СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Построение ландшафта - это задача компьютерной графики, которая состоит в генерации и визуализации модели местности с учетом рельефа, освещения, времени года и других релевантных параметров. Программное обеспечение для построения ландшафта находит применение и разработке компьютерных игр [1], планировании городских пространств [2] и других связанных с моделированием местности областях.

Цель работы – разработка программного обеспечения для построения трехмерного горного ландшафта. Реализованная программа должна иметь интерфейс для вращения и масштабирования модели, редактирования параметров ландшафта и изменения его текстур в зависимости от времени года (зима, лето).

Для достижения поставленной цели следует решить следующие задачи:

- описать существующие алгоритмы программного построения ландшафта и выбрать наиболее подходящие для решения поставленной задачи;
- спроектировать выбранные алгоритмы;
- на основе спроектированных алгоритмов реализовать и протестировать программное обеспечение для построения ландшафта;
- провести серию исследований производительности разработанного программного обеспечения.

1 Аналитический раздел

Раздел содержит описание алгоритмов, которые используются для построения ландшафта. В конце раздела приведено сравнение описанных алгоритмов. На основе сравнения сделан выбор алгоритмов для последующего проектирования и реализации.

1.1 Способы представления карты высот

Для каждой точки плоскости требуется хранить ее высоту. Выделяют три способа хранения информации о высотах ландшафта [3]:

1) регулярная сетка;

2) симплексная сетка;

3) карта сегментов.

1.1.1 Регулярная сетка

Плоскость $Z = 0$ трехмерного пространства рассекается сеткой с регулярным шагом δ, образуя группу ячеек. Каждая ячейка содержит информацию о высоте соответствующей точки и, возможно, дополнительную информацию (цвет данной точки и т.п.). Формально данная структура представляет двумерную матрицу $Mesh$, где $Mesh_{x,y}$ – информация о точке с соответствующими координатами в плоскости $Z = 0$ [3]. От величины параметра δ зависит детализация ландшафта.

1.1.2 Симплексная сетка

Симплекс в пространстве размерности n – это оболочка, образованная $(n + 1)$ вершиной, которые не лежат в одном пространстве размерности n [4]. Для двумерного пространства симплекс – это треугольник. Соответственно, наложение на плоскость симплексной сетки предполагает ее разбиение на треугольники, обычно равносторонние.

1.1.3 Карта сегментов

Подобно способу регулярной сетки, плоскость $Z = 0$ рассекается сеткой с заданным шагом δ, но теперь в соответствующей ячейке хранится информация

не о конкретной точке, а о целом сегменте в ее окрестности. Данные о сегментах хранятся в отдельных структурах, а ячейки матрицы хранят указатели на них. Данное представление ландшафта используется для построения больших пространств, не требующих высокой степени детализации [3].

1.2 Алгоритмы генерации ландшафта

Перед хранением высоты ландшафта требуется получить. Для этой задачи предназначены следующие алгоритмы.

1.2.1 Алгоритм Value Noise

Алгоритм Value Noise – наиболее примитивный алгоритм генерации высот ландшафта. Высота точки пространства зависит от случайной величины – значения шумовой функции. Отсутствие влияния на значение функции делает невозможным возможность модификации ландшафта. Случайность значения функции влечет резкость и рваность ландшафта. Добиться сглаживания можно за счет применения методов интерполяции, однако данный алгоритм все равно не подходит для построения детализированных ландшафтов с возможностью модификации [5].

1.2.2 Алгоритм Perlin Noise

Алгоритм призван снизить степень хаотичности получаемых значений высот и предоставить контроль над ними.

Кен Перлин уточнил понятие шумовой функции, обозначив следующие свойства [6]:

- функция является отображением $R^n \to R$, то есть ставит в соответствие набору $(x_1, ..., x_n)$ одно вещественное число;
- функция принимает вещественные значения из интервала $[0, 1]$;
- функция псевдослучайна, несмотря на то, что результат кажется случайным, для фиксированного набора входных параметров функция будет давать один и тот же результат.

Значение высоты данной точки плоскости напрямую связано со значением шумовой функции в данной точке. Значение шумовой функции фактически

зависит от положения точки в некоторой ячейке регулярной сетки и параметров четырех узлов данной ячейки. [6]. Этапы вычисления шумовой функции

1) наложить на плоскость пространства регулярную сетку;

2) каждому узлу сетки поставить в соответствие единичный вектор $\overrightarrow{g_i}$ некоторого направления;

3) определить ячейку сетки, в которую попадает рассматриваемая точка пространства (x, y);

4) определить попарные скалярные произведения векторов $\overrightarrow{g_0}$, $\overrightarrow{g_1}$, $\overrightarrow{g_2}$, $\overrightarrow{g_3}$ узлов ячейки на векторы $\overrightarrow{v_0}$, $\overrightarrow{v_1}$, $\overrightarrow{v_2}$, $\overrightarrow{v_3}$, проведенные из узлов в рассматриваемую точку пространства;

5) провести двумерную интерполяцию полученных значений для определения искомой величины в рассматриваемой точке пространства.

Полученное в результате работы шумовой функции значение масштабируется в соответствии с размерами сцены и является высотой точки x, y.

Для более реалистичной генерации высот ландшафта изложенные выше действия проделываются несколько раз. На каждом этапе ячейкам сетки соответствуют разные векторы, а отдельный этап именуется октавой. Результирующей высотой точки является некоторая комбинация вычисленных значений в каждой из октав.

Результирующий ландшафт может быть модифицирован посредством изменения входных параметров алгоритма: количества октав, количества узлов сетки, метода комбинирования значений нескольких октав.

Недостатком алгоритма является периодическое возникновение злокачественных артефактов и повторяющихся элементов.

1.2.3 Алгоритм Simplex Noise

Алгоритм является усовершенствованной версией алгоритма Perlin Noise, он призван снизить количество визуальных артефактов. Идея алгоритма – рассчет вклада в результирующий шум в точка относительно трех вершин симплекса на симплексной сетке, в который попала точка. В двумерном случае такая сетка получается путем сжатия обычной регулярной сетки вдоль диагонали и делением ее на треугольники.

Шаги алгоритма для вычисления шума в точке (x, y):

1) вычислить положение $(i,\ j)$ точки (x, y) на симплексной сетке;

2) вычислить вершины (x_0, y_0), (x_1, y_1), (x_2, y_2) исходной регулярной сетки, которые соответствуют вершинам симплекса симплексной сетки, в который попала точка (x, y) после смещения;

3) вычислить вклады n_0, n_1, n_2 от каждой вершины симплекса в шум точки x, y аналогично алгоритму Perlin Noise;

4) вычислить результирующий шум в точке (x, y) как комбинацию вкладов вершин симплекса.

Положение точки (x, y) на симплексной сетке вычисляется согласно соотношениям:

$$\begin{cases} i = \lfloor x + (x + y) \cdot F \rfloor \\ j = \lfloor y + (x + y) \cdot F \rfloor, \end{cases} \tag{1.1}$$

где коэффициент сжатия F определен как:

$$F = \frac{\sqrt{3} - 1}{2}.$$

Координаты точек регулярной сетки, соответствующие вершинам симплекса, в котором оказалась точка (x, y) после смещения, вычисляются согласно соотношениям:

$$\begin{cases} (x_0, y_0) = (x - i - (i + j) \cdot G, y - i - (i + j) \cdot G) \\ (x_1, y_1) = (x_0 - i + G, y_0 - j + G) \\ (x_2, y_2) = (x_0 - 1 + 2 \cdot G, y_0 - 1 + 2 \cdot G), \end{cases} \tag{1.2}$$

где коэффициент растяжения G определен следующим образов:

$$G = \frac{3 - \sqrt{3}}{6}.$$

Вклад n_i, $i \in \{0, 1, 2\}$ есть величина скалярного произведения вектора $\overrightarrow{g_i}$, ассоциированного с вершиной n_i, и вектора $\overrightarrow{v_i}$, проведенного из вершины n_i в рассматриваемую точку (x, y).

Результирующая величина шума есть некоторая комбинация величин n_0, n_1, n_2. Вид комбинации подбирается, исходя из требований к результирующему виду распределения шума.

Алгоритм предполагает использование нескольких этапов вычислений (октав) для более реалистичной генерации высот. То есть описанный выше процесс можно провести несколько раз, получить несколько величин шума в каждой точке и скомбинировать их в результирующее значение некоторым образом.

