



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

*K KУРСОВОЙ РАБОТЕ*

*НА ТЕМУ:*

*«Генератор трехмерного ландшафта»*

Студент ИУ7-53Б  
(Группа)

(Подпись, дата)

Лысцев Н. Д.  
(И. О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

Филиппов М. В.  
(И. О. Фамилия)

2023 г.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>4</b>
<b>1 Аналитический раздел</b>	<b>5</b>
1.1 Формализация объектов синтезируемой сцены . . . . .	5
1.2 Анализ способов представления данных о ландшафте . . . . .	5
1.2.1 Регулярная сетка . . . . .	5
1.2.2 Иррегулярная сетка . . . . .	6
1.2.3 Посегментная карта высот . . . . .	7
1.3 Анализ алгоритмов процедурной генерации ландшафта . . . . .	7
1.3.1 Алгоритм Diamond-Square . . . . .	7
1.3.2 Холмовой алгоритм . . . . .	8
1.3.3 Алгоритм шума Перлина . . . . .	9
1.4 Анализ алгоритмов удаления невидимых линий и поверхностей .	11
1.4.1 Алгоритм Робертса . . . . .	11
1.4.2 Алгоритм, использующий Z-буфер . . . . .	11
1.4.3 Алгоритм обратной трассировки лучей . . . . .	12
1.5 Анализ моделей освещения . . . . .	13
1.5.1 Модель освещения Ламберта . . . . .	13
1.5.2 Модель освещения Фонга . . . . .	13
1.6 Анализ алгоритмов закраски . . . . .	13
1.6.1 Алгоритм простой закраски . . . . .	13
1.6.2 Алгоритм закраски по Гуро . . . . .	14
1.6.3 Алгоритм закраски по Фонгу . . . . .	14
<b>2 Конструкторский раздел</b>	<b>17</b>
2.1 Требования к программному обеспечению . . . . .	17
2.2 Разработка алгоритмов . . . . .	17
2.2.1 Общий алгоритм построения изображения . . . . .	17
2.2.2 Алгоритм процедурной генерации ландшафта на основе шума Перлина . . . . .	19

2.2.3	Афинные преобразования . . . . .	21
2.2.4	Модель освещения Ламберта . . . . .	23
2.2.5	Алгоритм, использующий Z-буфер . . . . .	24
2.2.6	Алгоритм закраски по Гуро . . . . .	26
2.2.7	Алгоритм, использующий Z-буфер, объединенный с закраской по Гуро . . . . .	27
2.2.8	Выбор типов и структур данных . . . . .	28
2.2.9	Описание структуры программного обеспечения . . . . .	28
<b>3</b>	<b>Технологический раздел</b>	<b>31</b>
3.1	Средства реализации . . . . .	31
3.2	Реализация алгоритмов . . . . .	32
3.3	Интерфейс программы . . . . .	37
3.4	Демонстрация работы программы . . . . .	37
<b>4</b>	<b>Исследовательский раздел</b>	<b>41</b>
4.1	Технические характеристики . . . . .	41
4.2	Проведение первого исследования . . . . .	41
4.2.1	Цель первого исследования . . . . .	41
4.2.2	Наборы варьируемых и фиксированных параметров . . . . .	41
4.2.3	Результаты первого исследования . . . . .	42
4.3	Проведение второго исследования . . . . .	45
4.3.1	Цель второго исследования . . . . .	45
4.3.2	Наборы варьируемых и фиксированных параметров . . . . .	45
4.3.3	Результаты второго исследования . . . . .	45
<b>ЗАКЛЮЧЕНИЕ</b>		<b>49</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>		<b>52</b>

# ВВЕДЕНИЕ

В настоящее время технологии трехмерной графики стремительно развиваются. Одним из направлений применения этих технологий является создание видеоигр. Наибольшую популярность набирают игры с так называемым открытым миром.

Основой для открытого мира служит трехмерный ландшафт значительных размеров, который для реалистичности должен быть разнообразным и детализированным. Традиционные методы ручного моделирования ландшафта являются крайне громоздкими, трудоемкими и ограниченными в своей вариативности.

Возникает потребность в создании программного обеспечения, которое бы позволяло автоматизировать процессы создания реалистичного ландшафта, чтобы ускорить разработку игр и обеспечить большую степень креативной свободы для разработчиков.

Целью работы является разработка программного обеспечения для генерации и визуализации трехмерного ландшафта.

Для достижения желаемых результатов необходимо решить следующие задачи:

- 1) выполнить формализацию объектов синтезируемой сцены;
- 2) провести анализ существующих алгоритмов создания ландшафта и визуализации сцены;
- 3) выбрать подходящие алгоритмы для решения поставленной задачи.
- 4) реализовать выбранные алгоритмы;
- 5) провести исследование временных характеристик выбранных алгоритмов.

# 1 Аналитический раздел

В данном разделе дано формальное описание объектов синтезируемой сцены, проанализированы разные способы представления данных о ландшафте, рассмотрены алгоритмы процедурной генерации ландшафта, алгоритмы удаления невидимых линий и поверхностей, алгоритмы закраски и модели освещения.

## 1.1 Формализация объектов синтезируемой сцены

Сцена состоит из следующих объектов:

- ландшафт – трехмерная модель, описываемая полигональной сеткой **[lands]**;
- источник света – материальная точка, испускающая лучи света.

## 1.2 Анализ способов представления данных о ландшафте

Существует несколько основных принципов представления данных для хранения информации о ландшафте [1]:

- регулярная сетка высот (карта высот);
- иррегулярная сетка вершин и связей, их соединяющих;
- посегментная карта высот.

### 1.2.1 Регулярная сетка

Данные представлены в виде двумерного массива [2]. Каждый элемент массива имеет свои индексы  $[i, j]$ , являющиеся координатами расположения точки на плоскости. Для каждой вершины с индексами  $[i, j]$  в двумерном массиве определяется соответствующее значение высоты  $h_{ij}$ . На рисунке 1.1 показан пример представления ландшафта с помощью карты высот.

0	1	2	3	4
1	1	1	2	2
2	0	1	2	2
3	0	1	1	1
4	0	0	0	0

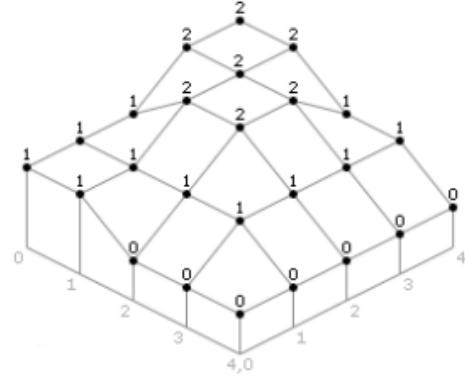


Рисунок 1.1 – Пример представления ландшафта с помощью карты высот [2]

К преимуществам данного способа можно отнести наглядность и простоту изменения данных, легкость нахождения координат и высоты на карте, возможность более точно производить динамическое освещение. Однако, у такого метода есть один существенный недостаток – избыточность данных (например, при задании плоскости) [1].

### 1.2.2 Иррегулярная сетка

Иррегулярная сетка позволяет размещать точки или узлы в произвольных местах в зависимости от необходимости и особенностей ландшафта.

У такого метода есть несколько существенных недостатков [1]:

- многие алгоритмы построения ландшафтом предназначены для регулярных сеток высот, поэтому оптимизация этих алгоритмов под иррегулярную сетку потребует значительных усилий и времени;
- из-за неравномерного расположения вершинных точек друг к другу возникает сложность при создании динамического освещения;
- сложности при хранении, модификации и просмотре такого ландшафта.

К плюсам данного способа можно отнести использование меньшей информации для построения ландшафта [1].

### **1.2.3 Посегментная карта высот**

Суть этого метода заключается в разделении изначальной области на небольшие участки (например, квадратной формы), и генерации высот на каждом участке отдельно [3].

К преимуществам данного способа можно отнести возможность представления больших открытых пространств, возможность хранения информации о других объектах (строения, пещеры, скалы), возможность создания нескольких вариантов одного и того же сегмента, но с разным уровнем детализации. К недостаткам можно отнести проблемустыковки разных сегментов, неочевидность представления данных, проблему модификации этих данных [1].

## **1.3 Анализ алгоритмов процедурной генерации ландшафта**

Основным критерием выбора алгоритма будет качество получаемого ландшафта, поскольку для решения задачи необходимо создавать правдоподобный рельеф.

### **1.3.1 Алгоритм Diamond-Square**

Данный алгоритм является расширением одномерного алгоритма *midpoint displacement* [4] на двумерную плоскость. Алгоритм *Diamond – Square* [5] выполняется рекурсивно и начинает работу с двумерного массива размера  $2^n + 1$ . В четырёх угловых точках массива устанавливаются начальные значения высот. Шаги *diamond* и *square* выполняются поочередно до тех пор, пока все значения массива не будут установлены.

- 1) Шаг *diamond* – для каждого квадрата в массиве, устанавливается срединная точка, которой присваивается среднее арифметическое из четырёх угловых точек плюс случайное значение.
- 2) Шаг *square* – берутся средние точки граней тех же квадратов, в которые устанавливается среднее значение от четырёх соседних с ними по осям точек плюс случайное значение.

На рисунке 1.2 показаны последовательность действий алгоритма *Diamond – Square* на примере массива 5x5.

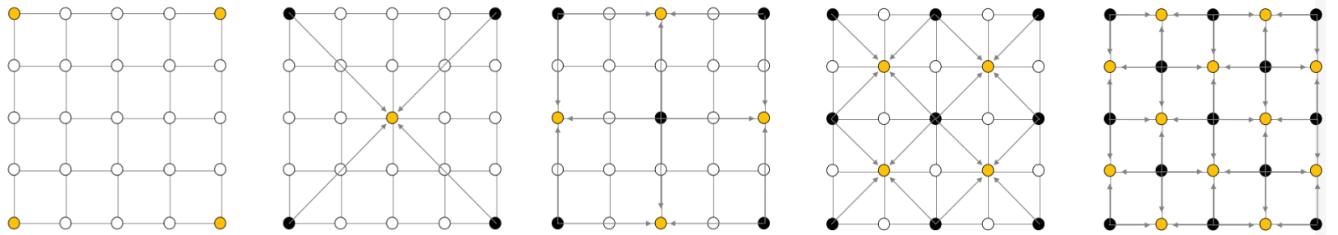


Рисунок 1.2 – Шаги, проходимые алгоритмом Diamond-Square на примере массива 5x5 [6]

К преимуществам данного алгоритма можно отнести простоту его реализации и возможность генерации различных ландшафтов с разнообразной детализацией.

К недостаткам можно отнести создание вертикальных и горизонтальных «складок» на краях карты из-за наибольшего значительного возмущения, происходящего в прямоугольной сетке [6], а также сложности с контролем получаемого ландшафта.

### 1.3.2 Холмовой алгоритм

Это простой итерационный алгоритм, основанный на нескольких входных параметрах. Алгоритм изложен в следующих шагах [7]:

- 1) Создается и инициализируется нулевым уровнем двухмерный массив;
- 2) Берется случайная точка на ландшафте или около его границ (за границами) 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) Выполнение пунктов 1) - 3)  $n$  раз, где  $n$  – выбранное количество шагов.
- 5) Проведение нормализации ландшафта;
- 6) Проведение «долинизации» ландшафта.

К преимуществам данного алгоритма можно отнести простоту его реализации, а также возможность контроля «гористости» ландшафта за счет этапов нормализации и долинизации.

Данный алгоритм имеет несколько недостатков:

- с помощью этого алгоритма тяжело смоделировать склон холма, или горы, так как в процессе генерации ландшафта используются только гладкие полусфера [8];
- этот алгоритм неприменим, когда требуется детализировано отобразить лишь часть всего ландшафта [8].

### 1.3.3 Алгоритм шума Перлина

Это математический алгоритм по генерированию процедурной текстуры псевдо-случайным методом [9]. Этот алгоритм может быть реализован для n-мерного пространства, но чаще его используют для одно-, двух-, трехмерного случая.

Рассмотрим версию этого алгоритма для двумерного случая:

Для карты высот создается сетка точек и в каждой точке сетки генерируется псевдослучайный единичный вектор-направления градиента. Для каждой точки с координатами  $(x, y)$  из карты высот определяется, в какой ячейке сетки находится точка, генерируются вектора, идущие от точек ячейки сетки до этой точки. На рисунке 1.3 показан пример векторов для точки  $(x, y)$ .

