УДК 004.94

Научная статья

https://doi.org/10.32603/2071-8985-2023-16-2-19-30

Метод процедурной генерации ландшафта с заданными пользователем параметрами

Д. С. Мельниченко, О. В. Букунова✉

Санкт-Петербургский государственный архитектурно-строительный университет, Санкт-Петербург, Россия

✉ bukunovaolga@yandex.ru

Аннотация. Рассмотрены цели внедрения процедурной генерации в игровую индустрию. Поставлена задача генерации карты бесконечных размеров с возможностью генерации по мере передвижения игрока. Предложен способ генерации ландшафта на основе математического метода бикубической интерполяции. Пользователь имеет возможность управлять входными параметрами, влияющими на общий вид результата, но сама карта строится на независимых значениях, что позволяет получать интересные рельефы. Проведено сравнение данного способа генерации с общеизвестным методом шума Перлина. Приведены результаты исследования и разобраны будущие шаги по улучшению данного метода процедурной генерации.

Ключевые слова: процедурная генерация ландшафта, игровая индустрия, бикубическая интерполяция, генерация бесконечной карты, генерация карты в реальном времени

Для цитирования: Мельниченко Д. С., Букунова О. В. Метод процедурной генерации ландшафта с заданными пользователем параметрами // Изв. СПбГЭТУ «ЛЭТИ». 2023. Т. 16, № 2. С. 19–30. doi: 10.32603/2071-8985-2023-16-2-19-30.

Конфликт интересов. Авторы заявляют об отсутствии конфликта интересов.

Original article

A Method for Procedural Generation of a Landscape with User-Defined Parameters

D. S. Melnichenko, O. V. Bukunova✉

Saint Petersburg State University of Architecture and Civil Engineering, Saint Petersburg, Russia

✉ bukunovaolga@yandex.ru

Abstract. The article discusses the goals of introducing procedural generation into the gaming industry. The task is to generate a map of infinite dimensions with the possibility of generation as the player moves. A method for generating a landscape based on the mathematical method of bicubic interpolation is proposed. The user has the ability to control input parameters that affect the overall appearance of the result, but the map itself is based on independent values, which allows you to get interesting reliefs. This generation method is compared with the well-known Perlin noise method. The results of the study are presented and the future steps to improve this method of procedural generation are analyzed.

Keywords: procedural landscape generation, gaming industry, bicubic interpolation, infinite map generation, real-time map generation

For citation: Melnichenko D. S., Bukunova O. V. A Method for Procedural Generation of a Landscape with User-Defined Parameters // LETI Transactions on Electrical Engineering & Computer Science. 2023. Vol. 16, no. 2. P. 19–30. doi: 10.32603/2071-8985-2023-16-2-19-30.

Conflict of interest. The authors declare no conflicts of interest.

Введение. В течение всего существования игровой индустрии главным вопросом были скорость и общая производительность игр. Современные персональные компьютеры позволяют рассчитывать огромные массивы значений, вследствие чего ограничения на «вычислительную» сложность игр снизились. Появилась возможность расчетов в реальном времени без излишней нагрузки на железо компьютера. В связи с этим реально использовать сложные алгоритмы, которые могут без каких-либо проблем вычислять все, что необходимо, в огромных объемах и быстро возвращать результирующие значения. Подобные алгоритмы позволяют отдать большую часть кропотливой работы компьютеру, что и помогает облегчить работу отдельных специалистов в области создания виртуальных объектов или игр. При этом можно не опасаться, что машина не справится или ошибется, тогда как человеческий фактор вносит элемент нестабильности во всю работу [1]. Одним из таких алгоритмов является процедурная генерация ландшафта. Процедурная генерация – незаменимый инструмент гейм-дизайна.

Выделим основные достоинства и недостатки процедурной генерации ландшафта. К преимуществам можно отнести:

1. Использование алгоритма без участия человека, что существенно упрощает сложную работу разработчика.

2. Возможность создания приближенного к бесконечности полотна карты, которую человек будет «просматривать» в течение нескольких лет.

3. Решение важного в наше время вопроса «разрушаемости» карты в игре. При генерации карты процедурным методом будет получена сетка высот, которую без проблем можно изменять, тем самым влияя на саму карту. Когда разработчик создает карту инструментами движка, на котором пишет игру, или другой программой 3D-моделирования, то в большинстве случаев получает нестатическую карту, не хранящую в себе значений высот. Это значительно затрудняет написание методов для изменения полученной сетки карты.

Рассмотрим недостатки:

1. Процедурная генерация (ПГД) обязательно включает в себя составляющую рандомизации значений [2]. Данный факт говорит нам о том, что даже наличие большого количества параметров не гарантирует получение требуемых значений высот. Разработчик, в свою очередь, имеет возможность проработать каждый мелкий элемент.

2. Многие задачи требуют реализации уникальных методов процедурной генерации, так как стандартные методы могут не удовлетворять каким-то важным требованиям задачи.

Сейчас существует несколько основных методов процедурной генерации, но лишь немногие можно применить для общего случая [3]–[6]. Большинство из существующих методов решают узконаправленные задачи, а значит, неприменимы для схожих задач. Несмотря на это, все методы процедурной генерации следуют давно определенным правилам. Все они должны обладать ключевыми свойствами, которые в разной мере необходимы для решения определенной задачи [7].

1. Скорость. Важный параметр, влияющий на скорость разработки программы, если алгоритм отвечает за одноразовую генерацию карты для статической карты в игре. Он влияет на скорость загрузки (создания) уровня игры, если алгоритм срабатывает каждый раз на создание отдельной локации – как разовой, так и той, к которой можно вернуться, и, наконец, на производительность самой игры, если он срабатывает множество раз в ходе игры – например, при генерации карты частями во время перемещений персонажа.

2. Надежность. От надежности алгоритма зависит пригодность его использования. В ПГД программист задает некоторое число исходных параметров, которые призваны регулировать результат, получаемый после срабатывания алгоритма. Необходимо, чтобы алгоритм мог гарантировать наиболее приближенное попадание в начальные условия. Имеет большое значение, что ПГД не просто служит удобным инструментом для ускорения производства игры, а может заменить работу разработчика, в том числе с учетом особенностей игры.

