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)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Расчетно-пояснительная записка

к курсовому проекту

Тема: Генерация трехмерного ландшафта

Дисциплина: Компьютерная графика

Студент	ИУ7-55Б		П.К Хетагуров
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Руководитель проекта			В.П. Степанов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Содержание

Введение **4**

1 Аналитическая часть **5**

1.1 Генерирование данных о структуре ландшафта 5

1.1.1 Представление данных о ландшафте 5

1.1.1.1 Карта высот 5

1.1.1.2 Вершины и связи между ними 6

1.1.1.3 Карта сегментов ландшафта 6

1.1.1.4 Сравнение 6

1.1.1.5 Вывод 6

1.1.2 Алгоритмы генерации карты высот 6

1.1.2.1 Шум Перлина 7

1.1.2.2 Холмовой алгоритм 7

1.1.2.3 Diamond-square алгоритм 8

1.1.2.4 Сравнение 8

1.1.2.5 Вывод 9

1.2 Построение трехмерного изображения на основе карты высот 9

1.2.1 Разбиение на полигоны 9

1.2.2 Удаление невидимых граней и поверхностей 9

1.2.2.1 Алгоритм, использующий Z-буфер 10

1.2.2.2 Алгоритм художника 10

1.2.2.3 Трассировка лучей 10

1.2.2.4 Сравнение 10

1.2.2.5 Вывод 11

1.2.3 Затенение 11

1.2.3.1 Затенение по Ламберту 12

1.2.3.2 Затенение по Гуро 12

1.2.3.3 Затенение по Фонгу 12

1.2.3.4 Сравнение 12

1.2.3.5 Вывод 13

1.3 Постановка задачи 13

1.4 Вывод 13

2 Конструкторская часть **14**

2.1 Архитектура программного обеспечения 14

2.1.1 Модуль генерации ландшафта 14

2.1.2 Модуль камеры . . . 15
2.1.2.1 Преобразование координат объектов в координаты наблюдателя . . . 15
2.1.2.2 Перспективное преобразование . . . 16
2.1.3 Модуль затенения . . . 16
2.1.4 Модуль рендеринга сцены . . . 16
2.1.5 Модуль пользовательского интерфейса . . . 16
2.2 Вывод . . . 16

3 Технологическая часть **18**
3.1 Выбор языка программирования . . . 18
3.2 Интерфейс . . . 18
3.3 Ограничения . . . 19
3.4 Листинги программы . . . 19
3.5 Вывод . . . 20

4 Экспериментальная часть **21**
4.1 Использованное оборудование . . . 21
4.2 Исследование времени обработки кадра . . . 21
4.3 Пример работы программы . . . 22
4.4 Вывод . . . 23

Заключение **24**

Список литературы **25**

Введение

Компьютерные системы уже глубоко проникли во все сферы жизни и являются неотъемлемыми составляющими человеческой деятельности. Организация работы предприятий, проектирование ракет, моделирование химических процессов - все это значительно облегчилось после широкого распространения компьютеров.

При все большем усложнении информационных систем и развитии компьютерной техники, росли и требования к таким системам. Очень быстро появилась потребность в визуализации данных, полученных или обработанных с помощью уже существующего программного обеспечения. Ответом на эту потребность стала машинная графика - область компьютерной науки, отвечающая за обработку, синтез и распознавание изображений. В частности выделилось такое направление машинной графики, как 3D-моделирование, отвечающее за синтез и обработку изображений объемных объектов.

В настоящее время существует большое количество задач, решаемых с помощью 3D-моделирования. Например, высокоточное моделирование деталей и объектов, добавление спецэффектов при производстве фильмов, визуализация объектов в компьютерных играх. Одной из таких задач является генерация трехмерного ландшафта.

Целью работы является создание программного продукта, позволяющего генерировать трехмерные модели ландшафта.

Для достижения поставленной цели необходимо решить следующие задачи:

1. проанализировать методы генерирования ландшафтов;
2. спроектировать программное обеспечение, позволяющее генерировать трехмерный ландшафт;
3. реализовать спроектированное программное обеспечение.

1 Аналитическая часть

В данном разделе будут предъявлены требования к разрабатываемому программному обеспечению, будут рассмотренны основные теоретические сведения связанные с трехмерной генерацией ландшафта, проанализированы существующие решения.

Целью работы является создание программного продукта, позволяющего генерировать трехмерные модели ландшафта.

Задачу генерации трехмерного ландшафта можно решать различными способами, но все из них можно разделить на следующие этапы:

1. генерация данных о структуре ландшафта;
2. построение трехмерного изображения по сгенерированным данным.

Рассмотрим первый этап.

1.1 Генерирование данных о структуре ландшафта

Сначала необходимо выбрать способ представления данных о ландшафте, ведь в зависимости от этого будут варьироваться алгоритмы генерации.

1.1.1 Представление данных о ландшафте

Существуют различные способы представления данных о ландшафте, вот некоторые из них:

1. карта высот (height map);
2. вершины и связи между ними;
3. карта сегментов ландшафта.

Рассмотрим каждый из них подробней.

1.1.1.1 Карта высот

Представление данных в виде карты высот предполагает создание двухмерного масива, являющегося 'снимком' высоты местности. Каждый элемент массива представляет собой число, являющееся высотой в точке с координатами x и y, равными индексам в двухмерном массиве с некоторым смещением.

Минусом данного способа представления данных является избыточность при представлении некоторых поверхностей. Например, для описания простой плоскости можно использовать всего 3 точки. Плюсами же являются простота представления и изменения данных, наглядность, возможность быстро определить высоту в заданной точке. Этот формат хранения является наиболее распространенным и поэтому существует возможность работать с картой высот в других программах [1].