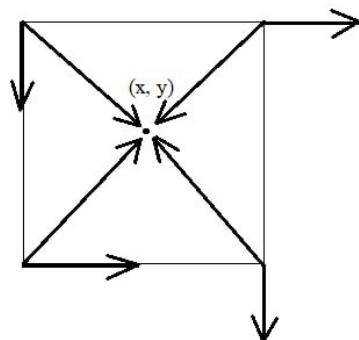


Рисунок 1.3 – Пример сгенерированных векторов для точки  $(x, y)$

Вычисляются четыре скалярных произведения векторов-направлений градиента и вектора, идущего от связанной с ним точки сетки к точке с координатами  $(x, y)$ .

Далее, чтобы получить значение высоты  $z$  в точке  $(x, y)$ , необходимо провести двумерную интерполяцию на основе полученных скалярных произведений. Для создания плавного перехода между значениями предварительно производятся вычисления весов измерений по  $x$  и по  $y$ . Веса определяются функцией

$$\text{smootherstep}(t) = 6t^5 - 15t^4 + 10t^3 \quad (1.1)$$

где вместо  $t$  подставляются значения  $x$  и  $y$ . Далее, используя метод последовательной интерполяции по каждому измерению, получаем значения двух одномерных линейных интерполяций с использованием веса по  $x$ , и в конце проводим одну одномерную линейную интерполяцию с этими вычисленными значениями, но уже по весу измерения  $y$ . Полученное значение и будет высотой в данной точке  $(x, y)$  карты высот [10].

Чтобы контролировать генерацию шума, существует набор параметров:

- октавы – количество уровней детализации шума;
- лакунарность – множитель, который определяет изменение частоты с ростом октавы;
- стойкость – множитель частотной амплитуды, который определяет изменение амплитуды с ростом октавы.

Для достижения качественного детализированного ландшафта можно смешивать шумы разных частот и амплитуд.

Преимущества:

- комбинация различных октав в алгоритме шума Перлина позволяет добавлять дополнительные уровни детализации к сгенерированному шуму, тем самым повышая реалистичность и детализацию ландшафта;
- зная лишь параметры генерации, можно получить высоту в любой точке карты без необходимости знания высот в соседних точках карты.

Недостатком является низкая детализация и реалистичность сгенерированного ландшафта без использования механизма комбинации октав.

## **1.4 Анализ алгоритмов удаления невидимых линий и поверхностей**

Главным требованием при выборе алгоритма будут высокая скорость работы, чтобы пользователю не приходилось ждать долгой загрузки изображения.

### **1.4.1 Алгоритм Робертса**

Алгоритм Робертса работает в объектном пространстве только с выпуклыми телами. Если тело не является выпуклым, то его предварительно нужно разбить на выпуклые составляющие [11].

Этот алгоритм выполняется в 4 этапа:

- 1) Подготовка исходных данных – составление матрицы тела для каждого тела сцены;
- 2) Удаление ребер, экранируемых самим телом;
- 3) Удаление ребер, экранируемых другими телами;
- 4) Удаление линий пересечения тел, экранируемых самими телами и другими телами, связанными отношением протыкания.

Алгоритм работает только с выпуклыми телами, что является недостатком данного алгоритма. Также к недостатку можно отнести то, что вычислительная сложность теоретически растет как квадрат числа объектов. К преимуществам можно отнести высокую точность вычислений.

### **1.4.2 Алгоритм, использующий Z-буфер**

Это один из простейших алгоритмов удаления невидимых поверхностей. Работает этот алгоритм в пространстве изображения[11].

В данном алгоритме используется два буфера: буфер кадра и z-буфер. Буфер кадра используется для заполнения атрибутов (интенсивности) каждого

пикселя в пространстве изображения. z-буфер – отдельный буфер глубины, используемый для запоминания координаты  $z$  или глубины каждого видимого пикселя в пространстве изображения [11].

Вначале в z-буфер заносятся минимально возможные значения  $z$ , а буфер кадра заполняется фоновым значением интенсивности или цвета. Затем каждый многоугольник преобразовывается в растровую форму. Суть работы алгоритма заключается в следующем: в процессе работы глубина (значение координаты  $z$ ) каждого нового пикселя, который надо занести в буфер кадра сравнивается с глубиной того пикселя, который уже есть в z-буфере. Если новый пиксель оказывается расположен ближе к наблюдателю, то новый пиксель заносится в буфер кадра. При этом в z-буфер заносится глубина нового пикселя. Если сравнение дало противоположный результат, то никаких действий не производится.

К преимуществам данного алгоритма относится простота его реализации, возможность работы со сценами любой сложности, отсутствие необходимости в сортировке по приоритету глубины.

К недостаткам можно отнести большой объем требуемой памяти, трудоемкость и высокая стоимость устранения лестничного эффекта, трудоемкость реализации эффектов прозрачность и просвечивания.

#### **1.4.3 Алгоритм обратной трассировки лучей**

Наблюдатель видит объект посредством испускаемого источником света, который падает на этот объект и согласно законам оптики некоторым путем доходит до глаза наблюдателя. Алгоритм имеет такое название, потому что эффективнее с точки зрения вычислений отслеживать пути лучей в обратном направлении, то есть от наблюдателя к объекту.

Преимуществами данного алгоритма являются возможность его использования в параллельных вычислительных системах и высокая реалистичность получаемого изображения, а недостатком – большое количество вычислений и медленная работа алгоритма.

## **1.5 Анализ моделей освещения**

В данном разделе будут рассмотрены две модели освещения: модель Ламберта и модель Фонга.

### **1.5.1 Модель освещения Ламберта**

Модель Ламберта моделирует идеальное диффузное освещение. Считается, что свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль  $N$ ) и направление на источник света (вектор  $L$ ). Рассеянная составляющая рассчитывается по закону косинусов (закон Ламберта) [12].

### **1.5.2 Модель освещения Фонга**

Модель Фонга – классическая модель освещения. Модель представляет собой комбинацию диффузной составляющей (модели Ламберта) и зеркальной составляющей и работает таким образом, что кроме равномерного освещения на материале может еще появляться блик. Местонахождение блика на объекте, освещенном по модели Фонга, определяется из закона равенства углов падения и отражения. Если наблюдатель находится вблизи углов отражения, яркость соответствующей точки повышается [12].

## **1.6 Анализ алгоритмов закраски**

Методы закраски используются для затенения полигонов модели в условиях некоторой сцены, имеющей источники освещения. Существует три основных алгоритма, позволяющих закрасить полигональную модель.

### **1.6.1 Алгоритм простой закраски**

Суть данного алгоритма заключается в том, что для каждой грани объекта находится вектор нормали, и с его помощью в соответствии с выбранной моделью освещения вычисляется значение интенсивности, с которой закрашивается вся грань. При данной закраске все плоскости (в том числе и те, что аппроксимируют фигуры вращения), будут закрашены однотонно, что в случае с фигурами

вращения будет давать ложные ребра [11].

К преимуществам можно отнести простоту реализации алгоритма и его быстродействие. В качестве недостатков можно выделить нереалистичность результата.

### **1.6.2 Алгоритм закраски по Гуро**

Данный алгоритм позволяет получить более сглаженное изображение. Это достигается благодаря тому, что разные точки грани закрашиваются разным значением интенсивности.

Алгоритм состоит из следующих шагов:

- 1) Вычисление векторов нормалей к каждой грани;
- 2) Вычисление векторов нормали к каждой вершине грани путем усреднения нормалей примыкающих граней;
- 3) Вычисление интенсивности в вершинах грани в соответствии с выбранной моделью освещения;
- 4) Выполнение линейной интерполяции интенсивности вдоль ребер;
- 5) Выполнение линейной интерполяции вдоль сканирующей строки.

Преимуществами данного алгоритма являются хорошее сочетание с моделью освещения с диффузным отражением, а также более высокая, по сравнению с алгоритмом простой закраски, реалистичность получаемого изображения.

К недостаткам можно отнести отсутствие учета кривизны поверхности [11].

### **1.6.3 Алгоритм закраски по Фонгу**

В алгоритме закраски по Фонгу используется билинейная интерполяция не интенсивностей в вершинах полигона, а билинейная интерполяция векторов нормалей. Благодаря такому подходу изображение получается более реалистичным. Однако для достижения такого результата требуется больше вычислительных затрат [11].

Данный алгоритм состоит из следующих шагов:

- 1) Вычисление векторов нормалей к каждой грани;
- 2) Вычисление векторов нормали к каждой вершине грани путем усреднения нормалей примыкающих граней;
- 3) Выполнение линейной интерполяции нормалей вдоль ребер;
- 4) Выполнение линейной интерполяции нормалей вдоль сканирующей строки.
- 5) Вычисление интенсивности в каждой вершине

## Вывод

Таблица 1.1 – Сравнение способов представления данных о ландшафте

Способ	Наглядность представления данных	Сложность модификации данных
Регулярная сетка	Высокая	Низкая
Иррегулярная сетка	Высокая	Средняя
Посегментная карта высот	Средняя	Высокая

Таблица 1.2 – Сравнение алгоритмов процедурной генерации ландшафта

Алгоритм	Качество ландшафта	Отсутствие артефактов	Контроль ландшафта
Diamond-Square	Среднее	–	Низкая
Холмовой алгоритм	Среднее	+	Средний
Шум Перлина	Высокое	+	Высокая

Таблица 1.3 – Сравнение алгоритмов удаления невидимых линий и поверхностей

Алгоритм	Сложность алгоритма	Скорость работы	Типы объектов
Алгоритм Робертса	$O(n^2)$	Средняя	Выпуклые многогранники
Алгоритм с $z$ -буфером	$O(np)$	Высокая	Произвольные
Алгоритм с обратной трассировкой лучей	$O(np)$	Низкая	Произвольные

Таблица 1.4 – Сравнение моделей освещения

Модель освещения	Реалистичность изображения	Объем вычислений
Модель Ламберта	Средняя	Средний
Модель Фонга	Высокая	Большой

Таблица 1.5 – Сравнение алгоритмов закраски

Алгоритм закраски	Скорость работы	Реалистичность изображения	Сочетание с диффузным отражением
Простая закраска	Высокая	Низкая	Высокое
Закраска по Гуро	Средняя	Средняя	Высокое
Закраска по Фонгу	Низкая	Высокая	Средняя

В данном разделе были рассмотрены существующие методы для визуализации и построения трехмерного ландшафта. В качестве способа представления данных о ландшафте была выбрана регулярная карта высот ввиду своей простоты представления данных, в качестве алгоритма генерации карты высот был выбран алгоритм шума Перлина. Поскольку основным критерием выбора алгоритмов была скорость их работы, то для удаления невидимых линий и поверхностей был выбран алгоритм, использующий Z-буффер, а в качестве модели освещения и алгоритма закраски были выбраны модель Ламберта и алгоритм закраски по Гуро.

## **2 Конструкторский раздел**

В данном разделе будет приведены требования к разрабатываемому ПО, будут приведены алгоритмы построения и визуализации ландшафта, будет приведена структура разрабатываемого ПО.

### **2.1 Требования к программному обеспечению**

Разрабатываемое программное обеспечение должно предоставлять пользователю следующую функциональность:

- изменение параметров алгоритма шума Перлина для генерации ландшафта;
- изменение положение источника света;
- возможность управления положением модели (перемещение, масштабирование, поворот);
- задание и изменение уровня моря в интерактивном режиме;

При этом разрабатываемая программа должна удовлетворять следующим требованиям:

- время отклика программы должно быть менее 1 секунды для корректной работы в интерактивном режиме;
- программа должна корректно реагировать на любые действия пользователя.

### **2.2 Разработка алгоритмов**

#### **2.2.1 Общий алгоритм построения изображения**

На рисунке 2.1 представлена схема общего алгоритма построения изображения.

