Оглавление

[Введение 1](#_Toc81227314)

[Аналитическая часть 2](#_Toc81227315)

[Конструкторская часть 3](#_Toc81227316)

[Алгоритм генерации ландшафта 3](#_Toc81227317)

[Алгоритм удаления невидимых поверхностей 4](#_Toc81227318)

[Выбор структур данных 6](#_Toc81227319)

[Технологическая часть 7](#_Toc81227320)

[Основные инструменты используемые для реализации проекта 7](#_Toc81227321)

[Реализация 10](#_Toc81227322)

# Введение

Генерация ландшафта постоянно используются при моделировании трехмерных виртуальных сцен. Подобные сцены используются в фильмах с использованием компьютерной графики, трехмерных играх или в качестве демонстрационного материала для различных архитектурных объектов.

В кинематографе трехмерные ландшафты применяются, например, для создания сцен больших размеров, ручное моделирование которых оказалось бы нецелесообразно. Компьютерные игры используют методы генерации ландшафтов для создания уникальных игровых миров.

Инструменты для моделирования ландшафтов зачастую являются частью крупного программного продукта, установка и освоение подобных программ может быть довольно сложным процессом. Они позволяют сэкономить время для художника, способствуют быстрой итерации изменений.

Основной целью проекта является разработка программы генерации и построения трехмерного изображения ландшафта.

Для реализации данного проекта, необходимо решить ряд задач:

* выбрать представление данных о ландшафте;
* выбрать алгоритм генерации ландшафта;
* решить вопрос представления ландшафта;
* выбрать алгоритм удаления невидимых поверхностей;
* выбрать необходимые структуры данных для изображения ландшафта.

# Аналитическая часть

## Представление данных о ландшафте

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

* Первый – использование регулярной сетки высот (карты высот);
* Второй – использование иррегулярной сетки вершин и связей, их соединяющих;
* Третий – хранение карты ландшафта, но в данном случае хранятся не конкретные высоты, а информация об использованном блоке. В этом случае создается некоторое количество заранее построенных сегментов, а на карте указываются только индексы этих сегментов.

### Регулярная сетка высот

В карте высот данные представлены в виде двухмерного массива, каждому элементу которого соответствует значение высоты в точке (x, y), где x и y – индексы матрицы.

С помощью этого способа можно представить достаточно обширные пространства. Но у него есть один существенный недостаток — слишком много описаний для точек, а также, в некоторых случаях, наблюдается избыточность данных (например, когда задается простая плоскость, то в этом случае, для построения простой плоскости будет использоваться множество точек, хотя можно было обойтись тремя). Но с другой стороны, в некоторых ситуациях, данная избыточность может стать плюсом. В случае, если мы хотим изменить какую-то конкретную высоту или область ландшафта, мы легко можем это сделать. Кроме этого, для каждого элемента такой карты можно хранить не только значения высот, но и другие параметры, которые хранят информацию об особенностях ландшафта в конкретной точке (например, цвет). Также, поскольку вершины расположены регулярно и достаточно близко, можно более точно производить динамическое освещение.

### Иррегулярная сетка

Еще один способ представления данных для ландшафтов — иррегулярная сетка вершин и связей их соединяющих.

По сравнению с картой высот используется значительно меньше информации для построения ландшафта. При данном подходе необходимо хранить значения высот вершин и информацию о связях между ними только для конкретного, небольшого количества точек. Это дает выигрыш в скорости при передаче огромных массивов информации, в процессе визуализации ландшафта. Однако алгоритмы построения ландшафтов в основном предназначены для регулярных карт высот. Оптимизация таких алгоритмов под этот способ потребует значительных усилий. Также, поскольку вершины расположены достаточно далеко друг от друга и неравномерно, возникают сложности при динамическом освещении. Кроме того, хранение, просмотр и модификация такого ландшафта тоже предоставляет сложности.

### Посегментная карта высот

В данном способе также используются карты высот. Только вместо высот в ней хранятся индексы *ландшафтных сегментов*. Как эти сегменты представлены, в принципе, роли не играет. Они могут быть и регулярными, и иррегулярными (причем можно использовать и те, и другие одновременно).

Это дает нам следующие преимущества:

* возможность представления огромнейших открытых пространств;
* кроме самих ландшафтов в таких блоках можно хранить и информацию о зданиях, строениях, растениях, специфических ландшафтных решениях (например, пещеры или скалы, нависающие друг над другом);
* возможность создания нескольких вариантов одного и того же сегмента, но при разной степени детализации. В зависимости от скорости или загруженности компьютера можно выбирать более или менее детализованные варианты.