3. Контролепригодность. Алгоритм должен не только попадать в заданные значения, но и иметь достаточное количество входных данных, которыми можно беспрепятственно управлять для получения требуемого в конкретное время результата.

4. Разнообразие. Каждый раз, запуская алгоритм в новой сессии, мы ожидаем увидеть уникальный контент. Выполнение данного пункта гарантирует разнообразие игрового опыта, а следовательно, не вызывает ощущения однообразия игры.

5. Креативность и правдоподобие. В конечном счете – это главный критерий для генерации. Мы создаем алгоритм для работы человека, а не подражания ей. Результат не должен вызывать ощущение нереальности. Игры призваны для погружения в свои миры, а для этого игрок должен верить в происходящее на экране, верить в неподдельность как отдельно взятых частей, так и всего контента в целом.

На текущий момент во всех движках, позволяющих реализовать взаимодействие персонажа и окружающей среды, как и в различных программах для визуализации каких-то ландшафтов, предлагается вручную создавать весь ландшафт проекта. Если это малые проекты, то такая задача занимает от нескольких часов до пары суток. Если же это крупный проект, который требует детальной проработки деталей и продумывания как небольших участков карты, так и ее цельного образа, то такая задача может занимать до нескольких месяцев работы. Также существует необходимость визуализации некоторых эффектов, задач, поверхностей и ландшафтов с возможностью их детального рассмотрения. Визуализация такой задачи в движке методом процедурной генерации позволила бы увидеть результат на новом уровне. Сейчас существует немного методов процедурной генерации, которые решали бы именно эту категорию задач.

В ходе реализации работы потребовалось изучить разные аспекты для улучшения алгоритма:

1. В языке программирования C++ реализован алгоритм генерации случайных чисел. В ходе реализации возникает проблема его использования, так как последовательность генерируется на основе некоторого «семени». Стандартный вариант «семени» – получение времени в секундах, прошедшего с 1 янв. 1970 г. В текущем исследовании такой метод не был действенным, так как подразумевалась возможность построения платформ чаще, чем один раз в секунду. Для корректного выполнения задачи пришлось использовать другой вид случайной генерации. Было реализовано добавление к числу возвращаемого функцией time () в C++ целого числа, получаемого на основе последовательности числа пи.

2. Даже самые мощные и оптимизированные программы для визуализации не могут создать бесконечное количество копий объекта. Большинство этих программ подразумевают постоянное обновление объектов, размещенных на сцене, а все такие объекты хранятся в оперативной памяти устройства для более быстрого обновления. Лучшим решением в таком случае будет размещение копий одного объекта с одним или несколькими различными параметрами. Это позволяет уменьшить количество однообразной копируемой информации о схожих по какому-то критерию объектов. Данный подход уже был реализован в движке Unreal Engine и остается лишь правильно его использовать. За реализацию отвечает метод Instanced Static Mesh. Объект, на основе которого создаются инстансы, разрабатывается с переменными, которые и должны хранить уникальность каждой копии.

Постановка задачи. Целью работы стала реализация уникального метода процедурной генерации ландшафта для внедрения в существующий проект с возможностью взаимодействия между персонажем и сгенерированным ландшафтом, создание карты средствами Unreal Engine на основе реализованного метода, а также выявление его преимуществ и недостатков в сравнении с существующими стандартными методами процедурной генерации. Для выполнения поставленной цели необходимо было решить следующие задачи: иметь возможность создавать карту бесконечных размеров, обладающую кусочной гладкостью; построение карты должно быть максимально независимым от игрока; результирующий вид карты должен имитировать реальный ландшафт, который мы могли бы наблюдать в реальной жизни.

Предлагаемый алгоритм подразумевает быструю и креативную генерацию карты высот. Данную карту каждый разработчик может использовать в зависимости от задачи. Наличие карты обеспечивает доступ ко всем высотам, как отдельно взятой карты, так и больших областей, составленных путем стыковки карт и сглаживания переходов. Алгоритм не имеет ограничения по высотам и имеет хороший уровень сглаживания переходов между различными точками исходных данных.

Быстрота алгоритма позволяет генерировать ландшафт по мере перемещения персонажа по миру, что, в свою очередь, позволяет увеличить размеры конечного мира. Границей размеров может служить лишь количество памяти на жестком диске.

Креативность генерации должна достигаться за счет большого количества параметров, доступ-

ных разработчику. Они управляют направлениями роста, скоростью роста, а также общим значением высот в результирующих данных отдельных карт.

Методы исследования. В работе использовались следующие методы научного исследования: эксперимент, гипотетико-дедуктивный метод, анализ, системный подход. Активно изменялись параметры как платформы, так и всей карты в целом. Оценивалась корректность результирующих значений и влияние параметров на общий вид карты. Выдвигалось множество гипотез, выполнение которых в совокупности давало бы возможность соблюдения всех требований, прилагаемых к карте. Были проанализированы существующие методы процедурной генерации, а также некоторые математические методы. На их основе были сформированы ключевые свойства, которыми должен обладать новый метод процедурной генерации. Исследовалась зависимость между платформами и всей картой. Анализировалось общее поведение карты в зависимости от различных параметров платформ.

В ходе исследования были рассмотрены следующие методы процедурной генерации. Генерация «тайтлами», которая подразумевает сбор карты из уже существующих заготовок, и шум Перлина, который представляет собой алгоритм градиентного шума [8], [9]. Исходя из задач данного проекта было решено, что все перечисленные методы не полностью соответствуют требованиям. Одним из наиболее важных критериев было соблюдение независимости входных данных для получения уникальных рельефов итоговой карты. Потребовался математический метод, на основе которого можно получать кусочно-гладкие поверхности, преобразуя некоторые входные параметры, которыми может управлять пользователь.

