

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)



КУРСОВОЙ ПРОЕКТ ПО КУРСУ
«МЕТОДЫ И СРЕДСТВА ТЕХНОЛОГИЙ МУЛЬТИМЕДИА»

Процедурная генерация ландшафтов в среде Matlab

Студент:
Колесников Е. В.

Преподаватель:
Крапивенко А. В.

25 декабря 2015 г.

Содержание

1 Введение	2
1.1 Общая информация	2
1.2 Постановка задачи	4
2 Построение карты высот	4
2.1 Метод диаграммы Вороного	4
2.1.1 Диаграмма Вороного	4
2.1.2 Алгоритм релаксации Ллойда	5
2.2 Метод холмов	7
2.3 Алгоритмы Midpoint displacement и Diamond-Square	7
2.3.1 Описание алгоритма	7
2.3.2 Описание исходного кода	9
2.3.3 Исходный код	9
2.3.4 Пример работы программы	12
2.4 Фрактальный шум	14
2.4.1 Описание алгоритма	14
2.4.2 Исходный код	16
2.4.3 Пример работы программы	17
3 Сравнение рассмотренных алгоритмов	17
3.1 Анализ накладываемых сеток	17
3.2 Анализ алгоритмов построения карты высот	19
3.3 Подведение итогов	20

1 Введение

1.1 Общая информация

Природный ландшафт – территория, которая не подверглась изменению в результате хозяйственной и иной деятельности и характеризуется сочетанием определенных типов рельефа местности, почв, растительности, сформированных в единых климатических условиях. Шаги создания искусственного природного ландшафта:

- Создание карты высот.** Изначально имеется плоская двумерная сетка, каждой клетке которой в последствии присваивается некоторая высота. (В общем случае сетка не обязана быть прямоугольной, однако в подавляющем большинстве случаев удобнее использовать сетку из квадратных ячеек).
- Распределение биомов.** Биомы представляют собой отдельные зоны ландшафта с разным рельефом, растениями, животными и блоками, составляющими ландшафт. Определение различных зон ландшафта (тундра, пустыня, тропический лес, равнина,...) производится с помощью построенной карты высот. На (рис. 1) можно увидеть визуальное изображение различных биомов, в зависимости от температуры и влажности климата.

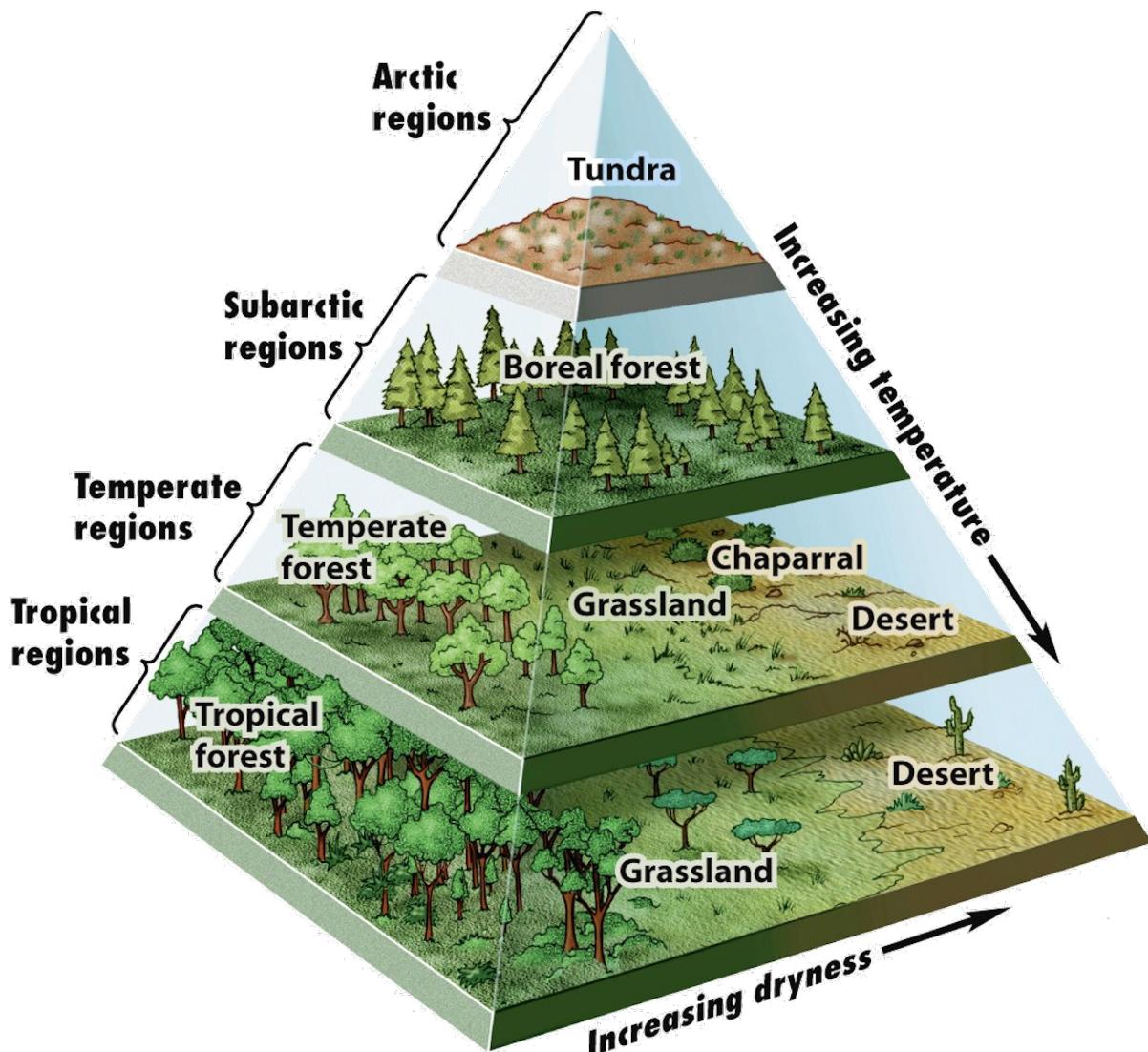


Рис. 1: Биомы

3. Создание рек.
4. Эмуляция поверхностей морей, океанов, озер и рек. В большинстве случаев нет необходимости реализовывать полноценную физику воды, достаточно просто создать карту высот водной поверхности (рис. 2) и изменять её со временем по какому-то закону.

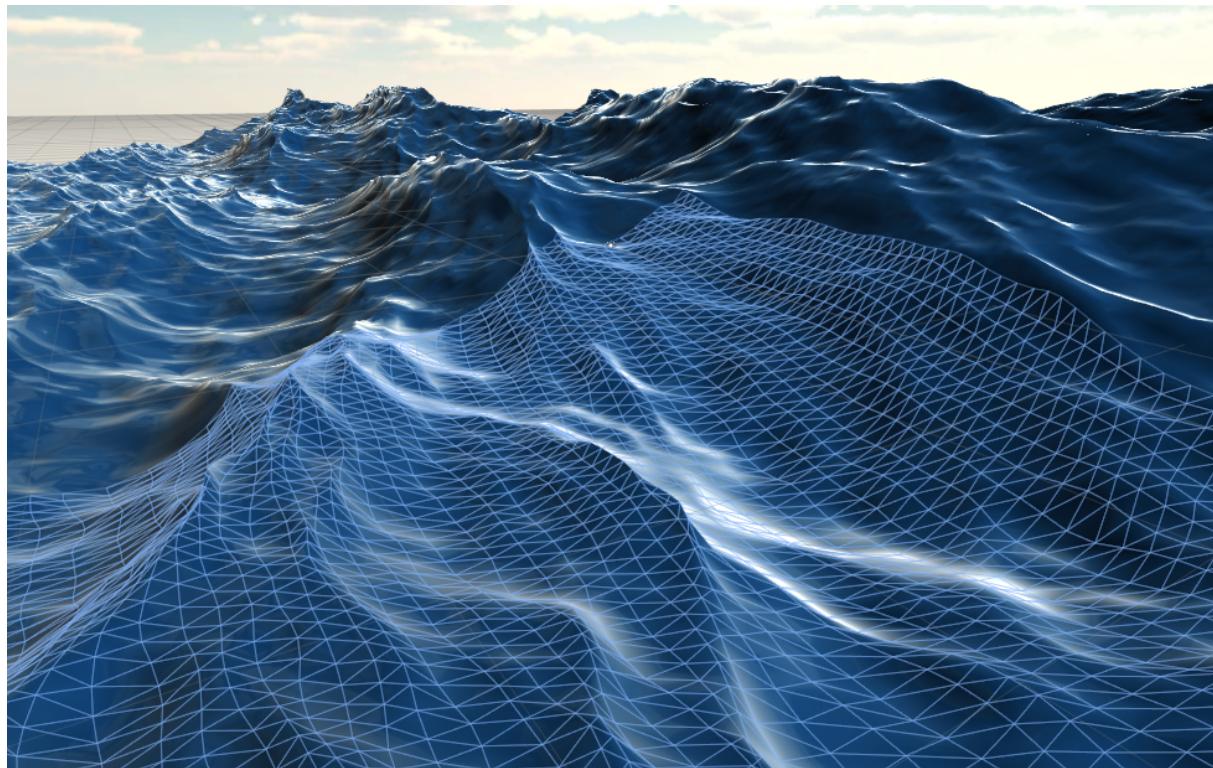


Рис. 2: Водная поверхность

5. Эрозия. Эрозия – эмуляция воздействия рек, озер (рис. 3), ветров и температур на поверхность земли: образование русла рек, перенос песка и мягкого грунта, образование пляжей на берегах.