Переход в симплексную сетку позволяет снизить количество узлов, участвующих в вычислении шума и реализует меньше количество артефактов и повторяющихся элементов [4].

1.3 Алгоритмы удаления невидимы ребер

1.3.1 Алгоритм Робертса

Алгоритм Робертса поддерживает исключительно выпуклые объекты. В основе алгоритма лежит последовательное удаление линий, экранируемых самим объектом и другими объектами сцены. Если объект является невыпуклым, его необходимо разбить на выпуклые подобъекты и применить алгоритм к ним [7].

Алгоритм состоит из следующих шагов:

1) подготовить данные, сформировать матрицу объекта;

2) удалить невидимые рёбра объекта, экранируемые его собственными гранями;

3) исключить рёбра, скрытые другими объектами сцены;

4) устранить линии пересечения объектов, экранируемых самими объектами и другими элементами.

В худшем случае при наличии N ребер на сцене количество операций сравнения при подготовке кадра составит $O(N^3)$. Алгоритм не предполагает отображения теней [8].

1.3.2 Алгоритм Z-буфера

Алгоритм Z-буфера использует два буфера: буфер кадра для хранения атрибутов пикселей (цвет или интенсивность) и Z-буфер для хранения значений глубины каждого пикселя сцены [7]. Стоит заметить, что в данном контексте z – это отдаленность текущей точки от наблюдателя, а не ее высота.

Алгоритм состоит из следующих шагов:

1) инициализировать Z-буфер минимальными значениями глубины, а буфер кадра — фоновым значением;

2) для каждого пикселя объекта вычислить его глубину z и сравнить с уже записанным в Z-буфер значением;

3) если новый пиксель ближе к наблюдателю, обновить оба буфера: в Z-буфер записать новую глубину, а в буфер кадра — новые атрибуты пикселя;

4) если пиксель находится дальше – игнорировать его.

Алгоритм не требует предварительной сортировки объектов по глубине, допускает наличие теней. Требует хранения буферов кадра и координаты z точек. Трудоемкость алгоритма оценивается как $O(C \cdot N)$, где C – количество пикселей окна отрисовки кадра, N – количество ребер [8].

1.4 Модели освещения сцены

1.4.1 Модель Ламберта

Модель Ламберта описывает диффузное освещение, при котором свет рассеивается равномерно во всех направлениях от поверхности объекта. Это предполагает, что яркость объекта зависит только от угла падения света и не зависит от позиции наблюдателя [7].

Алгоритм:

1) определить нормаль к поверхности в каждой точке;
2) вычислить угол между нормалью и направлением на источник света;
3) рассчитать интенсивность освещения как произведение интенсивности источника на косинус угла падения света.

1.4.2 Модель Фонга

Модель Фонга сочетает в себе как диффузное освещение, так и зеркальное отражение света. В этой модели учитывается, что часть света отражается под углом, противоположным углу падения, что создаёт яркие блики на поверхности объекта [7].

Алгоритм:

1) вычислить диффузное освещения аналогично модели Ламберта;
2) определить направления отражённого луча от поверхности;
3) вычислить угол между направлением на наблюдателя и направлением отражённого света;
4) рассчитать зеркальную составляющую освещения, зависящую от этого угла и гладкости поверхности.

1.5 Модели закраски

1.5.1 Закраска по Гуро

Метод Гуро выполняет интерполяцию цвета вдоль граней полигона [7].

Алгоритм:

1) освещение рассчитать только в вершинах многоугольника;

2) значения освещения в вершинах интерполировать вдоль рёбер;

3) полученные значения использовать для закраски внутренних точек многоугольника.

1.5.2 Закраска по Фонгу

Метод Фонга улучшает модель Гуро, интерполируя не освещение, а нормали к поверхности. Это позволяет более точно учитывать изменения освещения [7]. Алгоритм:

1) вычислить нормали в каждой вершине многоугольника.

2) интерполировать величины нормалей вдоль рёбер и внутри многоугольника.

3) освещение рассчитать для каждой точки внутри многоугольника на основе интерполированных нормалей.

Метод Фонга даёт более точное освещение и плавные блики, но требует большего количества вычислений.

1.6 Сравнительный анализ алгоритмов

1.6.1 Алгоритмы генерации ландшафта

Для построения реалистичного горного ландшафта алгоритм должен удовлетворять следующим свойствам:

1) модифицируемость – пользователь должен иметь возможность влиять на результуриующий ландшафт;

2) реалистичность – результатом работы алгоритма должен быть ландшафт с минимальным количеством артефактов.

В таблице 1.1 представлены характеристики описанных ранее алгоритмов генерации ландшафта:

Таблица 1.1 – Сравнение алгоритмов генерации ландшафта

Алгоритм	Количество дополнительных точек для вычисления высоты в одной точке	Возможность модификации ландшафта	Наличие артефактов
Value noise	0	–	–
Perlin noise	4	+	+
Simplex noise	3	+	–

На основе описанных характеристик и требований для реализации был выбран алгоритм Simplex Noise.

1.6.2 Способы представления карты высот

В таблице 1.2 представлены характеристики описанных ранее способов хранения карты высот:

Таблица 1.2 – Сравнение способов представления карты высот

Алгоритм	Возможность матричных преобразований	Совместимость с декартовой системой координат
Регулярная сетка	+	+
Симплексная сетка	–	–
Карта сегментов	–	–

Для хранения информации о ландшафте будет использоваться регулярная сетка. Данный с способ совместим с декартовой системой координат и может быть представлен в виде двумерного массива.

1.6.3 Алгоритмы удаления невидимых ребер

Алгоритм Робертса имеет кубическую трудоемкость в зависимости от количества ребер изображаемого полигона – $O(N^3)$, тогда как алгоритм Z-буфер имеет трудоемкость $O(CN)$, где C – количество пикселей, N – количество ребер [8].

Модель ландшафта имеет огромное количество ребер – каждая точка связана с четырьмя соседними. Потому предпочтение будет отдано алгоритму Z-буфер, так как количество ребер вносит меньший вклад в трудоемкость.

1.6.4 Модели освещения

Модель Фонга является предпочтительной при моделировании сцен с большим количеством световых источников и отражающих поверхностей. В рассматриваемой задаче моделирование отражений не рассматривается, а источник света один, потому для сокращения вычислений и повышения производительности без ущерба реалистичности изображения будет применена модель освещения Гуро.

1.6.5 Методы закраски

При визуализации большой сцены, как и в случае с алгоритмом удаления ребер, крайне важна скорость работы. В рамках отсутствия поверхностей, которые могут создавать блики, для повышения производительности программного обеспечения будет применен метод закраски Гуро.

Вывод

В разделе были описаны некоторые алгоритмы, использующиеся для программного построения горных ландшафтов. В конце было проведено сравнение описанных алгоритмов, на основании чего были выбраны следующие алгоритмы:

– алгоритм Simplex Noise для генерации ландшафта;

– регулярная сетка для хранения высот ландшафта;

– алгоритм Z-буфера для удаления невидимых ребер;

– модель освещения Ламберта;

– закраска по Гуро.

2 Конструкторский раздел

Раздел содержит схемы выбранных для решения поставленной задачи алгоритмов.

