



**Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Генерация поверхностей трехмерного ландшафта»

Руководитель курсового проекта

О.В.Кузнецова

(И.О.Фамилия)

(Подпись, дата)

Студент

ИУ7-56Б

(Группа)

Ж.Р.Турсунов

(И.О.Фамилия)

(Подпись, дата)

Москва, 2020 г.

Министерство науки и высшего образования Российской Федерации Федеральное
государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э.Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э.Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
(Индекс)
_____ И.В.Рудаков
(И.О.Фамилия)
« ____ » _____ 2020 г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине Компьютерная графика

Студент группы ИУ7-56Б

Тема курсового проекта Генерация поверхностей трехмерного ландшафта

Направленность КП (учебный, исследовательский, практический, производственный, др.)

производственный

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

1. Техническое задание Разработать программу генерации случайных поверхностей ландшафта на основе карт высот. Генерация изображения карты высот должен быть на основе «Шума Перлина» или «Холмового алгоритма». Пользователю необходимо дать выбор генерации одного из этих методов. Расчет карты освещенности ландшафта должен происходить с использованием нескольких источников света на основе метода Фонга. Также необходимо разработать две программы, программу «Редактор» для генерации карты высот, расчета карты освещенности с поддержкой множества настроек и программу «Просмотр» для визуализации трехмерного ландшафта в реальном времени, в окружении воды и неба, на основе карт и текстур, сгенерированных заранее в программе «Редактор». Дать возможность для навигации по заданному миру.

2. Оформление курсового проекта

- Расчетно-пояснительная записка на 30-35 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

- Перечень графического (илюстративного) материала (чертежи, плакаты, слайды и т.п.) На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания « ____ » _____ 2020 г.

Руководитель курсового проекта

(Подпись, дата)

О.В.Кузнецова

(И.О.Фамилия)

Студент

(Подпись, дата)

Ж.Р.Турсунов

(И.О.Фамилия)

Содержание

Введение	5
1 Аналитическая часть	7
1.1 Обзор и анализ существующего по и обоснование необходимости разработки	7
1.1.1 Bryce	7
1.1.2 Vue 5 Esprit	8
1.1.3 VistaPro	9
1.1.4 Вывод	10
1.2 Алгоритмы загрузки и генерации ландшафта	10
1.2.1 Представление данных о ландшафте	10
1.2.2 Алгоритмы генерации рельефа	12
1.2.3 Вывод	18
1.3 Алгоритмы визуализации ландшафта и окружающей среды	19
1.3.1 Использование карт освещенности	19
1.3.2 Наложение текстур	19
1.3.3 Смешивание текстур	20
1.3.4 Мипмапы	21
1.3.5 Алгоритм Z-буфера	22
1.3.6 Вывод	23
1.4 Выводы из аналитической части	23
2 Конструкторская часть	24
2.1 Разработка алгоритмов	24
2.2 Освещение	26
2.3 Модель освещения Фонга и просчет теней	28
2.4 Просчет теней	29
2.5 Вычисление точки пересечения луча с треугольником для построения тени	30
2.6 Вывод из конструкторской части	32
3 Технологическая часть	33
3.1 Средства реализации	33
3.2 Структура программы	33
3.3 Входные и выходные данные	34
3.4 Интерфейс программы	35
3.4.1 Интерфейс программы «Editor»	35
3.4.2 Интерфейс программы «Viewer»	36
3.5 Вывод из технологической части	37
4 Исследовательская часть	38
4.1 Системные характеристики	38
4.2 Постановка эксперимента	38

4.3 Сравнительный анализ на основе замеров времени работы программы	38
4.4 Примеры работы программы	42
4.5 Вывод из исследовательской части	43
Заключение	44
Список литературы	45

Введение

Роль компьютеров с каждым днём становится всё больше и больше. С каждым разом увеличивается область применения этой технологии. Для каждой области применения необходимы свои собственные программы, для решения конкретных задач. Благодаря этому в настоящее время постоянно появляются новые предметы изучения и исследования. Это целая наука, занимающаяся построением реальных графических изображений посредством вычислительных действий. Компьютерная графика закрепила за собой много важных задач, среди которых присутствуют такие, как визуализация результатов, полученных при обработке данных, моделирование реальных процессов и объектов. Военная область – одна из тех, кто не только первыми опробовали компьютеры, но и послужили в формирование компьютерной графики как науки. Например, для обучения и подготовки к действиям в боевых условиях пилотов самолетов, танков или какой-нибудь другой бронетехники, было необходимым создание симуляторов реальных технических средств. Было гораздо безопасно и эффективно, в плане военного бюджета посадить человека за симулятор для получения первоначальных навыков, чем на реальный объект. А для создания симулятора требовалось получать реалистичные изображения различных объектов, например, местности, на которой ведутся учения, причем эти изображения должны быть получены в реальном времени, то есть так, чтобы в зависимости от действий обучаемого соответственно менялись и параметры системы, такие как положение на местности, погодные условия. Спустя время стало ясно, что такие системы можно использовать не только для обучения новичков, но и для планирования реальных боевых операций на любой территории, о которой есть определенные данные.

«Трехмерное моделирование», ветвь 3D-моделирования, которая занимается проектированием и созданием трехмерных реалистичных изображений всех видов поверхностей и ландшафтов. В настоящее время 3D-моделирование является важнейшей областью компьютерной графики, так как построение трехмерного изображения, близкого к реальному, является достаточно сложной задачей. Но, благодаря получаемым знаниям каждый день, невообразимому быстрому росту производительности вычислительных систем, эта область активно развивается.

Для простоты область 3D-моделирования можно разделить на 2 подобласти. Одна из них – это генерация трехмерных изображений в реальном времени, а другая – генерация высоко реалистичных трехмерных изображений. Высоко реалистичные изображения требуют очень больших объемов вычислений и производительности от машин, в то время как в первой подобласти применяется система условных допущений, позволяющая применять более простые алгоритмы, и тем самым значительно уменьшить объем вычислений, что, правда, сказывается на качестве изображения. Как уже было сказано, алгоритмы 3D-моделирования делятся на две группы.

В первую входят такие алгоритмы, как алгоритм Робертса, алгоритм Варнока, алгоритм, использующий z-буфер, которые в свою очередь используют простейшие алгоритмы построения линий, треугольников, закрашивание замкнутых областей. То есть алгоритмы, изученные нами в ходе курса «Компьютерной графики». Также к этой группе можно отнести алгоритмы, выполняющие отсечение.

Целью данной работы является генерация случайного трехмерного ландшафта.

Для достижения поставленной цели необходимо решить следующие задачи:

1. формализовать решаемую задачу;
2. рассмотреть и проанализировать существующие алгоритмы решения задачи, выбрать подходящий;
3. изучить физические принципы и законы, по которым строится выбранный метод визуализации;
4. выбрать наиболее подходящую реализацию выбранного алгоритма;
5. разработать архитектуру и структуру программы;
6. выбрать средства программной реализации;
7. разработать программные модули.

1 Аналитическая часть

В данной части будут рассмотрены алгоритмы генерации, визуализации ландшафта и окружающей среды.

1.1 Обзор и анализ существующего по и обоснование необходимости разработки

На сегодняшний день существуют множество программ генерирующих 3D ландшафт.[7] К более известным можно отнести следующие программы:

1. Bryce;
2. Vue 5 Esprit;
3. MojoWorld;
4. World Builder;
5. VistaPro.

1.1.1 Bryce

Bryce — одно из лучших профессиональных решений для моделирования как фотoreалистичных, так и фантастических ландшафтов, позволяющее создавать миры с планетами, океанами, островами, горами и пр. Пакет прекрасно подходит в случаях разработки пейзажей для трехмерных игр, для оригинальных изображений в сфере полиграфии или видео и активно используется профессиональными дизайнерами.

Недостатки:

- большой объем данных;
- большие требования к аппаратным ресурсам.

Преимущества:

- особые пакеты с частями пейзажа;
- расширенная поддержка экспорта и импорта.

На Рисунке 1 показан внутренний интерфейс программы Bryce.

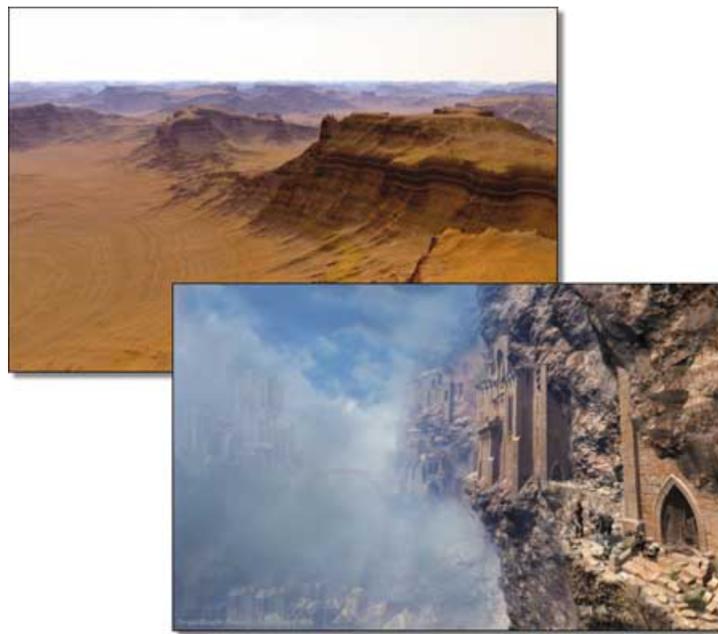


Рис. 1: Интерфейс программы Bryce.

1.1.2 Vue 5 Esprit

Vue 5 Esprit позволяет создавать как фотореалистичные, так и великолепные фантастические ландшафты с горами и водной гладью, великолепными атмосферными эффектами и разнообразной растительностью.

Недостатки:

- очень плохая поддержка импорта и экспорта;
- слишком много ненужных возможностей.

Преимущества:

- более привычный интерфейс;
- небольшой объем.

На Рисунке 2 показан внутренний интерфейс программы Vue 5 Esprit.

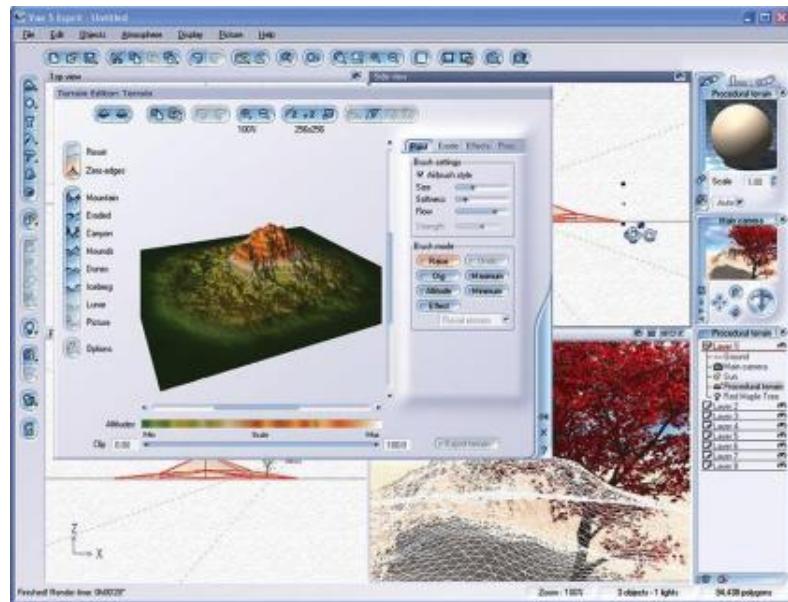


Рис. 2: Интерфейс программы Vue 5 Esprit.

1.1.3 VistaPro

VistaPro имеет самую длительную историю развития и когда-то работал еще под MS-DOS, позволяет строить как фотопрералистичные пейзажи, полностью соответствующие существующим в действительности в том или ином уголке планеты, так и совершенно фантастические ландшафты, возникающие в воображении художника.

Недостатки:

- ограниченность баз данных объектов;
- малое количество настроек для регулирования небесных, атмосферных и прочих ландшафтных параметров.

Преимущества:

- невысокие требования к аппаратным ресурсам;
- поддержка баз данных ГИС;
- небольшая цена.

На Рисунке 3 показан внутренний интерфейс программы VistaPro.

