II Соревнование молодых исследователей программы «Шаг в будущее» в Сибирском и Дальневосточном федеральных округах

### РАЗРАБОТКА ПРОЦЕДУРНОГО ГЕНЕРАТОРА БЕСКОНЕЧНЫХ ЛАНДШАФТОВ

*Автор проекта:* ***Корж Даниил Андреевич****, учащийся 11 класса физико-математического профиля МБОУ «Лицей №1» г. Иркутска Иркутской области*

*Руководитель:* ***Нефедьева Марина Анатольевна****, учитель Информатики, МБОУ «Лицей №1» г. Иркутска Иркутской области*

г. Усолье-Сибирское, Иркутская область  
2017 год

### Содержание

[Актуальность 3](#_Toc477572726)

[Цель 4](#_Toc477572727)

[Задачи 4](#_Toc477572728)

[Введение 5](#_Toc477572729)

[1 Процедурная генерация ландшафтов 6](#_Toc477572730)

[1.1. История возникновения 6](#_Toc477572731)

[1.2. Основные этапы 7](#_Toc477572732)

[1.3. Виды алгоритмов генерации карт высот 7](#_Toc477572733)

[1.4. Алгоритм Diamond-Square 8](#_Toc477572734)

[1.5. Создание карты биомов 9](#_Toc477572735)

[2 Когерентные шумы 10](#_Toc477572736)

[2.1. Частота 10](#_Toc477572737)

[2.2. Многооктавнось 11](#_Toc477572738)

[2.3. Лакунарность 11](#_Toc477572739)

[2.4. Персистентность 11](#_Toc477572740)

[3 Реализация проекта 12](#_Toc477572741)

[3.1. Технологии 12](#_Toc477572742)

[3.2. Отложенные вычисления 12](#_Toc477572743)

[3.3. Структура модели 12](#_Toc477572744)

[3.4. Используемые модификаторы 13](#_Toc477572745)

[3.5. Многопоточность и синхронизация 13](#_Toc477572746)

[Заключение 15](#_Toc477572747)

[Список литературы 16](#_Toc477572748)

[Приложения 17](#_Toc477572749)

### Актуальность

Идея данной работы состоит в оптимизации процесса создания ландшафтов для индустрии 3d моделирования и других, смежных с ней областей деятельности, таких как киноиндустрия, разработка компьютерных игр и.т.п. Создание трёхмерного ландшафта является продолжительным трудом большого числа CGI специалистов. Процедурная генерация позволяет ускорить процесс путём создания пусть и не идеальной, но подходящей для дальнейших преобразований, или использования в необработанном виде, заготовки. Это сокращает сроки создания готовой локации для проекта с двух-трёх недель, до пяти-четырёх дней.

### Цель

Создать рабочий процедурный генератор реального времени выполнения, основанный на двумерных когерентных шумах с применением игрового движка “Unity3D” используя язык программирования C#.

### Задачи

* Изучить разновидности, историю возникновения алгоритмов процедурной генерации ландшафтов.
* Рассмотреть применение данных технологий и алгоритмов в реальной жизни.
* Создать удобную для расширения и редактирования абстрактную модель процедурного генератора
* Создать реализацию данной модели и использовать её для написания наглядной демонстрации процедурного генератора реального времени выполнения и расширения редактора Unity3d

### Введение

Со времён появления 3D графики, компьютерной анимации и прочих связанных с ними дисциплин (как например: 3D моделирование), всегда остро стояла проблема создания локации для сцены (анимации, мультфильма, презентации), так как создание реалистичного, физически корректного, и при этом визуально красивого ландшафта является плодом долгого труда дизайнеров, моделлеров, текстурировщиков. И учитывая, что уровень мощности компьютеров лишь недавно стал достаточно высоким, закономерно возник вопрос: «Возможно-ли оптимизировать процесс создания локации?». Ответом на него является концепция процедурной генерации, алгоритмы соответствующие которой используются по сей день.

### Процедурная генерация ландшафтов

* 1. История возникновения

Создание ландшафта даже в современное время является достаточно сложным технологический процессом. Но ещё более затратным это было, когда мощностей ПК не хватало даже на отрисовку простейшей 3D графики в режиме реального времени. Именно в те времена начинал свою карьеру один из ключевых фигур в развитии компьютерной анимации в целом, и процедурной генерации ландшафтов в частности, один из основателей студии Pixar – Лорен Карпентер.