Исходя из преимуществ и недостатков различных математических методов построения поверхностей по некоторому набору точек, был выбран наиболее подходящий метод бикубической интерполяции [10]. Он помог преобразовывать независимые входные данные в кусочно-гладкую поверхность. Этот метод позволил написать несложный алгоритм процедурной генерации. Расчет выполняется за короткое время и не сильно нагружает компьютер, что дает возможность строить карту по мере продвижения персонажа без каких-либо задержек по времени. Данный метод подходил под все требования задач исследования. В результате расчетов получался массив данных, который уже можно было использовать для построения карты.

Поверхность, полученная в результате бикубической интерполяции, является гладкой функцией на границах соседних квадратов, в отличие от поверхностей, полученных в результате билинейной интерполяции или интерполяции методом ближайшего соседа.

Решение проблемы. Решалась задача интерполирования значений внутри равномерной квадратной сетки. Общий вид функции, которая задает интерполированную поверхность, записывается следующим образом:

$$\rho(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a(i, j) x^i y^j.$$

Для нахождения всех неизвестных коэффициентов $a(i, j)$ необходимо подставить в вышеуказанную формулу все исходные узловые значения и решить систему уравнений. Далее остается лишь вычислить с помощью функции все значения, лежащие внутри сетки, и передать их в Unreal Engine 5 для дальнейшей визуализации.

Сразу после получения необходимых данных возник вопрос их визуализации. Были проанализированные схожие решения, принятые в популярных играх, таких как: Minecraft, No Man's sky, The Witcher 3, а также предложенные в других проектах [11], [12]. Наиболее подходящим примером стала игра Minecraft. Из нее был интегрирован метод визуализации: разбиение всей карты на кубы, а также разбиение отдельных участков карты на «чанки» (в дальнейшем платформы).

Были проведены опыты с визуализацией разных масштабов. При слишком больших масштабах терялись отдельные элементы рельефа, тогда как при малых размерах терялась возможность прохождения по карте персонажа, а также ее схожесть с ландшафтом земли. В итоге за стандартный размер одной платформы был принят квадрат со сторонами 50 × 50.

Так как все узловые значения должны быть независимы, необходимо создать алгоритм, регулирующий корректность данных. Также надо добавить правила, по которым будут создаваться значения в зависимости от текущего типа местности. Было принято решение о введении нового параметра, отвечающего за дестабилизацию внутри платформы. Он генерировался на основе текущей местности.

На движке Unreal Engine была реализована функция, которая строила одну платформу. Функ-

ция получала начальные данные, а именно общую высоту платформы и процент дестабилизации значений в узловых точках сетки. Каждое значение в сетке получалось отклонением от общей высоты (в процентах) дестабилизации платформы с помощью рандомизации языка программирования C++. Далее мы возвращали большой просчитанный массив значений размером 50 × 50. Функция по полученным значениям строила платформу из кубов, последовательно размещая их на сцене.

В результате получалась платформа, которая полностью удовлетворяла задачам работы. Однако из-за метода визуализации появилась проблема: когда между двумя соседними элементами платформы была разница более двух кубов, можно было наблюдать зазор в карте. Эту проблему удалось решить, возвращая значения разницы между текущим элементом платформы и его соседом. Далее мы заполняли зазоры нужным количеством кубов. Таким образом получилась цельная платформа, состоящая из 2500–2600 кубов.

Позже выяснилось, что при размещении более 20 платформ стало не хватать ОЗУ компьютера. С этим справились, реализовав возможность движка Unreal Engine – Instanced Static Mesh. Данная реализация позволила создавать платформы, размещая несколько тысяч копий одного объекта. Каждая копия имела возможность хранить в себе заранее определенные параметры и включать в себя все базовые параметры начального объекта. Это решение помогло снизить нагрузку на ОЗУ и дало возможность размещать большое количество платформ.

Так как платформы были полностью независимыми, т. е. не соединяющимися с соседними, были получены зазоры в координатах z между стыковыми элементами соседних платформ. Данная проблема была решена «соединением» крайних узловых значений каждой пары соседних платформ (рис. 1). В рамках этого решения появилась необходимость строить платформы последовательно, так как для передачи крайних узловых значений предыдущей платформы требовалось ее построение. Согласно цели проекта, персонаж должен был проходить по карте по очереди, такое ограничение последовательного построения карты не создавало неудобств пользователю.

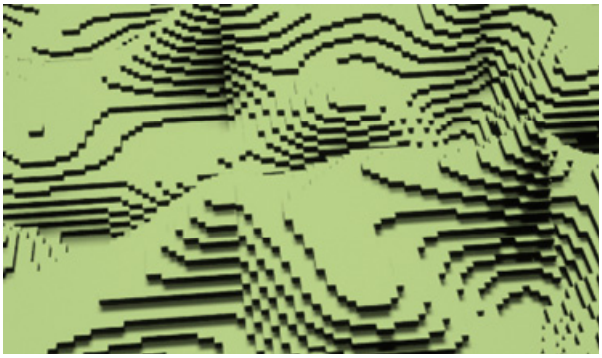

Рис. 1. Вид сверху на стыки платформ
Fig. 1. Top view of the platform joints

Из используемого математического метода сформулирована следующая проблема. Не было возможности контролировать элементы между крайними узлами сетки, так как не было корректирующих условий на границе сетки. Из стыковки платформ посредством приравнивания крайних узловых значений соседних платформ следовало, что элементы между стыковыми узлами могли сильно отличаться и иметь зазоры между этими платформами. Для устранения таких зазоров было выбрано три возможных варианта сглаживания стыков: соединение платформ «усредненной высоты»; подстраивание элементов одной платформы таким образом, чтобы они сходились к значениям второй; создание алгоритма, который будет сглаживать стыки, исходя из текущего рельефа. Рис. 2 демонстрирует аккуратные переходы между платформами.