Но мы также сталкиваемся со следующими проблемами:

* сложность стыковки разных сегментов
* данные представляющие ландшафт не тривиальны (сложно представить, как это будет выглядеть)

### Вывод

Поскольку для достижения нашей цели нет необходимости в представлении огромных ландшафтов, можно откинуть вариант посегментной карты высот. А между регулярной и иррегулярной картой высот, логично выбрать первый вариант, т.к. в данном случае, у нас не будет проблем с генерацией ландшафта, его освещением и хранением данных о нём.

## Выбор алгоритма генерации ландшафта

### «Простой» способ

Данный алгоритм можно разделить на следующие шаги:

1. Заполнение карты высот случайными значениями в определённом диапазоне;
2. Усреднение значений высот. Для каждой высоты берутся её собственное значение и значения всех соседних вершин, а затем их сумма делится на их количество и присваивается текущей вершине. Максимальное количество вершин для вычисления – 9.

Плюсом данного алгоритма является максимальная лёгкость понимания и простота реализации. Но данный алгоритм не даёт реалистичных результатов. Ландшафт полученный с помощью него выглядит ломанной равниной при маленьком диапазоне случайных чисел и как очень скалистые горы при большом. Можно улучшить вид ландшафта, применив к нему шаг 2 несколько раз, но потребует лишнее время при генерации ландшафта.

### Шум Перлина

В бытовом смысле, «шум» — это случайный мусор. Шум является белым если все значение вычисленные с его помощью независимы друг от друга. Например, если мы будем генерировать с его помощью строчку из нулей и единиц, то в среднем их количество должно совпадать. Шум полезен для генерации случайных шаблонов, особенно для непредсказуемых природных явлений. Однако большинство вещей не чисто случайны. Дым, облака, ландшафт могут иметь некий элемент случайности, но они были созданы в результате очень сложных взаимодействий множества крохотных частиц. Белый шум содержит независимые частицы. Для генерации ландшафта белый шум не подойдёт.

Шум Перлина – это градиентный шум, состоящий из набора псевдослучайных единичных векторов (направлений градиента), расположенных в определенных точках пространства и интерполированных функцией сглаживания между этими точками. В отличии от белого шума, шум Перлина непрерывный и плавный – нету резких переходов от маленьких значений к большим. Для генерации шума Перлина в одномерном пространстве необходимо для каждой точки этого пространства вычислить значение шумовой функции, используя направление градиента (или наклон) в указанной точке. Алгоритм шума Перлина можно масштабировать одно-, двух- и трёхмерного вида. Более того, в алгоритм можно ввести четвёртое временное измерение, позволяя алгоритму динамически изменять текстуры во времени.

Шум Перлина широко используется в двухмерной и трёхмерной компьютерной графике для создания таких визуальных эффектов, как дым, облака, туман, огонь и т. д. Он также очень часто используется как простая текстура, покрывающая геометрическую модель. В отличие от растровых текстур, шум Перлина является процедурной текстурой, и поэтому он не занимает память, но вместе с тем исполнение алгоритма требует неких вычислительных ресурсов.

### Холмовой алгоритм (Hill Algoritm)

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

* Создаем двухмерный массив и инициализируем его нулевым уровнем (заполняем все ячейки нолями);
* Берем случайную точку на ландшафте или около его границ (за границами), а также берем случайный радиус в заранее заданных пределах. Выбор этих пределов влияет на вид ландшафта — либо он будет пологим, либо скалистым;
* В выбранной точке "поднимаем" холм заданного радиуса;
* Возвращаемся ко второму шагу и так далее до выбранного количества шагов. От него потом будет зависеть внешний вид нашего ландшафта;
* Проводим нормализацию ландшафта;
* Проводим "долинизацию" ландшафта. Делаем его склоны более пологими.

Для получения реалистичных результатов с помощью этого алгоритма потребуется много вычислительных ресурсов и большинство ландшафтов будут похожи на горы или холмистую равнину.

### Diamond square

Самым же распространенным и дающим одни из самых реалистичных результатов является алгоритм diamond-square (или square-diamond), расширение одномерного алгоритма midpoint displacement на двумерную плоскость.

Алгоритм midpoint displacement – рекурсивный. Изначально любым образом задаётся высота на концах отрезка и разбивается точкой посередине на два под-отрезка. Эту точку смещают на случайную величину и повторяют разбиение и смещение для каждого из полученных под-отрезков. И так далее — пока отрезки не станут длиной в один пиксель. Случайные смещения должны быть пропорциональны длинам отрезков, на которых производятся разбиения. Данный алгоритм можно использовать, например, для генерации линии горизонта.