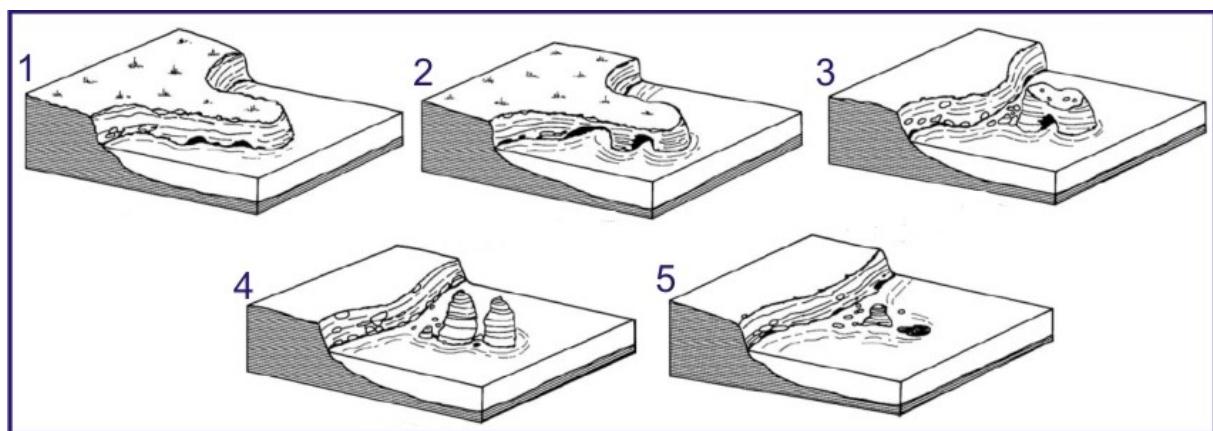


Рис. 3: Водная эрозия

6. Добавление флоры и фауны. В зависимости от биомов, ландшафт имеет различный травяной покров, количество каменистых участков, различные виды растений и обитающих животных.

Можно добавить еще много шагов, улучшающих ландшафт и приближающих его к реальному, таких как создание пещер, подземных вод, разработка подводного пространства и наоборот – воздушного.

1.2 Постановка задачи

В данной работе будут рассмотрены и запрограммированы в среде Matlab несколько алгоритмов процедурной генерации карт высот: *метод холмов*, *Midpoint displacement*, *Diamond-Square* и *алгоритм на основе шума Перлина*; алгоритм генерации неравномерной сетки: *метод диаграммы Вороного*, а также будет подробно рассмотрен алгоритм, позволяющий эмулировать движения поверхности океана, основанный на преобразовании Фурье с добавлением шума.

2 Построение карты высот

Построение карты высот – важнейший этап построения ландшафта, более подробно этот процесс представляет из себя определение того, на какой высоте находится каждая точка поверхности земли. На данном этапе речь идет о конструктировании нетекстуированной, неосвещенной трехмерной сетки (рис. 4).

Основная проблема заключается в том, чтобы создать сетку, которая бы напоминала земную поверхность.

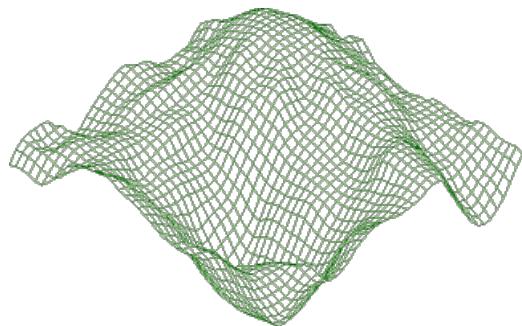


Рис. 4: Карта высот

2.1 Метод диаграммы Вороного

Во всех рассматриваемых в данной работе алгоритмах используется равномерная квадратная сетка с квадратными ячейками, однако с помощью метода диаграммы Вороного создают неравномерную, нерегулярную сетку. Ключевыми понятиями в данном разделе являются “диаграмма Вороного”, “триангуляция Делоне” и “реалаксация Ллойда”.

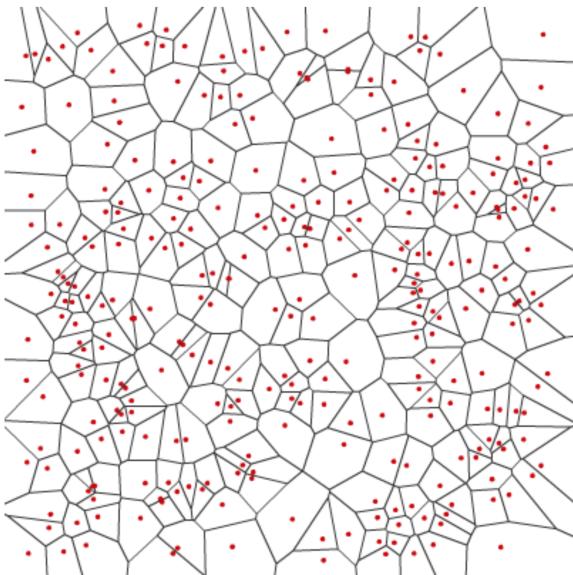
2.1.1 Диаграмма Вороного

Диаграмма Вороного конечного множества точек S на плоскости представляет такое разбиение плоскости, при котором каждая область этого разбиения образует множество точек, более близких к одному из элементов множества S , чем к любому другому элементу множества. Программно диаграмму Вороного можно построить с помощью алгоритма Форчуна.

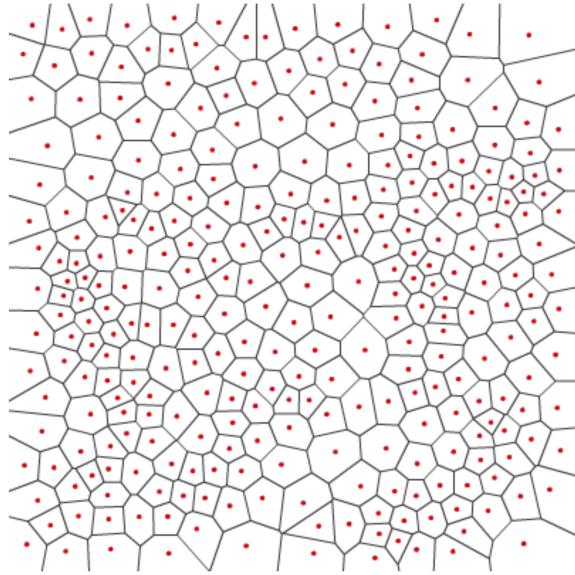
Алгоритм основан на применении заметающей прямой. Заметающая прямая – это вспомогательный объект, представляющий собой вертикальную прямую линию. На каждом шаге алгоритма диаграмма Вороного построена для множества, состоящего из заметающей прямой и точек слева от неё. При этом граница между областью Вороного, прямой и областями точек состоит из отрезков парабол (так как геометрическое место точек, равноудалённых от заданной точки и прямой – это парабола). Прямая движется слева направо. Каждый раз, когда она проходит через очередную точку, эта точка добавляется к уже построенному участку диаграммы. Добавление точки к диаграмме при использовании двоичного дерева поиска имеет сложность $O(\log n)$, всего точек n , а сортировка точек

по x -координате может быть выполнена за $O(n \log n)$, поэтому вычислительная сложность алгоритма Форчуна равна $O(n \log n)$.

Алгоритм начинается со случайного бросания точек на плоскость. Затем по этим точкам строится диаграмма Вороного (полигоны Вороного). Пример получающихся полиномов можно посмотреть на (рис. 5а).



(a) До релаксации Ллойда



(b) После релаксации Ллойда

Рис. 5: Диаграммы Вороного

В полученном результате есть несколько недостатков: форма и размер полигонов слишком нерегулярны. В данной задаче необходимо получить квазислучайные точки, которые бы равномерно располагались на плоскости. Достигнуть такого эффекта можно, применив к полученным полигонам релаксацию Ллойда (рис. 5б).

2.1.2 Алгоритм релаксации Ллойда

Алгоритм релаксации Ллойда – итеративный процесс, каждая итерация которого состоит из трех последовательных шагов:

1. Построение диаграммы Вороного для заданных точек (ядер).
2. Для каждого полинома диаграммы Вороного вычисляется центроид.
3. Ядро каждого полинома смещается в центроид соответствующего полинома.

Пример работы алгоритма релаксации Ллойда можно увидеть на (рис. 6). Плюсами обозначаются центроиды полиномов Вороного, порожденным соответствующими ядрами, которые обозначены красными точками.

После того, как полигоны более менее равномерно распределены, диаграмма Вороного разбивается на два графа (рис. 7б): граф ядер и ребер. У первого графа вершинами являются ядра, которые порождают полигоны Вороного, а ребра связывают ядра, порождаемые полигонами которых соприкасаются. Такое представление называется триангуляцией Делоне (рис. 7а) и используется в задачах связности, таких как построение маршрута. У второго графа вершинами являются углы полигонов, а ребрами – ребра полигонов.