2.0.1 Обобщенный алгоритм отрисовки кадра

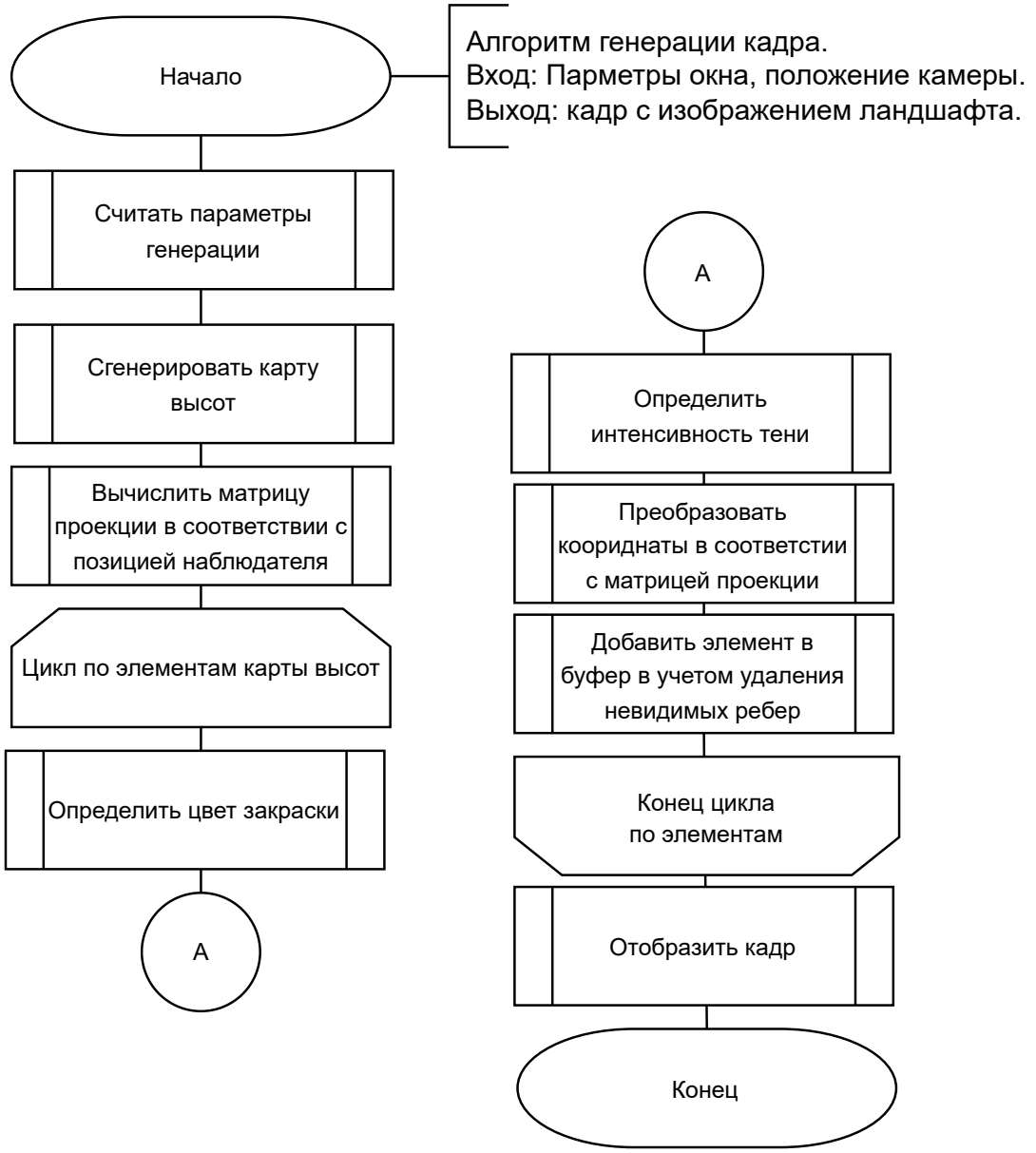

Рисунок 2.1 – Схема обобщенного алгоритма отрисовки кадра

2.0.2 Алгоритм Simplex Noise

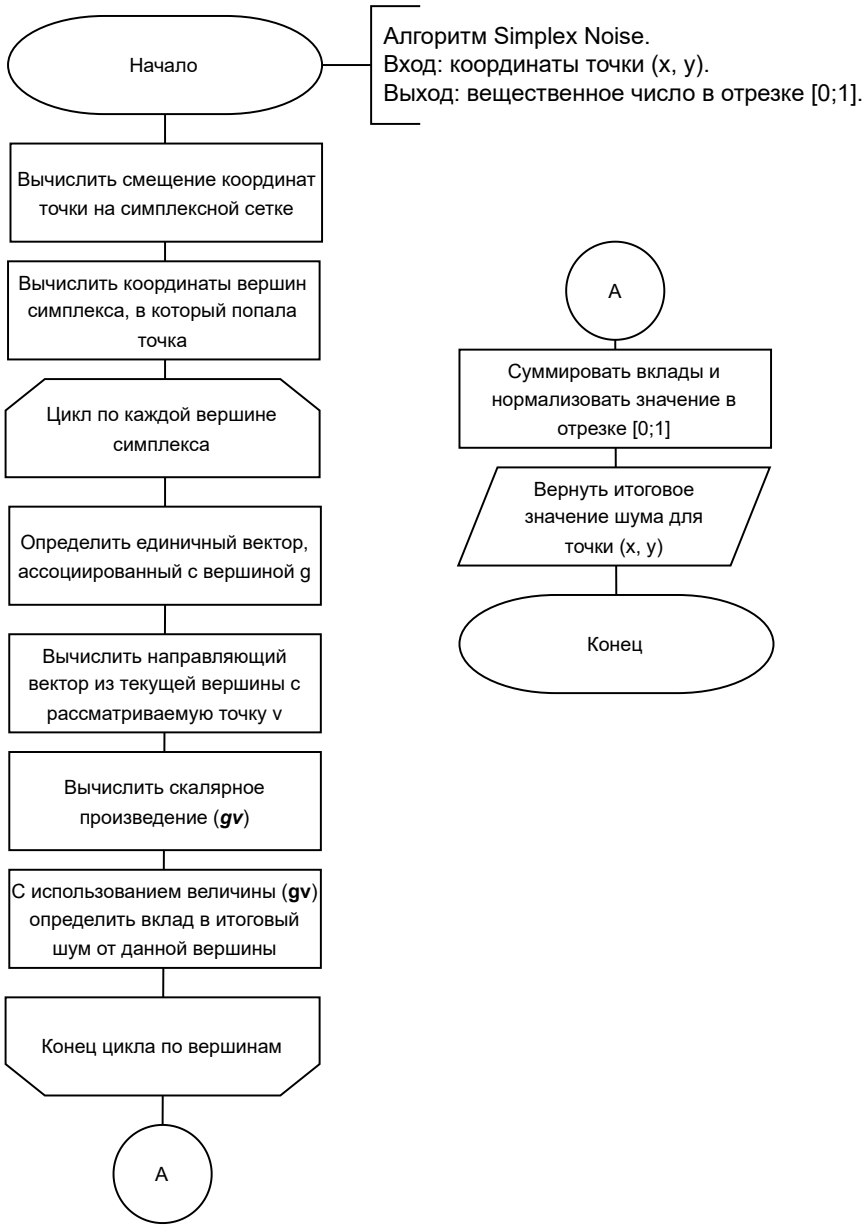

Рисунок 2.2 – Схема алгоритма Simplex Noise

Вклад в величину шума от вершины симплекса n_i, $i \in \{1, 2, 3\}$ определен как

$$n_i = \begin{cases} 0, & t_i < 0, \\ t_i^4 \cdot (\overrightarrow{g_i} \cdot \overrightarrow{v_i}), & t_i \geq 0, \end{cases}$$

где

$$t_i = 0,5 - (x_i - x)^2 - (y_i - y)^2.$$

Вклад n_i считается равным нулю, если расстояние от i-й вершины до точки (x, y) больше $0,5$.