В 1977 году будучи инженером Boeing он начал экспериментировать с компьютерной графикой. Изначально его проектом была презентация летающих моделей самолётов. В какой-то момент времени к нему пришла идея создать красивый задник, фон, на котором был бы огромный ландшафт с высоты птичьего полёта. В том году он не смог это реализовать и отложил реализацию проекта на более поздний срок.

В следующем году ему на глаза в магазине попалась книга, Бенуа Мандельброта «Фракталы: форма, случайность и размерность». Конкретно в данной книге его заинтересовало то, что Мандельброт привёл массу примеров из реальной жизни и описал их соответствующими формулами.

Изучив данную книгу, он серьёзно заинтересовался применением данных теорий и стал искать способы реализации. Всего за три дня он смог нарисовать вполне себе узнаваемые, даже в чём-то уже по идее выдающиеся горные ландшафты.

За основу он взял большую геометрическую фигуру, состоящую из нескольких больших треугольников. Используя более крупные треугольники, Карпентер подразделял их на четыре маленьких добавляя случайное отклонение и циклично повторял эту процедуру снова и снова, пока у него не получался горный ландшафт. Таким образом, ему удалось стать первым художником, применившим в компьютерной графике фрактальный алгоритм для построения изображений. Как только стало известно о проделанной работе, энтузиасты по всему миру подхватили эту идею и стали использовать фрактальный алгоритм для имитации реалистичных природных форм.

Через несколько лет Карпентер смог использовать свои наработки в куда более масштабном проекте. Им стал анимационный ролик «Vol Libre», который был показан SIGGRAPH (ежегодная конференция по вопросам компьютерной графики, созываемая организацией ACM SIGGRAPH) в 1982ом году. Данный ролик потряс присутствующую публику и через некоторое время Лорен получил приглашение от LucasFilm. Анимация рендерилась на компьютере VAX-11/780 от Digital Equipment Corporation с тактовой частотой пять мегагерц, причем прорисовка каждого кадра занимала около получаса.

В дальнейшем данная идея была подхвачена некоторыми IT компаниями для создания своих программных продуктов для генерации ландшафтов – Terragen и World Machine, ставшими стандартом в индустрии 3d моделирования, и Game Development’а.

* 1. Основные этапы

При генерации ландшафта основной деталью является карта высот – двумерный массив чисел *(Иллюстрация 2)*, каждое из которых обозначает высоту на участке с данными координатами. После процесса генерации желательно произвести нормализацию (приведение к единичным размерам) получившегося массива, для дальнейшего использования в других алгоритмах.

Второй этап – создание карты биомов. Биомом называется совокупность экосистем одной природно-климатической зоны *(Иллюстрация 3)* [4]. В случае создания ландшафта методами генерации, он определяет какой на данной координате будет климат.

Третьим этапом является генерации дополнительных деталей – расстановка травы, деревьев, камней, и прочих деталей, для повышения детализации и реализма полученного ландшафта на основе получившейся в предыдущем шаге карты биомов.

* 1. Виды алгоритмов генерации карт высот

Самый простой, и непритязательный алгоритм – это заполнение каждого элемента массива случайным числом. Результатом данного метода является двумерный шум, при визуализации которого будут видны очень резкие переходы и спады (от минимума до максимума в пределах одной клетки). В последствии можно применять техники размытия изображения, а также комбинацию различных карт для создания неровностей, чтобы получить нормальный результат. Именно поэтому данный алгоритм имеет право на жизнь.

Каждый алгоритм генерации ландшафтов уникален, и зависит от конкретных целевых потребностей. Но в современном мире сложилось несколько типов алгоритмов генерации:

1. Результат вычисления n-мерных когерентно-шумовых функций
2. Метод «Midpoint-Displacement» и его расширения «Diamond-Square»
3. Алгоритмы на базе диаграмм Воронного (используется в паре с распределителем высот, на основе предыдущих методов) [3]
4. Холмовой алгоритм [2]

Также данные типы могут комбинироваться между собой, для создания гибридных алгоритмов.

* 1. Алгоритм Diamond-Square

Алгоритм Diamond-Square был придуман уже известным нам Лореном Карпентером, как один из экспериментов с алгоритмами генерации (именно он был представлен на SIGGRAPH 1982 в ролике «Vol Libre»).

