

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	4
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....	5
ВВЕДЕНИЕ.....	6
1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ	8
1.1 Методы представления и генерации ландшафтов.....	8
1.1.1 Способы представления данных для генераторов ландшафта	8
1.1.2 Методы генерации ландшафтов	11
1.2 Рельеф морского дна.....	12
1.2.1 Срединно-океанические хребты.....	13
1.2.2 Континентальные окраины	14
1.2.3 Глубоководные котловины	15
2 ПРОЕКТИРОВАНИЕ АЛГОРИТМА	17
2.1 Формулирование требований	17
2.2 Предлагаемое решение	17
2.3 Архитектура алгоритма	21
3 ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ РЕАЛИЗАЦИИ.....	24
3.1 Способ представления данных	24
3.2 Способ визуализации.....	26
3.2.1 Описание формата vox	27
3.3 Реализация алгоритма.....	29
3.3.1 Задание исходных данных	29
3.3.2 Разбиение дна на отдельные плиты	29
3.3.3 Установка высоты и скорости плит	30

3.3.4 Создание объектов рельефа	30
3.3.5 Симуляция	31
3.3.6 Сохранение ландшафта	37
3.3.7 Блок-схема алгоритма.....	37
3.3.8 Результаты работы алгоритма	44
3.4 Доработка результата	47
3.4.1 Шумовые алгоритмы	47
3.4.2 Выбор шума.....	53
4 ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ АЛГОРИТМА	56
4.1 Конфигурация системы	56
4.2 Описание эксперимента	56
4.3 Результаты эксперимента.....	57
ЗАКЛЮЧЕНИЕ	61
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	63

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

DAG – ненаправленный граф без циклов (directed acyclic graph)

DEM – цифровая модель рельефа (digital elevation model)

CGAN – условная GAN (conditional GAN)

GAN – генеративно-состязательная сеть (generative adversarial network)

PCG – процедурная генерация контента (procedural content generation)

PROGAN – постепенно растущая GAN (progressively growing GAN)

RaLSGAN – Релятивистская усредненная сеть наименьших квадратов GAN
(Relativistic average Least Squares GAN)

RIFF - Формат файла обмена ресурсами (Resource Interchange File Format)

SSVDAFs - разреженные воксельные DAG с учетом симметрии (Symmetry-aware Sparse Voxel DAGs)

SVO – разреженное воксельное октодерево (sparse voxel octree)

Voxel – объёмный пиксель (volumetric pixel)

Г.К. – глубоководная котловина

К.О. – континентальная окраина

ОЗУ – Оперативное запоминающее устройство

ПО – программное обеспечение

СОХ – срединно-океанический хребет

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Гипсографическая кривая — интегральная функция распределения глубин океана и высот земной поверхности.

Глубоководный желоб - глубокая и длинная впадина на дне океана.

Шельф - выровненная область подводной окраины, примыкающая к суше и характеризующаяся общими с ней чертами рельефа и геологической структурой.

Рифтогенез - процесс горизонтального растяжения литосферных плит.

Субдукция - процесс, при котором происходит погружение одних блоков земной коры под другие.

Абраузия - процесс механического разрушения и сноса горных пород в береговой зоне водоёмов волнами и прибоем.

Шум — беспорядочные колебания различной физической природы, отличающиеся сложностью временной и спектральной структуры

Дизеринг (Dithering) – процесс подстановки шума в исходную функцию с целью устранения ее искажений.

Алгоритмический шум — это процедурный метод для моделирования и вычисления шума.

ВВЕДЕНИЕ

Почти в каждом фильме или видеоигре есть какая-то виртуальная местность, на которой происходит действие сюжета. С развитием компьютерной графики и методов трехмерной визуализации потребители стали предъявлять все больше требований к контенту, а художникам и дизайнерам приходится прилагать все больше усилий для создания правильной атмосферы и погружения туда игроков или зрителей. Конечно, человеческие ресурсы ограничены, а также стоят денег и времени, поэтому производители виртуальных миров все чаще прибегают к методам автоматической генерации контента и, в частности, ландшафтов. Помимо развлекательных целей генерацию ландшафтов также используют для различных симуляторов, как, например, в работе [1] и других областей компьютерной графики.

Изначально процедурная генерация контента появилась как средство преодоления технических ограничений компьютеров – память систем сорокалетней давности просто не могла вместить в себя все элементы, требующиеся для игры [2]. За время существования PCG было проведено множество исследований в этой области, разработано много различных алгоритмов для генерации контента, однако, несмотря на то, что большую часть поверхности земли занимают водоемы, практически все алгоритмы сосредоточены на формировании поверхности суши и исследуют процессы, влияющие на нее [3]. На самом деле, водоемам на поверхности суши, таким как реки, озера, болота и другие уделено достаточно внимания в литературе, т. к. они участвуют в процессах водной эрозии, что влияет на генерируемый рельеф. Например, в работе [4] генерация основных элементов ландшафта, таких как горы и холмы происходит после генерации дренажных бассейнов и зависит от их расположения, а в работе [5] основными методами, формирующими рельеф являются два алгоритма водной эрозии, основанные на движении воды.

При этом формирования дна океанов и, соответственно морей, которые являются их частью, остается темой малоизученной, поэтому целью данной исследовательской работы является разработка алгоритма генерации океанического дна.

Для достижения поставленной цели необходимо выполнить ряд задач. Во-первых, исследовать существующие методы генерации ландшафтов. Во-вторых, разобраться в принципах формирования рельефа морского дна, а также построить модель формирования подводного ландшафта. И в заключении на основе собранной разработать алгоритм, который будет автоматически создавать ландшафт океанического дна, опираясь на разработанную модель, а также провести эксперименты, для определения производительности разработанного алгоритма.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

Сперва мы рассмотрим какие используются методы представления и генерации ландшафтов в компьютерной графике, а затем перейдем к рельефу морского дна.

1.1 Методы представления и генерации ландшафтов

Существуют различные способы представления исходных данных для генераторов ландшафта, а также различные способы генерации, каждый из которых обладает своими преимуществами и недостатками. В данном разделе мы рассмотрим используемые на практике в играх методы представления и генерации ландшафтов [6].

1.1.1 Способы представления данных для генераторов ландшафта

Есть две основных подгруппы на которые разделяются способы представления данных: карты высот, представленные тем или иным способом либо волюметрические (объемные) данные.

1.1.1.1 Карты высот

Карты высот — это способ представления ландшафта, который описывает местность проводя соответствие между координатами, заданными на плоскости и высотой их поднятия над уровнем земли.

Наиболее распространенным примером карты высот является изображение, состоящее из оттенков серого, где наиболее высокую точку описывает белый цвет, а низкую — черный, пример данной карты показан на рисунке 1.

Еще одним дискретным представлением карты высот является двумерная сетка, где в каждой ячейке записана высота точки. Для представления непрерывных пространств значения карты высот вблизи нужной точки интерполируют. Существуют также непрерывные представления карты высот, которые задаются функциями. Этот метод хорош

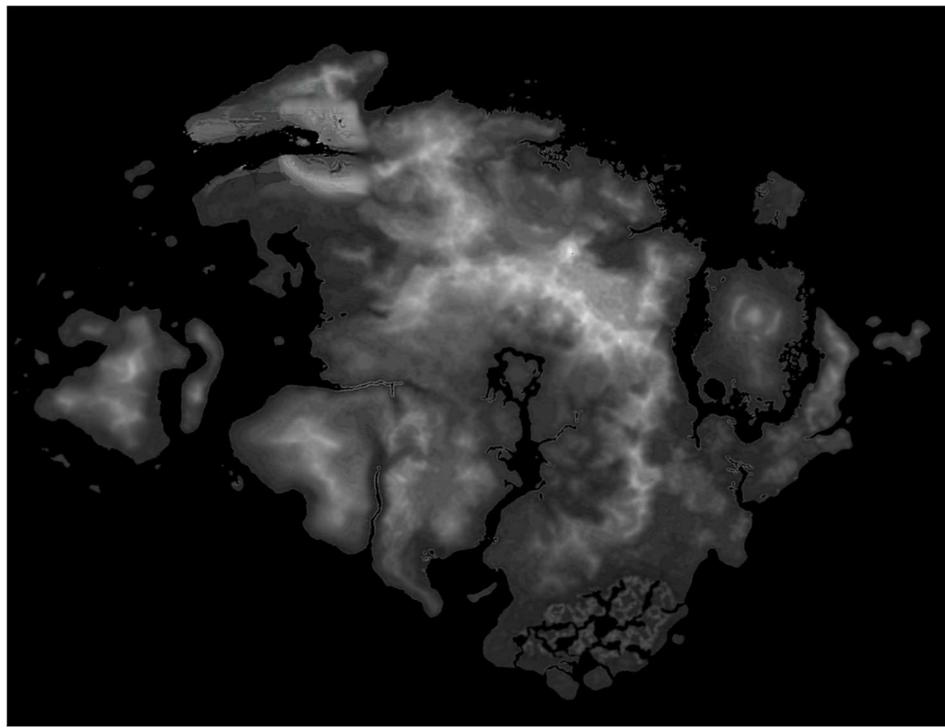


Рисунок 1 - Представление карты высот в виде градаций серого

тем, что практически не занимает место в памяти, но он может быть вычислительно сложным. Примеры дискретного и функционального подхода изображены на рисунке 2.

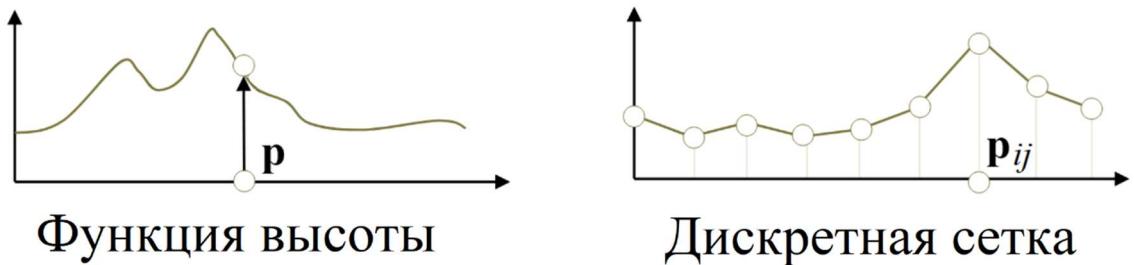


Рисунок 2 - Способы задания карты высот непрерывный и дискретный

Еще одним методом представления карты высот являются слои материалов. В таком подходе пользователь описывает материалы и их отношения между собой, т. е. какой материал должен быть под или над текущим. У этого метода также есть дискретный и непрерывный способы представления. В первом пользователь задает непосредственно набор материалов, а во втором набор функций их описывающий, такой подход изображен на рисунке 3.

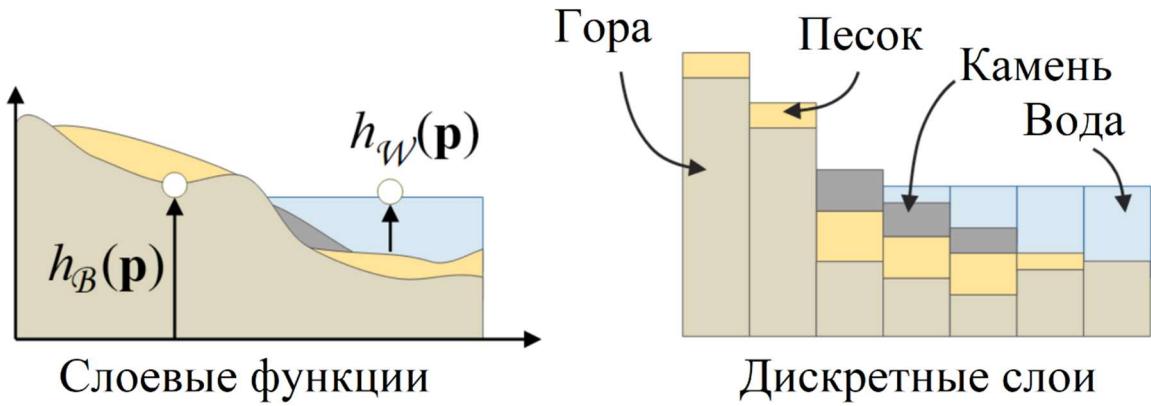


Рисунок 3 - Способ задания карты высот слоями

К общим плюсам представления местности в виде карты высот можно отнести малое потребление памяти, а к минусам – неспособность представлять объемные объекты такие как пещеры, либо навесы.

1.1.1.2 Объемные данные

Объемные данные — это способ задания ландшафта, при котором каждая точка в трехмерном пространстве имеет свои свойства и материалы. Такой подход можно реализовать тремя способами.

Во-первых, с помощью разбиения пространства на равномерные кусочки, называемые voxелями. В таком подходе плюсом является удобство работы с каждым элементом пространства, его легкое изменение или удаление, однако он требует большого количества памяти. Во-вторых, можно использовать функциональный подход и задать функцию, которая для каждой точки пространства возвращает набор ее свойств или материалов. В этом подходе устраняется проблема voxелей, потому что для сохранения функции не требуется много памяти, однако подобрать нужный набор функций может быть проблематично. К тому же, в данном методе требования к памяти заменяются вычислительными требованиями. Данные методы изображены на рисунке 4.

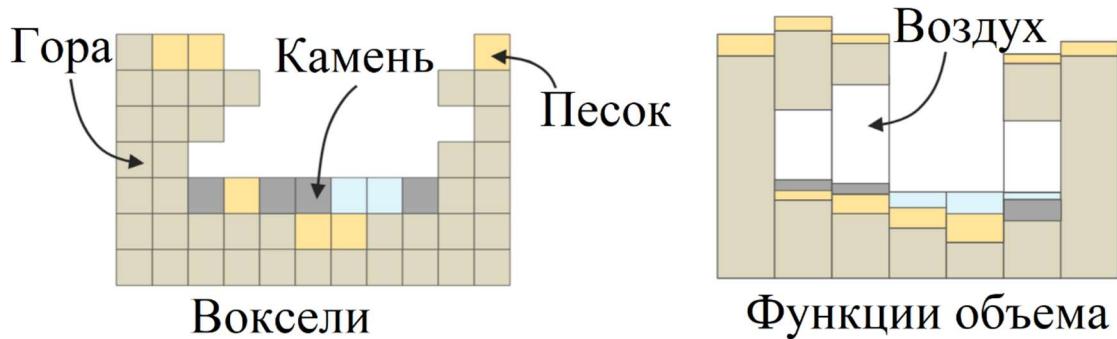


Рисунок 4 - Представление в виде объемных данных

Также существуют подходы, которые пытаются объединять воксельное и функциональное представление.

1.1.2 Методы генерации ландшафтов

Процедурная генерация в своей основе содержит три главных метода: физическая симуляция, генерация по примерам и генерация на основе различных алгоритмов.

1.1.2.1 Генерация ландшафтов на основе физической симуляции

В основе физической симуляции лежит итеративное применение агентов симуляции к исходному ландшафту. Под агентами тут понимаются различные природные процессы такие как водная эрозия, пожары или влияние ветра. В зависимости от поставленной задачи может применяться как один агент, так и множество различных, как например в работе [7], где разработали фреймворк, который использует дождь, гравитацию, огонь, молнию и температуру для генерации реалистичных ландшафтов, а также поддерживает добавление новых агентов.

С помощью физической симуляции можно генерировать реалистичные ландшафты, однако этот процесс может требовать большое количество времени. Поэтому некоторые работы используют алгоритмы приближения, как например разработанный в статье [8] метод наклонных для приближенного влияния эрозии, вместо настоящей симуляции.

1.1.2.2 Генерация ландшафтов по примерам

Генерация по примерам основана на использовании данных, полученных из существующих рельефов, например их фотографий или спутниковых снимков, эти данные называются цифровой моделью рельефа – DEM. DEM используются для обучения искусственного интеллекта, который в свою очередь генерирует новые реалистичные ландшафты. Зачастую обучают модели, основанные на GAN. Например, в работе [9] для генерации ландшафтов на основе спутниковых снимков использовались модели CGAN и PROGAN, а в работе [10] используют RaLSGAN для генерации городского ландшафта на основе спутниковых снимков и данных из OpenStreetMap.

При таком подходе получаются натуральные ландшафты, однако для его применения необходимо обучение моделей на большом количестве данных, которые не всегда доступны.

1.1.2.3 Алгоритмическая генерация

Алгоритмическая генерация, как и указано в названии, основана на использовании различных алгоритмов, например существует множество различных вариаций шумовых алгоритмов [11], они достаточно широко распространены и многие основаны на шуме Перлина, но также есть и другие, как например, поисковые алгоритмы или алгоритмы, основанные на конечных автоматах. Например, в работе [12] используют конечные автоматы и функцию соответствия для получения результатов, которые бы удовлетворили дизайнера.

С помощью алгоритмической генерации удается добиться создания рельефа в реальном времени, однако зачастую он получается однообразным. При использовании функций соответствия удается улучшить визуальные характеристики получаемых рельефов, однако подбор таких функций – еще одна трудная задача.

1.2 Рельеф морского дна

Рассвет изучения рельефа морского дна произошел с изобретением эхолота – прибора, который рассчитывает расстояние по скорости отражения

звучка от поверхности. Благодаря этому изобретению человечество сильно продвинулось в изучении моря и океана.

Согласно данным, представленным на гипсографической кривой, показанной на рисунке 5, средняя глубина океана достигает 3800 метров, а максимальная глубина – 11022 метра.

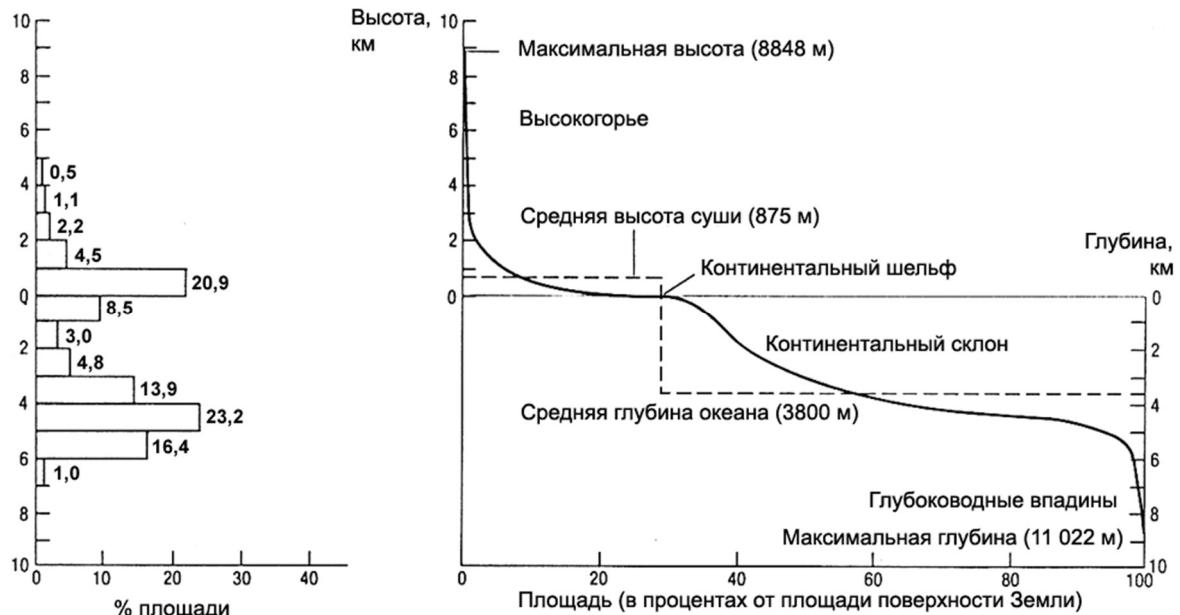


Рисунок 5 - график соответствия высоты земли занимаемой площади

К основным формам рельефа морского дна относятся: срединно-океанские хребты (СОХ), континентальные окраины и глубоководные, или абиссальные, котловины [13].

1.2.1 Срединно-океанические хребты

Срединно-океанские хребты названы так потому, что проходят в середине большинства океанов, кроме Тихого, как показано на рисунке 6. Образование СОХ происходит за счет рифтогенеза - раздвижения литосферных плит и выхода подземной магмы на поверхность океанического дна [14], этот процесс представлен на рисунке 7. Скорость движения литосферных плит составляет от 2x до 15 см в год [13]. СОХ возвышается над дном океана в среднем на 2 км, расположен на глубине около 2,5 км, и в ширину имеет около одного километра.