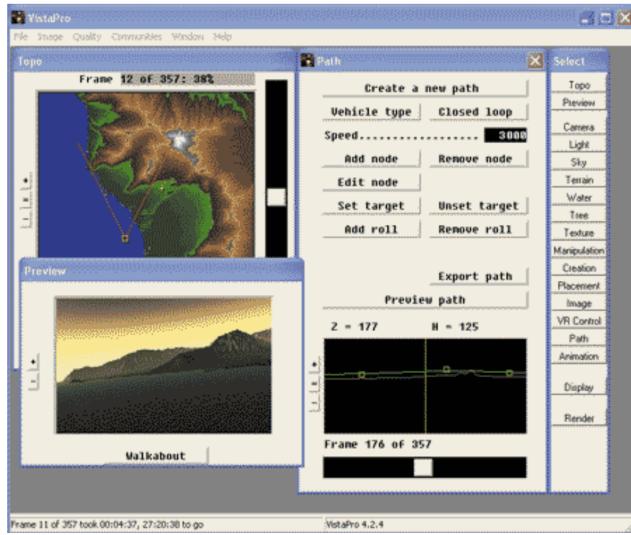


Рис. 3: Интерфейс программы VistaPro.

1.1.4 Вывод

Не смотря на то, что существует более чем достаточное количество программного обеспечения для генерации ландшафта, характерными минусами для всего программного обеспечения является высокие требования к аппаратным ресурсам и большой объем пакетов. В своей программе я устранил эти недостатки, сделав его более гибким к системным требованиям.

1.2 Алгоритмы загрузки и генерации ландшафта

1.2.1 Представление данных о ландшафте

Существует несколько основных принципов представления данных для хранения информации о ландшафтах:

1. использование регулярной сетки высот (или еще другое название Карта Высот - HeightMap);
2. использование иррегулярной сетки вершин и связей, их соединяющих (т.е. хранение простой триангулизированной карты);
3. хранение карты ландшафта, но в данном случае хранятся не конкретные высоты, а информация об использованном блоке. В этом случае создается некоторое количество заранее построенных сегментов, а на карте указываются только индексы этих сегментов.

В этом проекте был выбран первый способ представления ландшафтов.

Height Map: Данные представлены в виде двухмерного массива. Уже заданы две координаты (x, y - по высоте и ширине массива), и третья координата задается значением в конкретной ячейке, это высота.

Плюсы данного подхода:

1. простота реализации: легкость нахождения координат (и высоты) на карте, простая генерация ландшафта по карте высот или методом шума Перлина;
2. наглядность: в любой программе просмотра графических файлов можно сразу увидеть или изменить всю информацию;
3. скорость: благодаря конвейерной архитектуре процессора, просчет и вывод таких карт высот производится очень быстро (динамическое освещение, так как освещенность вершины напрямую зависит от расстояния от этой вершины до источника освещения).

Минусом данного подхода является большое количество избыточных данных (особенно для поверхностей, близких к плоским).

На Рисунке 4 показана визуализация данного подхода.

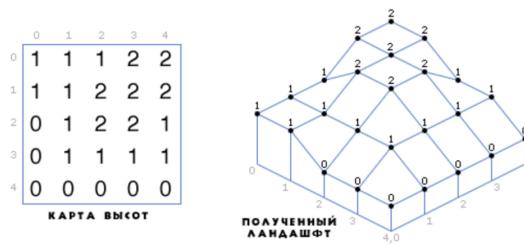


Рис. 4: Графическое представление метода Height Map

Иррегулярная сетка вершин и связей: Еще один способ представления данных для ландшафтов про которую стоит упомянуть — иррегулярная сетка вершин и связи их соединяющих. Зачастую такие решения применяются в специализированных пакетах для игр. И хранятся в виде трехмерных моделей.

Это дает основной выигрыш по сравнению с картами высот:

1. Используется значительно меньше информации для построения ландшафта. Здесь необходимо хранить только значения высот каждой вершины и связи эти вершины соединяющие. Это дает выигрыш в скорости при передаче огромных массивов информации, в процессе визуализации ландшафта.

Но также имеются множество недостатков:

1. алгоритмы построения ландшафтов в основном предназначены для регулярных карт высот. Оптимизация таких алгоритмов под этот способ потребует значительных усилий;
2. сложности при динамическом освещении — вершины расположены достаточно далеко друг от друга и неравномерно;
3. хранение, просмотр, модификация такого ландшафта также представляет сложности. При использовании карт высот достаточно применить простые и "стандартные" средства пиксельной графики. Например MS Paint. Тут же потребуются "весомые" пакеты.

Далее на Рисунке 5 показан метод иррегулярной сетки вершин и связей:

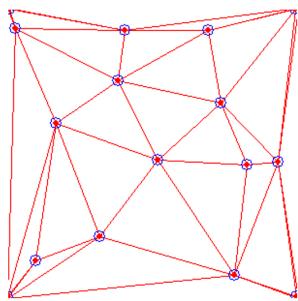


Рис. 5: Графическое представление метода иррегулярной сетки вершин и связей

1.2.2 Алгоритмы генерации рельефа

Шум Перлина: При разработки данной программы возникает вопрос: откуда брать информацию для генерации карты высот? Конечно же можно просто загружать монохромное изображение и на его основе генерировать ландшафт. Но если нужно каждый раз генерировать разные карты высот (например, для компьютерных игр или демонстрационных программ, таких как эта), то на помощь приходит следующий метод на основе шума Перлина.[1]

Изображение (или какой-либо другой объем) – вне зависимости от диапазона значений его элементов – полностью накрывается сеткой, представляющей диапазон вещественных чисел. Таким образом, создается шум на сетке, представляющей по всему изображению значения между 0 и 4. Каждое число порождает линию сетки, а значит, все стороны каждого квадрата последней имеют длину, равную одной единице.

Выбранный масштаб влияет на сложность шума. Большое число квадратов на сетке изображения создает более «плотно упакованный» шум, подобный белому шуму на экране плохо настроенного телевизора. Меньшее число квадратов на сетке порождает «клубящийся» шум, внешне похожий на облака. В каждой точке на сетке строится случайный вектор нормали. Это обычный двумерный вектор единичной длины, который указывает в случайном направлении в пределах каждого из квадратов. Традиционный способ создания таких векторов – организация справочной таблицы из 256 векторов, которые охватывают полный круг, и последующий случайный выбор одного из них для каждой точки на сетке. Это гарантирует распределение векторов, которые могут с равной вероятностью указывать в любом направлении. Далее для каждого пикселя изображения находится та из ячеек сетки, где он находится.

Таким образом, определяется значение, которое основано исключительно на данных этой ячейки. Следующий шаг – создать четыре диагональных вектора, соединяющих углы ячейки с текущим пикселием.

Ниже на Рисунке 6 приведено графическое представление данного принципа:

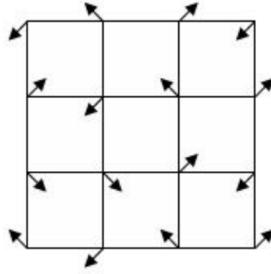


Рис. 6: Графическое представление одного из этапов построения шума

Каждый угол ячейки сетки теперь является базой для двух векторов – случайного единичного вектора и вектора в направлении пикселя, который необходимо построить. Для каждой пары таких векторов находится скалярное произведение. Оно даст скалярное значение высоты каждого из углов сетки. Далее необходимо объединить эти четыре значения и найти высоту пикселя, который надо сгенерировать. Делать это можно по-разному, получая различные результаты, однако чаще всего используется взвешенная интерполяция четырех значений с учетом близости текущей позиции к каждому углу сетки.

Основным плюсом использования шумовой функции при генерации ландшафта является то, что нет необходимости хранить карту высот, а достаточно лишь использовать данные справочной таблицы векторов, – все остальное для восстановления конечной карты высот ландшафта сделает шумовая функция.

Ниже на Рисунке 7 показан шум Перлина:

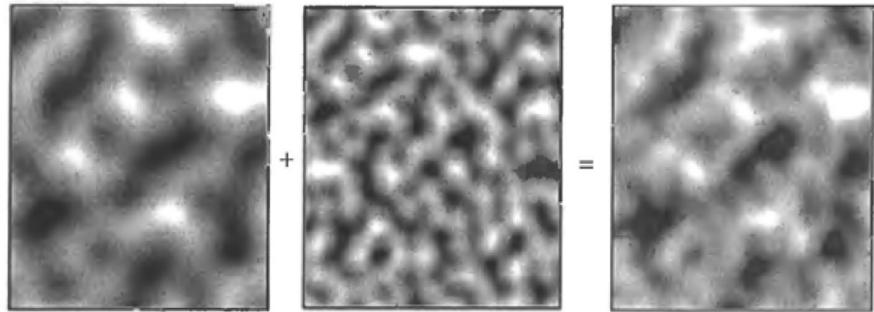


Рис. 7: Пример изображения шума Перлина, полученный сложением двух октав

Холмовой алгоритм: Это простой итерационный алгоритм, основанный на нескольких входных параметрах.[5]

Алгоритм изложен в следующих шагах:

1. создание и инициализация двухмерного массива с нулевым уровнем (все ячейки заполнены нулями);
2. выбирается случайная точка и случайный радиус в заранее заданных пределах. Выбор этих пределов влияет на вид ландшафта - либо он будет пологим, либо скалистым;
3. в выбранной точке "поднимается" холм заданного радиуса;

4. возвращение ко второму шагу и так далее до выбранного количества шагов. От него будет зависеть внешний вид ландшафта;
5. нормализация;
6. долинизация.

Первый, второй и четвертые шаги тривиальны, пятый и шестой шаг будет рассмотрен далее. Теперь же изучим третий шаг. Фактически холм в этом случае половина шара, чем больше радиус - тем больше холм (и выше). Математически это похоже на перевернутую параболу:

$$z = r^2 - ((x_2 - x_1)^2 + (y_2 - y_1)^2) \quad (1)$$

здесь (x_1, y_1) - заданная точка, r - выбранный радиус, (x_2, y_2) - высота холма.

На Рисунке 8 показан одиночный холм:

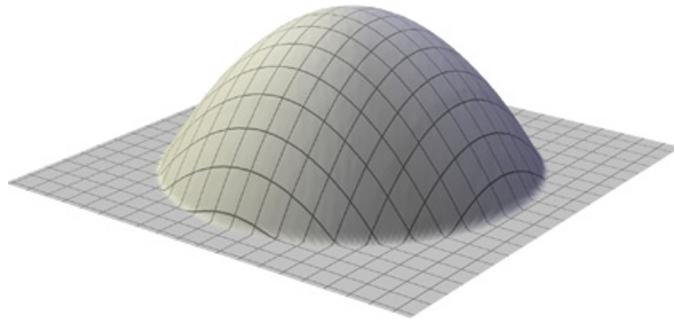


Рис. 8: Одиночный холм

Чтобы сгенерировать ландшафт полностью необходимо построить множество таких холмов. Но есть еще две вещи на которые необходимо обратить внимание. Первое - игнорирование отрицательных значений высоты холма. Второе - при генерации последующих холмов лучше добавлять полученное значение для данного холма к уже существующим значениям. Это позволяет строить более правдоподобный ландшафт, нежели правильно очерченные округлые холмы. Ниже на Рисунке 9 представлен ландшафт при большом количестве итераций:

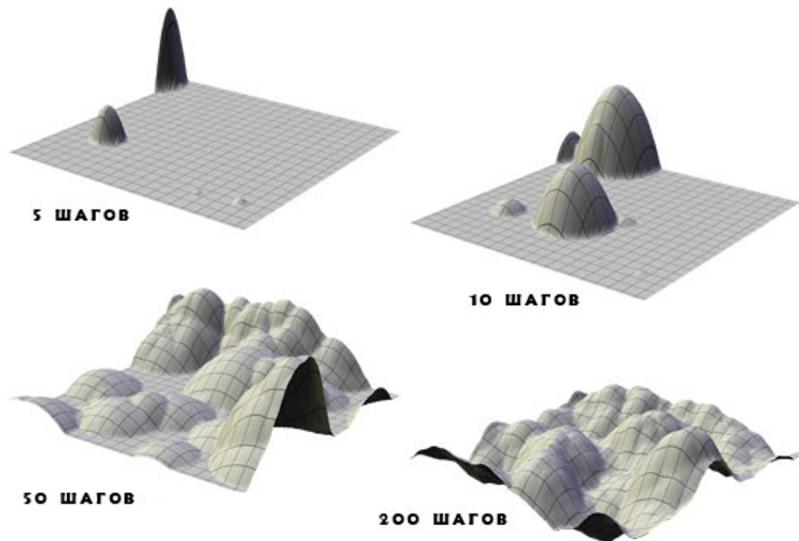


Рис. 9: Ландшафт при большом количестве итераций

Нормализация ландшафта

При генерации значений для ландшафта не учитывались выходы этих значений за некоторые пределы (например, если ландшафт будет храниться в монохромной картинке, то необходимо, чтобы все значения находились в пределе от 0 до 256). Для этого необходимо произвести нормализацию значений. Математически нормализация — это процесс получения значений из одного предела, и перевод его в другие пределы. Вот как это выглядит графически на Рисунке 10:

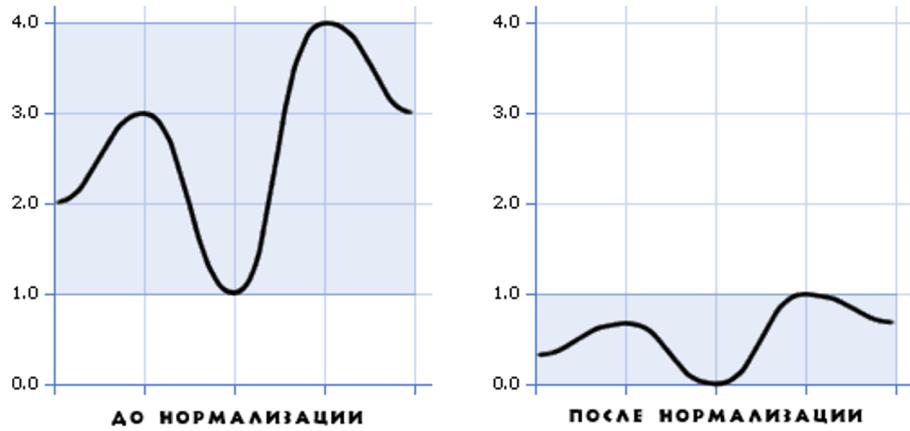


Рис. 10: Разница использования «Нормализации»

Чтобы это сделать произведем следующие действия:

1. сперва проходим по всему массиву и запоминаем наибольшее и наименьшее значения;
2. после этого производится нормализация конкретных значений в пределах от 0 до 1. Формула выглядит так:

$$z = \frac{z - \min}{\max - \min} \quad (2)$$

Долинизация ландшафта

Данный ландшафт уже можно использовать, но если присмотреться, то в нем достаточно мало долин. Склоны холмов излишне крутые, необходимо сделать их более пологими. Идея «Долинизации» состоит в следующем - взять от каждого значения квадратный корень. Это в большей степени влияет на средние значения, практически не затрагивая минимумов и максимумов. Графически это выглядит так как на Рисунке 11:

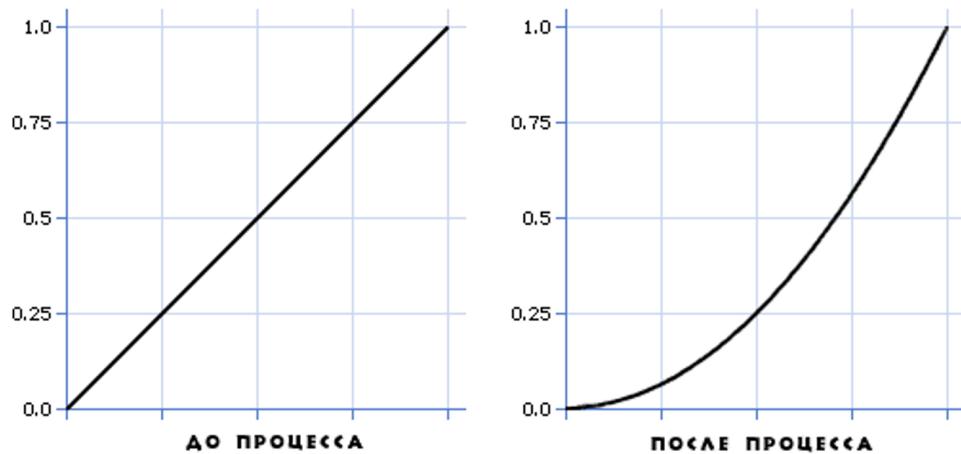


Рис. 11: Разница использования «Долинизации»

На Рисунке 12 можно увидеть преобразования ландшафта после «Долинизации» и «Нормализации»:

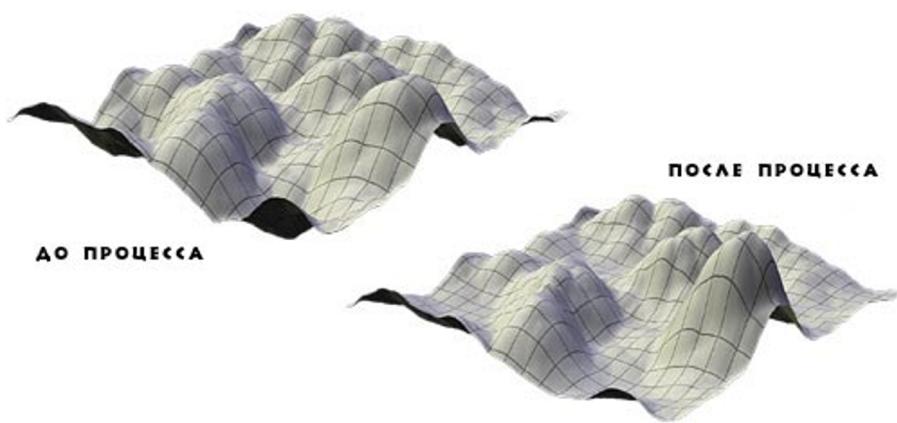


Рис. 12: Разница ландшафтов при использования «Долинизации» и «Нормализации»

В основном, рассмотренные выше алгоритмы предназначены для создания простого холмистого или гористого ландшафта. Но существуют и другие типы ландшафтов. Например, острова, озерные ландшафты. Их можно реализовать достаточно просто:

1. создание простого, достаточно холмистого ландшафта;
2. перемещение уровня воды вверх или вниз.

Так как уже было сказано выше, что ландшафт будет генерироваться двумя видами (остров и холмы). Ввиду того, что холмы уже рассмотрены. Необходимо понять, как правильно создать остров, чтобы он был реалистичным.

Во многих случаях используется уже рассмотренный алгоритм для генерации ландшафтов. Но иногда необходимо сгенерировать острова, или остров. В этом поможет слегка модифицированный алгоритм. В исходном алгоритме выбиралась центральная точка случайным образом, и она могла располагаться в любой части ландшафта. Теперь же необходимо, чтобы холмы были расположены ближе к центру.

Чтобы сделать это, вводятся две переменные (которые потом будем случайным образом изменяться), назовем их расстояние и угол. Расстояние будет означать, как далеко от центра находится центральная точка для одиночного холма. Оно может изменяться от ноля (прямо по центру карты высот) до половины величины карты высот минус радиус холма. Это позволит избежать ситуаций пересечения холмов с краем карты высот. Угол будет показывать, в каком направлении от центра нужно будет поставить холм. Изменяется угол будет в пределах от 0 до двух Пи. Используя эти два значения, можно получить значения (x, y) для центральной точки конкретного холма и использовать их как и в простом алгоритме.

Ниже представлен способ получения значений для x и y :

$$\theta = \text{random}(0, 2\pi) \quad (3)$$

$$distance = \text{random}\left(0, \frac{\text{size}}{2} - radius\right) \quad (4)$$

$$x = \frac{\text{size}}{2} + \cos(\theta) * distance \quad (5)$$

$$y = \frac{\text{size}}{2} + \sin(\theta) * distance \quad (6)$$

здесь $size$ - размер карты высот, $distance$ - расстояние, θ - угол. Важно, чтобы радиус был меньше половины размера карты высот.

Поработав с этими величинами, можно получить довольно прилично выглядящий остров. На Рисунке 13 показан этот остров:

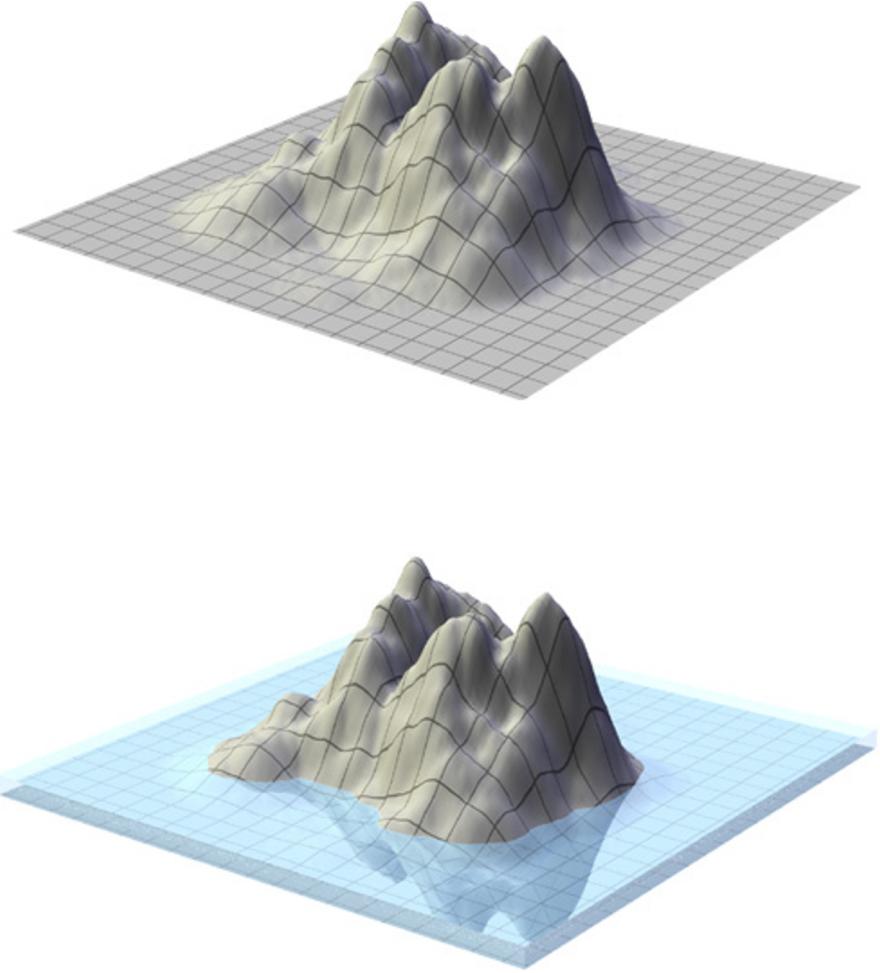


Рис. 13: Получение из обычного холма остров

Выше были разобраны некоторые алгоритмы построения карт высот для ландшафтов. Рассмотрим еще некоторые сопутствующие операции:

1. сглаживание (или размытие). Низкочастотный фильтр для уменьшения эффектов угловатости. При многократном применении позволяет добиться очень гладких очертаний ландшафта;
2. превращение гористой местности в холмистую. Различия выделяются по очертанию вертикальных разрезов у вторых более пологие края;
3. создание пляжей и отмелей в случае с островами и берегами. Для этого места соприкосновения с водой сглаживают. Хотя могут существовать и скалистые пляжи и просто скалы.

1.2.3 Вывод

В данном разделе были рассмотрены алгоритмы представления данных и генерации ландшафта. Из перечисленных выше алгоритмов представления данных был выбран алгоритм использования регулярной сетки высот, так как является самым популярным и простым в реализации методом.

1.3 Алгоритмы визуализации ландшафта и окружающей среды

1.3.1 Использование карт освещенности

В широком смысле *карта освещения* – это структура данных, хранящая информацию об освещенности (яркости) поверхностей трехмерной сцены. Карты освещения рассчитываются предварительно для неподвижных объектов и позволяют ускорить рисование освещенной сцены. Сейчас под картами освещения чаще всего подразумевают одну из их разновидностей – текстурные карты освещения. Они представляют собой изображения, накладываемые при рисовании «поверх» основных текстур; при этом яркость точки на карте освещения используется для модуляции яркости точки основной текстуры. Этот процесс продемонстрирован на Рисунке 14:



Рис. 14: Использование карты освещенности

Вычисление результирующего цвета

$$R = R_{text} * R_{light}$$

$$G = G_{text} * G_{light}$$

$$B = B_{text} * B_{light}$$

Значение компонентов цвета лежат в диапазоне [0, 1]. Если карта освещенности будет полностью белой, то результирующая текстура будет такой же, как и основная текстура (компоненты везде домножаются на 1, так как белый цвет это $R = 1, G = 1, B = 1$) если карта освещенности будет полностью черной, то результирующая текстура тоже черной (компоненты везде домножаются на 0, так как черный цвет это $R = 0, G = 0, B = 0$), что соответствует отсутствие источников освещения.[2]

1.3.2 Наложение текстур

Текстуры позволяют увеличить детализированность изображения, не добавляя в сцену дополнительную геометрию, и поэтому широко распространены в трехмерной графике . Как правило, трехмерная модель, созданная в пакете трехмерного моделирования, содержит не только информацию о геометрии, но и текстурные координаты – пары чисел U и V, указывающие на точку текстуры. Текстурные координаты задаются в вершинах граней, и задача наложения текстур сводится к интерполяции текстурных координат U и V по всем точкам грани.