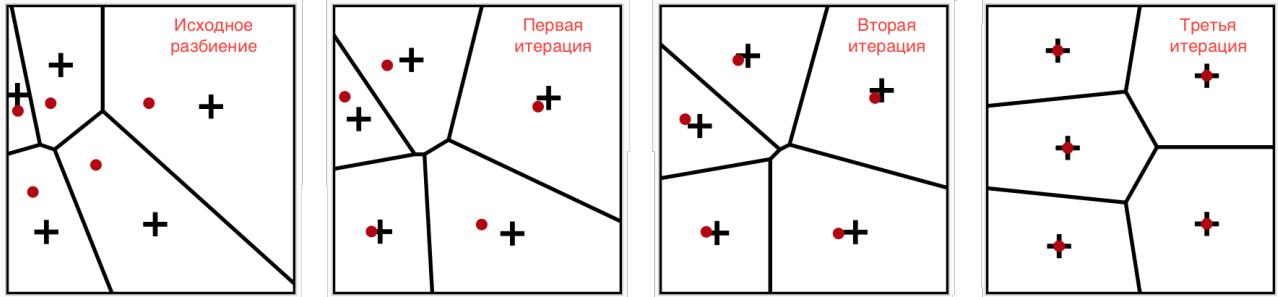


Рис. 6: Пример работы алгоритма релаксации Ллойда

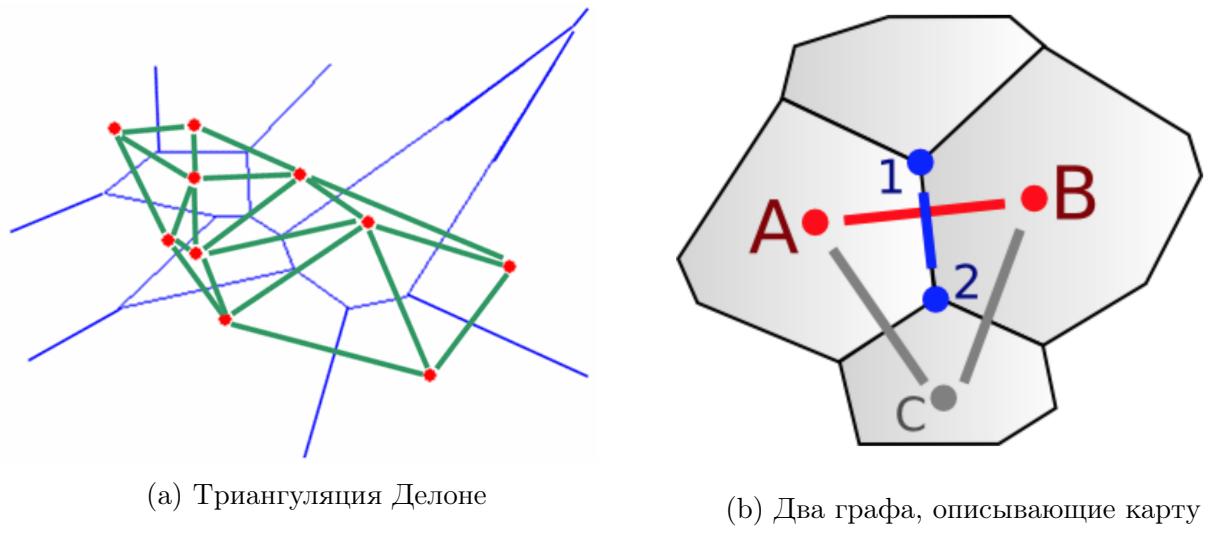


Рис. 7: Разбиение сетки на два графа

Два рассмотренных графа логически связаны между собой. Каждый треугольник в триангуляции Делоне соответствует одному углу полигона диаграммы Вороного. И соответственно каждый полигон диаграммы Вороного соответствует как минимум одному углу триангуляции Делоне. Каждое ребро в графе Делоне соответствует единственному ребру в графе Вороного. Рассмотрим (рис. 7b): полигоны A и B связаны друг с другом, поэтому красное ребро между вершинами A и B принадлежит графу Делоне. Синее ребро полигона, соединяющее углы 1 и 2 принадлежит графу Вороного.

Таким образом, треугольник $A - B - C$ в триангуляции Делоне, соединяющий три полигона, можно представить с помощью угла 2. А углы в триангуляции Делоне соответствуют полигонам диаграммы Вороного.

Во многих случаях следует избавиться от явно выделенной полигональной структуры сетки. Самым простым и в то же время единственным способом достичь этого можно, сместив углы полигонов случайным образом.

Вершины A , 1, B , 2 образуют четырехугольник, который в свою очередь можно раз-

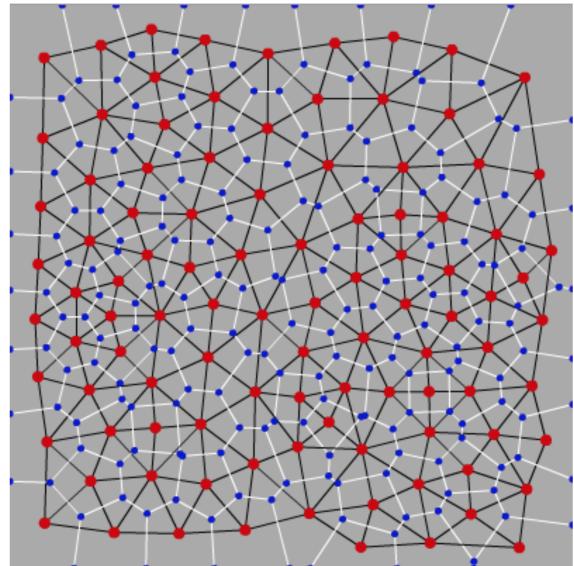
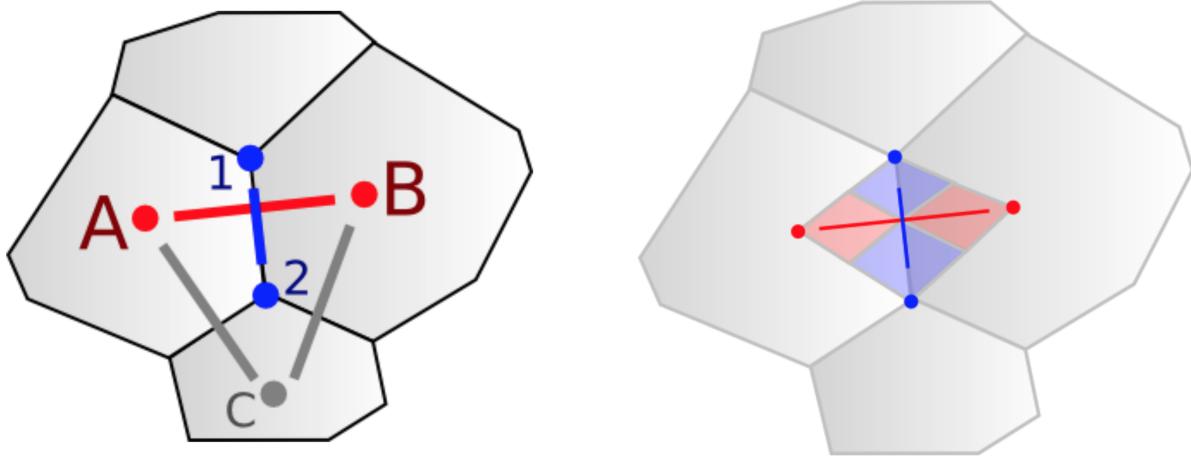


Рис. 8: Пример полученной сетки, состоящей из двух графов

вить.



(a) Исходный вариант

(b) Области смещений вершин

Рис. 9: Дуальные графы

бить на четыре четырехугольника меньшего размера, каждый из которых порожден своей вершиной (рис. 9b). Каждый из маленьких четырехугольников – областя возможного смещения соответствующей вершины. Однако для того, чтобы не появлялось новых вершин, необходимо смещать уже существующие вдоль перпендикулярных линий, показанных на (рис. 9b).

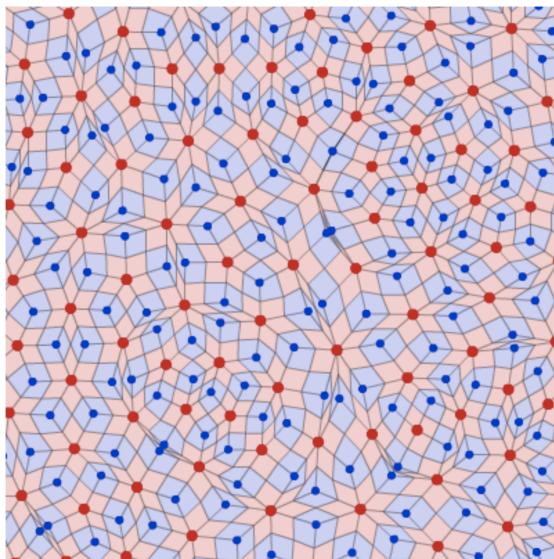


Рис. 10: Разбиение карты на области смещений

алгоритм не является процедурным и реализуется “вручную”.

Таким образом всю карту можно разбить на четырехугольники, являющиеся областями смещений. Смещаая каждую точку вдоль одного из направлений, получают нерегулярную сетку.

2.2 Метод холмов

Простым и достаточно эффективным методом создания природного ландшафта, дающего приемлемые результаты – метод холмов. Данный метод заключается в последовательном добавлении “выпукостей” в произвольных местах – холмов различной высоты. В результате аккуратного наложения этих холмов друг на друга и, возможно, добавление небольшого шума, можно получить более менее реалистичный ландшафт. Следует отдельно заметить, что данный алгоритм не является процедурным и реализуется “вручную”.