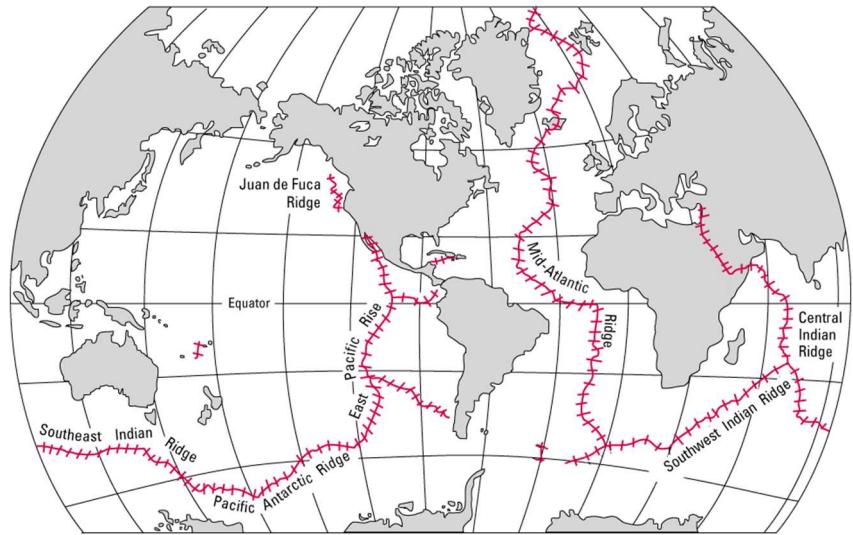


Рисунок 6 - расположение срединно-океанских хребтов

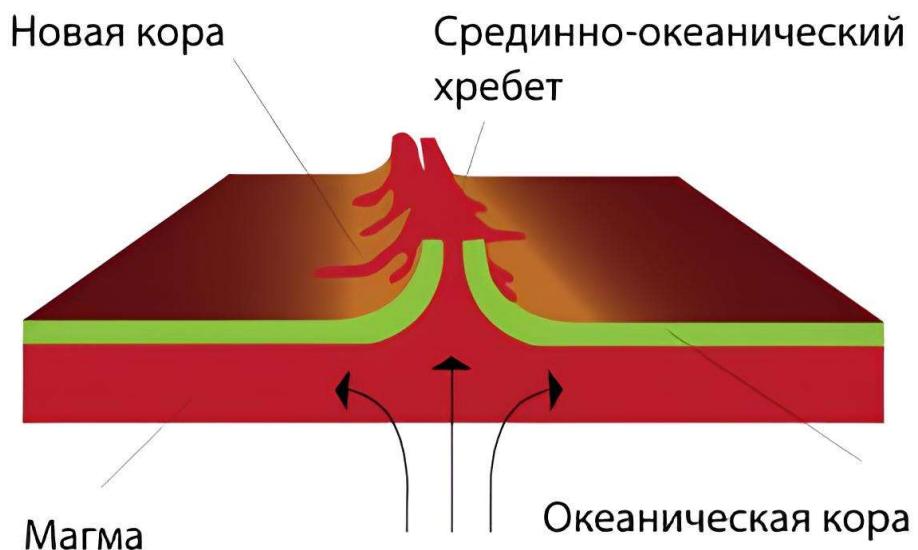


Рисунок 7 - формирование срединно-океанского хребта

1.2.2 Континентальные окраины

Континентальная окраина — это область перехода, связывающая между собой океан и континент. Есть два типа континентальных окраин — пассивные и активные [13]. Пассивная континентальная окраина характеризуется низкой сейсмической активностью — это окраины, которые под действием силы тяжести и накопившихся осадков постепенно опускаются и в основном представляют собой шельф на глубине до 200 м [15]. Активная континентальная окраина характеризуется наличием высокой сейсмической активности, присутствием глубоководных желобов и представляет собой

процесс субдукции – океанская литосферная плита, погружается под материковую [16]. Процесс субдукции представлен на рисунке 8.

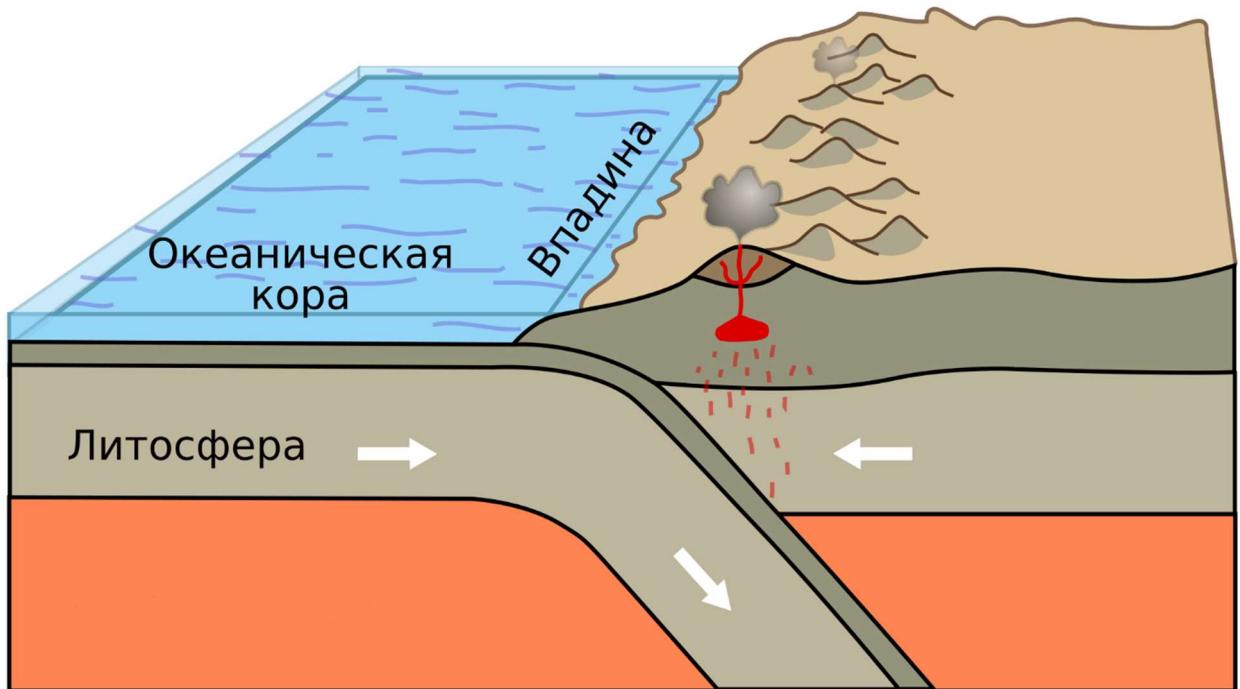


Рисунок 8 - погружение литосферной плиты под материк

1.2.3 Глубоководные котловины

Глубоководные котловины расположены между континентальными окраинами и срединно-океаническими хребтами [13] и представляют собой понижение дна океана или моря обычно овальной или изометрической (округой) формы [17]. Углубление обычно достигает 2.5 км до 6 км. Рельеф глубоководных котловин может быть весьма разнообразным, однако большая их часть представляет собой плоские или холмистые равнины и иногда возвышенности. Дно котловин зачастую плоское, либо покрыто группами гор [18]. Такие подводные горы, называемые гайотами, формируются из вулканов, разрушенных абразией, за счет чего их вершина превращается в плато, а сами они постепенно опускаются вместе с литосферной плитой [13]. На рисунке 9 представлен процесс образования гайота.

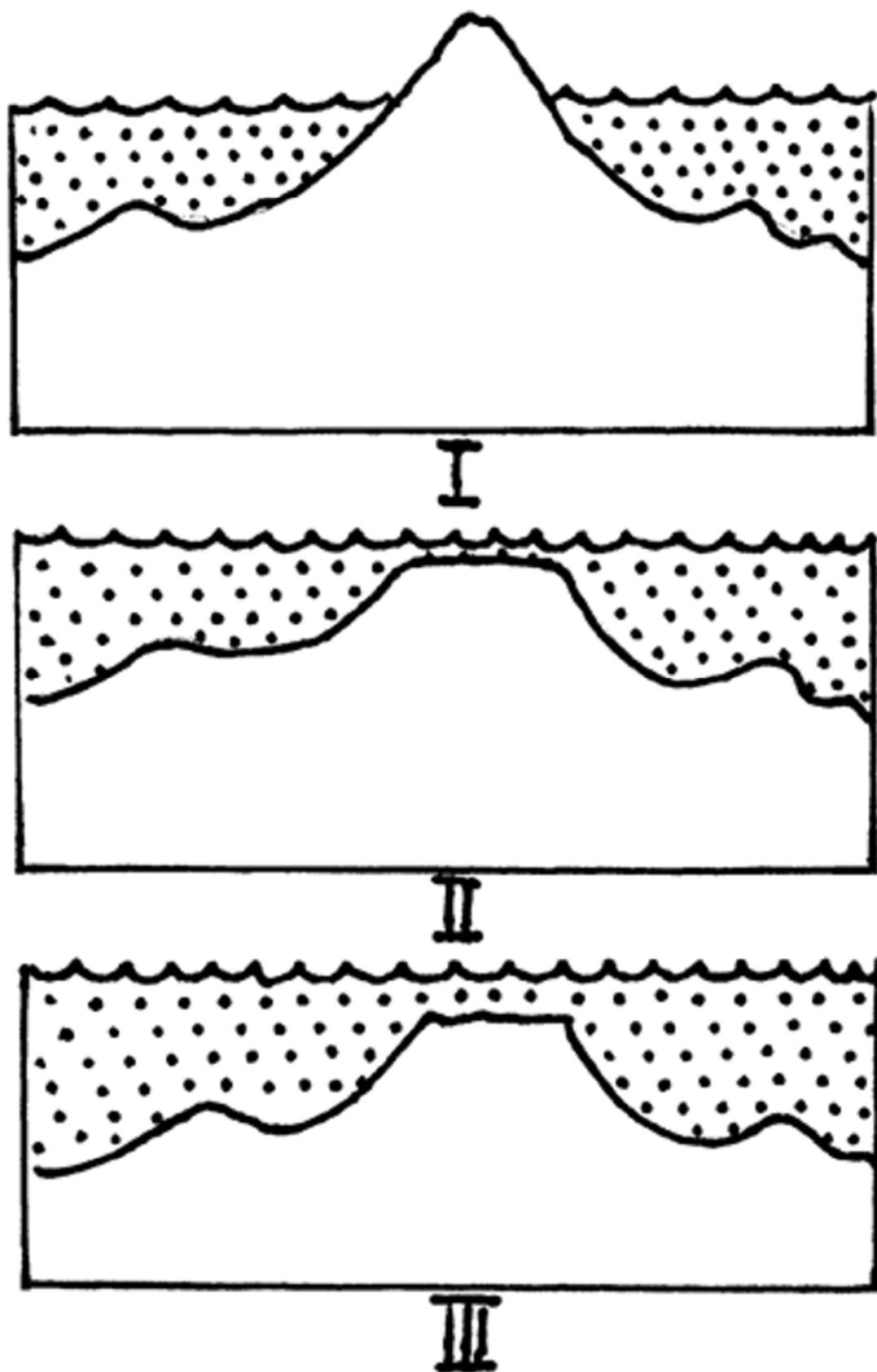


Рисунок 9 - Образование гайота. I — вулканический остров; II — срезание морской абразией вершины острова; III — опускание океанского дна

2 ПРОЕКТИРОВАНИЕ АЛГОРИТМА

2.1 Формулирование требований

Сформулируем требования для разрабатываемого алгоритма. Во-первых, алгоритм должен быть способен генерировать перечисленные ранее основные виды морского рельефа. Во-вторых, алгоритм должен быть расширяемым – помимо основных видов в будущем можно добавить и другие формы рельефа. В-третьих, генерация рельефа должен опираться на настоящие процессы, происходящие при формировании морского дна.

2.2 Предлагаемое решение

На основании данных, полученных из исследования устройства поверхности морского дна и процессов формирования основных форм рельефа, а также выдвинутых требований, можно предложить следующий итерационный алгоритм генерации морского дна:

1) Задание исходных данных.

Исходными данными для алгоритма будут размер генерируемого ландшафта, и число лет, на которое алгоритм будет опираться при выборе числа итераций.

2) Разбиение дна на отдельные плиты

Так как основной элемент дна океана — это литосферные плиты, то сперва формируется дно заданного размера и разбивается случайным образом на литосферные плиты различных размеров. Для случайного разбиения на карте выбираются несколько точек – их количество зависит от реализации, и вокруг них строится диаграмма вороного. На диаграмме вороного точка P принадлежит области O , построенной вокруг точки P_o , если выполняется условие, приведенное в следующей формуле:

$$||P, P_o|| \leq ||P, P_v|| \quad \forall v \in 1..n,$$

где n это число точек, вокруг которых мы строим области диаграммы вороного.

3) Установка высоты и скорости плит

Согласно гипсографической кривой глубина океана достигает 11-и километров, поэтому изначальной высотой мы выберем число в диапазоне от 1000 до 10000 метров, чтобы была возможность расширять рельеф вверх и вниз. Т. к. литосферные плиты в океане не стоят на месте, а движутся и за счет этого происходит образование новых форм рельефа, то каждой плите устанавливается скорость в диапазоне от 2-х до 15-и сантиметров в год.

4) Создание объектов рельефа

На карте случайным образом генерируется каждый из рассмотренных в ходе работы объектов. Сперва определяется количество объектов каждого вида. Далее каждый объект инициализируется, для дальнейшей генерации. При инициализации COX строится маршрут, проходящий между соседними плитами, вдоль которого будет идти симуляция. Инициализацией континентальной окраины является поиск подходящей плиты – эта плита должна быть на краю карты. Единственным ограничением на создание глубоководных котловин с гайотами является то, что они не могут находиться на континентальной окраине, их инициализация представляет собой выбор точки, вокруг которой будет строиться котловина и радиус котловины. Таким образом инициализацию гайота и котловины можно описать следующими формулами:

$$R = \text{random}\left(\frac{B_r}{4}, \frac{B_r}{2}\right), \quad (1)$$

$$C_x = \text{random}(B_x, B_x + B_r - R), \quad (2)$$

$$C_y = \text{random}(B_y, B_y + B_r - R), \quad (3)$$

где B_r – радиус котловины, в которой находится гайот, B_x и B_y – координаты центра котловины, R – радиус гайота, C_x и C_y – координаты центра гайота.

Изначально гайот формируется в пирамиду, т. е. набор вокселей в радиусе гайота, наслоенных друг на друга с уменьшением радиуса в верхушке, т.е:

$$V_z = Z_0 + C * (R - \max(C_x - x, C_y - y)), \forall (x, y) \in G, \quad (4)$$

$$G = \{(x, y) | (x - C_x)^2 + (y - C_y)^2 \leq R^2\},$$

где Z_0 – уровень высоты, на которой расположено основание гайота, R , C_x и C_y получены в формулах (1)-(3), V_z - искомая высота текущей точки из G , константа C – отвечает за множитель начальной высоты гайота относительно его радиуса.

5) Симуляция

Симуляция состоит из повторяющихся шагов, количество шагов зависит от числа лет, заданных при запуске алгоритма, т. е. каждый шаг представляет собой временной промежуток, в течение которого изменяется рельеф дна. При генерации COX, вдоль маршрута полученного при инициализации симулируется раздвижение плит и образование гор и углублений на всем протяжении COX. Процесс формирования COX можно формализовать следующим образом:

$$V_{z,i}(x, y) = V_{z,i-1}(x, y) - (s_i - s_{i-1}), \quad (5)$$

$$s_i = \frac{s(x, y)}{100} \sum_0^{i-1} dt_i, \forall (x, y) \in G,$$

$$G = \{(x_0 + k * dx, y_0 + k * dy) | k \in P(V)\} \forall (x_0, y_0) \in P,$$

где $V_{z,i}(x, y)$ – глубина точки с координатами (x, y) на шаге i , $V_{z,i-1}(x, y)$ – глубина этой точки на предыдущем шаге, $s(x, y)$ – скорость плиты (см/год), которой принадлежит координата (x, y) , dt_i – количество лет, прошедшее с предыдущей итерации, $P(V)$ – множество точек, которые будут смешены вниз в результате образования COX – зависит от реализации, P множество точек COX, dx и dy – смещения вдоль $P(V)$.

Уравнение (4) описывает процесс опускания дна при раздвижении плит. Для формирования горных поверхностей по окраинам COX используется похожая формула, но изменяется знак и область значений:

$$V_{z,i}(x, y) = V_{z,i-1}(x, y) + (s_i - s_{i-1}), \forall (x, y) \in G,$$

$$G = \{(x_0 + k * dx, y_0 + k * dy) | k \in P(V)\} \forall (x_0, y_0) \in P,$$

где $P(V)$ - множество точек, которые будут подняты в результате образования СОХ – зависит от реализации, а все не обозначенные элементы в этой формуле означают то же самое, что и в формуле (5).

Для генерации глубоководной котловины необходимо также сгенерировать и гайоты внутри нее. Сама котловина представляет собой поверхность круглой формы, которая со временем опускается все глубже относительно остального ландшафта. На очередном шаге глубина точки вычисляется следующим образом:

$$V_{z,i} = V_{z,i-1} - s_i, \forall V_z \in G,$$

где $V_{z,i}$ – искомая глубина на текущем шаге, $V_{z,i-1}$ – глубина текущей точки на предыдущем шаге, G – множество точек, лежащих в котловине, i – номер текущего шага, а s_i – размер текущего сдвига, который вычисляется по формуле:

$$s_i = \frac{(\sum_{j=0}^{i-1} dt_j)}{t_s} - \sum_{j=0}^{i-1} s_j, \quad (6)$$

где dt_j – количество лет, прошедшее с предыдущего вычисления, а t_s константа, обозначающая за сколько лет требуется для смещения, $s_0 = 0$ – начальное смещение. Т. е. на каждом шаге вычисляется смещение относительно уже произведенных смещений на предыдущих итерациях.

Гайот изначально представляет собой фигуру пирамидальной формы, верхушка которого стачивается с течением времени, соответственно уменьшая его высоту и сам он погружается вместе с основанием котловины. Процесс абразии описывается следующей формулой:

$$V_{z,i} = V_{z,i-1} - \begin{cases} dz, & \text{если } V_{z,i-1} = Z_i + H_i \\ 0, & \text{иначе} \end{cases},$$

где $Z_i = Z_0 - i$ и $H_i = 2 * R - i$ – текущая глубина основания и высота гайота соответственно, Z_0 и R имеют то же значение, что и в формуле (4), а dz обозначает на сколько нужно сместить текущую точку и зависит от реализации.

Генерацией континентальной окраины является ее постепенное «погружение под материк», т. к. материковая часть не рассматривается в данной работе, то погружение будет происходить только окраинной части океана и представлять собой загибание ландшафта на краю плиты. Процесс субдукции может быть формализован следующим образом:

$$V_{z,i}(x, y) = V_{z,i-1}(x, y) + (s_i - s_{i-1}), \forall (x, y) \in G,$$

$$G = \{ (x_0 + i * dx, y_0 + i * dy) | k = 0..s_i \}, \forall (x_0, y_0) \in P,$$

где $V_{z,i}(x, y)$ – высота точки с координатами (x, y) на шаге i , $V_{z,i-1}(x, y)$ – высота этой же точки на шаге $i - 1$, s_i – взято из формулы (6), P – множество точек лежащих на краю карты, а также dx и dy – смещения вдоль границы плиты.

6) Сохранение ландшафта

Когда все итерации выполнены сгенерированный рельеф сохраняется в файл для последующей визуализации.

2.3 Архитектура алгоритма

При разработке архитектуры алгоритма учитывалось, что в будущем в него могут быть добавлены новые элементы ландшафта. На основе этого было разработано несколько классов:

- 1) Класс Generator – главный класс, отвечающий за процесс подготовки карты к генерации ландшафта, хранению воксельных данных, и итерации в процессе генерации.
- 2) Класс Voxel – данный класс представляет собой минимальный элемент пространства, из которого строится ландшафт.
- 3) Класс Plate – данный класс содержит информацию о литосферной плите.
- 4) Интерфейс LandscapeElement – этот интерфейс разработан для поддержания добавления новых элементов ландшафта, и он довольно прост – для соответствия данному интерфейсу классам нужно реализовать всего два метода в которых происходит инициализация и процесс пошаговой генерации соответственно.

- 5) Классы DeepSeaBasin, Guyot, MidOceanRidge и ContinentalMargin – представляют собой элементы ландшафта: глубоководная котловина, гайот, СОХ и континентальная окраина соответственно.
- 6) Класс VoxWriter – класс, отвечающий за запись сгенерированного ландшафта в файл в формате vox.

Класс генератор создает карту вокселей заданного пользователем размера, затем разбивает ее на части, устанавливает свойства плит и генерирует элементы ландшафта, после чего запускает процесс итерации, в котором каждому элементу рельефа по очереди передается управление, чтобы он сгенерировал очередную порцию «себя». По окончании процесса итерации, длительность которого зависит от количества лет, введенного пользователем, полученная карта со всеми изменениями передается классу VoxWriter, который генерирует выходной файл в формате vox, для отображения в MagicalVoxel. На рисунке 13 представлена диаграмма классов и связей между ними.