В аффинном текстурировании используется линейная интерполяция:

$$u_\alpha = (1 - \alpha)u_0 + \alpha u_1 \quad (7)$$

Здесь u_0 и u_1 – значения текстурной координаты U , заданные на концах некоторого отрезка. Точно такая же формула используется для координаты V . Этот метод работает быстро, но дает некорректные результаты для граней, расположенных под углом к экрану, так как при интерполяции не учитываются значения глубины точек.

Перспективно-корректное текстурирование, как следует ожидать из названия, дает корректные результаты для произвольных граней благодаря дополнительной интерполяции глубины точек:

$$u_\alpha = \frac{(1 - \alpha) \frac{u_0}{z_0} + \alpha \frac{u_1}{z_1}}{(1 - \alpha) \frac{1}{z_0} + \alpha \frac{1}{z_1}} \quad (8)$$

Здесь z_0, z_1 – глубины концов отрезка, на котором проводится интерполяция. Ниже на рисунке 15 представлено сравнение методов текстурирования:

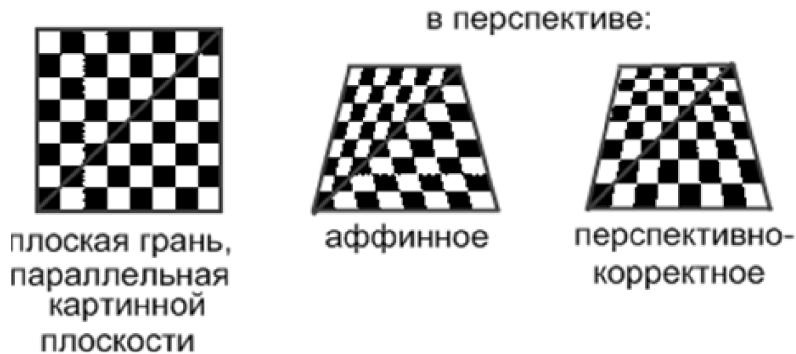


Рис. 15: Сравнение методов текстурирования

1.3.3 Смешивание текстур

Прозрачность реализуется с помощью специального режима смешения цветов (*blending*). Алгоритм смешения комбинирует цвета так называемых входящих пикселей с цветами соответствующих пикселей, уже хранящихся в буфере. Для смешения используется четвертая компонента цвета – альфа-компонент, поэтому этот режим называют еще альфа-смешиванием. Программа может управлять интенсивностью альфа-компоненты точно так же, как и интенсивностью основных цветов, т.е. задавать значение интенсивности для каждого пикселя или каждой вершины примитива. Расчет результирующего цвета каждого пикселя:

$$res = c_{src} * k_1 + c_{dst} * k_2 \quad (9)$$

Параметр src определяет, как получить коэффициент k_1 исходного цвета пикселя, а dst задает способ получения коэффициента k_2 для цвета в буфере кадра. Параметр c_{src} – цвет исходного пикселя, c_{dst} – цвет пикселя в буфере кадра ($res, k_1, k_2, c_{src}, c_{dst}$ – четырехкомпонентные RGBA-векторы).

Если в сцене есть несколько прозрачных объектов, которые могут перекрывать друг друга, корректный вывод можно гарантировать только в случае выполнения следующих условий:

1. все прозрачные объекты выводятся после непрозрачных;
2. при выводе объекты с прозрачностью должны быть упорядочены по уменьшению глубины, т.е. выводиться, начиная с наиболее удаленных от наблюдателя.

1.3.4 Мипмапы

Мипмапы или Мип-карты — предрассчитанный, оптимизированный набор изображений связанных с одной текстурой и предназначенный для увеличения скорости рендеринга и улучшения качества изображения.[4]

Каждое следующее изображение в наборе вдвое меньше предыдущего. То есть самое первое имеет размер равный размеру текстуры, второе вдвое меньший, третье — вчетверо и так далее до размера 1x1 тексель. На Рисунке 16 представлен пример Мип-карты:

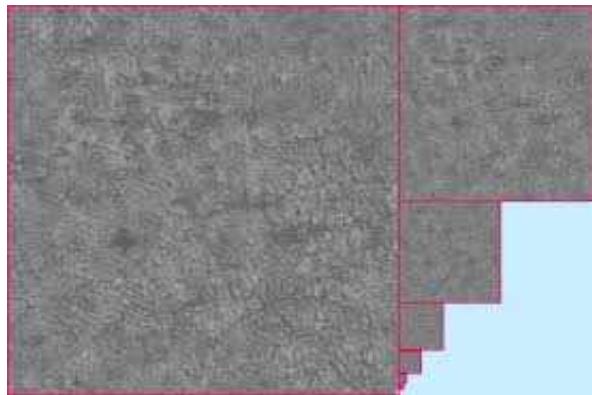


Рис. 16: Пример Мип-карты

Смысл такого вот пред рассчитанного набора состоит в том, что при текстурировании будет выбираться изображение с наиболее подходящим размером. Предположим, что на модель натянута текстура размером 512x512. Модель стоит далеко от камеры и геометрические размеры на экране у нее малы (скажем 3 пикселя) При отключенном мипмапинге видеокарте придётся выбирать, какой тексель из большой текстуры будет использован для расчёта цвета точки. Этот выбор может меняться при изменении ракурса камеры, что приведёт к мерцанию объектов вдали.

При включенному мипмапинге видеокарта выберет более подходящий размер текстуры, и будет производить выборку из него, что избавит нас от артефактов изображения. Использование мипмапинга также повышает быстродействие, так как более эффективно используется текстурная кэш-память видеокарты, и меньше данных приходиться передавать по шине.

Генерацию мип-уровней можно сделать несколькими способами:

1. указать видеокарте сгенерировать мип-уровни. Качество не будет очень хорошим, так как для генерации будет использоваться простой Box фильтр;
2. генерировать мипмапы самому, воспользовавшись пакетами для редактирования изображений;
3. отрисовать все мип-уровни вручную.

При применении *Билинейной фильтрации* или *Трилинейной фильтрации* с мипмапингом можно получить более размытые текстуры на поверхностях вдали или под углом.

Билинейная фильтрация - процесс извлечения нескольких пикселей исходной текстуры с последующим усреднением их значений для получения окончательного значения пикселя. Понятие «билинейная фильтрация», точно так же, как и сходное понятие «трилинейная фильтрация», применимо только к двумерным текстурам. Для трехмерных, например, данное понятие неприменимо, а понятие трилинейной фильтрации имеет совершенно другое значение. Пример исходного кода функции билинейной фильтрации.

Трилинейная фильтрация - усовершенствованный вариант билинейной фильтрации. Цвет пикселя высчитывается как средневзвешенное восьми текстелей: по четыре на двух соседних MIP-текстурах. В случае, если формулы MIP-текстурирования дают самую крупную или самую маленькую из MIP-текстур, трилинейная фильтрация вырождается в билинейную.

MIP-текстурирование, повышая чёткость изображения и процент попаданий в кэш на дальних расстояниях, имеет серьёзный недостаток: ясно видны границы раздела между MIP-уровнями. Трилинейная фильтрация позволяет исправить этот недостаток ценой некоторого снижения резкости текстур.

1.3.5 Алгоритм Z-буфера

Это один из простейших алгоритмов удаления невидимых поверхностей. Работает этот алгоритм в пространстве изображения. Идея z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пикселя в пространстве изображения, z-буфер – это отдельный буфер глубины, используемый для запоминания координаты z, или глубины каждого видимого пикселя в пространстве изображения.[6]

Главное *преимущество* алгоритма – его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в z-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине. При этом, алгоритм z-буфера, будучи реализованный аппаратно, является самым быстрым алгоритмом удаления невидимых поверхностей.

Основной недостаток алгоритма – большой объем требуемой памяти. Другой недостаток алгоритма z-буфера состоит в трудоемкости и высокой стоимости устранения лестничного эффекта, а также реализации эффектов прозрачности и просвечивания.

При визуализации изображения, как пиксельная информация, так и глубина усредняются. В этом методе требуются очень большие объемы памяти. Например, изображение размером 512x512x24 бита, использующее z-буфер размером 20 бит на пиксель, разрешение которого повышенено в 2 раза по осям x и y, и на котором устранена ступенчатость методом равномерного усреднения, требует почти 6 МБ памяти.

1.3.6 Вывод

Здесь были рассмотрены алгоритмы визуализации ландшафта и окружающей среды. Были показаны плюсы и минусы использования mip-карты. Представлены решения проблем полученных при использовании mip-уровней. Были рассмотрены способы увеличения скорости показана изображения.

1.4 Выводы из аналитической части

В данном разделе были представлены алгоритмы загрузки данных о ландшафте. Были представлены преимущества и недостатки этих методов. Также было подробно описано про порядок визуализации ландшафта и окружающей среды. Побробная схема реализация всех этих алгоритмов будет представлена в следующем разделе.

2 Конструкторская часть

В данном разделе представлены подробное описание процесса освещения, расчета теней, а также схемы алгоритмов указанных в аналитической части.

2.1 Разработка алгоритмов

В данном пункте будут рассмотрены схемы следующих алгоритмов:

1. Холмовой алгоритм;
2. Алгоритм Z-буфера.

На Рисунках 17 и 18 представлены схемы алгоритмов z-буфера и холмового алгоритма:



Рис. 17: Схема алгоритма Z-буфера

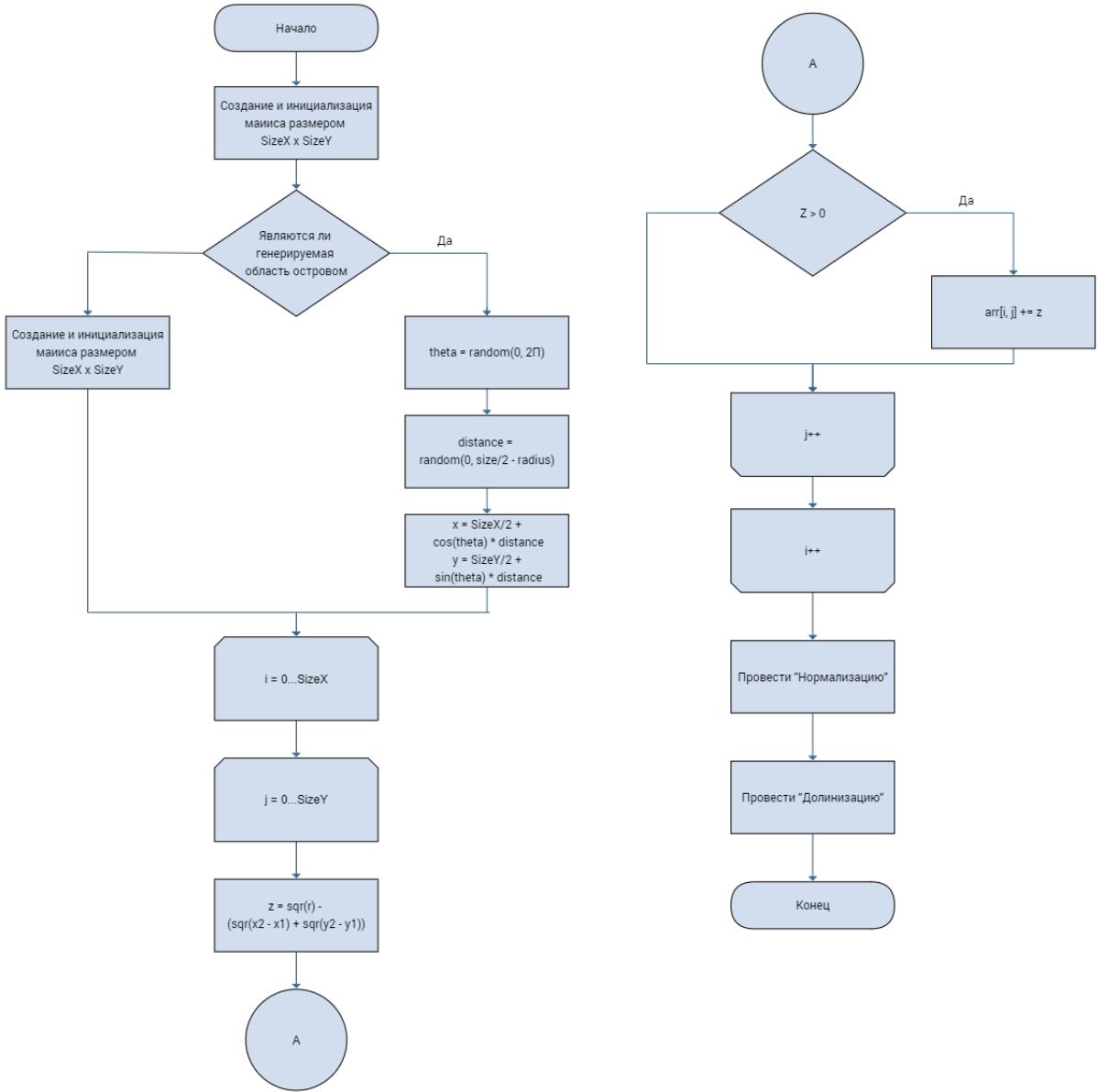


Рис. 18: Схема холмового алгоритма

2.2 Освещение

В любом трёхмерном приложении использование какой-либо модели освещения всегда придаёт реалистичность обрабатываемой сцене. Как правило, в ней включается закон, по которому рассчитывается освещённость точки в пространстве, и метод закраски освещённого многоугольника. От выбора той или иной модели освещения зависит качество изображения, построенного компьютером, и скорость работы программы.

