СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность(Удалить). Шейдерные программы[0] – инструменты, с помощью которых можно добиться невероятных результатов в компьютерной графике. Профессиональные программисты, работающие с графикой, с помощью шейдеров создают всевозможные эффекты: добиваются эффекта фотореалистичности итогового изображения, поражают эффектами из миллиардов частиц, акцентируют внимание замыливанием заднего фона, придают плоскостям эффект объёмности и многое другое. Главной целью написание шейдеров является изменение конвейера рендеринга графики для получения желаемого результата. Современные компьютеры способны в реальном времени рассчитывать 60 раз в секунду динамические картины, близкие к максимальному фотореализму, в высоком разрешении, что ещё 20 лет назад было невозможно, и всё это благодаря шейдерам. Получив возможность менять графический конвейер, программисты смогли увеличить производительность систем построения изображений, перекладывая всё больше вычислений на графические ускорители. Ранее такой подход был невозможен, так как порядок построения изображения и функциональные возможности были строго ограничены и не могли быть изменены, из-за чего приходилось выполнять некоторые расчёты, такие как освещение, на центральном процессоре. Появление возможности писать шейдеры и встраивать их в графический конвейер изменило мир компьютерной графики в сторону гибкости и производительности. Сегодня шейдеры являются актуальной темой исследований, так как они являются важной частью любой программы, которая работает с графикой и использует для этого аппаратное ускорение, ведь благодаря им возможно быстро преобразовать геометрию, рассчитать промежуточные данные и вычислить итоговый цвет сразу для нескольких сотен пикселей в итоговом изображении. Инструментом для создания шейдера может быть любой текстовый редактор, однако не у всех есть достаточные знания для написания шейдеров, способных выдать желаемый результат. Некоторые программы стараются предоставить простой интерфейс для настройки заранее подготовленных программ, однако такой имеет ряд ограничений и не всегда оправдывает ожидания.

Главной проблемой при написании шейдеров является то, что сам процесс является нетривиальным. Одной из причин является существование разных популярных графических API, а также существование нескольких языков программирования шейдеров. Это создаёт некоторые проблемы при написании шейдеров для разных графических API, так как каждый API использует свои языки программирования шейдеров. Каким должен быть инструмент разработки шейдеров, который устранит эти недостатки?

Объект и предмет(Удалить). Объектом исследования являются шейдеры, их структура и способы их применения для решения практических задач. Предметом исследования является процесс написания шейдеров, особенности их реализации в разных API и способы упрощения разработки шейдеров.

Цель и задачи(Удалить). Целью дипломной работы является разработка программы для создания шейдеров, с помощью которой можно будет реализовать низкоуровневую и высокоуровневую логику шейдеров, используя для этого только унифицированные графические элементы. Основные задачи, которые должны быть решены для достижения цели:

- анализ исходных данных;
- выбор инструментальных средств для реализации;
- проектирование программного обеспечения;
- разработка графического интерфейса;
- реализация функциональных частей;
- тестирование результатов.

Предполагается, что в результате выполнения дипломной работы будет реализована программа, функциональные и графические особенности которой позволят упростить и ускорить процесс разработки шейдеров для разных API.

Методы исследования(Удалить).

1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ

При разработке программного обеспечения важно ещё в начале пути разработать архитектуру, которая будет одновременно гибкой, масштабируемой и производительной. Разработка такой архитектуры может занять много времени, если подходить к процессу без каких-либо знаний о предметной области и существующих решений. Это значит, что перед началом проектирования программы необходимо собрать и тщательно проанализировать все доступные данные. Полученные результаты необходимо использовать для полного описания предметной области. Проанализировав аналоги и прототипы разрабатываемого программного продукта будут получены данные для подготовки к проектированию с учётом всех плюсов и минусов, выявленных в аналогичных продуктах.

1.1 Описание предметной области

Растеризация – процесс, в результате которого получается растровое изображение. В компьютерной графике растеризация может быть выполнена двумя способами:

— Software – цвет каждой точки результирующего изображения вычисляется на центральном процессоре;

— Hardware acceleration (аппаратное ускорение) – цвет каждой точки вычисляется на отдельном устройстве – графическом ускорители.

С распространение и удешевлением видеокарт аппаратное ускорение стало активно применяться в компьютерных играх для достижения более качественных и реалистичных результатов с максимально возможной скоростью. Для этого часть работы по построению изображения перекладывалась с процессора на видео ускоритель. В конце 2000 года компания Microsoft осуществила релиз нового API – DirectX 8.0. Основным нововведением нового API стало понятие «Шейдерная модель» – новый подход к разработке, при котором программисту давалась возможность самостоятельно определить поведение видео ускорителя на каждом этапе построения конечного изображения. Для получения желаемого результата программисту предлагается написать шейдер, который затем будет выполняться на виде ускорители и обрабатывать данные так, как это было задумано самим программистом.