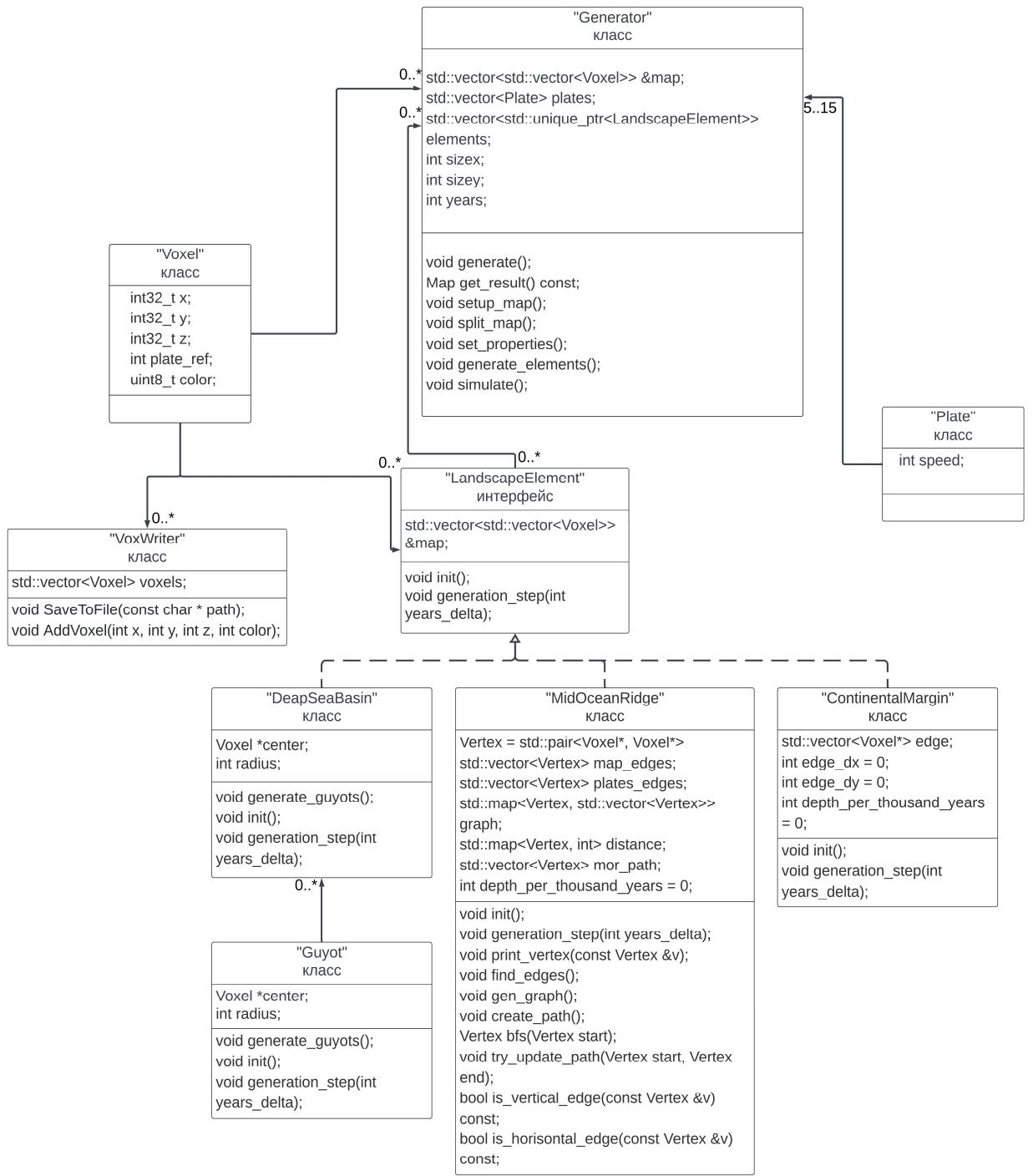


Рисунок 13 - ER диаграмма классов в алгоритме

3 ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ РЕАЛИЗАЦИИ

3.1 Способ представления данных

В ходе исследовательской части работы выяснилось, что существуют две основные группы данных, представленных картой высот и объемными данными. Каждый из способов имеет свои достоинства и недостатки, однако для реализации данного алгоритма было выбрано волюметрическое представление по следующим причинам:

- 1) Простота использования – т. к. воксели представляют собой отдельные единицы пространства, то их перемещение, добавление или удаление являются простыми, с точки зрения реализации, операциями.
- 2) Снижение требований к памяти – одна из главных проблем воксельной графики – высокое потребление памяти, однако современные вычислительные системы располагают большим объемом памяти и это число постоянно растет. Никого уже не удивишь восемью гигабайтами ОЗУ на смартфоне, а в домашних системах можно встретить 64 и более гигабайт. К тому же одной из первых воксельных игр считается «Comanche: Maximum Overkill» - она вышла в 1992 году и уже тогда компьютеры были способны справиться с волюметрическими данными.
- 3) Возможности оптимизации – несмотря на то, что проблема потребления памяти постепенно уходит в прошлое, она все еще может быть актуальна для старых устройств. Решением данной проблемы являются различные структуры данных, которые позволяют хранить воксели более компактно. Одной из самых первых, известных и распространённых структур является разреженное воксельное октодерево [19]. Однако помимо данной структуры позже появились разреженные воксельные DAG высокого разрешения [20] и SSVDAg, который позволяют уменьшить расход памяти в 55 раз по сравнению с SVO [21].

4) Возможности взаимодействия – помимо простоты использования во время генерации, подход с использованием воксельных представлений дает большой простор для взаимодействия с полученным ландшафтом уже на этапе использования в конечном продукте. С использованием воксельной графики можно добиться высокой динамичности окружения – его разрушаемости или модификации. Как пример можно привести нашумевшую игру «Teardown» или уже много лет удерживающую популярность «Minecraft». На рисунке 10 представлен пример разрушаемости в «Teardown», а на рисунке 11 представлен пример модификации ландшафта в «Minecraft».



Рисунок 60 - Благодаря вокселям в игре может быть достигнута невероятная разрушаемость.



Рисунок 11 - Пример модификации мира Minecraft

На основании приведенных данных можно заключить, что использование вокселей на современных компьютерах не является проблемой и при этом несет в себе дополнительные преимущества по сравнению с традиционным представлением. Скорее всего в ближайшем будущем их применение будет распространяться все сильнее.

3.2 Способ визуализации

Для визуализации сгенерированного ландшафта было выбрано свободно распространяемое ПО MagicalVoxel, позволяющее бесплатное использование в любых типах проектов. MagicalVoxel – это воксельный редактор и рендер движок, он поддерживает анимации и уже успел зарекомендовать себя в работах по генерации воксельных ландшафтов [22]. Для описания воксельного

контента данный движок использует собственный формат файлов с расширением «.vox», он основан на RIFF формате, разработанном компаниями Microsoft и IBM для хранения данных. Плюсом MagicalVoxel является то, что он позволяет экспортить полученный «vox» файл в другие форматы, например «obj», которые широко используются во всех современных графических приложениях, например движках «Unity» и «Unreal Engine», а также его простота по сравнению с этими движками. Интерфейс MagicalVoxel приведен на рисунке 12.

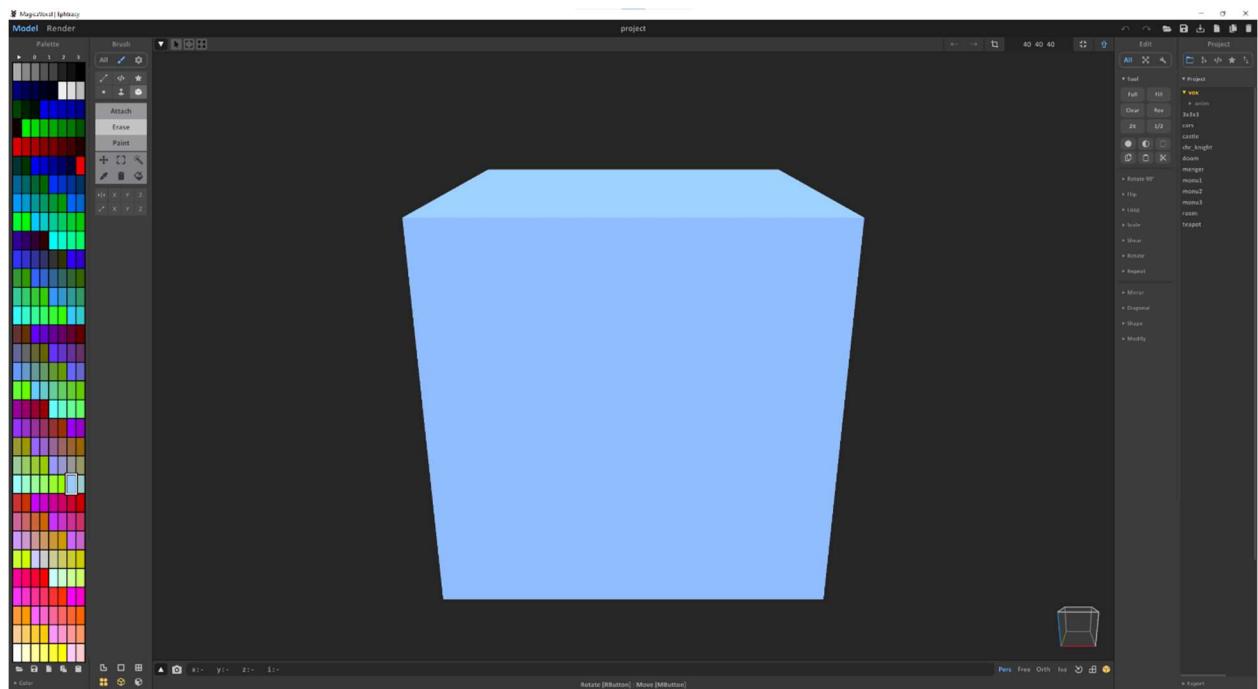


Рисунок 12 - Интерфейс движка Magical Voxel очень прост по сравнению с Unity или Unreal Engine

3.2.1 Описание формата vox

Бинарный файл формата «vox» представляет собой набор частей, называемых чанками, каждый из которых описывает свое содержимое и содержимое вложенных в нее частей, если таковые имеются. Данное описание взято с гитхаба разработчиков [23].

Содержимое данного файла начинается с четырехбайтовой строки “vox”, за которой следует четырехбайтовое число 150, обозначающие, тип и версию формата файла. Первой частью файла является чанк с идентификатором «MAIN», его описание приведено в таблице 1.

Таблица 1 - описание чанка "MAIN"

Количество байт	Значение
4	Идентификатор чанка
4	Количество байт содержимого (N)
4	Количество байт содержимого детей (M)
N	Содержимое чанка
M	Содержимое детей

Содержимое чанка «MAIN» представляет собой набор из следующей последовательности частей: первым идет необязательный чанк с идентификатором «PACK», затем идет одна или более групп из двух чанков с идентификаторами «SIZE» и «XYZI» соответственно, а завершается все необязательным чанком с идентификатором «RGBA».

Часть «PACK» представляет собой четырехбайтовый идентификатор и еще одно четырехбайтовое число, обозначающее количество моделей в данном файле. При его отсутствии предполагается, что в файле записана одна модель.

Далее идет набор пар чанков «SIZE» и «XYZI», описывающие модели, представленные в файле. «SIZE» представляет собой четырехбайтовый идентификатор и три четырехбайтовых числа, описывающих размер модели по осям x, у и z соответственно. Затем идет описание самой модели, представляющее собой четырехбайтовый идентификатор «XYZI», за которым следует четырехбайтовое число N, обозначающее количество вокселей в данной модели и затем N четырехбайтных векторов, каждый вектор представляет собой позицию x, y, z по одному байту плюс однобайтовое число i, которое представляет собой индекс в таблице цветов из чанка «RGBA».

«RGBA» чанк является не обязательным, при его отсутствии будет использована стандартная таблица цветов, предусмотренная рендер движком. Сам по себе он представляет собой идентификатор, за которым следует 256 четырехбайтных чисел, описывающих цвет в формате RGBA – по одному байту на каждую составляющую.

Сам по себе формат «квох» является довольно простым, однако при этом он позволяет визуализировать большое количество различных данных, практически никак не ограничивая пользователей.

3.3 Реализация алгоритма

Для реализации алгоритма, описанного в предыдущей части, был выбран язык C++. Выбор языка основывался на нескольких принципах. Во-первых, C++ известен своей производительностью, его использование позволит быстро генерировать ландшафт, что может быть критично в некоторых приложениях. Во-вторых, этот язык широко распространен в мире компьютерной графики, а значит его использование потенциально расширяет возможности применения алгоритма – например его будет легче встраивать в другие приложения, связанные с графикой. В-третьих, автору этот язык знаком более всего, поэтому сокращается вероятность внесения ошибок в алгоритм, а также время написания программы.

Реализация алгоритма состоит из нескольких основных частей: получение ввода пользователя, генерация данных о рельефе и запись результата в файл. Наиболее важной и интересной частью является процесс генерации, поэтому он будет рассмотрен наиболее подробно.

3.3.1 Задание исходных данных

От пользователя на вход алгоритм получает четыре параметра: `sizex` – длина карты по оси x, `sizey` – длина карты по оси y, `years` – время генерации ландшафта и `output` – имя файла для сохранения результата.

3.3.2 Разбиение дна на отдельные плиты

Первым делом алгоритм генерирует карту вокселей размера `sizex * sizey`. Далее на этой карте строится разбиение на литосферные плиты. Для этого генерируется случайное число в диапазоне от 5-и до 15-и, эти числа были подобраны на практике, так разбиение получается не сильно раздробленным, но количество плит кажется достаточным. Далее на карте генерируются точки (x, y), в соответствии с количеством плит. По этим картам строится диаграмма вороного [24]. На диаграмме вороного точка Р принадлежит области О,

построенной вокруг точки P_o , если выполняется условие, приведенное в следующей формуле:

$$|P.x - P_o.x| + |P.y - P_o.y| \leq |P.x - P_v.x| + |P.y - P_v.y| \forall v \in 1..n,$$

Где n это число точек, вокруг которых мы строим области диаграммы вороного. Получившиеся в результате разбиения области есть искомые литосферные плиты. Также на этом этапе каждому voxelю присваивается цвет, в соответствии с номером плиты, для визуального различия плит.

Пример разбиения на плиты приведен на рисунке 14.

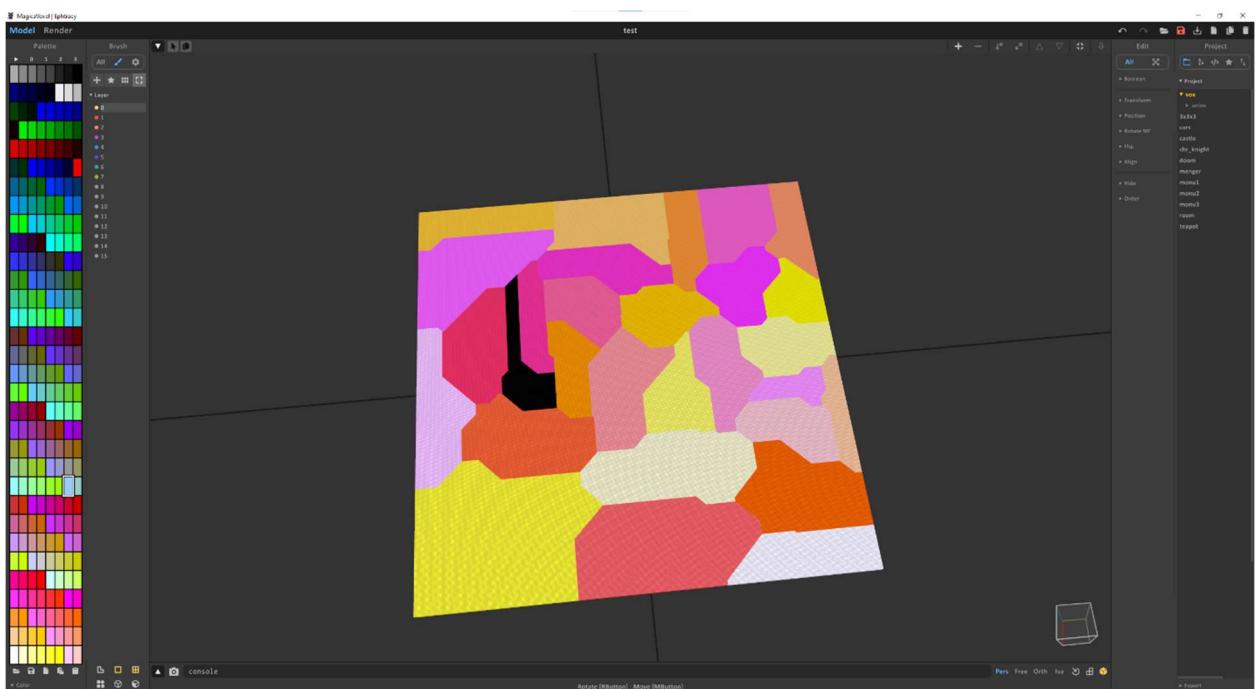


Рисунок 14 - Пример разбиения карты на плиты с помощью диаграммы вороного. Плиты раскрашены разным цветом для визуального различия

3.3.3 Установка высоты и скорости плит

Из анализа, проведенного в исследовательской части работы, был сделан вывод, что высоту плит следует установить в диапазоне от 1000 до 10000 метров – для этого просто выбираем случайное число из этого промежутка и устанавливаем его всем voxelям, со скоростью тоже самое, однако число из промежутка от 2-х до 15-и сантиметров в год устанавливается не voxelям по отдельности, а плите в целом.

3.3.4 Создание объектов рельефа

Исходя из аналитических данных видно, что COX очень протяженный объект, поэтому было решено генерировать не более одного COX на карте. Для генерации COX выбирается случайное число 0 или 1 – будет он присутствовать на ландшафте или нет.

Количество континентальных окраин ограничено двумя, т. к. расстояние между континентами очень велико.

Аналитические данные не накладывают ограничений на глубоководные котловины, однако, чтобы карта не выглядела перегруженной, их количество ограничено количеством плит. Количество гайотов внутри глубоководной котловины зависит от ее радиуса – чем больше радиус, тем больше гайотов может поместиться внутри одной котловины.

После создания каждого элемента происходит его инициализация.

3.3.5 Симуляция

Симуляция начинается с инициализации созданных элементов.

3.3.5.1 Инициализация глубоководной котловины и гайота

Инициализация гайота и глубоководной котловины представляет собой просто сохранение данных об их положении и радиусе. Место положения котловины определяется выбором случайной точки на карте, радиусом также является случайное число. Место положения гайота определяется случайной точкой внутри радиуса котловины, радиусом гайота является случайное число от четверти до половины радиуса котловины. Центр и радиус каждого гайота рассчитываются по следующим формулам (1) – (3).

Изначально гайот формируется в пирамиду, т. е. набор вокселей в радиусе гайота, наслоенных друг на друга с уменьшением радиуса в верхушке, т.е:

$$V_z = Z_0 + 2 * (R - \max(C_x - x, C_y - y)), \forall (x, y) \in G, \quad (7)$$

$$G = \{(x, y) | (x - C_x)^2 + (y - C_y)^2 \leq R^2\},$$

где Z_0 – уровень высоты, на которой расположено основание гайота, R , C_x и C_y получены в формулах (1)-(3), V_z - искомая высота текущего вокселя из

G , константа 2 – отвечает за множитель начальной высоты гайота относительно его радиуса.

3.3.5.2 Инициализация COX

Наиболее сложной является инициализация COX - для его инициализации нам требуется найти путь между литосферными плитами, вдоль которого будет проходить симуляция, при этом мы хотим, иметь достаточно длинный путь, чтобы COX выглядел более реалистично. Для поиска пути сперва определим какие воксели могут ему принадлежать. Для простоты, т. к. путь ищется на двумерной карте, в описании его поиска мы будем использовать слова воксель и точка как синонимы.

Элементом пути мы назовем пару точек, являющихся граничными точками литосферных плит, т. е. таких точек, которые находятся на плите с номером x и имеют своим соседом точку, находящуюся на плите с номером y , при этом $x \neq y$. Мы называем точки соседями, если они имеют общую границу по горизонтали или по вертикали. Крайним элементом пути мы назовем такой элемент пути, который лежит на краю карты, т. е. обе точки которого имеют одинаковый x или y , при этом одинаковая координата является либо нулем, либо числом, равным размеру карты по этой координате. Например, для карты размера $4*4$ пара точек $(3,1)$ и $(4,1)$ является крайним элементом пути, а пара точек $(3,2)$ и $(4,2)$ не является (ось x направлена слева на право, а ось y сверху вниз). Пример получившегося множества точек, которые могут быть на пути COX на карте из двух плит изображен на рисунке 15.

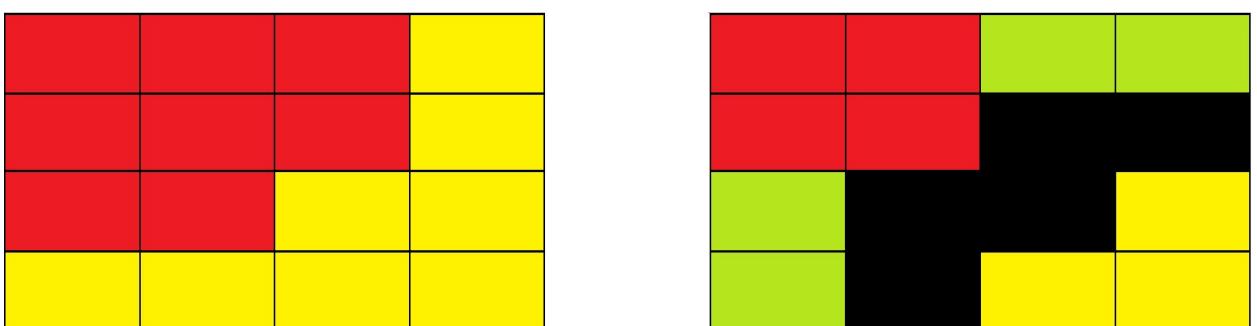


Рисунок 15 - Две плиты (слева) и те же плиты, на которых черным выделены элементы пути, а зеленым крайние элементы пути.