2.3 Алгоритмы Midpoint displacement и Diamond-Square

2.3.1 Описание алгоритма

Алгоритмы Midpoint displacement и Diamond-Square рассматриваются вместе, т.к. являются, по сути, одним и тем же алгоритмом с единственным отличием, которое будет отдельно оговорено.

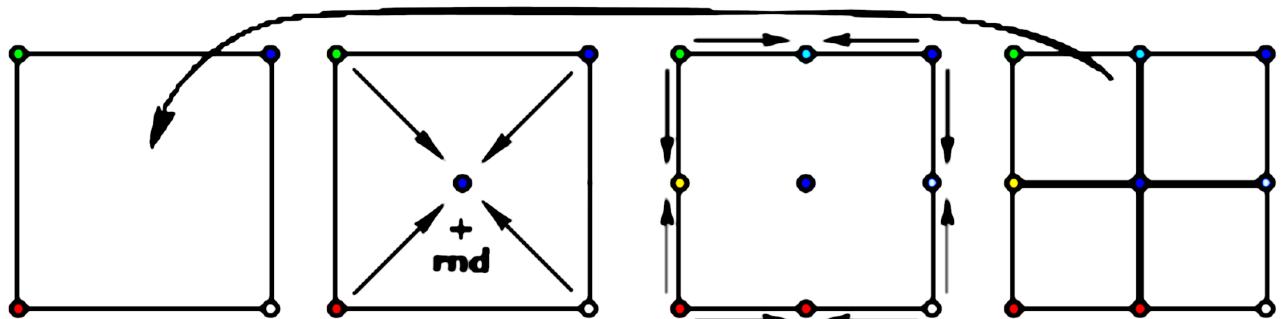
Diamond-Square – самый распространенный алгоритм генерации ландшафтов, а также дающий один из самых реалистичных результатов. Ландшафты, получающиеся с его помощью, называют фрактальными, хотя на самом деле их следует называть стохастико-фрактальными, т.к. на каждой итерации добавляется шум, не допускающий самоподобия.

То, что роднит оба эти алгоритма с фракталами – их рекурсивное поведение. Изначально четырем углам всей карты высот присваиваются случайные величины. Для удобства далее будем работать с квадратной картой, длина стороны которой является степенью двойки.

После того, как заданы границы квадрата, разобьем его на четыре равных квадрата, в каждом из которых известно значение одного из углов. Значение высоты в центральной точке – сумма усреднения высот всех четырех угловых точек исходной карты высот и случайного значения (шума). На практике шум должен быть не полностью случайным, а функцией, зависящей от расстояния между соседними точками, т.к. чем меньше расстояние, тем меньше должно быть в среднем значение шума.

После того, как определена высота центральной точки, необходимо определить высоты на гранях квадрата. На этом шаге главное и единственное различие между алгоритмами Midpoint displacement и Diamond-Square. В алгоритме Midpoint displacement высоты на гранях линейно интерполируются между двумя ближайшими точками (рис. 11), в то время как в алгоритме Diamond-Square высоты на гранях вычисляются по более сложному алгоритму: высота на грани вычисляется как сумма усреднения соседних высот (окрестность фон Неймана) и шума, зависящего от расстояния между точками (рис. 12).

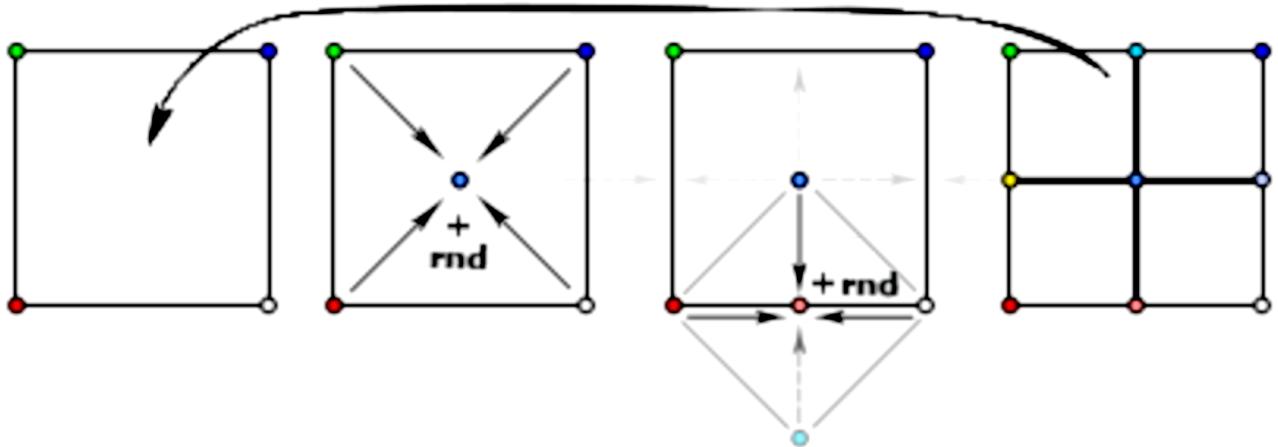
Рис. 11: Midpoint displacement



Алгоритм Diamond-Square состоит из двух шагов: “square” (генерация центральной точки) и “diamond” (генерация точек на гранях). Важно заметить, что шаг “diamond” использует центральные точки соседних квадратов, поэтому для корректного вычисления вершин, необходимо обсчет вести слоями: сначала для всех квадратов выполнить шаг “square”, а затем для всех ромбов выполнить шаг “diamond”. Тот факт, что алгоритм выполняется слоями для большого числа квадратов, означает, что алгоритм очень хорошо распараллеливается на GPU.

Отдельно необходимо рассматривать точки на границах квадрата. Т.к. алгоритм использует точки, которые не входят в рассматриваемый квадрат, то им присваивается какое-то стандартное значение высоты – 0. Однако можно представить, что плоскость свернута в тор. В таком случае точки, лежащие левее левой границы будут лежать несколько левее правой границы; границы, лежащие выше верхней границы будут лежать несколько выше нижней границы. Аналогично рассматриваются точки, лежащие правее правой границы и ниже нижней границы.

Рис. 12: Diamond-Square



2.3.2 Описание исходного кода

Исходный код можно разбить на три последовательные части:

1. Генерация карты высот.
2. Пост обработка карты высот.
3. Текстуризация карты высот и ее рендеринг.

Генерация карты высот представляет из себя итеративный процесс, повторяющий одни и те же последовательные шаги:

1. Square-алгоритм: генерация центральных точек.
2. Diamond-алгоритм: генерация точек на ребрах.
3. Recalculation: пересчитать координаты квадратов.

Вышеприведенный итеративный процесс повторяется до тех пор, пока не будет достигнута необходимая вычислительная точность, которая в свою очередь может зависеть либо от математической модели, симуляция которой будет производиться, либо, как вариант, от параметров монитора.

В данной работе постобработка карты высот представляет из себя сглаживание каждой высоты.

2.3.3 Исходный код

```

1 clear; close all;
2 rng(5); % set seed
3 %% initialization
4 %waterlevel = -35; % for power = 9
5 waterlevel = -20; % for power = 8
6 power = 8; % 8 -- good detalization
7 L = 2 ^ (power+1) + 1; % axes lengths
8 H = zeros(L,L); % height map
9 %% initialization of border heights
10 H(1,1) = 0; H(1,L) = 0;
```

```

11 H(L,1) = 0; H(L,L) = 0;
12 % initialization of index array
13 I = [ 1, 1, L, L, 0, 0 ];
14 %% applying transformations
15 H = diamond_square(I,H); % applying the diamond-square algorithm
16 % applying aftertransformations
17 H = H + abs(min(min(H))) + waterlevel; % setting water level
18 H = smooth_map(H);
19 %% plotting the height map
20 h.fig = figure(1); hold on;
21 h.surf = handle(surf(H));
22 zlimit = zlim; zmin = zlimit(1);
23 set(h.surf, 'ZData', [NaN, zmin+zeros(1, size(H,2)), NaN ; ...
24                                zmin+zeros(size(H,1),1), H, zmin+zeros(size(H,1),1); ...
25                                NaN, zmin+zeros(1, size(H,2)), NaN]);
26 axis square; axis equal; axis off; view(80,30);
27 demcmap([waterlevel max(max(H))]); shading interp; shadem([50 25], 'sun');

```

```

1 % Diamond-Square algorithm for terrain generation
2 function H = diamond_square(I,H)
3 % Input: I - index array, H - height map
4 % Ouput: H - height map
5     s = 1; % squares on level
6     while max(size(I)) ~= 0
7         [I,H] = square_alg(I,H); % for all squares
8         H = diamond_alg(I,H,s); % for all squares
9         [I,s] = recalculate(I,s); % reculculate squares
10    end
11 end

```

```

1 function [I,H] = square_alg(I,H)
2 % for all squares in I: create middle point, interpolate value with rand
3 % and update I with mid point coords
4
5 [N,~] = size(I);
6
7 for i = 1:N
8     % save coordinates
9     x0 = I(i,1); y0 = I(i,2);
10    x1 = I(i,3); y1 = I(i,4);
11    % get max distance
12    d = max(abs(y1-y0), abs(x1-x0));
13    % compute middle point local coordinates
14    tx = floor((x1-x0-1)/2) + 1;
15    ty = floor((y1-y0-1)/2) + 1;
16    % save local coordinates
17    I(i,5) = tx; I(i,6) = ty;
18    % compute height in middle point (using interpolation) with random
19    H(x0+tx,y0+ty) = (H(x0,y0) + H(x0,y1) + H(x1,y0) + H(x1,y1)) / 4 + ...
        rnd(d);
20 end
21
22 end

