Санкт-Петербургский Государственный Университет

Прикладная Математика и Информатика

Нелинейная динамика, информатика и управление

Сальникова Полина Валерьевна

Процедурная Генерация Контента

Бакалаврская работа

Научный руководитель:

Старший преподаватель кафедры Прикладной Кибернетики СПбГУ,

Селеджи С.М.

Рецензент:

Кандидат физико-математических наук, старший научный сотрудник кафедры

Прикладной Кибернетики СПбГУ, Кудряшева Е.В.

Санкт-Петербург

2016

SAINT-PETERSBURG STATE UNIVERSITY

Applied Mathematics and Computer Science

Nonlinear Dynamics, Computer Science and Control

Polina Salnikova

Procedural Content Generation

Bachelor's Thesis

Scientific supervisor:

Associate Professor of Applied Cybernetics Department,

St. Petersburg State University, Svetlana Seledzhi

Reviewer:

Ph.D, Researcher of Applied Cybernetics Department,

St. Petersburg State University, Elena Kudryashova

Saint-Petersburg

2016

Содержание

Введение ……..…2

1. Основная структура данных
   1.1 Постановка задачи……..……5
   1.2 Мозаика Вороного……..…..…10
   1.3 Алгоритм Фортуна…….…..12
   1.4 Алгоритм Ллоида…….……22
   1.5 Смежные графы вершин и граней…….…...….25

2. Заполнение карты
   2.1 Генерация участка суши …….……..……..27
   2.2 Карта высоты…….……...……...……29
   2.3 Генерация рек…….…….…….30
   2.4 Карта влажности…….……..…31
   2.5 Типы местности……...……32

3. Описание реализации
   3.1 Краткое описание……...…..33
   3.2 Классы и Методы……...…..34
   3.3 Результаты выполнения программы……..37

Заключение……...…..41
Список литературы ……..…... 42

Введение

Целью данной бакалаврской работы является исследование методов процедурной генерации контента и создание программы, осуществляющей реализацию этих методов. Результатом проделанной работы будет программа-генератор, позволяющая строить изображения двумерных карт местности. Создаваемая система предполагает дальнейшее развитие и расширение в будущем, с последующей интеграцией в компьютерную игру, в которой она будет выполнять функцию генератора карт-уровней.

Процедурная генерация (ПГК) — это группа методов, в которых при помощи особых вычислительных алгоритмов создаются модели, которые изображают различные объекты — дома, растения, ландшафт, текстуры [1. 1].

Построение моделей различных объектов методом ПГК— распространенная задача, которая находит применение во многих сферах, связанных с компьютерной графикой, например, в создании компьютерных игр или мультфильмов. Этот метод используется тогда, когда нужно получить большое множество моделей с уникальными характеристиками, которое нельзя получить в разумные сроки и при разумных трудозатратах обычным способом построения вручную.

Например, при создании текстур вручную, художники используют два метода или их комбинацию. Первый метод основан на использовании реальных фотоизображений с последующим редактированием. Второй — это создание текстуры полностью при помощи графического планшета. Эти методы обладают серьезным недостатком — невозможностью быстрого итерирования вариантов текстуры. В обоих случаях такие текстуры лишены даже минимальной параметризации картинки и как следствие — возможности

быстрого изменения путём редактирования этих параметров. Кроме того, создание текстур этими методами сводится к рутинной работе: множество движений кисточкой, бесконечное перебирание фототекстур в поисках подходящей и ретуширование с целью правки нежелательных элементов. При процедурной генерации текстуры, цвет каждого пикселя изображения формируется алгоритмически, что позволяет избежать всю рутинную работу и значительно снизить трудозатраты [2. 1].

По сравнению с традиционными методами создания графического контента, процедурная генерация обладает следующими значительными преимуществами:

- Возможность вносить изменения в любой этап формирования изображения (недеструктивное редактирование).

- Возможность увеличения или уменьшения размера изображения без потерь детализации.

- Высокая скорость создания нового контента на основе существующих наработок путём их модифицирования и комбинирования.

- Быстрое создание подобных изображений. После того, как реализован процедурный генератор, формирующий необходимое изображение, для получения подобного нужно всего лишь изменить входное значение у генератора случайных чисел.

- Малый размер хранимых данных, который не зависит от разрешения картинки. В файл сохраняется только описание алгоритма, занимающее несколько килобайт.

После анализа научной литературы по этой теме, были сформулированы две основные задачи, решению которых посвящена данная работа:

1. Исследование существующих алгоритмов ПГК для создания карт и их возможностей.
2. Реализация метода процедурной генерации контента посредством объектно-ориентированного программирования, а именно языка Java.

1. Основная структура данных

1.1. Постановка задачи

Для того чтобы реализовать методы процедурной генерации карт местности прежде всего необходимо разработать алгоритм прорисовки наиболее разнообразных изображений, исходя из небольшого количества заданных параметров.
Предполагается, что в полученных изображениях будут отображены следующие географические объекты:

- Суша/вода
- Береговая линия
- Реки
- водоемы
- возвышенности/низины
- горы

Также, важно иметь в виду, что сгенерированные карты будут использованы в качестве уровней в компьютерной игре. С одной стороны это упрощает задачу, так как изображения местности, в данном случае, не должны быть ограничены излишней реалистичностью в ущерб функциональности с точки зрения игрового процесса. С другой стороны, построение сложных игровых процессов требует подробной параметризации изображений.

Таким образом, в основе этого алгоритма должна лежать гибкая структура данных, которая будет обеспечивать наиболее оптимальный процесс генерации карт.

Основная структура данных должна соответствовать следующим требованиям:

- возможность действующих процессов и процедур легко считывать информацию, необходимую для построения изображения
- удобная навигация по всей области прорисовки изображения
- возможность генерации сложных геометрических объектов, таких как реки, дороги, водоемы, острова.
- возможность дальнейших модификаций изображения

На данный момент среди наиболее часто используемых алгоритмов ПГК можно выделить три метода, с помощью которых можно построить основу для карты местности:

1. Генерация Карт Высот (HeightMaps)
2. Процедурный Шум (NoiseFunctions)
3. Генерация Плиточной структуры (TileMaps)

Генерация карт высот – это техника построения изображений, в основе которой лежит двухмерный массив, элементами которого являются значения высоты для определенных участков местности. На рис 1.1 изображена карта высот, а на рис 1.2, участок местности, построенный на основе этой карты.

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

рис 1.1

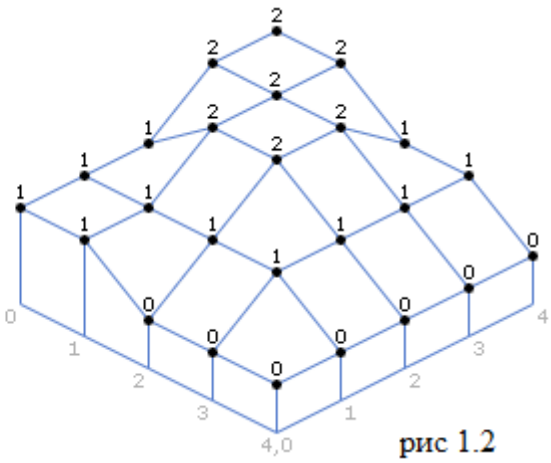


рис 1.2

Не трудно заметить, что подобная техника наиболее эффективна при создании трехмерных изображений. Этот метод лежит в основе таких алгоритмов как Алгоритм Смещения Центра (midpoint displacement), Фрактальная Генерация (Fractals), Алгоритм Ромб-Квадрат (Diamond-Square) [2. 3].

Понятие «Шум» подразумевает собой набор случайных чисел из заданного промежутка значений, без закономерностей и корреляции между двумя соседними числами. Элементарный пример шума – изображение, построенное путем присвоения каждому пикселю случайным образом значения 0(черный цвет) или 1(белый цвет) (Рис 1.3).

Процедурный Шум – это шум, который создается посредством программного кода. Например, при генерации двухмерной текстуры методом процедурного шума, цвет каждого пикселя изображения вычисляется при помощи алгоритмов и математических функций. Наиболее известным из таких алгоритмов является Шум Перлина (Perlin's Noise). Пример изображения, построенного с помощью Шума Перлина (рис 1.4) [1. 1].

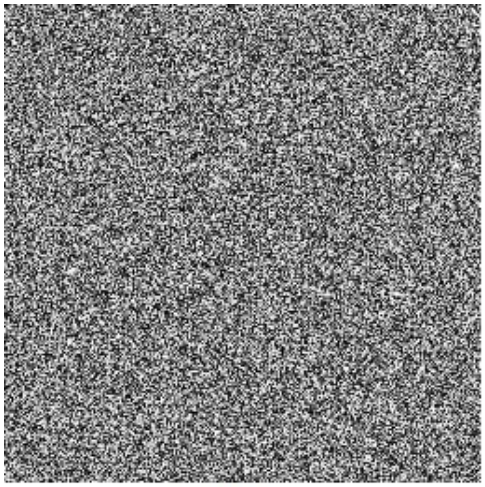


рис 1.3

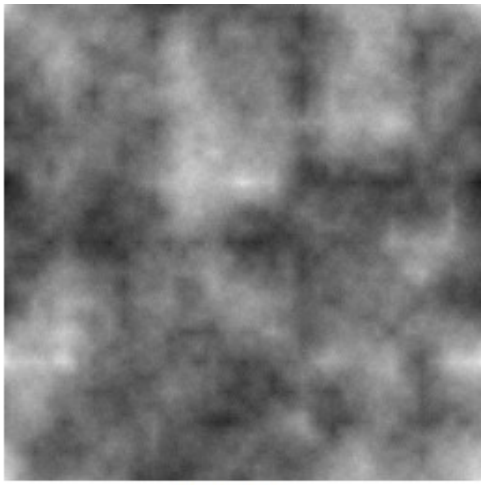


рис 1.4

Алгоритмы шума имеют широкое применение в области компьютерной графики и чаще всего используются при генерации текстур и создании сложных графических объектов, таких как огонь, облака, туман. Также, Шум Перлина применяется при генерации карт местности (рис 1.5 и рис 1.6).

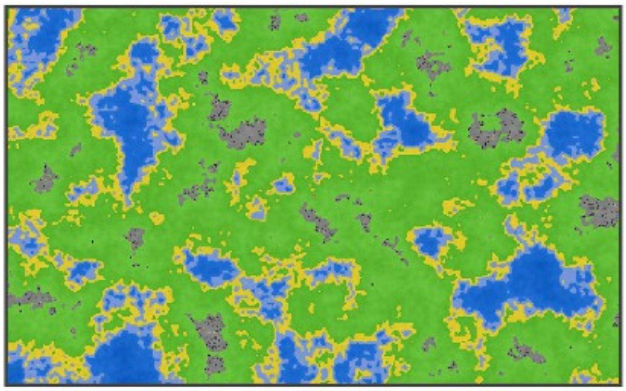


рис 1.5

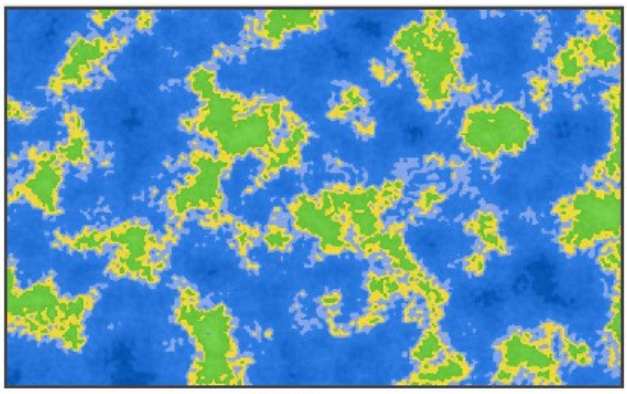


рис 1.6

Очевидно, что подобные изображения крайне трудно параметризировать. Также существенным недостатком шумов, является тот факт, что при частом использовании функций случайных чисел, становится трудно контролировать окончательный результат.

Генерация плиточной структуры, подразумевает собой разбиение области изображения на отдельные сегменты (плитки). Процесс построения изображения осуществляется посредством манипуляций плитками или группами плиток, в основном, путем присваивания им значений цвета.

