# Тема проекта

Я Пронин А.С. из группы ИУ7-52б, и тема моего проекта генерация трехмерного ландшафта.

# Цель

Цель работы – разработать программу генерации и визуализации ландшафта. А поставленные задачи вы видите на слайде.

# Представление данных о ландшафте

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

* первый – карта высот;
* второй – иррегулярная сетка вершин и связей, их соединяющих;
* третий – посегментная карта высот 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.

Сравним принципы представления данных о ландшафте: 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

На слайде вы можете видеть сравнение проанализированных мною алгоритмов генерации ландшафтов. Мною был выбран алгоритм diamond-square.

Сравним алгоритмы генерации ландшафта: «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.

В данном алгоритме 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. 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.

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

# Выбор алгоритма удаления невидимых поверхностей

На следующем слайде представлено сравнение проанализированных мною алгоритмов удаления невидимых поверхностей. Для решения этой задачи, я выбрал алгоритм, использующий Z-buffer.

Сравним алгоритмы удаления невидимых поверхностей: алгоритм плавающего горизонта можно сразу отбросить потому, что кроме визуализации ландшафта, нам будет необходимо реализовать для него трёхмерные преобразования и освещение. А если выбирать между алгоритмом Робертса и z-буфером, для реализации поставленной цели, по моему мнению, разумно выбрать второй вариант по следующим причинам: во-первых, алгоритм использующий z-buffer имеет линейную зависимость вычислительной трудоёмкости от кол-ва объектов, а при использовании алгоритма Робертса, для достижения зависимости приближённой к линейной, необходимо произвести предварительную приоритетную сортировку вдоль оси z и использовать простые габаритные или минимаксные тесты. Во-вторых, алгоритму z-буфера безразлично на сколько сложна визуализируемая сцена. Кроме этого, для поставленной задачи, первый этап алгоритма Робертса оказывается бессмысленным, т.к. он работает только для выпуклых многогранников, а мы работаем со случайно сгенерированным ландшафтом. Следовательно, для решения задачи удаления невидимых поверхностей, был выбран алгоритм, использующий z-buffer.

Работает этот алгоритм в пространстве изображения. Он использует два буфера: буфер кадра и Z-buffer. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пиксела в пространстве изображения, а 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).

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.

# Выбор структур данных

На данном слайде, вы можете видеть диаграммы основных классов, созданных и используемых мной.

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

# Основные инструменты, используемые для реализации и исследования

1. Язык программирования C++.
2. Кроссплатформенный фреймворк Qt.
3. Среда разработки Qt creator.
4. Система версионного контроля git.
5. Библиотека clock().

# Пример работы программы

На следующих слайдах вы можете наблюдать, примеры результата работы моей программы.

# Сравнительный анализ времени выполнения алгоритма ZBuffer с использованием параллельных вычислений и без

Также мною был проведён сравнительный анализ времени выполнения алгоритма ZBuffer с использованием параллельных вычислений и без. Результаты которого вы можете видеть на таблице.

Чтобы провести сравнительный анализ времени выполнения алгоритмов, замерялось реальное время работы алгоритма ZBuffer 100 раз и делилось на кол-во итераций. В таблице 4.1 показаны результаты тестирования алгоритма, использующего параллельные вычисления, для разного кол-ва потоков.

Как и ожидалось, наиболее быстрым, является алгоритм, использующий параллельные вычисления.

# Оценка времени выполнения алгоритма diamond-square в зависимости от размера.

Вторым моим исследованием была оценка времени выполнения алгоритма diamond-square в зависимости от размера. Для этого замерялось реальное время для карт высот следующих размеров: 33x33, 65x65, 129x129, 257x257, 513x513, 1025x1025. Алгоритм выполнялся раз, где N – размер стороны карты высот ландшафта.

На соответствующей таблице, показаны результаты моего эксперимента. (Шаг “Smoothing” это усреднение каждой вершины, используя соседние.)

Как и ожидалось, с ростом размера карты высот будет увеличиваться и время выполнения алгоритма diamond-square. Кроме этого, время выполнения алгоритма увеличивается на ~11% если выполнять шаг “Smoothing”.

# Заключение

По итогу проделанной работы была достигнута цель – разработана программа генерации и визуализации трехмерного изображения.

Также были решены все поставленные задачи, перечисленные на экране.

# Демонстрация

Теперь можно приступить к демонстрации ПО.