1.1.1.2 Вершины и связи между ними

Если хранить информацию о ландшафте в виде набора вершин и связей между ними, то основным преимуществом, по отношению к предыдущему случаю, будет хранение существенно меньшего количества информации. Но этот способ лишается преимуществ, которыми обладало хранение информации в виде карты высот. Так, изменение и просмотр данных, представленных таким образом, требует специализированного программного обеспечения, а основные алгоритмы построения ландшафта требуют адаптации, так как заточены на использование карты высот [2].

1.1.1.3 Карта сегментов ландшафта

Карта сегментов ландшафта - двухмерный массив, похожий на карту высот, но определяющий не высоту в конкретной точке, а номер созданного заранее блока ландшафта. Плюсом данного способа является снижение требуемых данных для хранения, ведь ландшафтные блоки могут быть большого размера. Минусами являются необходимость прегенерации большого количества различных ландшафтных блоков и проблема совместимости этих блоков между собой [3].

1.1.1.4 Сравнение

В таблице 1 представлено общее сравнение вышеперечисленных способов представления ландшафтов.

Таблица 1 – Сравнение способов представления данных о ландшафте

	Наличие сторонних программ редактирования	Отсутствие необходимости адаптации алгоритмов генерации	Отсутствие избыточности данных
Карта высот	+	+	-
Иррегулярная сетка вершин	-	-	+
Карта сегментов ландшафта	-	-	-

1.1.1.5 Вывод

Из вышесказанного можно сделать вывод, что представление в виде карты высот наиболее простое и применимо в большинстве случаев. Для дальнейших рассуждений выберем его.

1.1.2 Алгоритмы генерации карты высот

Второй этап в генерации данных о структуре ландшафта - собственно генерация. Существуют различные способы генерации карты высот, вот некоторые из них:

1. шум Перлина;
2. холмовой алгоритм;
3. diamond-square алгоритм.

Рассмотрим каждый из них подробней.

1.1.2.1 Шум Перлина

При применении шума Перлина вся площадь карты накрывается сеткой. В каждом узле сетки строится случайный двухмерный вектор единичной длины, указывающий в случайном направлении. Таким образом каждый пиксель изображения попадает в случайную клетку, вектора в углах которой указывают в случайном направлении. Далее создаются четыре вектора, соединяющие углы ячейки сетки с данным пикселем. После этого для каждой пары векторов, выходящих из одного угла, находится скалярное произведение. Полученные четыре значения надо объеденить, для получения высоты в заданном пикселе. Наиболее распространена интерполяция этих четырех значений в зависимости от близости пикселя к углам сетки. При увеличении количества узлов сетки, получается более частый шум, похожий на белый, а при уменьшении наоборот, более гладкий [4].

На рисунке 1 представлены векторы, о которых шла речь.

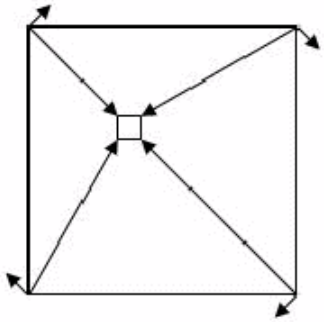

Рисунок 1 – Определение высоты в точке внутри ячейки сетки

Поверх уже сгенерированного шума можно наложить ещё один шум, меньшей частоты и более частой сетки, что придаст ландшафту реалистичный вид. Этот способ позволяет легко повторить генерацию, достаточно знать только вектора в каждом угле сетки.

1.1.2.2 Холмовой алгоритм

Изначально считается, что все точки находятся на одной высоте, в дальнейшем в произвольных местах задаются холмы различной высоты и диаметра. В результате наложения холмов друг на друга получается ландшафт [5].

1.1.2.3 Diamond-square алгоритм

Этот алгоритм рекурсивен. Изначально вся карта накрывается одной ячейкой и каждой вершине этой ячейки случайным образом присваивается высота. Далее происходит этап square - в квадрате определяется центральная точка, путем усреднения угловых точек и добавления случайного отклонения. Далее происходит этап diamond - определяются высоты точек, лежащих в серединах сторон. Для этого используются как точки, лежащие сверху и снизу, так и точки справа и слева, найденные на шаге square. Далее происходит рекурсивное повторение этого алгоритма для каждого квадрата. Заметим, что так как в шаге diamond используются точки, полученные на шаге square, то шаг square проводится для всех текущих квадратов, так же, как и шаг diamond для всех получившихся ромбов. Для простоты применения этого алгоритма необходимо использовать квадратную карту со стороной, равной степени двойки [6].

На рисунке 2 визуально представлена одна итерация этого алгоритма.

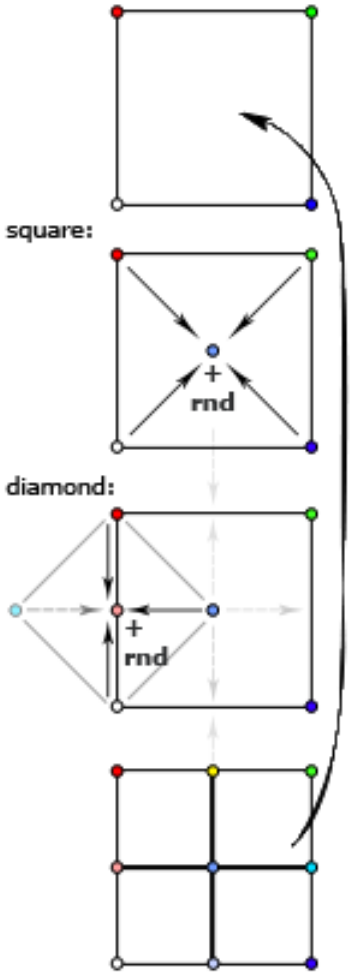

Рисунок 2 – Алгоритм diamond-square

1.1.2.4 Сравнение

В таблице 2 представлено общее сравнение вышеперечисленных способов генерации карты высот.