Данный алгоритм является расширением алгоритма Midpoint-Displacement. Принцип работы данного алгоритма состоит:

Алгоритм Diamond-Square является расширением Midpoint-Displacement, который предназначен для создания двумерных ломаных прямых в двумерном пространстве на плоскость (двумерный массив высот размерностями ). Состоит он из двух шагов: diamond и square, которые исполняются по одному алгоритму, разница состоит в определяемых соседних точках: [1]

1. Определение высот точек-соседей
   1. Diamond – вертикальные и горизонтальные соседи
   2. Square – диагональные соседи
2. Определение значения середины данного квадрата формулой:  
   Где I – количество клеток на сторону квадрата;  
   N – среднеарифметическое угловых значений данного квадрата;  
   R – «Шероховатость».

Работа алгоритма для массива 5х5 показана на иллюстрации 1.

На SIGGRAPH 1986 данный алгоритм был проанализирован Гэвином Миллером, так были обнаружены недостатки данного алгоритма: складки на краях ландшафта, а также резкие перепады высот, и очень крутые горные пики. [3]

* 1. Создание карты биомов

Существуют различные методы создания карт биомов, большинство из них базируются на создании таблицы или диаграммы заданных биомов, ячейки или области которой обозначают биомы, основываясь на значениях высот, влажности или температуры. На их основе определяется климат данного участка карты. Например, снежные шапки на вершинах гор, образующиеся из-за понижения температуры с увеличением высоты, или пустыни, образующиеся ввиду сильной удалённости от водных источников и высокой температуры воздуха. Одним из показательных примеров является диаграмма классификации биомов Ричарда Уиттакера *(Иллюстрация 4)*.

Наиболее распространённым методом создания данной таблицы является построение температурной карты на основе карты высот по принципу уменьшения температуры с увеличением высоты. Другим измерением таблицы является текущий уровень влажности клетки, определяющейся близостью водных источников. А также уровнем осадков, который также определяется на основе водных источников (создания карты испарений и её смещения на расстояние, определяемое случайно сгенерированной розой ветров), что является усложнённым вариантом реализации.

Существует несколько классификаций биомов, включающих от 10 до 32 типов. Распределение биомов происходит по принципу широтной и вертикальной зональностей, а также секторности. На территории России и сопредельных стран выделяют 13 наземных биомов.

### Когерентные шумы

Когерентный шум – один из видов сглаженных псевдослучайных шумов.

Когерентный шум генерируется когерентного-шумовой функцией, которая имеет три важных свойства:

1. Передача одного и того же аргумента всегда будет возвращать то же самое выходное значение.
2. Небольшое изменение аргумента приведет к небольшому изменению выходного значения.
3. Большое изменение аргумента приведет к случайному изменению выходного значения.

Для n-мерной функции когерентного шума требуется n-мерное входное значение. Её выходное значение всегда является скаляром.

Двумерные сложные когерентные шумы могут использоваться для создания карт высот местности без дополнительной обработки, так как они поддаются тонкой настройке. Размер получаемых карт и порядок вычисления отдельных точек не имеют значения.

Все когерентного-шумовые функции имеют ряд характеристик.

* 1. Частота

Частотой когерентного-шумовой функции называют число циклов на единицу длины, которое выдает конкретная функция.

Когерентно-шумовые функции обладают следующими свойствами, подобными свойствам синусоидальной волны:

1. Он имеет периодические циклы длины 1 / f, где f - его частота.
2. В начале каждого цикла оно выдает значение нуля.

В отличие от синусоидальной волны, выход функции когерентного шума может пересекать нуль в середине цикла.

Изменения выходных значений функции в зависимости от частоты показана на иллюстрации 6.

* 1. Многооктавнось

Когерентно-шумовые функции делятся на простые и сложные. Сложные шумы образуются путём последовательного сложения выходных значений простой функции, с каждым шагом изменяющей свои характеристики, подчиняющиеся определённым зависимостям.

Данные функции когерентного шума называются октавами, потому что каждая октава имеет, по умолчанию, удвоенную частоту предыдущей октавы. Музыкальные тона также обладают этим свойством. Музыкальный тон, соответствующий ноте до, который на одну октаву выше предыдущего тона, имеет удвоенную частоту.

Количество октав управляет количеством деталей, что наглядно показано на иллюстрации 7. Добавление большего количества октав увеличивает детальность шума, однако увеличивая временя вычисления.

* 1. Лакунарность

Коэффициент, определяющий, насколько быстро частота увеличивается для каждой последующей октаве в сложной когерентно-шумовой функции. Частота каждой последующей октаве равна произведению частоты предыдущей октавы и значение лакунарности.