Эта довольно простая, с виду, структура имеет пару существенных преимуществ. Во-первых, использование плиток значительно упрощает процесс параметризации изображения. Во-вторых, эта структура легко совместима с другими алгоритмами процедурной генерации, такими как, например, Шум Перлина.

В качестве примера карты местности, построенной на плитках, можно привести карту из игры Цивилизация (рис 1.7)

рис 1.7

Таким образом, в результате изучения разнообразных алгоритмов процедурной генерации графических изображений, в качестве основной структуры данных была выбрана Плиточная Структура. Она является наиболее оптимальной и отвечает всем заявленным в данном проекте требованиям.

Для построения этой структуры в данной работе будет использоваться алгоритм Мозаика Вороного [1. 4].

1.2 Мозаика Вороного

Под мозаикой (диаграммой) Вороного понимается следующее. Пусть на всей плоскости задано некое ограниченное и не содержащее одинаковых элементов множество точек (узлов)

$$\{C_i\} \quad i = 1, 2, 3 \ \dots N. \qquad \text{(Рис 2.3)}$$

Порождаемая этим множеством мозаика состоит из полигонов (или плиток), соответствующих узлам C_i и определяемых как

$$V_i = \{x\colon\ d(x, C_i) < d(x, C_m) \ \mid \ \forall\, m \neq i\} \qquad \text{(Рис 2.4)}$$

где $d(x, C_m)$ – эвклидово расстояние между точкой x на плоскости и точкой множества C_s.

Другими словами, к данному полигону мозаики относятся все точки плоскости, расстояние которых до соответствующего полигону V_i узла C_i меньше расстояния до любого другого узла.

рис 2.3

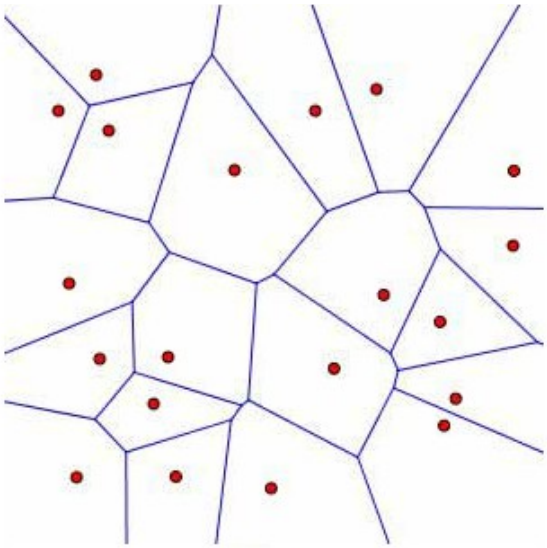


рис 2.4

Мозаика Вороного обладает следующими свойствами:

- **Грани Мозаики**: каждая точка, находящаяся на грани диаграммы Вороного, равноудалена от двух ближайших узлов p_i и p_j. Таким образом, можно нарисовать окружность с центром в этой точке, так, что узлы p_i и p_j окажутся на границе этой окружности и ни один другой узел не окажется внутри этой окружности. Рис 2.5.

- **Вершины Мозаики:** точка, в которой пересекаются три полигона диаграммы, отвечающие узлам p_k,p_i и p_j называется вершиной мозаики. Такая точка также будет равноудалена от этих трех узлов. Рис 2.6.

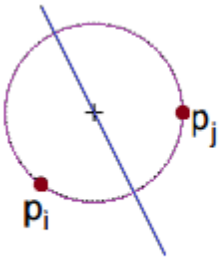


рис 2.5

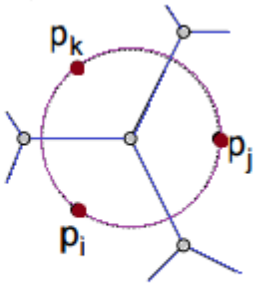


рис 2.6

Для программной реализации построения Мозаики Вороного используется Алгоритм Фортуна [1. 1].

1.3.1 Алгоритм Фортуна

Алгоритм Фортуна относится к типу алгоритмов Заметающей Прямой (Sweep Line Algorithms) [1. 5]. Идея алгоритмов такого типа заключается в том, чтобы ввести прямую линию (Заметающую Прямую), которая будет двигаться по плоскости, останавливаясь в определенных точках. Все геометрические операции такого алгоритма будут выполняться лишь над теми объектами, которые либо пересекают Заметающую Прямую, либо находятся в ее непосредственной близости, во время ее остановки. Решение будет достигнуто тогда, когда Заметающая Прямая пройдет по всем объектам.

Во время выполнения алгоритма Фортуна, Заметающая Прямая будет двигаться сверху вниз по плоскости, на которой находятся случайным образом расположенные точки, которые впоследствии станут узлами диаграммы Вороного.

Для начала, если взять любой узел p и прямую l (не содержащую точку p), то множество таких точек, расстояние которых до p будет равняться расстоянию до l образует параболу (рис 3.1).

Эту параболу мы обозначим за $P_{p,l}$, и будем считать, что она разделяет плоскость на две области: одна состоит из точек, которые ближе к узлу p, а вторая из точек, которые ближе к прямой l.

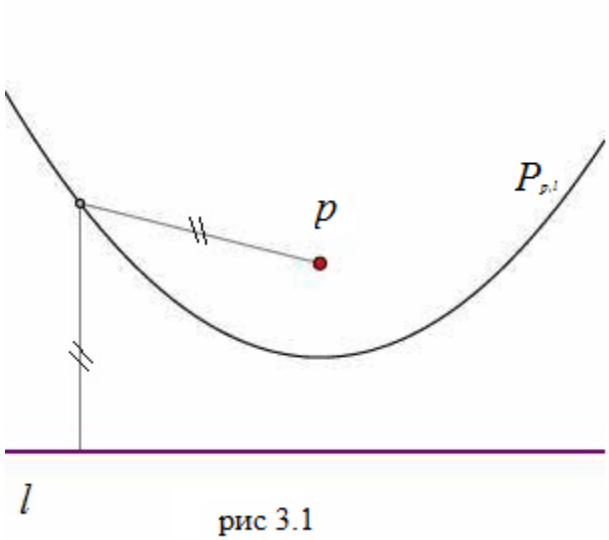


рис 3.1

Это очень важный момент, так что рассмотрим его подробнее. Рассмотрим точку q с координатами $q = (q_x, q_y)$, и с расстоянием до узла p равным $d(q,p)$.

Горизонтальная Прямая l является Заметающей Прямой. Обозначим ее вертикальную координату за l_y.

Таким образом, расстояние между точкой q и прямой l равно $q_y - l_y$, а условие того, что точка q лежит на параболе $P_{p,l}$, можно записать как

$$d(q,p) = q_y - l_y,$$

следовательно,

$$d(q,p) < q_y - l_y, \text{ если } q \text{ лежит над } P_{p,l},$$

$$d(q,p) = q_y - l_y, \text{ если } q \text{ лежит на } P_{p,l},$$

$$d(q,p) > q_y - l_y, \text{ если } q \text{ лежит под } P_{p,l}.$$

Итак, по ходу выполнения алгоритма, Заметающая Прямая l будет двигаться по плоскости сверху вниз. В каждый момент времени мы будем рассматривать те узлы, которые лежат над прямой и порожденные ими параболы $P_{p,l}$.

Например, учитывая множество узлов и положение заметающей прямой, изображенных на рис 3.2 , мы будем рассматривать параболы на рис 3.3.

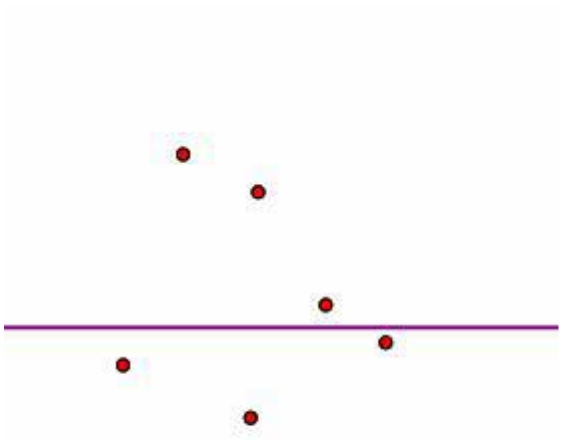


рис 3.2

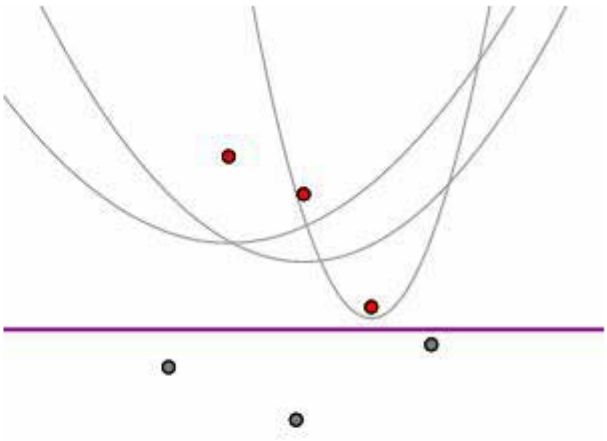


рис 3.3

Следующим важным элементом алгоритма Фортуна является Параболическая

Кривая (Beach Line).

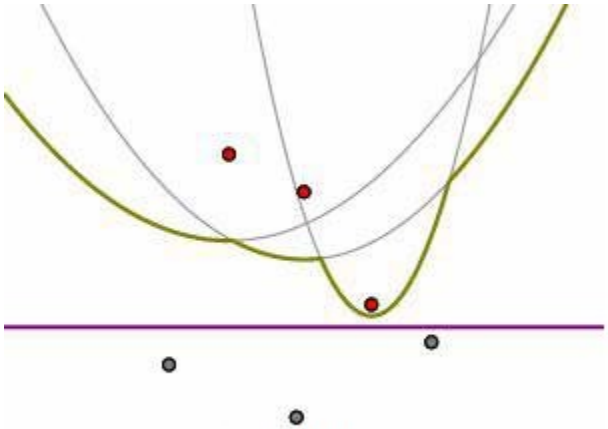


рис 3.4

Эта кривая состоит из нижних параболических дуг, каждая из которых соответствует одному из узлов, находящихся над Заметающей Прямой. На рис 3.4 Параболическая Кривая выделена зеленым цветом.

Параболическая Кривая очень удобна для конструирования мозаики Вороного. Например, если данная точка лежит над кривой, то она находится ближе к одному из узлов диаграммы над прямой l чем к самой прямой l. Это означает, что данная точка лежит внутри одной из плиток мозаики, через которую уже прошла заметающая прямая.

Определим теперь, в какой момент кривая проходит через фиксированную точку q. Допустим, что расстояние от точки q до узла p_1 не больше, чем ее расстояние до любого другого узла; то есть

$$d(q, p_1) \leq d(q, p_i), \forall i.$$

Условие того, что q лежит на параболе $P_{p1,l}$ выражается следующим образом:

$$d(q, p_1) = q_y - l_y \ ,$$

и следовательно:

$$d(q, p_i) \geq d(q, p_1) = q_y - l_y$$

Это означает, что, когда точка q попадает на параболу $P_{p1,l}$, она не может находиться над какой-либо другой параболой $P_{pi,l}$. Следовательно, когда q оказывается на параболе $P_{p1,l}$, она оказывается на параболической кривой, а

именно, на одной из составляющих ее параболических дуг, каждая из которых соответствует одному из узлов диаграммы.

Те точки параболической кривой, которые лежат на двух составных дугах одновременно, называются Граничными Точками (Break Points). Таким образом, применяя предыдущее рассуждение, можно сделать вывод, что граничные точки находятся на одинаковом расстоянии от двух ближайших узлов диаграммы.

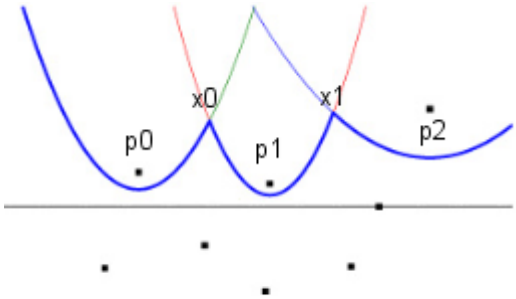


рис 3.5

На рис 3.5, x0, x1 – граничные точки, и

$$d(x_0,p_0) = d(x_0,p_1),$$

$$d(x_1,p_1) = d(x_1,p_2)$$

Таким образом, граничные точки лежат на Гранях мозаики Вороного, и с их помощью, по ходу выполнения алгоритма Фортуна можно будет обозначить грани полигонов.