Таблица 2 – Сравнение способов генерации карты высот

	Возможность влиять на генерацию	Возможность повторения генерации
Шум Перлина	+	+
Холмовой алгоритм	-	-
Diamond-square	+	-

1.1.2.5 Вывод

Шум Перлина предоставляет более гибкую настройку, позволяя накладывать несколько шумов разной частоты и меняя шаг сетки. Для получения такого же предсказуемого эффекта при генерации другими алгоритмами потребуется дополнительная обработка шумом Перлина, поэтому было принято решение изначально использовать шум Перлина.

1.2 Построение трехмерного изображения на основе карты высот

Построение изображения на основе карты высот можно разделить на следующие этапы:

1. разбиение на полигоны;
2. удаление невидимых граней и поверхностей;
3. затенение.

Рассмотрим каждый этап.

1.2.1 Разбиение на полигоны

Самым простым многоугольником является треугольник и, как следствие, с первого взгляда было бы логично разбить модель на треугольные полигоны. Но у треугольных полигонов есть несколько недостатков. Так, треугольные полигоны хуже деформируются и могут создавать артефакты, что, при условии совместимости со сторонними редакторами, может привести к нежелательным сложностям. Четырехугольные же полигоны, помимо легкости в деформации, позволяют уменьшить количество вычислений без существенного снижения качества [7].

Было принято решение использовать четырехугольные полигоны.

1.2.2 Удаление невидимых граней и поверхностей

Рассмотрим некоторые алгоритмы удаления невидимых граней и поверхностей:

1. алгоритм, использующий Z-буфер;
2. алгоритм художника;
3. метод трассировки лучей.

Рассмотрим каждый из них.

1.2.2.1 Алгоритм, использующий Z-буфер

Алгоритм работает в пространстве изображения. Идея алгоритма крайне проста. Создается буфер, хранящий z-координату каждого пикселя изображения, и заполняется минимальным значением. При обработке очередного пикселя, значение его z-координаты сравнивается с хранящимся в буфере и, если его координата больше хранящейся, то её значение заносится в буфер, а сам пиксель выводится на экран, закрашивая предыдущий [8].

Заметим, что размер буфера будет равен размеру пространства изображения. Тем самым, память, требуемая для работы алгоритма, определяется размерами изображения. Основным плюсом данного алгоритма является простота реализации и возможность обрабатывать полигоны объекты в произвольном порядке.

1.2.2.2 Алгоритм художника

Идея алгоритма заключается в изображении объектов, начиная с самых удаленных от наблюдателя. Для использования алгоритма художника сначала необходимо провести сортировку объектов по глубине и занести их в список приоритетов. В окончательном списке никакие из двух элементов не должны перекрывать друг друга. После этого объекты отображаются, начиная с самого удаленного от наблюдателя [9].

При использовании данного алгоритма существует сильная зависимость скорости работы от взаимного расположения объектов. Так, при циклическом перекрытии многоугольников используются сложные тесты, для определения ближайщего к наблюдателю. В худшем случае многоугольники необходимо разбивать на несколько новых.

1.2.2.3 Трассировка лучей

В этом методе через каждый пиксель картинной плоскости выпускается луч. Из всех граней, с которыми этот луч пересекся, выбирается ближайшая, и точка пересечения отображается.

С помощью данного метода возможно построения высококачественных изображений, алгоритм легко распараллеливается, вычислительная сложность слабо зависит от сложности сцены. Существенным же минусом является низкая производительность [10].

1.2.2.4 Сравнение

В таблице 3 представлено сравнение вышеперечисленных алгоритмов.

Таблица 3 – Сравнение алгоритмов визуализации

	Маленькая вычислительная сложность	Легкость реализации	Работа с произвольными входными данными
Алгоритм, использующий Z-буфер	+	+	+
Алгоритм художника	+	+	-
Трассировка лучей	-	-	+

1.2.2.5 Вывод

Из проведенного сравнения видно, что наиболее оптимальным алгоритмом, в отсутствии требования к фотореалистичному изображению, является алгоритм, использующий Z-буфер.

1.2.3 Затенение

В моделях затенения освещенность складывается фоновой, диффузной и зеркальной составляющих [11].

Фоновая составляющая - некоторая константа освещения, присутствующая в каждой точке, вне зависимости от расположения точки и источника света.

Дифузная составляющая - это компонента, отвечающая за рассеивание света после попадания на поверхность.

Зеркальная составляющая отвечает за моделирование отражающих свойств материала. Её добавление позволяет отобразить блики на поверхности.

На рисунке 3 визуально представлены все компоненты затенения и получившееся изображения.

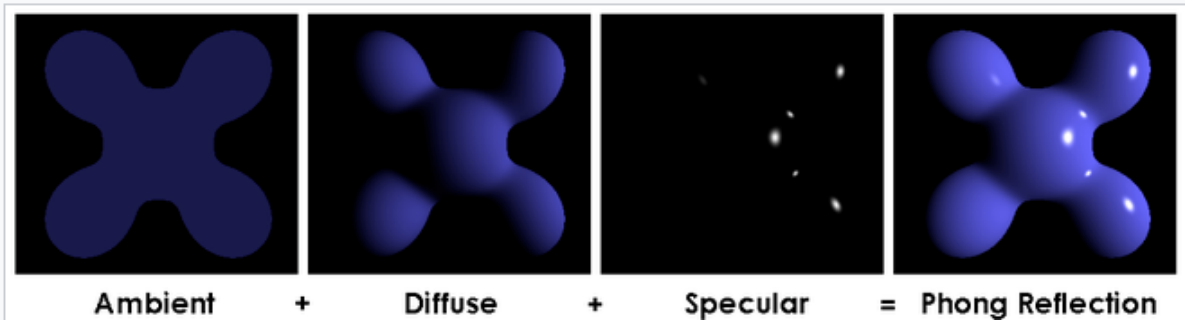

Рисунок 3 – Компоненты затенения

Основные модели затенения:

1. затенение по Ламберту;
2. затенение по Гуро;
3. затенение по Фонгу.