Обычно освещённость некоторой точки, принадлежащей грани в пространстве, складывается из рассеянной освещённости и диффузного отражения — потока света, отражающегося от поверхности объекта. Иногда к ним добавляют зеркальное отражение — поток света, отражающийся от внешней поверхности объекта под тем же углом, под которым он падал на эту поверхность. Однако в данной работе зеркальное отражение света не учитывается, так предварительно производится расчет кар-

ты освещенности, которая затем модулируется с текстурой и накладывается на трехмерный объект, а при расчете зеркальной компоненты света учитывается положение наблюдателя, которое может быть различным в определенный момент времени (нужно будет пересчитывать карту освещенности заново, что недопустимо). Кроме того, расчёт интенсивности зеркального отражения, например по модели Фонга, требует немалых вычислительных затрат. Для него требуется рассчитывать угол между вектором наблюдения и вектором отражения и возводить косинус этого угла в некоторую степень, зависящую от свойств поверхности.

Диффузное отражение присуще матовым поверхностям. Матовой можно считать такую поверхность, размер шероховатостей которой настолько велик, что падающий луч рассеивается неравномерно во все стороны. Такой тип отражения характерен, например, для гипса, песка, бумаги. Диффузное отражение описывается законом Ламберта, согласно которому интенсивность отраженного света пропорциональна косинусу угла между направлением на точечный источник света и нормалью к поверхности(10).

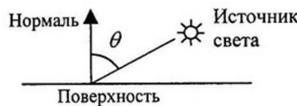


Рис. 19: Матовая поверхность

$$I_d = I K_d \cos \theta \quad (10)$$

Здесь I_d - интенсивность источника света, K_d - коэффициент, который учитывает свойства материала поверхности. Интенсивность отраженного света не зависит от расположения наблюдателя.

Матовая поверхность имеет свой цвет. Наблюдаемый цвет матовой поверхности определяется комбинацией собственного цвета поверхности и цвета излучения источника света (в данной работе цвет излучения источника считается белым, поэтому учитывается только цвет поверхности).

Для точечного источника света можно еще усовершенствовать модель отражения, если учесть, что энергия уменьшается пропорционально квадрату расстояния и пропорционально расстоянию.

Кроме того, в данной программе вместо интенсивности света используется расширенная величина цвета света, которая состоит из трех компонент (R, G, B)

Итак, цвет в данной точке для одного источника направленного освещения рассчитывается по следующей формуле:

$$I_{reflection} = I_{light} K_d \cos \theta \quad (11)$$

Для точечного источника света:

$$I_{reflection} = \frac{I_{light}}{c_1 + c_2 d + c_3 d^2} K_d \cos \theta \quad (12)$$

I - интенсивность источника света(R, G, B).

K_d - способность материала отражать диффузный свет (тоже имеет R, G, B).

d - расстояние от источника света до рассматриваемой точки поверхности, c_1, c_2, c_3 – произвольные коэффициенты угасания, которые находятся эмпирическим путем

Для определения косинуса угла между вектором нормали к поверхности и вектором, определяющим положение источника света в пространстве, следует воспользоваться скалярным произведением. Пусть имеется вектор нормали $N(X_N, Y_N, Z_N)$ и две точки – $P(X_P, Y_P, Z_P)$, принадлежащая поверхности, и $L(X_L, Y_L, Z_L)$, определяющая положение источника. Вектор, направленный от точки поверхности к источнику света, имеет следующие координаты: $V(X_L - X_P, Y_L - Y_P, Z_L - Z_P)$. Тогда

$$N * V = |N||V| \cos \theta \quad (13)$$

$$|N| = \sqrt{X_N^2 + Y_N^2 + Z_N^2} \quad (14)$$

$$|V| = \sqrt{(X_L - X_P)^2 + (Y_L - Y_P)^2 + (Z_L - Z_P)^2} \quad (15)$$

Расспишем (13) используя (14) и (15):

$$N * V = X_N X_V + Y_N Y_V + Z_N Z_V \quad (16)$$

Следовательно угол равен:

$$\cos \theta = \frac{X_N X_V + Y_N Y_V + Z_N Z_V}{|N||V|} \quad (17)$$

Однако в программе используются, как правило, единичные вектора нормалей, что в данном случае позволяет уменьшить количество требуемых вычислений. В итоге:

$$\cos \theta = \frac{X_N(X_L - X_P) + Y_N(Y_L - Y_P) + Z_N(Z_L - Z_P)}{|V|} \quad (18)$$

Или более развернуто,

$$\cos \theta = \frac{X_N(X_L - X_P) + Y_N(Y_L - Y_P) + Z_N(Z_L - Z_P)}{\sqrt{(X_L - X_P)^2 + (Y_L - Y_P)^2 + (Z_L - Z_P)^2}} \quad (19)$$

2.3 Модель освещения Фонга и просчет теней

При использовании метода Фонга для определения цвета в каждой точке интерполируются не интенсивности отраженного света, а векторы нормалей[3]. Последовательность действий такова:

1. определяются нормали к граням;
2. по нормалям к граням определяются усредненные нормали в вершинах. В каждой точке за-крашиваемой грани определяется интерполированный вектор нормали;
3. по направлению векторов нормали определяется цвет точек грани в соответствии с принятой моделью отражения цвета.

Как уже было сказано, метод заключается в интерполяции вектора нормали. Для интерполяции будут использоваться векторы N_a, N_b, N_c , исходящие из начала координат плоскости проецирования и параллельными соответствующим нормалям N_a, N_b, N_c в вершинах a, b, c .

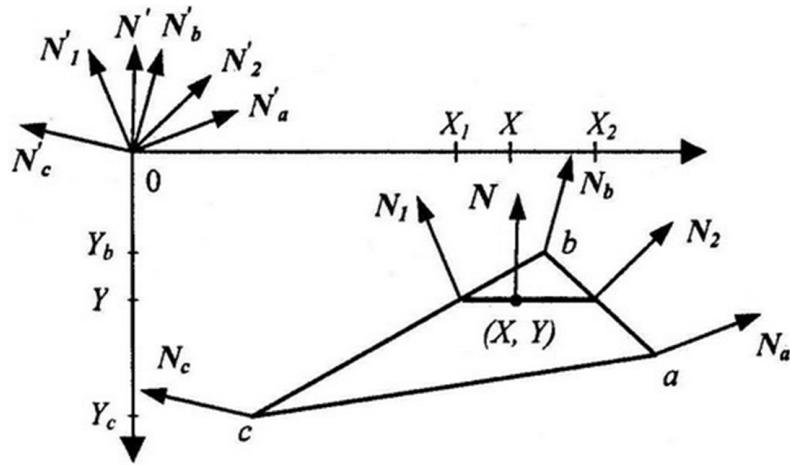


Рис. 20: Графическое представление векторов для интерполяции

Нахождение N_1' и N_2' производится следующим образом:

$$N_1' = \begin{pmatrix} X_{N1}' \\ Y_{N1}' \\ Z_{N1}' \end{pmatrix} = \begin{pmatrix} X_{Nb} + \frac{(X_Nc - X_Nb)(Y - Y_b)}{Y_c - Y_b} \\ Y_{Nb} + \frac{(Y_Nc - Y_Nb)(Y - Y_b)}{Y_c - Y_b} \\ Z_{Nb} + \frac{(Z_Nc - Z_Nb)(Y - Y_b)}{Y_c - Y_b} \end{pmatrix}$$

$$N_2' = \begin{pmatrix} X_{N2}' \\ Y_{N2}' \\ Z_{N2}' \end{pmatrix} = \begin{pmatrix} X_{Nb} + \frac{(X_Na - X_Nb)(Y - Y_b)}{Y_a - Y_b} \\ Y_{Nb} + \frac{(Y_Na - Y_Nb)(Y - Y_b)}{Y_a - Y_b} \\ Z_{Nb} + \frac{(Z_Na - Z_Nb)(Y - Y_b)}{Y_a - Y_b} \end{pmatrix}$$

Здесь $X_{Na}, Y_{Na}, Z_{Na}, X_{Nb}, Y_{Nb}, Z_{Nb}, X_{Nc}, Y_{Nc}, Z_{Nc}$ - координаты векторов N_a, N_b, N_c .

В данном случае необходимости в вычислении некоторых величин на каждом шаге нет. Так что их можно вычислить заранее. Теперь необходимо найти координаты вектора N' :

$$N' = \begin{pmatrix} X_N' \\ Y_N' \\ Z_N' \end{pmatrix} = \begin{pmatrix} X_{N1} + \frac{(X_{N2} - X_{N1})(X - X_1)}{X_2 - X_1} \\ Y_{N1} + \frac{(Y_{N2} - Y_{N1})(X - X_1)}{X_2 - X_1} \\ Z_{N1} + \frac{(Z_{N2} - Z_{N1})(X - X_1)}{X_2 - X_1} \end{pmatrix}$$

Вектор N' параллелен вектору N для нормали в точке (X, Y) , поэтому его можно использовать

для расчета отражения света так же, как и вектор нормали N .

Метод Фонга дает правильное закрашивание. Если интерполировать нормали передней грани, то по центру будут интерполированные нормали, параллельные лучам света. Поэтому центр передней грани будет светлее, чем края.

2.4 Просчет теней

Для построения сплошных теней на этапе вычисления «локальной» интенсивности цвета в точке объекта проверяется «видимость» каждого источника света из этой точки.

Принцип работы алгоритма:

1. из проверяемой точки строится луч, направленный на источник света;
2. производится поиск пересечений этого луча с примитивами сцены между проверяемой точкой и источником;
3. если найдено *хотя бы одно пересечение*, то проверяемая точка находится в тени. При расчете ее цвета источник, для которого проводилась проверка, не учитывается;
4. если пересечений не найдено, точка не в тени. При расчете ее цвета учитываем проверяемый источник.

Такой метод нахождения теней дает приемлемый результат до тех пор, пока на сцене нет прозрачных объектов. Также этот метод не позволяет достичь построения реалистичной тени, потому что не учитывается дифракция света (сглаженная тень у края).

2.5 Вычисление точки пересечения луча с треугольником для построения тени

Для вычисления точки пересечения луча, необходимо сначала определить точку пересечения этого луча с плоскостью, содержащей этот треугольник.

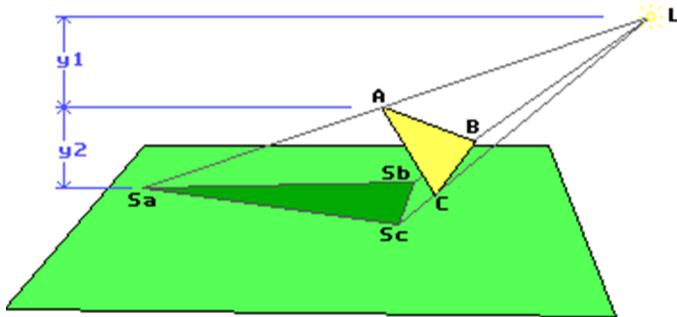


Рис. 21: Полученная тень от треугольника

Уравнение плоскости выглядит следующим образом:

$$Q(x, y, z) = Ax + By + Cz + D = 0. \quad (20)$$

Здесь коэффициенты A, B, C совпадают с координатами нормали к этой плоскости. Координаты нормали плоскости совпадают с координатами нормали треугольника, которые мы посчитали на этапе загрузки сцены.