Далее необходимо было создать правило, по которому присваиваются базовые высоты платформ, чтобы рельеф создавался в рамках не только платформы, но и всей карты. Решением стало использование того же алгоритма, что и для платформ (рис. 3). Такой выбор позволял сделать карту кусочно-гладкой, но не давал получить на ней интересный рельеф в виде озер, рек, впадин и ущелий. Была добавлена функция постобработки, которая добавляла недостающие элементы на уже готовую карту. На этом этапе возникло противоречие цели – генерация карты по мере продвижения персонажа. Этот вопрос был решен добавлением новой опции. Теперь карта просчитывалась сразу на некоторое количество платформ вперед, тем самым давая возможность использовать функцию постобработки до момента отображения платформы.

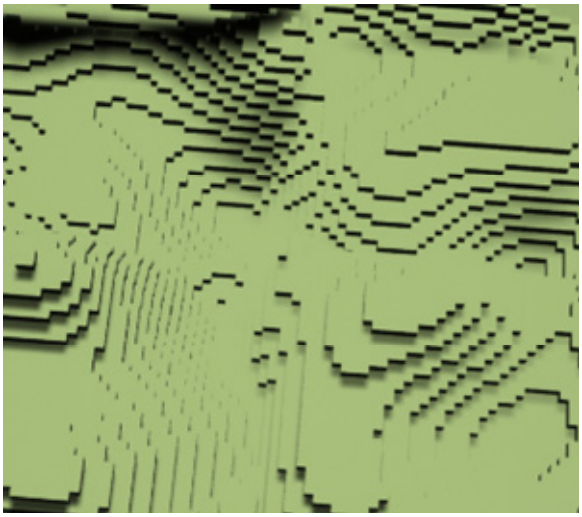

Рис. 2. Вид сверху на платформы
Fig. 2. Top view of the platforms

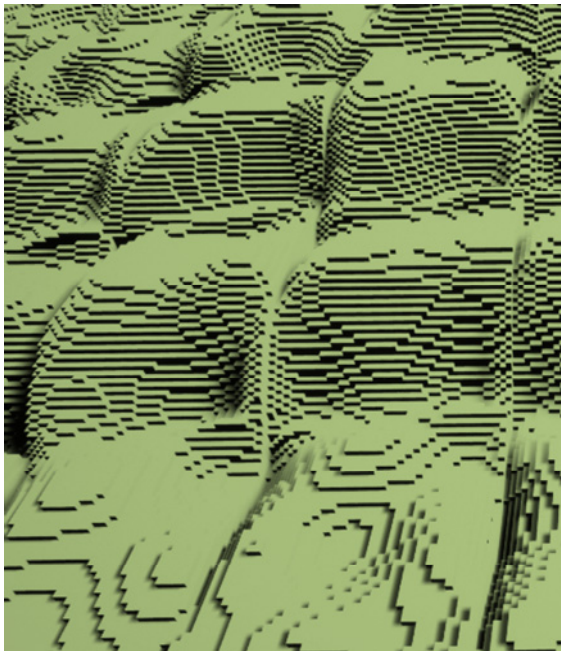

Рис. 3. Кусочно-гладкие платформы
Fig. 3. Piecewise smooth platforms

Оптимизация визуализации не полностью решила проблему загрузки памяти. Было принято решение – ввести параметр дальности прорисовки, который бы контролировал количество отрисовываемых карт на данный момент, и тем самым давал установить лимиты на использование памяти. Постоянно контролировалось количество входящих в радиус прорисовки платформ. В случае выхода из этого радиуса срабатывал алгоритм, который скрывал и сохранял положение элементов платформы. Такое решение упростило сохранение дополнительно размещенных элементов на платформах.

Текущее решение позволяет построить бесконечную, независимую от игрока, карту. На рис. 4 можно наблюдать итоговый вид карты от лица персонажа. Вся карта кусочно-гладкая и полностью «проходимая» для персонажа. Исходя из этого можно сказать, что цели и задачи, которые ставились в данной работе, успешно выполнены.

На этом этапе можно выделить следующие перспективы развития:

1. Будет доработан алгоритм стыка платформ, в котором алгоритм сглаживания будет автоматически выбирать, каким образом лучше сгладить стык между двумя соседними платформами.

2. Будет улучшен алгоритм генерации уникальных рельефов. Добавится показатель по проценту уникальных зон, исходя из которого алгоритм сможет равномерно распределять их по всей бесконечной карте.

3. Изменится алгоритм определений узловых значений, что позволит создавать более приближенные к реальности части отдельных платформ.

4. Будут улучшены алгоритмы сохранения данных и загрузки их из памяти в ОЗУ.

5. Персонажу будет добавлена возможность влиять на генерацию кусков карты, что позволит создавать уникальные рельефы, необходимые пользователю в каком-либо месте.

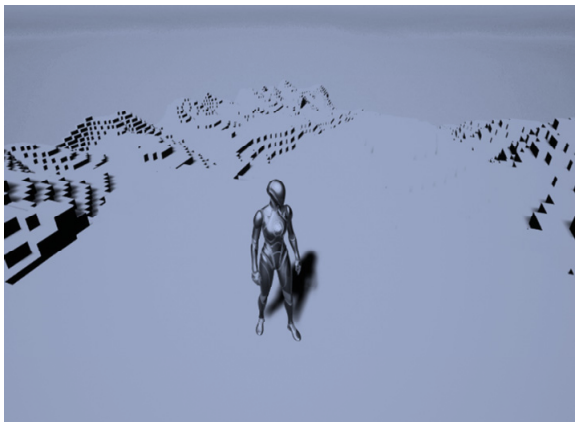

Рис. 4. Вид от лица игрока на часть построенной карты
Fig. 4. Player's view of a part of the constructed map

Результат необходимо было сопоставить с проектами, в которых применялась процедурная генерация ландшафта. Потребовалось получить некоторую статистику, которая помогла бы определить преимущества и недостатки данной работы.

Первой для сравнения была выбрана игра Minecraft. Рис. 5 демонстрирует ее визуализацию, так как с ней проще всего сопоставить результаты.

В Minecraft дан довольно малый спектр параметров, влияющих на генерацию ландшафта. Рис. 6 демонстрирует возможности настройки процедурной генерации игры Minecraft.