Рассмотрим каждый из них подробнее.

1.2.3.1 Затенение по Ламберту

Модель затенения по Ламберту является одной из самых простейших и позволяет описать идеальное диффузное освещение. Если N - вектор нормали в некоторой точке, L - вектор, направленный на источник света, а $I_{ambient}$ - фоновая составляющая, то освещённость по модели Ламберта считается по формуле (1):

$$I = I_{ambient} + max(0, dotProduct(N, L)) \tag{1}$$

Затенение по Ламберту предполагает выбор случайной точки на полигоне и экстраполирование цвета этой точки на весь полигон, что создает явный полигональный характер изображения [12].

1.2.3.2 Затенение по Гуро

При затенении по Гуро сначала считаются интенсивности света в каждой вершине, а после используется билинейная интерполяция интенсивностей, что убирает явную дискретность освещения. В этом методе может быть неверно определена зеркальная составляющая, так как происходит усреднение векторов нормалей. В методе гуро учитывается зеркальная составляющая, что отражено в формуле (2), использующейся для подсчета интенсивности [13].

$$I = I_{ambient} + kd * max(0, dotProduct(N, L)) + ks * max(0, dotProduct(N, \frac{L+V}{|L+V|}))^{SpecularPower}, \tag{2}$$

где N, L, $I_{ambient}$ тоже, что и в затенении по Ламберту, kd и ks - коэффициенты рассеивания и зеркального освещения соответственно, V - вектор, направленный из точки к наблюдателю, SpecularPower - коэффицент блеска материала.

1.2.3.3 Затенение по Фонгу

Формула для расчета интенсивности при затенении по Фонгу совпадает с формулой (2), но происходит не интерполирование интенсивностей, а интерполирование нормалей, что дает более точный учет зеркальной составляющей. Изображение получается более реалистичным, чем в методе Гуро, но трудоемкость также повышается [11].

1.2.3.4 Сравнение

В таблице 4 представлено сравнение вышеперечисленных моделей затенения.

Таблица 4 – Сравнение моделей затенения

	Маленькая вычислительная сложность	Устранение дискретизации закраски	Учет зеркальной составляющей
Затенение по Ламберту	+	-	-
Затенение по Гуро	+	+	+
Затенение по Фонгу	-	+	+

1.2.3.5 Вывод

Из рассмотренных моделей затенения была выбрана модель затенения по Гуро, из-за небольшой вычислительной сложности и учета зеркальной составляющей.

1.3 Постановка задачи

Разработать программное обеспечение, позволяющее генерировать и отображать ландшафт, представленный в виде карты высот. При генерации использовать шум Перлина. Дать возможность пользователю влиять на генерируемый ландшафт, путем изменения количества узлов сетки и добавления шумов меньшей частоты поверх сгенерированного ландшафта. При отображении использовать алгоритм Z-буфера и затенение по Гуро.

1.4 Вывод

В данном разделе были рассмотренны и проанализированы существующие алгоритмы решения задачи генерации и отображения ландшафта, уточнены требования к разрабатываемому ПО.

2 Конструкторская часть

В данном разделе будет проведено проектирование программного обеспечения.

Разрабатываемое приложение должно решать задачу синтеза динамического изображения, из чего вытекают следующие этапы решения:

- генерация данных об объекте на основе введенных данных;
- задание положения наблюдателя, картинной плоскости, размеров окна вывода, значений управляющих сигналов;
- преобразования координат объектов в координаты наблюдателя;
- отсечение объектов сцены по границам пирамиды видимости;
- вычисление двумерных перспективных проекций объектов на картинную плоскость;
- удаление невидимых линий и поверхностей при заданном положении наблюдателя;
- закрашивание и затенение видимых объектов сцены;
- вывод полученного полутонового изображения на экран растрового дисплея.

2.1 Архитектура программного обеспечения

В соответствии с этапами синтеза изображения были выделены следующие модули программы:

- модуль генерации ландшафта;
- модуль камеры;
- модуль затенения;
- модуль рендеринга сцены;
- модуль пользовательского интерфейса.

Рассмотрим каждый из модулей подробнее.

2.1.1 Модуль генерации ландшафта

Для генерации карты высот был выбран шум Перлина [14]. На первом этапе генерируется карта высот:

1. создается сетка размерами [i, j], где i - кол-во узлов сетки по вертикали, а j - по горизонтали;
2. каждому узлу сетки ставится в соответствие случайный еденичный вектор;
3. далее, для каждой точки карты высот выполняется следующее:

3.1. определяется, в какую ячейку сетки попала точка;

3.2. считаются скалярные произведения между вершинами ячейки и точкой;

3.3. происходит интерполяция полученных значений;

3.4. происходит нормализация значения (отображение в отрезок [0, 1]).

На втором этапе происходит расчет нормалей в каждой точке карты высот, построение полигонов и присвоение им базового материала (трава, вода, лед и т.д), который определяет базовый цвет и отражающие способности. На втором этапе идет подготовка данных для последующей закраски.

2.1.2 Модуль камеры

Для реализации функционала просмотра сцены реализуется камера. Данный модуль предоставляет возможность перемещение камеры, преобразование координат объектов в координаты наблюдателя, вычисление двумерных перспективных проекций объектов на картинную плоскость и отсечение объектов по границе пирамиды видимости.