1.3.2 Обозначение Граней

Итак, для построения мозаики Вороного, необходимо наблюдать за парами граничных точек по мере движения Заметающей Прямой по плоскости. Для этого, необходимо отметить, в какой момент определяется такая пара. Это происходит, когда новая параболическая дуга добавляется к кривой как показано на рис 3.6 – 3.8.

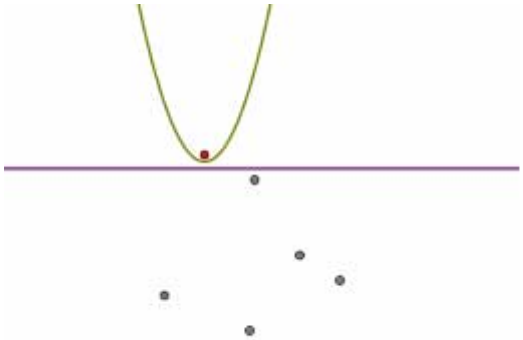


рис 3.6

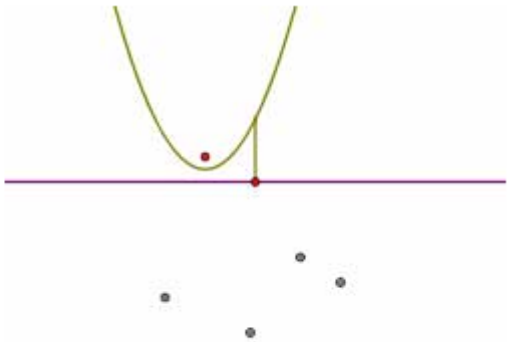


рис 3.7

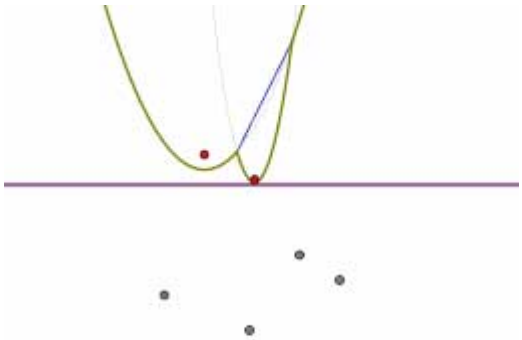


рис 3.8

Другими словами, пара граничных точек, соответствующих Грани мозаики Вороного появляются на параболической кривой в тот момент, когда Заметающая Прямая проходит через новый узел. Такое событие будем

называть Узловым Событием (Site Event).

1.3.3 Обозначение вершин

По мере движения заметающей прямой граничные точки двигаются вдоль грани мозаики до тех пор, пока не достигают вершины мозаики. Это происходит в тот момент, когда одна из параболических дуг исчезает. В следующей серии изображений рис 3.9 – 3.11 видно, как исчезает самая маленькая дуга.

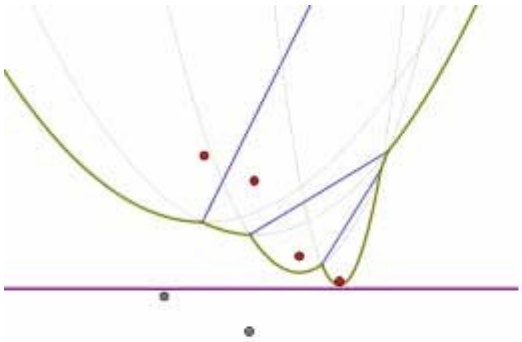


рис 3.9

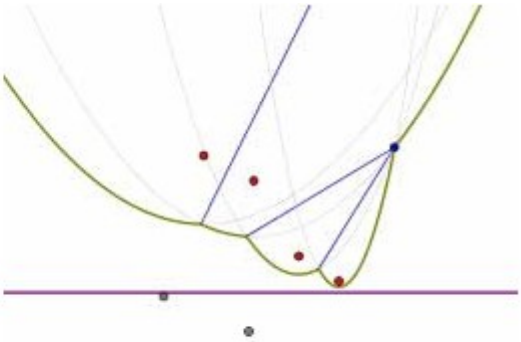


рис 3.10

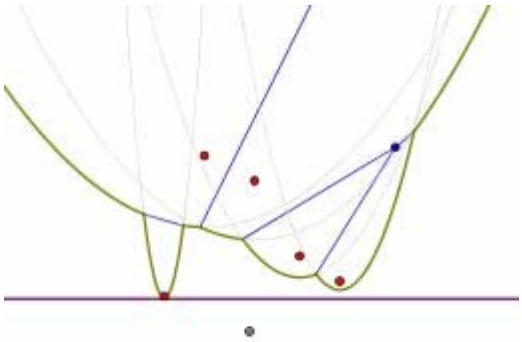


рис 3.11

Возникновение новой дуги на Параболической Кривой происходит в тот момент, когда Заметающая Прямая проходит через новый узел. Исчезновение дуги также нетрудно заметить. Когда параболическая дуга сужается в точку q, эта точка оказывается на трех параболах одновременно. Это означает, что q

равноудалена от трех узлов, соответствующих этим параболам, и эти три узла лежат на окружности с центром в точке q. Таким образом, вершина q определяется в тот момент, когда заметающая прямая проходит через нижнюю границу этой окружности, как показано на рис 3.12.

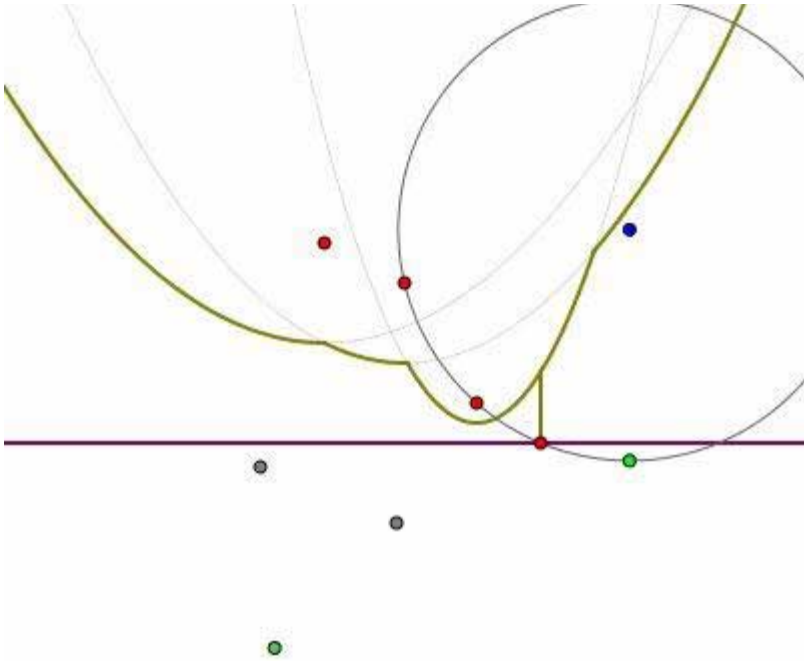


рис 3.12

Подобное явление будем называть Круговым Событием (Circle Event). Изменения в диаграмме происходят следующим образом рис 3.13 – 3.15

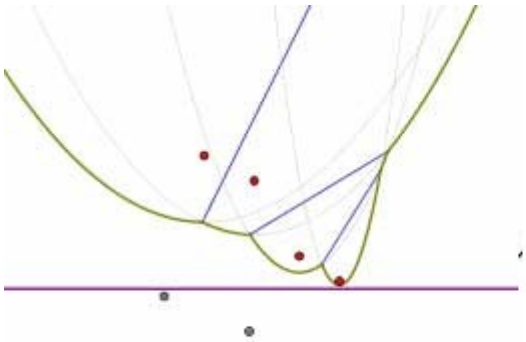


рис 3.13

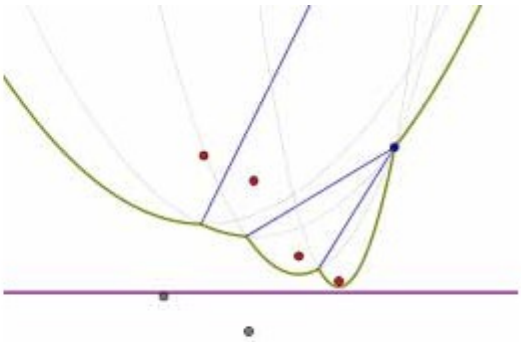


рис 3.14

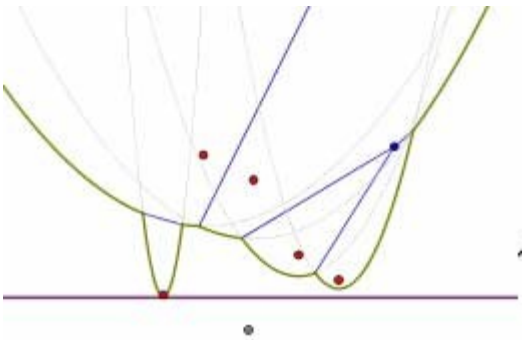


рис 3.15

1.3.4 Алгоритм

Таким образом, были определены все элементы, необходимые для описания Алгоритма Фортуна. Для обозначения граней и вершин диаграммы достаточно следить за появлением и исчезновением параболических дуг на параболической кривой. Для этого необходимо «пробегать» по параболической кривой слева направо и записывать порядок узлов диаграммы, соответствующих дугам. Известно, что порядок узлов не изменится до тех пор, пока Заметающая Прямая не спровоцирует Узловое Событие или Круговое Событие. Также, необходимо вести учет граничных точек посредством наблюдением за смежными дугами на параболической кривой.

Если следующее событие, спровоцированное Заметающей Прямой, будет Узловым Событием, мы попросту вносим новый узел в наш список узлов в том порядке, в котором соответствующая дуга появляется на параболической кривой. Таким образом, мы отмечаем новую грань на диаграмме.

Если следующее событие, спровоцированное Заметающей Прямой, будет Круговым Событием, мы отмечаем факт появления вершины в диаграмме, и то, что эта вершина является точкой смежности двух граней, которые соответствуют двум граничным точкам, которые сошлись. Мы также отмечаем новую грань, отвечающую новой граничной точке, которая является результатом кругового события.

Таким образом, диаграмма строится в результате конечной последовательности событий. Серия рисунков 3.16 – 3.29 иллюстрирует процесс вычисления диаграммы Вороного для заданного множества узлов. Круговое Событие обозначается зеленой точкой.