```

```

1 function H = diamond_alg(I,H,s)
2 % for all squares in I: calculate coordinates for the points

```

```

3 % on the borders and their heights
4
5 [N,~] = size(I);
6
7 for i = 1:N
8     % for all squares in I: calculate coordinates for the point on the left
9     % border and on the top border, calculate their heights
10    x0 = I(i,1); y0 = I(i,2);
11    x1 = I(i,3); y1 = I(i,4);
12    tx = I(i,5); ty = I(i,6);
13    H(x0,y0+ty) = ( border(I,H,s,i,'l') + H(x0,y0) + H(x0+tx,y0+ty) + ...
14        H(x0,y1) ) / 4 + rnd(max(tx,ty));
15    H(x0+tx,y0) = ( H(x0,y0) + border(I,H,s,i,'t') + H(x1,y0) + ...
16        H(x0+tx,y0+ty) ) / 4 + rnd(max(tx,ty));
17    if mod(i,s) == 0
18        % right border
19        H(x1,y0+ty) = ( H(x0+tx,y0+ty) + H(x1,y0) + H(x1,y1) ) / 4 + ...
20            rnd(max(tx,ty));
21    end
22    if i > (N-s)
23        % bottom border
24        H(x0+tx,y1) = ( H(x0,y1) + H(x0+tx,y0+ty) + H(x1,y1) ) / 4 + ...
25            rnd(max(tx,ty));
26    end
27 end
28
29 end

```

```

1 function [I1,S] = recalculate(I,s)
2 % for all squares in I: update the global array of squares
3 [N,M] = size(I);
4 I1 = zeros(4*N,M);
5
6 k = s;
7 S = 2 * s;
8
9 for i = 1:N
10    I1(floor((i-1)/k)*S+(2*i-1),:) = [ I(i,1), I(i,2), I(i,1) + ...
11        I(i,5), I(i,2) + I(i,6), 0, 0];
12    I1(floor((i-1)/k)*S+(2*i),:) = [ I(i,1) + I(i,5), I(i,2), I(i,3), ...
13        I(i,2) + I(i,6), 0, 0];
14    I1(floor((i-1)/k)*S+(2*i-1)+S,:) = [ I(i,1), I(i,2) + I(i,6), ...
15        I(i,1) + I(i,5), I(i,4), 0, 0];
16    I1(floor((i-1)/k)*S+(2*i)+S,:) = [ I(i,1) + I(i,5), I(i,2) + ...
17        I(i,6), I(i,3), I(i,4), 0, 0];
18 end
19
20 if I(1,5) == 1
21    I1 = [];
22 end
23
24 end

```

```

1 function H = smooth_map(H) % smooth the height map
2     [n,m] = size(H);
3     for i = 1:n
4         for j = 1:m

```

```

5      if i < 5 || i > n-5 || j < 5 || j > m-5
6          alpha = 1;
7      elseif i > floor((n-1)/2)-2 && i < floor((n-1)/2)+5
8          alpha = 1.25;
9      else
10         alpha = 1.25;
11     end
12     H(i,j) = ( elem(H,i-1,j-1) + elem(H,i,j-1) + elem(H,i+1,j-1) ...
13             + ...
14             elem(H,i,j-1) + elem(H,i,j) + elem(H,i,j+1) + ...
15             elem(H,i+1,j-1) + elem(H,i+1,j) + elem(H,i+1,j+1) ...
16             ) / 9 * alpha;
17 end

```

```

1 function r = border(I,H,s,i,t)
2     if t == 'l'
3         j = i - 1;
4     elseif t == 't'
5         j = i - s;
6     end
7     if j > 0
8         x0 = I(j,1); y0 = I(j,2);
9         xt = x0 + I(j,5); yt = y0 + I(j,6);
10        r = H(xt,yt);
11    else
12        r = 0;
13    end
14 end

```

```

1 function r = elem(H,i,j)
2     [n,m] = size(H);
3     if i > 0 && j > 0 && i <= n && j <= m
4         r = H(i,j);
5     else
6         r = 0;
7     end
8 end

```

```

1 function val = rnd(d)
2     %val = d/8*sign(randn(1)) * rand(1);
3     %val = d^(1/5) * rand(1);
4     %val = d/10 * rand(1);
5     val = d/10 * randn(1);
6     %val = rand(1);
7 end

```

2.3.4 Пример работы программы

В качестве примеров работы программы показано 4 различные карты высот, сгенерированные при различных ядрах случайных чисел: (рис. 13) – $seed = 5$; (рис. 14) – $seed = 7$; (рис. 15) – $seed = 8$; (рис. 16) – $seed = 111$.

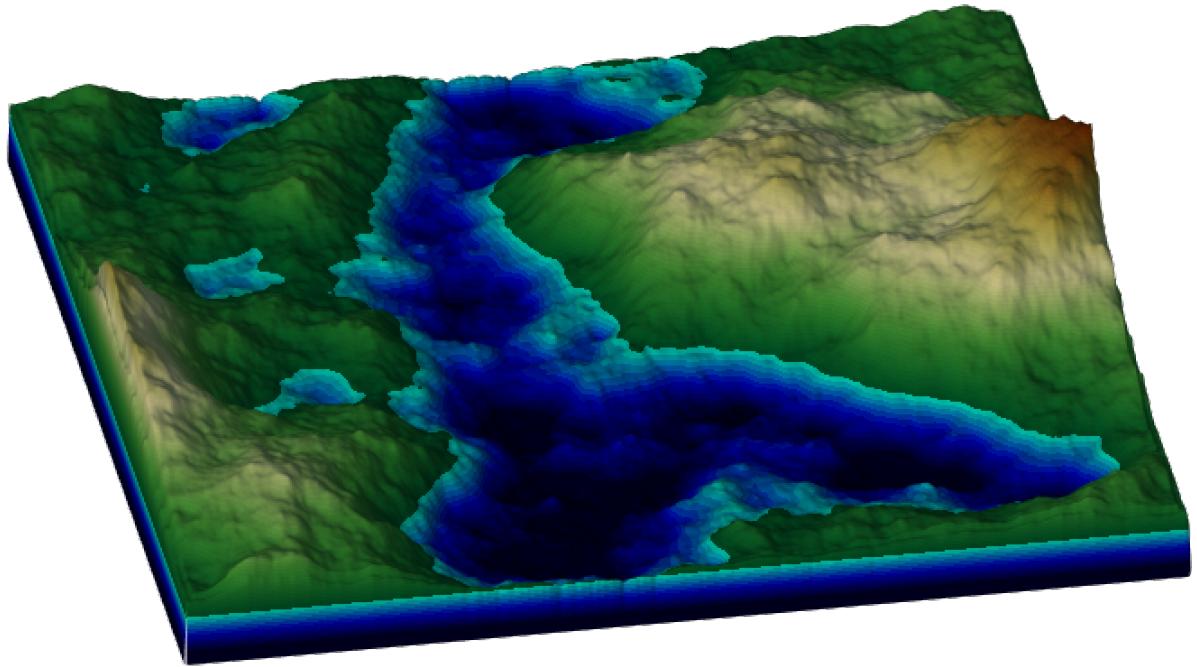


Рис. 13: Пример 1 ($seed = 5$)

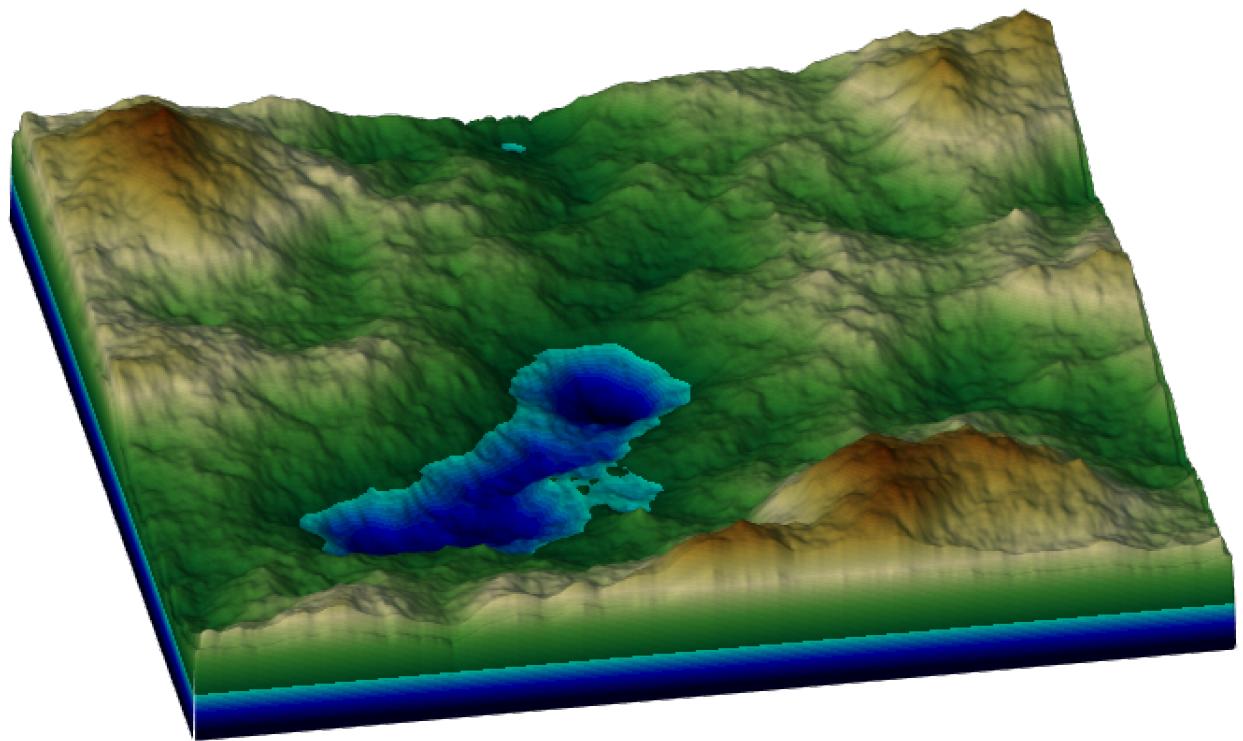


Рис. 14: Пример 2 ($seed = 7$)