2.1.2.1 Преобразование координат объектов в координаты наблюдателя

Для реализации данного преобразования была выбрана матрица LookAt [15]. Для построения этой матрицы необходимо знать следующее:

1. текущее положение камеры (радиус-вектор place);
2. точка, на которую направлен взгляд (радиус-вектор lookAt);
3. вектор, показывающий крен камеры (вектор up).

Если все из этого известно, то происходит вычисление новых координатных осей по формулам (3).

$$\begin{aligned} \overline{z} &= \frac{\overline{lookAt} - \overline{place}}{|\overline{lookAt} - \overline{place}|} \\ \overline{x} &= \frac{[\overline{up} \times \overline{z}]}{|[\overline{up} \times \overline{z}]|} \\ \overline{y} &= \frac{[\overline{z} \times \overline{x}]}{|[\overline{z} \times \overline{x}]|} \end{aligned} \tag{3}$$

Далее происходит формирование матрицы преобразования по формуле (4):

$$lookAtMatrix = \begin{pmatrix} x.X & y.X & z.X & 0 \\ x.Y & y.Y & z.Y & 0 \\ x.Z & y.Z & z.Z & 0 \\ -(\overline{x} * \overline{place}) & -(\overline{y} * \overline{place}) & -(\overline{z} * \overline{place}) & 1 \end{pmatrix} \tag{4}$$

Применение данной матрицы преобразует координаты объектов в координаты наблюдателя.

2.1.2.2 Перспективное преобразование

Перспективное преобразование происходит по самой простой матрице преобразования, представленной на формуле (5):

$$perspectiveMatrix = \begin{pmatrix} \frac{width/2}{d} & 0 & 0 & 0 \\ 0 & \frac{height/2}{d} & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{5}$$

где d - расстояние до картинной плоскости, width и height - размеры окна проеции.

После применения данной матрицы происходит перспективная проекция координат на картинную плоскость и масштабирование под окно проекции [16].

2.1.3 Модуль затенения

Как было сказано в аналитической части, реализовывается затенение по Гуро. Формула для данного затенения также представлена в аналитической части под номером (2). Вектор нормали для обрабатываемых точек и отражающие способности материала определяются при генерации ландшафта в модуле генерации ландшафта. Вектор направления на источник света рассчитывается каждый раз.

Также в этом модуле происходит наложение теней. Тени накладываются согласно буферу теней, который перестраивается после каждого изменения положения источника света. Принцип работы алгоритма наложения теней аналогична алгоритму z-буфера. В точке, где находится источник света, помещается камера. После этого от этой камеры производится построения изображения. Точки, которые в этом изображении оказались закрыты другими считаются находящимися в тени и этот источник света, при расчете освещенности, для них игнорируется.

2.1.4 Модуль рендеринга сцены

В данном модуле проводится вызов функционала предыдущих модулей, растеризация и вывод с помощью алгоритма z-буфера, описанного в аналитической части.

2.1.5 Модуль пользовательского интерфейса

Пользовательский интерфейс программы предоставляет возможность задания размера карты, количества узлов сетки для шума Перлина, изменение положения источника света, изменения детализации, изменение максимальной высоты.

2.2 Вывод

В данной части были определены основные модули и алгоритмы программы, которые необходимо реализовать. Вот некоторые из необходимых алгоритмов:

1. Z-буфер;

2. затенение по Гуро;

3. шум Перлина;

4. создание матриц перехода к координатам наблюдателя и перспективноя проекции;

5. наложение теней с помощью модификации Z-буфера;

Также был описан основной функционал модулей и приведено описание реализуемых алгоритмов.

3 Технологическая часть

В данной части будут выбраны средства реализации, описан интерфейс, описаны ограничения и приведены листинги некоторых частей программы.

3.1 Выбор языка программирования

Для реализации ПО был выбран язык программирования C#, так как он предоставляет весь необходимый функционал и код на нем достаточно прост в написании [21]. Для реализации интерфеса была выбрана графическая библиотека WinForms, так как она поддерживается C#, имеет удобный конструктор в среде разработки Visual Studio и применялся в предыдущих учебных курсах [18].

3.2 Интерфейс

Интерфейс приложения представлен на рисунке (4).