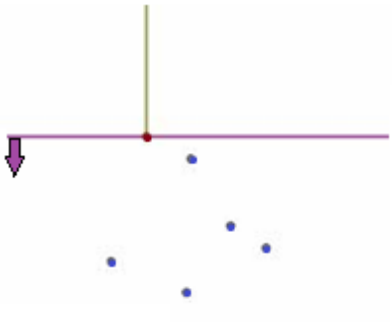


рис 3.16

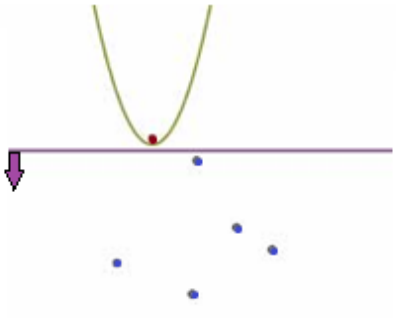


рис 3.17

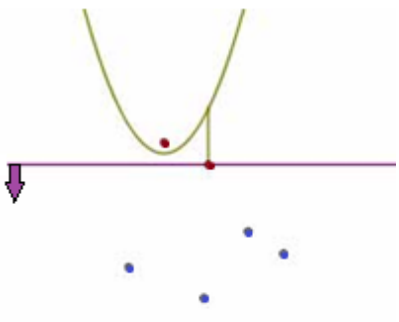


рис 3.18

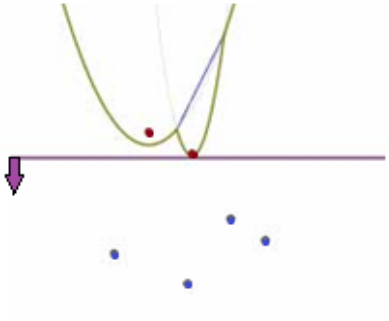


рис 3.19

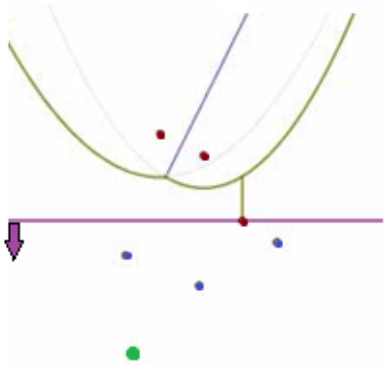


рис 3.20

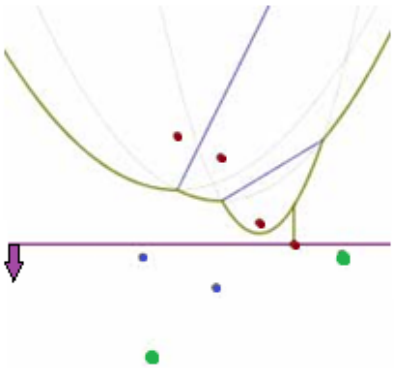


рис 3.21

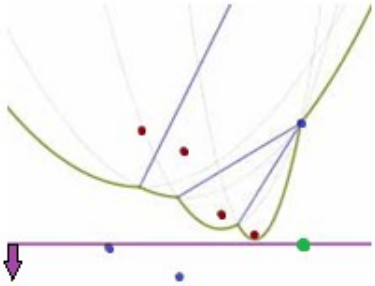


рис 3.22

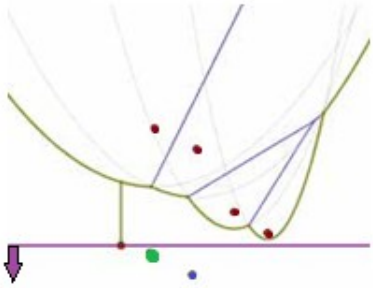


рис 3.23

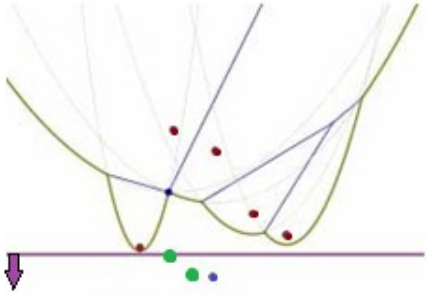


рис 3.24

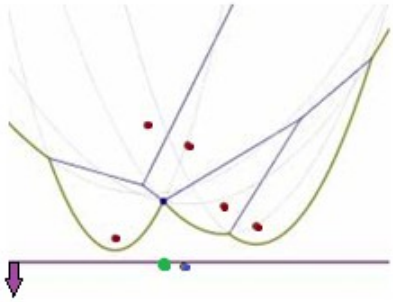


рис 3.25

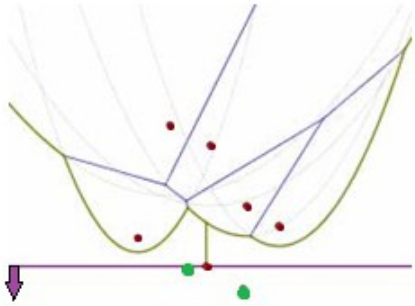


рис 3.26

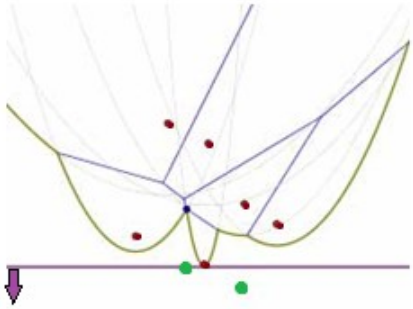


рис 3.27

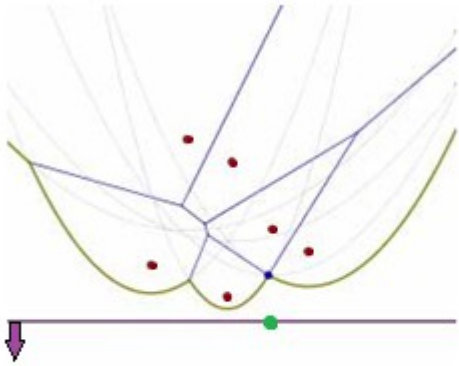


рис 3.28

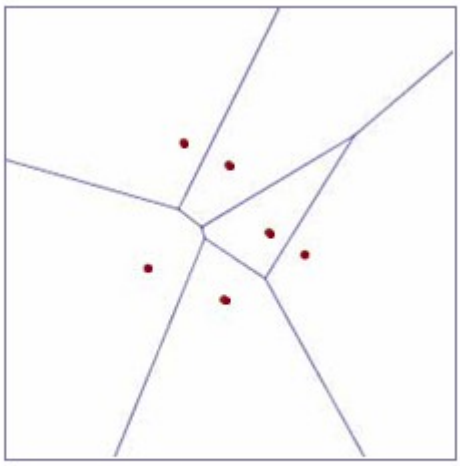


рис 3.29

На рис 3.29 изображен результат работы алгоритма Фортуна – диаграмма Вороного, из шести узлов.

1.4 Алгоритм Ллоида

Итак, с помощью алгоритма Фортуна можно генерировать диаграмму Вороного на основе множества случайных точек на плоскости. Рис 4.1

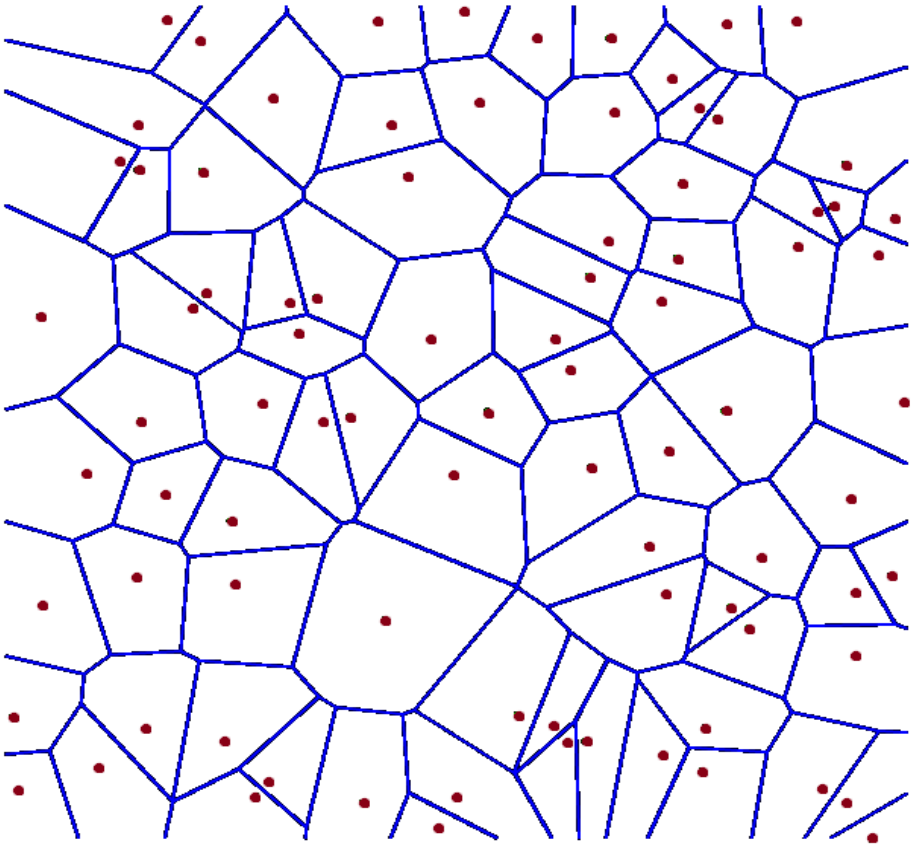


рис 4.1

Полученная диаграмма является довольно удобной структурой данных, на основе которой можно строить самые разнообразные изображения. Однако,

как нетрудно заметить на вышеприведенном рисунке, большинство плиток мозаики имеют неправильную форму. Это является естественным побочным эффектом генерации случайных точек и может сказаться на качестве построенного изображения.

Необходимо обеспечить более равномерное распределение точек, что в свою очередь приведет к более правильной форме полигонов. Для этой цели существует Алгоритм Ллоида, также известный как Итерации Вороного [1. 7].

Каждая итерация Алгоритма Ллоида состоит из трех шагов:

1. Построение Диаграммы Вороного из k узлов.
2. Вычисление центроида для каждого полигона мозаики.

   Центроидом полигона, состоящего из n вершин (x_0,y_0), (x_1,y_1), ..., $(x_{n-1},y_{n-1}$является точка (C_x, C_y), где:

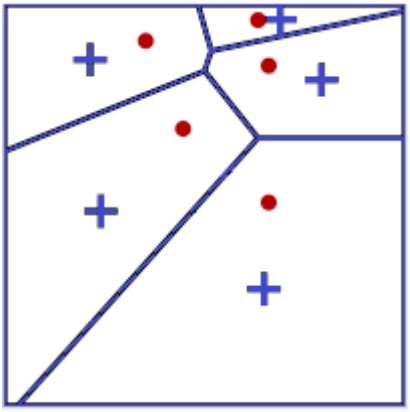
$$C_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i\ y_{i+1} - x_{i+1}\ y_i)$$

$$C_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i\ y_{i+1} - x_{i+1}\ y_i)$$

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i\ y_{i+1} - x_{i+1}\ y_i)$$

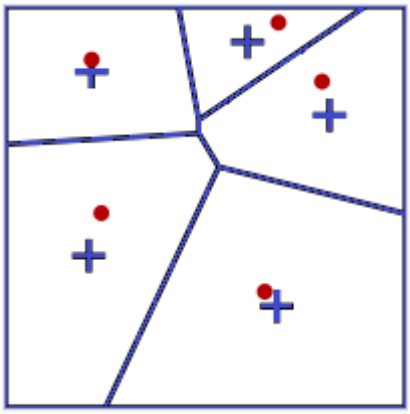
3. Смещение каждого узла диаграммы на место центроида соответствующего полигона

На нижеприведенных изображениях показано, как меняется диаграмма после 1,2,3, 15ти итераций. Рис 4.2 -4.5.



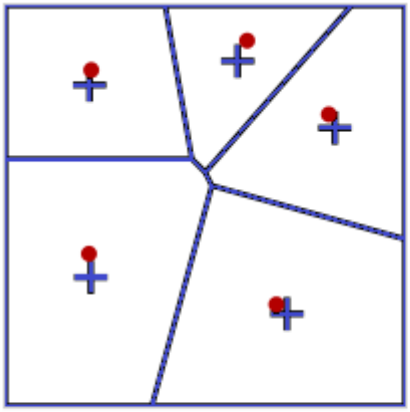
1я итерация

рис 4.2



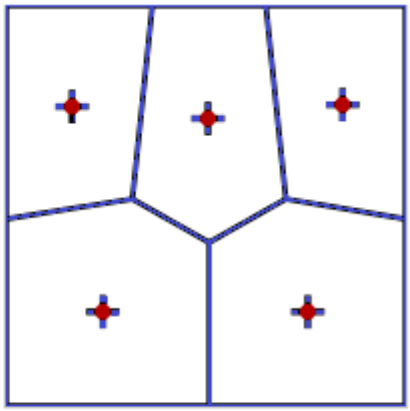
2я итерация

рис 4.3



3я итерация

рис 4.4



15я итерация

рис 4.5

Число необходимых итераций может варьироваться в зависимости от поставленных задач. Для данных целей достаточно двух. После выполнения алгоритма Ллоида, мозаика принимает более равномерный вид. Рис 4.6.