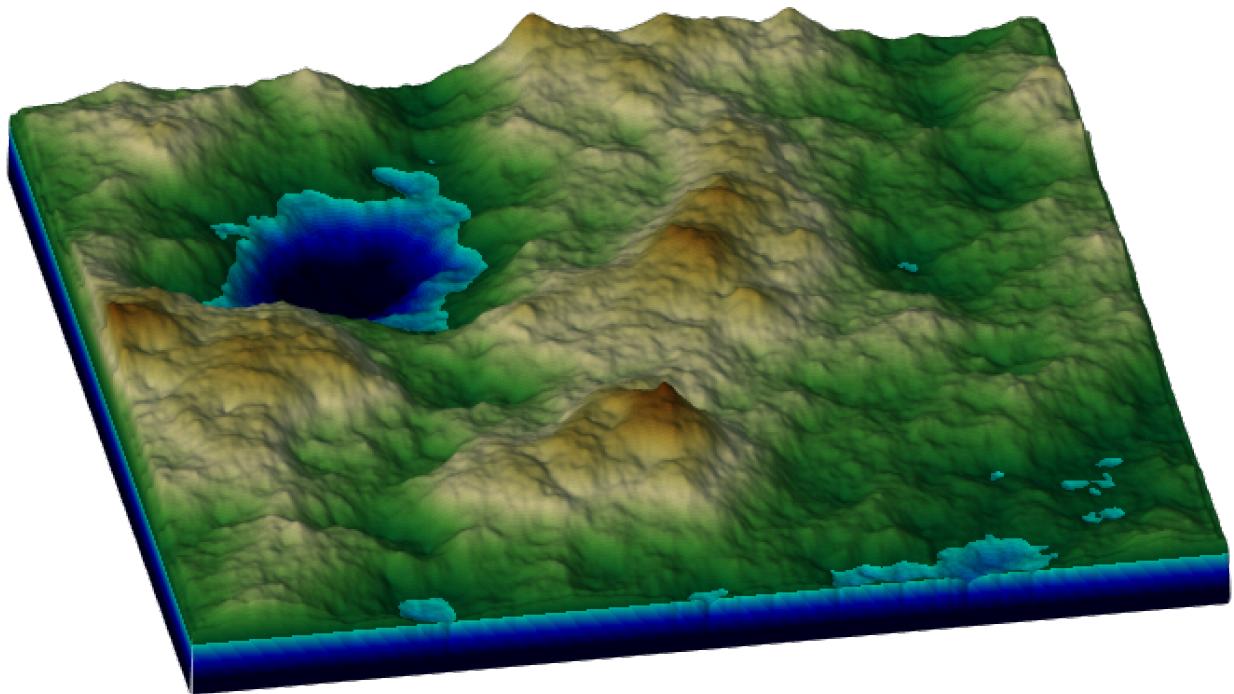


Рис. 15: Пример 3 ($seed = 8$)

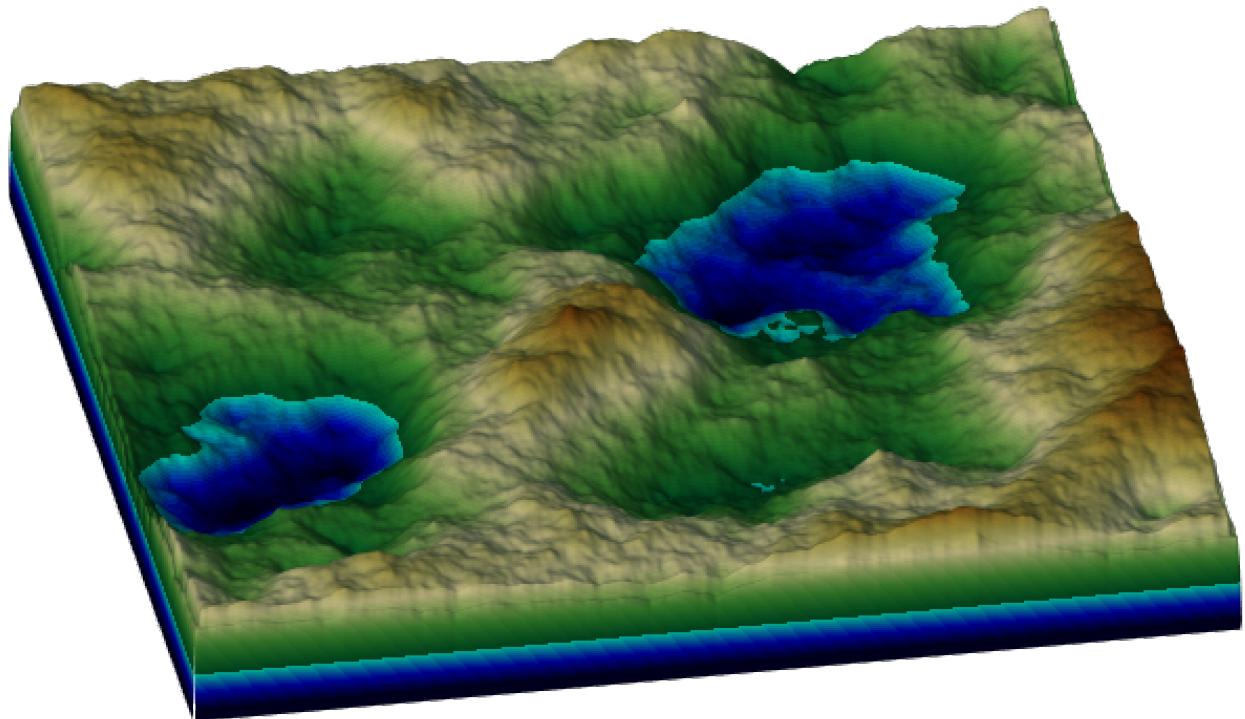


Рис. 16: Пример 4 ($seed = 111$)

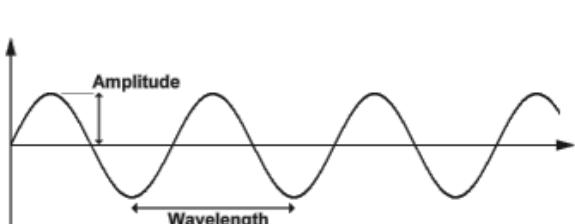
2.4 Фрактальный шум

2.4.1 Описание алгоритма

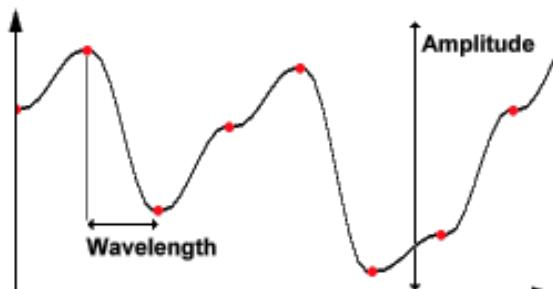
Перед тем, как описать сам алгоритм генерации ландшафта необходимо ввести такие понятия, как амплитуда и длина волн. Из физики известны соотношения, показанные

на (рис.17а). Длина волны – это расстояние от одной вершины к другой. Амплитуда – это высота волны.

На (рис.17б) изображена функция шума. Красные точки указывают на случайные значения, определенные по измерению функции. В этом случае амплитуда – это разница между минимальным и максимальным значениями которые у функции могут быть. Длина волны – это есть расстояние от одного красного пятна к другому.



(a) Физические понятия



(b) Аналоги физических понятий

Рис. 17: Необходимые понятия

Для удобства дальнейшего изложения, обозначим частотой величину обратную длине волны.

Теперь, если вы взять много таких гладких функций, с различной длиной волны и амплитудой, сложить их все вместе, чтобы создать хорошо зашумленную функцию, то получится фрактальный шум. Пример описанной процедуры можно посмотреть на (рис.18).