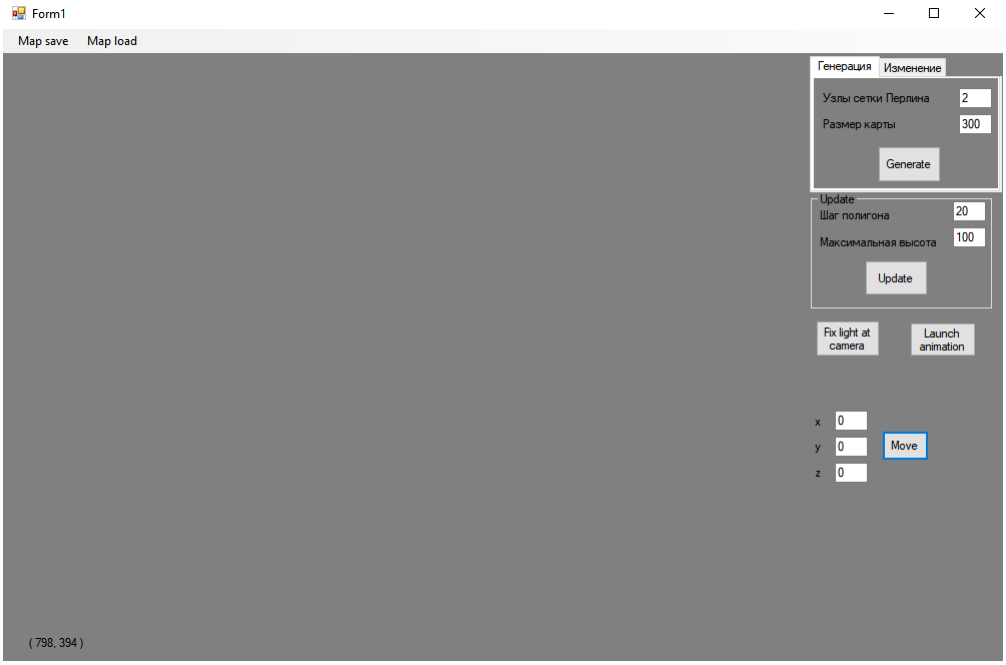

Рисунок 4 – Интерфейс приложения

Интерфейс представляет возможность загрузить и сохранить ландшафт в формате .png кнопками на панели инструментов. Сгенерировать и внести изменения в ландшафт можно на панели справо от рабочей области. Так же есть возможность переместить источник света на место камеры и запустить простую анимацию изменения положения источника света с помощью кнопок ‘fix light at camera‘ и ‘launch animation‘. Поворот камеры осуществляется с помощью зажатия левой кнопки мыши и её перемещения в пределах рабочей области.

3.3 Ограничения

Для работы программы требуется не менее 500мб оперативной памяти.

3.4 Листинги программы

На листинге (1) представлена обработка точки в алгоритме z-буфера.

Листинг 1 – Z-buffer

```
1  private void ProcessPoint(Bitmap image, Dot3d point, Color color)
2        {
3            int x = (int)point.X;
4            int y = (int)point.Y;
5
6            if (point.Z <= Zbuf[y][x])
7            {
8                Zbuf[y][x] = point.Z;
9                image.SetPixel(x, y, color);
10           }
11       }
```

На листинге (2) представлена генерация значения в точке с координатами (x, y) шумом Перлина.

Листинг 2 – Генерация шумом Перлина

```
1        public override double Generate(double x, double y)
2        {
3            x = InsideGridX(x);
4            y = InsideGridY(y);
5            int x0 = (int)x;
6            double dx = x - x0;
7            int y0 = (int)y;
8            double dy = y - y0;
9
10           var vx0y0 = grid[x0, y0];
11           var vx0y1 = grid[x0, y0 + 1];
12           var vx1y0 = grid[x0 + 1, y0];
13           var vx1y1 = grid[x0 + 1, y0 + 1];
14           var s = vx0y0.DotProduct( new Vector3d(dx, dy));
15           var t = vx1y0.DotProduct(new Vector3d(dx - 1, dy));
16           var u = vx0y1.DotProduct(new Vector3d(dx, dy - 1));
17           var v = vx1y1.DotProduct(new Vector3d(dx - 1, dy - 1));
18
19           var a = GetWeightedAverage(s, t, dx);
```

```
20              var b = GetWeightedAverage(u, v, dx);
21              var c = GetWeightedAverage(a, b, dy);
22
23              return Normalize(c);
24          }
```

На листинге (3) представлена предобработка (закраска, перспективное преобразование) перед отрисовкой z-буфером.

Листинг 3 – Верхний уровень обработки Z-буфером

```
1   public override void Process(Bitmap bitmap, Scene scene)
2         {
3             InitBuf(bitmap.Width, bitmap.Height, double.MaxValue);
4             Matrix4x4 mainMatrix = scene.GetMainTransform();
5             ViewFrustum unTransformed = scene.camera.GetFrustum();
6             Shader shader = new Shader(scene.lightSource, scene.camera.place.ToDot());
7
8             foreach (Object m in scene.GetObjects())
9             {
10                m.Colorize(shader);
11                Object transformedModel = m.Transform(mainMatrix);
12                transformedModel.Normilize();
13                ProcessModel(bitmap, transformedModel, mainMatrix, unTransformed,
                      shader);
14            }
15        }
```

3.5 Вывод

В данном разделе была реализовано требуемое программное обеспечение. Для написания программного обеспечения был выбран язык C#, проведена оптимизация в местах, где это возможно.

4 Экспериментальная часть

В данной части будет проведена попытка распараллеливания программы, диагностика работоспособности. Будут построены графики зависимости времени генерации от количества полигонов.

4.1 Использованное оборудование

Технические характеристики ЭВМ, на которой проводились тесты:

- опреационная система Windows 10 x64;
- 8 ГБ оперативной памяти;
- CPU - AMD FX(tm)-6350 Six-Core Processor 3.90GHz.

4.2 Исследование времени обработки кадра

На рисунке (5) приведены графики зависимости времени обработки кадра от количества полигонов на карте размерами 300 на 300 в случае, когда карта находится параллельно картинной плоскости (худший случай). Различные графики соответствуют различному размеру проекции. Размер окна проекции - 800x600.

