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 КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Программное обеспечение для генерации трёхмерного ландшафта»

Студент группы ИУ7-54Б _____ (Подпись, дата) Парфенов А. А. (Фамилия И.О.)

Научный руководитель _____ (Подпись, дата) Русакова З. Н. (Фамилия И.О.)

Москва, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка программного обеспечения для генерации реалистичного трёхмерного ландшафта. Для достижения данной цели необходимо решить следующие задачи:

1) провести анализ алгоритмов компьютерной графики для создания трёхмерных тел;
2) выбрать алгоритмы для решения поставленной задачи;
3) спроектировать алгоритм решения поставленной задачи;
4) выбрать средства реализации;
5) провести исследование зависимости времени генерации ландшафта от его размера.

1 Аналитическая часть

В данной части будут описаны алгоритмы.

1.1 Формализация объектов синтезируемой сцены

Объекты сцены:

1) ландшафт – поверхность задаваемая картой высот. Может быть задана как с помощью заранее предопределённых значений, так и методом случайной генерации;
2) источник света – точка в пространстве освещающая сцену в заданном направлении. Характеризуется цветом и интенсивностью;
3) камера – объект который задаёт направление взгляда наблюдателя, характеризуется вектором.

1.2 Методы описания поверхностей

1.2.1 Аналитическая модель

Данная модель подразумевает формальное описание математическими формулами. Обычно такая модель может быть описана функцией от двух переменных (1.1).

$$z = f(x, y) \tag{1.1}$$

Или может быть описана формулой (1.2).

$$F(x, y, z) = 0 \tag{1.2}$$

Зачастую для описания таких поверхностей используются кусочно заданные функции, например сплайны. Сплайн – это кусочно заданная функция пригодная для аппроксимации отдельных фрагментов поверхности.

Преимущества данного подхода:

— не большой объём занимаемой памяти;

— простой расчёт положения каждой точки модели.

Недостатки:

— расчёт по аналитическим формулам может занимать много времени;

— не для всякой поверхности удобно использовать данный подход или вовсе возможно;

1.2.2 Воксельная модель

Воксельная модель [1] является трёхмерным растром. Подобно пикселям которые располагаются на плоскости, воксели располагаются в некотором объёме.

Преимущества данной модели:

— простота описания сложных объектов;

— простота преобразований;

— простота отображения на экран.

Недостатки:

— большой объём требуемой памяти;

— проблемы с точностью модели;

— небольшая скорость обработки.

1.2.3 Неравномерная сетка

Равномерной сеткой назовём множество заданных точек вида: $\{(x_0, y_0, z_0) \dots (x_n, y_n, z_n)\}$, которые принадлежат поверхности.

Преимущества данной модели:

— можно использовать только важные опорные точки, что ведёт к уменьшению объёма требуемой памяти.

Недостатки:

— могут возникать проблемы при масштабировании изображения.

1.2.4 Равномерная сетка

Способ задания модели с помощью равномерной сетки отображён на рисунке 1.1.

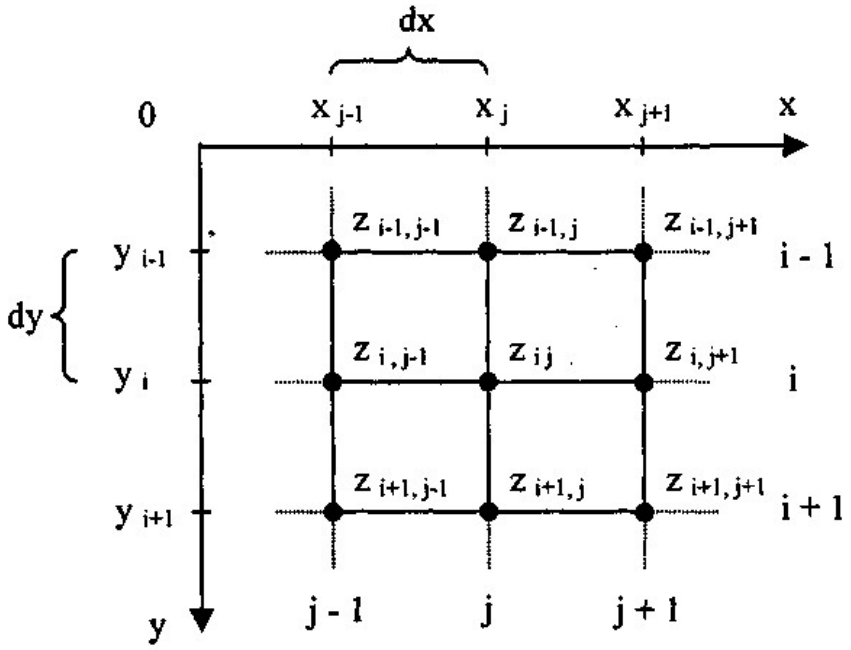

Рисунок 1.1 — Представление высот с помощью равномерной сетки

Каждый узел с индексами i, j равномерной сетки содержит высоту Z_{ij}. По сути мы получаем однозначную формулу (1.3) для получения высоты Z.

$$Z = f(i, j) \tag{1.3}$$

Преимущества данной модели:

— простота представления сложных моделей;

— быстрота обработки модели;

— возможность получить значения любой точки с помощью интерполяции.