Для поиска таких точек достаточно просмотреть все voxели на карте и сохранить такие пары, в которых выполняется условие для элемента пути и первой парой в точке является наименьшая из двух точек. Точка А считается меньше точки В, если $A.x < B.x$ или $A.x = B.x$ и $A.y < B.y$. Второе требование нужно, чтобы исключить одинаковые пары точек, различающиеся только порядком внутри пары. Одновременно с поиском элементов путей мы запоминаем крайние элементы пути.

После того, как все элементы пути найдены нам нужно построить граф, на котором уже сможем проложить маршрут для COX. В данном графе элементы пути будут являться узлами. Для построения графа на элементах пути рассмотрим все пары таких элементов. Назовем элемент пути вертикальным, если обе его точки имеют одинаковую x координату, горизонтальным элементом пути мы будем называть, если обе его точки имеют одинаковую y координату.

Между парой элементов пути мы добавляем ребро, если они оба представляют вертикальный элемент пути и координаты их меньших точек одинаковы по x и различаются на 1 по y, либо они оба представляют горизонтальный элемент пути и координаты их меньших точек одинаковы по y и различаются на 1 по x, либо один из них горизонтальный, а другой вертикальный, но при этом они имеют одну общую точку. На рисунке 16 представлены элементы пути, между которыми будет добавлено ребро.



Рисунок 16 - Элементы пути, между которыми будет добавлено ребро.
Черный и красный - вертикальные, желтый и зеленый - горизонтальные, оранжевый и голубой - горизонтальный и вертикальный. последний элемент может быть повернут на 90, 180 и 270 градусов.

Когда граф построен нам достаточно запустить поиск в ширину [25] из каждого крайнего элемента пути и выбрать самый длинный путь до другого крайнего элемента пути. Таким образом мы получим маршрут

протяженностью через всю карту без самопересечений. Пример построенного маршрута приведен на рисунке 17.

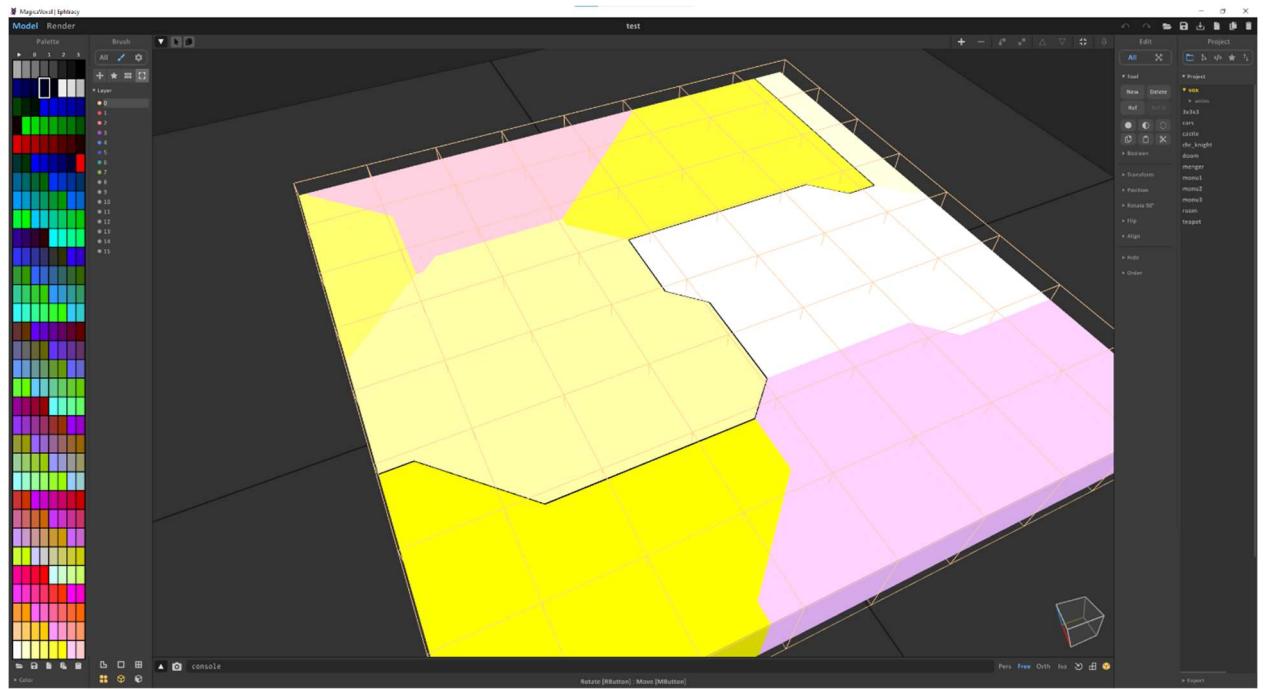


Рисунок 17 - Путь COX между плитами (выделен черным)

3.3.5.3 Инициализация континентальной окраины

Для инициализации континентальной окраины нам нужно выбрать одну из плит, которая будет ей являться и построить на этой плите путь, вдоль которого будет происходить процесс субдукции, а также поднять эту плиту, чтобы она выделялась на фоне остальных. Для этого на карте выбирается случайная точка, а затем выбирается ближайший к этой точке край карты. После выбирается плита, которая содержит проекцию точки на выбранный край карты. Например, пусть мы имеем карту размера $10 * 10$ и выбрали случайную точку с координатами $(6, 7)$. Тогда ближайшим к ней краем карты будет край, лежащий на прямой $y = 10$. А выбрана будет плита, которая содержит точку $(6, 10)$. Для построения пути достаточно пройти вдоль зафиксированной прямой и сохранить все воксели, которые лежат на выбранной плите.

3.3.5.4 Генерация элементов

Временным шагом для симуляции был выбран отрезок в 100 лет – за это время литосферная плита пройдет путь от двух до пятнадцати метров и будет заметен прогресс в формировании рельефа.

Формирование глубоководной котловины происходит путем опускания ее основания, т. е. уменьшения координаты z у всех вокселей, находящихся в радиусе котловины. На очередном шаге глубина вокселя вычисляется следующим образом:

$$V_{z,i} = V_{z,i-1} - s_i, \forall V_z \in G,$$

где $V_{z,i}$ – искомая глубина на текущем шаге, $V_{z,i-1}$ – глубина текущего вокселя на предыдущем шаге, G – множество вокселей, лежащих в котловине, i – номер текущего шага, а s_i – размер текущего сдвига, который вычисляется по формуле:

$$s_i = \frac{(\sum_{j=0}^{i-1} dt_j)}{t_s} - \sum_{j=0}^{i-1} s_j, \quad (8)$$

где dt_j – количество лет, прошедшее с предыдущего вычисления, а $t_s = 2000$ – константа, обозначающая за сколько лет происходит сдвиг на 1 воксель, $s_0 = 0$ – начальное смещение. Т. е. на каждом шаге вычисляется смещение относительно уже произведенных смещений на предыдущих итерациях.

Формирование гайотов внутри котловин происходит путем абразии, т. е. затирания острия верхушки и постепенного стачивания гайота. Процесс абразии описывается следующей формулой:

$$V_{z,i} = V_{z,i-1} - \begin{cases} 1, & \text{если } V_{z,i-1} = Z_i + H_i \\ 0, & \text{иначе} \end{cases},$$

где $Z_i = Z_0 - i$ и $H_i = 2 * R - i$ – текущая глубина основания и высота гайота соответственно, Z_0 и R имеют то же значение, что и в формуле (7).

Формирование СОХ происходит за счет рифтогенеза – на каждой итерации плиты раздвигаются все дальше друг от друга и между ними образуется все большее углубление. На краях плит вдоль маршрута СОХ образуются горы за счет выброса магмы на поверхность. Т. е. воксели между

плитами на каждой итерации опускаются, при этом чем дальше от края плиты, тем сильнее будет погружен воксель. Построение гор на окраинах происходит наоборот, чем дальше от края – тем выше точка. Процесс формирования COX можно formalизовать следующим образом:

$$V_{z,i}(x, y) = V_{z,i-1}(x, y) - (s_i - s_{i-1}), \quad (9)$$

$$s_i = \frac{s(x,y)}{100} \sum_0^{i-1} dt_i, \forall (x, y) \in G,$$

$$G = \{ (x_j + k * (-1)^j * dx, y_j + k * (-1)^j * dy) \mid k = 0..s_i \},$$

$$(x_j, y_j) \in dp, j = 1..2, \forall dp \in P$$

где $V_{z,i}(x, y)$ – глубина вокселя с координатами (x, y) на шаге i , $V_{z,i-1}(x, y)$ – глубина этого вокселя на предыдущем шаге, $S(x, y)$ – скорость плиты (см/год), которой принадлежит координата (x, y) , dt_i – количество лет, прошедшее с предыдущей итерации, dp – элемент пути состоящий из 2х вокселей, P – множество всех элементов пути COX,

$$(dx, dy) = \begin{cases} (0, 1), & \text{если } dp \text{ – горизонтальный элемент} \\ (1, 0), & \text{если } dp \text{ – вертикальному элементу} \end{cases}$$

Уравнение (9) описывает процесс опускания вокселей при раздвижении плит. Для формирования горных поверхностей по окраинам COX используется похожая формула, но изменяется знак и область значений:

$$V_{z,i}(x, y) = V_{z,i-1}(x, y) + (s_i - s_{i-1}), \forall (x, y) \in G,$$

$$G = \{ (x_j + (k + s_i) * (-1)^j * dx, y_j + (k + s_i) * (-1)^j * dy) \mid k = 0..s_i/4 \},$$

все не обозначенные элементы в этой формуле означают то же самое, что и в формуле (9).

Для формирования континентальной окраины мы симулируем процесс субдукции – на каждой итерации плита все глубже погружается под материк. Реализовано это достаточно просто – мы постепенно опускаем воксели, находящиеся на краю карты, и с каждой итерацией опускание затрагивает все большую часть плиты, этот процесс выражен следующей формулой:

$$V_{z,i}(x, y) = V_{z,i-1}(x, y) - (s_i - s_{i-1}), \forall (x, y) \in G,$$

$$G = \{ (x_0 + i * dx, y_0 + i * dy) | k = 0..s_i \}, \forall (x_0, y_0) \in P,$$

где $V_{z,i}(x, y)$ – высота вокселя с координатами (x, y) на шаге i , $V_{z,i-1}(x, y)$ – высота вокселя на шаге $i - 1$, s_i – взято из формулы (8), P – множество точек, лежащих на краю карты, а также

$$dx = \begin{cases} 1, & \text{если } x = 0 \quad \forall (x, y) \in P \\ -1, & \text{если } x = size_x \quad \forall (x, y) \in P, \\ 0, & \text{иначе} \end{cases}$$

$$dy = \begin{cases} 1, & \text{если } y = 0 \quad \forall (x, y) \in P \\ -1, & \text{если } y = size_y \quad \forall (x, y) \in P, \\ 0, & \text{иначе} \end{cases}$$

3.3.6 Сохранение ландшафта

Когда все итерации выполнены сгенерированный рельеф с помощью класса VoxWriter сохраняется в заданный пользователем файл для последующей визуализации.

3.3.7 Блок-схема алгоритма

Для более подробного представления о работе алгоритма на рисунках 18–23 приведены блок-схемы алгоритма и его подпрограмм.

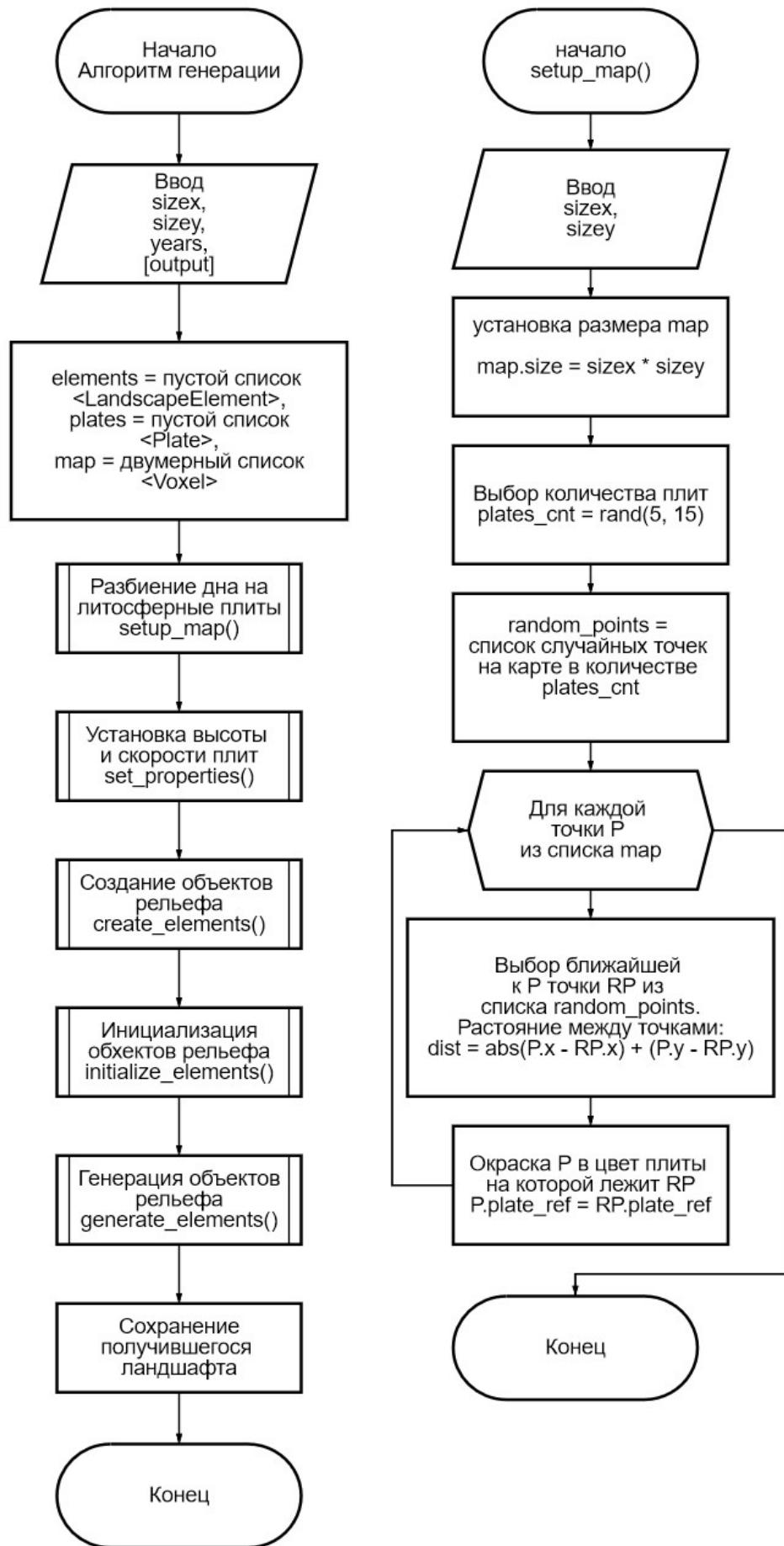


Рисунок 18 - Общий алгоритм работы и процедура setup_map

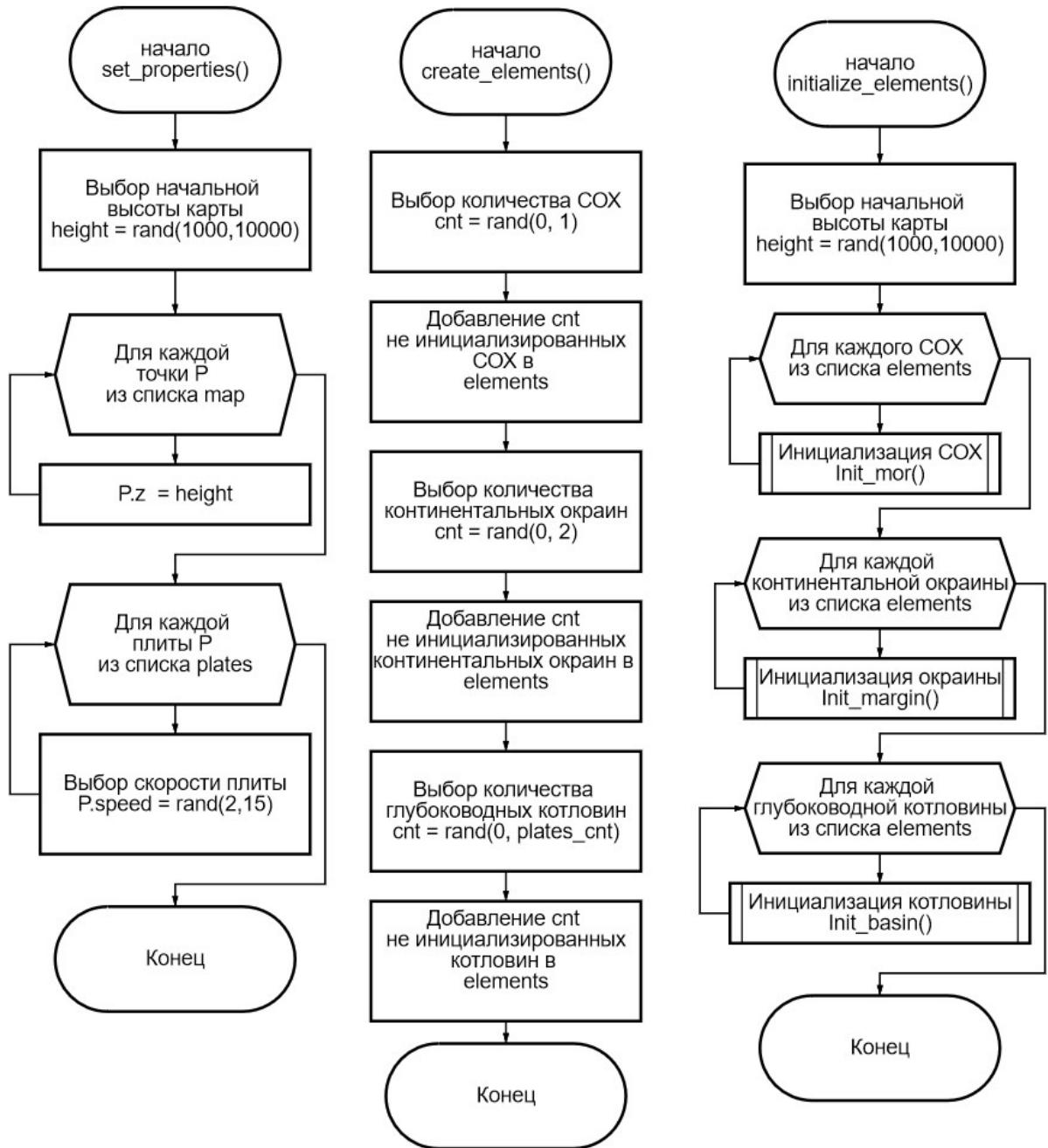


Рисунок 19 - схемы работы функций `set_properties`, `create_elements` и `initialize_elements`