* 1. Персистентность

Коэффициент, который определяет, как быстро амплитуда увеличивается для каждой последующей октаве в сложной когерентно-шумовой функции. Амплитуда каждой последующей октавы равна произведению частоты предыдущей октавы и значения персистентности.

### Реализация проекта

* 1. Технологии

На основе приведённых алгоритмов была написана реализация, с демонстрацией результата, выполненная на движке Unity3D, а также плагин (расширение редактора) для создания и дальнейшего использования полученных ландшафтов.

* 1. Отложенные вычисления

Ленивые вычисления *(англ. lazy evaluation, также отложенные вычисления)* – стратегия вычислений, согласно которой расчёты следует откладывать до тех пор, пока не понадобится их результат.

Данная стратегия часто используется в процедурных генераторах реального времени выполнения, используемых в компьютерных играх и технических демонстрациях. Производится расчёт только определённых частей карты, называемых чанками, входящих в заданную дальность видимости в зависимости от положения наблюдателя. Ввиду этого становится возможным генерация огромных, практически бесконечных пространств, ограниченных только вычислительными мощностями современных компьютерных систем.

* 1. Структура модели

Клетка – мельчайшая единица ландшафта, имеющая информацию о местоположении, высоте и ссылки на все применённые модификаторы.

Модификатор – сущность, хранящая особую информацию о клетке, а также методы модификации ландшафта путём взаимодействия с Unity API через абстракцию-обёртку TerrainStorage, направленную на повышение производительности. Модификатор считается применённым, если ссылка на него хранится в экземпляре клетки. Клетка может иметь несколько модификаторов. В программном коде представлен любым классом, имплементирующий интерфейс IModifier (содержащий методы Callback(Cell current) и Apply(TerrainStorage storage)).

Для обеспечения выполнения стратегии отложенных вычислений была создана модель способная рассчитывать клетку, или прямоугольный массив клеток определённого размера, по заданным координатам, применяя все заранее заданные модификаторы. В программном коде она представлена классом Core (содержащим методы GetCell(Pair position) и GetRect(Pair size, Pair position)).

* 1. Используемые модификаторы

Используется данный список модификаторов:

1. Основные, соответствуют биомам данной клетки:
   1. Трава (Grass) – модификатор, наличие которого определяет, что на данной клетке произрастают растения. Проявляет себя в случае, когда высота входит в диапазон от, до уровня произрастания травы до нижней границы уровня скалистой местности. Так же в данной работе был применён алгоритм генерации скал, основанный на подсчитывании дельты перепада высот между четырьмя соседями рассматриваемой клетки.
   2. Скалистая местность (Cliff) проявляет себя в случае, когда высота рассматриваемой клетки входит в диапазон от уровня скалистой местности, до нижней границы уровня снежных шапок
   3. Снежные шапки (AlpSnow) проявляет себя, когда высота рассматриваемой клетки входит в диапазон от верхней границы скалистой местности, до единицы (высота нормализована)
2. Технические, использующиеся для остальных манипуляций, таких как расчёт и применение высоты, генерация и расстановка дополнительных деталей и.т.п:
   1. Генератор высот (BillowHeightModifier) – модификатор, изменяющий высоту клетки. Основан на когерентно-шумовом генераторе Billow Noise, входящим в состав библиотеки LibNoise. Шум считается от текущих координат клетки. Применяется ко всем клеткам.
   2. Лес (RegularForest) – модификатор, отождествляющий дерево, или группу деревьев на клетке. Условием применения является положительное значение бинарной маски, получаемой с помощью вычисления однооктавного перлинового шума с последующим срезом значений меньше определённого порогового значения. Для дополнительного отсеивания создания дерева в случае положительного значения маски происходит с определённым шансом
   3. Камень (Rock) – модификатор, отождествляющий камень, или группу камней на клетке. Применяется с определённым шансом на любую клетку.
   4. Многопоточность и синхронизация

Так как API Unity3d не потокобезопасен и не поддерживает использование любых средств движка из потоков отличных от главного, становится невозможным многопоточный расчёт областей ландшафта, однако остаётся возможным использование мощностей, предоставляемых другими потоками кодом, не использующих методы Unity3d, таких как ядро и система модификаторов. Для того, чтобы связать обе части системы, была написана обёртка, назначение которой состоит в асинхронном расчёте чанка и последующее синхронное применение результата на представление данного чанка, как UnityTerrain.

