Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка
Кафедра програмних систем і технологій

ПРОЦЕДУРНА ГЕНЕРАЦІЯ ТЕРРЕЙНІВ

Курсова робота
Пояснювальна записка

Виконавець
студент групи ІПЗ-34-МС _____ Станіслав САВЕНКО
(підпис, дата)
Керівник
доцент кафедри ПСТ _____ к.ф.м.н. Сергій ДОЦЕНКО
(підпис, дата)

Київ-2021

ЗАВДАННЯ НА КУРСОВУ РОБОТУ

Тема роботи : «Процедурна генерація террейнів»

Строк здачі студентом закінченої роботи: 1 червня 2021

Вихідні дані до роботи:

Було розглянуті основні моменти роботи з террейнами. А саме були визначені основні типи алгоритмів генерації даних:

- Фрактальні
- Нефрактальні

В ході курсової роботи були визначенні основні способи зберігання інформації террейнів:

- Карти височин
- Іррегулярні сітки
- Сегменті карти височин

Були розглянуті способи виведення ландшафтів:

- Скалярні
- Воксельні

При виконанні робти задачі розглядалися як на практиці, так і в теорії, що надало знання та навички у роботі з террейнами.

АНОТАЦІЯ

Актуальність.

На даний момент медіа та ігрова індустрія набуває великої популярності. Для набуття більш складного зображення або для економії коштів, починається використовуватись комп’ютерна графіка, також з’явилися анімаційні мультфільми. Щоб отримати більш реалістичну гру використовується багато графічних засобів, для створення більш реалістичних графічних рішень. Симуляції не обійшли комп’ютерну графіку стороною та також її використовують у своїх цілях.

Одним з важливих елементів комп’ютерної є террейни, оскільки вони зображають велику площу навкруги. І не дивлячись на розмір террейнів вони є дуже економними в плані ресурсів, що дозволяє їх повсякденно використовувати.

Об'єктом дослідження є процес генерації данних террейнів та повноцінного виводу даних у вигляді террейнів.

Предметом дослідження є способи та алгоритми створення та виведення ландшафтів.

Метою дослідження є отримання даних, які використовуються для сучаних способів та алгоритмів для генерації террейнів, виводу їх для звичайних користувачів, взаємодію них з користувачем.

Методи дослідження в роботі використовуються методи комп’ютерного моделювання, статистичні та емпіричні методи.

Практична цінність роботи полягає в можливості ознайомитися з методологіями роботи з ландшафтами, отримання навичок процедурної генерації ландшафтів, роботу з ними, тощо.

ЗМІСТ

ВСТУП

Ця курсова робота присвячена дослідженню задач генерації випадкових даних, роботою над методами процедурної генерації даних та представленні їх у вигляді ландшафтів.

На даний момент вміння генерувати террейни є одним з важливим для роботи Game розробників, розробників симуляцій, оскільки правильна робота з террейнами передбачає можливість створювати масштабні карти для анімацій, ігор та симуляцій. Дана технологія надає можливість створювати ландшафт мінімально затрачуючи ресурс продуктивності, що надає можливість використовувати дану технологію не тільки зверх-продуктивних комп’ютерах, а ще й на рядових машинах, що надає можливість використовувати террейни на велику аудиторію.

До того ж технологія є непростою та дуже цікавою. Є немало способів генерації випадкових даних для генерації псевдовипадкових даних, при цьому дані можна зберегти декількома способами, кожний з яких має свої сильні сторони перед іншими. Також представлень даних в виді террейну є безліч.

При цьому є багато можливостей описати взаємодію користувача з ландшафтом. Хтось може використовувати ландшафти тільки у дизайнерських цілях, хтось може використовувати, щоб працювати з тунелями та розломами.

1. ТЕОРЕТИЧНІ ОСНОВИ ГЕНЕРАЦІЇ ТЕРРЕЙНІВ

1.1 Задачі генерації террейнів

Моделювання рельєфу місцевості відіграє важливу роль комп’ютерній графіці . Автоматична генерація місцевостей має багато застосувань в районах, починаючи від формування ландшафту для медіа, генерації випадкових середовищ для ігор та генерації місцевостей для різних тренувань та симуляцій.