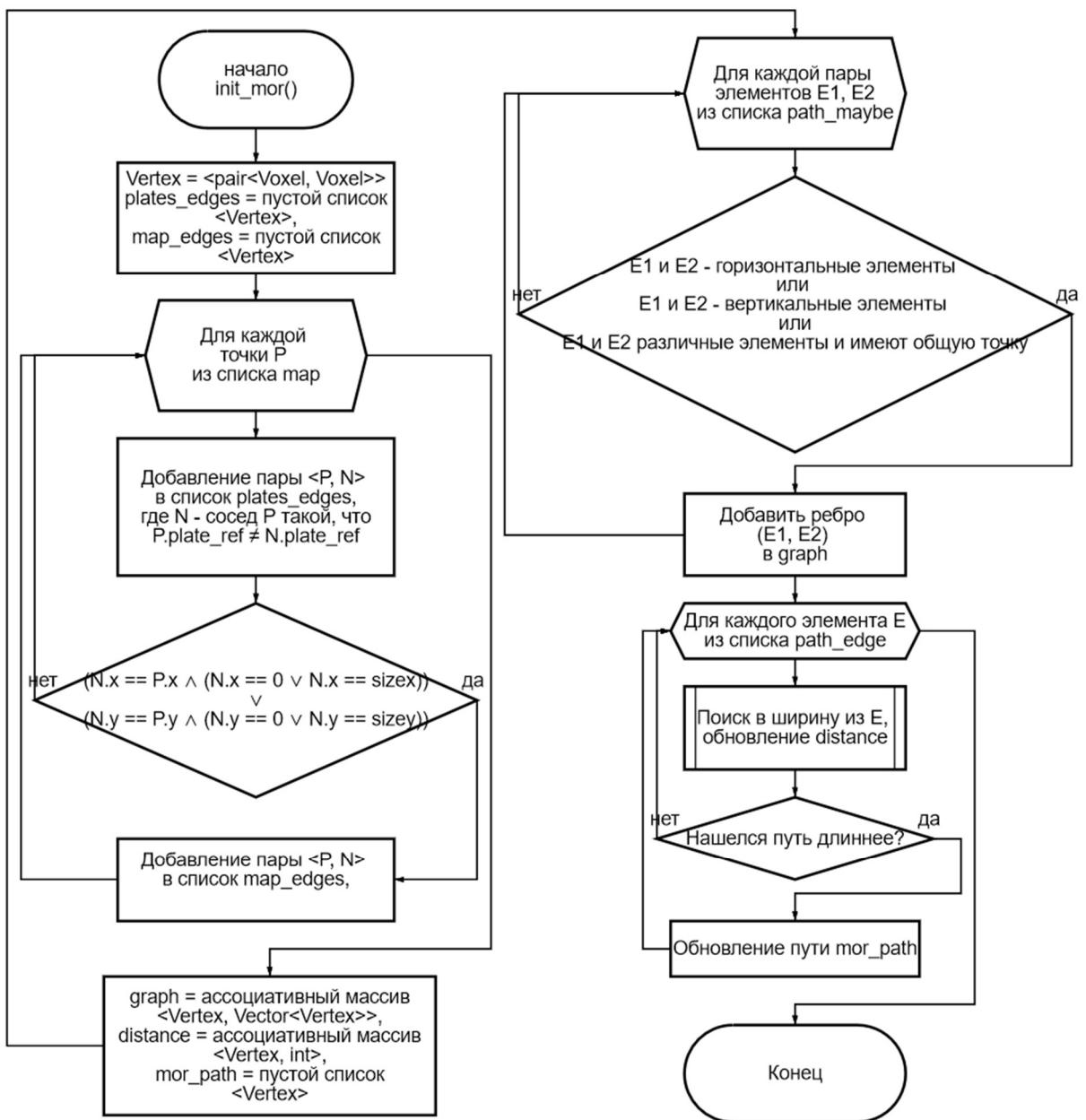


Рисунок 20 - Инициализация СОХ

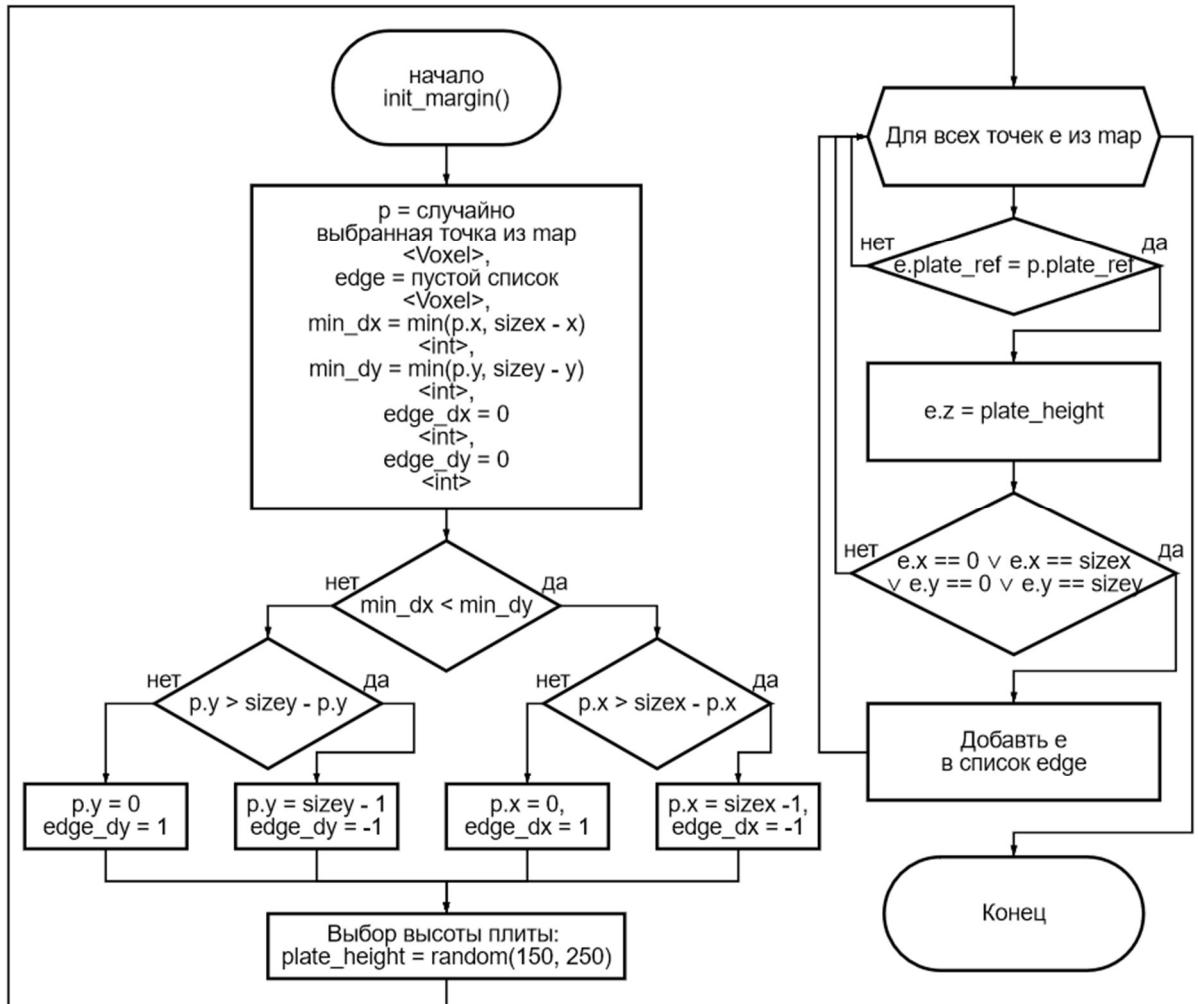


Рисунок 21 - инициализация континентальной окраины

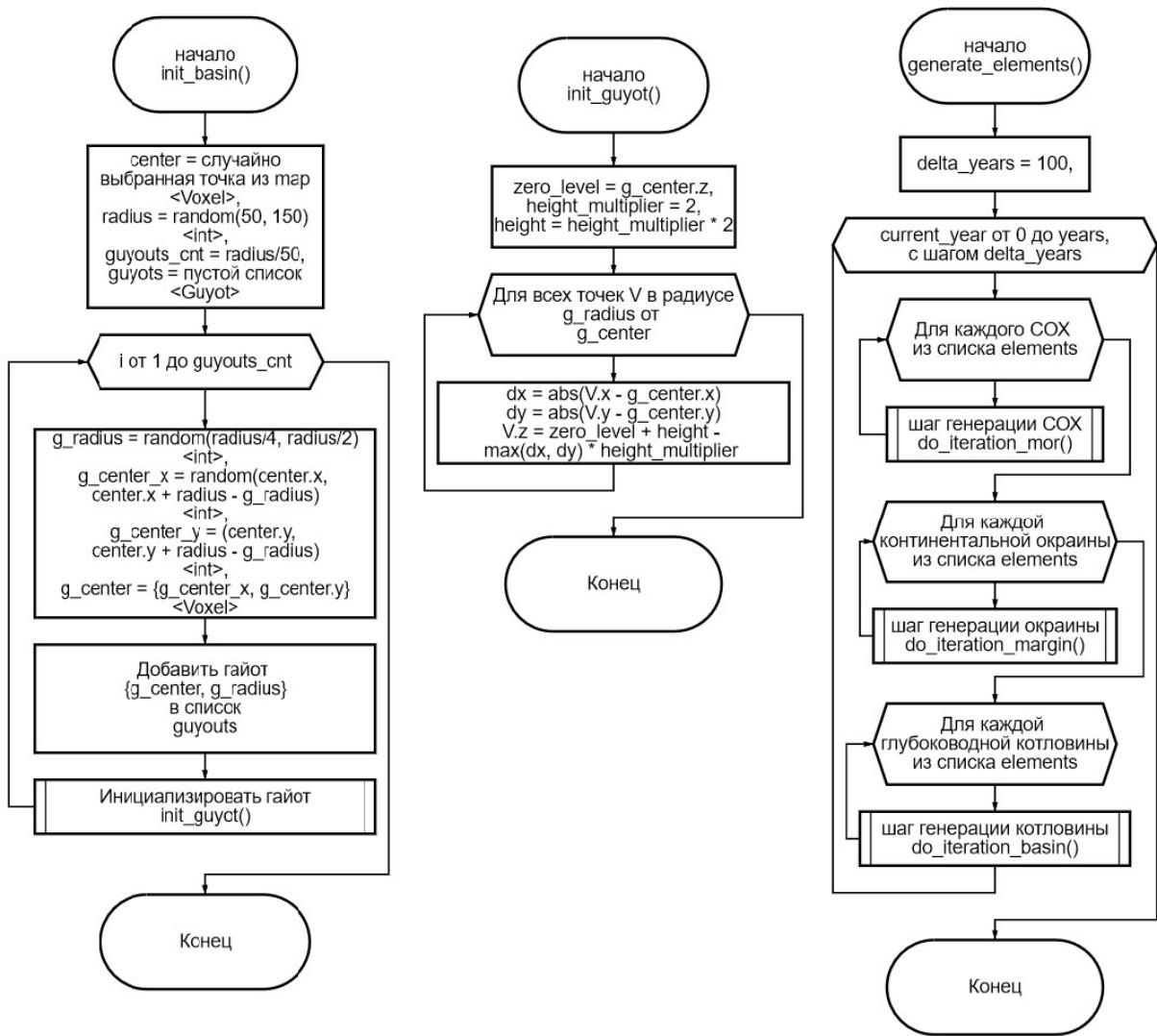


Рисунок 22 - Инициализация глубоководной котловины и гайота, а также генерация элементов

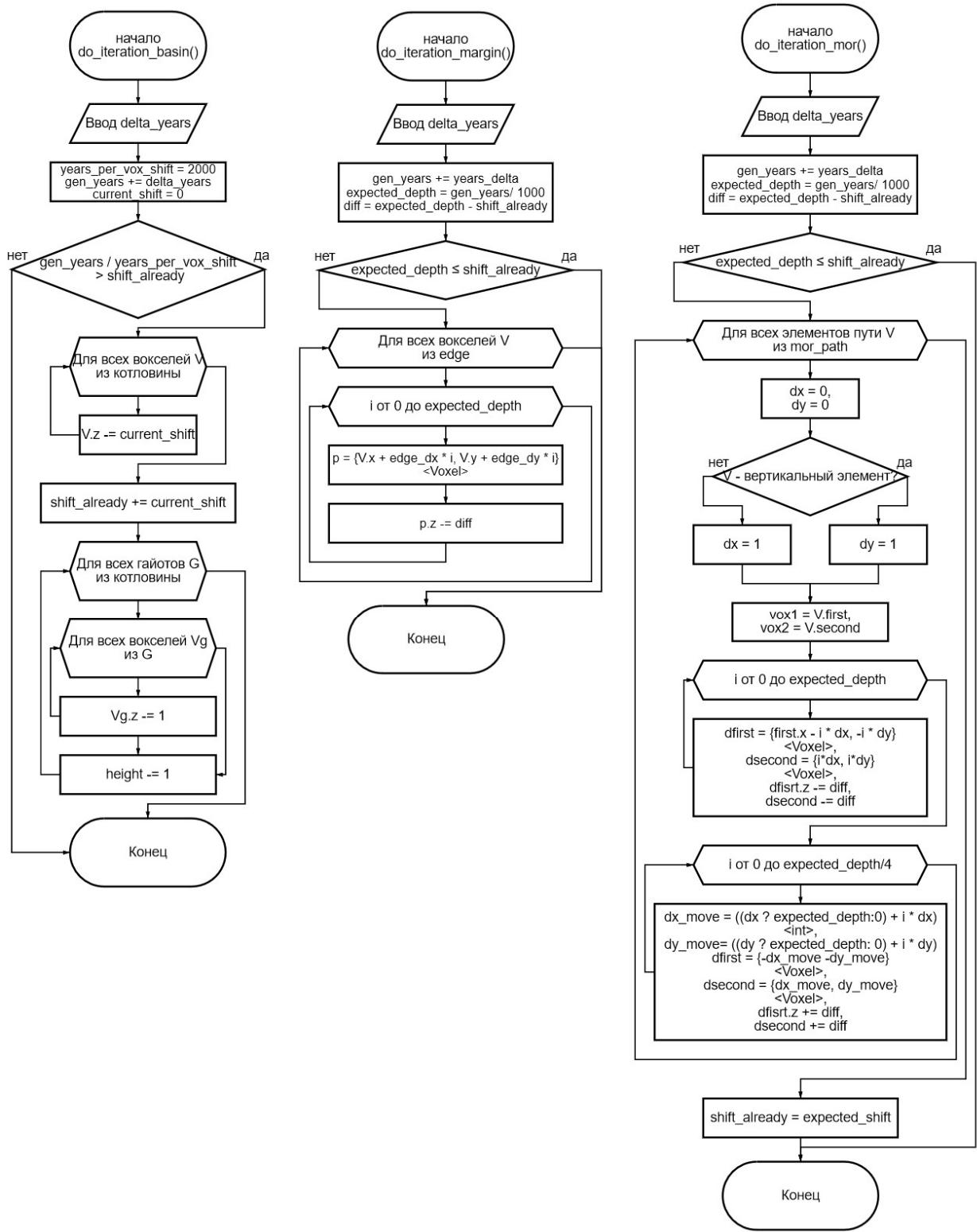


Рисунок 23 - Процессы генерации каждого элемента рельефа

3.3.8 Результаты работы алгоритма

Для отделения литосферных плит друг от друга все они покрашены в разные цвета, также генерируемые элементы тоже выделены разными цветами. На рисунке 24 представлен результат глубоководной котловины.

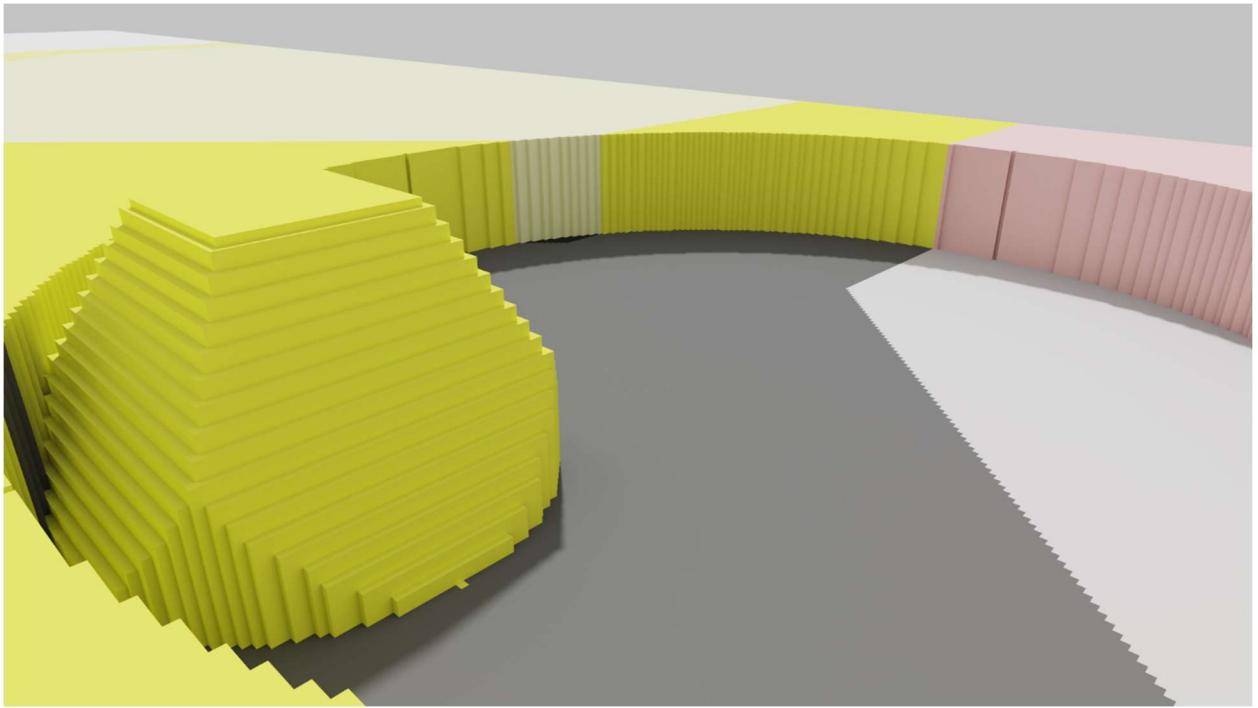


Рисунок 24 - гайот внутри глубоководной котловины

На рисунке 25 показаны результаты формирования континентальной окраины, а на рисунке 26 изображен результат формирования срединно-океанического хребта. Рисунки 27 и 28 показывают результат формирования всех элементов сразу на одной карте.

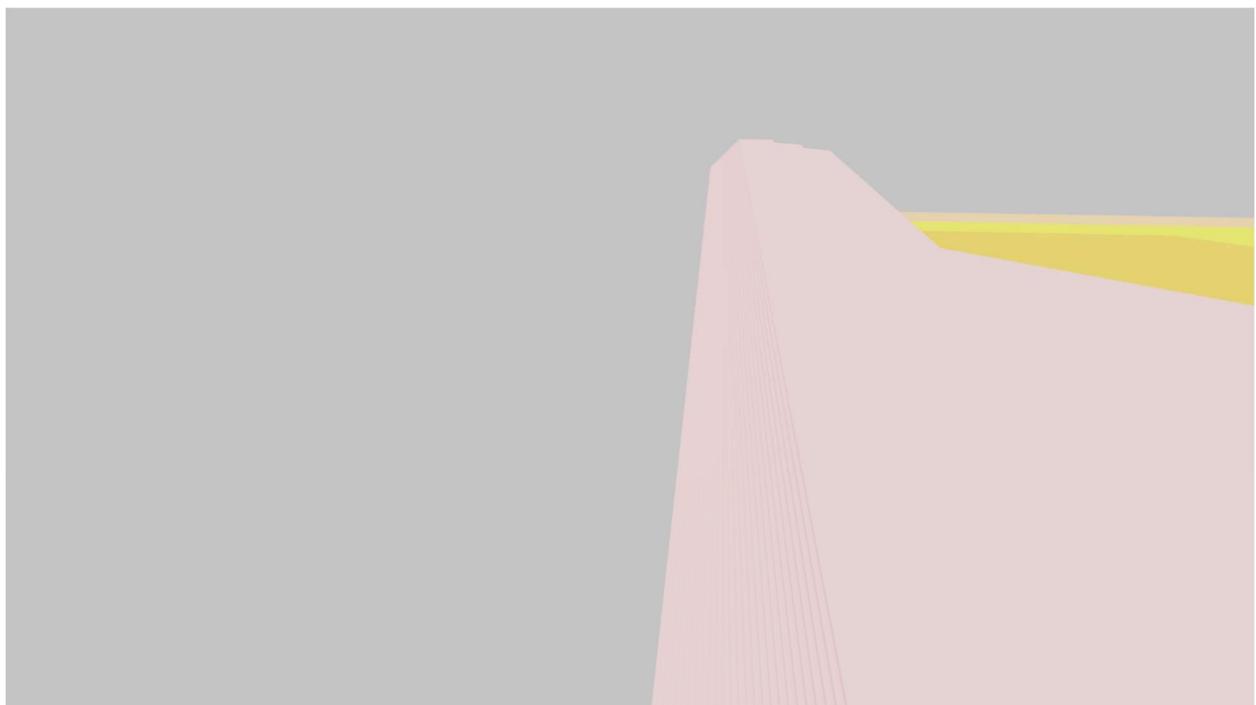


Рисунок 25 - континентальная окраина. Край литосферной плиты постепенно опускается под материк