2.0.3 Алгоритм Simplex Noise с комбинацей октав

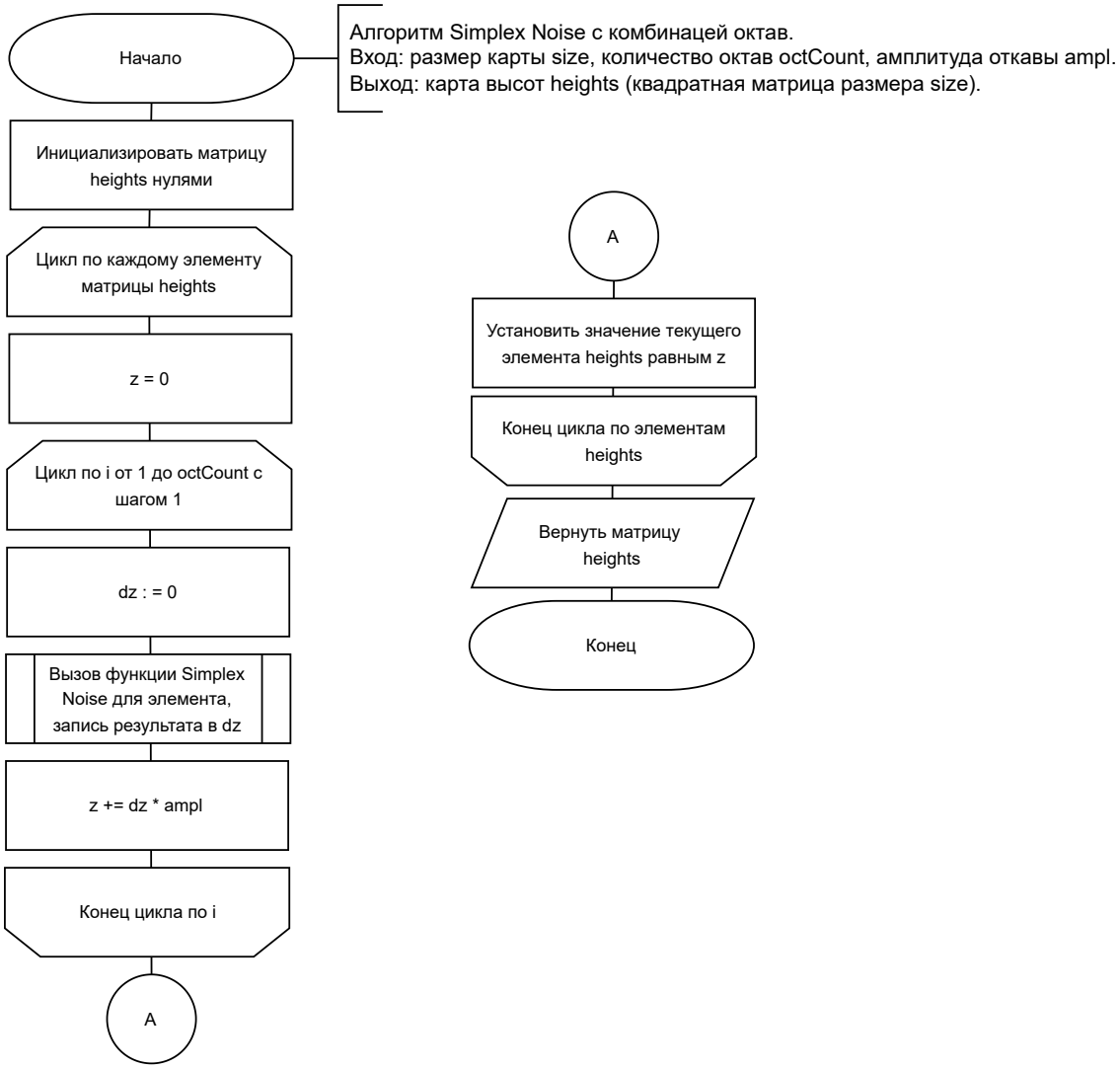

Рисунок 2.3 – Схема алгоритма Simplex Noise с комбинацией октав

Результирующая величина шума определена как взвешенная сумма вида

$$noise(z) = \sum_{i=1}^{octCount} z_i * ampl_i,$$

где $octCount$ – количество октав, $ampl_i$ – вес вклада от текущей октавы.

2.0.4 Алгоритм определения цвета

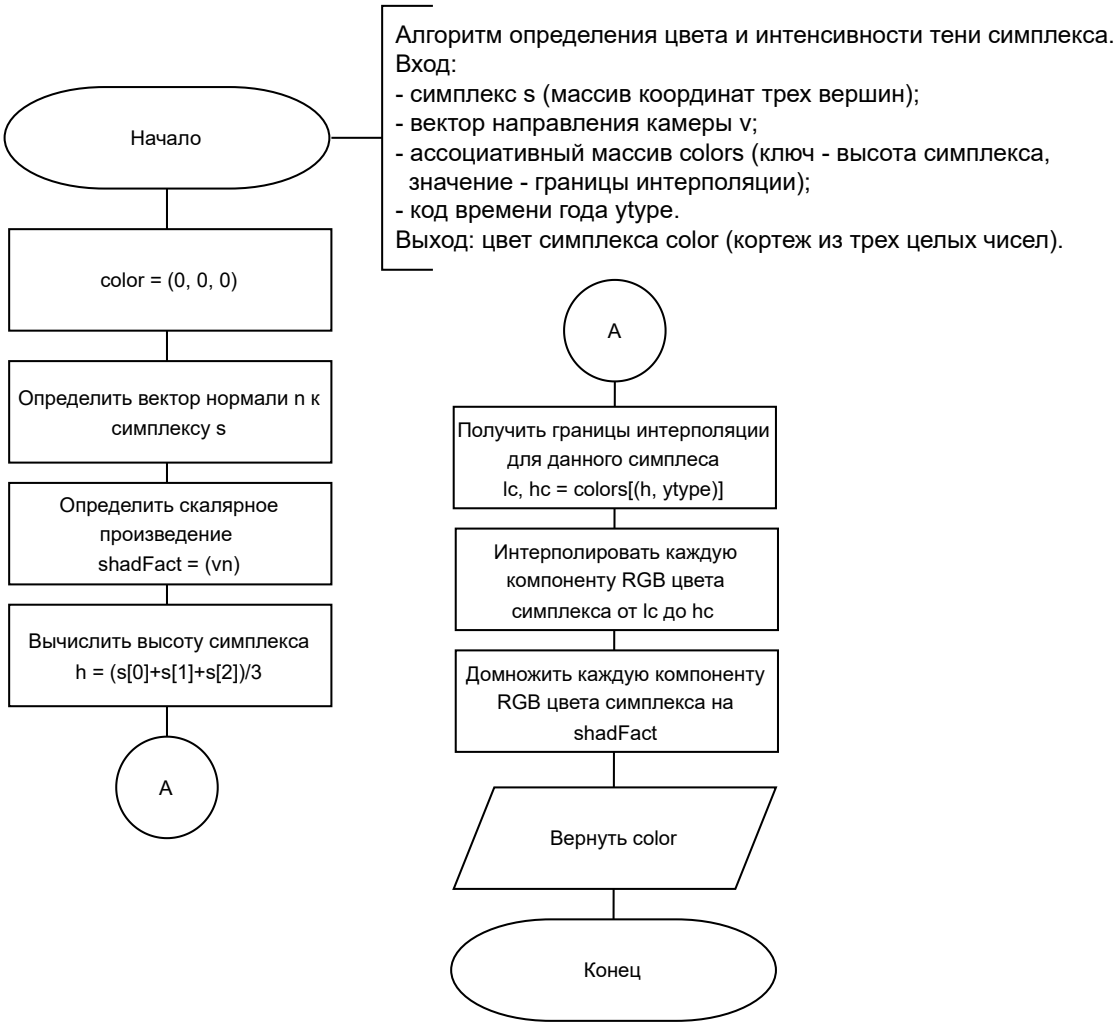

Рисунок 2.4 – Схема алгоритма определения цветовой характеристики симплекса

Для определения цвета симплекса используется факт естественного изменения окраса ландшафта с изменением его высоты. Высота ландшафта лежит в диапазоне $[h_{min}; h_{max}]$. Данный диапазон разбивается некоторым количеством высот h_i, каждой из которых соответствует некоторый цвет c_i. Для текущего симплекса s определяется его высота h_s, которая попадет в некоторый отрезок $[h_i; h_{i+1}]$. Цвет симплекса s определятся путем интерполяции от c_i до c_{i+1} соответственно положению h_i внутри $[h_i; h_{i+1}]$.