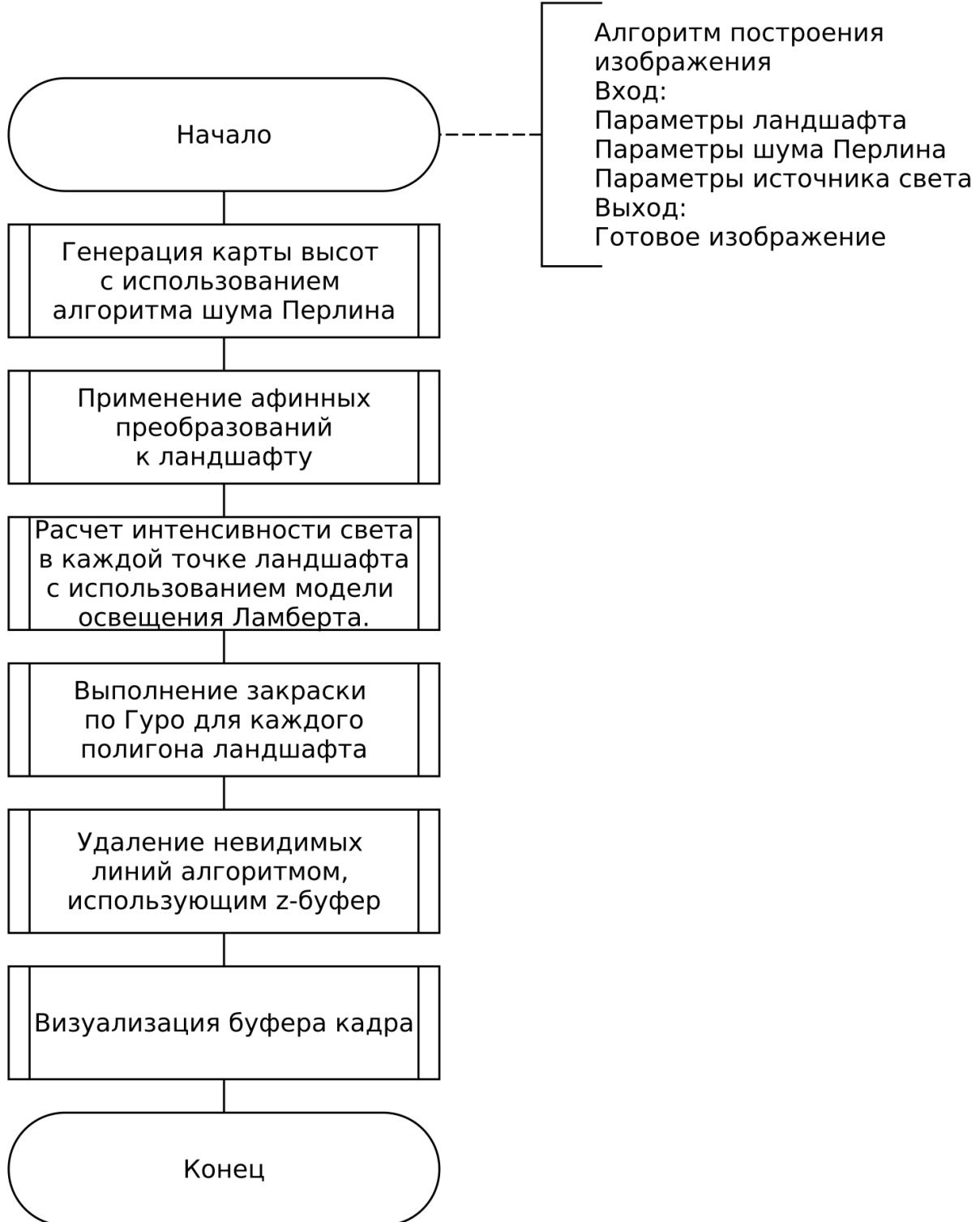


Рисунок 2.1 – Схема алгоритма генерации карты высот на основе шума Перлина

## 2.2.2 Алгоритм процедурной генерации ландшафта на основе шума Перлина

На рисунке 2.2 представлена схема алгоритма генерации карты высот на основе шума Перлина.

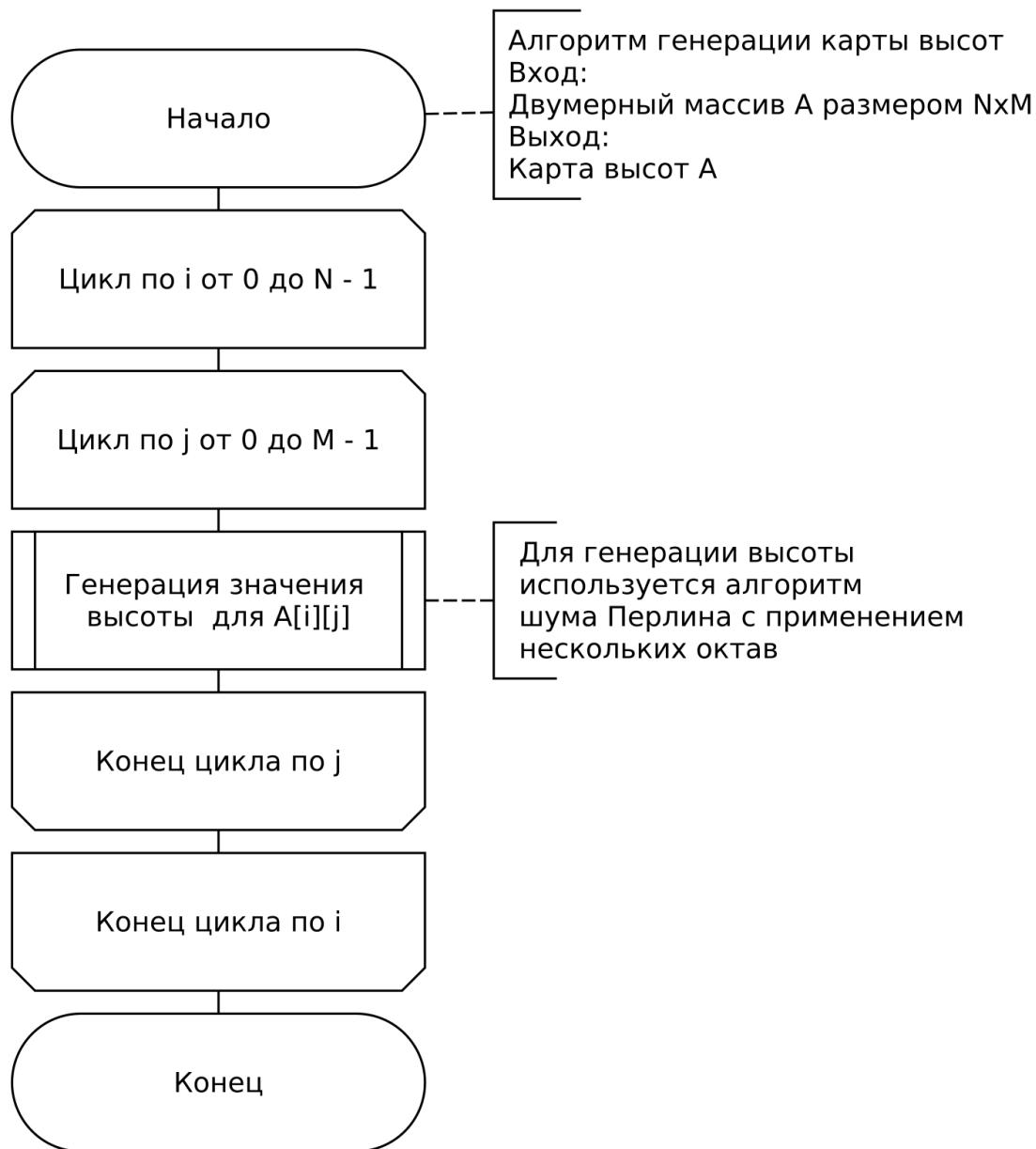


Рисунок 2.2 – Схема алгоритма генерации карты высот на основе шума Перлина

На рисунке 2.3 представлена схема алгоритма шума Перлина с применением нескольких октав.

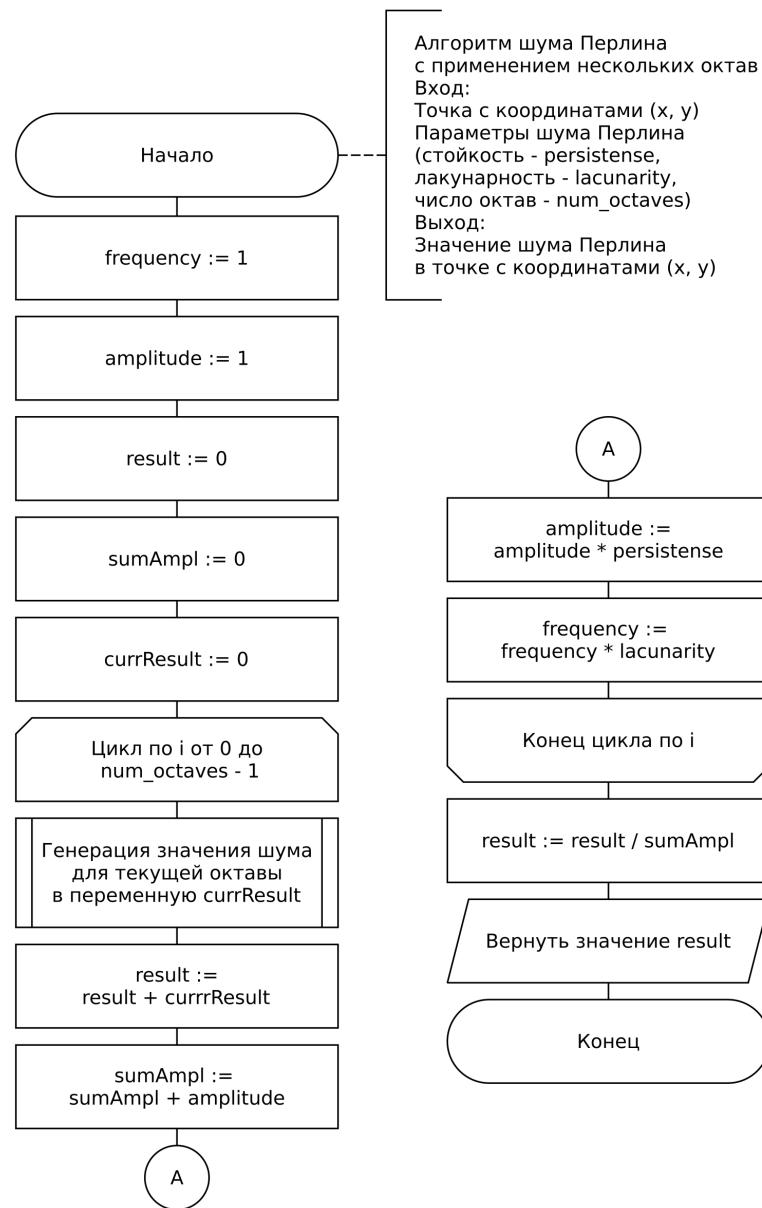


Рисунок 2.3 – Схема алгоритма шума Перлина с применением нескольких октав

На рисунке 2.4 представлена схема алгоритма шума Перлина.