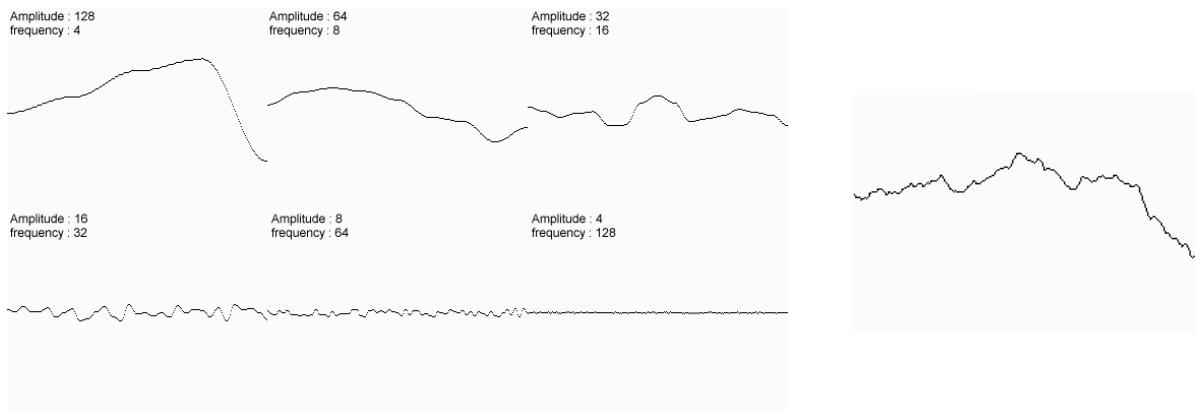


Рис. 18: Одномерный фрактальный шум

Чтобы избежать повторения слов «амплитуда» и «частота» все время, используется единственное число для определения каждой амплитуды и каждой частоты. Эта величина называется настойчивостью(Persistence). Определение настойчивости:

$$\text{frequency(частота)} = 2^i$$

$$\text{amplitude(амплитуда)} = \text{persistence(стойкость)}^i$$

Иллюстрацию эффекта настойчивости можно видеть на (рис.19) – (рис.22).

Каждую добавленную последующую шумовую функцию называют октавой. Причиной этого является то, что каждая функция шума имеет частоту в два раза большую чем предыдущая. В музыке октавы также обладают этим свойством.



Рис. 19: Настойчивость = $\frac{1}{4}$
 Частоты: 1, 2, 4, 8, 16, 32
 Амплитуды: 1, $\frac{1}{4}$, $\frac{1}{16}$, $\frac{1}{64}$, $\frac{1}{256}$, $\frac{1}{1024}$



Рис. 20: Настойчивость = $\frac{1}{2}$
 Частоты: 1, 2, 4, 8, 16, 32
 Амплитуды: 1, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{32}$



Рис. 21: Настойчивость = $\frac{1}{\sqrt{2}}$
 Частоты: 1, 2, 4, 8, 16, 32
 Амплитуды: 1, $\frac{1}{\sqrt{2}}$, $\frac{1}{2}$, $\frac{1}{2\sqrt{2}}$, $\frac{1}{4}$, $\frac{1}{4\sqrt{2}}$



Рис. 22: Настойчивость = 1
 Частоты: 1, 2, 4, 8, 16, 32
 Амплитуды: 1, 1, 1, 1, 1, 1

То, сколько октав будет складываться целиком зависит от пользователя. При используя функции фрактального шума для рендеринга изображения на экран, наступает момент, когда октава может иметь слишком высокую частоту, чтобы быть отображаемой. Там просто может не быть достаточно пикселей на экране, чтобы воспроизводить все мелкие детали на очень высокой функции шума.

Т.к. для разных частот задаются в разных точках, то для того, чтобы сложить полученные функции, необходимо их интерполировать. Для большей реалистичности используется кубическая интерполяция.

2.4.2 Исходный код

```

1 rng(8); % set seed
2
3 persistence = 0.6;
4 octaves = 7;
5
```

```

6 xlim = 10;
7 ylim = 10;
8
9 [Xq,Yq] = meshgrid(1:1/(2^(octaves-1)):xlim,1:1/(2^(octaves-1)):ylim);
10 H = zeros(size(Xq));
11
12 for i = 1:octaves
13     freq = 2^(i-1);
14     ampl = 5*persistence^(i-1);
15     [X,Y] = meshgrid(1:1/freq:xlim,1:1/freq:ylim);
16     H = H + interp2(X,Y,ampl*rand(size(X)),Xq,Yq,'cubic');
17 end
18
19 minHeight = min(min(H));
20 H = H - minHeight;
21
22 % heuristic
23 heuristicHeight = 0.71 * max(max(H));
24 H = 1.7 * H + 0.1 * H .* (H > heuristicHeight) - sin(H);
25 H = smooth_map(smooth_map(H));
26
27 % setting water level
28 maxHeight = max(max(H));
29 waterLevel = 0.45*maxHeight; % 45% of heights
30 H = H - waterLevel;
31
32 %% plotting the height map
33 h.fig = figure(1); hold on;
34 h.surf = handle(surf(H));
35 zlimit = zlim; zmin = zlimit(1);
36 set(h.surf, 'ZData', [NaN, zmin+zeros(1, size(H,2)), NaN; ...
37 zmin+zeros(size(H,1), 1), H, zmin+zeros(size(H,1), 1); ...
38 NaN, zmin+zeros(1, size(H,2)), NaN]);
39 axis square; axis equal; axis off; view(80,30);
40 demcmap(H); shading interp; shadem([50 25], 'sun');

```

2.4.3 Пример работы программы

В качестве примеров работы программы показано три различные карты высот, сгенерированные при одинаковых ядрах случайных чисел, но разных количествах октав:

3 Сравнение рассмотренных алгоритмов

3.1 Анализ накладываемых сеток

В работе были рассмотрены два вида накладываемых сеток: обычные квадратные сетки вида

$$\Omega_h = \{ \langle x_i, y_j \rangle \mid x_i = ih, y_j = jh, i = 0, \dots, N, j = 0, \dots, M \}$$

и сетки, основанные на вычислении диаграммы Вороного. Сетки, построенные с помощью диаграммы Вороного являются сильно нерегулярными из-за их стохастической природы, поэтому заметить какой-либо “шаблон” у данных сеток практически невозможно. Это несомненно является большим плюсом по сравнению с сетками, которые генерируются алгоритмами нестochasticеской природы. Существенным минусом алгоритма является его сложность реализации: самым сложным этапом является написание алгоритма построения диаграммы Вороного, например, алгоритма Форчуна.

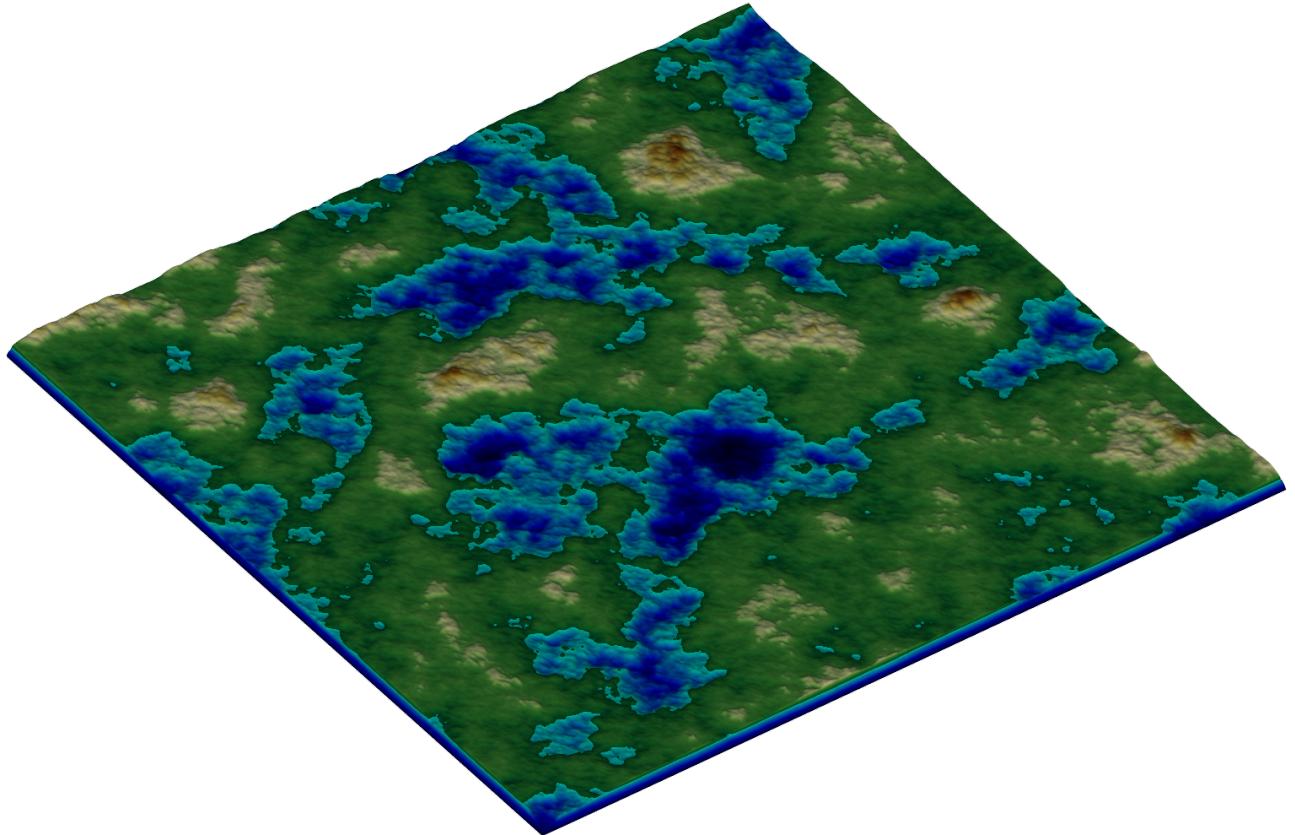


Рис. 23: Пример 1 ($seed = 8$, $octaves = 8$)