2.0.5 Алгоритм отрисовки кадра

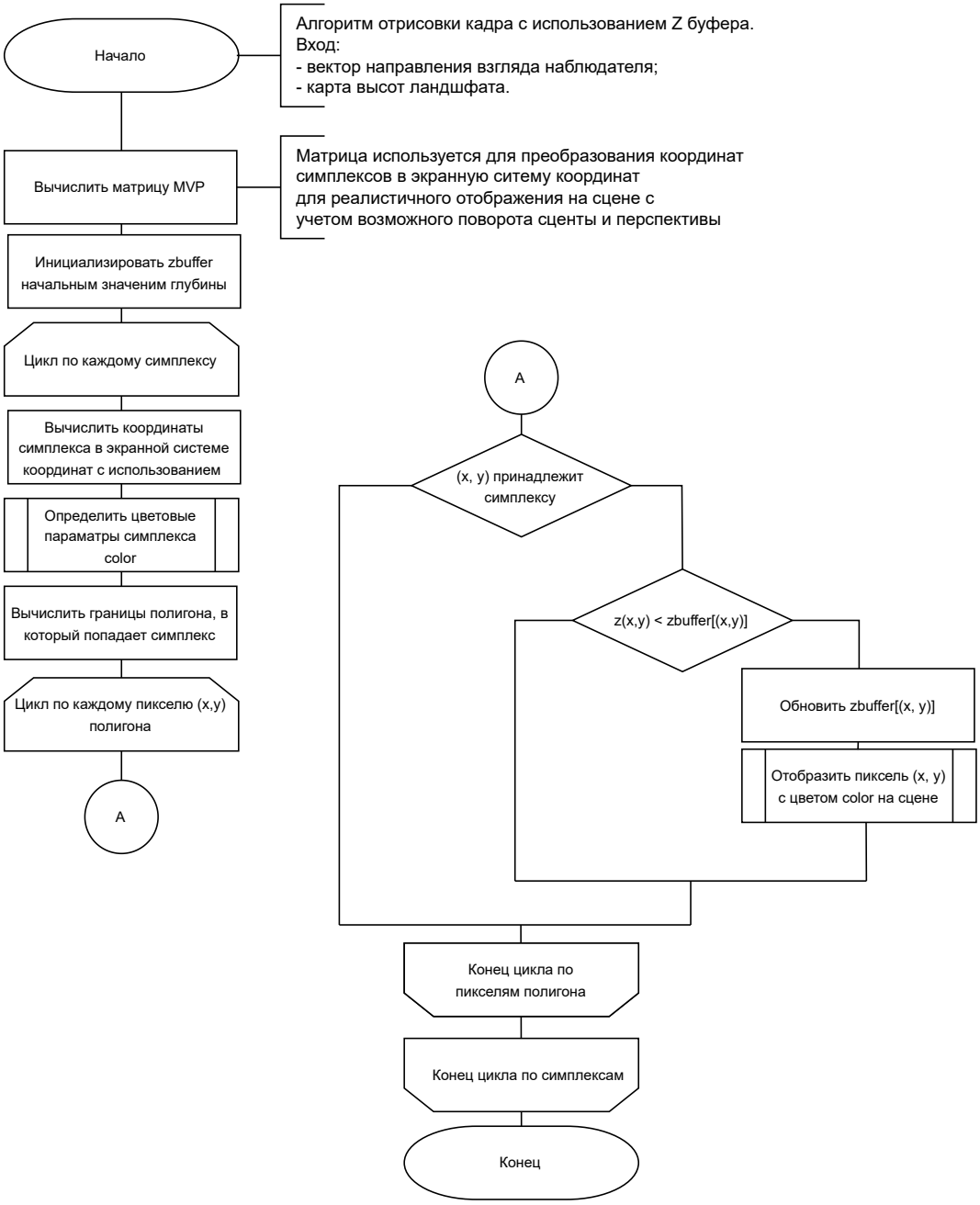

Рисунок 2.5 – Алгоритм отрисовки кадра с использованием Z-буфера

Вывод

Описаны алгоритмы:

- обобщенный алгоритм отрисовки кадра;
- алгоритм Simplex Noise генерации шума;
- алгоритм Simplex Noise с комбинацией октав;
- алгоритм определения цвета и тени симплекса;
- алгоритм отрисовки кадра с использованием Z-буфера.

Приведены схемы соответствующих алгоритмов.

3 Технологический раздел

Раздел содержит:

– описание средств реализации ПО;

– описание модели ПО и использованный структур данных;

– примеры работы программы;

– описание функционального тестирования.

3.1 Средства реализации ПО

Программное обеспечение реализовано с использованием языка C++. Данный язык предоставляет необходимый для решения поставленной задачи инструментарий: библиотеку для работы с графикой *GLFW* [9], библиотеку операций линейной алгебры *GLM* [10].

3.2 Описание модели ПО

При реализации была использована объектно-ориентированная модель ПО. В таблице 3.1 приведено описание реализованных классов.

Таблица 3.1 – Реализованные классы

Название класса	Ответственность класса
Application	Отображение окна, интерфейса, обработка сигналов
RenderEngine	Отрисовка ландшафта
Fjord	Модификация и храненеие ландшафта
HeightGenerator	Интерфейс генератора высот
OctaveGenerator	Генератор высот с помощью алгоритма Simplex Noise с генерацией октав
HeightGenerator_Creator	Интерфейс создателя генератора
OctaveGenerator_Creator	Создатель объекта OctaveGenerator

На рисунке 3.1 изображена *UML* диаграмма реализованных классов.

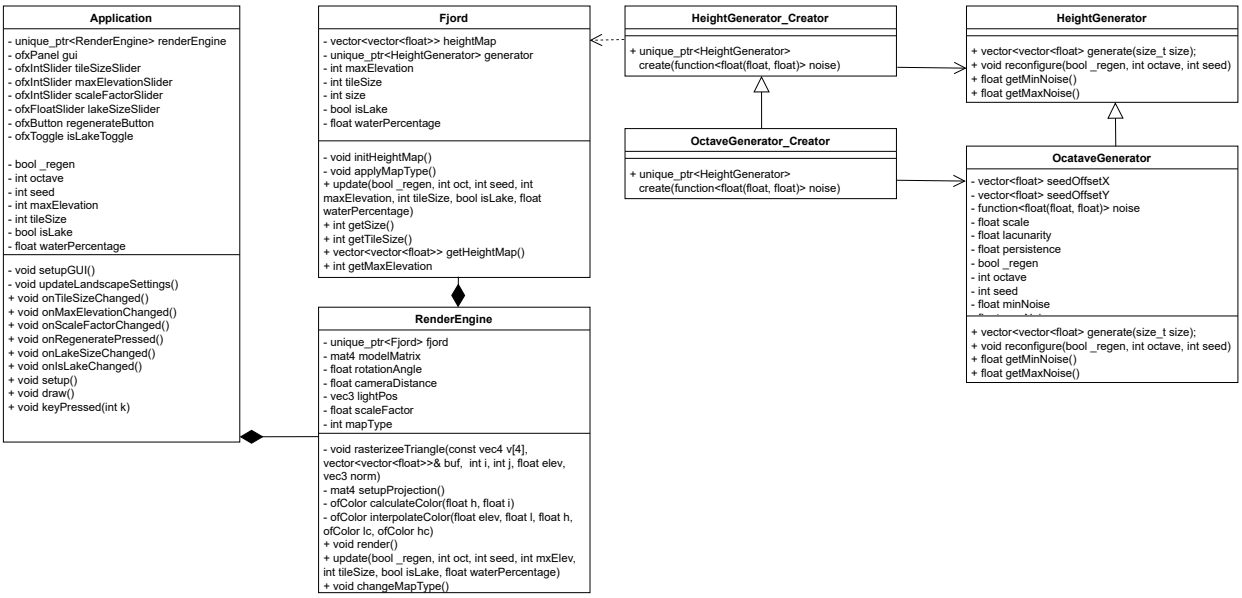

Рисунок 3.1 – *UML* диаграмма реализованных классов

3.3 Описание структур данных

Для хранения карты высот использовалась квадратная матрица с вещественными числами в качестве элементов.

Для поддержания соответствия между вершинами симплексов и векторами $\overrightarrow{g}$ использовался одномерный массив $perm$ на 256 элементов, заполненный случайными целыми числами от 0 до 255. Координаты вектора $\overrightarrow{g_{ij}}$ определяются в соответствии с формулой:

$$\overrightarrow{g_{ij}} = (perm[i\%255], perm[j\%255]).$$

Для хранения информации о раскраске ландшафта использовался ассоциативный массив. Ключ – код времени года (летние текстуры или зимние), значение – массив кортежей вида $[h_{min}; h_{max}; c_{min}; c_{max}]$.

3.4 Описание интерфейса

Взаимодействие пользователя с программой происходит посредством графического интерфейса и клавиш на клавиатуре:

1) стрелка «влево» – поворот ландшафта относительно его центра влево;

2) стрелка «вправо» – поворот ландшафта относительно его центра вправо;

3) стрелка «вверх» – увеличение ландшафта (приближение камеры к его центру);

4) стрелка «вниз» – уменьшение ландшафта (отдаление камеры от его центра);

Графический интерфейс позволяет задать количество октав, высотность, разреженность ландшафта после перехода в режим «озеро» и шаг регулярной сетки.