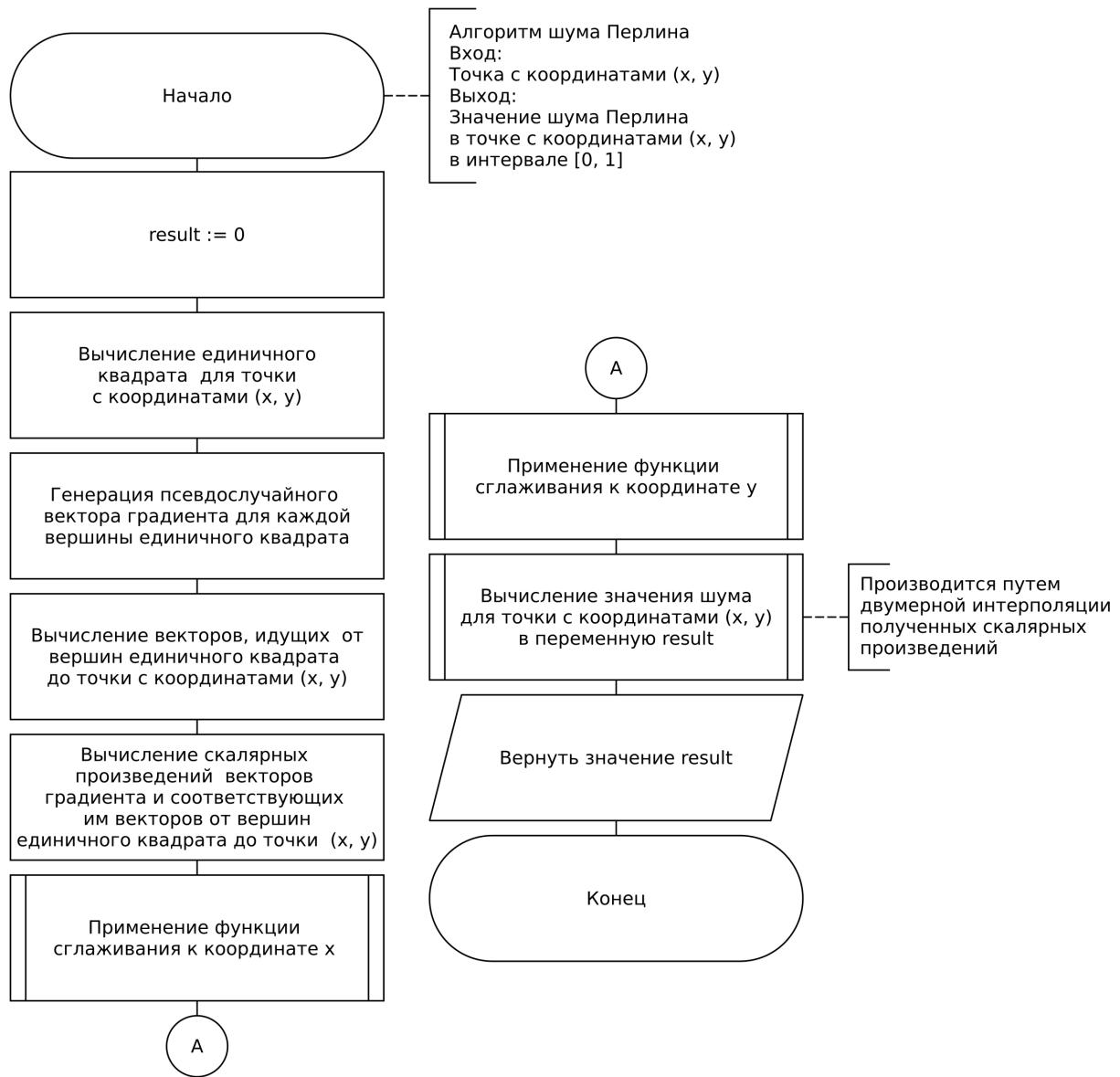


Рисунок 2.4 – Схема алгоритма шума Перлина

### 2.2.3 Афинные преобразования

Для осуществления управлением положения модели используются афинные преобразования [13], задающиеся матрицами. Изменение положения точки трехмерного пространства в результате применение одного из афинных преобразований задается формулой 2.1.

$$(x', y', z', 1) = (x, y, z, 1) \cdot M \quad (2.1)$$

где  $x, y, z$  – старые координаты точки,  $x', y', z'$  – новые координаты точки,  $M$  – матрица афинного преобразования (формулы 2.2 - 2.6).

Поворот вокруг одной из осей координат задается углом вращения  $\alpha$  и осуществляется с помощью соответствующей матрицы поворота (формулы 2.2 - 2.4).

Матрица поворота вокруг оси ОХ:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.2)$$

Матрица поворота вокруг оси ОY:

$$\begin{pmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.3)$$

Матрица поворота вокруг оси ОZ:

$$\begin{pmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.4)$$

Перенос в трехмерном пространстве задается значениями смещения положения точки вдоль координатных осей  $OX, OY, OZ - dx, dy, dz$  соответственно.

Матрица переноса имеет вид:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix} \quad (2.5)$$

Масштабирование задается значениями коэффициентов масштабирования по каждому из направлений  $OX, OY, OZ - kx, ky, kz$ .

Матрица масштабирования имеет вид:

$$\begin{pmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.6)$$

## 2.2.4 Модель освещения Ламберта

Согласно общему алгоритму построения изображения (рисунок 2.1) третьим шагом является вычисление интенсивности в каждой точке ландшафта с использованием модели освещения Ламберта.

В основе модели Ламберта лежит закон Ламберта [14], который утверждает, что интенсивность света, отраженного от материала, пропорциональна косинусу угла между нормалью к поверхности и направлением света. Это означает, что поверхности, ориентированные более к световому источнику, будут ярче, а те, которые ориентированы более в сторону от источника света, будут менее освещены. На рисунке 2.5 показан пример взаимного расположения вектора нормали (вектор  $N$ ) и вектора направления от точки к источнику света (вектор  $L$ ).

Пусть:

- $\vec{L}$  — единичный вектор направления от точки до источника;
- $\vec{N}$  — единичный вектор нормали;
- $I$  — результирующая интенсивность света в точке;
- $I_0$  — интенсивность источника света;

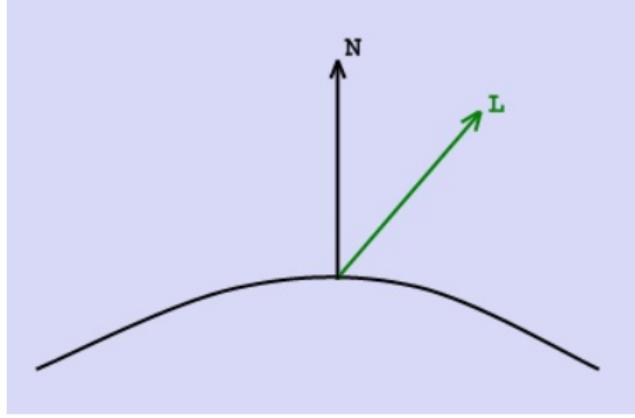


Рисунок 2.5 – Пример расположения векторов  $N$  и  $L$  в модели освещения Ламберта [12]

- $K_d$  — коэффициент диффузного освещения.

Формула расчета интенсивности имеет следующий вид:

$$I = I_0 \cdot K_d \cdot \cos(\vec{L}, \vec{N}) \cdot I_d = I_0 \cdot K_d \cdot (\vec{L}, \vec{N}) \quad (2.7)$$

### 2.2.5 Алгоритм, использующий Z-буфер

На рисунке 2.6 представлена схема алгоритма z-буфера.

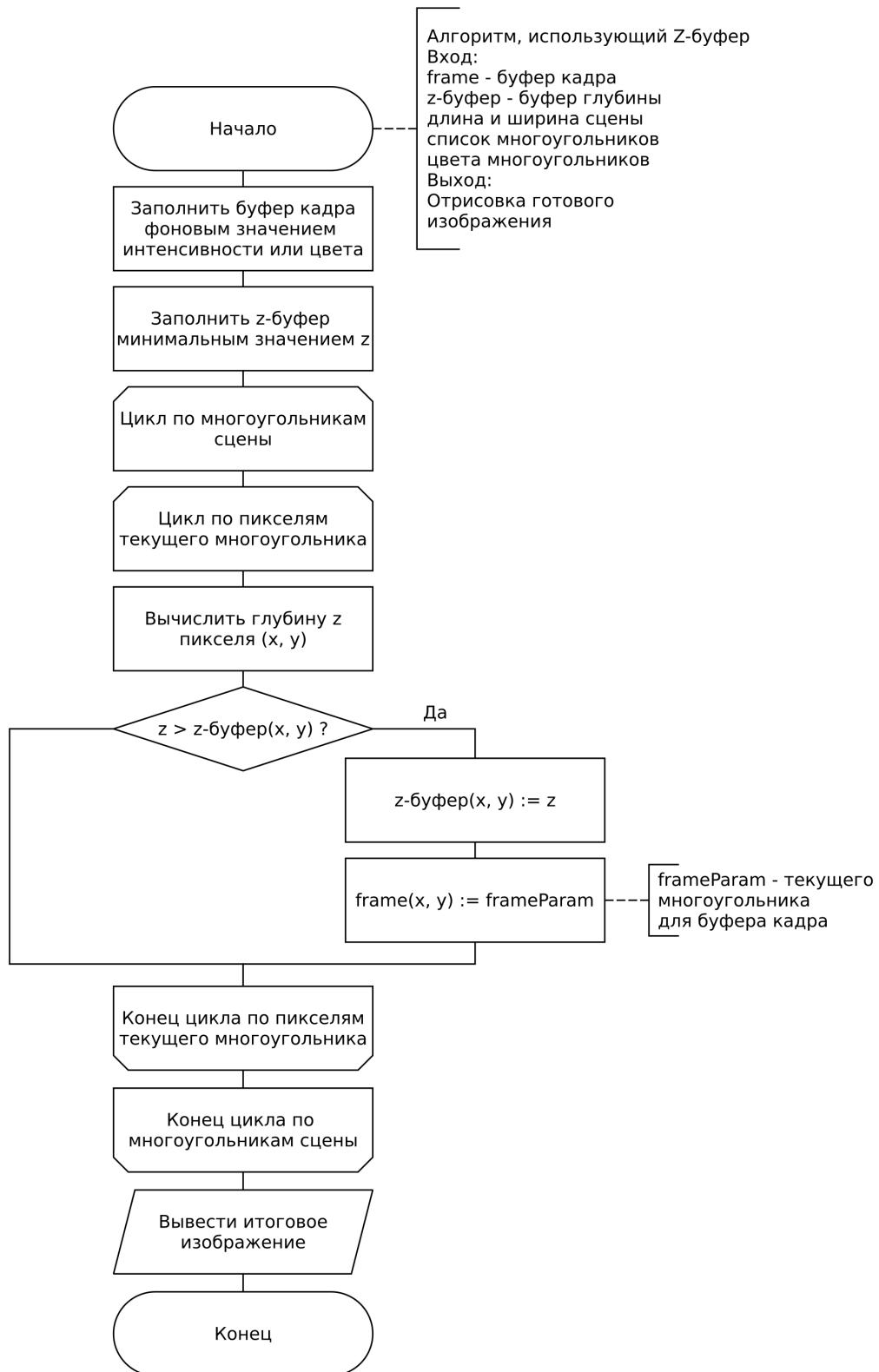


Рисунок 2.6 – Схема алгоритма z-буфера

## 2.2.6 Алгоритм закраски по Гуро

На рисунке 2.7 представлена схема алгоритма закраски по Гуро.

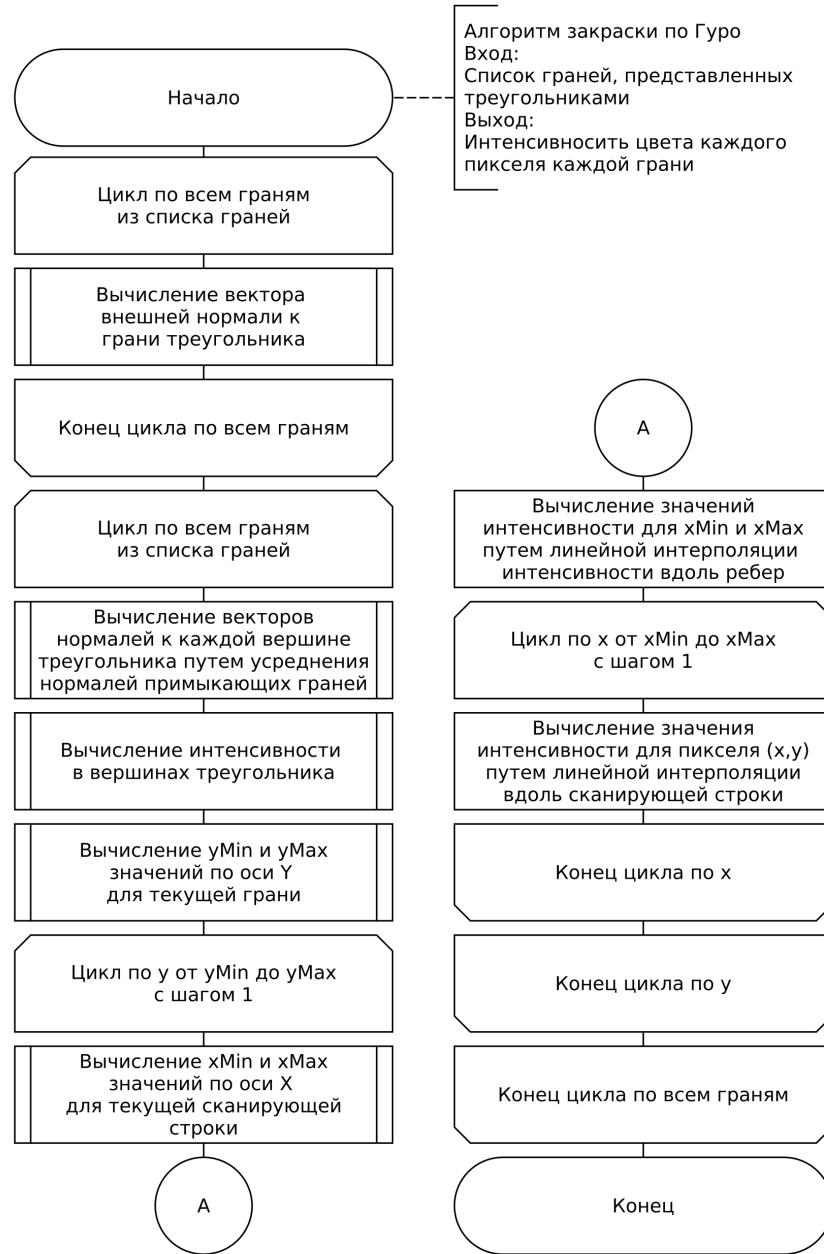


Рисунок 2.7 – Схема алгоритма закраски по Гуро

## 2.2.7 Алгоритм, использующий Z-буфер, объединенный с закраской по Гуро

На рисунке 2.8 представлена схема алгоритма, использующего Z-буфер, объединенного с закраской по Гуро.