Для нахождения свободного члена D необходимо подставить координаты любой точки треугольника, например, одной из вершин.

$$D = -Ax - By - Cz \quad (21)$$

По ходу выполнения программы значение D меняться не будет, поэтому его целесообразно посчитать при инициализации сцены и хранить, как и координаты нормали. Пересчитывать его необходимо только при изменении положения треугольника.

Теперь для нахождения точки пересечения подставим уравнения луча в уравнение плоскости.

$$A(x_1 + at) + B(y_1 + bt) + C(z_1 + ct) + D = 0 \quad (22)$$

Откуда получим

$$t = \frac{-(Ax_1 + By_1 + Cz_1 + D)}{(Aa + Bb + Cc)} \quad (23)$$

Если знаменатель этой дроби равен нулю, значит луч параллелен плоскости, в которой лежит треугольник. Точки пересечения нет.

Для нахождения координат точки пересечения надо подставить найденное значение параметра t в уравнения луча. Назовем точку пересечения D . Мы получим координаты x_D, y_D, z_D .

Теперь необходимо определить, попала ли точка D внутрь треугольника. Найдем координаты векторов AB, BC, CA (A, B, C – вершины треугольника) и координаты векторов AD, BD, CD . Затем найдем три векторных произведения:

$$nA = AB * AD \quad (24)$$

$$nB = BC * BD \quad (25)$$

$$nC = CA * CD \quad (26)$$

Эти вектора будут коллинеарные. Если все три вектора сонаправлены, то точка D лежит внутри треугольника. Сонаправленность определяется равенству знаков соответствующих координат всех трех векторов.

Операцию проверки принадлежности точки D треугольнику ABC можно ускорить. Если ортогонально спроектировать треугольник ABC и точку D на одну из плоскостей xOy, yOz или xOz , то попадание проекции точки в проекцию треугольника будет означать попадание самой точки в треугольник (конечно же, если уже известно, что точка D лежит в плоскости, содержащей треугольник ABC). При этом число операций заметно сокращается. Так для поиска координат всех векторов нужно искать по две координаты на каждый вектор, а при поиске векторных произведений нужно искать только одну координату (остальные равны нулю).

Для проверки сонаправленности векторов, полученных при вычислении векторного произведения нужно проверить знаки этой единственной координаты для всех трех векторов. Если все знаки больше нуля, или меньше нуля, то вектора сонаправлены. Равенство нулю одного из векторных произведений соответствует случаю, когда точка D попадает на прямую, содержащую одну из сторон треугольника.

Кроме того, перед вычислениями векторов и векторных произведений можно провести простой габаритный тест. Если проекция точки D лежит правее, левее, выше или ниже каждой из проекций вершин треугольника, то она не может лежать внутри.

Остается добавить, что для проецирования лучше выбирать ту из плоскостей, площадь проекции треугольника на которую больше. При таком условии исключается случай проецирования треугольника в отрезок (при условии, что проверяемый треугольник не вырожден в отрезок). Кроме того, при увеличении площади проекции уменьшается вероятность ошибки. Для определения такой плоскости проецирования достаточно проверить три координаты нормали треугольника. Если z -координата нормали больше (по абсолютному значению) x и y , то проецировать надо на плоскость xOy . Если y больше чем x и z , то проецируем на xOz . В оставшемся случае – на yOz .

2.6 Вывод из конструкторской части

В данном разделе были представлены схемы используемых алгоритмов, а именно алгоритмы z-буфера и холмовой алгоритм. Также было подробно описано таких важных частей как: освещение, просчет теней.

3 Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации.

3.1 Средства реализации

В данной работе используется такие языки программирования как C# и C++. Изначально был выбран только C#, но в дальнейшем было решено использовать и C++. Это связано с тем, что этот язык довольно новый для меня и использование его в моем проекте предоставило бы опыт разработки.

Весь программный модуль состоит из 2 частей:

1. программа «Editor»;
2. программа «Viewer».

Языком программирования для написания программы «Editor» был выбран язык C#. С помощью C# можно быстро писать программы под Windows, благодаря удобному синтаксису, интеллектуальным подсказкам и управляемому коду. Используя управляемый код, не нужно заботится о сборке мусора, об указателях и о некоторых базовых структурах и алгоритмах – все это уже реализовано. Главным минусом разработки на C# является низкая производительность программ. Но это уже компенсируется современным аппаратным обеспечением. Кроме того, в редакторе нам и не требуется высокая производительность, потому что там рассчитываются только карты и нет интерактивных элементов, в отличие от программы просмотра.

Языком программирования для написания программы просмотра ландшафтов был выбран язык C++. Язык C++ сочетает в себе удобство с точки зрения структуры кода (объекты, перегрузка функций, операторов, обработка исключений и др.) и программирование на низком уровне (арифметика указателей, ассемблерные вставки, выравнивание памяти, директивы компилятора). Минусы языка C++ тоже имеются: ручная сборка мусора (возможные утечки памяти из-за этого).

В качестве среды разработки была выбрана IDE Visual Studio 2008. Была выбрана именно эта версия, так как она идеально подходит с точки зрения функциональности, простоты использования и используемой памяти по сравнению с более поздними версиями. Также эта среда разработки может сильно оптимизировать код, используя раскрытие циклов, встраивание функций, использование суперскалярных команд процессора SSE и другие, тем самым увеличив производительность.

В качестве технологического подхода к разработке проекта был выбран объектно-ориентированный подход. Этот выбор обуславливается легкостью и быстрой разработки программы.

3.2 Структура программы

На рисунке 22 представлена структура разработанного программного продукта.

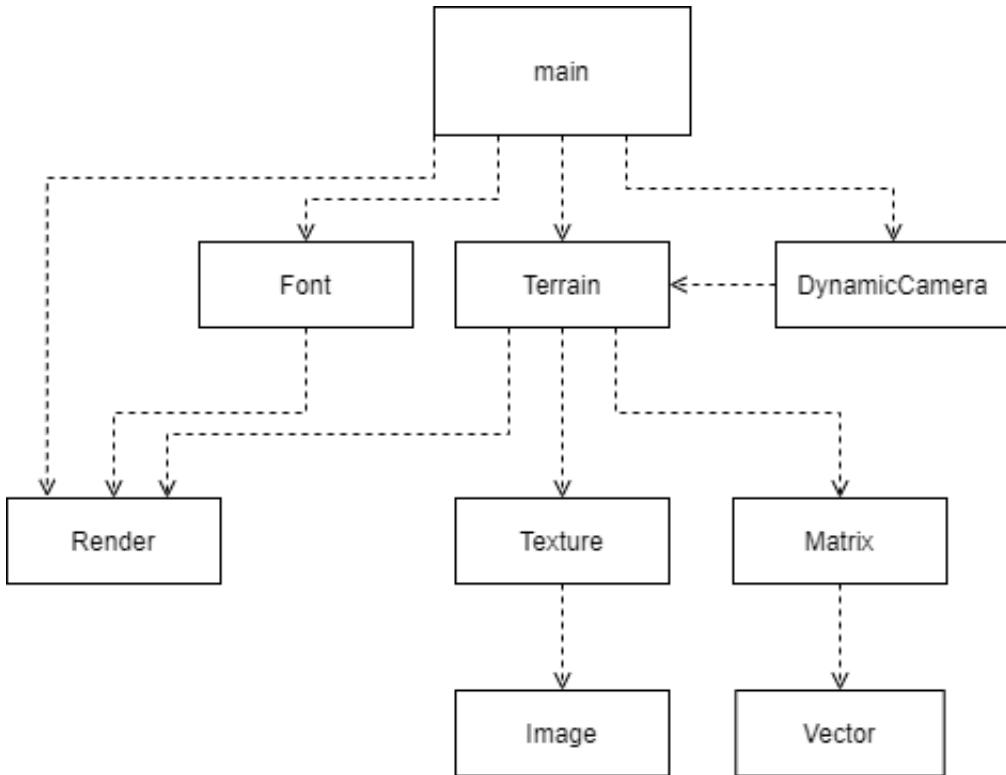


Рис. 22: Структура разработанной программы

Main – модуль, в котором происходит связывание всех компонентов программы.

Terrain – описывает непосредственно геометрию ландшафта (вершины, индексы, нормали), текстуры ландшафта, неба, воды. Также этот класс отвечает за визуализацию всех этих объектов.

DynamicCamera – реализация камеры, ее передвижения, вращения, а также связь с ландшафтом.

Font – модуль для инициализации и вывода текста на экран.

Render – модуль для реализации визуализации в 3-мерном режиме.

Image – модуль для загрузки и обработки изображений.

Texture – модуль для загрузки изображений и инициализаций структур.

Vector – модуль для работы с векторами (сложение, вычитание, скалярное произведение, векторное произведение).

Matrix – модуль для работы с матрицами (умножение, транспонирование, нахождение обратной).

Матрицы используются для реализации композиции трансформаций (перемещение, вращение, поворот), для реализации геометрических алгоритмов.

3.3 Входные и выходные данные

Входными данными для данной программы является карта высот, карта освещенности и текстура ландшафта, 5 кубических текстур неба и текстура воды в графических файлах в формате “.bmp”, также файл конфигурации, где указаны пути к этим ресурсам. В нем содержится вся информация, необходимая для работы программы. Так же при изображении ландшафта принимаются данные с клавиатуры и мышки, которые интерпретируются как команды пользователя, и в зависимости от них строится последующее изображение.

Выходными данными программы является анимационный ряд с изображением трехмерного

ландшафта, построенного и отображенного на основании входных данных. Также выходными данными является информация о количестве кадров, выводимых на экран за секунду.

3.4 Интерфейс программы

Ниже представлен интерфейс полученного программного обеспечения.

3.4.1 Интерфейс программы «Editor»

Интерфейс программы «Editor» состоит из нескольких блоков:

1. моделирование;
2. освещение;
3. параметры генерации;
4. предпросмотр
5. подсчет времени.

В блоке «Моделирование» пользователю представляется возможность загрузить данные из файла или сгенерировать их самому. Также пользователь может выбрать из ранее подготовленных данных, текстуру неба и воды. Комплекс «Освещение» можно создать карту освещенности с различно настроенными источниками света в количестве не более 5. Для каждого источника света также можно настраивать цвет фонового освещения, диффузного освещения, вектор направления (или позицию с коэффициентами угасания $c_1c_2c_3$ для точечных источников). Блок «Параметры генерации» отвечает за выбор типа ландшафта, размер, материал.

Также пользователю предлагается выбор метода генерации(Холмовой, шум Перлина). Как дополнительными параметрами можно выбрать каким будем итоговая местность: островом или долиной. Также предлагается сгладить ландшафт, а именно сделать его более пологим. Выбрав все выше перечисленные параметры можно воспользоваться отделом *предпросмотр* полученного ландшафта. Как бонус пользователю выводится итоговое время ушедшее для генерации карты высот, освещения и текстуры. На Рисунках 23 и 24 представлен интерфейс модуля «Editor»:

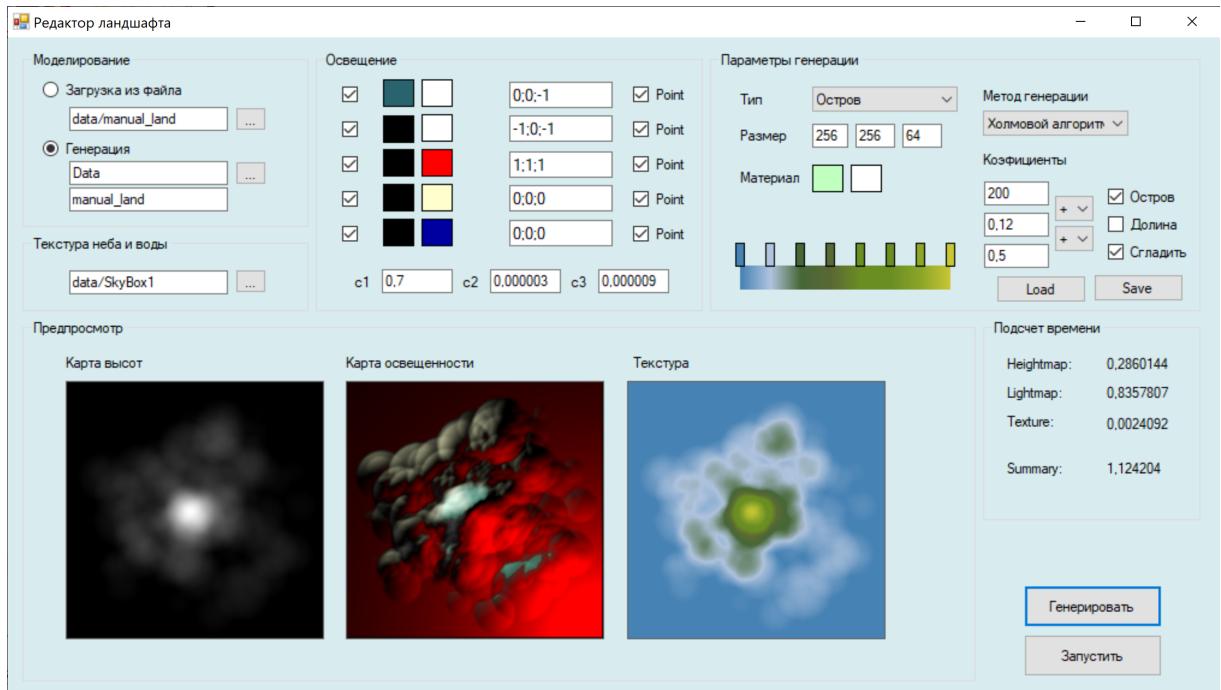


Рис. 23: Интерфейс программы «Editor»