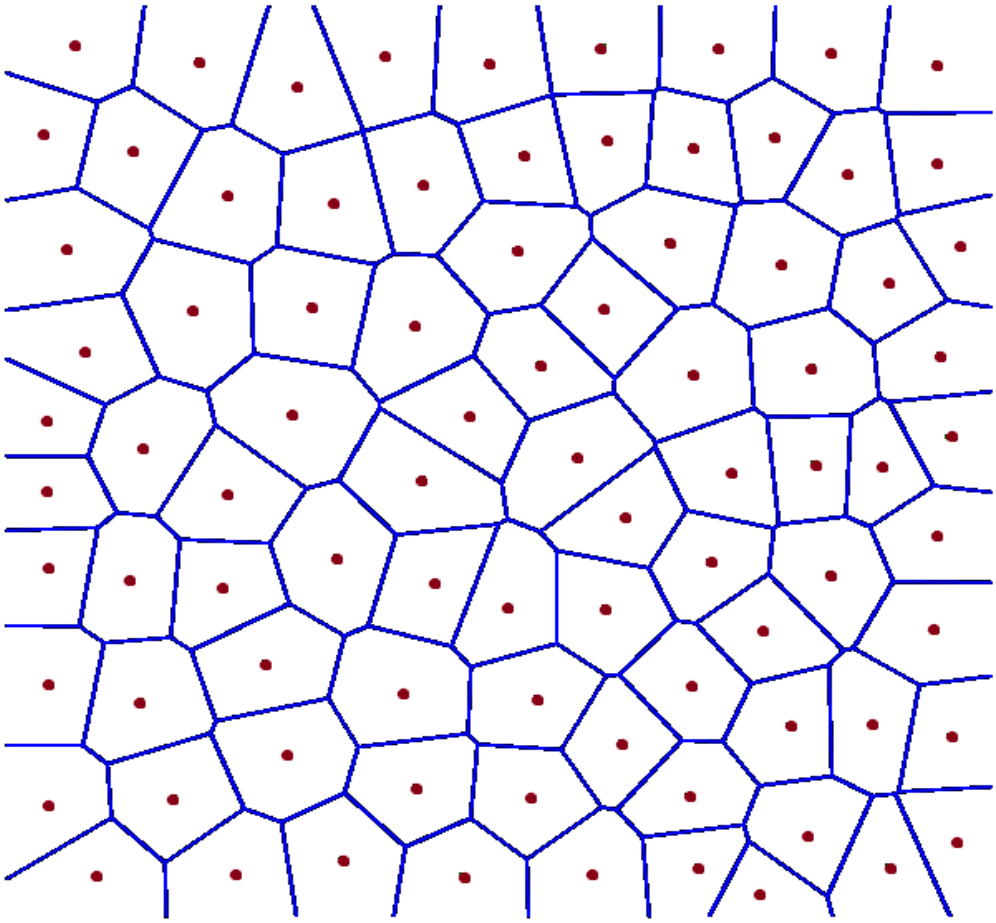


рис 4.6

1.5 Смежные Графы

Не трудно заметить, что построенная геометрическая конструкция порождает два взаимосвязанных графа: узлы и углы. Первый граф содержит узлы мозаики Вороного и линии, соединяющие каждый узел с узлами, соответствующими смежным полигонам. (Этот граф называется триангуляцией Делоне). Второй граф содержит углы каждого полигона, и линии, их соединяющие. Этот граф соответствует форме Мозаики.

Граф Делоне и граф Вороного взаимосвязаны. Каждый треугольник графа Делоне соответствует углу полигона из графа Вороного. Каждый полигон графа Вороного соответствует одному из углов треугольника графа Делоне. Каждая грань графа Делоне соответствует одной из граней графа Вороного.

Эта связь показана на рис 5.1.

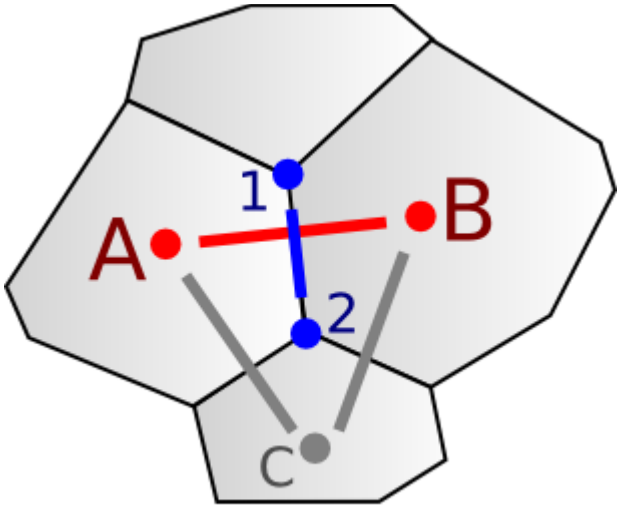


рис 5.1

Полигоны узлов A и B являются смежными, и в графе Делоне узлы A и B соединены линией, выделенной красным цветом. Синим цветом, выделена грань полигона, соединяющая угол 1 и угол 2 в графе Вороного. Таким образом, каждая грань из графа Делоне соответствует ровно одной грани в графе Вороного.

В графе Делоне, треугольник ABC соединяет три полигона, и соответствует углу 2 смежного графа. Таким образом, углы графа Делоне отвечают полигонам графа Вороного и наоборот. На рис 5.2 приведено изображение мозаики, со смежными графами. В графе Вороного, синим цветом, отмечены углы полигонов и белым цветом, линии, их соединяющие. В графе Делоне узлы мозаики отмечены красным цветом, и черным, линии, их соединяющие.

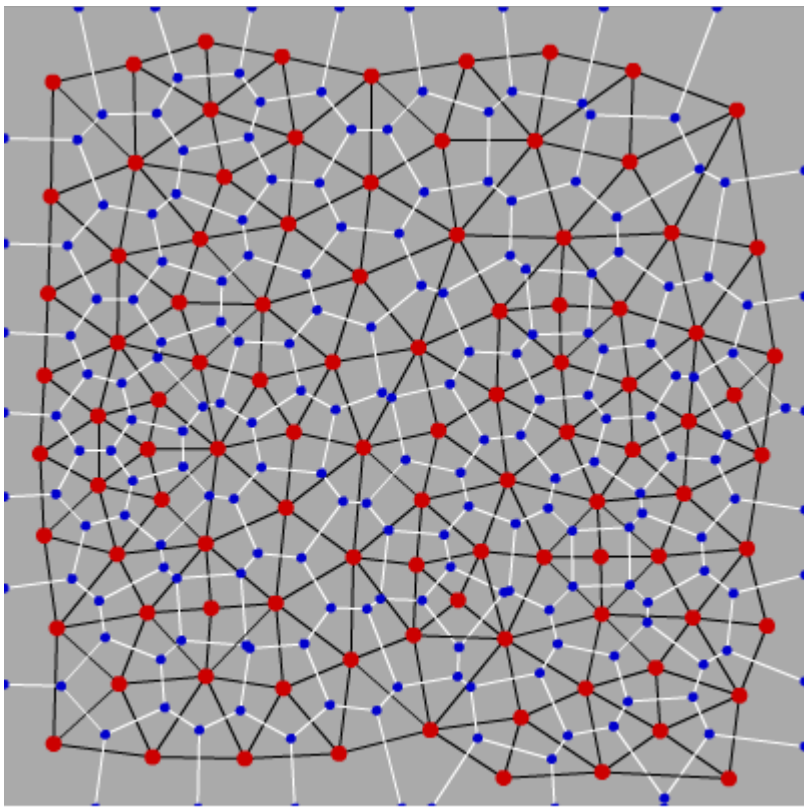


рис 5.2

Мозаика Вороного, в совокупности с порожденными ею смежными графами представляет собой структуру данных, которая идеально подходит в качестве основы для построения изображений методом процедурной генерации.

2. Заполнение Карты

2.1 Генерация Участка Суши

Теперь, когда была построена основная структура карты, дальнейшее построение изображения осуществляется путем «раскрашивания» мозаики, то

есть присвоение значения цвета разным частям основной геометрической структуры (полигоны, грани, углы). Этот процесс необходимо разбить на несколько этапов.

В первую очередь необходимо разделить карту на две области: вода и суша. В плане реализации, это осуществляется с помощью булевских функций. Пример подобного разделения можно увидеть на рис 2.1

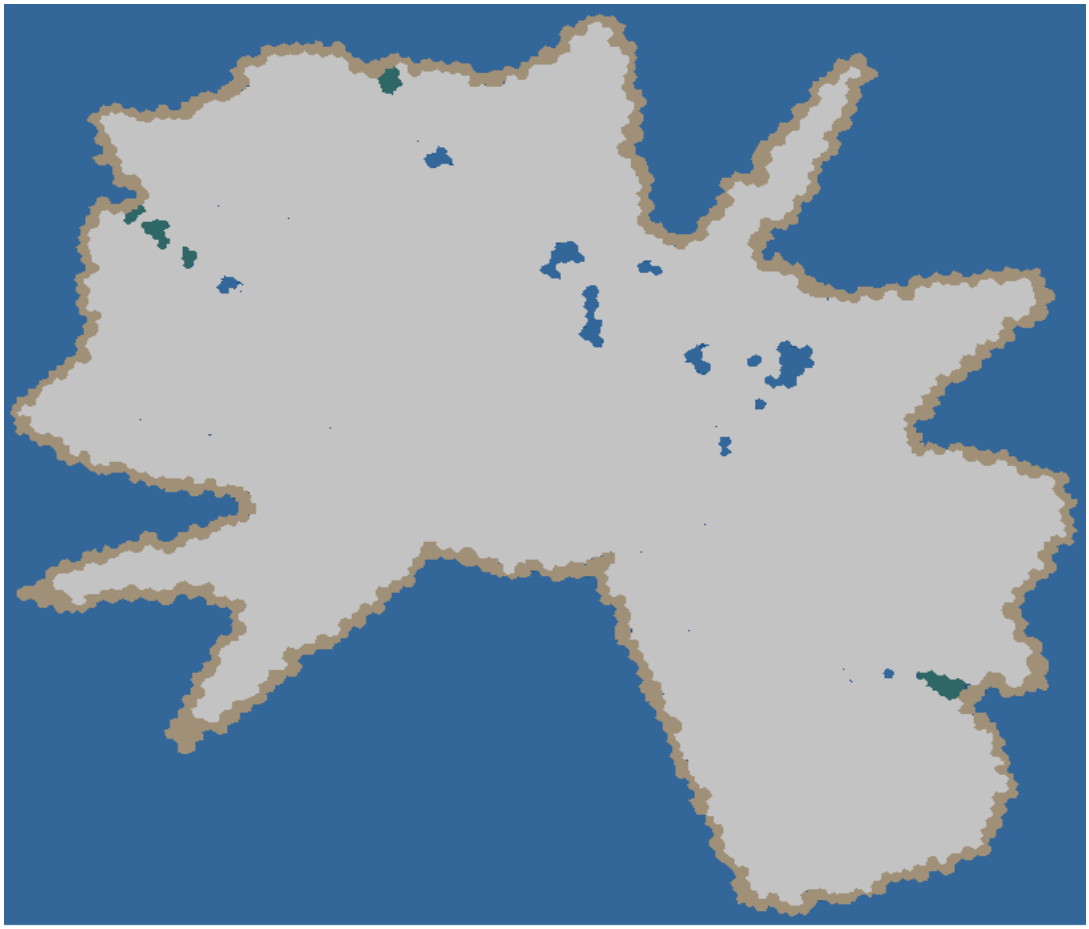


рис 2.1

В качестве типа местности была выбрана островная структура. Преимущество использования острова в качестве карты-уровня для компьютерной игры, заключается в том, что исключена возможность дойти до конца карты.

Прорисовка береговой линии осуществляется посредством генерации синусоидальных волн.

Дальнейшие процессы «раскрашивания» карты будут происходить, не выходя за пределы суши.

2.2 Генерация карты высот

Построение карты высот осуществляется путем присвоения каждому полигону числового значения высоты, которое впоследствии будет использовано для «раскраски». В качестве меры высоты в данной работе используется расстояние до берега: чем дальше от берега, тем больше значение. На рис 2.2

можно увидеть схематичное изображение подобной карты (чем светлее полигоны, тем они выше).

рис 2.2

В дальнейшем, цвет каждого полигона будет частично зависеть от его значения высоты. На более высоких участках карты будут находиться снег, скалы, тундра. На участках средней высоты – кустарники, пустыня, луга, леса. На самых низких участках – пляж, луга, тропический лес.