Недостатки:

— равномерная сетка может быть представлена как однозначная формула (1.3), это обозначает невозможность задания поверхностей которые соответствуют неоднозначным функциям;

— невозможность задания вертикальных граней.

Выбор метода описания поверхности

Для решения поставленной задачи был выбран метод представления ландшафта в виде равномерной сетки. Ограничения этого метода, связанные с однозначностью функции z = f(x, y) и невозможностью моделирования

вертикальных граней, являются незначительными и вполне приемлемыми. Метод регулярной сетки обеспечивает простоту описания поверхности и часто используется для моделирования рельефа земной поверхности.

1.3 Алгоритмы построения карты высот

1.3.1 Задание высот случайными числами

Данный метод задаёт каждую точку карты высот случайным числом.

Преимущества данного алгоритма:

— простая реализация.

Недостатки:

— невозможность построения реалистичного ландшафта;

— слабый контроль.

1.3.2 Холмовой алгоритм

В данном методе все точки карты высот изначально находятся на одном уровне. Затем случайным образом вносятся холмы с случайно заданным радиусом.

Преимущества данного алгоритма:

— возможность получения реалистичного ландшафта.

Недостатки:

— нереалистичность ландшафта при малом количестве итераций;

— необходимость различных оптимизаций алгоритма для достижения реалистичного ландшафта.

1.3.3 Построение карты высот с помощью диаграммы Вороного

Диаграмма вороного [2] – это разбиение плоскости на n областей, в виде выпуклых многоугольников для n точек. Каждая область содержит в себе только одну точку. Имеется четыре алгоритма постройки диаграммы и самым быстрым и известным является алгоритм Форчуна. Пример диаграммы изображён на рисунке 1.2.

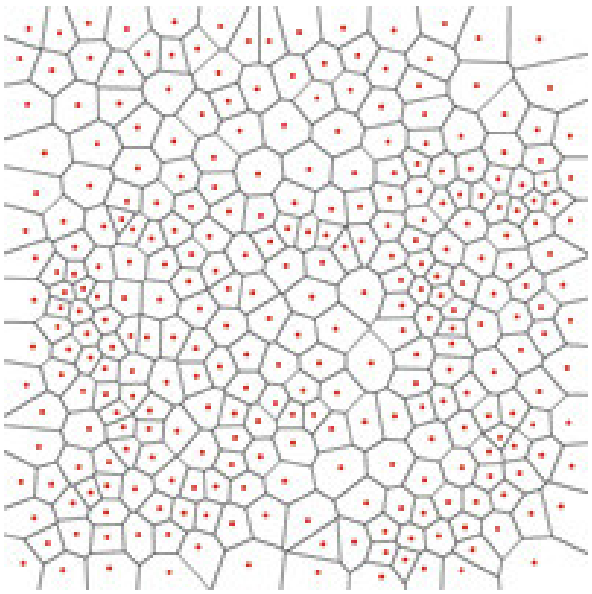

Рисунок 1.2 — Диаграмма Вороного построенная с помощью алгоритма Форчуна

1.3.4 Алгоритм diamond-square

Данный алгоритм [3] работает с равномерной сеткой. Он является расширением алгоритма «midpoint displacement» который работает с отрезком. Суть алгоритма заключается в внесении случайного смещения по центру отрезка, затем отрезок рабивается на две равные части, после чего алгоритм рекурсивно повторяется для этих половинок. Причём на каждой итерации алгоритма максимальное значение смещения по модулю уменьшается. Пример работы этого алгоритма изображён на рисунке 1.3

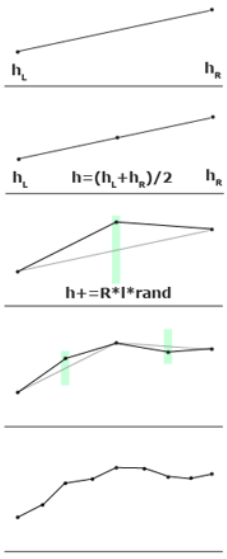

Рисунок 1.3 — Пример работы алгоритма midpoint displacement

Этот алгоритм можно обобщить и для плоскости, работать он будет следующим образом: плоскость разбивается на четыре равных квадрата, значение в углах квадратов получаются путём интерполяции и добавлением случайного смещения.

Алгоритм «diamond-square» отличается от двумерного «midpoint displacement», тем что он состоит из двух этапов:

1) «square» – определяет точку посередине квадрата путём интерполяции высот угловых точек и добавления смещения;
2) «diamond» – определяет высоту точек лежащих на серединах сторон квадратов, путём интерполяции значений концов стороны и двух середин смежных квадратов.

Пример работы данного алгоритма изображён на рисунке 1.4.

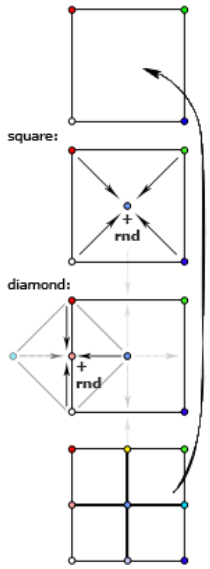

Рисунок 1.4 — Пример работы алгоритма diamond-square

Преимущества данного алгоритма:

— получение крайне реалистичного ландшафта;
— простота реализации.