На рисунке 3.2 изображена панель графического интерфейса программы.

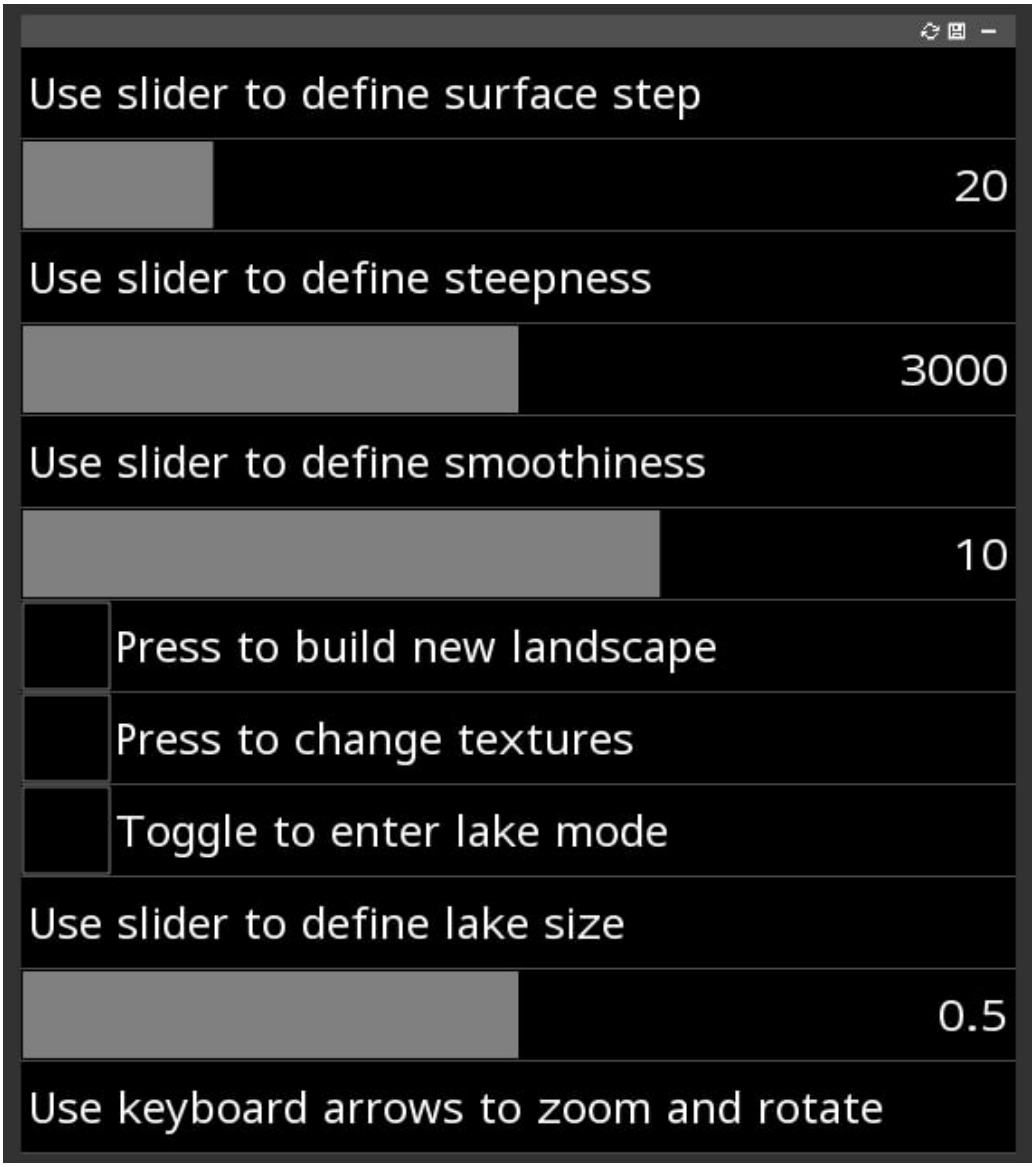

Рисунок 3.2 – Панель графического интерфейса программы

3.5 Пример работы программы

На рисунках 3.3 – 3.6 изображены примеры работы программы.