Также, карта высот будет играть важную роль в процессе генерации рек.

2.3 Генерация рек

Реки в данной работе будут зарождаться в горах и впадать в океан, стекая с более высоких участков к более низким. Чтобы это реализовать, необходимо вычислить наиболее оптимальный путь вниз для каждой точки диаграммы

Вороного.

Таким образом, чтобы нарисовать реку, нужно случайным образом выбрать точку, и следовать по уже вычисленному пути вниз (рис 2.3)

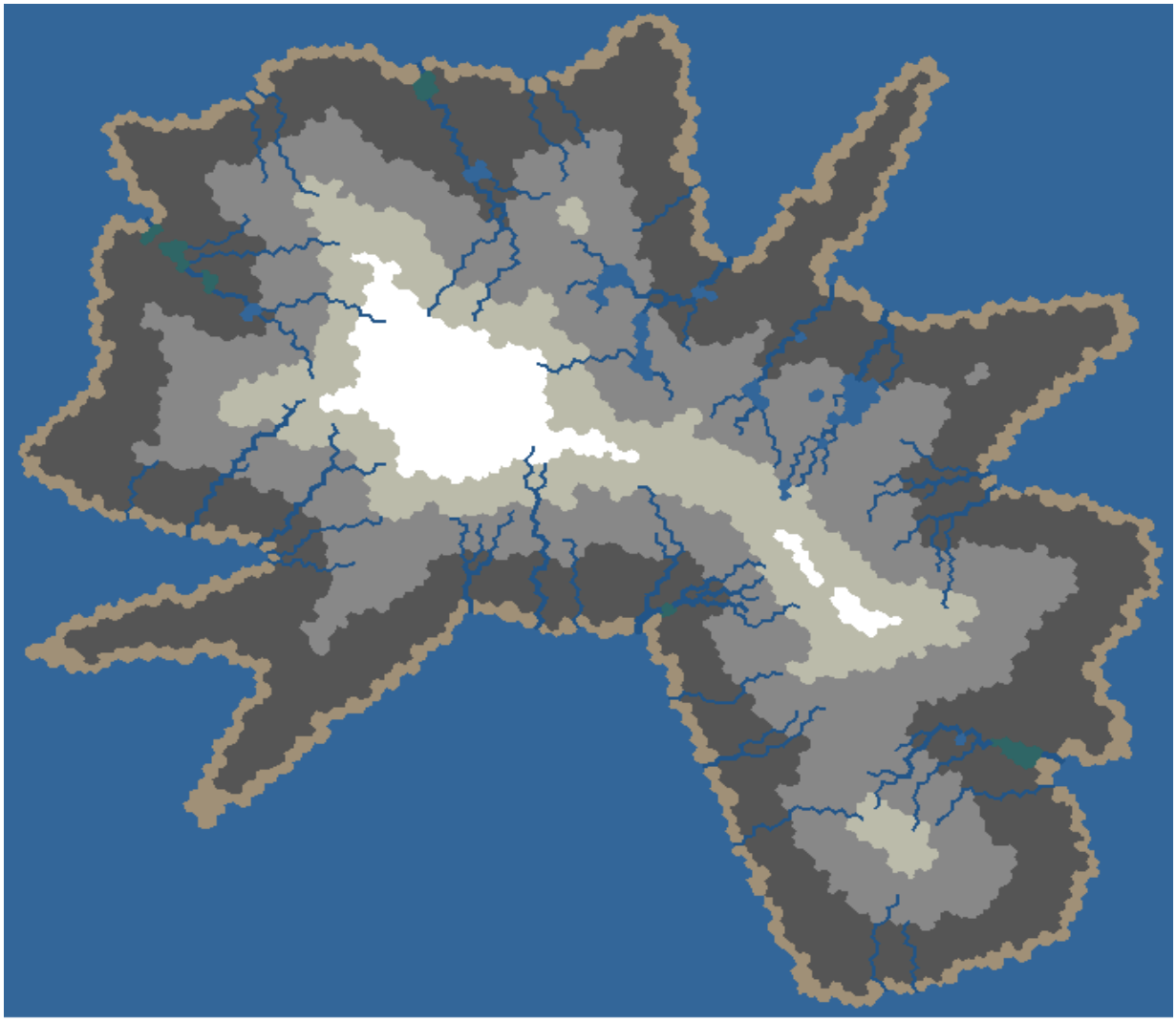


рис 2.3

2.4 Распределение Влажности

Карта Влажности строится тем же методом, что и карта Высот - путем присвоения значения влажности каждому полигону. Только мерой влажности в этом случае будет являться расстояние то источников пресной воды (случайно

сгенерированных рек и озер), чем дальше от воды, тем меньше влажность. Пример подобной карты изображен на рис 2.4

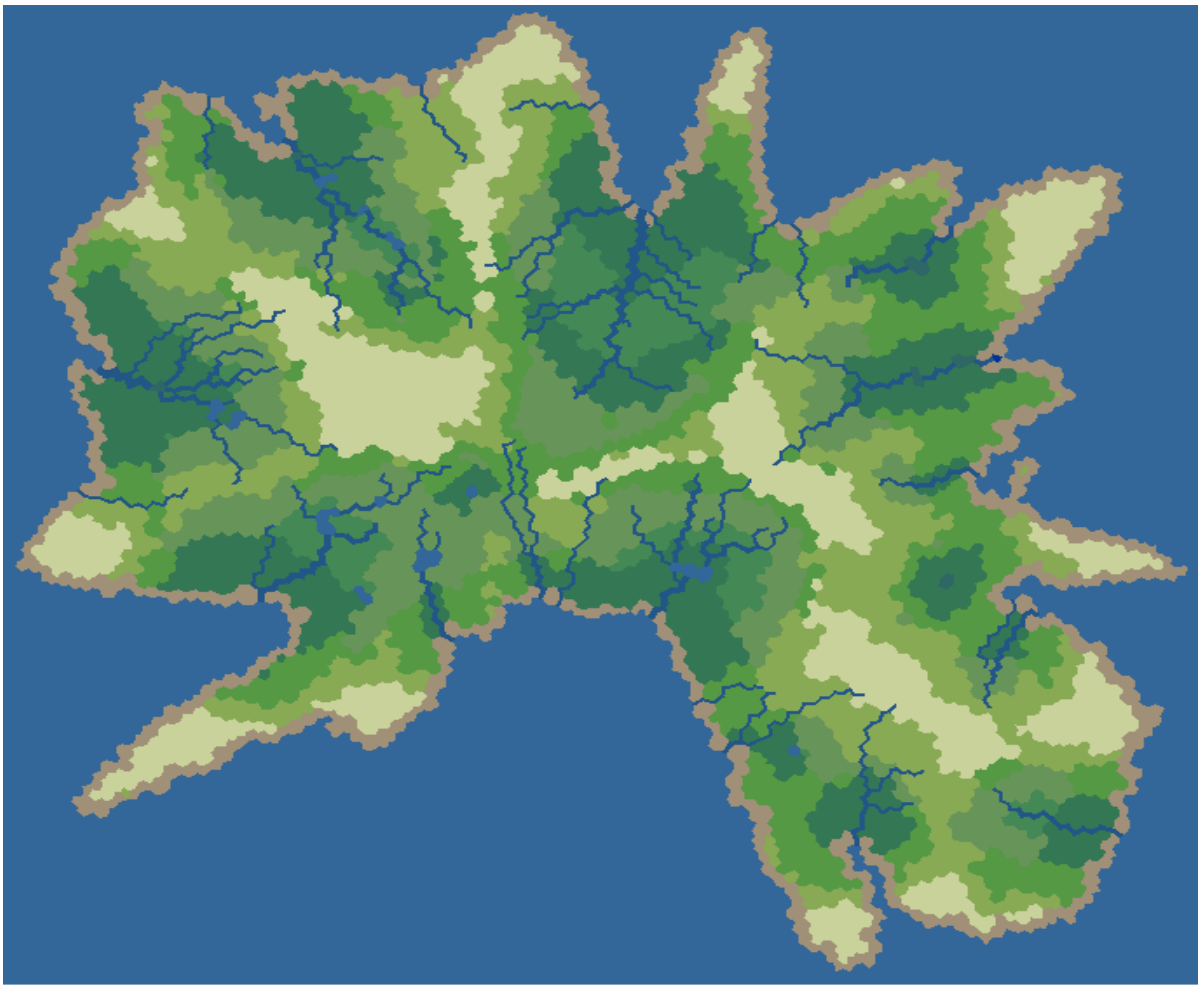


рис 2.4

На этой карте полигоны с меньшим значением влажности были закрашены в более светлый цвет.

2.5 Типы местности

Теперь после Построения карт Высоты и влажности, каждому полигону было присвоено значение высоты и значение влажности. В соответствии с этими значениями, каждому полигону присваивается значение цвета. Ниже приведена диаграмма «раскраски», используемая в данной программе:

<table>
<tr><th rowspan="2">Уровень Высоты</th><th colspan="6">Уровень Влажности</th></tr>
<tr><th>6 (макс.)</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1 (мин.)</th></tr>
<tr><th>4 (макс.)</th><td colspan="3">снег</td><td>тундра</td><td>пепел</td><td>вулкан</td></tr>
<tr><th>3</th><td colspan="2">тайга</td><td>Кустарная местность</td><td colspan="3">умеренная пустыня</td></tr>
<tr><th>2</th><td>умеренный тропический лес</td><td colspan="2">умеренный лиственный лес</td><td colspan="2">луг</td><td>умеренная пустыня</td></tr>
<tr><th>1 (мин.)</th><td colspan="2">тропический лес</td><td colspan="2">лес</td><td>луг</td><td>субтропическая пустыня</td></tr>
</table>

«Раскраска» является окончательным этапом заполнения карты. После него изображение принимает следующий вид (рис 2.5):

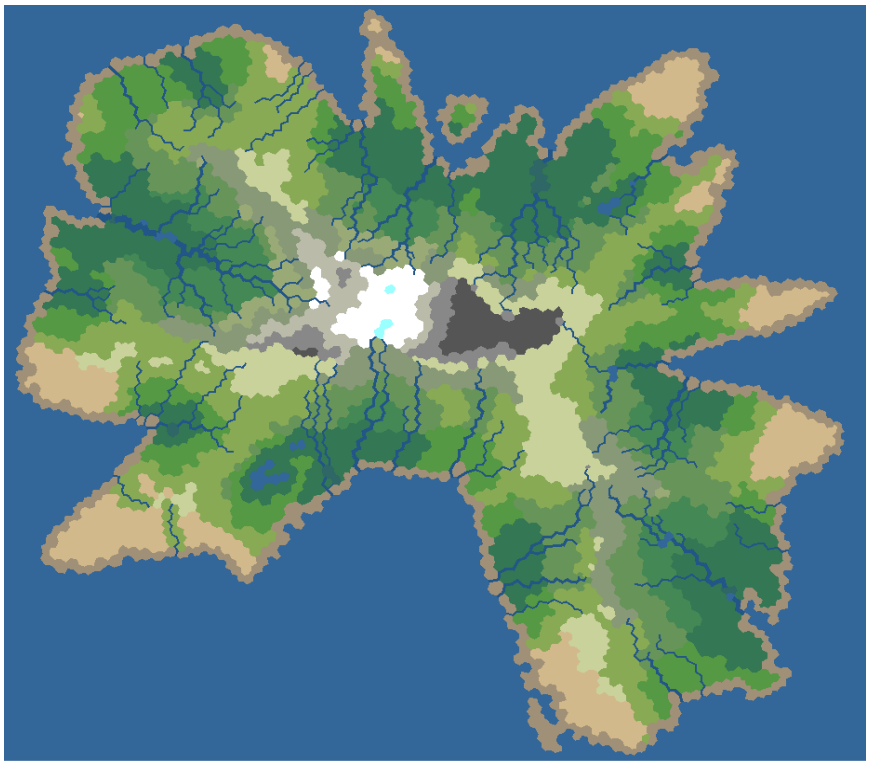


рис 2.5

3. Реализации Программы

3.1 Краткое описание

Для разработки программы использовался объектно-ориентированный язык

Java. Выбор данного языка обусловлен тем, что полученный генератор карт можно будет в дальнейшем интегрировать в java-игру для мобильных устройств. В качестве среды разработки была выбрана IDE Net Beans, так как она является наиболее знакомой и удобной для меня.