### Заключение

Процедурная генерация, как видно из данной работы – очень перспективное направление в создании различных информационных продуктов, экономии огромного количества времени и ресурсов. В данной работе были рассмотрены различные методы процедурной генерации, а также их вариации и совокупности. Данная тема пока не имеет широкого обозрения со стороны мира IT, в силу своей инновационности, но неуклонно развивается, с каждым днём находя всё новые и новые области применения.

### Список литературы

1. Алгоритм «diamond-square» для построения фрактальных ландшафтов  
   //” Хабрацентр им. Denull, Хабрахабр”   
   <https://habrahabr.ru/post/111538/>
2. Генерация трёхмерных ландшафтов  
   //” IXBT.com – специализированный российский информационно-аналитический сайт с самыми актуальными новостями из сферы IT”  
   <http://www.ixbt.com/video/3dterrains-generation.shtml>
3. Diamond-Square algorithm  
   //” Wikipedia.org – multilingual, web-based, free-content encyclopedia project supported by the Wikimedia Foundation and based on a model of openly editable content”  
   <https://en.wikipedia.org/wiki/Diamond-square_algorithm>
4. Биом  
   //“ Wikipedia.org – общедоступная мультиязычная универсальная интернет-энциклопедия со свободным контентом”  
   <https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%BE%D0%BC>
5. Polygonal Map Generation for Games  
   //” Red Blob Games”  
   <http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/>
6. Glossary  
   //”Libnoise – a portable, open-source, coherent noise-generating library for C++

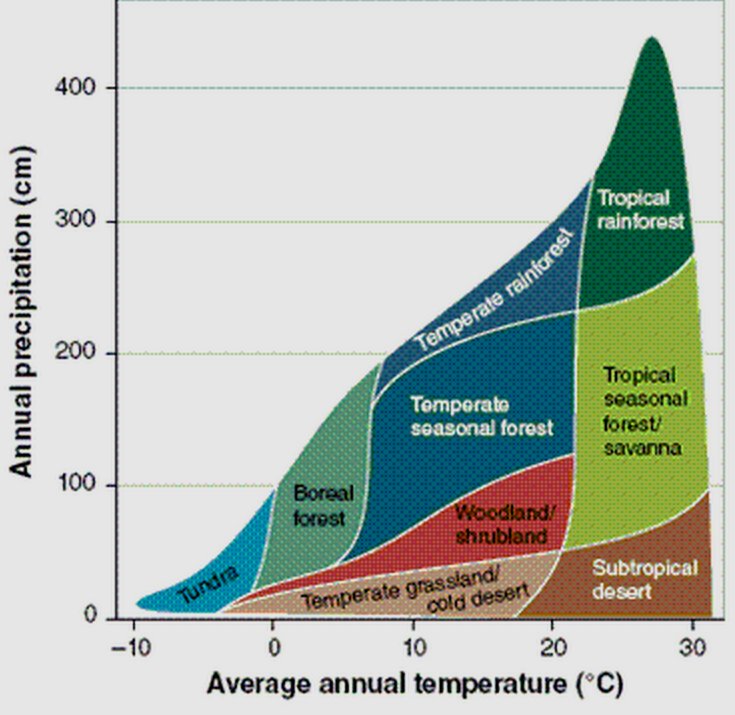
<http://libnoise.sourceforge.net/glossary/index.html>

### Приложения

### C:\Users\student\Downloads\Diamond_Square.svg.png

*Иллюстрация 1 – процесс работы алгоритма Diamond-Square на основе матрицы 5х5 клеток*

|  |  |  |
| --- | --- | --- |
| https://abitu.net/public/jbimages_images/9288de155b6c37121febb89c570149bf.png | | https://abitu.net/public/jbimages_images/22505702365a0ee5af0741f00f8d3e98.png |
| *Иллюстрация 2 – пример карты высот, генерируемых алгоритмом Diamond-Square* | *Иллюстрация 3 – карта биомов, соответствующая иллюстрации* |



*Иллюстрация 4 – диаграмма классификации  
 биомов Ричарда Уиттакера*

|  |  |
| --- | --- |
| Non-coherent-noise function graph  Coherent-noise function graph |  |
| *Иллюстрация 5 – сравнение графиков функций белого и когерентного шумов* | *Иллюстрация 6 – графики когерентно-шумовой функции при разных значениях частоты* |
| Perlin-noise function graph with one octavePerlin-noise function graph with two octaves | Perlin-noise function graph with four octaves  Perlin-noise function graph with eight octaves |
| *Иллюстрация 7 – графики шума Перлина  при различных количествах применямых октав* | |