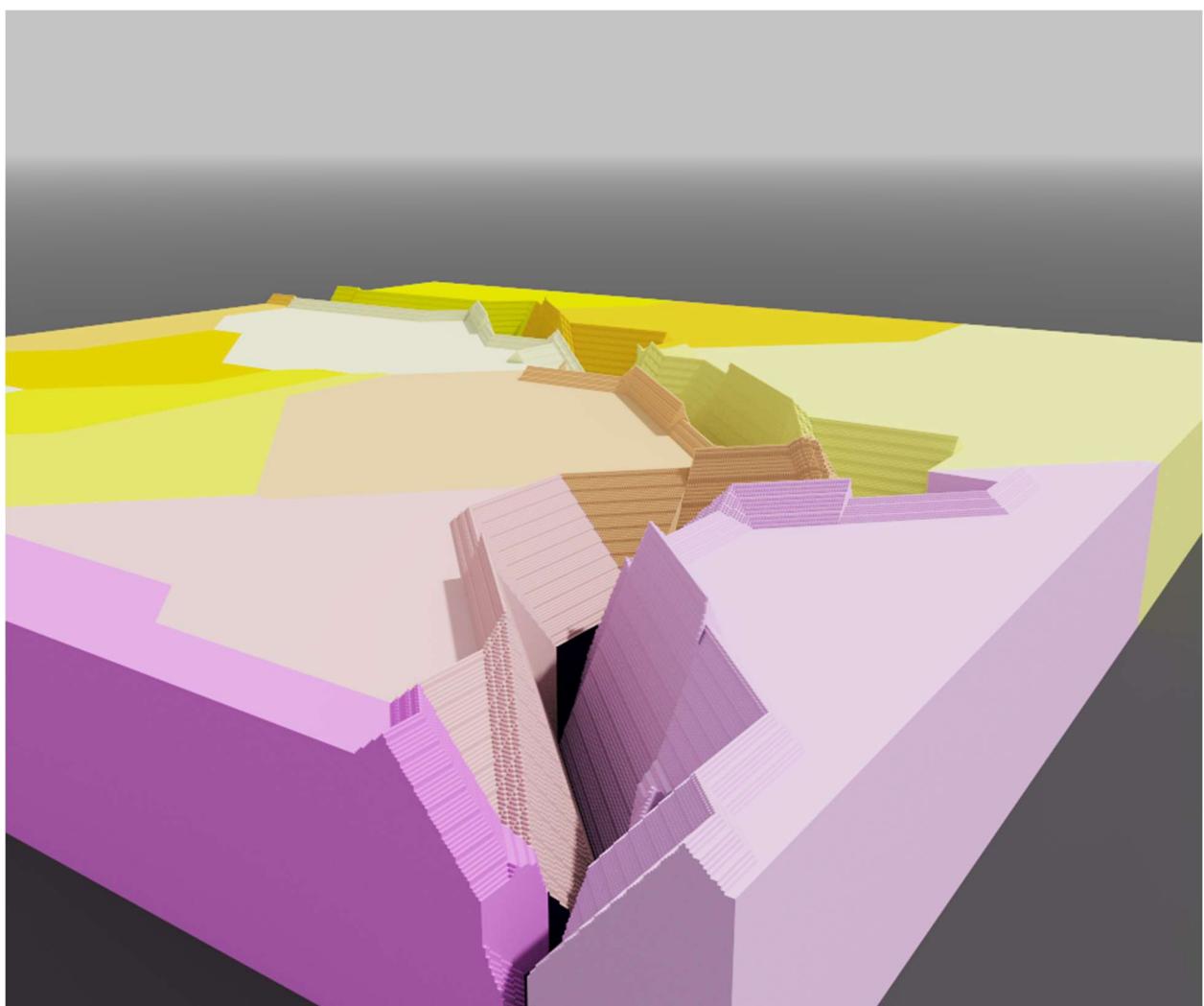


Рисунок 26 - срединно океанический хребет

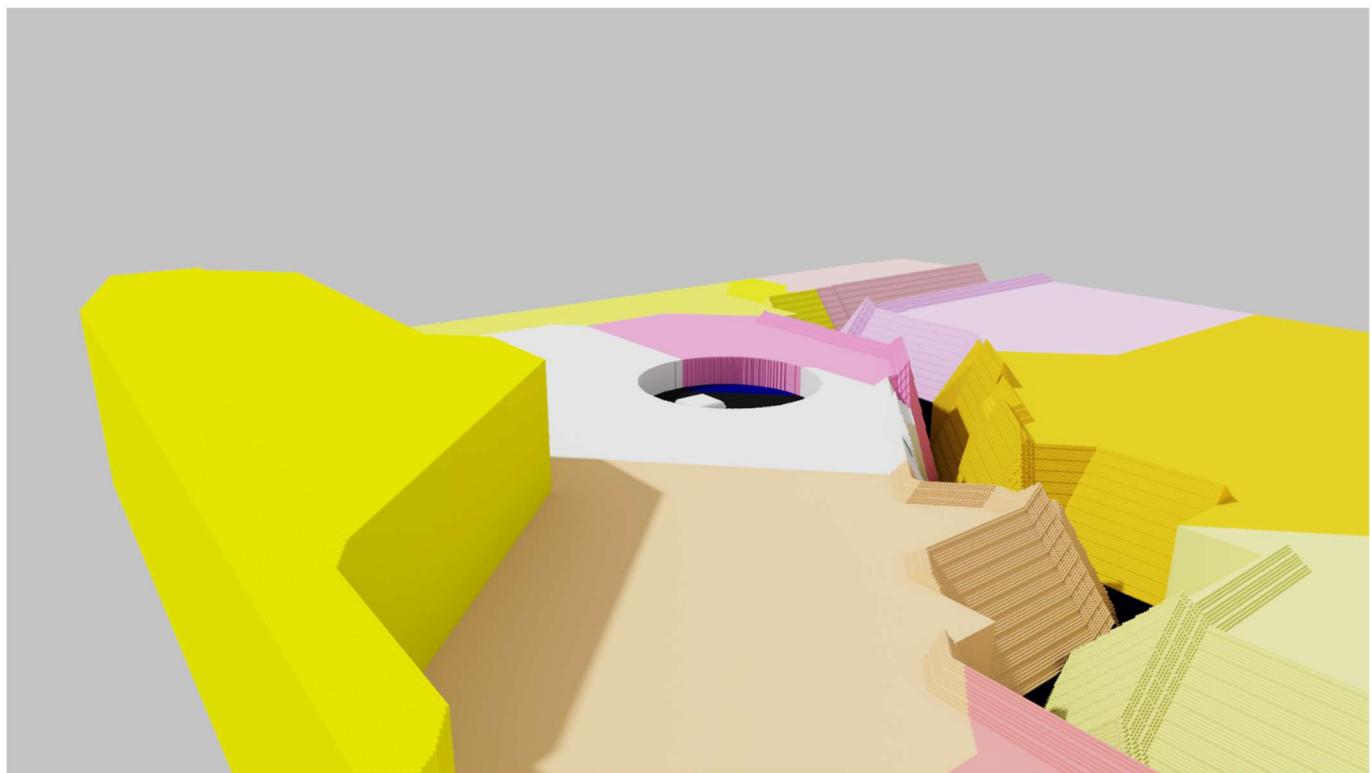


Рисунок 27 - СОХ, континентальная окраина и глубоководная котловина на одной карте

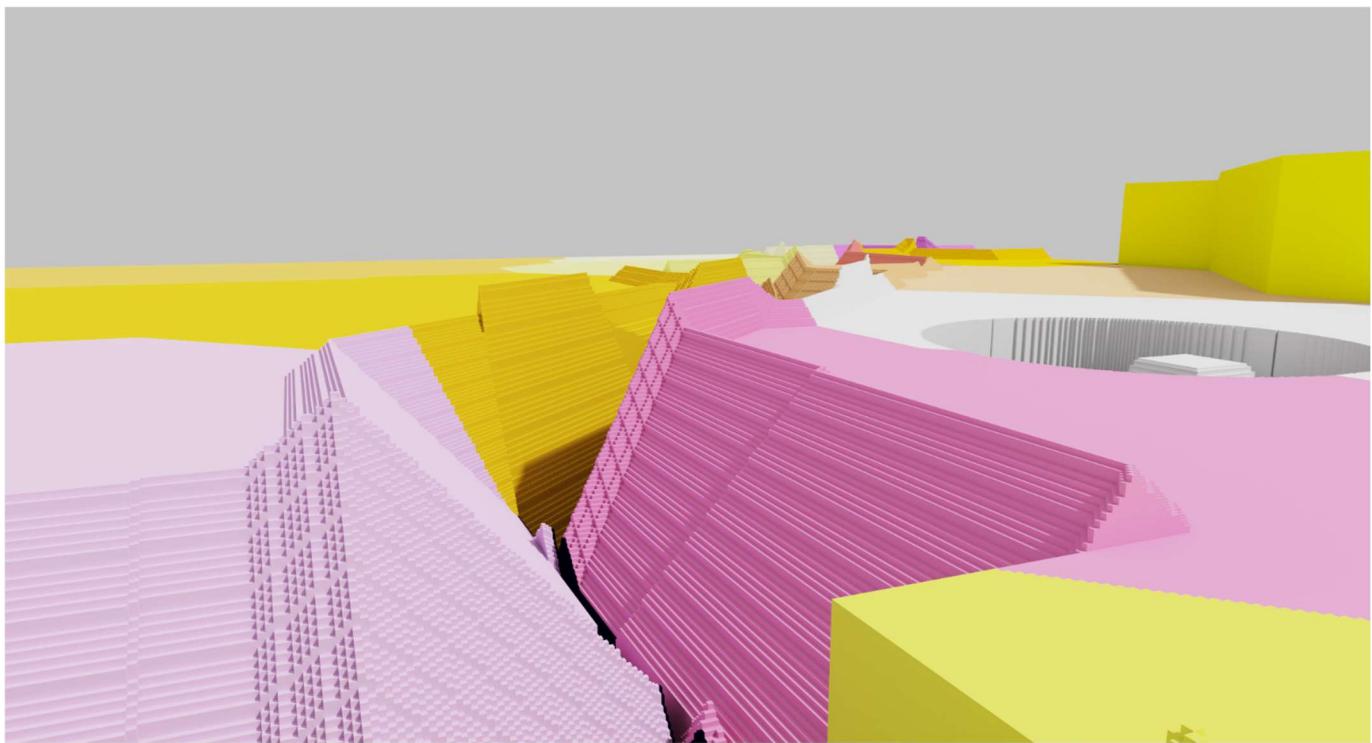


Рисунок 28 - СОХ, континентальная окраина и глубоководная котловина на одной карте, второй вид

3.4 Доработка результата

В текущей реализации есть проблема - помимо основных элементов дно представляет собой ровную поверхность, что нарушает ощущение реальности пейзажа, поэтому требуется придать дну большую естественность – сделать его более искривленным. Для решения этой проблемы можно использовать шумовые алгоритмы. Далее мы рассмотрим виды шумов и выберем наиболее подходящий.

3.4.1 Шумовые алгоритмы

Шумовые алгоритмы получили широкое распространение не только в компьютерной графике, но и в других областях, например так называемый синий шум используется в процессе дизайна аудио файлов и изображений [26], а фрактальный шум используется трейдерами для технического анализа финансовых рынков [27].

В работе [28] рассматриваются различные виды алгоритмов процедурной генерации шума и приводится список их выгодных качеств:

1. Компактность – алгоритмы практически не занимают места в памяти
2. Желаемое разрешение – алгоритм способен создавать шум любого требуемого разрешения
3. Непрерывность и не периодичность – алгоритм может генерировать шум без самоповторов
4. Параметризация – алгоритм можно настраивать под свои нужды с помощью параметров
5. Случайный доступ – алгоритм способен сгенерировать шум в любой точке пространства за фиксированное время независимо от предыдущих результатов.

Одним из простейших шумов является белый шум – это последовательность равномерно распределенных независимых случайных чисел, пример белого шума приведен на рисунке 29.

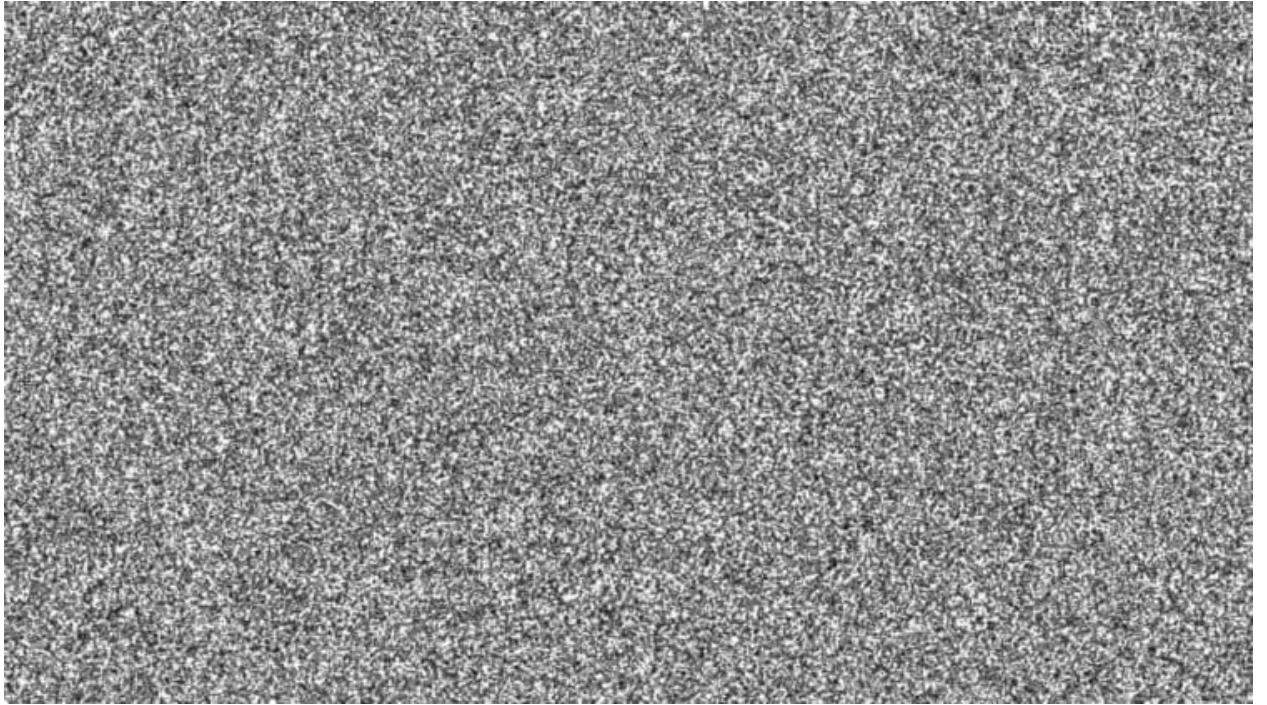


Рисунок 29 - Изображение сгенерированное с помощью белого шума

Однако в нашей природе местность редко бывает совершенно случайной - она демонстрирует плавные переходы и структуры. Для генерации ландшафтов, визуально соответствующих реальному миру, нам нужны алгоритмы, способные повторять эти плавные переходы. Далее мы рассмотрим такие алгоритмы, которые чаще всего используются в компьютерной графике. В работе [28] процедурные шумы разделяются на два типа: градиентные сеточные шумы и разреженные сверточные шумы.

3.4.1.1 Градиентные сеточные шумы

Одним из первых и наиболее распространенным градиентным сеточным шумом в генерации ландшафтов на сегодняшний день является шум Перлина [29], который он разработал в 80-х годах во время работы над спецэффектами для фильма «Трон». Суть шума Перлина заключается в разбиении исходного пространства на равномерную сетку и присваивании случайных градиентов узлам этой сетки. Далее, для каждой точки, в которой необходимо рассчитать значение шума вычисляются вектора до узлов сетки, в которой она находится, эти вектора отмечены красным на рисунке 30.

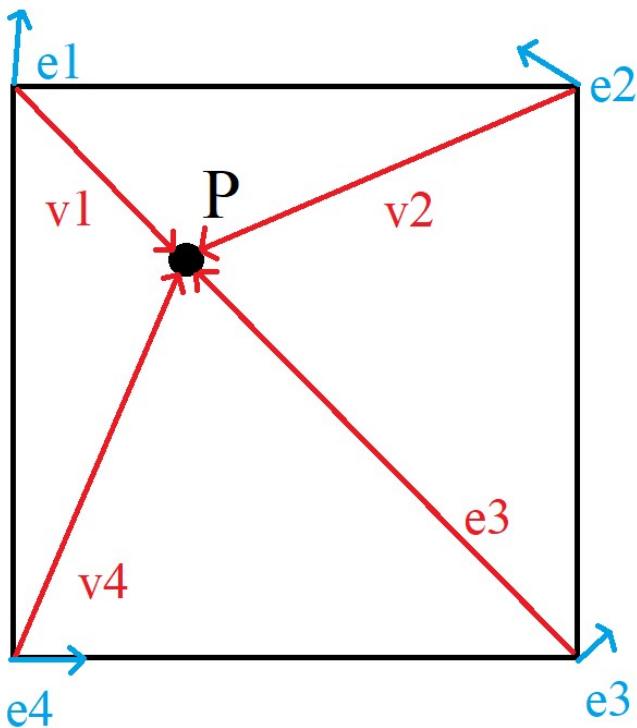


Рисунок 30 - Вычисление шума Перлина для точки Р. $e1, e2, e3, e4$ - случайные градиенты в узлах сетки

Затем они скалярно умножаются на градиенты из узлов, а результаты умножения интерполируются. Результат интерполяции является искомым шумом. Для улучшения результатов интерполяции также используется специальная функция, зачастую называемая функция затухания.

Весь процесс вычисления шума Перлина описывается в следующей формуле

$$P(x, y) = \text{lerp}(u, \text{lerp}(v, n00, n10), \text{lerp}(v, n01, n11)),$$

где $\text{lerp}(t, a, b)$ – функция линейной интерполяции между a и b с параметром t , $n00, n01, n10, n11$ – скалярные произведения градиентных векторов на вектора до точки Р, а u и v – это результат применения функции затухания к исходным координатам. Согласно [30] для функции затухания в алгоритме Перлина принято использовать функцию, представленную формулой

$$F(x) = 6x^5 - 15x^4 + 10x^3$$

Пример рельефа, сгенерированного с помощью шума Перлина приведен на рисунке 31.

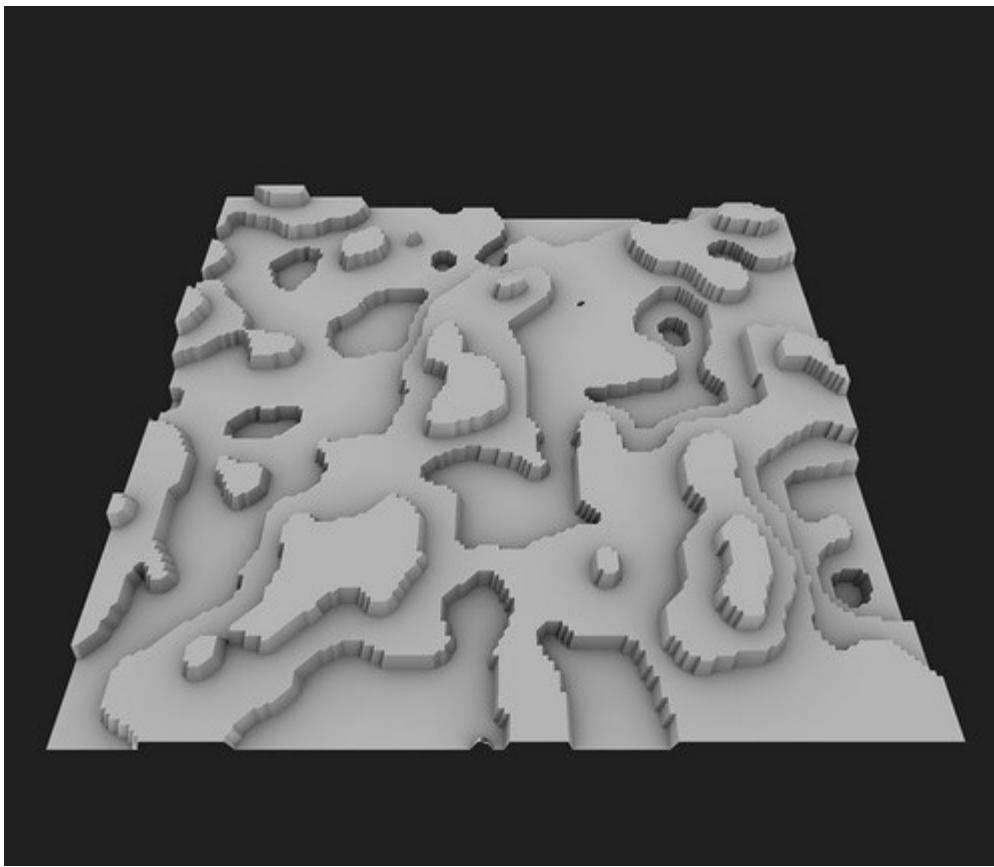


Рисунок 31 - Рельеф, сгенерированный с помощью шума Перлина

Помимо шума Перлина к градиентным шумам также относятся Симплекс шум и Value noise.

Value noise – это крайне простой вид шума, который также как и шум Перлина разбивает пространство с помощью сетки, а затем в каждом узле сетки генерирует случайное значение. Для вычисления значения в интересующей нас точке используется простая интерполяция значений ближайших узлов сетки.

Симплекс Шум – это шум, также разработанный Перлином, как улучшение исходного шума – он более вычислительно легкий в пространствах больших размерностей. Идея Симплекс Шума очень похожа на исходный шум Перлина, только пространство разбивается на сетку из треугольников, а не квадратов, затем каждому узлу также присваивается случайный градиентный вектор и производятся такие же вычисления как в шуме Перлина.

3.4.1.2 Разреженные сверточные шумы

Идея разреженных сверточных шумов заключается в подборе двух функций – функции ядра и импульсной функции и взаимодействия этих функций, для вычисления шума в интересующей нас точке. В данном виде шума также присутствует сетка, в узлах которой вычисляются значения импульсной функции, данная сетка может быть как равномерная, так и не равномерная. Для вычисления сверточного шума используется формула

$$N(x, y) = \iint Y(u, v)k(x - u, y - v)dudv,$$

где $Y(u, v)$ – это вклад импульсной функции, в данной точке. В качестве импульсной функции обычно выбирается дельта-функция Дирака, умноженная на какое-то случайное число, а $k(x - u, y - v)$ – это функция ядра. Функция Y приведена в формуле (10)

$$Y(x, y) = \sum_k a_k \delta(x - x_k, y - y_k), \quad (10)$$

где $\delta(x, y) = \begin{cases} 1, & x = y = 0 \\ 0, & \text{иначе} \end{cases}$, a_k – случайное число.

Одним из примеров разреженного сверточного шума является шум Габора. В шуме Габора в качестве импульсной функции используется дельта-функция Дирака, представленная в формуле (10), а в качестве ядра используется довольно сложная Гaborова функция, вычисляемая по формуле

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right),$$

где $x' = x \cos \theta + y \sin \theta$, $y' = -x \sin \theta + y \cos \theta$, λ – длина волны, θ – ориентация ядра, ψ – фазовый сдвиг, σ – стандартное отклонение, γ – пространственное соотношение сторон.

Такие шумы сильно затратнее для вычисления, чем градиентные версии, однако они могут использоваться для генерации более интересных и сложных текстур, например волос, как показано на рисунке 32, данный пример взят из работы [28].

