Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

МОДЕЛИРОВАНИЕ ЛАНДШАФТОВ В OPENGL

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Суворова Александра Дмитриевича

Научный руководитель
к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2018

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

API — Application Programming Interface;

SDK — Software Development Kit;

CPU — Central Processing Unit, центральный процессор;

GPU — Graphics Processing Unit, графический процессор;

LWJGL — LightWeight Java Game Library, легковесная библиотека Java для разработки игр;

GLSL — Graphics Library Shader Language, язык для программирования шейдеров;

VBO — Vertex Buffer Object, объект вершинного буфера;

VAO — Vertex Array Object, объект вершинного массива.

ВВЕДЕНИЕ

Разработка видеоигр стала одним из основных направлений развития компьютерной графики. Для следования современным тенденциям необходимо создание более продвинутых технологий для реализации реалистичного изображения. Это требует разработки новых аппаратных и программных возможностей. В связи с тем, что прежние способы получения желаемого результата становятся неактуальными, их приходится перекладывать на новую техническую базу. Поэтому целью данной работы является реализация приложения, моделирующего водную и земную поверхность с использованием современных графических возможностей.

Были поставлены следующие задачи:

1. Разработать приложение на языке Kotlin и Java;
2. В качестве графического движка использовать OpenGL;
3. Использовать специальные API, предоставляющие SDK для работы с OpenGL;
4. Реализовать шейдеры на языке GLSL;
5. Создать программу, моделирующую ландшафт земной поверхности в 3D-пространстве.

В первой главе рассматриваются теоретические основы моделирования земной поверхности и главные концепции генерации структур ландшафта. Вторая глава охватывает тему особенностей работы с библиотекой OpenGL. Третья глава содержит описание реализованного приложения, она включает в себя подробное изложение об используемых технологиях, а также детально описывает основную структуру приложения.

1 Моделирование земной поверхности

На основе, OpenGL — это наиболее известный и широко используемый 2D/3D-графический программный продукт. Под OpenGL на базовом уровне понимают программную спецификацию, определяющую некоторый набор функций, с помощью которых можно управлять графикой и изображениями. Реализация данных спецификаций — задача разработчиков. Преимущественно разработчиками OpenGL библиотек являются производители видеокарт. Данные OpenGL библиотеки призваны эффективно использовать возможности оборудования. При невозможности использования оборудования, должна существовать программная эмуляция. На данный момент эффективные реализации OpenGL существуют для Windows, Unix, Mac OS.

Основным принципом работы OpenGL является получение наборов векторных примитивов, дальнейшей математической обработки полученных данных и построение картинки по результатам обработки. Векторные модификации и растеризация представляют собой конечный детерминированный автомат. Спецификация OpenGL точно определяет, каким должен быть результат выполнения каждой функции и как она должна выполняться. После этого, так как спецификация OpenGL является низкоуровневой, разработчикам, внедряющим её, необходимо описать точные последовательности шагов для обработки и построения результирующих примитивов. Поскольку данная спецификация не дает никаких конкретных сведений о реализации, фактические разработанные версии OpenGL могут иметь разные реализации, если их результаты соответствуют спецификации (и, таким образом, одинаковы для пользователя).

Одной из положительных особенностей OpenGL является возможность добавления разработчиком в библиотеку нового функционала через механизм расширений [1]. Таким образом, разработчик свободно может использовать новые технологии без ожидания их реализации в новых версиях OpenGL. Для этого достаточно просто проверить, поддерживается ли это аппаратной составляющей. Если какое-либо расширение пользуется большим спросом, то оно становится частью следующей версии OpenGL.

OpenGL можно представить как конечный автомат. Он содержит большой набор переменных, которые определяют, как OpenGL должен работать в некоторый момент времени. Под состоянием обычно подразумевают контекст OpenGL. Таким образом, в процессе работы OpenGL его состояние постоянно

меняется, устанавливая некоторые новые параметры и манипулируя буферами, а затем визуализируется на основании текущего контекста.

1.1 Принципы построения 3D изображений

До появления специальных графических ускорителей полную отрисовку кадров картинки выполнял центральный процессор (CPU) [2]. Однако, это являлось слишком нерациональным его использованием, так как для такой задачи больше подходит специально созданный для этого модуль, который смог бы разгрузить CPU. Таким модулем стал графический ускоритель (GPU). Каждая видеокарта проходит определенные этапы конвейера визуализации для построения итогового графического представления.