Шейдер – специальная программа, экземпляры которой выполняются на графическом процессоре параллельно. На сегодняшний день существует несколько видов шейдеров, которые выполняются на разном этапе в графическом конвейере:

— Vertex shaders – шейдер, в котором происходит обработка вершин геометрии;

— Geometry shaders – шейдер, в котором происходит генерация новых примитивов на основе уже существующих вершин;

— Fragment shaders – шейдер, в котором высчитывается итоговый цвет каждого фрагмента в итоговом изображении;

— Tessellation shaders – шейдер, который используется для изменения сетки объекта в зависимости от определённых условий, которые определяются в этом типе шейдеров;

— Compute shaders – используются для вычисления информации любого желаемого типа, например, анимации или особой формы освещения.

Основными языками программирования шейдеров являются GLSL и HLSL. GLSL используется чаще всего совместно с программами, которые для построения изображения используют OpenGL API. HLSL чаще всего используется в программах, использующих DirectX API. Оба языка могут использоваться для написания шейдеров с последующей их компиляцией для использования совместно с Vulkan API.

OpenGL и Vulkan являются кроссплатформенными API, благодаря чему они могут использоваться на любой платформе без необходимости полностью переписывать программный код. Для OpenGL существует несколько разных спецификаций: OpenGL, OpenGL ES, WebGL и другие. Отличия между ними минимальны и основные подходы к разработки очень сильно похожи, однако есть и различия, определяемые спецификой целевой платформы. DirectX является разработкой компании Microsoft, поэтому данный API разрабатывается только для операционной системы Windows.

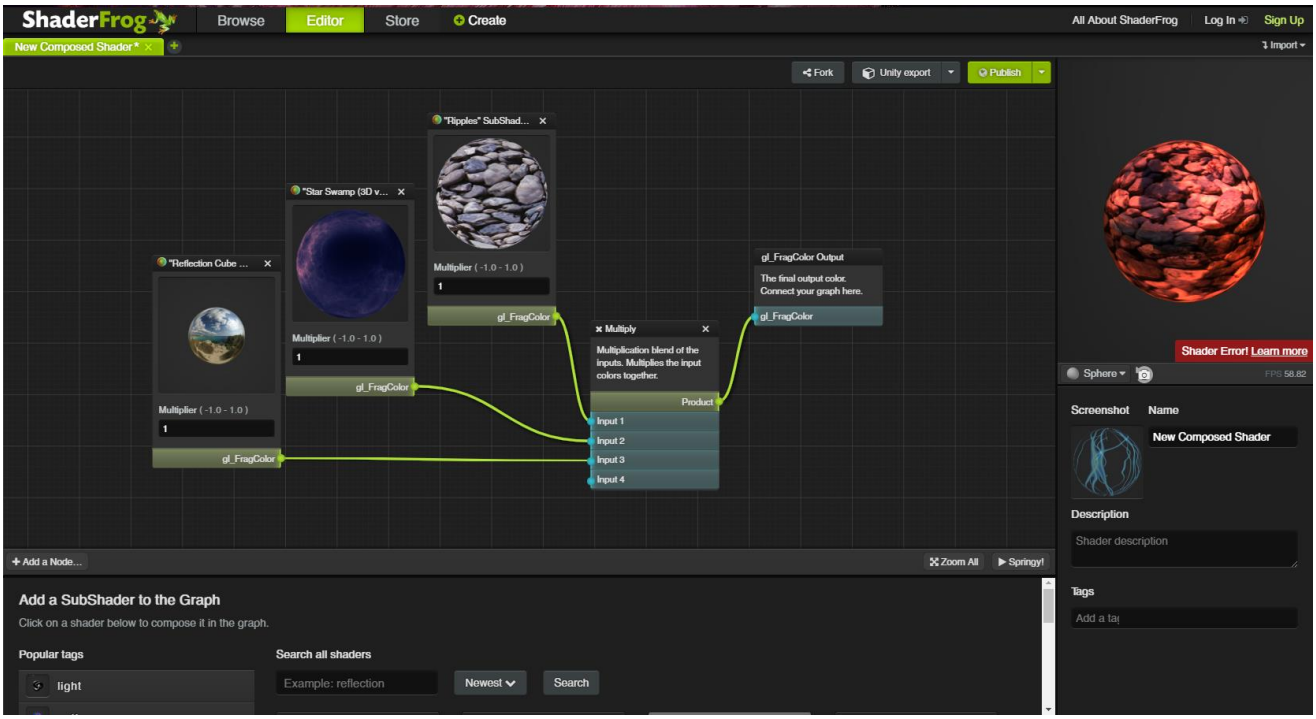
Визуальное программирование – способ создания программ путём манипулирования графическими объектами вместо написания её текста. Преимуществом является визуальная составляющая данного подхода, так как глядя на графические объекты чаще всего проще понять логику программы.