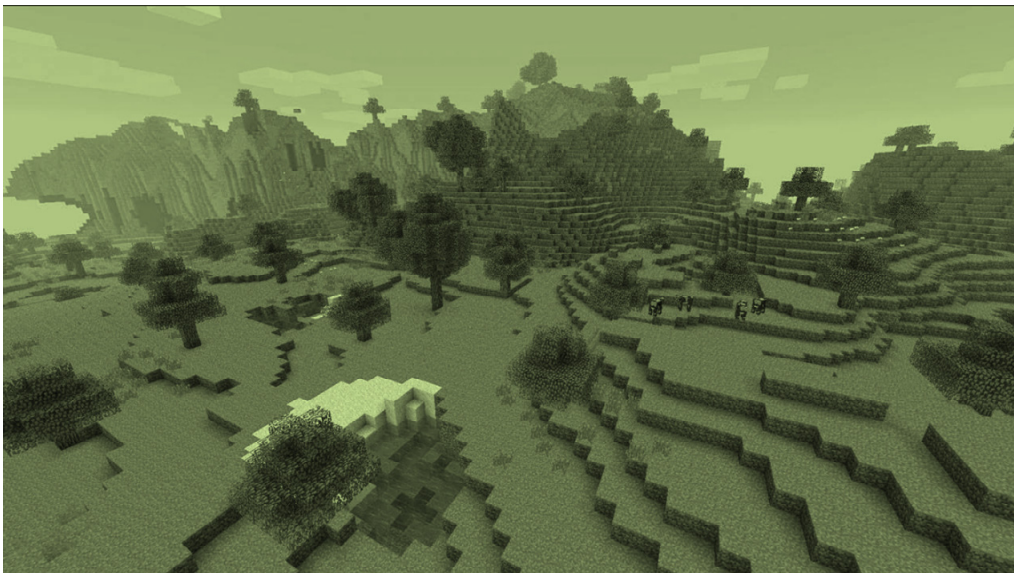

Рис. 5. Вид карты игры Minecraft
Fig. 5. Map view of the Minecraft game

Рис. 6. Настройки мира в игре Minecraft
Fig. 6. World settings in the Minecraft game

В текущей работе не реализован пользовательский интерфейс, однако уже можно влиять на параметры генерации внутри программы. Нам доступны параметры:

1. Общая дестабилизация мира.

2. Параметр, отвечающий за настройки рельефа в зависимости от текущего типа местности. Тем самым имеется возможность настроить визуальный вид каждой местности вроде гор, холмов и т. д.

3. Количество определенных типов местности (в процентах).

4. Количество уникальных рельефов.

Несмотря на малое количество параметров при создании мира, в Minecraft всегда генерируется уникальный и разнообразный контент. Кроме того, имеется возможность повторить какой-то уже сгенерированный мир, если будет указано одно и то же семя генерации. В настройках игры это семя называется «ключ для генератора мира» (табл. 1).

При использовании процедурной генерации мы всегда будем получать уникальный контент при каждом запуске генератора. Отсутствие возможности ввода семени и его случайная генерация средствами C++, а также метод, который позволяет генерировать карту по мере передвижений персонажа, позволяет создать бесконечное количество вариаций карты.

Стоит отметить, что метод процедурной генерации Minecraft – шум Перлина. Это позволяет использовать трехмерный вариант генерации ландшафта, а значит, возможность генерации пе-

Табл. 1. Сравнение главных параметров в процедурной генерации (Minecraft)
Tab. 1. Comparison of the main parameters in procedural generation

Параметр	Представленная работа	Minecraft
Скорость	Возможность быстрой генерации в реальном времени	Быстрая, но единоразовая генерация всего мира, занимающая около 20 с
Надежность	Полная проверка целостности карты. При указании критически больших параметров возможность генерации карты, для прохождения которой необходимо будет задействовать средства взаимодействия игры	Полная проходимость карты на любой локации без необходимости дополнительного взаимодействия с игрой
Контролепригодность	Возможность задания большого количества параметров, которые позволяют задать многие детали ландшафта и рельефов	Указание семени, которое может выдавать разные варианты карты. Отсутствие параметров для пользователя, влияющих на итоговый вид карты
Разнообразие	Бесконечное количество разнообразных вариаций карты. Наличие возможности влиять на генерацию отдельных участков ландшафта	Огромное, но ограниченное количество вариаций карты. Единоразовая генерация карты без какого-либо участия пользователя
Креативность и правдоподобность	Хорошее подражание реальному ландшафту. Получение действительно уникальных видов, а также реалистичный вид смены разных видов местности	Наличие многих типов местности, отлично подражающих реальным видам местности. Высокий уровень детализации, который увеличивает уровень достоверности

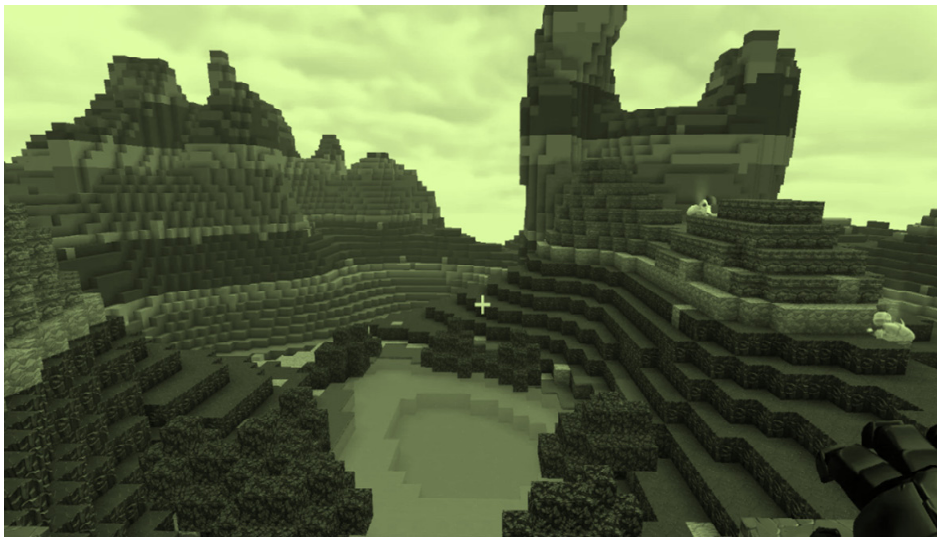

Рис. 7. Вид карты игры Creativerse
Fig. 7. Map view of the Creativerse game

щер и различных подземелий, тогда как представленная работа предлагает только генерацию поверхностного полотна карты.