Основные этапы конвейера:

1. Передача на конвейер входных данных в виде вершин некоторого объекта в пространстве с набором нужных атрибутов;
2. Обработка полученного объекта в соответствии с нужными требованиями;
3. Объединение объектов;
4. Растеризация;
5. Добавление цветовой составляющей;
6. Формирование результата.

Со временем стало понятно, что более качественное изображение требует реализации многих специальных конкретных алгоритмов на аппаратном уровне, которые бы были встроены в GPU. Для решения данного вопроса было придумано добавлять в видеокарты компьютеров эти алгоритмы, требующиеся для разработки, что позволяло избежать выполнения наиболее часто используемых действий для ускорения процесса. Однако быстрый рост их разнообразия привел к усложнению их реализации в следствии их огромного количества. Это привело к появлению специальных программ, которые позволяли собирать части графического процессора в собственные программируемые конвейеры, нужные именно для конкретного случая.

1.2 Генерация ландшафта

Существует несколько основных способов моделирования земной поверхности, каждый из которых требует собственное представление о входных данных. Одним из таких способов является построение карты высот. Карты

высот — это двухмерные карты, используемые для хранения значений высот поверхности. Принцип использования состоит в следующем: входное изображение покрывается некоторой координатной сеткой, значения которой колеблются в заданном диапазоне, а также определено расстояние между смежными парами вершин.

Одним из стандартных алгоритмов генерации ландшафта является шум Перлина. Он генерирует процедурную текстуру псевдо-случайным методом. Состоит из набора единичных векторов, которые расположены в некоторых точках определенного пространства, интерполированных функцией сглаживания. Интерполяция точек требуется для придания более гладкого эффекта поверхности. Самый простой способ интерполяции для данного случая — это линейная интерполяция. В современных графических системах данный подход является уже устаревшим, но из-за своей простоты не теряет актуальности в качестве наглядного эффекта сглаживания. Наиболее хорошие результаты достигаются с помощью следующих вариантов, не подразумевающих изменение локальных координат:

1. Билинейная интерполяция;
2. Кубическая интерполяция;
3. Косинусная интерполяция.

Для формирования конечного изображения для каждого пикселя определяется конкретный номер ячейки, с которой он связан. Это делается для нахождения ведущих диагональных векторов, которые соединяют углы каждой ячейки с данным пикселем.

Таким образом, каждый угол ячейки сетки становится основой для двух векторов — некоторого единичного вектора и вектора направления каждого пикселя. Для каждой пары векторов производится вычисления скалярного произведения, дающего значение высоты каждого из угла сетки. Объединяя для каждой клетки ее угловые значения, определяется результирующая высота пикселя.

Далее для структуризации изображения следует определить нужный уровень шума. Это реализуется с помощью итеративного наложения шумовых функций. Изображение, полученное после составления результирующих пиксельных высот, загружается в функцию повторно нужное количество раз. Каждая добавленная наложенная функция называется октавой. Суть повышения

уровня октав состоит в увеличении частоты генерируемого шума. Каждая последующая функция шума имеет частоту в два раза больше предыдущей.

Итоговое изображение высотной карты, построенной на основе алгоритма Шума Перлина, обычно является монохромным, так как нормализованное значение высоты пикселя будет находиться в диапазоне от 0 до 255.

1.3 Генерация воды

Для моделирования реалистичного движения воды обычно используется система уравнений Навье-Стокса. Для этого необходимо применять ее векторную форму (1). Она позволяет получить уравнения мелкой воды. Для их использования нужно определить две направленных оси и скорости движения воды вдоль них. Так как уравнения не учитывают вертикальный уровень поверхности, можно пренебречь глубиной воды во всех точках и взять их среднее значение.

$$\frac{\partial u}{\partial t} = \underbrace{-(u \cdot \nabla)u}_{advection} - \underbrace{\frac{1}{p}\nabla p}_{pressure} + \underbrace{v\nabla^2 u}_{diffusion} + \underbrace{F_{ext}}_{external forces}, \nabla \cdot u = 0 \tag{1}$$