Недостатки:

— шум при больших картах.

Выбор алгоритма построения карты высот

Поскольку для представления ландшафта используется равномерная сетка высот, предпочтительным алгоритмом является алгоритм «diamond-

square». Этот алгоритм также позволяет гибко настраивать параметры, что дает возможность получать разнообразные и одновременно очень реалистичные ландшафты.

1.4 Алгоритмы удаления невидимых линий и поверхностей

1.4.1 Алгоритм Варнока

Алгоритм работает в пространстве изображения. Производится разбиение области на «окна». Для каждого из них решается вопрос о том, пусто ли «окно», либо его содержимое «достаточно просто» для визуализации. Если это не так, то окно рекурсивно разбивается на фрагменты до тех пор, пока содержимое не станет «достаточно простым» для визуализации. В случае достижения минимального предела размера полученного окна необходимо вычислить глубину каждого элемента его содержимого и визуализировать тот из них, который расположен ближе остальных к точке наблюдения.

Преимущества данного алгоритма:

— при увеличении размеров однородных областей изображения повышается скорость работы.

Недостатки:

— в сложных сценах количество разбиений может стать чрезмерно большим.

1.4.2 Алгоритм Робертса

Данный алгоритм работает в объектном пространстве. Сначала для каждого объекта удаляются грани которые скрываются самим объектом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения частей, скрывающихся этими телами.

Преимущества данного алгоритма:

— высокая точность;

— низкая сложность математических операций.

Недостатки:

— значительное замедление при увеличении объектов сцены.

1.4.3 Алгоритм трассировки лучей

В этом алгоритме для каждого пикселя изображения выпускается луч, который пересекает грани объектов. Для визуализации выбирается ближайшее пересечение.

Преимущества данного алгоритма:

— реалистичность отображаемых гладких объектов без их аппроксимации примитивами;

— линейная зависимость вычислительной сложности от количества элементов на сцене.

Недостатки:

— требуется большое количество вычислений, в особенности при высоком разрешении создаваемого изображения.

1.4.4 Алгоритм, использующий Z-буфер

Алгоритм [4] функционирует в пространстве изображения и использует буфер кадра для хранения интенсивности каждого пикселя. Он также сохраняет глубину каждого пикселя в Z-буфере. При добавлении нового пикселя в буфер кадра его значение глубины сравнивается с глубиной уже сохраненного пикселя в Z-буфере. Если глубина нового пикселя меньше, он заменяет ранее сохраненный пиксель, и Z-буфер обновляется. Если же глубина нового пикселя больше, никаких действий не требуется.

Преимущества данного алгоритма:

— простота релизации;

— отсутствие необходимости сортировки элементов сцены.

Недостатки:

— увеличение затрат памяти на хранение дополнительной информации о глубине отображемых пикселов.

Выбор алгоритма удаления невидимых линий и поверхностей

Наиболее подходящим для решения поставленной задачи выбран алгоритм, использующий Z-буфер, поскольку он прост в реализации, обеспечивает достаточно реалистичное изображение для данной задачи и не

требует значительных вычислительных ресурсов, в отличие от алгоритмов, которые могут обеспечить более высокую степень реалистичности изображения.

1.5 Модели освещения

Модели освещения можно поделить на локальные и глобальные. Глобальные модели учитывают преломление и отражение света от объектов, локальные учитывают исключительно освещение самих источников.

Освещение объектов складывается из трёх составляющих:

1) фоновая – константа окружающего освещения, которая всегда будет придавать объекту некоторый оттенок;
2) диффузная – имитирует воздействие на объект направленного источника света;
3) зеркальная — имитирует блик света, который появляется на блестящих объектах.

1.5.1 Модель освещения Ламберта

Эта модель позволяет реализовать диффузное освещение объектов, при котором свет, попадая на поверхность, рассеивается равномерно во всех направлениях. При расчете такого освещения учитываются только ориентация поверхности и направление на источник света. Интенсивность отраженного света не зависит от положения наблюдателя. Цвет, который мы видим на матовой поверхности, определяется комбинацией ее собственного цвета и цвета света, исходящего от источника. Интенсивность отражения пропорциональна косинусу угла между внешней нормалью к поверхности и направлением на источник света.

1.5.2 Модель освещения Фонга

Эта модель позволяет сочетать диффузную и зеркальную составляющие, что дает возможность появления бликов на материале. Падающий и отраженный лучи находятся в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части. Интенсивность отраженной освещенности в данной точке зависит от того, насколько близки направления вектора, направленного на наблюдателя, и

отраженного луча.

1.5.3 Выбор модели освещения

Объекты создаваемой сцены не требуют зеркального отражения и рассматриваются как матовые. В связи с этим целесообразно использовать локальную модель освещения без зеркальной составляющей. Создание бликов отраженного света не является необходимым для данной задачи. Учитывая эти факторы, оптимальной моделью освещения будет модель Ламберта.

1.6 Алгоритмы закраски

1.6.1 Простой алгоритм закраски

При применении данного алгоритма окраска всей поверхности имеет одинаковый уровень интенсивности, и для вычислений используется закон Ламберта. Алгоритм обеспечивает высокое качество изображений только в том случае, если источник света и наблюдатель находятся на значительном расстоянии от объекта, а каждая грань тела не является аппроксимирующей поверхностью.