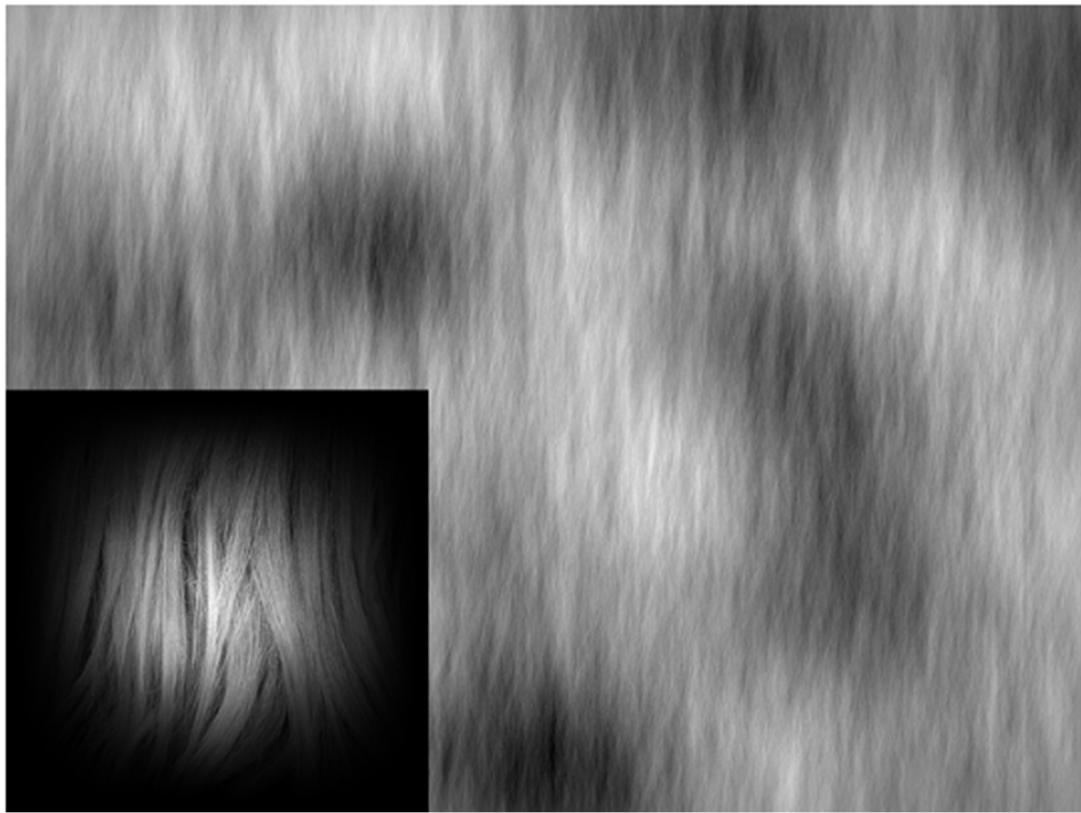


Рисунок 32 - Пример генерации текстуры с помощью сверточного разреженного шума. Слева внизу приведена текстура волос.

3.4.1.3 Другие виды шумов

Конечно, помимо представленных в предыдущих двух параграфах существует множество других шумов. Например, в работе [28] представлен еще один тип шумов, называемый «явные шумы», однако они не относятся к процедурным шумам, так как требуют предварительных вычислений и сохранения результата. Помимо этого, в работе [31] описываются различные подходы к генерации текстур, которые делятся на две больших группы: алгоритмическая генерация и фильтрация с пост-обработкой уже готовых текстур. Фильтрация и пост-обработка текстур не являются процедурной генераций, т. к. подразумевается наличие уже готовой текстуры, а остальные методы, либо пересекаются с уже рассмотренными, либо имеют большую вычислительную сложность в сравнении с шумами, как пример можно привести алгоритмы физической симуляции, алгоритм реакция-диффузия, преобразования матриц и прочие, также зачастую эти алгоритмы используются для генерации текстур, совершенно не похожих на какой-либо

рельеф. Пример текстуры, сгенерированной с помощью алгоритма матричных преобразований представлен на рисунке 33.

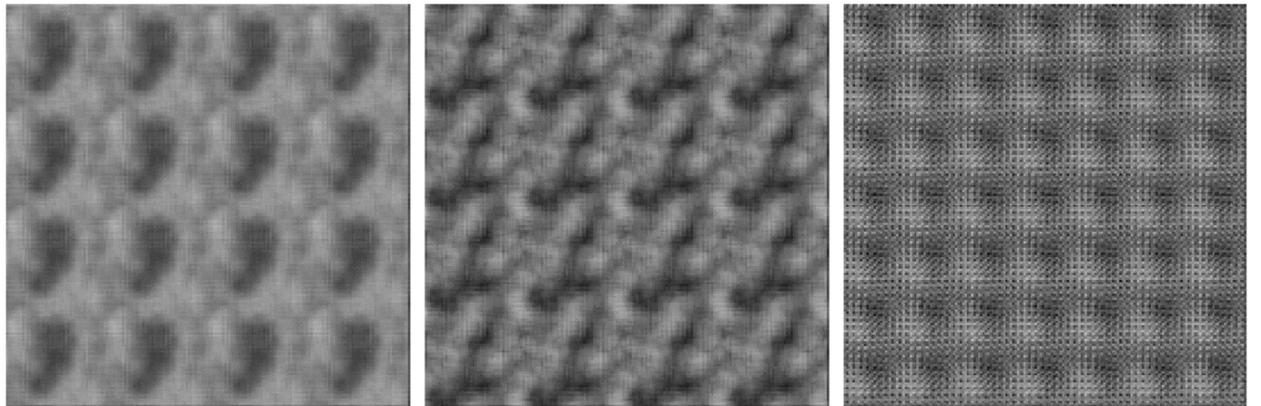


Рисунок 33 - карты высот, сгенерированные с помощью матричных преобразований

3.4.2 Выбор шума

По итогу рассмотрения различных процедурно генерируемых шумов было решено использовать шум Перлина по следующим причинам: во-первых, он наиболее легкий в плане вычислительной сложности, что подтверждается исследованиями проведенными в работе [32], в котором показано, что в двумерном случае он обгоняет даже свою улучшенную версию – симплекс шум, во-вторых, с момента своего создания и до сих пор он хорошо показывает себя в области генерации рельефов и используется на практике, в третьих, алгоритм генерации шума нам нужен только для придания поверхности чуть менее плоского вида – нам не требуется генерировать сложных узоров.

Так как целью данной работы не является реализация алгоритмического шума, то было решено использовать библиотеку «FastNoise» с открытым исходным кодом, рассматриваемую в работе [32] и использующуюся для генерации шума в движке Godot [33].

Генерация шума встраивается в алгоритм внутрь функции «`set_properties`», теперь для установки высоты мы не только выбираем случайное значение из диапазона [1000, 10000], но и строим шум относительно этого уровня. Блок-схема предыдущей и текущей реализации функции

«set_properties» для сравнения, а также функции «make_noise», отвечающую за генерацию шума на карте, приведены на рисунке 34.

Функция «Perlin», представленная на рисунке 6 находится в библиотеке

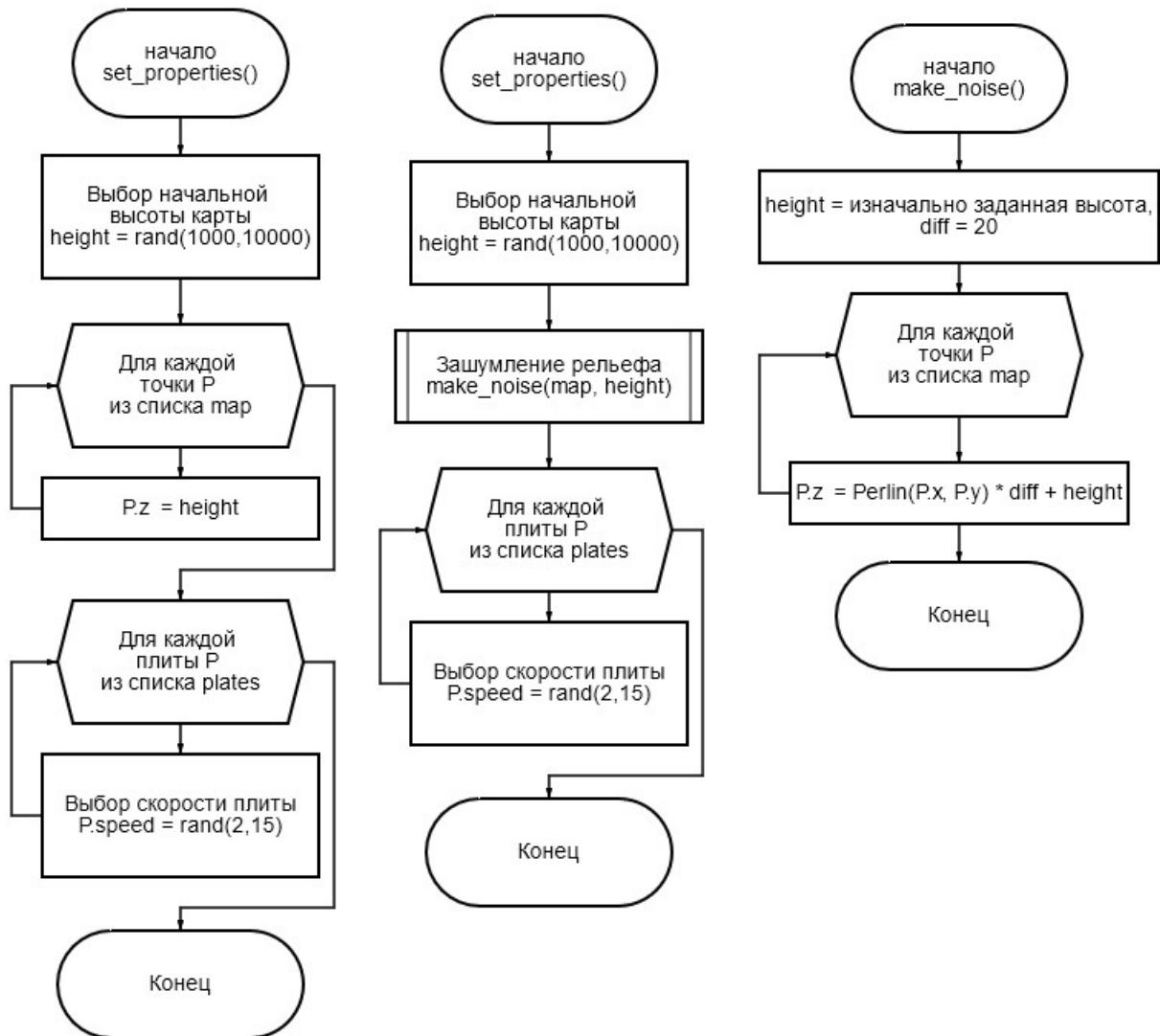


Рисунок 34 - Исходный вид функции set_properties (слева), текущий ее вид (посередине) и функция, отвечающая за шум (справа)

«FastNoise», она принимает на вход точку, в которой нужно сгенерировать шум и возвращает число в промежутке [-1, 1]. Результаты работы алгоритма Перлина можно увидеть на рисунке 35. А на рисунке 36 приведен результат работы алгоритма с применением шума Перлина.

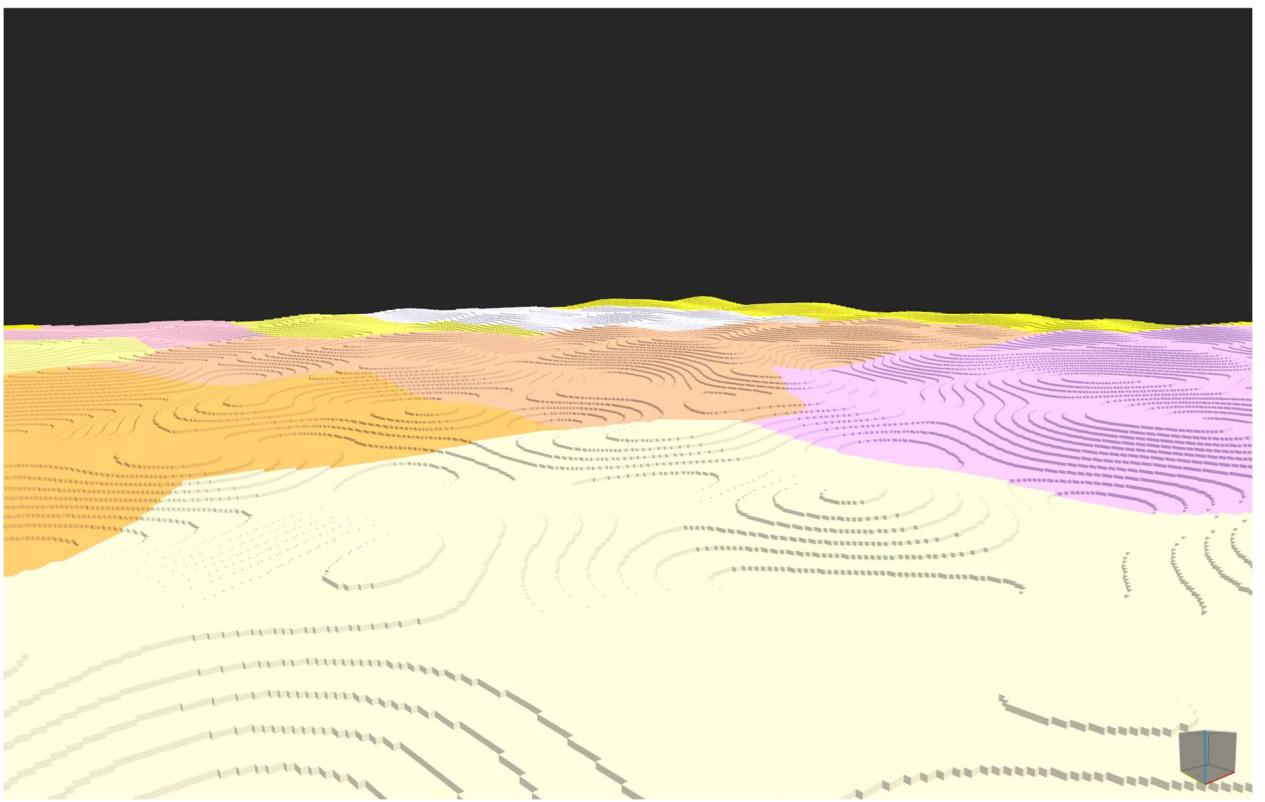


Рисунок 35 - Результат применения шума Перлина к рельефу

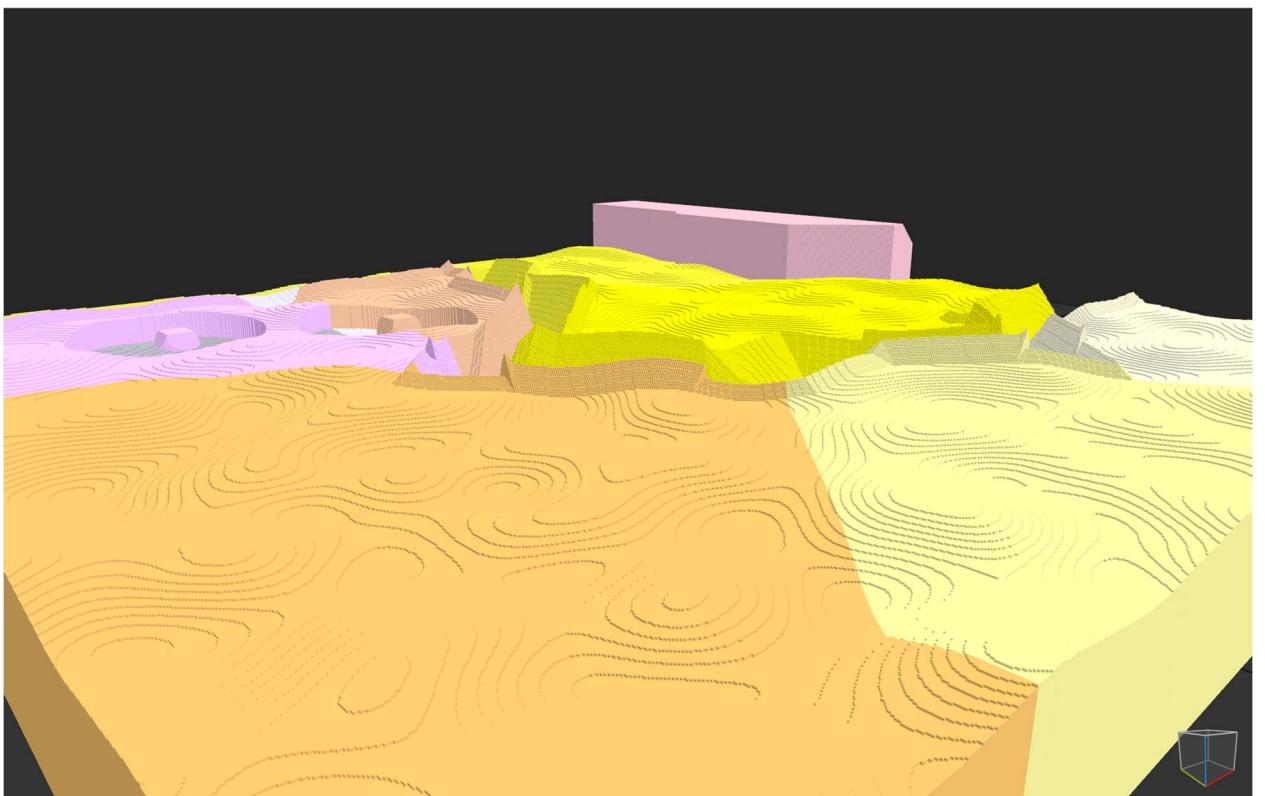


Рисунок 36 - Сгенерированный алгоритм с применением шума Перлина

4 ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ АЛГОРИТМА

4.1 Конфигурация системы

Перед тем, как приступить к описанию эксперимента и его результатам опишем конфигурацию, на которой эксперимент проводился.

Все измерения производились в подсистеме «wsl2» на windows 10. Ниже приведена информация о системе, полученная с помощью команды “uname -a”: «Linux 5.10.16.3-microsoft-standard-WSL2 #1 SMP Fri Apr 2 22:23:49 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux».

Флаги сборки приложения, сгенерированные системой CMake в режиме релизной сборки: «-O3 -DNDEBUG».

Версия компилятора: «clang version 10.0.0-4ubuntu1»

Модель используемого процессора: «Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz».

Количество доступной оперативной памяти: 16гб.

4.2 Описание эксперимента

Для проведения эксперимента был написан простейший класс таймер, который запускался при начале выполнения замеряемой функции и останавливался по ее завершении. Измерение каждой функции повторялось тысячу раз, после чего по полученным данным рассчитывались метрики.

Для замеров в эксперименте были выбраны все основные этапы алгоритма – генерация карты, разбиение карты на литосферные плиты, зашумление карты, инициализация каждого элемента рельефа по отдельности – COX, континентальная окраина, глубоководная котловина, а также итерация генерации каждого элемента. Помимо измерения каждой функции по отдельности также был замерен и алгоритм в целом, однако, чтобы исключить влияние случайного количества генерируемых элементов рельефа из замеров было решено добавить дополнительные параметры, указываемые пользователем при запуске алгоритма, отвечающие за количество каждого генерируемого элемента. Для замеров выполнения алгоритма полностью была

выбрана следующая конфигурация: 1 COX, 2 глубоководные котловины и 1 континентальная окраина.

Также для избежания появления лишнего шума в результатах каждый запуск процесса привязывался к одному ядру с помощью команды “taskset”. И

Измерения проводились на картах размера 500x500, 1000x1000, 5000x5000, с количеством лет генерации 1000, 5000, 10000 и 15000.

4.3 Результаты эксперимента

Сперва рассмотрим время генерации каждого элемента по отдельности, в зависимости от размера решетки, оно приведено на рисунке 37.