Основными источниками информации о языке программирования Java послужили: книга “Философия Java ”[1. 2], официальная документация Java [2. 4] и обучающий веб-ресурс Java-Course [2. 5]. Веб-сайты использовались наряду с книгой из-за превосходящего удобства поиска конкретной информации, многочисленных полезных советов в пользовательских комментариях к официальной документации и доступного изложения информации, предоставленного на сайте Java-Course.

Программа состоит из трех пакетов, содержащих 9 классов и одной вспомогательной библиотеки, переведенной с языка Action Script содержащей необходимые инструменты для реализации алгоритма Фортуна.

3.2 Классы и методы

Импортированная библиотека as3delauney [2. 6] содержит классы и методы, имплементирующие алгоритм Фортуна.

Класс Voronoi является ее основным классом. Объектом, описанным в этом классе, является основная структура диаграммы Вороного, построенная с помощью алгоритма Фортуна. Но методы этого класса, не подходят для целей данного проекта и являются неудобными для использования, поэтому они были модифицированы, чтобы соответствовать требованиям данной программы.

Библиотека as3delauney находится под лицензией MIT [2. 7], что предполагает ее свободное использование, модификацию, копирование, публикацию и распространение.

Классы Point, Center, Corner, Edge, Polygon описывают объекты типа «Точка», «Узел», «Угол», «Грань», «Полигон», которые составляют базовую геометрию данной программы.

Класс MosaicGraph является ключевым в данной работе. Объектом, описанным в этом классе, является приведенная к равномерному виду диаграмма Вороного, представленная в виде графов узлов и граней, каждый элемент которой имеет числовое значение, соответствующее определенному цвету и необходимое для «раскраски» карты.

Основные Методы класса MosaicGraph:

- buildGraph(); - осуществляет вычисление диаграммы Вороного с помощью модификации методов класса Voronoi импортированной библиотеки as3delaunay, и считывает необходимую информацию.

- useLloid(); - осуществляет выравнивание диаграммы Вороного с помощью Алгоритм Ллойда

- assignCornerHeight(); - осуществляет присваивание значения уровня высоты каждому углу диаграммы

- assignWaterCoastLand(); - разбивает множество полигонов мозаики на область воды, суши и берега (берег распознается программой, как полигоны, граничащие с водой и с сушей одновременно)

- redistribHeight(); - осуществляет перераспределение значения уровня высоты каждого угла диаграммы, таким образом, чтобы «низких» углов было больше, чем «высоких»

- assignPolyHeight(); - осуществляет присваивание значения уровня высоты каждому полигону, находящемуся на участке суши (вычисляется как среднее значение высоты принадлежащих этому полигону углов)

- calculateWayDown(); - осуществляет вычисление пути к океану от каждого угла диаграммы

- makeRivers(); - осуществляет прорисовку рек по граням диаграммы от случайно выбранных углов диаграммы с использованием вычисленного предыдущим методом пути до океана
- assignCornerMoist(); - осуществляет присваивание значения уровня влажности каждому углу диаграммы, находящемуся вблизи рек и озер

- redistribMoist(); - осуществляет присваивание значения уровня влажности каждому углу диаграммы, кроме тех, которые находятся

вблизи рек и озер

- assignPolyMoist(); - осуществляет присваивание значения уровня влажности каждому полигону, находящемуся на участке суши (значение высоты полигона, вычисляется как среднее значение влажности принадлежащих ему углов)

Класс ColoringGraph является наследником класса MosaicGraph.
Его методы осуществляют «раскраску» карты, посредством присваивания значения цвета каждому полигону, исходя из диаграммы зависимости высоты и влажности.

Класс MapManager является управляющим классом программы, содержащим пусковой метод **main()**. Этот класс позволяет создать начальную диаграмму Вороного, модифицировать ее в граф граней и узлов, содержащий графическую информацию, «раскрасить» и вывести на печать полученную карту.

3.3 Результаты выполнения программы

Результатом выполнения программы является схематичное изображение карты острова, которое можно в дальнейшем использовать в качестве уровня в

компьютерной игре.

На рис 3.1 изображена карта, построенная на мозаике Вороного из 20000 полигонов.

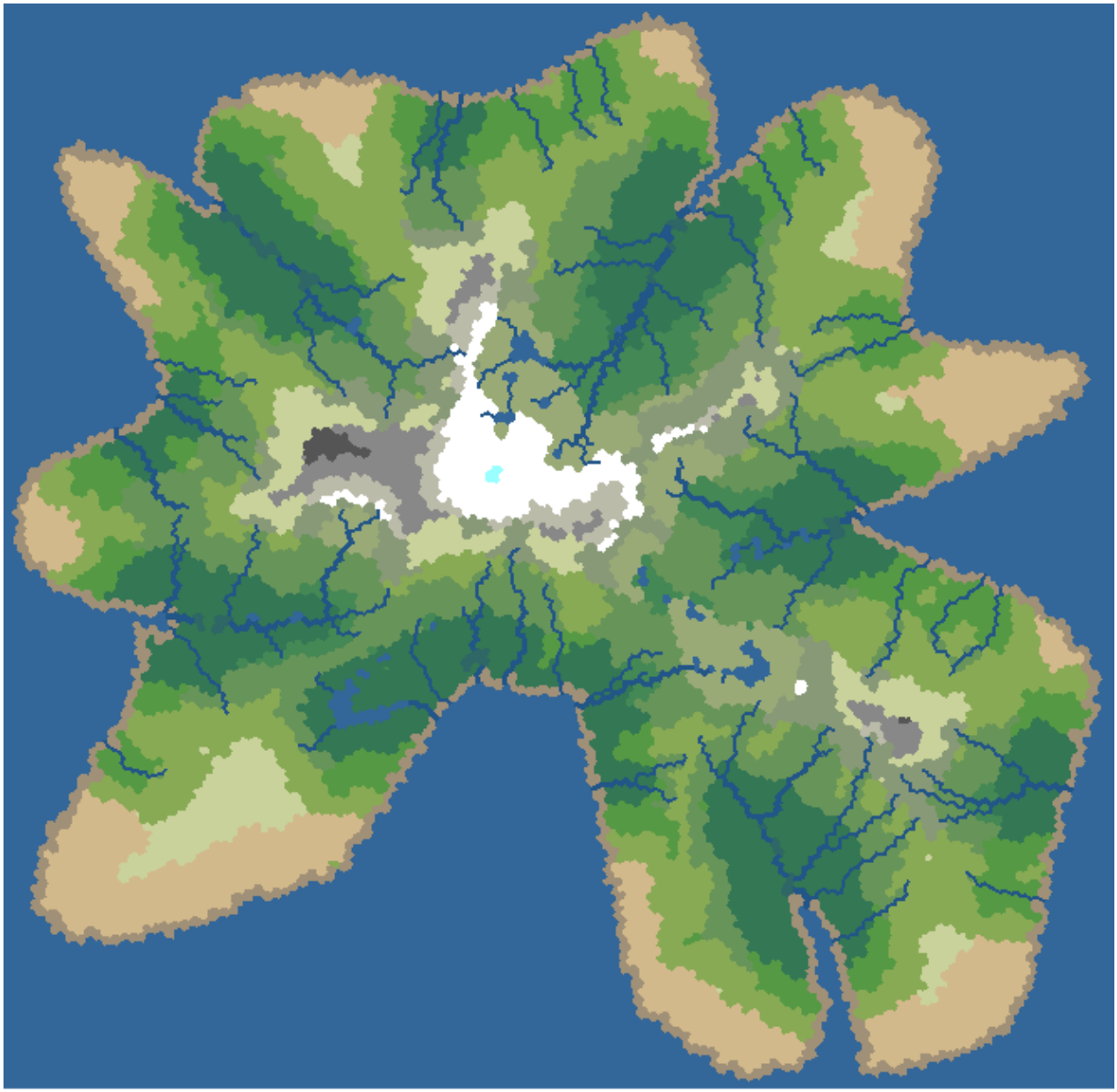


рис 3.1

На рис 3.2, 3.3 примеры карт, построенных на мозаике из 10000 полигонов.

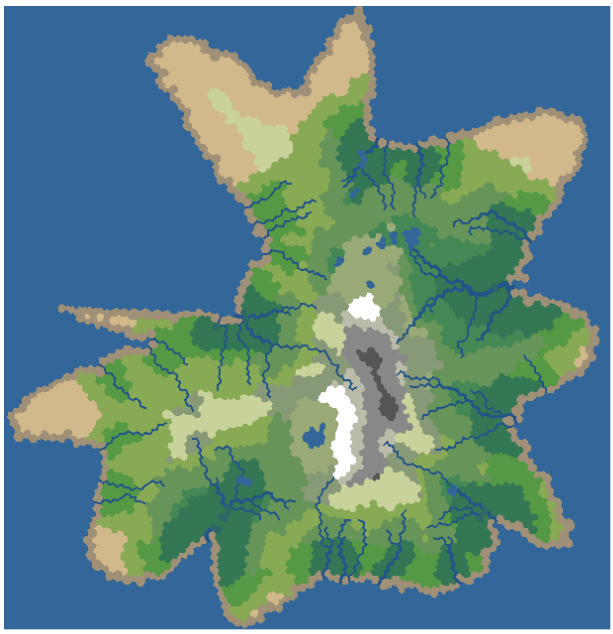


рис 3.2

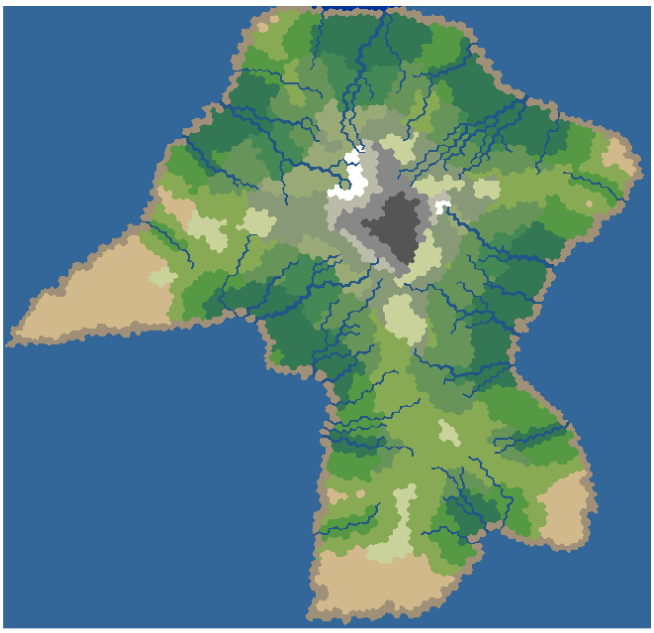


рис 3.3

На рис 3.4, 3.5 примеры карт, построенных на мозаике из 1000 полигонов. При малом количестве полигонов на изображении становится заметна плиточная структура карты.