Главными параметрами уравнения являются:

Уравнение движения, по своей сути, представляют иную форму второго закона Ньютона $\overrightarrow{F} = m \cdot \overrightarrow{a}$

Двумерный вариант уравнений Навье-Стокса (1.3):

$$x : \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial x} = -\frac{1}{r}\frac{\partial p}{\partial x} + u\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial x^2}\right),$$

$$y : \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial x} = -\frac{1}{r}\frac{\partial p}{\partial y} + u\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial x^2}\right),$$

$$divV = \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) = 0$$

Уравнение движения в данном виде имеет две компоненты скорости — по одной для каждого координатного направления. При переходе к безразмерным переменным возникает коэффициент Рейндольдса, сочетающий в себе коэффициенты вязкости и плотности.

Для применения уравнений Навье-Стокса в целях моделирования по-

верхности воды, необходимо внести условие того, что горизонтальное распространение структуры должно быть в некотором роде больше вертикального распространения. Совместное моделирование динамики движения текстур разных направлений является сложной задачей за счет невозможности описания параметров, изменяющиеся с глубиной. Поэтому зачастую раздельное моделирование вертикальной и горизонтальной поверхности воды является более рациональным решением. Для реализации естественных переходов от суши к воде, нужно дополнительно рассчитывать граничные условия на поверхности воды, а также некоторой области вокруг. Схожая ситуация требуется в случаях моделирования воды в закрытых пространствах. Таким образом, используя интегрированные уравнения Навье-Стокса по глубине, можно получить уравнения мелкой воды (1.3)

$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} = 0$$
$$\frac{\partial (hu)}{\partial t} + \frac{\partial (hu^2)}{\partial x} + gh\frac{\partial (h+z)}{\partial x} = 0,$$

где используются следующие обозначения:

1. h — высота поверхности;
2. u — скорость течения воды.

Трудности моделирования с использованием которых связаны с тем, что для описания взаимодействия жидкости на границах с твердой поверхностью, все необходимые вычисления требуется выполнять в условиях практически нулевой глубины жидкости, что приводит к усложнению задачи. Таким образом, для реализации изображения поверхности воды, для получения наиболее приближенных к вышеописанным выкладкам результатов, было принято решение использовать уравнение бегущей волны (2)

$$S = A\sin\left[\omega\left(t - \frac{x}{v}\right) + \varphi\right]. \tag{2}$$

По своей сути оно описывает координатную и временную зависимости скалярных, а также векторных величин. В случае гармонической структуры волн, наилучшим образом расположения координатных осей будет являться случай, когда направление распространения воды будет совпадать с одной из осей. В таком случае, волновые поверхности станут перпендикулярны к вы-

бранной оси, поскольку, колебания точек волны имеют один характер движения, то смещение поверхности в итоге будет зависеть от промежутка времени и выбранной оси.

1.4 Генерация освещения

Для эффективной генерации света можно использовать модель освещения по Фонгу, которая включает в себя следующие компоненты:

1. Отраженное освещение имитирует свет, испускаемый источником, отраженный после попадания в точку наблюдения (камеру);
2. Фоновое освещение имитирует свет, достигший конкретной выбранной точки после отражения от всех объектов освещения, при этом для расчетов не учитывается положение точки наблюдения за объектом;
3. Рассеянное освещение задает свет, рассеивающийся от попадания в выбранную точку. Оно зависит от угла падения света, однако, как и фоновое освещение, не учитывает положение камеры.

Все компоненты суммируются и создают итоговое освещение, однако разные источники света при схожих компонентах освещения дают различный результат. Выделяют следующие виды источников света:

1. Направленный свет является наиболее реалистичной имитацией. Он представляет собой бесконечно далеко удаленный источник света, такой как Солнце. Особенность такого источника в том, что из-за его удаленности, все испускаемые им лучи на поверхности освещаемого объекта, можно считать параллельными. Главным образом характеризуется направлением;
2. Точечный источник света представляет собой свет, распространяющийся во все стороны от некоторой точки. Следовательно, характеризуется такой источник только своим положением в пространстве;
3. Прожекторный источник света, отличающийся от точечного своей сферой влияния на объект освещения. Он характеризуется направлением и углом охвата.