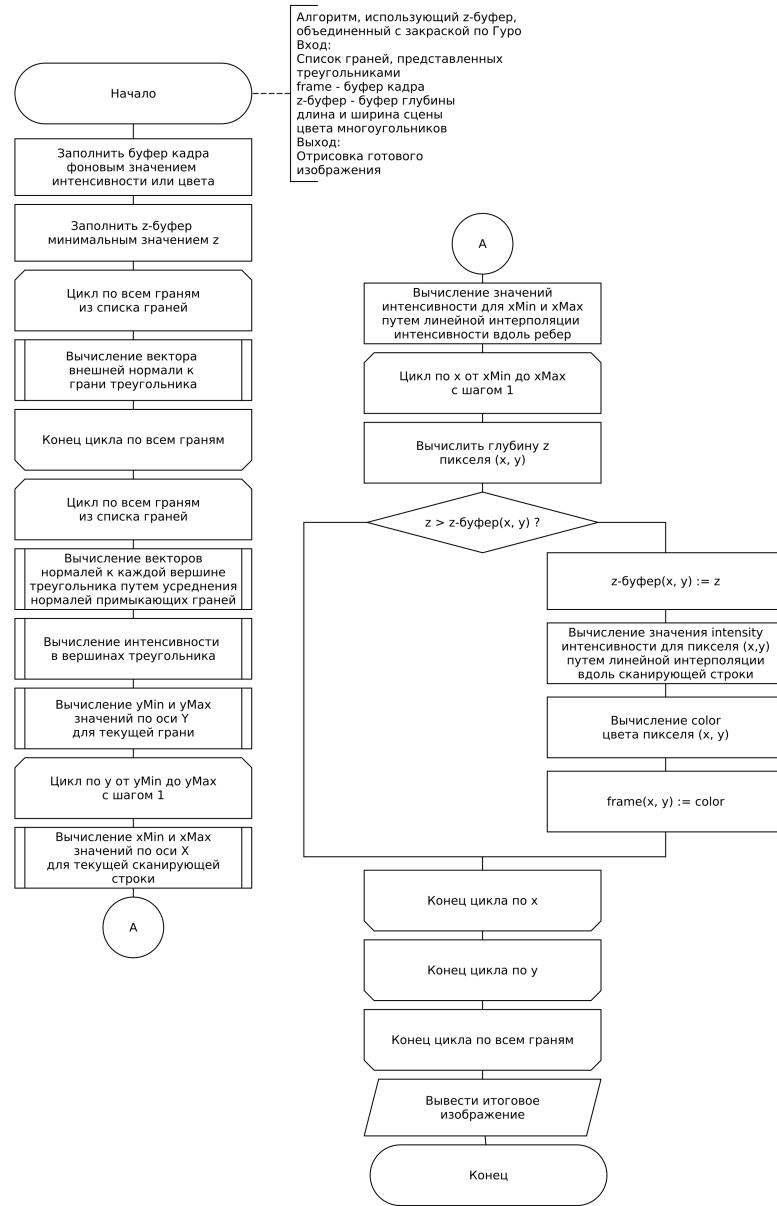


Рисунок 2.8 – Схема алгоритма, использующего Z-буфер, объединенного с закраской по Гуро

## 2.2.8 Выбор типов и структур данных

Для реализации разрабатываемого ПО необходимо использование структур данных, представленных в таблице 2.1.

Таблица 2.1 – Представление данных в ПО

Данные	Представление
Точка трехмерного пространства	Координаты по осям $x$ , $y$ и $z$
Вектор	Точка трехмерного пространства
Карта высот	Двумерный массив точек трехмерного пространства
Плоскость	Три точки трехмерного пространства
Источник света	Точка трехмерного пространства со значениями интенсивности и коэффициента

## 2.2.9 Описание структуры программного обеспечения

На рисунке 2.9 представлена диаграмма классов разрабатываемого программного обеспечения.

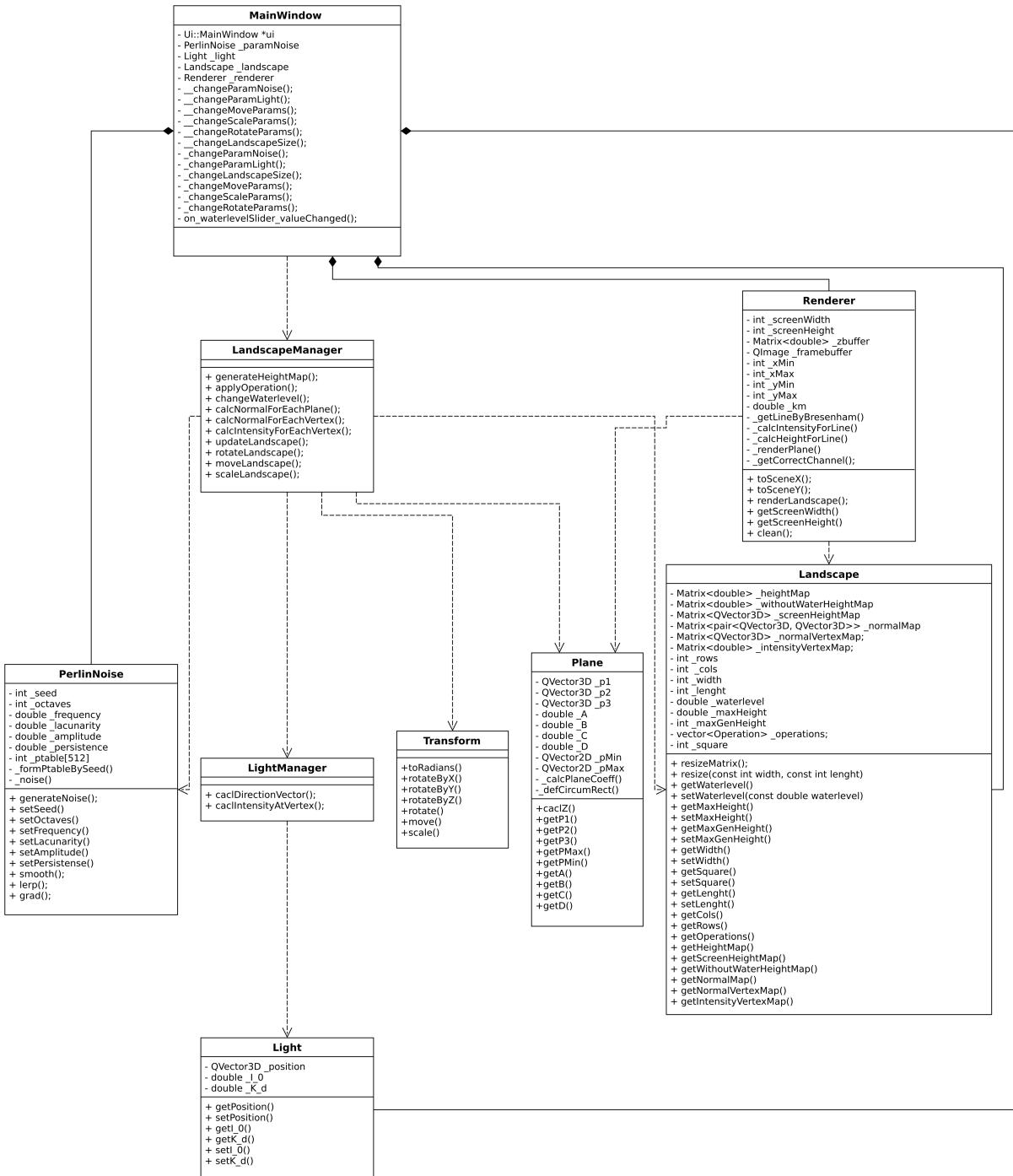


Рисунок 2.9 – Диаграмма классов

В разрабатываемом программном обеспечении реализуются следующие классы:

- **PerlinNoise** — класс, хранящий параметры алгоритма шума Перлина и реализующий возможность генерации высоты для переданной точки;
- **Plane** — класс для представления плоскости, являющейся треугольным полигоном;
- **Transform** — класс для осуществления афинных преобразований;
- **Light** — класс для представления точечного источника света;
- **LightManager** — класс для вычисления интенсивностей света в точке;
- **Landscape** — класс для представления трехмерного ландшафта;
- **LandscapeManager** — класс для осуществления всех операций по изменению ландшафта;
- **Renderer** — класс для растеризации ландшафта и вывода его на экран.

## Вывод

На основе данных, полученных из аналитического раздела, были описаны общий алгоритм построения изображения, приведены схемы алгоритма генерации карты высот, алгоритма закраски по Гуро и алгоритма удаления невидимых линий, использующего Z-буфер. Было дано математическое описание модели освещения Ламберта и приведены матрицы афинных преобразований. Также была приведена структура разрабатываемого ПО в виде диаграммы классов и были выдвинуты требования к ПО.

### 3 Технологический раздел

В данном разделе приведены средства реализации ПО, листинги реализованных алгоритмов, описание интерфейса и примеры работы программы.

#### 3.1 Средства реализации

Для реализации программного продукта был выбран язык программирования *C++* [15] по следующим причинам:

- данный язык преподавался в рамках курса по ООП;
- *C++* обладает высокой вычислительной производительностью; [16; 17], что очень важно для решения поставленной задачи;
- В стандартной библиотеке языка присутствует поддержка всех структур данных, выбранных по результатам проектирования;
- средствами *C++* можно реализовать все алгоритмы, выбранные в результате проектирования.

В качестве среды разработки предпочтение было отдано среде *QT Creator* [18] по следующим причинам:

- он обладает всем необходимым функционалом для написания, проектирования и отладки программ, а также создания графического пользовательского интерфейса;
- данная среда поставляется с фреймворком Qt [19], который содержит в себе все необходимые средства, позволяющие работать непосредственно с пикселями изображения;
- *QT Creator* позволяет работать с расширением *QT Design* [20], который позволяет создавать удобный и надежный интерфейс.

Для упрощения и автоматизации сборки проекта используется утилита *cmake* [21].

## 3.2 Реализация алгоритмов

На листингах 3.1 и 3.2 представлена реализация алгоритма генерации шума Перлина.

На листинге 3.3 представлена реализация функции расчета интенсивности света.

На листинге 3.4 представлена реализация афинных преобразований поворота, масштабирования и переноса.

