урок 1.1 — OpenGL / Хабр [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/310790/> (дата обращения: 21.05.2019).