2 Элементы OpenGL для генерации воды и ландшафта

Спецификация OpenGL содержит специальные методы для работы с графикой. Для разных целей используются абсолютно непохожие друг на друга, индивидуальные способы, однако можно выделить основные структуры, которые являются ключевыми для любых программ.

2.1 Шейдеры

Шейдеры — это специальная программа, выполняемая на графических процессорах, которая нужна для отрисовки графики на экране. Они определяют конечный вид изображений, который зависит от различных параметров, например, координат или цветов. Основное преимущество использования шейдеров — возможность создавать алгоритмы самостоятельно, не опираясь на стандартные внутренние возможности каждого графического процессора.

На основе [3], шейдеры бывают трех видов:

1. Вершинный шейдер;
2. Фрагментный шейдер;
3. Геометрический шейдер.

Каждый из видов ответственен за отдельную работу и выполняются на разных стадиях графического конвейера.

Вершинный шейдер обрабатывает информацию, которая сообщает о различных свойствах вершин некоторых многогранников. Одним из основных свойств является координата вершины в пространстве. Более расширенные возможности состоят в различных видах преобразований и генерации вершин, расчете силы и направления освещения, также многие другие. Фрагментный шейдер нужен для обработки пиксельных данных, то есть он работает с текстурами, добавляя изображению различные эффекты на последней стадии визуального конвейера. Геометрический шейдер создан для обработки целого набора вершин. В отличие от вершинного шейдера, он в состоянии порождать вершинные связи, детализируя изображения.

Для создания шейдеров нет единых правил, существует несколько подходов и целый набор различных языков программирования, которые, однако, имеют общую структуру — все они предоставляют определенные типы данных, специальные переменные и константы.

2.2 Трансформация изображений и камера

Чтобы избавиться от статической привязки объектов, нужно трансформировать их. Для этого существует несколько способов [4]:

1. Простейшим способом придать объекту движение является изменение координат всех вершин этого объекта. Главный недостаток этого способа — низкая производительность за счет длительных процессорных преобразований;
2. Наиболее удобным способом трансформировать объект является использование математических способов, а конкретно — использование матриц.

Разные виды матриц нужны для разных преобразований. Сложные комбинированные преобразования позволяют достичь сильных трансформаций. Можно выделить несколько видов матриц и соответствующих им преобразований:

1. Матрица сдвига позволяет двигать объекты;
2. Матрица вращения позволяет описать траекторию вращения объекта, вокруг заданной оси;
3. Матрица масштабирования позволяет производить однородные и неоднородные преобразования;
4. Комбинация матриц предоставляет возможность достичь нескольких эффектов, выполнив всего одно преобразование.

OpenGL предполагает, что все вершины, которые нужно отобразить в конкретно заданном виде после запуска шейдерных преобразований будут находиться в нормализованных координатах. Существует целая цепочка преобразований объектов из одной системы координат в другие.

3 Реализация приложения

В ходе работы было реализовано приложение, моделирующее ландшафт земной и водной поверхностей. В проекте был использован объектно-ориентированный подход, упрощающий представление всех моделей и предоставляющий их интуитивно понятное взаимодействие. Приложение было разделено на модули для упрощения изменения различных его фрагментов независимо друг от друга. В качестве основных языков программирования, был выбран язык Kotlin и Java. Используя спецификацию OpenGL, были реализованы шейдеры на языке GLSL, а для их взаимодействия с программным интерфейсом используется библиотека LightWeight Java Game Library (LWJGL). Для упрощения сборки проекта были использованы возможности инструмента Maven.

3.1 Используемые технологии

Язык Java, по статистике за последние несколько лет, входит в тройку самых популярных языков программирования. Более подробная информация в [5].

Язык Kotlin [6] — статически типизированный язык программирования, разрабатываемый компанией JetBrains с 2011 г. Kotlin полностью совместим с Java и обладает более компактным и интуитивно понятным синтаксисом, а также некоторыми другими особенностями, помогающими сделать разработку более эффективной.