Листинг 3.1 – Реализация алгоритма шума Перлина с применением нескольких октав

```
double PerlinNoise::generateNoise(const double x, const double y)
{
    int octaves = this->octaves;
    double frequency = this->frequency;
    double amplitude = this->amplitude;

    double result = 0;
    double sum_ampl = 0;

    for (; octaves > 0; octaves--)
    {
        result += this->noise(x * frequency, y * frequency) *
                  amplitude;
        sum_ampl += amplitude;

        amplitude *= this->persistence;
        frequency *= this->lacunarity;
    }

    return result / sum_ampl;
}
```

Листинг 3.2 – Реализация алгоритма шума Перлина для текущей октавы

```
double PerlinNoise::_noise(double x, double y)
{
    int xi = (int)(floor(x)) & 255;
    int yi = (int)(floor(y)) & 255;
    x -= floor(x);
    y -= floor(y);
    double u = smooth(x), v = smooth(y);

    double tl = grad(this->pTable[this->pTable[xi] + yi], x, y,
                      0);
    double tr = grad(this->pTable[this->pTable[xi] + yi + 1], x,
                      y - 1, 0);
    double bl = grad(this->pTable[this->pTable[xi + 1] + yi +
                      1], x - 1, y - 1, 0);
    double br = grad(this->pTable[this->pTable[xi + 1] + yi], x
                      - 1, y, 0);

    double x1 = this->lerp(tl, br, u), x2 = this->lerp(tr, bl, u);

    return this->lerp(0, 1, this->lerp(x1, x2, v));
}
```

Листинг 3.3 – Реализация функции расчета интенсивности света согласно с моделью освещения Ламберта

```
double LightManager::calcIntensityAtVertex(const Light &light,
                                            const QVector3D
                                              &direction,
                                            const QVector3D &normal)
{
    double _scalar_product = QVector3D::dotProduct(direction,
                                                    normal);
    double I = light.getI_0() * light.getK_d() * std::max(0.0,
                                                          _scalar_product);

    return I;
}
```

### Листинг 3.4 – Реализация афинных преобразований

```
void Transform::rotate(QVector3D &point, const Rotate &rotate,
    QVector3D &centerRotate)
{
    rotateByX(point, rotate.xAngle, centerRotate);
    rotateByY(point, rotate.yAngle, centerRotate);
    rotateByZ(point, rotate.zAngle, centerRotate);
}

void Transform::move(QVector3D &point, const Move &move)
{
    point.setX(point.x() + move.dx);
    point.setY(point.y() + move.dy);
    point.setZ(point.z() + move.dz);
}

void Transform::scale(QVector3D &point, const Scale &scale,
    QVector3D &centerScale)
{
    point.setX(scale.kx * point.x() + (1 - scale.kx) *
        centerScale.x());
    point.setY(scale.ky * point.y() + (1 - scale.ky) *
        centerScale.y());
    point.setZ(scale.kz * point.z() + (1 - scale.kz) *
        centerScale.z());
}
```

Листинг 3.5 – Реализация алгоритма закраски по Гуро в сочетании с алгоритмом удаления невидимых линий, использующим Z-буфер (часть 1)

```
void Renderer::_renderPlane(const Plane &screenPlane, const
    vector<double> &heights, const vector<double> &intensity, const
    double waterlevel, const double maxHeight)
{
    double yMin = screenPlane.getPMin().y();
    double yMax = screenPlane.getPMax().y();
    double a = screenPlane.getA(), c = screenPlane.getc();
    vector<Pixel> line1 =
        this->_getLineByBresenham(screenPlane.getP1(),
        screenPlane.getP2());
    vector<Pixel> line2 =
        this->_getLineByBresenham(screenPlane.getP2(),
        screenPlane.getP3());
    vector<Pixel> line3 =
        this->_getLineByBresenham(screenPlane.getP3(),
        screenPlane.getP1());

    this->_calcParamsLine(line1, intensity[0], intensity[1],
        heights[0], heights[1]);
    this->_calcParamsLine(line2, intensity[1], intensity[2],
        heights[1], heights[2]);
    this->_calcParamsLine(line3, intensity[2], intensity[0],
        heights[2], heights[0]);

    vector<Pixel> allLines;
    allLines.insert(allLines.end(), line1.begin(), line1.end());
    allLines.insert(allLines.end(), line2.begin(), line2.end());
    allLines.insert(allLines.end(), line3.begin(), line3.end());
#pragma omp parallel for
    for (int y = yMin; y <= yMax; ++y)
    {
        vector<Pixel> yn;
        std::copy_if(allLines.begin(), allLines.end(),
            std::back_inserter(yn), [y](const Pixel &point)
                { return point.vec.y() == y; });
    }
}
```

Листинг 3.6 – Реализация алгоритма закраски по Гуро в сочетании с алгоритмом удаления невидимых линий, использующим Z-буфер (часть 2)

```
std::sort(yn.begin(), yn.end(), [] (const Pixel &a, const Pixel &b)
           { return a.vec.x() < b.vec.x(); });

double deltaX = yn[yn.size() - 1].vec.x() - yn[0].vec.x();
double x0 = yn[0].vec.x();

double z = screenPlane.caclZ(x0, y);

double I = yn[0].I, deltaI = I - yn[yn.size() - 1].I;
double height = yn[0].vec.z(), deltaZ = height -
    yn[yn.size() - 1].vec.z();

for (int x = yn[0].vec.x(); x <= yn[yn.size() -
    1].vec.x(); ++x)
{
    int sceneX = this->toSceneX(x), sceneY =
        this->toSceneY(y);
    if (sceneX < 0 || sceneX >= this->_screenWidth ||
        sceneY < 0 || sceneY >= this->_screenHeight)
        continue;

    double deltaU = (x - x0) / deltaX - 0;

    if (z > this->_zbuffer[sceneX][sceneY])
    {
        this->_zbuffer[sceneX][sceneY] = z;
        this->_framebuffer.setPixelColor(sceneX, sceneY,
            this->_getColorByHeight(I, height, maxHeight,
                waterlevel));
    }
    z -= (a / c);
    I += (deltaI * deltaU);
    height += (deltaZ * deltaU);
}
}
```

### 3.3 Интерфейс программы

На рисунке 3.1 интерфейс реализованного ПО.

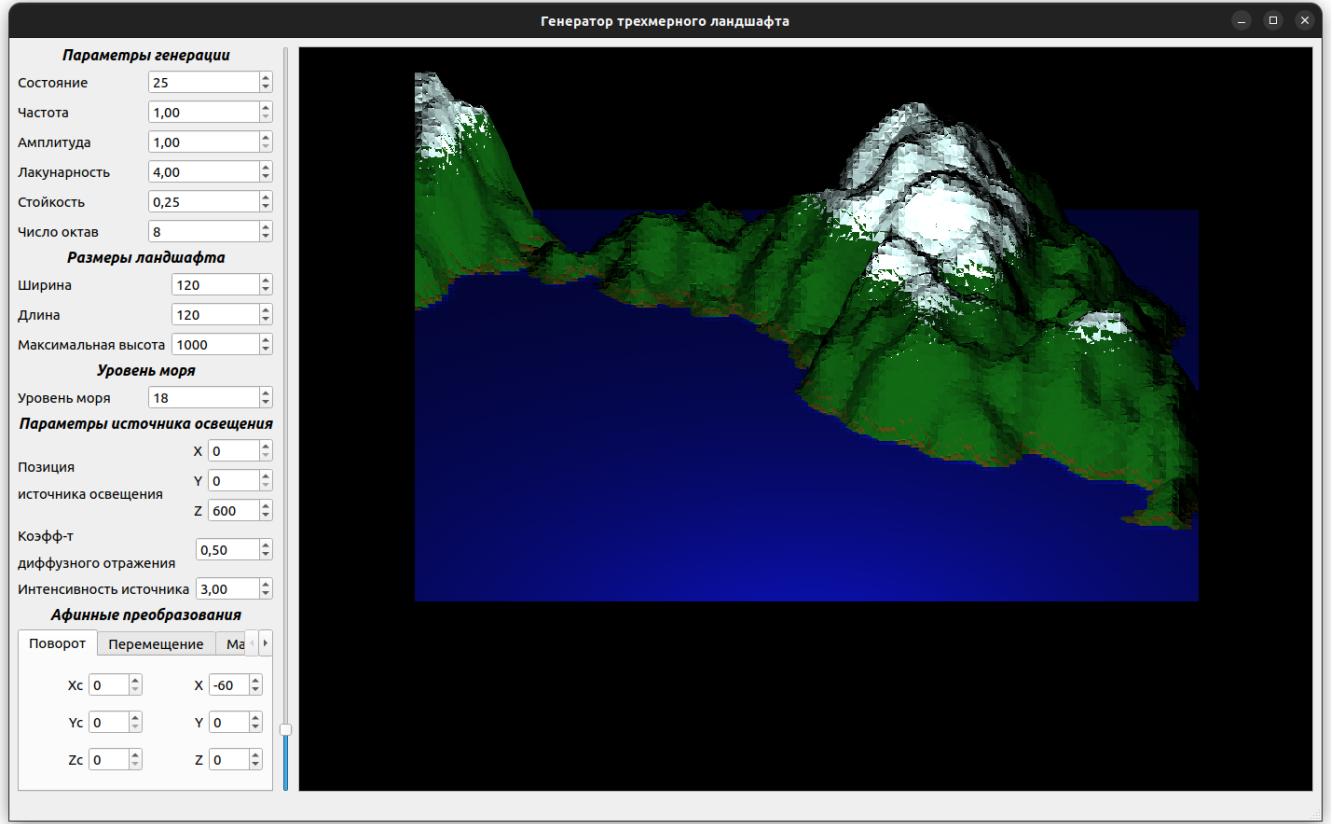


Рисунок 3.1 – интерфейс реализованного ПО

Пользователь может изменять вид ландшафта следующим образом:

- 1) изменить параметры генерации;
- 2) изменить размеры ландшафта;
- 3) изменить положение и параметры точечного источника света;
- 4) изменить уровень моря с помощью слайдера или через поле ввода;
- 5) осуществить поворот, перенос и масштабирование ландшафта.

### 3.4 Демонстрация работы программы

На рисунках 3.2 показаны примеры работы программы.