В случае же алгоритма diamond-square, вычисления производятся в двумерном пространстве – карте высот. На вход подаётся плоская поверхность, высота вершин которых равна нулю. Затем присваиваются значения к угловым высотам. После этого алгоритм можно разбить на два шага:

* Шаг diamond – в нём вычисляется срединная точка текущего квадрата для квадрата, путём усреднения значений угловых вершин и добавлением случайного числа.
* Шаг square – в нём вычисляются средние точки рёбер для текущего квадрата, путём усреднения вершин слева, справа, сверху и снизу и добавлением случайного числа.

Далее карта высот делится на 4 меньших квадрата и шаги diamond, square повторяются для них, пока квадраты не выродятся в точку.

### Вывод

«Простой» способ является слишком тривиальным решением поставленной задачи, которое не даёт удовлетворявших результатов, поэтому его можно сразу исключить. Холмовой алгоритм, требует слишком много вычислительных ресурсов и выдаёт однообразные ландшафты. Шум Перлина выглядит интересным решением, но алгоритм diamond-square наиболее распространён и даёт одни из самых реалистичных результатов. Кроме того, я лично заинтересовался в его реализации.

Следовательно, для генерации ландшафта я выбрал алгоритм diamond-square.

## Выбор алгоритма удаления невидимых поверхностей

Для решения задачи удаления невидимых поверхностей я выбрал алгоритм z-buffer в силу его безразличия к сложности сцены.

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

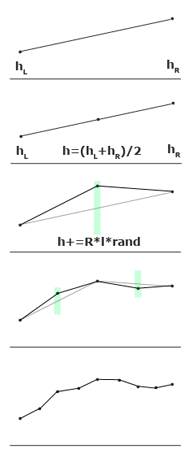
Основной недостаток алгоритма - большой объем требуемой памяти. Кроме этого сложно устранить лестничный эффект или реализовать эффект прозрачности.

## Представление ландшафта

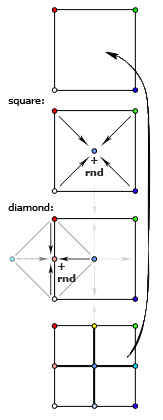
Чтобы изобразить поверхность ландшафта я решил использовать треугольные полигоны, т.к. на основе любых трёх точек может образовать поверхность, что будет очень удобно для нас в будущем.

# Конструкторская часть

## Diamond-square

Данный алгоритм является расширением одномерного алгоритма midpoint displacement на двумерную плоскость. В кратце алгоритм midpoint displacement можно описать так: изначально мы любым образом задаем высоту на концах отрезка и разбиваем его точкой посередине на два под-отрезка. Эту точку мы смещаем на случайную величину и повторяем разбиение и смещение для каждого из полученных под-отрезков. И так далее — пока отрезки не станут длиной в один пиксель. Важное замечание: случайные смещения должны быть пропорциональны длинам отрезков, на которых производятся разбиения. Например, мы разбиваем отрезок длиной *l* — тогда точка посередине него должна иметь высоту ( и — высоты на левом и правом конце отрезка, а константа *R* определяет «шероховатость» (roughness) получающейся ломаной и является главным параметром в данном алгоритме).

Если же говорить об алгоритме diamond-square, то можно разбить его на следующие шаги:

1. В начале нам необходимо задать начальные значения для угловых вершин (в моём случае это нули).
2. Шаг square. Определение центральной вершины, путем усреднения угловых и добавлением случайного числа.
3. Шаг diamond. Определение срединных вершин для ромбов путём усреднения вершин сверху, снизу, слева и справа. Если же какая-то из вершин выходит за границу карты высот, то такую точку можно либо не учитывать, либо считать равной нулю, благодаря чему, ближе к краям ландшафта он будет снижаться.
4. Повторить шаги square-diamond для меньших квадратов. При этом важно заметить, что две вершины, которые достались нам на шаге square, должны быть уже посчитаны — поэтому обсчет нужно вести «слоями», сначала для всех квадратов выполнить шаг «square» — затем для всех ромбов выполнить шаг «diamond» — и далее перейти к меньшим квадратам.

Кроме того, необходимо чтобы размерность карты высот была , где n – целое, натуральное число. Это необходимо для того, чтобы на каждом шаге square была центральная вершина.

## Алгоритм удаления невидимых поверхностей

Алгоритм, использующий z-буферэто один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом. Работает этот алгоритм в пространстве изображения. Данный алгоритм использует буфер кадра и z-буффер равных размеров. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пиксела в пространстве изображения, z-буфер - это отдельный буфер глубины, используемый для запоминания координаты *z* или глубины каждого видимого пиксела в пространстве изображения. В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по х и у наибольшего значения функции z (х, у).