В соответствии с [7], язык GLSL (OpenGL Shading Language) — C-подобный высокоуровневый язык для программирования шейдеров, разработанный компанией Khronos Group в 2004 г. Был создан для предоставления программистам устойчивого контроля графических потоков на более высоком уровне, нежели другие машинно-зависимые языки.

В соответствии с [8], Maven — это специальный инструмент для упрощения сборки проектов с помощью использования описания их общей структуры на специальном языке POM. Он использует декларативную сборку проекта с помощью встроенных и внешних дополнительных плагинов, позволяет использовать один файл проекта на любых операционных системах. Maven использует специальный репозиторий для подкачки всех нужных зависимостей, в следствии чего не нужно вручную добавлять множественные библиотеки, если нужной части в проекте не найдется, Maven сам дополнит ее или оповестит

об ошибке.

LWJGL — это библиотека, обеспечивающая кросс-платформенный доступ к популярным API-интерфейсам, используемым в разработке графических приложений [9].

В качестве среды разработки была выбрана Intellij Idea.

3.2 Основная структура приложения

Приложение представляет собой оконный интерфейс. Оно состоит из более чем 30 классов, которые разбиты на следующие модули:

1. Основной модуль запуска программы;
2. Модуль, отвечающий за работу с дисплеем и выводом полученной графики на экран;
3. Модуль, работающий с камерой, которая позволяет передвигаться по изображению;
4. Модуль, ответственный за хранение информации об изображении, используя специальные типы данных VBO, VAO;
5. Модуль, генерирующий все основные объекты приложения, включая ландшафт и освещение;
6. Модуль, содержащий главные настройки генерации графики, а также ее некоторые ее характеристики.

3.3 Алгоритмы

Для генерации ландшафта был использован алгоритм шума Перлина. Основные этапы алгоритма:

1. Вся поверхность покрывается сеткой заданной размерности;
2. В каждой точке сетки задается некоторый случайный единичный вектор;
3. Для всех ячеек строятся случайные диагональные вектора к каждому пикселю;
4. Определяются пары векторов — диагонального вектора, связанного с пикселем и случайного единичного вектора. Для этих пар находятся скалярные произведения, которые являются значением высот соответствующих углов сетки;
5. Произвести интерполяцию полученных значений для получения результирующего значения высоты;
6. Повторить алгоритм нужное количество раз;

ЗАКЛЮЧЕНИЕ

В ходе работы было реализовано приложение, моделирующее ландшафт земной поверхности на языках Kotlin и Java, используя спецификацию OpenGL, специальное API, предоставляемое библиотекой LWJGL, а также язык GLSL для написания шейдеров. Данное приложение может быть использовано, как демонстрация механизмов генерации ландшафтов в практическом курсе по Компьютерной графике.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 OpenGL Overview [Электронный ресурс]. — URL: https://www.opengl.org/about/ (Дата обращения 30.05.2018). Загл. с экр. Яз. англ.

2 Learn OpenGL [Электронный ресурс]. — URL: https://learnopengl.com/ (Дата обращения 30.05.2018). Загл. с экр. Яз. англ.

3 Шейдеры и GLSL [Электронный ресурс]. — URL: https://webglfundamentals.org/webgl/lessons/ru/webgl-shaders-and-glsl.html (Дата обращения 30.05.2018). Загл. с экр. Яз. рус.

4 MoltenGL [Электронный ресурс]. — URL: https://moltengl.com/moltengl/ (Дата обращения 30.05.2018). Загл. с экр. Яз. англ.

5 Технология Java [Электронный ресурс]. — URL: https://java.com/ru/about/ (Дата обращения 30.05.2018). Загл. с экр. Яз. рус.

6 Learn Kotlin [Электронный ресурс]. — URL: https://kotlinlang.org/ (Дата обращения 30.05.2018). Загл. с экр. Яз. англ.

7 *Вольф, Д.* OpenGL 4. Язык шейдеров / Д. Вольф. — Россия: ДМК Пресс.

8 Apache Maven Project [Электронный ресурс]. — URL: https://maven.apache.org/ (Дата обращения 30.05.2018). Загл. с экр. Яз. англ.

9 Lightweight Java Game Library 3 [Электронный ресурс]. — URL: https://www.lwjgl.org/#learn-more (Дата обращения 30.05.2018). Загл. с экр. Яз. англ.