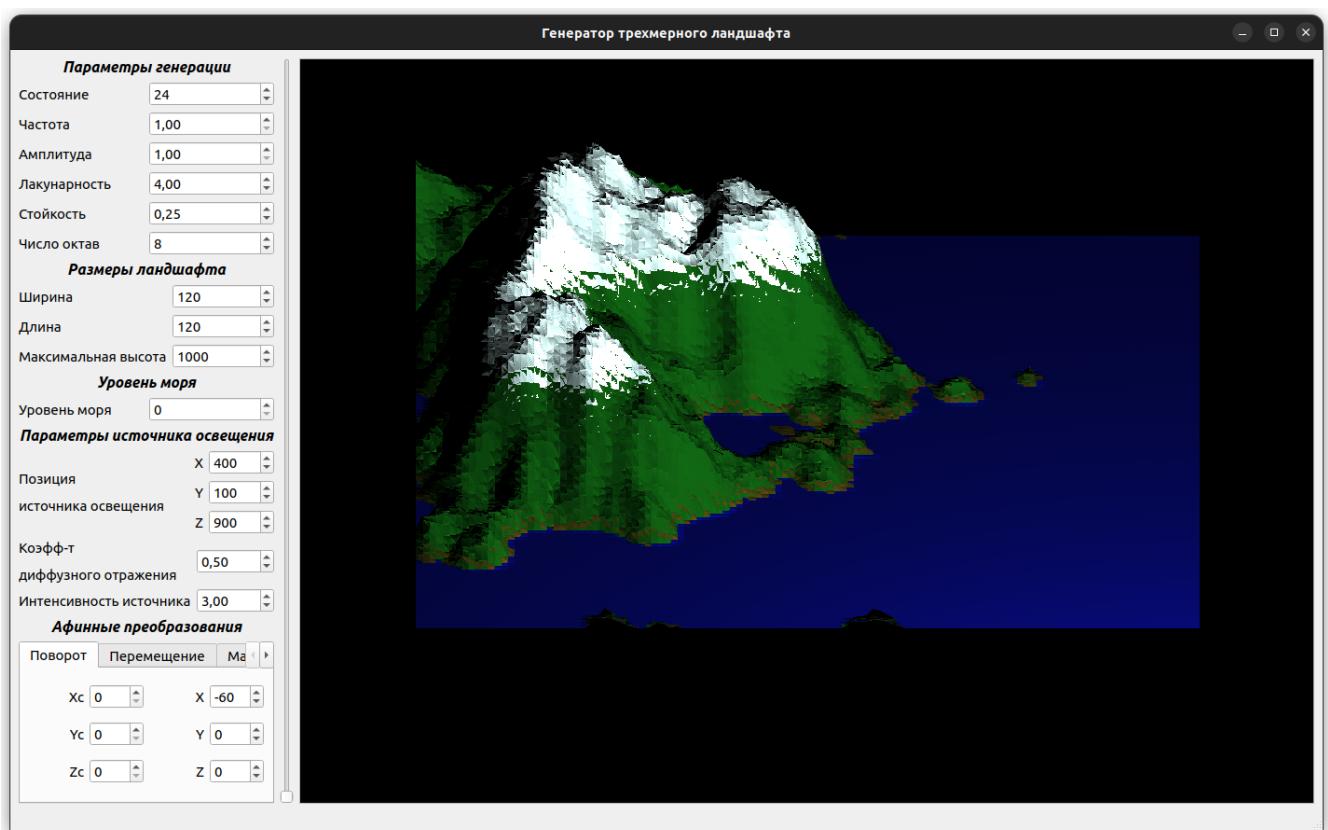


Рисунок 3.2 – Пример ландшафта

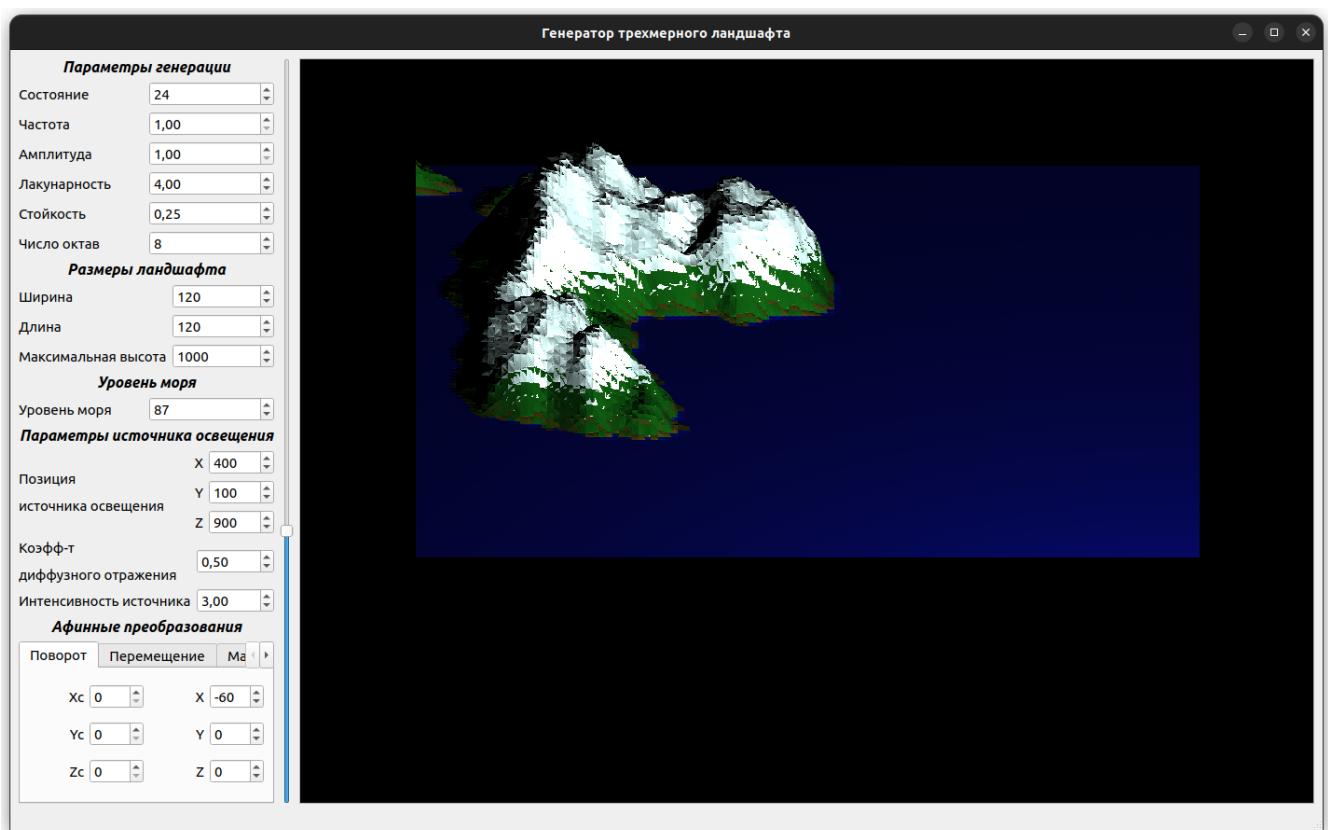


Рисунок 3.3 – Пример изменения уровня моря в ландшафте

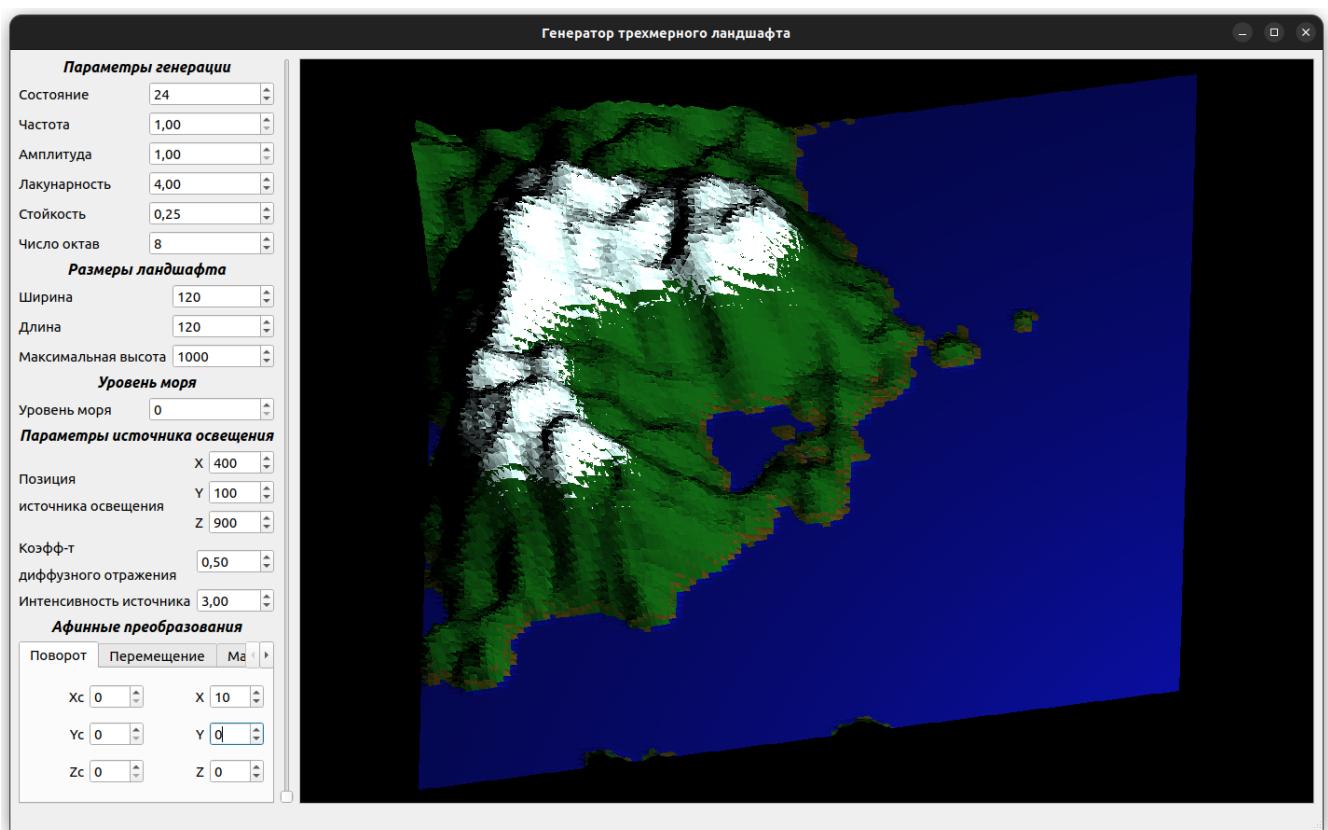


Рисунок 3.4 – Пример применения операции поворота к ландшафту

## **4 Исследовательский раздел**

В разделе представлены результаты проведенных исследований, в котором сравниваются временные характеристики работы реализованного программного обеспечения.

### **4.1 Технические характеристики**

Технические характеристики устройства, на котором проводились исследования:

- операционная система: Ubuntu 22.04.3 LTS x86\_64 [22];
- оперативная память: 16 Гб;
- процессор: 11th Gen Intel® Core™ i7-1185G7 @ 3.00 ГГц × 8.

Исследования проводились на ноутбуке, который был подключен к сети электропитания. Во время проведения исследований ноутбук был загружен только встроенными приложениями окружения и самим окружением.

### **4.2 Проведение первого исследования**

#### **4.2.1 Цель первого исследования**

Целью первого исследования является проведение сравнительного анализа времени построения одного кадра ландшафта в зависимости от его размеров, заданных в количестве треугольных полигонов.

#### **4.2.2 Наборы варьируемых и фиксированных параметров**

Замеры проводились для ландшафта размером  $10 \times 10$ ,  $50 \times 50$ ,  $100 \times 100$ ,  $200 \times 200$ ,  $300 \times 300$ ,  $400 \times 400$ ,  $500 \times 500$ ,  $600 \times 600$ ,  $700 \times 700$ .

Для каждого размера ландшафта измерялось время 20 визуализаций, после чего вычислялось среднее арифметическое из полученных значений.

В качестве фиксированных параметров были выбраны следующие:

- состояние (равно 25)
- частота (равна 1)
- амплитуда (равна 1)
- лакунарность (равна 4)
- стойкость (равна 0.25)
- число октав (равно 8)
- максимальная высота (равно 1000)
- уровень моря (равен 0)
- позиция источника освещения (равна точке с координатами (0, 0, 600))
- коэффициент диффузного отражения источника (равна 0.5)
- интенсивность источника (равна 3)

### **4.2.3 Результаты первого исследования**

Результаты замеров времени построения одного кадра ландшафта от размеров ландшафта приведены в таблице 4.1 и на рисунке 4.1.

Таблица 4.1 – Результаты замеров времени построения одного кадра ландшафта от размеров ландшафта

Размеры ландшафта, количество треугольных полигонов	Время построения, секунды
$10 \times 10$	0.035
$50 \times 50$	0.106
$100 \times 100$	0.340
$200 \times 200$	1.134
$300 \times 300$	2.595
$400 \times 400$	4.899
$500 \times 500$	8.524
$600 \times 600$	12.226
$700 \times 700$	17.154

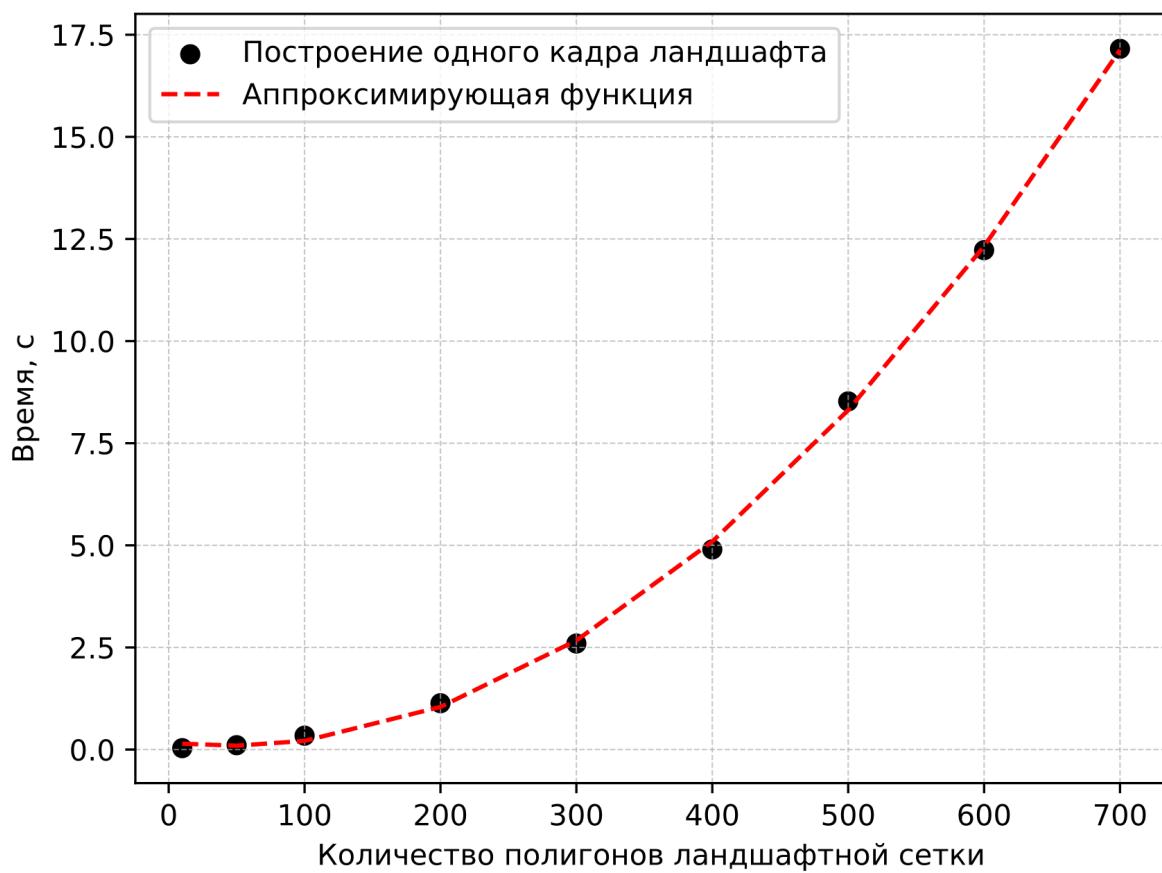


Рисунок 4.1 – Результаты замеров времени построения одного кадра ландшафта от размеров ландшафта

С помощью метода наименьших квадратов было установлено, что полученные дискретные значения результатов аппроксимируются квадратичной функцией вида  $a \cdot x^2 + b \cdot x + c$  с невязкой, имеющей значение  $\approx 0.135$ .