Преимущества данного алгоритма:

— простота реализации;

— быстродействие

Недостатки:

— видимые переходы между гранями у аппроксимированных гладких объектов;

— низкая правдоподобность изображения зеркальных бликов.

1.6.2 Закраска Гуро

Алгоритм закраски Гуро основан на интерполяции интенсивности и позволяет получать качественные изображения гладких поверхностей. Суть метода заключается в том, что освещение не усредняется для всей грани, а линейно интерполируется между вершинами. В этой модели нормали задаются в вершинах.

Преимущества данного алгоритма:

— достаточно качественное отображение гладких поверхностей;

Недостатки:

— появление полос Маха;

— низкая правдоподобность изображения зеркальных бликов.

1.6.3 Закраска Фонга

Алгоритм закраски Фонга основан на интерполяции вектора нормали. Цвет, в свою очередь, рассчитывается для каждого пикселя в отдельности.

Преимущества данного алгоритма:

— высокая правдоподобность изображения зеркальных бликов;

— достаточно качественное отображение гладких поверхностей.

Недостатки:

— требуется большое количество вычислительных ресурсов.

Выбор алгоритма закраски

Для демонстрации алгоритма генерации ландшафтных поверхностей не обязательно добиваться высокой реалистичности цветового отображения. В данном случае предпочтительнее использовать максимально эффективный алгоритм, а именно – плоскую закраску, поскольку значительное время обработки будет затрачено на трехмерные преобразования сцены, требующие длительных тригонометрических вычислений.

Вывод

В этом разделе были рассмотрены методы создания реалистичных изображений. Описаны характеристики объектов сцены и способы их представления, а также проанализированы различные алгоритмы рендеринга и модели освещения. На основе проведенного анализа был сделан выбор в пользу равномерной сетки, модели освещения Ламберта и алгоритма использующего Z буфер. Эти методы являются наиболее подходящими для разработки алгоритма, способного реалистично отображать реалистичный ландшафт.

2 Конструкторская часть

2.1 Требования к программному обеспечению

Разрабатываемое программное обеспечение должно предоставлять следующие функциональные возможности:

Генерация карты высот

— Задание размера матрицы высот;
— задание значения шума.

Ландшафт

— Задание высоты ландшафта;
— задание уровня воды.

Источник света

— Задание положения источника света.

Преобразования ландшафта

— Поворот ландшафта;
— перемещение ландшафта;
— масштабирование ландшафта.

2.2 Общий алгоритм поставленной задачи

1) Сгенерировать или загрузить карту высот ландшафта;
2) рассчитать экранные координаты;
3) рассчитать вектор нормали для каждого полигона;
4) рассчитать интенсивность для каждого полигона на основе вектора нормали;
5) рассчитать цвет каждого полигона на основе карты высот;
6) отобразить сцену на экран.

2.3 Описания алгоритмов

2.3.1 Алгоритм триангуляции равномерной сетки

Пусть есть карта высот заданная равномерное сеткой m размером $n \times n$.

1) получить элементы карты высот $m[i][j], m[i][j+1], m[i+1][j], m[i+1][j+1]$, где $i, j \in [0, n-1]$;
2) сохранить треугольник составленный из вершин $m[i][j], m[i+1][j], m[i+1][j+1]$;
3) сохранить треугольник составленный из вершин $m[i][j], m[i][j+1], m[i+1][j+1]$.

2.3.2 Алгоритм diamond–square

Пусть есть карта высот заданная равномерное сеткой m размером $n \times n$.

Пусть $stepwide = n$

Шаг «square»:

1) Вычислить расстояние до соседей: $neighbourdist = stepwide/2$;
2) вычислить среднее значение высот по ячейкам индексы которых вычисляются как сумма индексов текущей ячейки и матричное произведение $(neighbourdist) \times (offset)$, где $offset \in \{(-1,-1),(1,1),(1,-1),(-1,1)\}$;
3) прибавить к этому значению случайный шум;
4) записать полученное значение в текущую ячейку.

Шаг «diamond»:

1) Вычислить расстояние до соседей: $neighbourdist = stepwide/2$;
2) вычислить среднее значение высот по ячейкам индексы которых вычисляются как сумма индексов текущей ячейки и матричное произведение $(neighbourdist) \times (offset)$, где $offset \in \{(0,-1),(0,1),(1,0),(-1,0)\}$;
3) прибавить к этому значению случайный шум;
4) записать полученное значение в текущую ячейку.

Основной алгоритм:

1) Шаг «square»;
2) шаг «diamond»;

3) уменьшить значение шума;

4) уменьшить значение $stepwide$ вдвое.

5) повторять эти действия пока ширина квадрата больше единицы.

2.3.3 Улучшение алгоритма diamond–square

Хотя алгоритм «diamond–square» генерирует реалистичные ландшафты, у них есть излишняя «шероховатость». Для того чтобы побороться с данным явлением был разработан следующий алгоритм сглаживания:

1) получить ширину и высоту карты высот;

2) завести временную карту высот размером со сглаживаемой;

3) заполнить каждый узел временной карты высот средним значением соответствующего узла из сглаживаемой карты высот и всех смежных с ним узлов из той же сглаживаемой карты;

4) скопировать в сглаживаемую карту временную.