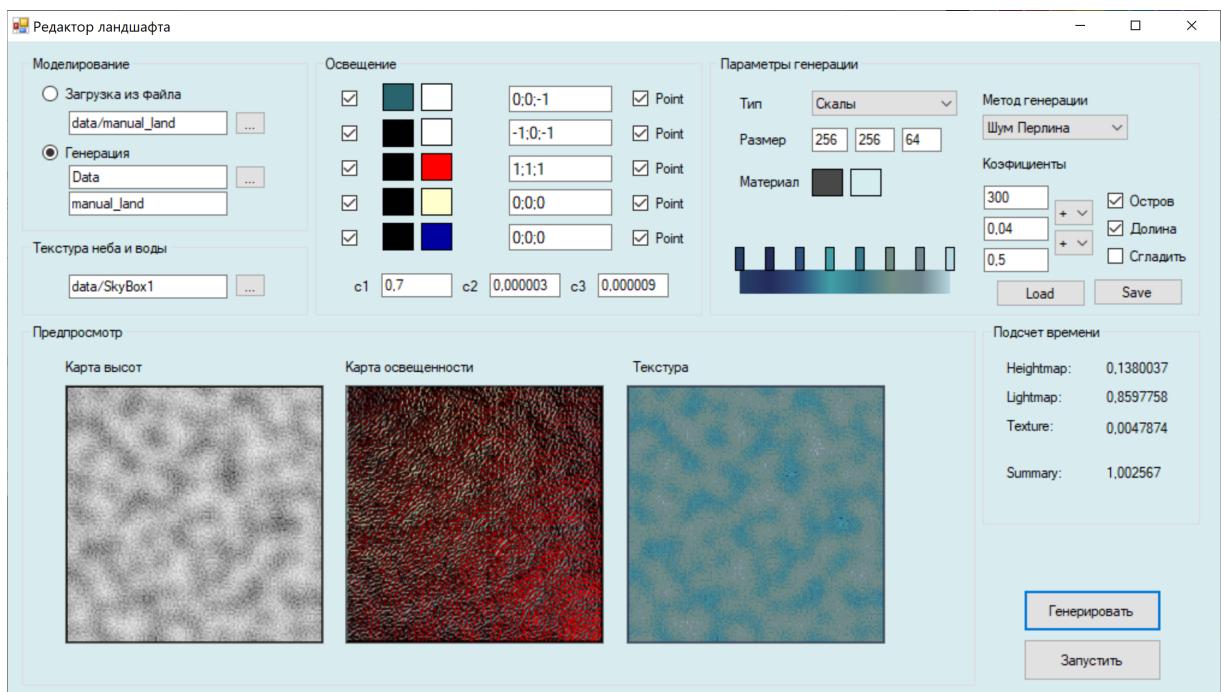


Рис. 24: Интерфейс программы «Editor»

3.4.2 Интерфейс программы «Viewer»

Эта программа позволяет просматривать ландшафты в интерактивном режиме, построенные из ранее сгенерированных карты высот, текстур и карт освещенности. Также в этой программе есть поддержка неба, водной поверхности, отражений от воды.

Интерфейс этой программы довольно простой, что дает возможность любому пользователю разобраться за считанные секунды.

W, A, S, D – навигация по пространству

[] – увеличение/уменьшение уровня моря.

z, x, c – включение/отключение соответственно текстуры, карты освещенности и карты деталей.

v – привязка камеры к поверхности ландшафта (нельзя будет опуститься под поверхность) На

Рисунке 25 представлен интерфейс модуля «Viewer»:

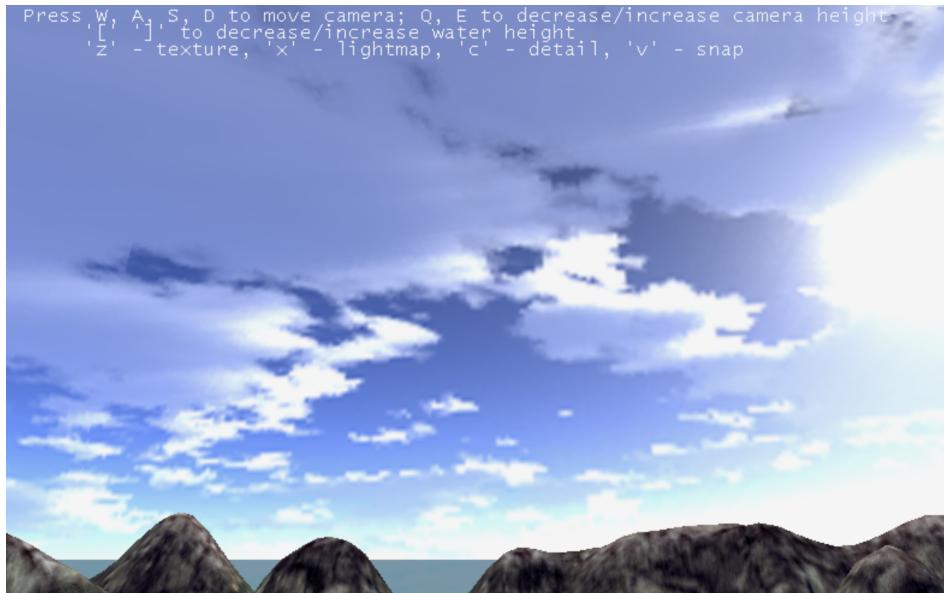


Рис. 25: Интерфейс программы «Viewer»

3.5 Вывод из технологической части

В данном разделе было описано и обосновано выбор языка и средств реализации данного проекта. Также была предоставлена блок-схема и описана структура полученного ПО. Был продемонстрирован внутренний интерфейс программы: «Editor», «Viewer».

4 Исследовательская часть

В данном разделе будет проведен эксперимент и сравнительный анализ. Также будут показаны примеры работы программы

4.1 Системные характеристики

Характеристики компьютера на котором проводился эксперимент:

1. операционная система - Windows 10;
2. процессор - Intel(R) Core(TM) i7-10510U CPU @1.80GHz 2.30GHz;
3. объем оперативной памяти - 16 ГБ;
4. количество ядер - 4;
5. количество логических процессов - 8;
6. видеокарта - NVIDIA GeForce GTX 1650 with Max-Q Design;
7. объем видеокарты - 4 ГБ.

4.2 Постановка эксперимента

В рамках данного проекта были проведены эксперименты, описанные ниже:

1. сравнение и анализ времени построения карты освещенности от количества источников света;
2. сравнение и анализ времени построения карты освещенности от ее размера;
3. сравнение и анализ числа кадров в секунду, генерируемых программой, от числа граней с наложением карт освещения и без него.

4.3 Сравнительный анализ на основе замеров времени работы программы

На рисунке 26 показаны результаты первого эксперимента, суть которого заключается в анализе времени построения карты освещенности от количества источников света. Количество источников света изменяется от 1 до 5. Для данного эксперимента было выбрано 4 размера карты освещенности:

1. 128x128;
2. 256x256;
3. 512x512;
4. 1024x1024.

Ниже приведена полученная диаграмма:

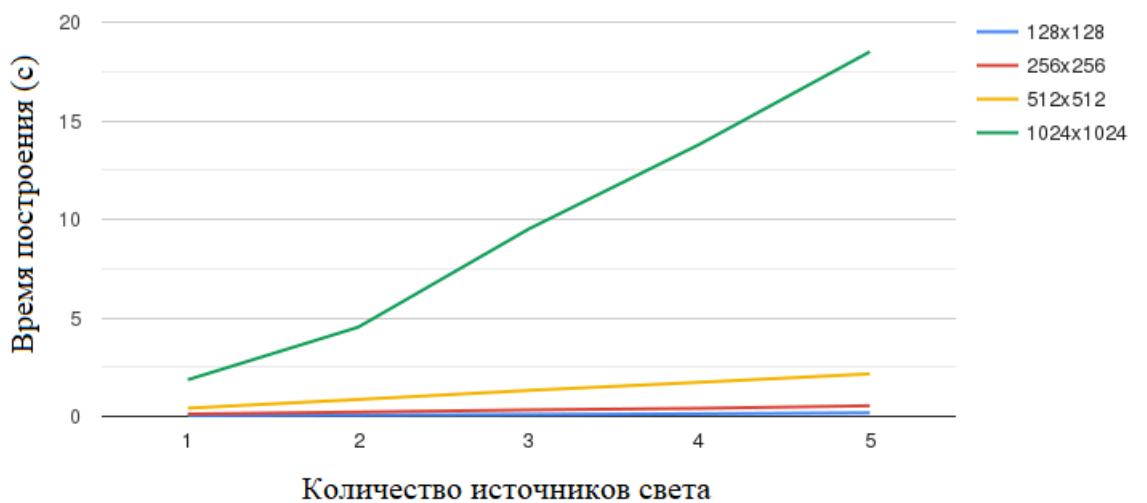


Рис. 26: Сравнение времени построения карты освещенности от количества источников света.

Из данного графика видно, что скорость просчета карты освещенности прямо пропорциональна количеству источников света на сцене. Это соответствует и теоретическим расчетам: каждый пиксель карты освещенности обрабатывается от одного источника освещения только один раз.

На рисунке 27 показаны результаты второго эксперимента, суть которого заключается в анализе времени построения карты освещенности от ее размера. В данном эксперименте размер карты варьируется в следующих значениях:

1. 128x128;
2. 256x256;
3. 512x512;
4. 1024x1024;
5. 2048x2048.

Ниже приведена полученная диаграмма:



Рис. 27: Сравнение времени построения карты освещенности от ее размера.

Из данной зависимости становится понятным, что затрачиваемое время на просчет карты освещенности 2^N , где N – размер карты.

Третий эксперимент был проведен, для того чтобы сделать анализ зависимости числа кадров в секунду, генерируемых программой, от числа граней с наложением карт освещения и без него, для двух способов наложения карт освещения – двухпроходного рисования и мультитекстурирования. На Рисунках 28 и 29 показаны двухпроходное рисование и мультитекстурирование соответственно:

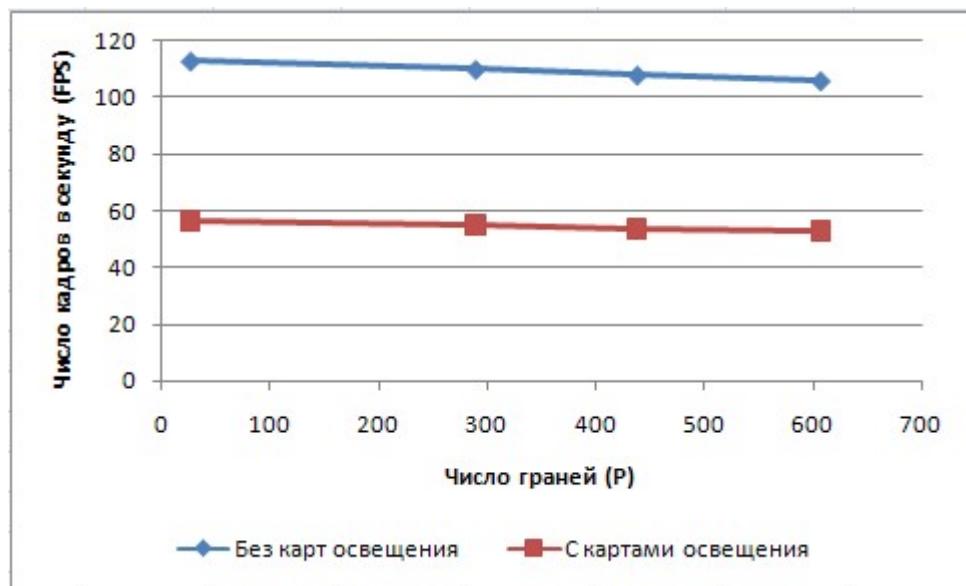


Рис. 28: Двухпроходное рисование.