Второй игрой для сравнения результатов стала схожая с Minecraft по подходу визуализации игра Creativerse (на рис. 7 можно видеть, как она выглядит), но сами локации и подход к генерации заметно отличаются. Мы видим красивые местности на карте, несколько уходя от схожести с реальным миром.

В отличие от Minecraft, Creativerse предоставляет больший выбор параметров для влияния на игру рис. 8 демонстрирует настройку генерации мира. Эти параметры отвечают за общую процедурную генерацию мира, включая многие аспекты игры.

Процедурная генерация Creativerse не подлежит регулированию со стороны пользователя, но выдает хорошие результаты (табл. 2).

Третий вариант игры с процедурной генерацией – Spore [13]. Часть контента данной игры происходит на процедурно-сгенерированных планетах. На рис. 9 можно наблюдать одну из процедурно-сгенерированных карт. Одним из подходов игры считается соответствие планет неким законам формирования планет во вселен-

ной. Многие планеты схожи с Землей и генерируются по тем же правилам. Эта игра использует свой метод процедурной генерации ландшафта планет, который не основан на каких-то классических вариантах.

Рис. 8. Настройки мира в игре Creativerse
Fig. 8. World settings in the Creativerse game

Здесь весь контент сгенерирован изначально и повторяется от игрока к игроку. Мы не можем повлиять на этот процесс, а значит, всегда будем получать одни и те же планеты, которые сформировались однажды по параметрам, указанным лишь разработчиками.

Планеты в игре Spore достаточно интересны, но когда исследуется уже 10-я планета, то становится очевидным, что все планеты очень схожи. Данный метод генерации основывается на размещении нескольких тысяч копий одной планеты с небольшим варьированием параметров. Рассмотрим важное преимущество использования процедурной генерации в игровой индустрии. Разработчики после генерации всех планет выборочно изменяли некоторое их количество, оставляя «пасхалки» для игроков. Несмотря на процедурную генерацию всех планет независимо от разработчиков, этот подход дал возможность постобработки результатов.

Преимуществом метода, представленного в данной статье, служит более высокое содержание уникального контента (в процентах) и возможность управления видом итоговой карты. Также надо упомянуть о генерации карты в режиме реального времени и возможности легкого изменения карты персонажем (табл. 3).

В результате мы получили возможность строить и демонстрировать ландшафт. Данные воз-

Табл. 2. Сравнение главных параметров в процедурной генерации (Creativerse)
Tab. 2. Comparison of the main parameters in procedural

Параметр	Представленная работа	Creativerse
Скорость	Возможность быстрой генерации в реальном времени	Единоразовая минутная генерация карты
Надежность	Полная проверка целостности карты. При указании критически больших параметров, возможность генерации карты, для прохождения которой необходимо будет задействовать средства взаимодействия игры	В игре существуют локации, сгенерированные процедурно, которые невозможно пройти игроку без использования некоторых особенностей игры, включающих «разрушение» карты. Это говорит о недостаточной надежности алгоритма. Данная проблема решается взаимодействием с игрой
Контролепригодность	Возможность задания большого количества параметров, которые позволяют задать многие детали ландшафта и рельефов	Пользователь никак не может повлиять на генерацию ландшафта и его надежность зависит лишь от кода игры
Разнообразие	Бесконечное количество разнообразных вариаций карты. Наличие возможности влиять на генерацию отдельных участков ландшафта	Огромное, но ограниченное количество вариаций карты. Единоразовая генерация карты без какого-либо участия пользователя
Креативность и правдоподобность	Хорошее подражание реальному ландшафту. Получение действительно уникальных видов, а также реалистичный вид смены разных видов местности	Генерация ландшафта ставит упор на создание уникальных локаций, аналоги которых могут не существовать в мире. Тут целесообразно оценить саму креативность, которая реализована очень хорошо

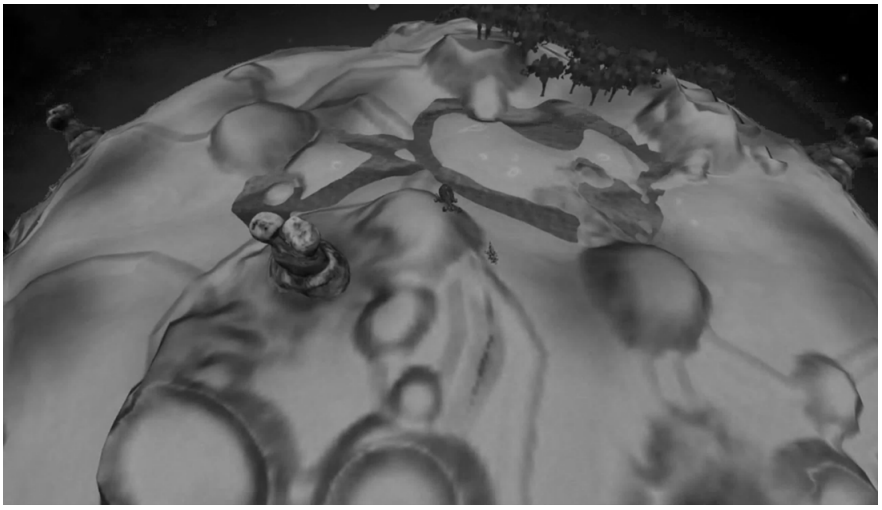

Рис. 9. Вид процедурно-генерируемых планет игры Spore
Fig. 9. View of the procedurally generated planets of the Spore game