1.2 Анализ аналогов и прототипов

Для более продуктивной и результативной разработки программного продукта необходимо найти и проанализировать аналоги и прототипы проектируемого продукта. Это делается с целью выявления плюсов и минусов существующих решений, что в дальнейшем будет влиять на всю архитектуру и конечные возможности готового программного продукта. В результате поиска инструментов для графического программирования шейдеров были найдены следующие решения: ShaderFrog, GSN Composer, прочие. Прочие программные продукты не рассматриваются, так как в них отсутствует визуальное программирование.

ShaderFrog – WEB приложение для разработки шейдеров на языке программирования шейдеров GLSL. Основным преимуществом является кроссплатформенность, которая достигается за счёт использования браузера в качестве платформы. Присутствует как текстовый, так и графический редакторы, однако минусом является то, что в визуальном

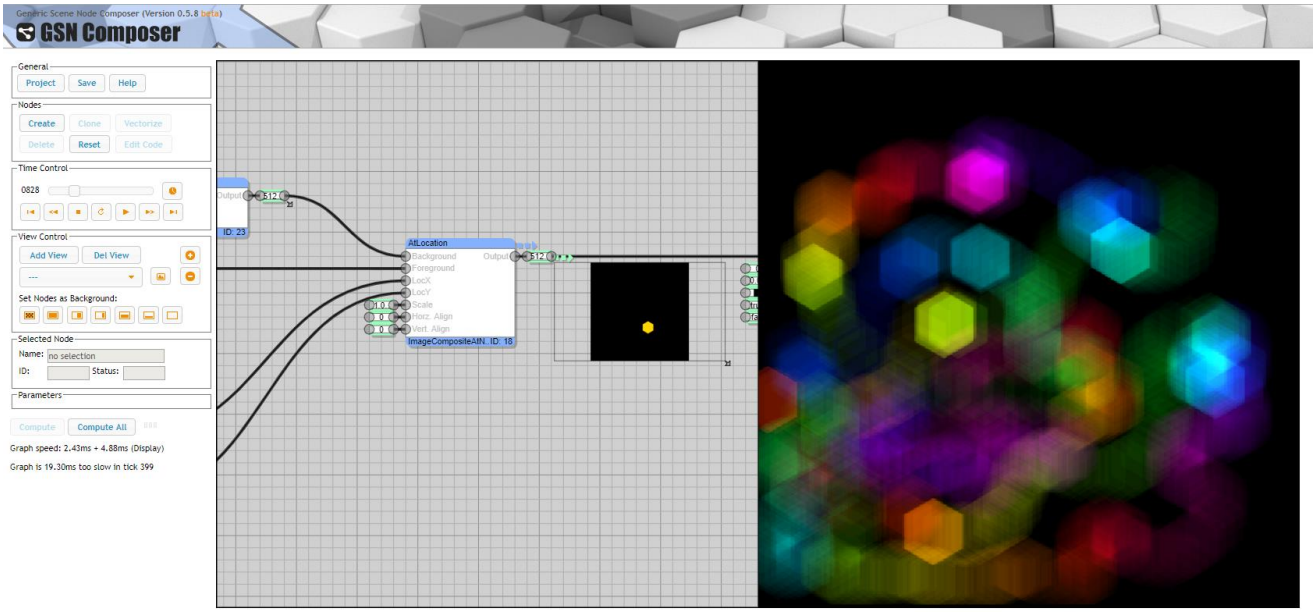
редакторе можно использовать только готовые компоненты, написанные в текстовом редакторе. Это создаёт ряд сложностей для активного творчества неподготовленных пользователей. В качестве дополнительной возможности присутствует возможность экспорта наработок для использования в других системах: Three.js, IOS, Unity, однако последние доступны только обладателям платной подписки.



рис

GSN Composer – так же является WEB приложением, однако, в отличии от ShaderFrog, основной способ разработки шейдеров – визуальное программирование с использованием узлов. Разработчику доступно большое количество узлов, выполняющих разные задачи, которые могут быть использованы для достижения поставленной цели. Присутствует возможность экспорта проекта в виде WEB содержимого для встраивания результата на свои HTML страницы.

Минусом этих решений является то, что они, в основном, ориентированы на генерацию кода для программ, использующих API OpenGL. Так же у них нет полностью бесплатной возможности экспортировать проект в другие среды, что было бы очень удобно в некоторых случаях. Одним из таких случаев может стать разработка шейдера, который будет использоваться как OpenGL API в приложениях для всех систем, так и, например, в приложениях, использующих API DirectX, написанных для операционной системы Windows.