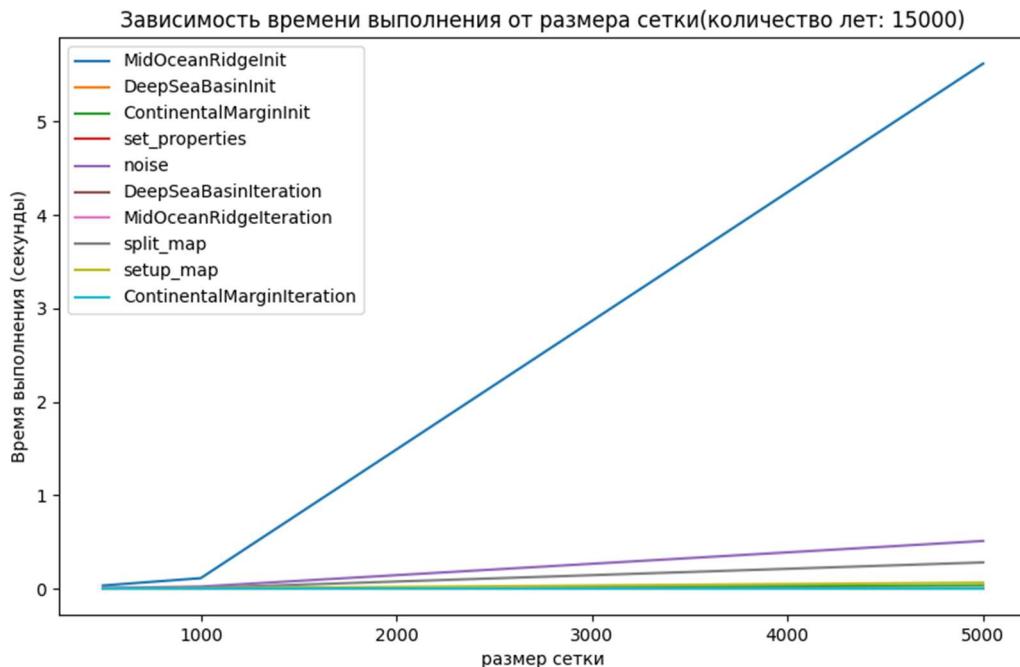


Рисунок 37 - Время исполнения различных функций в зависимости от размера решетки

Ожидаемо наибольшее время работы в генерации ландшафта занимает инициализация COX, т. к. ей требуется обойти всю карту несколько раз и построить маршрут, к тому же при генерации происходит много работы с памятью, что также влияет на процесс. Помимо COX из графика видно, что больше остальных время занимают функции генерации шума и разбиения карты на литосферные плиты. Время, занимаемое остальными функциями, в том числе итерациями генерации каждого из элементов близко к 0. Далее,

рассмотрим на примере самого ресурсоемкого процесс – генерации COX – соотношение времени итераций и времени инициализации, оно представлено на рисунке 38.

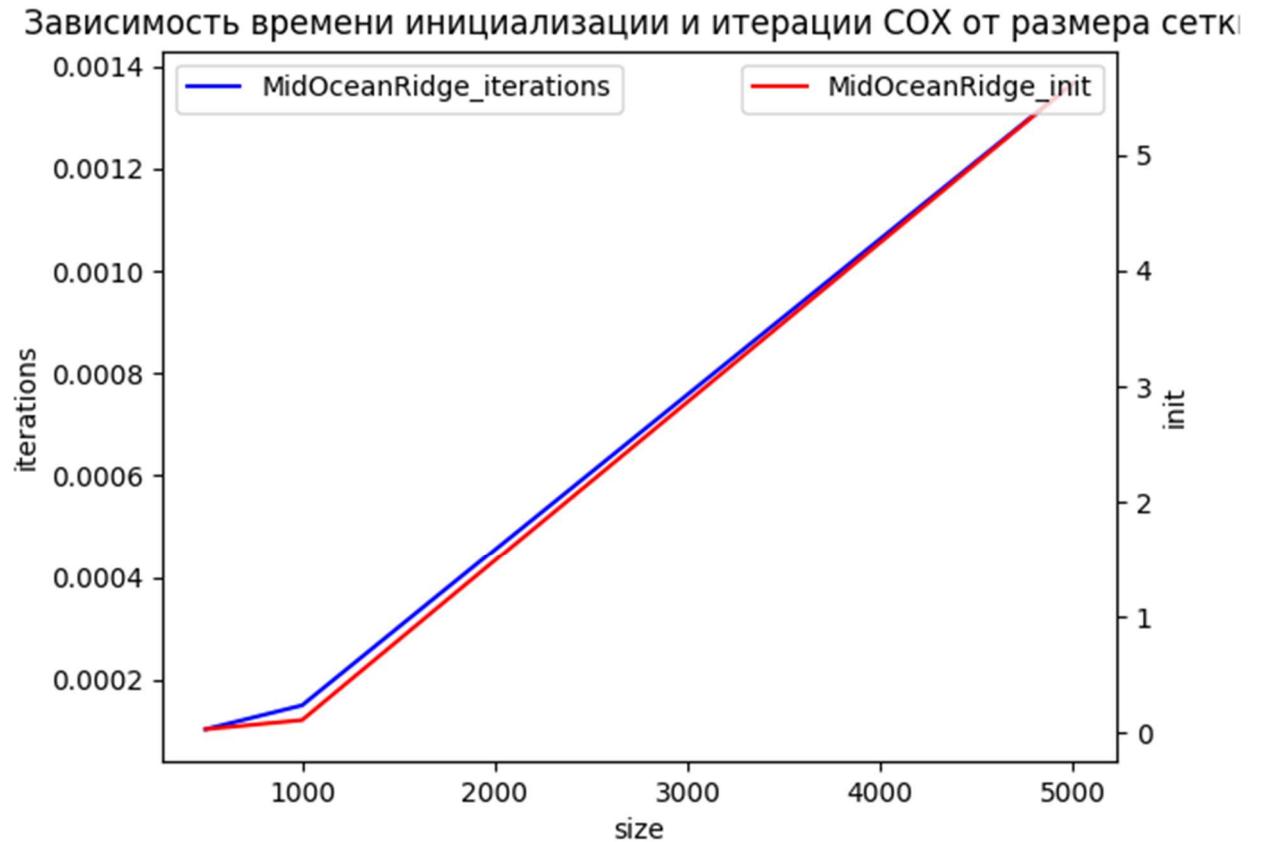


Рисунок 38 - Сравнение времени исполнения инициализации и одной итерации генерации COX. Слева шкала времени итераций (синий график), справа – инициализация (красный график)

Видно, что среднее время итераций занимает около миллисекунды, в то время как инициализация происходит за секунды. Однако для остальных форм рельефа это не так, т. к. для инициализации глубоководной окраины и котловины практически не требуется никаких вычислений, то эти функции также занимают миллисекунды.

Далее рассмотрим, сколько времени занимает выполнение алгоритма от начала и до конца в зависимости от количества лет – рисунок 39 и от размера сетки – рисунок 40.

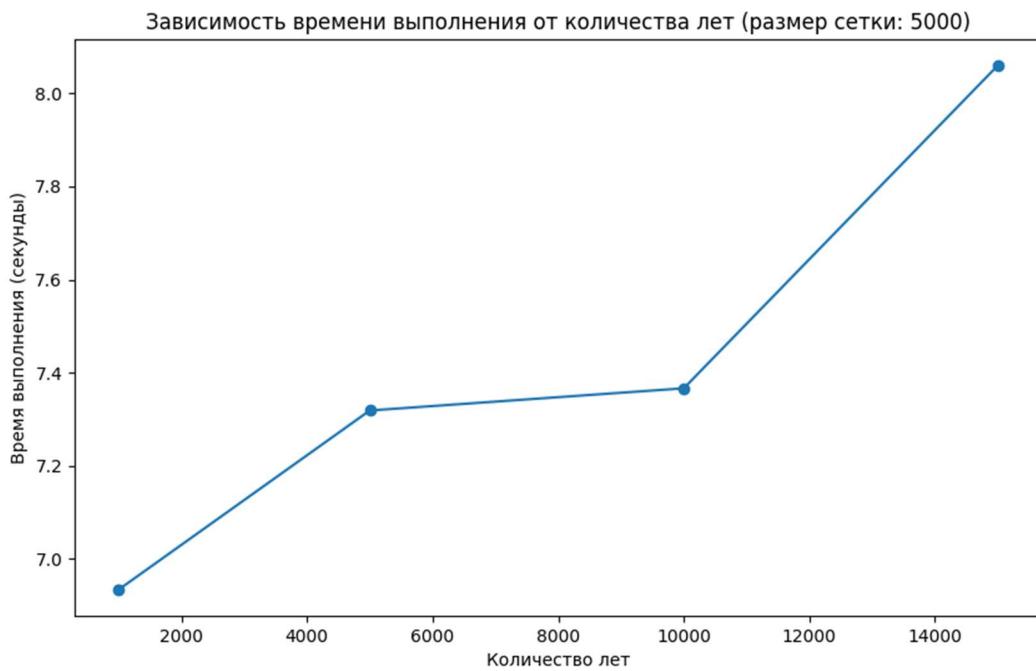


Рисунок 39 - Зависимость времени выполнения алгоритма от количества лет

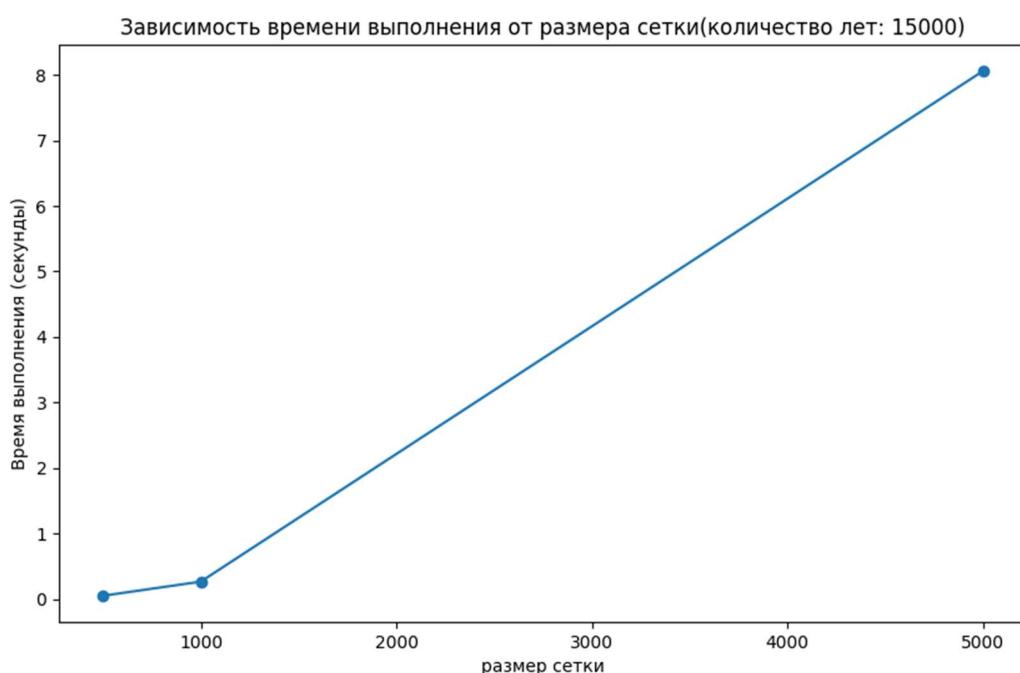


Рисунок 40 - Время работы алгоритма в зависимости от размера решетки

Из рисунков 39 и 40 видно, что при увеличении количества лет производительность понижается не так сильно, как при увеличении размера

сетки, поэтому можно сделать вывод, что размер сетки влияет на производительность сильнее, чем количество итераций.

Более подробно все результаты измерений в зависимости от размера сетки приведены в таблице 2.

Таблица 2 - Зависимость скорости генерации элементов и ландшафта в целом от размера карты (в секундах)

Этап/размер сетки	500x500	1000x1000	5000x5000
Инициализация К.О.	$25 * 10^{-5}$	$145 * 10^{-5}$	$3403 * 10^{-5}$
Итерация К.О.	$1.5 * 10^{-6}$	$1.8 * 10^{-6}$	$5.7 * 10^{-6}$
Инициализация Г.К	$5.32 * 10^{-5}$	$1.5 * 10^{-5}$	$9.2 * 10^{-5}$
Итерация Г.К.	$1.4 * 10^{-6}$	$2.2 * 10^{-6}$	$1.7 * 10^{-6}$
Инициализация COX	$3 * 10^{-3}$	0.11	5.62
Итерация COX	$104 * 10^{-6}$	$136 * 10^{-6}$	$150 * 10^{-6}$
Подготовка карты	$493 * 10^{-6}$	$2.2 * 10^{-4}$	$63 * 10^{-3}$
Разбиение карты	$149 * 10^{-5}$	$719 * 10^{-5}$	0.28
Генерация шума	$5 * 10^{-3}$	0.02	0.5
Весь алгоритм	0.05	0.26	8.06

Из проведенного эксперимента видно, что наиболее дорогостоящей операцией является инициализация COX, поэтому можно попробовать поискать оптимизации для его работы. Однако, если проинициализировать все элементы заранее, то можно использовать алгоритм, для итерационной генерации ландшафта в реальном времени. К тому же в текущей реализации алгоритм является однопоточным, что не учитывает возможности современных процессоров и тем более видеокарт, поэтому еще одним направлением для дальнейших исследований может являться распараллеливание алгоритма на ЦПУ или перенос его на графические ускорители. Также возможны исследования в направлении потоковой генерации алгоритма, т. е. генерировать только те части, которые нужны в конкретный момент времени, а не всю карту целиком.

ЗАКЛЮЧЕНИЕ

Процедурная генерация является распространенным методом помощи дизайнерам в создании контента, одним из видов которого являются ландшафты, форма и размеры которых ограничены лишь фантазией дизайнера и требованиями заказчика.

В ходе данной научно-исследовательской работы были рассмотрены современные методы процедурной генерации ландшафтов, которые представляют собой три основных вида: физическая симуляции, алгоритмы и генерация по примерам. Также рассмотрены способы представления ландшафтов, состоящие их двух основных групп – объемные данные и карты высот. Проведен анализ преимуществ данных подходов и представлений, сильные и слабые стороны каждого из них.

Так как целью работы является генерация океанического и морского дна, то были исследованы основные геоморфологические процессы их формирования, рассмотрены основные формы рельефа, которыми являются средне-океанические хребты, глубоководные котловины и континентальные окраины.

Итогом исследовательской работы является представленный в последней части алгоритм генерации ландшафта морского дна, основанный на рассмотренных процессах и включающий в себя все основные формы рельефа. Конечно, океаническое дно содержит в себе не только элементы, рассмотренные в данной работе – существуют также различные впадины, склоны, подводные желоба и хребты, а также различные виды биомов, которые могут влиять на них, поэтому в качестве дальнейшего направления исследований можно предложить исследование менее значимых элементов морского дна, а также интересным направлением для исследований может являться место соприкосновения океана и суши.

Итогом практической работы является реализованный на языке программирования C++ алгоритм, способный генерировать океаническое дно и все основные элементы рельефа присутствующие на нем. За счет

итеративного подхода можно посмотреть не только конечный этап построения дна, но и проследить его генерацию в любой момент времени. На данный момент алгоритм имеет некоторые недостатки и главный из них – минимальный контроль над генерируемыми ландшафтами. В качестве дальнейших улучшений можно предложить пользователю вводить больше данных, например позволить ему указывать количество элементов рельефа каждого вида, позволить указывать количество плит, или даже указывать само разбиение на плиты с помощью подачи на вход предварительно сгенерированных файлов в формате vox.

В ходе проведенного эксперимента было замерено время выполнения алгоритма с различными входными параметрами. Наиболее вычислительно затратным ожидаемо оказался алгоритм инициализации COX, но при этом итерации генерации всех форм рельефа исполняются достаточно быстро. Также эксперимент дал понимание слабых мест в производительности алгоритма и пути для дальнейшего его улучшения.

Нам не известно о существовании способа точной оценки качества получаемых результатов, поэтому все результаты оценивались «на глаз». Основные элементы рельефа на наш взгляд получились достаточно красиво.

Разработанный алгоритм, при должном развитии может быть использован не только при генерации уровней в играх, но также и при исследованиях морского дна и истории его развития, не требуя для этого погружения, а также при обучении школьников или студентов в области геологии и геоморфологии.

Результаты данной выпускной квалификационной работы, исходный код разработанного алгоритма, а также инструкция по сборке и запуску алгоритма опубликованы на гитхабе, получить к ним доступ можно по следующей ссылке: <https://github.com/GitCaptain/ocean-landscape>.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Лаптев Н. В., Лаптев В. В., Гергет О. М. ПРОЦЕДУРНАЯ ГЕНЕРАЦИИ КАРТЫ МЕСТНОСТИ ДЛЯ ОБУЧЕНИЯ АГЕНТА НАВИГАЦИИ В НЕИЗВЕСТНОЙ СРЕДЕ //Математические методы в технологиях и технике. – 2021. – №. 6. – С. 71-77.
- [2] Shaker N., Togelius J., Nelson M. J. Procedural content generation in games. – 2016.
- [3] Valencia-Rosado L. O., Starostenko O. Methods for procedural terrain generation: a review //Pattern Recognition: 11th Mexican Conference, MCPR 2019, Querétaro, Mexico, June 26–29, 2019, Proceedings 11. – Springer International Publishing, 2019. – С. 58-67.
- [4] Fischer R., Boeckers J., Zachmann G. Procedural generation of landscapes with water bodies using artificial drainage basins //Computer Graphics International Conference. – Cham : Springer Nature Switzerland, 2022. – С. 345-356.
- [5] Št'ava O. et al. Interactive terrain modeling using hydraulic erosion //Proceedings of the 2008 acm siggraph/eurographics symposium on computer animation. – 2008. – С. 201-210.
- [6] Galin E. et al. A review of digital terrain modeling //Computer Graphics Forum. – 2019. – Т. 38. – №. 2. – С. 553-577.
- [7] Cordonnier G. et al. Authoring landscapes by combining ecosystem and terrain erosion simulation //ACM Transactions on Graphics (TOG). – 2017. – Т. 36. – №. 4. – С. 1-12
- [8] Furtado H. Procedural Generation of Volumetric Data for Terrain. – 2019.
- [9] Panagiotou E., Charou E. Procedural 3D terrain generation using Generative Adversarial Networks //arXiv preprint arXiv:2010.06411. – 2020.
- [10] Li W. Terrain synthesis for treadmill exergaming in virtual reality //2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW). – IEEE, 2023. – С. 263-269
- [11] Lagae A. et al. A survey of procedural noise functions //Computer Graphics Forum. – Oxford, UK : Blackwell Publishing Ltd, 2010. – Т. 29. – №. 8. – С. 2579-2600
- [12] Ashlock D., McGuinness C. Landscape automata for search based procedural content generation //2013 IEEE Conference on Computational Intelligence in Games (CIG). – IEEE, 2013. – С. 1-8.
- [13] Короновский Н.В. Общая геология: учебное пособие, электронное издание сетевого распространения. – М.: «КДУ», «Добросвет», 2018. – ISBN 978-5-7913-1025-5.
- [14] СРЕДИННО-ОКЕАНИЧЕСКИЕ ХРЕБТЫ // Большая российская энциклопедия. Электронная версия (2017); <https://old.bigenc.ru/geography/text/4161181> Дата обращения: 17.01.2023

- [15] КОНТИНЕНТАЛЬНЫЕ ОКРАИНЫ // Большая российская энциклопедия. Электронная версия (2016); <https://old.bigenc.ru/geology/text/2093230> Дата обращения: 17.01.2024
- [16] Соколов С. Д. АКТИВНЫЕ КОНТИНЕНТАЛЬНЫЕ ОКРАИНЫ // Большая российская энциклопедия. Электронная версия (2016); <https://old.bigenc.ru/geology/text/1807858> Дата обращения: 17.01.2024
- [17] Мазарович А. О. Строение дна Мирового океана и окраинных морей России. – 2006.
- [18] Сафьянов Г. А. Абиссальная равнина // Большая российская энциклопедия: научно-образовательный портал – URL: <https://bigenc.ru/c/abissal-naia-ravnina-6abc28/?v=5267594>. – Дата обращения: 17.01.2023
- [19] Laine S., Karras T. Efficient sparse voxel octrees //Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games. – 2010. – С. 55-63.
- [20] Kämpe V., Sintorn E., Assarsson U. High resolution sparse voxel dags //ACM Transactions on Graphics (TOG). – 2013. – Т. 32. – №. 4. – С. 1-13.
- [21] Villanueva A. J., Marton F., Gobbetti E. SSVDAGs: Symmetry-aware sparse voxel DAGs //Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. – 2016. – С. 7-14.
- [22] Furtado H. Procedural Generation of Volumetric Data for Terrain. – 2019.
- [23] Файл формата «vox» [Электронный ресурс]. URL: <https://github.com/ephtracy/voxel-model/blob/master/MagicaVoxel-file-format-vox.txt> (дата обращения 20.01.2024)
- [24] De Berg M. Computational geometry: algorithms and applications. – Springer Science & Business Media, 2000.
- [25] Кормен Т. и др. Алгоритмы. Построение и анализ:[пер. с англ.]. – Издательский дом Вильямс, 2009.
- [26] Yan D. M. et al. A survey of blue-noise sampling and its applications //Journal of Computer Science and Technology. – 2015. – Т. 30. – №. 3. – С. 439-452.
- [27] Бутовский ММ. Технический анализ и фрактальные методы в исследовании финансовых рынков. Вестник Бурятского государственного университета. Математика, информатика. 2011(9):237-44.
- [28] Cook R. et al. State of the Art in Procedural Noise Functions. – 2010.
- [29] Perlin K. An image synthesizer //ACM Siggraph Computer Graphics. – 1985. – Т. 19. – №. 3. – С. 287-296.
- [30] Perlin K. Improving noise //Proceedings of the 29th annual conference on Computer graphics and interactive techniques. – 2002. – С. 681-682.

- [31] Dong J. et al. Survey of procedural methods for two-dimensional texture generation //Sensors. – 2020. – Т. 20. – №. 4. – С. 1135.
- [32] Kopel M., Maciejewski G. Comparison of Procedural Noise-Based Environment Generation Methods //Computational Collective Intelligence: 12th International Conference, IICCI 2020, Da Nang, Vietnam, November 30–December 3, 2020, Proceedings 12. – Springer International Publishing, 2020. – С. 878-887.
- [33] Библиотека процедурной генерации шума, [Электронный ресурс] URL: <https://github.com/Auburn/FastNoiseLite> – Дата обращения: 20.04.24