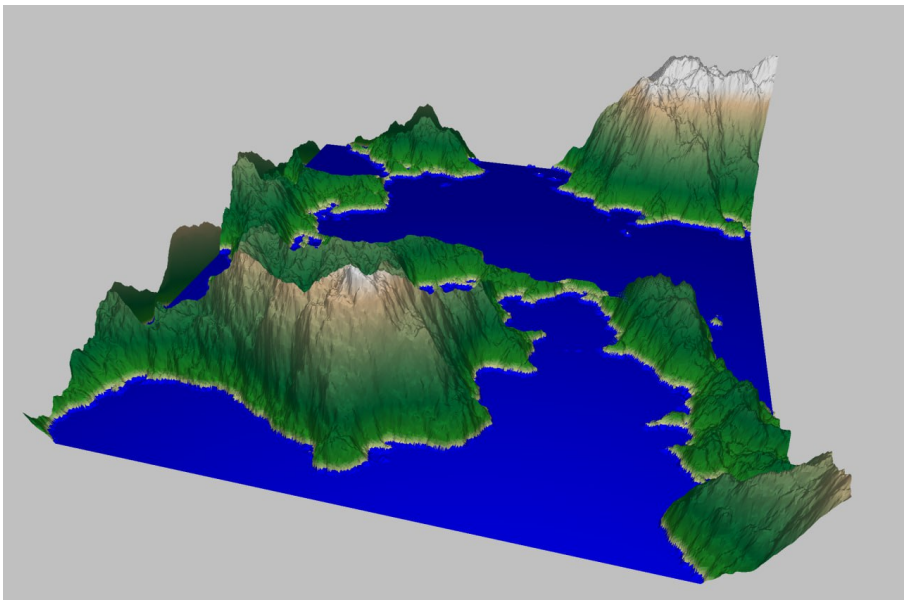

Рисунок 3.3 – Летний горный ландшафт, генерация в 10 октав и шаг сетки 10

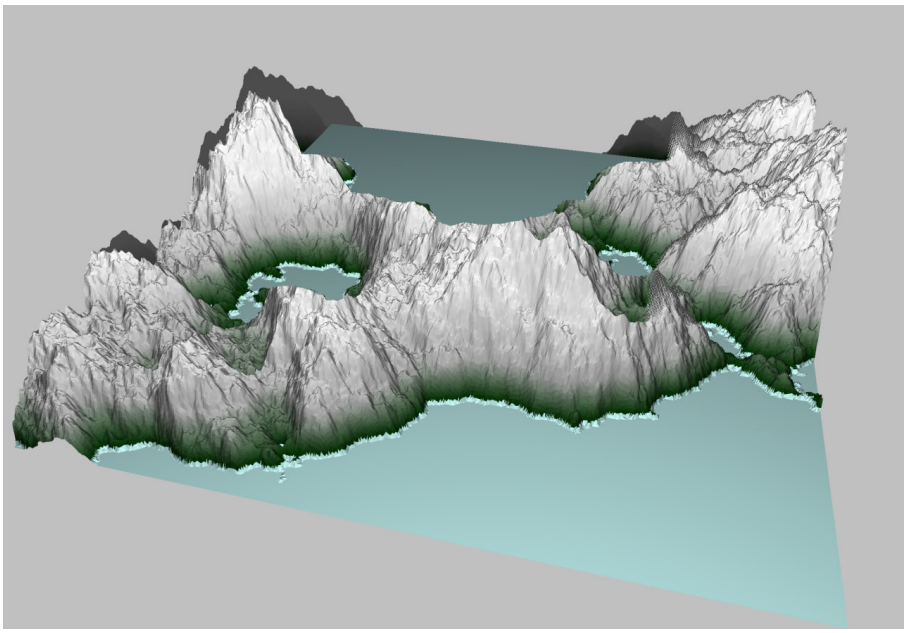

Рисунок 3.4 – Зимний горный ландшафт, генерация в 8 октав и шаг сетки 10

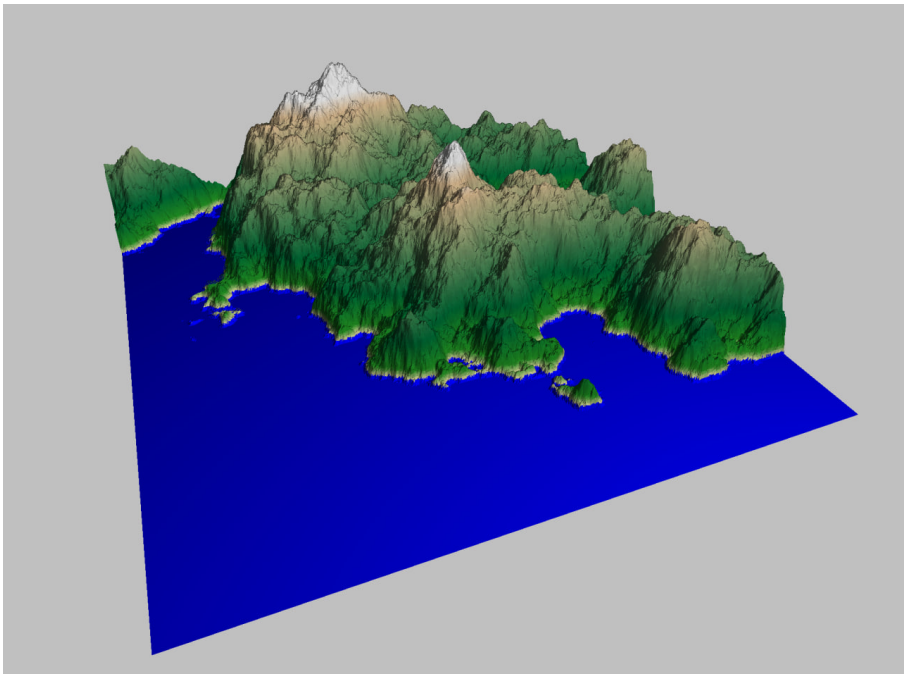

Рисунок 3.5 – Повернутый ландшафт, генерация в 12 октав и шаг сетки 10

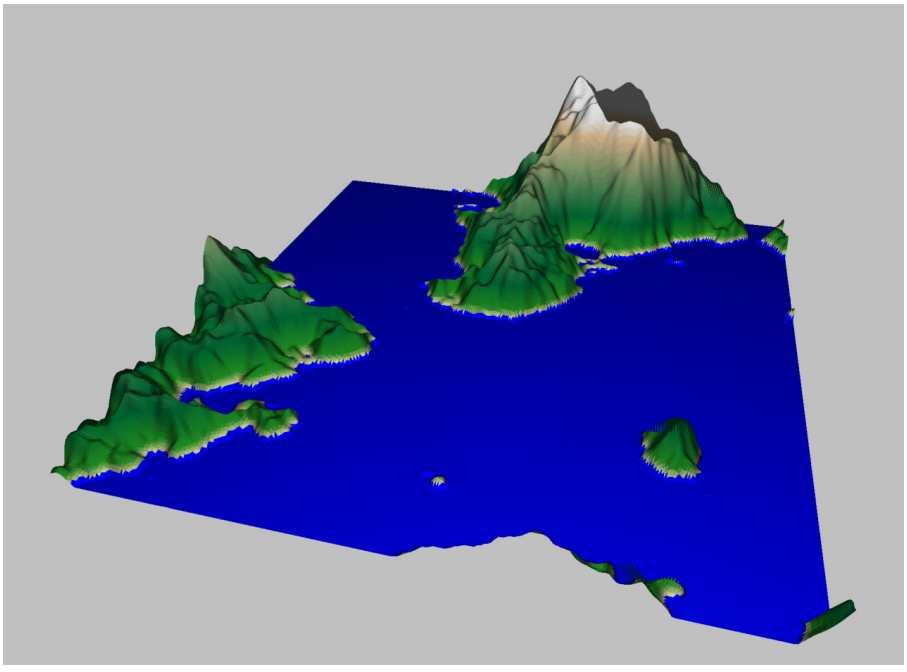

Рисунок 3.6 – Летний ландшафт, генерация в 5 октав и шаг сетки 50

3.6 Функциональное тестирование программы

В таблице 3.2 приведено описание проведенного функционального тестирования. Результат работы программы – кадр с изображением ландшафта. Тест считался пройденным, если 5 опрошенных человек соглашались с тем, что результат теста удовлетворяет ожидаемому.

Таблица 3.2 – Описание функционального тестирования

№	Описание теста	Ожидаемый результат	Тест пройден
1	Установка летних текстур	По кадру ясно, что время года – лето	+
2	Установка зимних текстур	По кадру ясно, что время года – зима	+
3	Шаг сетки 10 10 октав	Уступы гор детализированы	+
4	Шаг сетки 10 5 октав	Горы стали более гладкими в сравнении с предыдущим тестом	+
5	Шаг сетки 80 10 октав	Горы сглажены больше походят на мультяшные	+
6	Разреженность ландшафта 80	Горы редки, большая часть сцены – вода	+
7	Разреженность ландшафта 10	Большая часть ландшафта – горы	+
8	Высотность 0	Сцена плоская, только вода	+
9	Поворот влево	Сцена повернулась влево	+
10	Поворот вправо	Сцена повернулась вправо	+
11	Приближение сцены	Сцена стала ближе	+
12	Отдаление сцены	Сцена стала дальше	+

Вывод

Программное обеспечение реализовано и протестировано. В разделе описана выбранная модель ПО и реализованные классы. Описан пользовательский интерфейс, приведены примеры работы программы.

4 Исследовательский раздел

Раздел содержит описание проведенных исследований временных характеристик реализованного ПО. Здесь и далее под временем подразумевается именно процессорное время.

4.1 Условия проведения исследования

Замер времени выполнен на ноутбуке *HUAWEI MateBook 14 Core Ultra*. Его характеристики:

– процессор *Intel® Core™ Ultra 5 processor 125H*;

– объем оперативной памяти 16 ГБ [11];

– ОС *Windows Home 11*.

Для замера времени использовался функционал встроенной в C++ библиотеки *chrono*. Функция *std::chrono::steady_ clock::now()* возвращает текущее время в секундах. Для хранения времени работы функции использовался тип данных *std::chrono::duration<double>*.

Каждый замер выполнялся 20 раз. В качестве результирующего значения для данного замера выбиралось среднее арифметическое.

4.2 Исследование №1

В первом исследовании проводилось исследование зависимости времени генерации карты высот от количества октав. В таблице 4.1 приведены значения неизменных в ходе данного исследования параметров параметров.

Таблица 4.1 – Неизменные параметры первого исследования

Параметр	**Значение**
Линейный размер карты	10000
Шаг регулярной сетки	20

Результаты исследования представлены в таблице таблице 4.2.

Таблица 4.2 – Результаты замера зависимости процессорного времени генерации карты высот от количества октав

№	**Количество октав, шт.**	**Процессорное время работы, сек.**
1	1	0.027
2	2	0.070
3	3	0.093
4	4	0.130
5	5	0.150
6	6	0.195
7	7	0.237
8	8	0.282
9	9	0.340
10	10	0.368
11	11	0.427
12	12	0.453
13	13	0.521
14	14	0.549
15	15	0.609

С помощью метода линейной регрессии по полученным данным была получена аппроксимирующая линейная зависимость

$$f(x) = 0.038x.$$

Среднеквадратичное отклонение для данной зависимости рассчитано по формуле (4.1)

$$\text{RMSE} = \sqrt{\frac{1}{15}\sum_{i=1}^{15}(y_i - \hat{y}_i)^2}, \tag{4.1}$$

оно составило 0.022.

На рисунке 4.1 показаны полученные данные и аппроксимирующая прямая.