## Выбор структур данных

Поскольку мы используем карту высот для представления ландшафта, разумно использовать матрицу со значениями каждой вершины, для генерации карты высот. Но поскольку нам необходимо реализовать трёхмерные преобразования для нашего ландшафта, нам нужно также знать не только высоту каждой вершины, но и две другие координаты, поэтому на основе сгенерированной карты высот будет создана матрица точек с координатами X, Y и Z.

Как было написано выше, для изображения ландшафта используются треугольные полигоны. Каждый полигон будет состоять из 3-ёх точек матрицы, которую мы получили ранее. Для того чтобы хранить эти треугольные полигоны используем массив.

На основе этих данных мы уже можем изобразить каркас нашего ландшафта, но нам необходимо также решить задачу удаления невидимых поверхностей. Поскольку мы выбрали алгоритм, использующий Z-buffer, нам необходимо создать еще две матрицы: буфер кадра и Z-буфер. Для реализации нашего алгоритма, также необходимо иметь возможность вычислять координаты Z для каждого X, Y наших полигонов, поэтому нужно вычислять коэффициенты плоскости для каждого полигона.

# Технологическая часть

## Основные инструменты используемые для реализации проекта

1. Язык программирования C++

Данный язык был выбран, в первую очередь потому, что я его знаю лучше остальных. Но выбрал для изучения я его не случайно, а за его плюсы:

1. Высокая производительность: Язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы. Один из базовых принципов C++ «не платишь за то, что не используешь» то есть ни одна из языковых возможностей, приводящая к дополнительным накладным расходам, не является обязательной для использования. Имеется возможность работы с памятью на низком уровне.
2. Кроссплатформенность: стандарт языка C++ накладывает минимальные требования на ЭВМ для запуска скомпилированных программ.
3. Доступность: Для C++ существует огромное количество учебной литературы, переведённой на всевозможные языки. Язык имеет высокий порог вхождения, но среди всех языков такого рода обладает наиболее широкими возможностями.
4. Поддержка различных стилей программирования: традиционное императивное программирование (структурное, объектно-ориентированное), обобщённое программирование, функциональное программирование, порождающее метапрограммирование.

Это не все достоинства C++, но основные, в связи с которыми я выбрал именно этот язык.

1. Кроссплатформенный фреймворк Qt

Для реализации моего проекта, необходима была библиотека, упрощающая работу с графикой и сетями. На примете были Qt и MFC. Чтобы сделать выбор, мне пришлось их сравнить.

Qt – кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++. Есть также «привязки» ко многим другим языкам программирования: Python — PyQt, PySide; Ruby — QtRuby; Java — Qt Jambi; PHP — PHP-Qt и другие.

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода (кроссплатформенность). Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Комплектуется визуальной средой разработки графического интерфейса Qt Designer, позволяющей создавать диалоги и формы.

Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Мета-объектная система — часть ядра фреймворка для поддержки в С++ таких возможностей, как сигналы и слоты для коммуникации между объектами в режиме реального времени и динамических свойств системы

Одним из преимуществ проекта Qt является наличие качественной документации, в отличие, например, от wxWidgets. Статьи документации снабжены большим количеством примеров. Исходный код самой библиотеки хорошо форматирован, подробно комментирован, что также упрощает изучение Qt.

Отличительная особенность — использование мета-объектного компилятора — предварительной системы обработки исходного кода. Расширение возможностей обеспечивается системой плагинов, которые возможно размещать непосредственно в панели визуального редактора. Но минусом получается то, что код написанный с помощью Qt нельзя скомпилировать на другом компьютере без установки фреймворка.

Microsoft Foundation Classes – библиотека на языке C++, разработанная Microsoft и призванная облегчить разработку GUI-приложений для Microsoft Windows путём использования богатого набора библиотечных классов.

Во-первых если сравнивать только работу с GUI, то данная библиотека работает только под Windows, то есть ни о какой кроссплатформенности речи не идёт. Но не стоит забывать о том что Qt в отличии от MFC имеет множество других полезных классов, и что важно в данном случае, упрощает работу с сетью. Во-вторых, если же в MFC создать каркас приложения без дизайнера достаточно сложно, то в Qt это зачастую даже намного удобнее и проще.

Поскольку функционал Qt намного шире, то для реализации проекта был выбран именно фреймворк Qt.

1. Среда разработки Qt creator

Qt Creator (ранее известная под кодовым названием Greenhouse) — кроссплатформенная свободная IDE для языков С, С++ и QML. Разработана Trolltech (Digia) для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML. Поддерживаемые компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

Основная задача Qt Creator — упростить разработку приложения с помощью фреймворка Qt на разных платформах. Поэтому для работы с данной библиотекой был выбран именно он.

1. Система версионного контроля git:

Для хранения исходников используется система Git (на портале github.com), т.к. это крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

## Реализация

Тестовый текст.