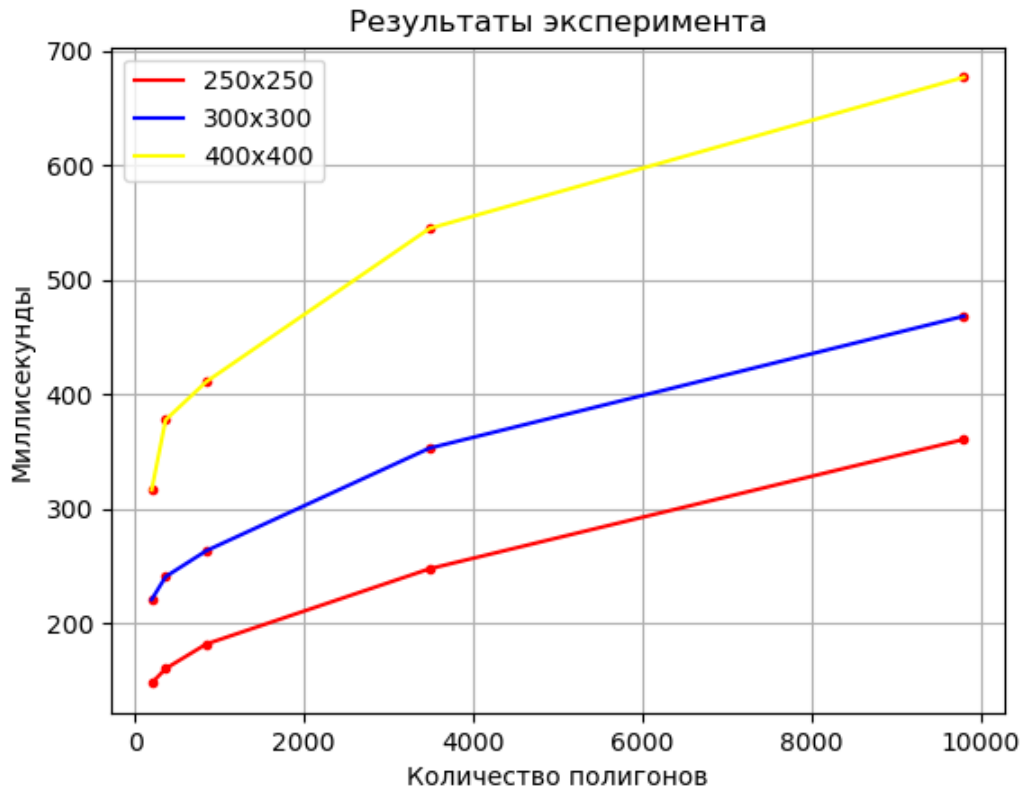

Рисунок 5 – Результаты эксперимента

Как видно из графиков, наибольшее влияние на время отрисовки кадра влияет размер отрисовываемой области после проекции. Это объясняется тем, что операция вывода точки без использования

библиотека типа OpenGL и DirectX черезвычайно долгая [19][20]. Это видно из графика на рисунке (6), показывающего отношение времени на вывод пикселей к общему времени обработки кадра.

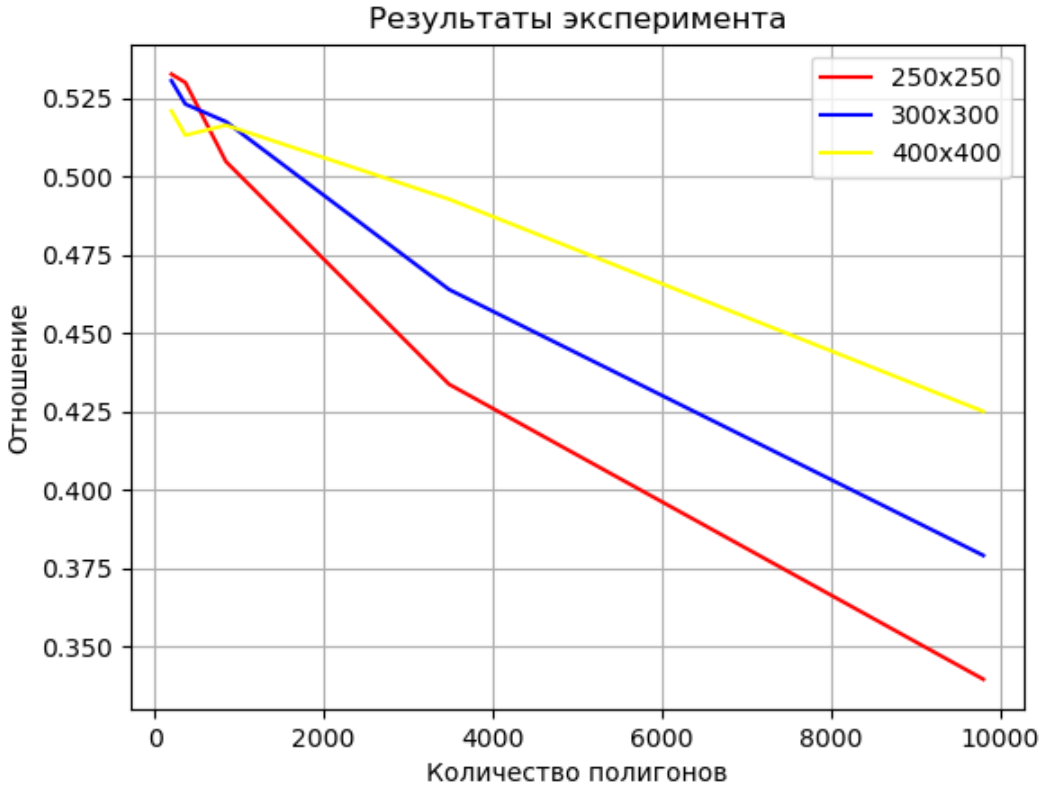

Рисунок 6 – Результаты эксперимента

На предыдущем графике видно, что при увеличении размера изображения увеличивается и степень влияния времени вывода пикселя на общее время. Но, в рамках одного размера изображения, при увеличении количества полигонов степень влияния наоборот, уменьшается. Это обусловлено тем, что количество пикселей остается прежним, но увеличиваются вычислительные затраты на детализацию изображения.

Вторым по размеру вклада в общее время является время закраски по Гуро. В среднем оно вносит 25% ко времени обработки кадра. Третьим - время растеризации, вносящее 20%.

4.3 Пример работы программы

На рисунке (7) представлен пример работы программы.