Табл. 3. Сравнение главных параметров в процедурной генерации (Spore)
Tab. 3. Comparison of the main parameters in procedural

Параметр	Представленная работа	Spore
Скорость	Возможность быстрой генерации в реальном времени	Процедурная генерация применялась на момент создания игры
Надежность	Полная проверка целостности карты. При указании критически больших параметров, возможность генерации карты, для прохождения которой необходимо будет задействовать средства взаимодействия игры	Данная игра не подразумевает полного использования планет, но в ней нельзя найти ни одной планеты, которая вследствие процедурной генерации имела бы дефект
Контролепригодность	Возможность задания большого количества параметров, которые позволяют задать многие детали ландшафта и рельефов	Пользователь не может повлиять на генерацию планет
Разнообразие	Бесконечное количество разнообразных вариаций карты. Наличие возможности влиять на генерацию отдельных участков ландшафта	Можно наблюдать множество планет, но со временем начинаем замечать похожие друг на друга планеты, что говорит об ограниченности вариаций планет
Креативность и правдоподобность	Хорошее подражание реальному ландшафту. Получение действительно уникальных видов, а также реалистичный вид смены разных видов местности	Вид планет не может не поражать. Несмотря на то, что можно заметить схожесть некоторых планет, прекрасно видна креативность каждой планеты. Все они создавались на основе общих знаний о различных планетах, что показывает на правдоподобие каждой отдельно взятой планеты

можности определяют широкий спектр для использования этого метода процедурной генерации ландшафта.

В первую очередь, это использование метода для создания игр. Многие компании, которые производят игры, нуждаются в сокращении времени создания карт и локаций. Данный метод позволяет создавать реалистичный контент, а значит, вполне подходит для внедрения в игры, от которых сейчас требуют максимальную реалистичность и высокое качество графики. Данную процедурную генерацию легко внедрить в любые проекты, а также она достаточно гибка и разработчик, желающий несколько видоизменить алгоритм, без проблем сможет это реализовать.

Во вторую очередь, это визуализация поверхностей или расчетов. Многие математические пакеты не могут полностью продемонстрировать результаты вычислений, что сильно снижает скорость и понимание результатов пользователем

(в процентах). Пользуясь предложенным методом, можно визуализировать вычисления и дать возможность «пощупать» их. Как упоминалось ранее, мы легко можем взаимодействовать с построенной поверхностью, что открывает множество перспектив для использования построенной поверхности.

Также можно реализовать программу, которая позволит создавать ландшафт на основе рисунков или заданных особым образом областей, что дает возможность применения метода в образовательных программах для профессий, связанных с созданием, изменением или демонстрацией какой-то области, содержащей ландшафт.

Сейчас существуют профессии, которые подразумевают создание какого-то визуального контента для демонстрации проектов домов, дизайна персонажей на фоне определенных локаций, создание определенных сцен с уникальным ландшафтом. Все они подразумевают создание ландшафта с нуля. Зачастую это требует больше времени, чем создание основного содержания визуализации, и ставится вопрос целесообразности траты этого времени. Используя текущую работу, мы можем снять эту задачу, позволяя генерировать ландшафт, автоматически задав несколько необходимых параметров. Результирующая поверхность может быть откорректирована, что говорит о гибкости данного решения. Скорректировать какие-то детали может быть быстрее, чем создать локацию с нуля.

Сама возможность быстрого построения независимого динамического ландшафта позволяет демонстрировать различные динамические эффекты. Это может быть генерация водной поверхности, визуализация музыки, что интересует многих музыкантов, демонстрация каких-то последствий воздействия на ландшафт, а также возможность создания мира пользователем, находящимся внутри самой программы и не изменяющим код алгоритма.

В заключение можно сказать, что направление процедурной генерации очень востребовано в наше время. Можно утверждать, что сейчас существует недостаточно универсальных методов, которые позволяют не только решать любую задачу, но и выбирать, какие ключевые свойства процедурной генерации в этой задаче приоритетны. Исходя из данной работы, можно утверждать следующее: использование метода бикубической интерполяции в процедурной генерации дает возможность создания метода генерации ландшафта средствами Unreal Engine, который по многим параметрам можно оценить как достаточно хороший и применимый в различных областях.

Список литературы

1. Мельниченко Д. С. Метод процедурной генерации ландшафта // Материалы LXXV Науч.-практ. конф. студентов, аспирантов и молодых ученых «Актуальные проблемы современного строительства». СПб., 2022. С. 15–22.

2. Генерация псевдослучайных чисел в программировании. И как у меня псевдо-получилось их сгенерировать. URL: https://habr.com/ru/post/686190/ (дата обращения 11.10.2022).

3. Никулин Е. А. Процедурный шум и мультивариантный тайлинг // Тр. НГТУ им. Р. Е. Алексеева. 2016. № 1 (112). С. 17–24.

4. Колесников С. В. О решении задачи генерации ландшафтов // Решетневские чтения. 2013. Т. 2. С. 214–215.

5. Федоров К. Б. Процедурная генерация ландшафтов / науч. рук. О. Б. Фофанов // Молодежь и современные информационные технологии: сб. тр. XVI Междунар. науч.-практ. конф. студентов, аспирантов и молодых ученых. Томск: Изд-во ТПУ, 2019. С. 500–501.

6. Меженин М. Г. Обзор систем процедурной генерации игр // Вестн. ЮУрГУ. Сер. «Вычислительная математика и информатика». 2015. Т. 4, № 1. С. 5–23.

7. Грег С. Н. Создание 3D-ландшафтов в реальном времени с использованием C++ и DirectX. 2-е изд., перераб. и доп. СПб.: КУДРИЦ-Образ, 2006. 368 с.: ил.

8. Сальникова П. В. Процедурная генерация контента: бакалаврская работа. СПб., 2016. 42 с. URL: https://nauchkor.ru/uploads/documents/587d362e5f1be77c40d588b6.pdf?ysclid=lbw7xzyqf3912164476 (дата обращения 11.10.2022).