2.3.4 Получение экранных координат по карте высот

Данные получаемые после алгоритма «diamond–square» нуждаются в преобразовании в экранные координаты. Такое преобразование можно осуществить с помощью следующего алгоритма:

1) получить карту высот;

2) создать матрицу трёхмерных точек такой же размерности как карта высот;

3) получить значения $\Delta x, \Delta y$, как частное размерностей экрана и размерностей карты высот;

4) получить минимальное и максимальное значения карты высот;

5) для каждой трёхмерной точки рассчитать её значение как $(i \times \Delta x, j \times \Delta y, ((H_{max} - H_{min}) \times H_{screen})$, где i – индекс соответствующего узла по строке, j – индекс соответствующего узла по столбцу, H_{screen} – заранее предопределённая величина для матрицы трёхмерных точек.

2.3.5 Алгоритм использующий Z буфер

1) Заполнить буфер кадра значением фона;

2) заполнить Z буфер минимальным значением;

3) для каждого пикселя растровой развертки объектов сцены сравнить глубину Z(x,y) с Z буфера в той же точке, если Z(x,y) > Z буфера (x,y), то заменить Z буфера (x,y) на z(x,y) и заменить цвет буфера кадра (x,y) на цвет текущего пикселя.

2.3.6 Получение цвета для полигона

Важной частью реалистичного ландшафта, является палитра цветов, используемая для его отображения. В качестве палитры я взял палитру типичную для топографических карт. Сама палитра представляет из себя зависимость цвета от высоты. Высоты распределены от 0 до 1.

Алгоритм получения цвета полигона от по его высоте:

1) вычислить высоту полигона как среднее значение из высот его вершин;
2) определить по таблице с палитрой между какими табличными высотами находится высота полигона;
3) совершить линейную интерполяцию между этими табличными значениями по цвету, результатом будет значение цвета для данного полигона.

2.3.7 Модель освещения Ламберта

Для каждого пикселя растровой развёрстки посчитать интенсивность по формуле (2.1).

$$I = I_0 \times K_d \times \cos\theta \tag{2.1}$$

Где I_0 – начальная интенсивность света, K_d – коэффициент диффузного отражения, θ – угол между вектором нормали поверхности и вектором образованным центром поверхности и источником света.

2.3.8 Улучшения освещения

Учитывая алгоритм работы с моделью Ламберта очевидно, что если посмотреть на карту с обратной стороны, то она будет окрашена в те же цвета, что и лицевая, чтобы исправить это, был добавлен «корректирующий вектор» и вектор направления камеры.

Корректирующий вектор

Изначально $V_{corr} = (0, 0, 1)$, однако при каждом аффинном преобразовании (исключая перенос), аналогичное преобразование применяется и к корректирующему вектору, действует правило, что центр масштабирования и поворота для данного вектора всегда находится в точке $C = (0, 0, 0)$.

Введения данного вектора требует внесения следующего изменения в алгоритм расчёта нормали для полигона:

1) получить нормаль для очередного полигона;
2) рассчитать скалярное произведения вектора нормали полигона и корректирующего вектора;
3) в случае отрицательности полученного произведения, инвертировать координаты вектора.

Камера

Для того чтобы правильно отобразить обратную стороны ландшафта требуется учитывать направление взгляда наблюдателя. Направления взгляда наблюдателя было введено через абстрактную камеру. У камеры есть вектор, отвечающий за то куда она направленна. Для поставленной задачи это направление константно и равно: $V_{cam} = (0, 0, -1)$.

С учетом этих улучшений алгоритм получения интенсивности полигона был модифицирован следующим образом:

1) получить вектор нормали для данного полигона;
2) высчитать скалярное произведение для вектора нормали и инвертированного вектора направления камеры;
3) если результат полученного произведения отрицателен присвоить данному полигону интенсивность равную 0, т. е. лицевая сторона данного полигона находится вне поля зрения наблюдателя.

2.3.9 Алгоритм движения источника света

Чтобы промоделировать движение источника вокруг заданной оси, нужно изменять его положение раз в какое-то время, в случае поставленной задачи изменение происходит раз в одну секунду. Для вычислений новых параметров ландшафта необходимо завести отдельный поток для достижения удобства пользования программным обеспечением.

2.3.10 Сохранение истории операций над ландшафтом

При изменении ландшафта, при вводе параметров могут возникнуть ошибки на стороне пользователя, для того чтобы удобно их править была введена история операций над ландшафтом.

Сами операции сохраняются как данные необходимые для совершения этой операции, а так же сохраняется позиция источника света которая была установлена до совершения операции.

Для каждой сохранённой операции должен быть определён метод *do* и *undo*. Из названия ясно, что они позволяют совершить сохранённое преобразование над ландшафтом и обратное преобразование от сохранённого соответственно.

2.3.11 Сохранение и загрузка ландшафта

Ландшафт сохраняется в виде текстового файла в который записывается размер карты высот и сама карта высот. При загрузке карты значения считываются из файл в объект ландшафта.

2.4 Представление данных в программном обеспечении

Таблица 2.1 — Представление данных в программе

Данные	**Представления**
Карта высот	Матрица вещественных значений
Точка в пространстве	Трёхмерный вектор
Источник света	Точка в пространстве
Камера	Трёхмерный вектор
История операций	Вектор указателей на класс операций

Вывод

В конструкторской части работы были представлены требования к программе, описания алгоритмов, выбранные типы и структуры данных.