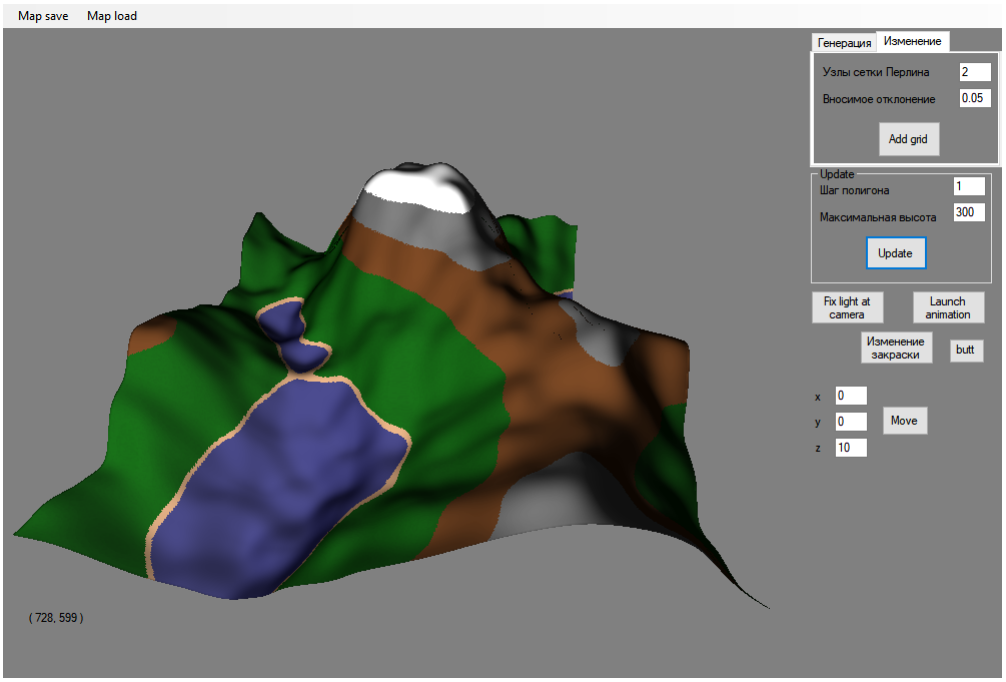

Рисунок 7 – Пример работы программы

4.4 Вывод

Программа работоспособна, функциональность, заложенная на этапе постановки ТЗ реализована. Анализ показал, что существует большой потенциал для ускорения вычислений, используя вычисления на GPU с использованием библиотек DirectX и OpenGL.

Заключение

В данной курсовой работе было разработано программное обеспечение, реализующее генерацию и отрисовку ландшафта. Пользователю был предоставлен функционал влияния на генерацию, путем задания различных параметров генерации. Есть возможность сохранять сгенерированные карты высот и загружать ранее сохраненные или полученные из других источников.

Программное обеспечение работоспособно соответствует поставленным требованиям. Цель работы достигнута, все задачи выполнены.

Список литературы

[1] Карта высот. Wikipedia [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: https://en.wikipedia.org/wiki/Heightmap

[2] Генерация трехмерных ландшафтов [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: https://www.ixbt.com/video/3dterrains-generation.shtml

[3] Генерация трехмерных ландшафтов [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: https://www.ixbt.com/video/3dterrains-generation.shtml

[4] Шум перлина [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: https://medium.com/@yvanscher/playing-with-perlin-noise-generating-realistic-archipelagos-b59f004d8401

[5] Холмовой алгоритм [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: https://www.ixbt.com/video/3dterrains-generation.shtml

[6] Diamond-square algorithm [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: http://jmecom.github.io/blog/2015/diamond-square/

[7] Polygon Mesh [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: https://en.wikipedia.org/wiki/Polygon_mesh

[8] Томский политехнический университет [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: http://compgraph.tpu.ru/zbuffer.htm

[9] kursgm.ru [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: http://kursgm.ru/grpraktik/grafikomp21.htm

[10] Томский политехнический университе [Электронный ресурс]. Режим доступа: (дата обращения - 08.12.2020) Свободный. URL: http://compgraph.tpu.ru/ray.htm

[11] Bui Tuong Phong, Illumination for computer generated pictures, Communications of ACM 18 (1975), no. 6, 311–317

[12] Ikeuchi, Katsushi (2014). "Lambertian Reflectance". Encyclopedia of Computer Vision. Springer. pp. 441–443.

[13] Gouraud, Henri (1971). "Continuous shading of curved surfaces"(PDF). IEEE Transactions on Computers. C-20 (6): 623–629. doi:10.1109/T-C.1971.223313.

[14] Perlin, Ken (July 1985). "An Image Synthesizer". SIGGRAPH Comput. Graph. 19 (97–8930): 287–296

[15] DirectX. MatrixLookAt [Электронный ресурс]. Режим доступа: (дата обращения - 14.12.2020) Свободный. URL: https://docs.microsoft.com/en-us/windows/win32/direct3d9/d3dxmatrixlookatlh

[16] Perspective matrix [Электронный ресурс]. Режим доступа: (дата обращения - 14.12.2020) Свободный. URL:https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/building-basic-perspective-projection-matrix

[17] C# [Электронный ресурс]. Режим доступа: (дата обращения - 14.12.2020) Свободный. https://docs.microsoft.com/en-us/dotnet/csharp/

[18] Официальный сайт WinForm [Электронный ресурс]. Режим доступа: (дата обращения - 14.12.2020) Свободный. URL: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-5.0

[19] Официальный сайт OpenGL [Электронный ресурс]. Режим доступа: (дата обращения - 14.12.2020) Свободный. URL: https://www.opengl.org/

[20] DirectX [Электронный ресурс]. Режим доступа: (дата обращения - 14.12.2020) Свободный. URL: https://www.ixbt.com/video/dirxfaq.html

[21] Golang [Электронный ресурс]. Режим доступа: (дата обращения - 20.11.2020) Свободный. URL: https://ru.wikipedia.org/wiki/C_Sharp

[22] Visual Studio [Электронный ресурс]. Режим доступа: (дата обращения - 20.11.2020) Свободный. URL: https://visualstudio.microsoft.com/ru/