рис

1.3 Постановка задачи проектирования

При проектирование программного продукта важно определить приоритетные задачи, невыполнение которых станет критической ошибкой в проектировании. Так же стоит выбрать средства для раз

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Патамушта так нада!

2.1 Функциональная структура

Собрать DFD и расписать потоки данных

2.2 Варианты использования

Собрать Use case и расписать зависимости между процессами

2.3 Описание динамики функционирования

Собрать State Machine и расписать все состояния

2.4 Описание архитектуры

Архитектура конечного программного продукта должна, в первую очередь должна быть масштабируемой для ускоренного внедрения дополнительных компонентов. Для достижения поставленной цели систему стоит разбить на функциональные модули, которые будут независимы друг от друга.

ТУТ БУДЕТ ДИАГРАММА СТРУКТУРЫ ПРОГРАММА

Описание диграммы активностей…

А ТУТ БУДЕТ ДИАГРАММА АКТИВНОСТЕЙ

2.5 Архитектурное моделирование

Описание основных классов и интерфейсов

Описание интерфейса и классов камеры. Описание интерфейса и классов сцены рендеринга. Описание интерфейса и классов компилятора. Описание классов редактора.

Ссылка на диаграмму классов ()

…

Готовое программное обеспечение, перед началом использования, необходимо установить на персональном компьютере. В качестве операционной системы могут быть выбраны: Windows, Linux, Mac OSX. Компоненты, необходимые для работы системы представлены на рисунке (диаграмма развёртывания).

3 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ

После проектирования программного обеспечения начинается процесс его разработки. Разработка программной составляющей программного обеспечения делится на два этапа: программирование и тестирование. Во время программирования программы реализуются все архитектурные решения, заложенные на этапе проектирования. На этапе тестирования выявляются слабые места в программном коде для их дальнейшего исправления. Так же тесты выполняются каждый раз при внесении изменения в программе для проверки на корректность работы того, что ранее работало и проходило тестирование.

3.1 Реализация программы

Реализация программы начинается с разработки самого главного компонента программы – графического редактора, с помощью которого будет реализовываться все идеи конечного пользователя. Решить поставленную задачу можно двумя способами: реализация функционала с нуля или использование готового решение. Для ускорения разработки было принято решение выбрать второй вариант, внеся в него все необходимые изменения для получения необходимого результата. Отличным кандидатом является библиотека Qt Node Editor, оригинальный исходный код которой можно посмотреть в репозитории её автора. При работе программы появляется необходимость использовать узлы с переменной моделью – структура узла может изменятся во время работы программы в зависимости от текущей ситуации. Для достижения поставленной задачи был внесён ряд изменений в основных классах библиотеки. Такой решение принято в основном из-з отсутствия возможности масштабировать существующую систему для получения желаемого результата из-за некоторых ограничений самой библиотеки.

Изменение 1

Изменение 2

Изменение 3

На основе исходных и изменённых компонентов реализуется набор типов данных и узлов, работающих с ними.

Одним из возможных изменений является возможность дропа объектов внутри сцены. Для этого необходимо активировать и переопределить события дропа объектов в сцене.

Виджет, хранящий набор всех узлов для их вставки в редактор представляет из себя виджет с панелью для скроллинга содержимого и набором подвиджетов внутри. Каждый узел будет представлять из себя виджет, при клике на котором создаётся DragAndDrop объект, предназначенный для перемещения в окно, готовое его принять.

Следующим модулем, который будет реализован, является модуль графических сцен. Каждая графическая сцена должна реализовывать интерфейс сцены. Каждая сцена должна представлять из себя виджет, в котором происходит рендеринг итогового изображения с последующим выводом на экран. Сама же сцена будет управляться виджетом-контейнером, который будет отвечать за создание виджета целевой сцены, а также за управление этой сценой в рамках интерфейса. Первым виджетом сцены будет виджет, который для рендеринга изображения будет использовать OpenGL API. Одно из решений, которое в него заложено – использование динамической последовательности вызова функций. Это достигается за счёт построения вектора из функторов, которые будут вызываться по порядку при каждом рендеринге изображения. Набор функций выбирается в зависимости от конфигурации, которая будет установлена в данном виджите.

Компилятор – модуль, задача которого состоит в том, чтобы получить данные логики из графического редактора и преобразовать их в целевой исходный код шейдера. В рамках этой работы ограничением будет компиляция только Fragment Shader для доступных целевых API – DirectX и OpenGL.

Далее идёт реализация OGL COMPILE логики

3.2 Тестирование программы

Пока ещё нет тестирования