9. Барсуков А. В. Применение алгоритма Midpoint-displacement в процедурной генерации ландшафтов // Науч.-образоват. журн. для студентов и преподавателей «StudNet». 2020. № 2. С. 469–478.

10. Keys R. Cubic convolution interpolation for digital image processing // IEEE Transactions on Acoustics, Speech, and Signal Proc. 1981. Vol. 29, no. 6. P. 1153–1160. doi: 10.1109/TASSP.1981.1163711.

11. Селянин Н. А. Система трехмерного моделирования ландшафта. Выпускная квалиф. работа: URL: http://elar.uspu.ru/bitstream/uspu/14697/1/Selyanin2.pdf (дата обращения 12.10.2022).

12. Болтов Ю. Ф., Дудаков Л. С., Тарлыков А. В. Решение задачи визуализации процедурно генерируемого ландшафта при помощи алгоритма на основе структуры «Дерево квадрантов». // Тр. учеб. заведений связи. 2016, № 2. С. 28–33.

13. Как создавали Spore: интервью с разработчиками. URL: https://habr.com/ru/post/440154/ (дата обращения 12.10.2022).

Информация об авторах

Мельниченко Дмитрий Сергеевич – студент Санкт-Петербургского государственного архитектурно-строительного университета. СПбГАСУ, Санкт-Петербург, Российская Федерация, 190005, 2-я Красноармейская, д. 4.
E-mail: dminryi.m@mail.ru

Букунова Ольга Викторовна – канд. техн. наук, доцент Санкт-Петербургского государственного архитектурно-строительного университета. СПбГАСУ, Санкт-Петербург, Российская Федерация, 190005, 2-я Красноармейская, д. 4.
E-mail: bukunovaolga@yandex.ru
ORCID: 0000-0002-5721-1795

References

1. Mel'nichenko D. S. Metod procedurnoj generacii landshafta // Materialy LXXV Nauch.-prakt. konf. studentov, aspirantov i molodyh uchjonyh «Aktual'nye problemy sovremennogo stroitel'stva». SPb., 2022. S. 15–22. (In Russ.).

2. Generacija psevdosluchajnyh chisel v programmirovanii. I kak u menja psevdo-poluchilos' ih sgenerirovat'. URL: https://habr.com/ru/post/686190/ (data obrashheniya 11.10.2022). (In Russ.).

3. Nikulin E. A. Procedurnyj shum i mul'tivariantnyj tajling // Tr. NGTU im. R. E. Alekseeva. 2016. № 1 (112). S. 17–24. (In Russ.).

4. Kolesnikov S. V. O reshenii zadachi generacii landshaftov // Reshetnevskie chtenija. 2013. T. 2. S. 214–215. (In Russ.).

5. Fjodorov K. B. Procedurnaja generacija landshaftov / nauch. ruk. O. B. Fofanov // Molodezh' i sovremennye informacionnye tehnologii: sb. tr. XVI Mezhdunar. nauch.-prakt. konf. studentov, aspirantov i molodyh uchjonyh. Tomsk: Izd-vo TPU, 2019. S. 500–501. (In Russ.).

6. Mezhenin M. G. Obzor sistem procedurnoj generacii igr. // Vestn. JuUrGU. Ser. «Vychislitel'naja matematika i informatika». 2015. T. 4, № 1. S. 5–23. (In Russ.).

7. Greg S. N. Sozdanie 3D-landshaftov v real'nom vremeni s ispol'zovaniem S++ i DirectX. 2-e izd., pererab. i dop. SPb.: KUDRIC-Obraz, 2006. 368 s.: il. (In Russ.).

8. Sal'nikova P. V. Procedurnaja generacija kontenta: bakalavrskaja rabota. SPb., 2016. 42 s. URL: https://nauchkor.ru/uploads/documents/587d362e5f1be77c40d588b6.pdf?ysclid=lbw7xzyqf3912164476 (data obrashheniya 11.10.2022). (In Russ.).

9. Barsukov A. V. Primenenie algoritma Midpoint-displacement v procedurnoj generacii landshaftov // Nauch.-obrazovat. zhurn. dlja studentov i prepodavatelej «StudNet». 2020. № 2. S. 469–478. (In Russ.).

10. Keys R. Cubic convolution interpolation for digital image processing // IEEE Transactions on Acoustics, Speech, and Signal Processing. 1981. Vol. 29, no. 6. P. 1153–1160. doi: 10.1109/TASSP.1981.1163711.

11. Seljanin N. A. Sistema trehmernogo modelirovanija landshafta. Vypusknaja kvalifikacionnaja rabota: URL: http://elar.uspu.ru/bitstream/uspu/14697/1/Selyanin2.pdf (data obrashheniya 12.10.2022). (In Russ.).

12. Boltov Ju. F., Dudakov L. S., Tarlykov A. V. Reshenie zadachi vizualizacii procedurno generiruemogo landshafta pri pomoshhi algoritma na osnove struktury «Derevo kvadrantov» // Tr. ucheb. zavedenij svjazi. 2016. № 2. S. 28–33. (In Russ.).

13. Kak sozdavali Spore: interv'ju s razrabotchikami. URL: https://habr.com/ru/post/440154/ (data obrashheniya 12.10.2022). (In Russ.).

Information about the authors

Dmitry S. Melnichenko – student of Saint Petersburg State University of Architecture and Civil Engineering. Saint Petersburg, Russian Federation, 190005, 2nd Krasnoarmeyskaya, 4.
E-mail: dminryi.m@mail.ru

Olga V. Bukunova – Cand. Sci. (Eng.), Assistant Professor of Saint Petersburg State University of Architecture and Civil Engineering. Saint Petersburg, Russian Federation, 190005, 2nd Krasnoarmeyskaya, 4.
E-mail: bukunovaolga@yandex.ru
ORCID: 0000-0002-5721-179

Статья поступила в редакцию 30.11.2022; принята к публикации после рецензирования 25.12.2022; опубликована онлайн 28.02.2023.

Submitted 30.11.2022; accepted 25.12.2022; published online 28.02.2023.