3 Технологическая часть

3.1 Средства реализации программы

В качестве языка программирования для реализации курсовой работы был выбран язык C++ [7] по следующим причинам:

— В стандартной библиотеке языка присутствует поддержка всех структур данных, выбранных по результатам проектирования;

— высокая производительность;

— наличие фреймворка Qt [8]

— большое количество актуальной документации.

3.2 Реализация алгоритмов

На листингах 3.1, 3.2 представлен код ключевых классов программного обеспечения.

Листинг 3.1 — Класс ландшафта

```
class Terrain
{
  private:
  Matrix<double> height_mtr;
  Matrix<QVector3D> screen_mtr;

  vector<QVector3D> normal_plains;
  vector<double> intensity;

  double max_height;
  double min_height;

  double water_level = 0;

  QVector3D center_;

  QVector3D corrector_vector = {0, 0, 1};


  public:
  explicit Terrain(): Terrain(0)
```

```
22    {
23    }
24
25    explicit Terrain(size_t n): max_height(INT_MIN), min_height(
          INT_MAX)
26    {
27      height_mtr.resize(n);
28      for (auto &v: height_mtr)
29      v.resize(n);
30
31      screen_mtr.resize(n);
32      for (auto &v: screen_mtr)
33      v.resize(n);
34    }
35
36    Matrix<QVector3D> &get_screen_matrix()
37    {
38      return screen_mtr;
39    }
40
41    const Matrix<QVector3D> &get_screen_matrix() const
42    {
43      return screen_mtr;
44    }
45
46    const Matrix<double> &get_height_matrix() const
47    {
48      return height_mtr;
49    }
50
51    Matrix<double> &get_height_matrix()
52    {
53      return height_mtr;
54    }
55
56    vector<QVector3D> &get_normal_matrix()
57    {
58      return normal_plains;
59    }
60
```

```
61    vector<double> &get_intensity_matrix()
62    {
63      return intensity;
64    }
65
66    const vector<double> &get_intensity_matrix() const
67    {
68      return intensity;
69    }
70
71    const vector<double> &operator[](size_t index) const
72    {
73      return height_mtr[index];
74    }
75
76    vector<double> &operator[](size_t index)
77    {
78      return height_mtr[index];
79    }
80
81    size_t size() const
82    {
83      return height_mtr.size();
84    }
85
86    double get_max_h() const
87    {
88      return max_height;
89    }
90
91    void set_max_h(double h)
92    {
93      max_height = h;
94    }
95
96    double get_min_h() const
97    {
98      return min_height;
99    }
100
```

```
101   void set_min_h(double h)
102   {
103     min_height = h;
104   }
105
106   void set_center(const QVector3D& vector_)
107   {
108     center_ = vector_;
109   }
110
111   QVector3D& get_center()
112   {
113     return center_;
114   }
115
116   void clear();
117
118   void intensity_clear()
119   {
120     intensity.clear();
121     intensity.resize(0);
122   }
123
124   QVector3D& get_v_corrector()
125   {
126     return corrector_vector;
127   }
128
129   double get_water_level() const
130   {
131     return water_level;
132   }
133
134   void set_water_level(double water_level)
135   {
136     this->water_level = water_level;
137   }
138
139 };
```

Листинг 3.2 — Реализация алгоритма diamond-square

```
static random_device rd;
static mt19937 random_generator;

void Diamond_Square::init_rand_dev()
{
  random_generator.seed(rd());
}


double Diamond_Square::random(double noise)
{
  uniform_real_distribution<double> distribution{-noise, noise};
  return distribution(random_generator);
}


void Diamond_Square::diamond_step(Terrain &terrain, int step_wide
    , double noise)
{
  int neighbour_dist = step_wide / 2;
  vector<pair<int, int> > diamond_offset = {{-1, -1}, {-1, 1},
     {1, 1}, {1, -1}};

  for (size_t row = neighbour_dist; row < terrain.size(); row +=
     step_wide)
  {
    for (size_t col = neighbour_dist; col < terrain.size(); col
       += step_wide)
    {
      terrain[row][col] = strategy->avg(terrain, row, col,
         neighbour_dist, diamond_offset) + random(noise);
      terrain.set_max_h(max(terrain.get_max_h(), terrain[row][col
         ]));
      terrain.set_min_h(min(terrain.get_min_h(), terrain[row][col
         ]));
    }
  }
}