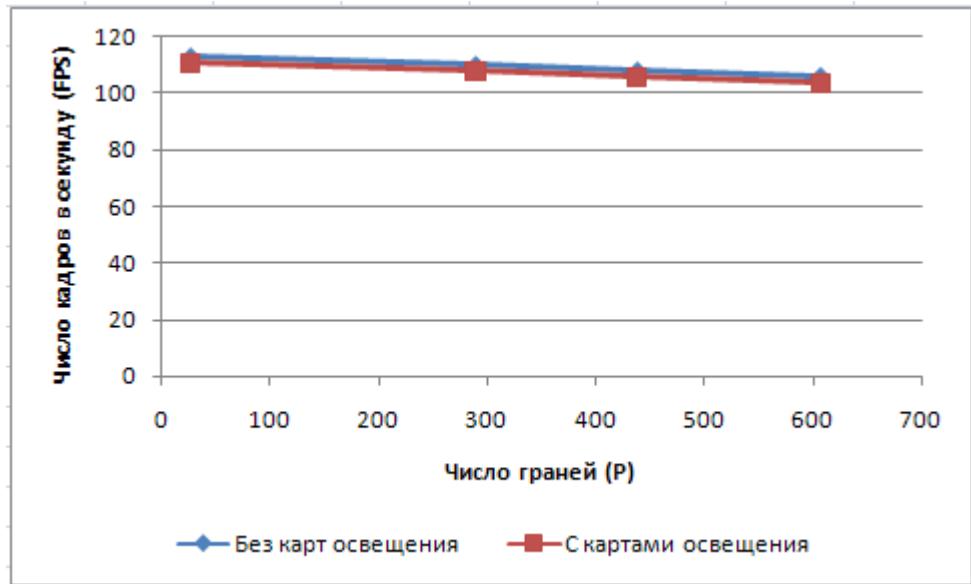


Рис. 29: Мультитекстурирование.

Суть двухпроходного рисования в том, что каждая грань рисуется два раза – один раз с основной текстурой, второй раз с картой освещения в режиме «смешивания» - поэтому скорость рисования сцены падает в два раза. *Мультитекстурирование* – операция, реализованная аппаратно и позволяющая наложить несколько текстур в один проход, поэтому скорость рисования сцены падает очень незначительно. В обоих случаях число кадров в секунду выше требуемого для интерактивных программ уровня в 25 кадров/с.

На Рисунке 30 показано сравнение использования карт освещения с простой моделью освещения, реализованной программно (освещенность вычисляется для каждой грани в момент рисования):

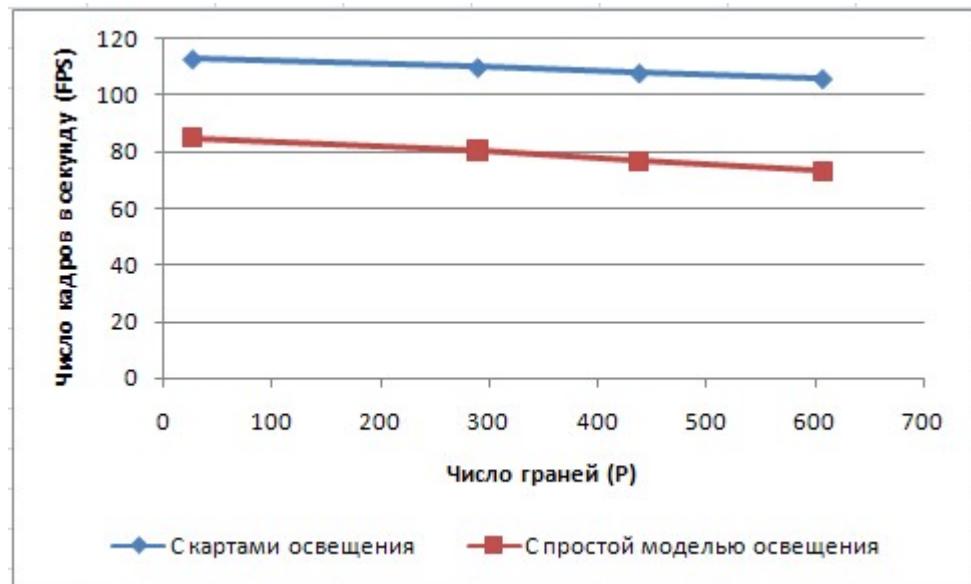


Рис. 30: Сравнение использования карт освещения с простой моделью освещения.

Как видно из Рисунка 30, использование карт освещения опережает по скорости даже простую модель освещения, позволяя при этом получить гораздо более реалистичное изображение.

4.4 Примеры работы программы

Ниже на Рисунках 31 и 32 показаны примеры работы программы:

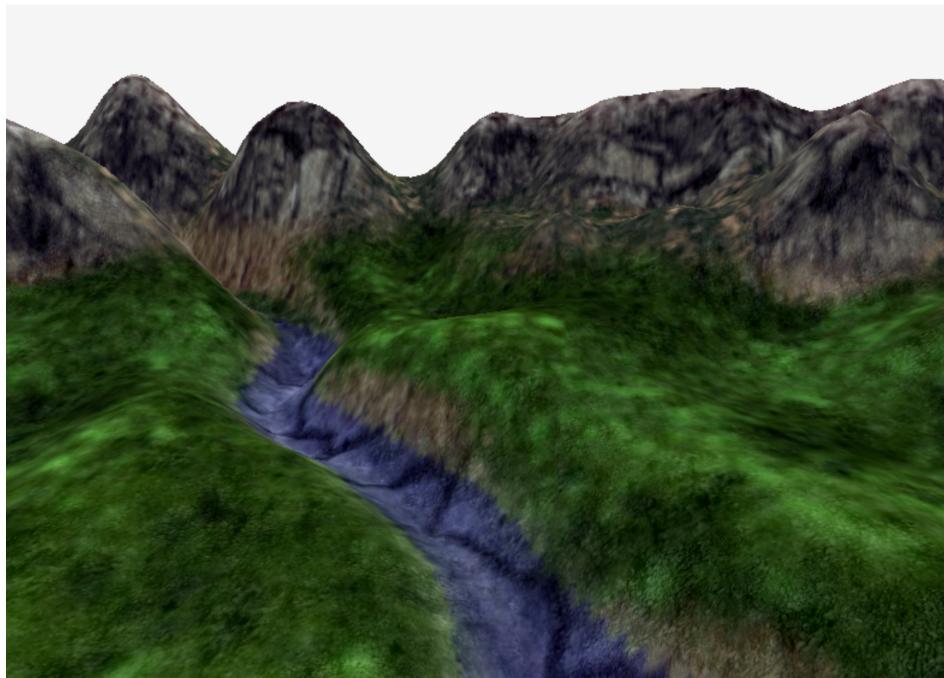


Рис. 31: Пример работы программы без текстуры неба.

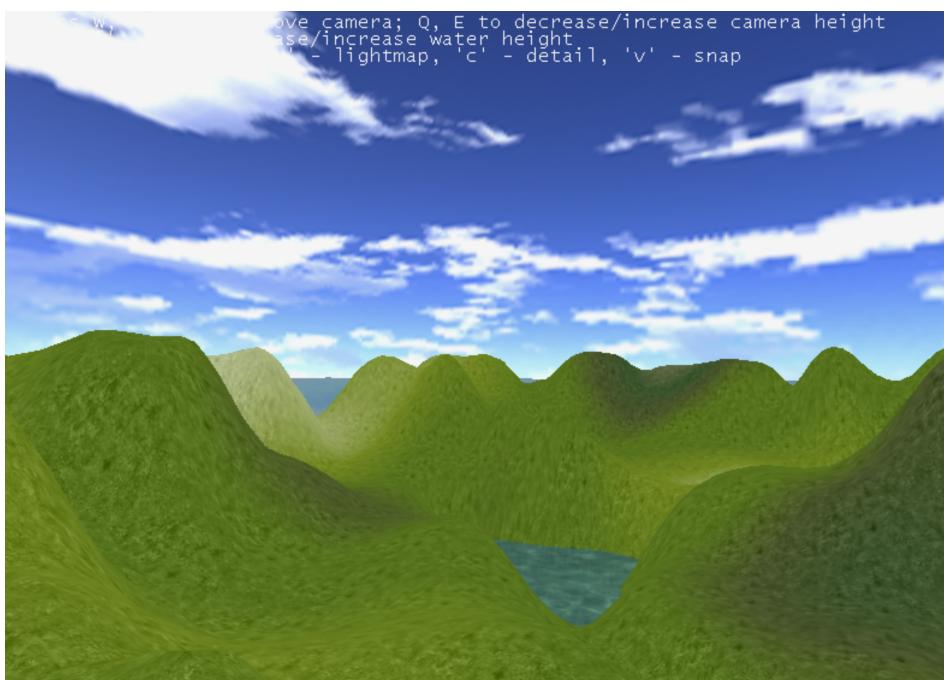


Рис. 32: Пример работы программы с текстурами воды и неба.

4.5 Вывод из исследовательской части

В данном разделе были показаны примеры работ программы. Также были продемонстрированы результаты исследования зависимости времени создания карты освещенности от количества источников света. Это исследование показало нам, что скорость просчета карты освещенности прямо пропорциональна количеству источников света на сцене. Это соответствует и теоретическим расчетам: каждый пиксель карты освещенности обрабатывается от одного источника освещения только один раз. Из второго эксперимента становится понятным, что затрачиваемое время на просчет карты освещенности 2^N , где N – размер карты.

Заключение

Разработанный программный комплекс обеспечивает возможность генерации или загрузки карт высот, карты освещенности и текстуры. Также он позволяет строить изображения ландшафта с наложением текстуры в реальном времени. Проведены исследования работы алгоритмов на различных наборах исходных данных.

Результаты работы программы можно использовать для рисования освещенной сцены в реальном времени, с такой высокой скоростью, которая невозможна без использования предварительного расчета освещения. Получаемые изображения обладают высокой степенью реалистичности. Благодаря тому, что форматы результатов (набор карт освещения) являются широко распространенными в современных приложениях трехмерной графики, программу можно легко использовать на практике для решения реальных задач.

Тем не менее, существует множество путей усовершенствования описанного программного комплекса:

1. генерируемые текстуры ландшафта все-таки выглядят не очень реалистично, из-за того, что на них не накладывается текстура грунта, травы;
2. в данной программе нет генерируемых растительности и деревьев, а также смена времени суток. Внесение данных нововведений, несомненно, добавило бы реалистичности в генерируемый мир и расширило бы его возможности.

Список литературы

- [1] Scott Turner. *Шум Перлина, процедурная генерация контента и интересное пространство*/Э.Л. РЕСУРС] Режим доступа: URL: <https://habr.com/post/440286/>. (дата обращения: 07.10.2020).
- [2] Wikipedia. *RGB*/Э.Л. РЕСУРС] Режим доступа: URL: <https://ru.wikipedia.org/wiki/RGB>. (дата обращения: 01.11.2020).
- [3] Кулагин Денис. *Модель отражения Фонга*/Э.Л. РЕСУРС] Режим доступа: URL: https://compgraphics.info/3D/lighting/phong_reflection_model.php. (дата обращения: 12.10.2020).
- [4] Мир Знаний. *Мипмапы*/Э.Л. РЕСУРС] Режим доступа: URL: <https://smekni.com/a/311940-4/generatsiya-i-postroenie-izobrazheniy-landshafta-v-realnom-vremeni-4/>. (дата обращения: 01.11.2020).
- [5] Денис Кожухов. *Генерация трехмерных ландшафттов*/Э.Л. РЕСУРС] Режим доступа: URL: <https://www.ixbt.com/video/3dterrains-generation.shtml>. (дата обращения: 10.09.2020).
- [6] Вадим Репин. *Иерархический Z-буфер*/Э.Л. РЕСУРС] Режим доступа: URL: <https://www.ixbt.com/video/hierar-z.html>. (дата обращения: 12.10.2020).
- [7] Светлана Шляхтина. *Генераторы ландшафттов* /Э.Л. РЕСУРС] Режим доступа: URL: <https://compress.ru/article.aspx?id=16597>. (дата обращения: 27.11.2020).