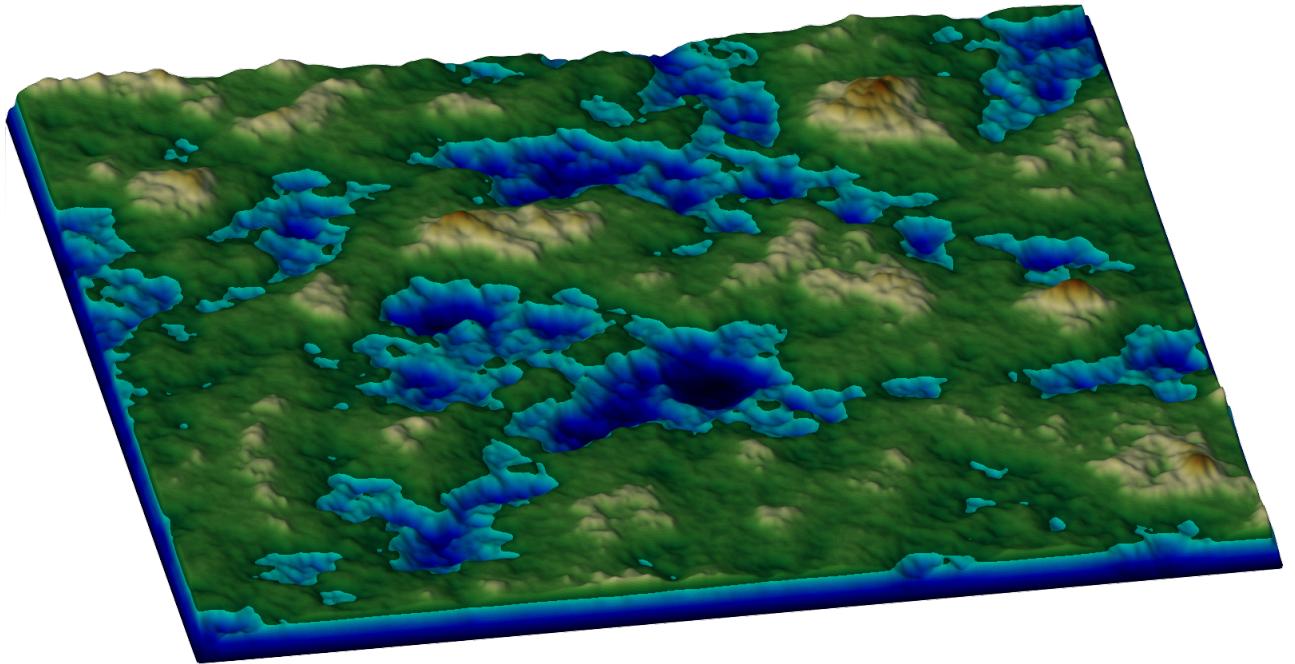


Рис. 24: Пример 2 ($seed = 8$, $octaves = 7$)

Даже учитывая тот факт, что алгоритм Форчуна имеет относительно малую вычислительную сложность $O(n \log n)$, у метода диаграмм Вороного есть ограничение. Если предположить, что требуемая сетка очень большая, что справедливо во многих случаях, то необходимо уметь разбивать вычисление всей сетки на вычисление нескольких “окон”. Именно на данном этапе у оригинального метода диаграммы Вороного возникают про-

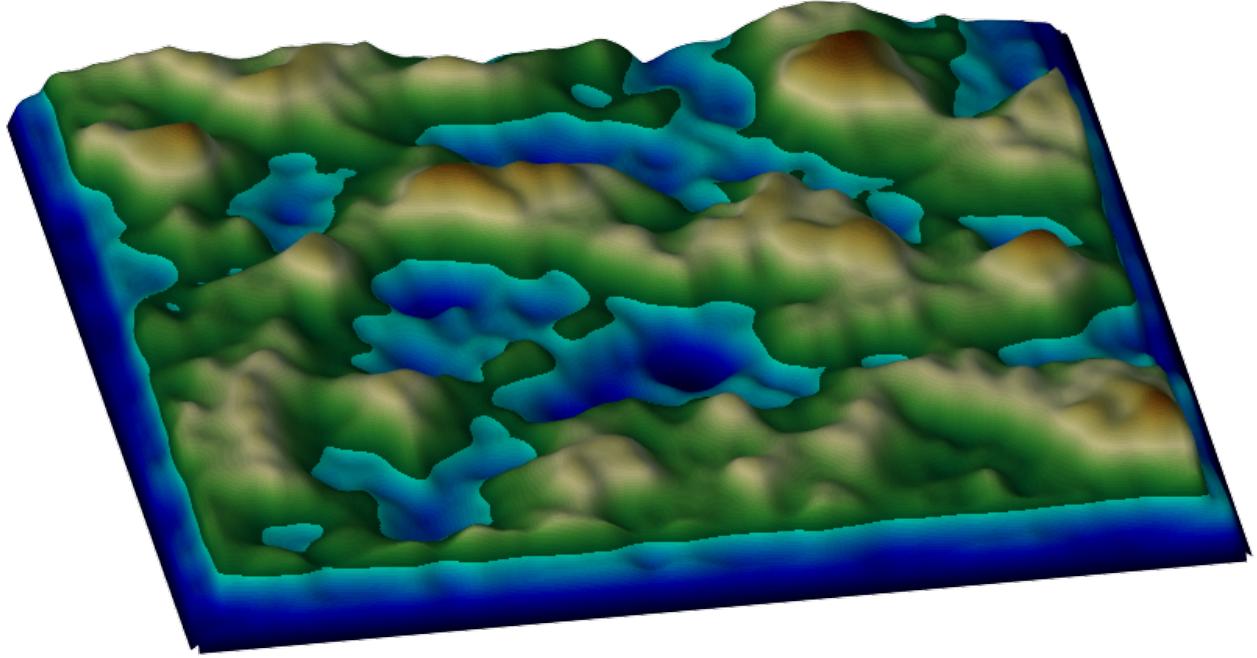


Рис. 25: Пример 3 ($seed = 8$, $octaves = 5$)

блемы. Если посмотреть на (рис.8), то можно заметить, что сетка создается в некоторой внутренней области, а для того, чтобы строить большие сетки, необходимо, чтобы сетка строилась во всем “окне”.

Мне кажется, что подобная проблема может быть легко решена с помощью введения граничных точек, общих для связных “окон”, которые не изменяют своего местоположения во время релаксации Ллойда. Таким образом множество точек каждого “окна” разбивается на два множества:

$$\bar{\Omega} = \Omega \bigcup \partial\Omega,$$

где под Ω понимается множество внутренних точек, над которыми происходят все вычисления, описанные в параграфе (2.1), а под $\partial\Omega$ – неизменяемое множество граничных точек.

Несравненно более простым видом сеток являются квадратные сетки. Свойства квадратных сеток противоположны свойствам сеток Вороного: квадратные сетки элементарно строятся, однако для того, чтобы получить хороший результат, необходимо использовать очень мелкую сетку, что сильно нагружает систему. Именно из-за этой проблемы введены такие приемы, как LOD-техники и Tessellation shader в языке OpenGL.

3.2 Анализ алгоритмов построения карты высот

В работе были подробно рассмотрены два алгоритма генерации полей высот: Diamond-square и Fractal noise и упомянут метод холмов.

Метод холмов является показывает очень хорошие результаты если его использует умелый художник. Но т.к. алгоритм сильно зависит от человеческого фактора, то он является исключительно медленным и непостоянным.

Алгоритм Diamond-square также показывает неплохой результат, но только на первый взгляд. Карта высот, получающаяся в результате работы данного алгоритма может создать лишь скалистую местность. Если сравнить результат работы алгоритма с реальными ландшафтами, то окажется, что в реальной природе ландшафт в подавляющем

большинстве случаев является очень пологим, а все неровности и неточности искажаются флорой и фауной. Именно такого типа карты получаются у алгоритма Fractal noise.

Опять же для больших карт, необходимо уметь разбивать вычисления на “окна”. На данном этапе алгоритм Diamond-square сильно проигрывает алгоритму Fractal noise. Основная проблема Diamond-square заключается в том, что алгоритм рекурсивен и каждый шаг на k -ой итерации зависит от всех шагов на $(k - 1)$ -ой итерации. Один из возможных методов решения данной проблемы является ленивая динамика, однако для полной уверенности необходимо реализовать данный метод и провести эксперимент.

В моей реализации алгоритм Fractal noise также не способен на генерацию больших карт, однако для того, чтобы он стал на это способен, необходимо существенно меньше программировать, но воспользоваться математическим аппаратом. Если в моей реализации используется генератор случайных чисел для каждой точки $\langle x, y \rangle$, то для больших карт необходимо создать поле случайных значений, т.е. поле $\langle x, y \rangle$ должно быть случайным, однако для одинаковых значений x, y , генератор должен возвращать одинаковые случайные значения.

Далее следует отметить, что все алгоритмы построения карт высот используют какого-либо вида конечно-разностные сетки. Оба рассмотренных алгоритма строятся на квадратных сетках, однако если проинтерполировать полученную функцию высоты и наложить сетку Вороного, то визуально ландшафт может измениться. Насколько хорошие результаты дает такой метод может показать только эксперимент.

3.3 Подведение итогов

Каждый метод имеет свои области применения, однако для конкретной задачи процедурной генерации ландшафта очень хорошим выбором, дающим превосходные результаты является алгоритм Fractal noise на квадратных сетках.