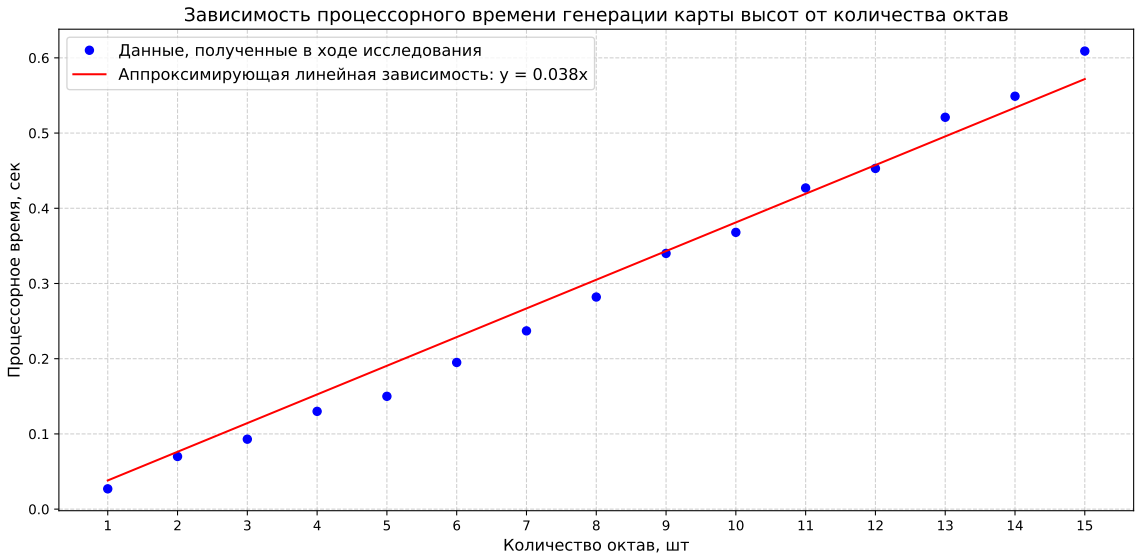

Рисунок 4.1 – Зависимость процессорного времени генерации карты высот от количества октав и аппроксимирующая ее линейная зависимость

На основании результатов исследования можно заключить, что процессорное время работы алгоритма генерации высот с использованием алгоритма *Simplex Noise* линейно зависит от количества октав генерации.

4.3 Исследование №2

Во втором исследовании проводилось исследование зависимости времени отрисовки кадра ландшафта в зависимости от его линейного размера. Карта имеет квадратную форму. Размер – это количество точек вдоль одной из сторон карты, в которых вычислено значение высоты.

Результаты исследования представлены в таблице 4.3.

Таблица 4.3 – Результаты замера процессорного времени отрисовки кадра с изображением ландшафта при различном линейном размере карты

№	**Линейный размер карты, шт**	**Процессорное время работы, сек.**
1	100	2.672
2	200	4.555
3	300	7.900
4	400	12.879
5	500	18.867
6	600	25.133
7	700	35.089
8	800	51.282
9	900	63.640
10	1000	74.341

С помощью метода полиномиальной регрессии по полученным данным была получен аппроксимирующий полином второй степени

$$f(x) = 8.337 \cdot 10^{-5} \cdot x^2 - 9.681 \cdot 10^{-3} \cdot x + 2.970$$

Аналогично предыдущему исследованию, среднеквадратичное отклонение для данной зависимости рассчитано по формуле (4.1), оно составило 1.552.

На рисунке 4.2 показаны полученные данные и аппроксимирующий полином.

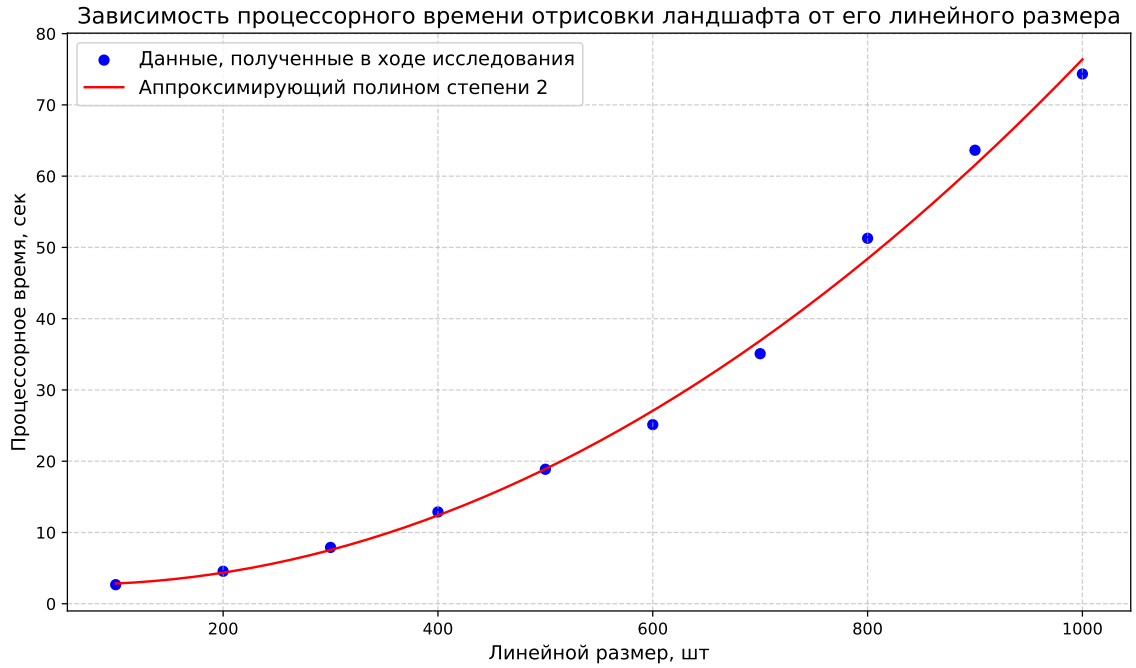

Рисунок 4.2 – Зависимость процессорного времени отрисовки кадра ландшафта от линейного размера карты

На основании результатов исследования можно заключить, что процессорное время процесса отрисовки кадра ландшафта квадратично зависит от линейного размера карты.

Вывод

В разделе описаны условия проведения и результаты исследований временных характеристик реализованного ПО.

В первом исследовании был проведен замер процессорного времени работы алгоритма генерации карты высот в зависимости от количества октав. На основе полученных данных сделан вывод о линейном характере данной зависимости.

Во втором исследовании был проведен замер процессорного времени алгоритма построения кадра ландшафта в зависимости от линейного размера карты. На основе полученных данных сделан вывод о квадратичном характере данной зависимости.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы решены следующие задачи:

– описаны существующие алгоритмы, применяемые при программной построении ландшафта, проведен сравнительный анализ алгоритмов, выбраны алгоритмы для последующей реализации;

– выбранные алгоритмы дополнены и формализованы;

– реализовано и протестировано программное обеспечения для построения реалистичного горного ландшафта;

– проведена серия исследований временных характеристик разработанного ПО.

Цель работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Shen Z.* Procedural Generation in Games: Focusing on Dungeons. — 2022.
2. *Paar P.* Landscape visualizations: Applications and requirements of 3D visualization software for environmental planning. — 2005.
3. Генерация трехмерных ландшфатов. [Электронный ресурс]. — Режим доступа: https://www.ixbt.com/video/3dterrains-generation.shtml.
4. Simplex Noise. Simplexes and radial kernels. [Электронный ресурс]. — Режим доступа: https://catlikecoding.com/unity/tutorials/pseudorandom-noise/simplex-noise/.
5. Terrain generation algorithms. [Электронный ресурс]. — Режим доступа: https://trepo.tuni.fi/bitstream/handle/10024/147549/SainioNiko.pdf.
6. Procedural generation. Perlin Noise. [Электронный ресурс]. — Режим доступа: https://www.cs.umd.edu/class/spring2018/cmsc425/Lects/lect13-2d-perlin.pdf.
7. *Д. Р.* Алгоритмические основы машинной графики: Пер. с англ. — СПб: БХВ-Петербург, 1989. — С. 512.
8. *С.В. П.* Компьютерная графика. — Москва: Издательство Московского государственного университета леса, 2008. — С. 39.
9. GLFW Documentation [Электронный ресурс]. — Режим доступа: https://www.glfw.org/docs/3.3/index.html (дата обращения: 20.10.2024).
10. GLM Documentation [Электронный ресурс]. — Режим доступа: https://openframeworks.cc/documentation/glm/ (дата обращения: 20.10.2024).
11. Huawei MateBook Core Ultra Specification [Электронный ресурс]. — Режим доступа: https://consumer.huawei.com/en/laptops/matebook-14-2024/specs/ (дата обращения: 20.10.2024).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе, содержащая 17 слайдов.