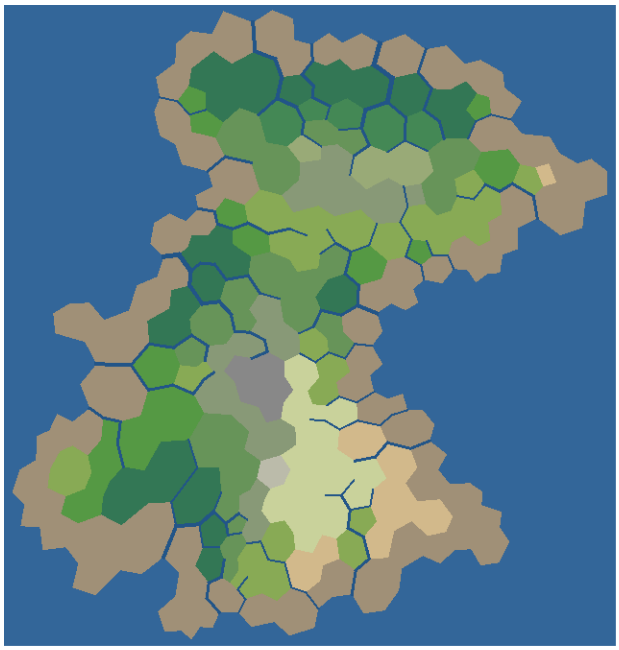


рис 3.4

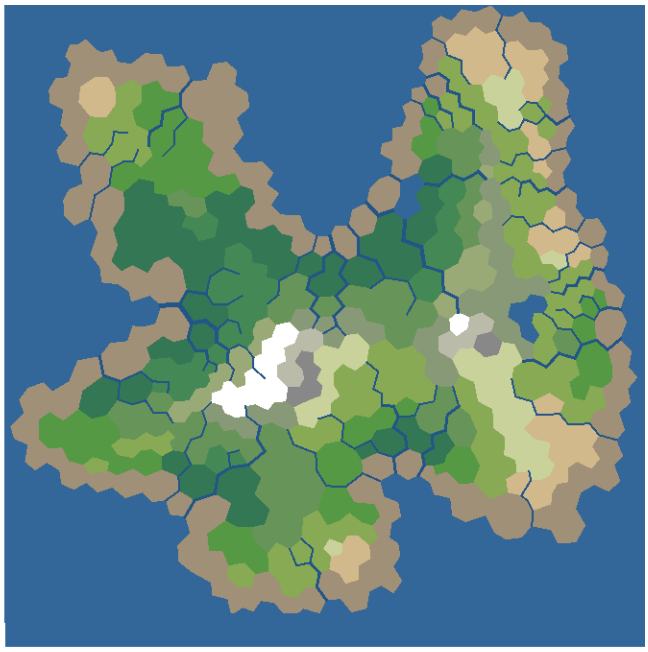


рис 3.5

Также, есть возможность построить изображение основной геометрической

структуры, состоящей из мозаики Вороного и смежных графов узлов и вершин. Пример такого изображения – на рис 3.6.

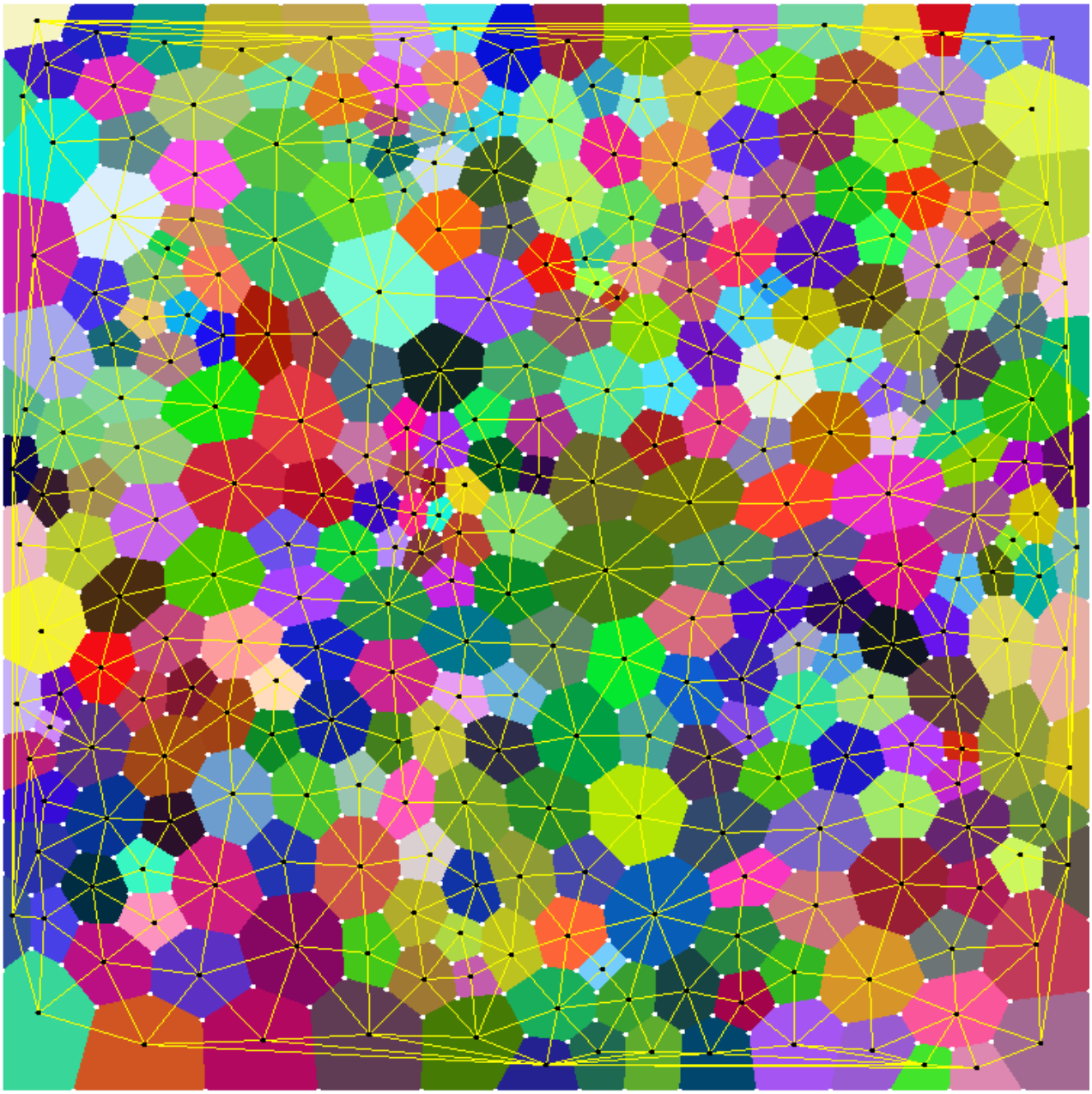


рис 3.6

На рис 3.7, совмещены изображение карты и основной структуры, на которой

она была построена.

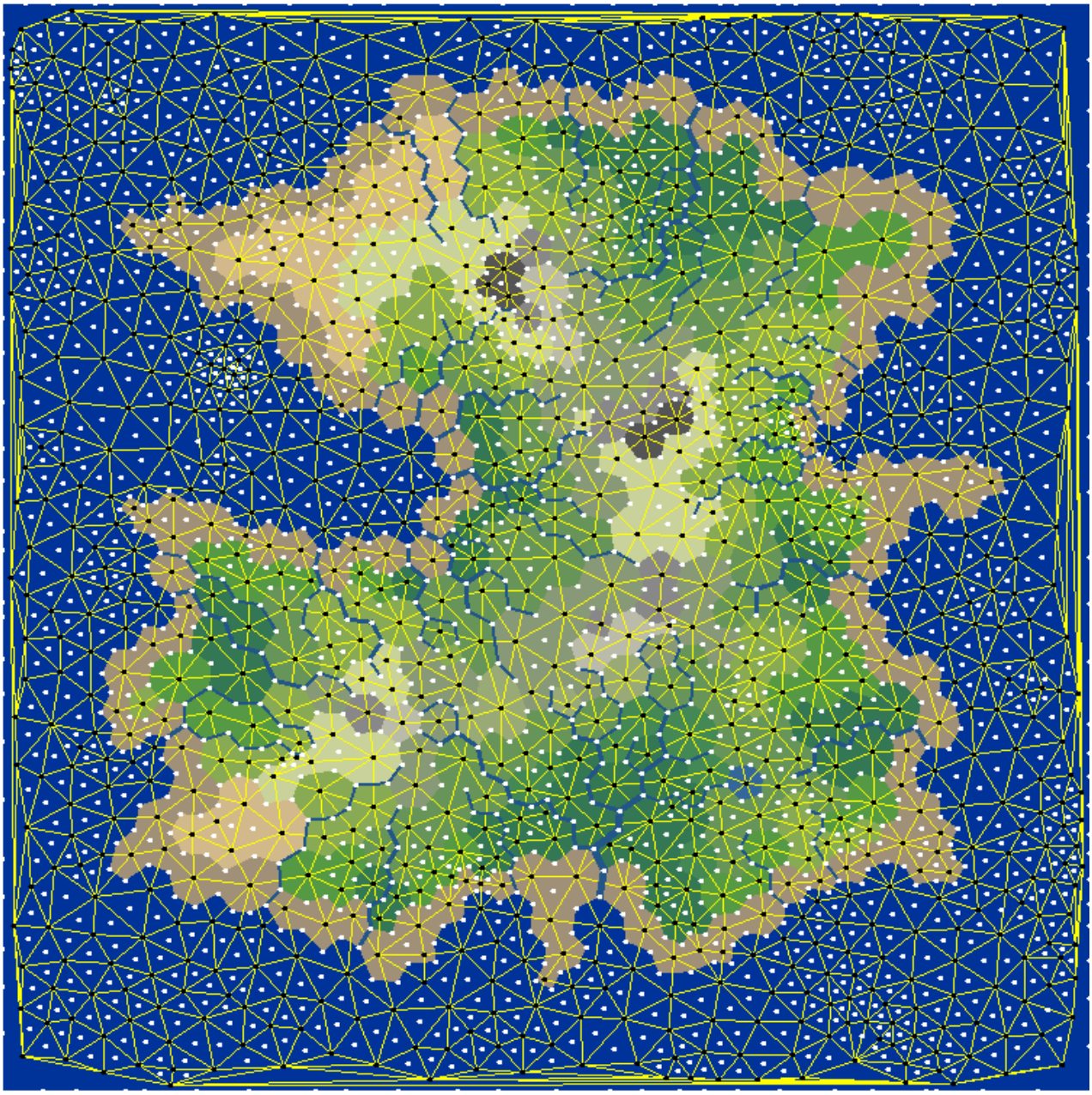


рис 3.7

Карты получаются достаточно разнообразными. Все алгоритмы выполняются корректно. Все элементы местности отображаются даже при малом количестве полигонов.

Заключение

В бакалаврской работе была исследована техника Процедурной Генерации Контента. Проанализированы варианты основной структуры данных для генератора карт местности, приведены различия рассмотренных вариантов, их плюсы и минусы. Превосходство выбранной, окончательной структуры данных, как более гибкой, универсальной и расширяемой, доказано. Подробно описаны алгоритмы ПГК, используемые для построения изображений. Приведен список классов и методов, описывающий реализацию данных алгоритмов. Приведены примеры изображений, получаемых в результате выполнения программы.

В результате выполнения бакалаврской работы разработана программа-генератор двухмерных карт местности, в основе которой лежит гибкая структура данных, обеспечивающая возможность ее дальнейшего расширения с последующим интегрированием в компьютерную игру для мобильных устройств.

В дальнейшем планируется добавить несколько вариантов типа местности, изображенной на карте, помимо островной (таких как материк или архипелаг), и возможность построения островных форм другими методами, помимо генерации синусоиды (например, с использованием функции Шума Перлина).

Список используемой Литературы

1. 1 David S. Ebert, Ken Perlin, Steven Woorley, Darwyn Peachey, F.Kenton Musgrave “Texturing & Modeling: A Procedural Approach”
1. 2 Брюс Эккель “Философия Java”
1. 3 Steve Oudot “Delaunay Triangulation”
1. 4 В.В. Галицкий, Е.В. Мироненко “Мозаика Вороного на плоскости”
1. 5 M. de Berg, M. van Kreveld, M. Overmas, O. Schwarzkopf “Computational Geometry”
1. 6 William L. Raffe, Fabio Zambetta, and Xiaodong Li “A Survey of Procedural Terrain Generation Techniques using Evolutionary Algorithms”
1. 7 Du, Qiang, Faber, Vance; Gunzburger, Max (1999), "Centroidal Voronoi tessellations: applications and algorithms"

Интернет ресурсы:

2. 1 https://habrahabr.ru/company/unigine/blog/167075/ - статья, посвященная процедурной генерации текстур
2. 2 http://pcg.wikidot.com/pcg-algorithm:map-generation - краткий обзор алгоритмов ПГК
2. 3 http://www.stuffwithstuff.com/robot-frog/3d/hills/index.html - пособие по генерации карт высот.
2. 4 http://www.oracle.com/technetwork/java/javase/documentation/index.html - документация Java
2. 5 http://java-course.ru/begin/introduce/ - пособия по java – программированию
2. 6 https://github.com/nodename/as3delaunay - библиотека, имплементирующая алгоритм Фортуна
2. 7 https://opensource.org/licenses/MIT - лицензия MIT