Таким образом, по результатам первого исследования можно сделать вывод о том, что в зависимости от размеров ландшафта наблюдается квадратичная зависимость времени построения кадра. Это означает, что увеличение размеров ландшафта приводит к более значительному увеличению времени построения кадра.

## **4.3 Проведение второго исследования**

### **4.3.1 Цель второго исследования**

Целью второго исследования является проведение сравнительного анализа времени генерации одного кадра ландшафта в зависимости от количества октав.

### **4.3.2 Наборы варьируемых и фиксированных параметров**

Замеры проводились для ландшафта с количеством октав, равным 1, 2, 3, 4, 5, 6, 7, 8, 9.

В качестве фиксированных параметров были выбраны следующие:

- состояние (равно 25)
- частота (равна 1)
- амплитуда (равна 1)
- лакунарность (равна 4)
- стойкость (равна 0.25)
- размеры ландшафта (равны  $120 \times 120$ )
- максимальная высота (равно 1000)
- уровень моря (равен 0)
- позиция источника освещения (равна точке с координатами  $(0, 0, 600)$ )
- коэффициент диффузного отражения источника (равна 0.5)
- интенсивность источника (равна 3)

### **4.3.3 Результаты второго исследования**

Результаты замеров времени построения одного кадра ландшафта от его размеров приведены в таблице 4.2 и на рисунке 4.2.

Таблица 4.2 – Результаты замеров времени построения одного кадра ландшафта от количества октав

Число октав	Время построения, секунды
1	0.405
2	0.424
3	0.442
4	0.468
5	0.503
6	0.512
7	0.521
8	0.529
9	0.548

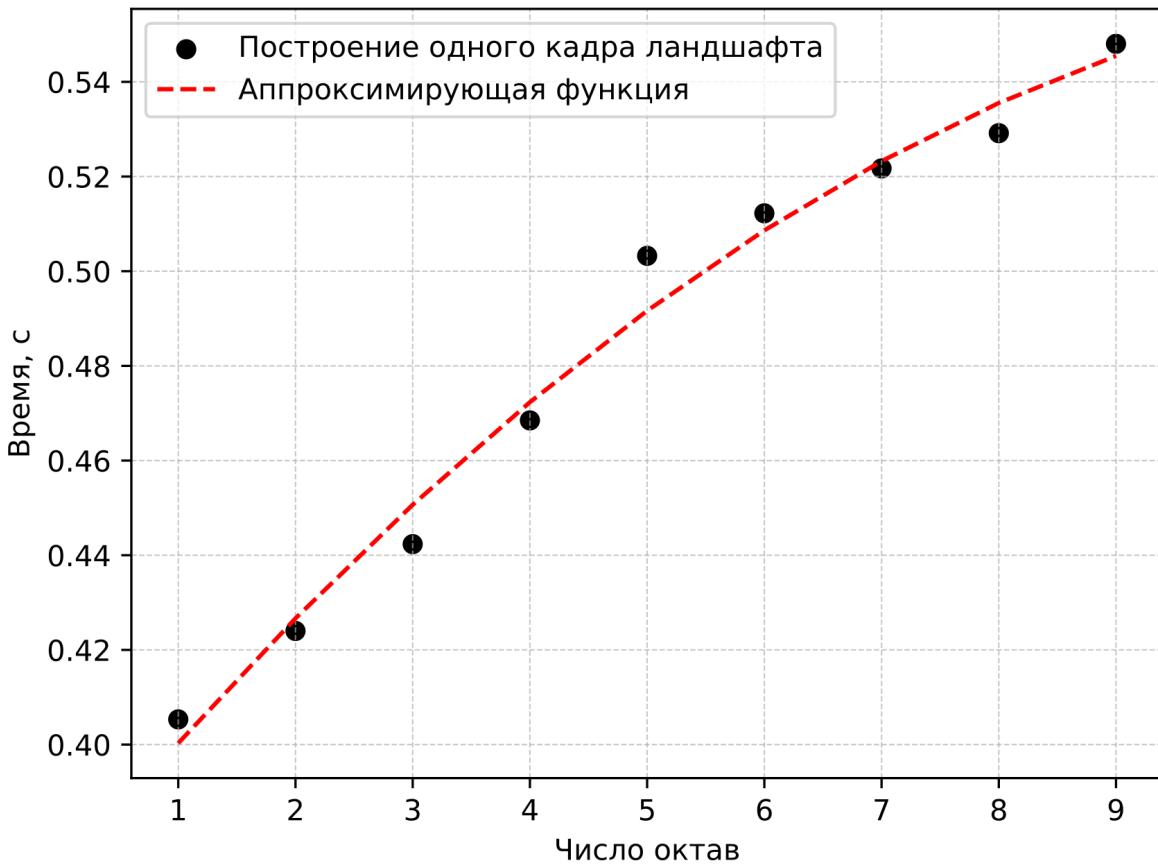


Рисунок 4.2 – Результаты замеров времени построения одного кадра ландшафта от количества октав

С помощью метода наименьших квадратов было установлено, что полученные дискретные значения результатов аппроксимируются квадратичной функцией вида  $a \cdot x^2 + b \cdot x + c$  с невязкой, имеющей значение  $\approx 0.0003$ .

С ростом числа октав наблюдается незначительный рост времени построения одного кадра: для построения одного кадра ландшафта с 9-ю октавами требуется всего в 1.35 раз больше времени, чем для построения одного кадра ландшафта с 1-й октавой.

## **Вывод**

В данном разделе были проведены исследования по замеру времени построения одного кадра ландшафта от размеров ландшафта и времени построения одного кадра ландшафта от числа октав.

По результатам обоих исследований можно сделать вывод о том, что формирование карты высот на основе шума Перлина с использованием нескольких октав требует разные временные затраты. Время построения одного кадра ландшафта в зависимости от размеров ландшафта имеет квадратичную зависимость и начинает резко увеличиваться начиная с размеров ландшафта в  $200 \times 200$ : для отображения ландшафта такой размерности по сравнению с размером  $100 \times 100$  требуется в 3.33 раза больше времени, для отображения ландшафта размером  $700 \times 700$  по сравнению с размером  $300 \times 300$  требуется в 6.61 раза больше времени.

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта были решены следующие задачи:

- 1) выполнена формализация объектов синтезируемой сцены;
- 2) проведен анализ существующих алгоритмов создания ландшафта и визуализации сцены;
- 3) выбраны подходящие алгоритмы для решения поставленной задачи.
- 4) реализованы выбранные алгоритмы;
- 5) проведено исследование временных характеристик выбранных алгоритмов.

Целью работы, а именно разработка программного обеспечения для генерации и визуализации трехмерного ландшафта также была достигнута.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Генерация трехмерных ландшафтов [Электронный ресурс]. — Режим доступа: <https://www.ixbt.com/video/3dterrains-generation.shtml> (дата обращения: 03.07.2023).
2. Применение дерева квадрантов в визуализации ландшафтов с изменяемым уровнем детализации [Электронный ресурс]. — Режим доступа: [https://www.elibrary.ru/download/elibrary\\_34887617\\_70147073.pdf](https://www.elibrary.ru/download/elibrary_34887617_70147073.pdf) (дата обращения: 19.10.2023).
3. Исследование методов компьютерного моделирования поверхностей Земли для тестирования программного обеспечения обработки изображений со спутников [Электронный ресурс]. — Режим доступа: [https://www.elibrary.ru/download/elibrary\\_39141278\\_24936959.pdf](https://www.elibrary.ru/download/elibrary_39141278_24936959.pdf) (дата обращения: 19.10.2023).
4. Применение алгоритма midpoint-displacement в процедурной генерации ландшафтов [Электронный ресурс]. — Режим доступа: <https://cyberleninka.ru/article/n/primenie-algoritma-midpoint-displacement-v-protsedurnoy-generatsii-landshaftov> (дата обращения: 19.10.2023).
5. Процедурная генерация текстур на основе алгоритма Diamond-Square [Электронный ресурс]. — Режим доступа: [https://libeldoc.bsuir.by/bitstream/123456789/31470/1/Multan\\_Protsedurnaya.PDF](https://libeldoc.bsuir.by/bitstream/123456789/31470/1/Multan_Protsedurnaya.PDF) (дата обращения: 19.10.2023).
6. Diamond-square algorithm [Электронный ресурс]. — Режим доступа: [https://en.wikipedia.org/wiki/Diamond-square\\_algorithm](https://en.wikipedia.org/wiki/Diamond-square_algorithm) (дата обращения: 04.07.2023).
7. Terrain Generation Tutorial: Hill Algorithm [Электронный ресурс]. — Режим доступа: <https://www.stuffwithstuff.com/robot-frog/3d/hills/hill.html> (дата обращения: 08.07.2023).

8. Аникеев А. Генерация и моделирование 2D ландшафта по контрольным значениям с использованием Unity на языке программирования c# // . — Новосибирск, Новосибирский государственный технический университет, 2020. — С. 9.
9. Perlin noise [Электронный ресурс]. — Режим доступа: [https://en.wikipedia.org/wiki/Perlin\\_noise](https://en.wikipedia.org/wiki/Perlin_noise) (дата обращения: 08.07.2023).
10. Снук Г. 3D-ландшафты в реальном времени на C++ и DirectX 9 //. — Пер. С англ М.: КУДИЦ-ОБРАЗ, 2007. — С. 368.
11. Д. Р. Алгоритмические основы машинной графики: Пер. с англ. //. — СПб: БХВ-Петербург, 1989. — С. 512.
12. Простые модели освещения [Электронный ресурс]. — Режим доступа: <https://cgraph.ru/node/435> (дата обращения: 16.07.2023).
13. Афинные преобразование объектов в компьютерной графике [Электронный ресурс]. — Режим доступа: [https://www.elibrary.ru/download/elibrary\\_22077864\\_43675515.pdf](https://www.elibrary.ru/download/elibrary_22077864_43675515.pdf) (дата обращения: 17.12.2023).
14. Иванов В.П. Б. А. Трехмерная компьютерная графика //. — под ред. Г.М. Полищук. М.: Радио и связь, 1995. — С. 224.
15. Справочник по языку C++ [Электронный ресурс]. — Режим доступа: <https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-170> (дата обращения: 28.09.2022).
16. Сравнение языков программирования C++ и Python [Электронный ресурс]. — Режим доступа: [https://www.elibrary.ru/download/elibrary\\_42416104\\_98504159.pdf](https://www.elibrary.ru/download/elibrary_42416104_98504159.pdf) (дата обращения: 18.12.2023).
17. C++ против Java: базовое сравнение, ключевые различия и сходства [Электронный ресурс]. — Режим доступа: <https://itgap.ru/post/cpp-vs-java> (дата обращения: 18.12.2023).
18. Qt Creator Manual [Электронный ресурс]. — Режим доступа: <https://doc.qt.io/qtcreator/> (дата обращения: 18.12.2023).

19. All Qt Documentation [Электронный ресурс]. — Режим доступа: <https://doc.qt.io/all-topics.html> (дата обращения: 18.12.2023).
20. Qt Designer Manual [Электронный ресурс]. — Режим доступа: <https://doc.qt.io/qt-6/qtdesigner-manual.html> (дата обращения: 18.12.2023).
21. CMake Documentation and Community [Электронный ресурс]. — Режим доступа: <https://cmake.org/documentation/> (дата обращения: 18.12.2023).
22. Ubuntu 22.04.3 LTS (Jammy Jellyfish) [Электронный ресурс]. — Режим доступа: <https://releases.ubuntu.com/22.04/> (дата обращения: 28.09.2022).