```

```
33 void Diamond_Square::diamond_step_row(Terrain &terrain, int
       step_wide, double noise)
34 {
35   int neighbour_dist = step_wide / 2;
36   vector<pair<int, int> > square_offset = {{-1, 0}, {0, -1}, {1,
       0}, {0, 1}};
37
38   for (size_t row = neighbour_dist; row < terrain.size(); row +=
       step_wide)
39   {
40     for (size_t col = 0; col < terrain.size(); col += step_wide)
41     {
42       terrain[row][col] = strategy->avg(terrain, row, col,
           neighbour_dist, square_offset) + random(noise);
43       terrain.set_max_h(max(terrain.get_max_h(), terrain[row][col
           ]));
44       terrain.set_min_h(min(terrain.get_min_h(), terrain[row][col
           ]));
45     }
46   }
47 }
48
49 void Diamond_Square::diamond_step_col(Terrain &terrain, int
       step_wide, double noise)
50 {
51   int neighbour_dist = step_wide / 2;
52   vector<pair<int, int> > square_offset = {{-1, 0}, {0, -1}, {1,
       0}, {0, 1}};
53
54   for (size_t row = 0; row < terrain.size(); row += step_wide)
55   {
56     for (size_t col = neighbour_dist; col < terrain.size(); col
         += step_wide)
57     {
58       terrain[row][col] = strategy->avg(terrain, row, col,
           neighbour_dist, square_offset) + random(noise);
59       terrain.set_max_h(max(terrain.get_max_h(), terrain[row][col
           ]));
60       terrain.set_min_h(min(terrain.get_min_h(), terrain[row][col
           ]));
```

```
61      }
62    }
63  }
64
65  Terrain Diamond_Square::generate_terrain()
66  {
67    Terrain terrain(n);
68
69    int step_wide = n;
70    double rand_noise = 1;
71
72    while (step_wide > 1)
73    {
74      diamond_step(terrain, step_wide, rand_noise);
75      diamond_step_row(terrain, step_wide, rand_noise);
76      diamond_step_col(terrain, step_wide, rand_noise);
77
78      step_wide /= 2;
79      rand_noise *= smooth;
80    }
81
82    return terrain;
83  }
```

3.3 Интерфейс программы

Интерфейс программы представлен на рисунке 3.1.

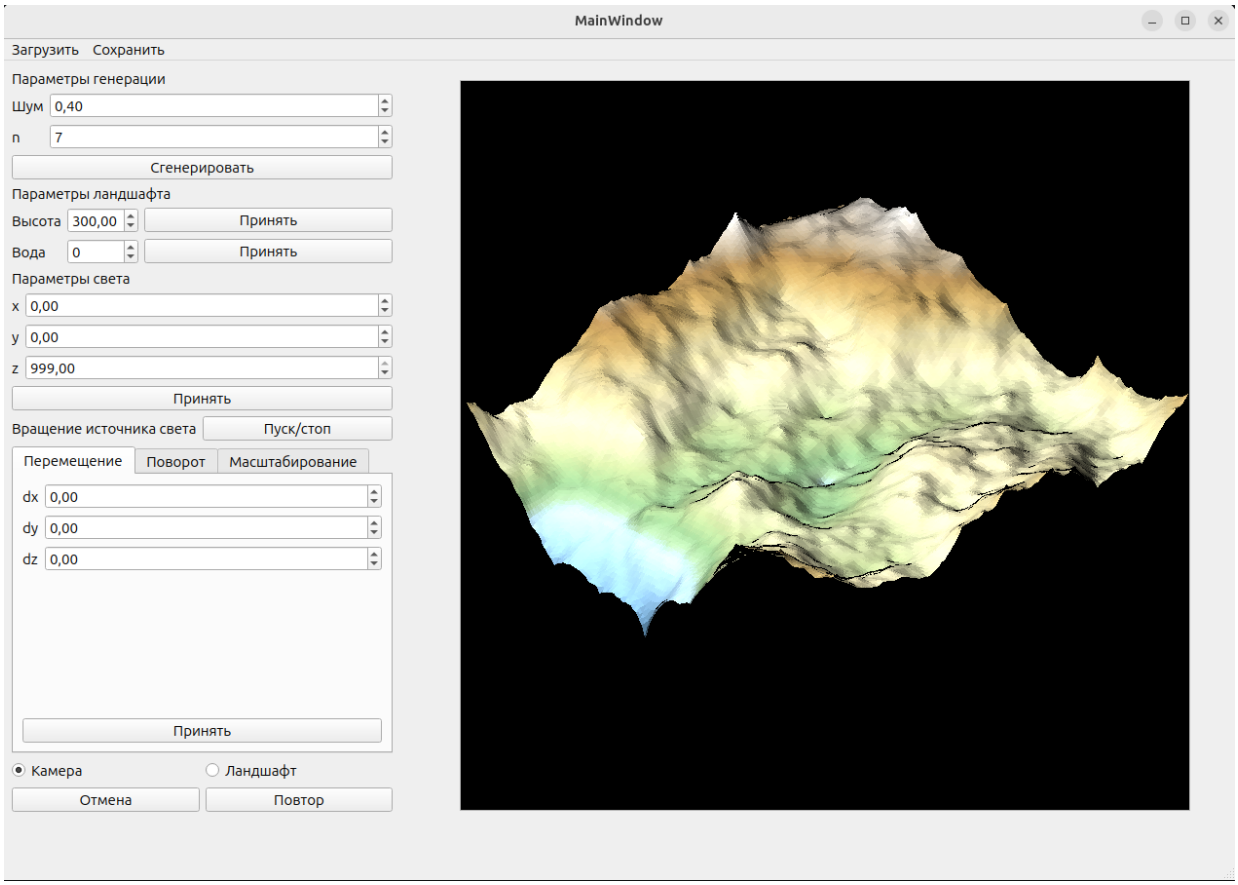

Рисунок 3.1 — Интерфейс программного обеспечения

Вывод

В данном разделе были выбраны средства реализации для программного обеспечения, и предоставлены реализации разработанных алгоритмов. Также описан графический интерфейс.

4 Исследовательская часть

4.1 Технические характеристики

— процессор – Intel® Core™ i5-4590 × 4 [6];

— объём оперативной памяти – 24 ГБ;

— операционная система – Ubuntu 24.04.1 LTS [5].

4.2 Исследование

Было проведено исследование зависимости времени генерации ландшафта от его размера. Результаты замеров приведены в таблице 4.1. График зависимости приведён на рисунке 4.1.

Таблица 4.1 — Данные замеров

n	время генерации
2	94239
3	100341
4	86143
5	148750
6	225966
7	433758
8	1134237
9	3378200
10	11926117
11	42428064
12	160699571

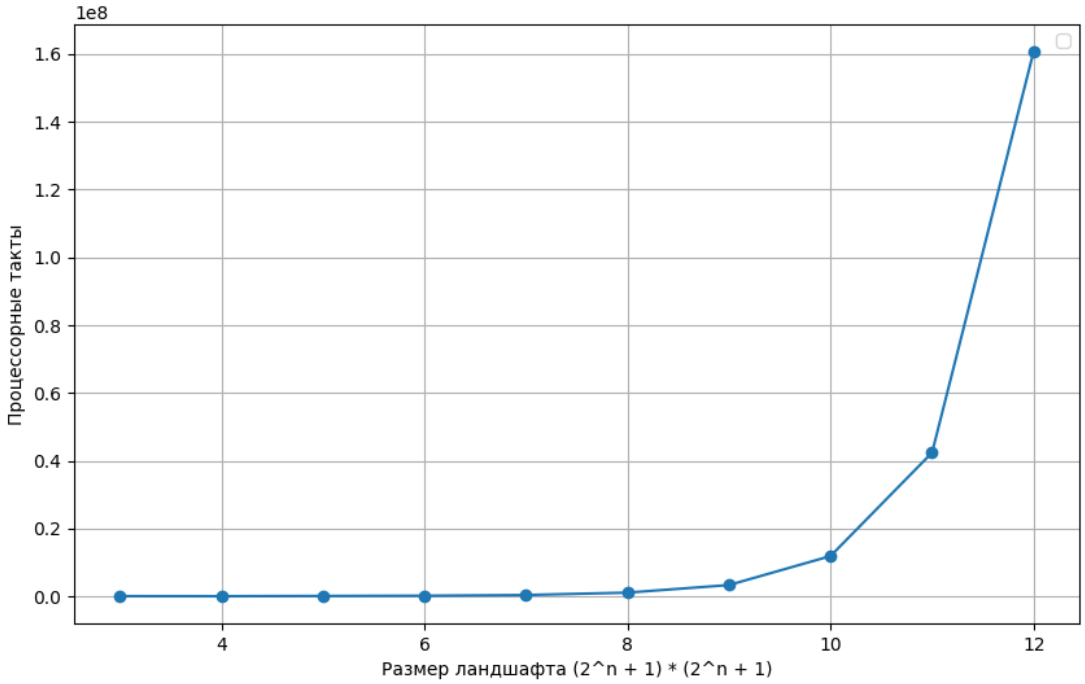

Рисунок 4.1 — График зависимости времени генерации ландшафта от его размера

Вывод

Из проведенного исследования можно сделать вывод: при размерах карты больших чем $(2^{11} - 1) \times (2^{11} - 1)$ алгоритм генерации ландшафта становится крайне времязатратным.

ЗАКЛЮЧЕНИЕ

Цель работы была достигнута: была выполнена разработка программного обеспечения для генерации реалистичного трёхмерного ландшафта. Для достижения цели были решены следующие задачи:

1) проведён анализ алгоритмов компьютерной графики для создания трёхмерных тел;
2) выбраны алгоритмы для решения поставленной задачи;
3) спроектирован алгоритм решения поставленной задачи;
4) выбраны средства реализации;
5) проведены исследование зависимости времени генерации ландшафта от его размера.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Воксельная модель [Электронный ресурс]. Режим доступа: https://www.media.mit.edu/projects/making-data-matter/overview/ (Дата обращения 16.12.24)

2. Диаграмма Вороного [Электронный ресурс]. Режим доступа: https://people.csail.mit.edu/indyk/6.838/handouts/lec8.pdf (Дата обращения 16.12.24)

3. Разработка адаптивного модуля создания и исследования виртуальных моделей объектов окружающей среды. Шардаков В.М., Извозчикова В.В., Запорожко В.В., Парфёнов Д.И.

4. Роджерс Д. Алгоритмические основы машинной графики: Пер. с англ. — Москва : Мир, 1989. — С. 512.

5. Сайт дистрибутива [Электронный ресурс]. Режим доступа: https://ubuntu.com/ (Дата обращения 16.12.24)

6. Документация к процессору [Электронный ресурс]. Режим доступа: https://www.intel.com/content/www/us/en/products/sku/80815/intel-core-i54590-processor-6m-cache-up-to-3-70-ghz/specifications.html (Дата обращения 16.12.24)

7. Документация C++ [Электронный ресурс]. Режим доступа: https://isocpp.org/std/the-standard (Дата обращения 16.12.24)

8. Документация Qt [Электронный ресурс]. Режим доступа: https://doc.qt.io/ (Дата обращения 16.12.24)

Приложение А