Генерація процедурного ландшафту має мати три важливі особливості. По-перше, генерація ландшафту повинна бути швидкою, а це означає, що вона повинна використовувати лише частку обчислювальної потужності комп'ютерної системи. По-друге, вона повинна бути, як випадковою, так і структурованою таким чином, щоб створювати різноманітний і реалістичний вміст. По-третє, генерація повинна бути керованою і інтуїтивною.

Ландшафт використовується для двох важливих цілей у візуалізації сцени. Террейн є основним, окремим будівельним блоком на якому розміщуємо всі інші об’єкти на сцені. Розміщення всіх інших об’єктів на ньому також допомагає швидше розпізнавати напрям «вгору» та орієнтацію у віртуальному просторі. Ландшафт також використовуються для значного збільшення загального реалізму всієї сцени. Якщо наведені віртуальні об’єкти розміщені на поверхні ландшафту, то загальне відчуття сцени стане набагато достовірнішим та реалістичним. Саме тому поверхня повинна містити деталі, які не можуть бути повністю відтворені лише застосовуючи прості текстури.

Генерація ландшафтів має велике значення для результуючого програмного продукту. Від якості зовнішньої сторони, а також комфорту ігрового процесу напряму залежить прибутковість гри, її сприймання користувачами, а також подальший розвиток, тому створення нових, оптимізованих методів генерації ландшафтів є актуальною практичною задачею.

1.2 Основи генерації ландшафтів

Ландшафт – це відображення ділянки землі, яке розглядається з урахуванням її природних особливостей, військових переваг тощо. Ландшафт або рельєф (також топографічний рельєф) передбачає вертикальні та горизонтальні розміри поверхні суші.

Дані для генерування ландшафту можуть бути згенеровані, декількома шляхами. Оскільки ландшафт представлений у виді комп'ютерної графіки, його цифрові дані можна згенерувати 4 способами:

- Генерація ландшафту на основі реальної місцевості:

За допомогою сучасних ресурсів можна отримати цифровий вид реальної місцевості, та згенерувати цифровий вид ландшафту на основі отриманих даних. Цей спосіб надає велику гнучкість і можливість редагувати будь-яку частину террейну.

- Моделювання ландшафту:

Створення місцевості за допомогою заздалегідь розроблених інструментів, які дозволяють моделювати цифрові дані для ландшафту.

Генератор сценаріїв – програмне забезпечення, яке використовується для створення зображень ландшафту, 3D-моделей та анімацій. Ці програми часто використовують процедурне генерування для створення пейзажів. Основними елементами ландшафтів, створених декораційними генераторами, є місцевість, вода, листя та хмари.

- Генерація ландшафту:

Генерація ландшафту за допомогою математичних функції, які надають можливість створювати ландшафт нескінченого розміру(якщо упускати фактор пам’яті).

- Комбінована генерація:

Генерація ландшафту за допомогою декількох перелічених способів. Даний спосіб надає гнучкість та можливість генерувати ландшафти з більш складним та неоднозначним рельєфом.

1.3 Способи представлення ландшафтів

При створенні даних потрібно записати їх у такому виді, щоб їх можна було відобразити у графічному виді. Існує декілька способів для запису цифрових даних террейнів:

- Запис даних у вигляді карт височин:

Дані представлені у вигляді двомірного масиву. Вже задані дві координати, а третя координата представлена в вигляді значення в конкретній комірці.

Зазвичай карта висот зберігається у вигляді зображення. Такий спосіб зберігання даних дозволяє з легкістю вносити корективи та більш-менш представляти заздалегідь дані до виводу у графічному вигляді, тому що можна буде побачити як ландшафт виглядає з верху, де колір відповідає за висоту. По стандартам такі зображення мають чорно-білу палітру кольорів, але є можливість записувати дані в картину в більш різних тонах, що надасть більший діапазон значень для запису даних, але через це буде важче до виводу карти висот у вигляді ландшафту зрозуміти її вигляд.

Але такий спосіб має і недоліки, тому що для генерації ландшафту потрібно надати немалі простори пам’яті, тому що при цьому способі потрібно зберігати багато інформації для кожної точки зображення. Тому при створені відносно простих

ландшафтів, цей спосіб може програвати іншим способам зберігання інформації, щодо даних.

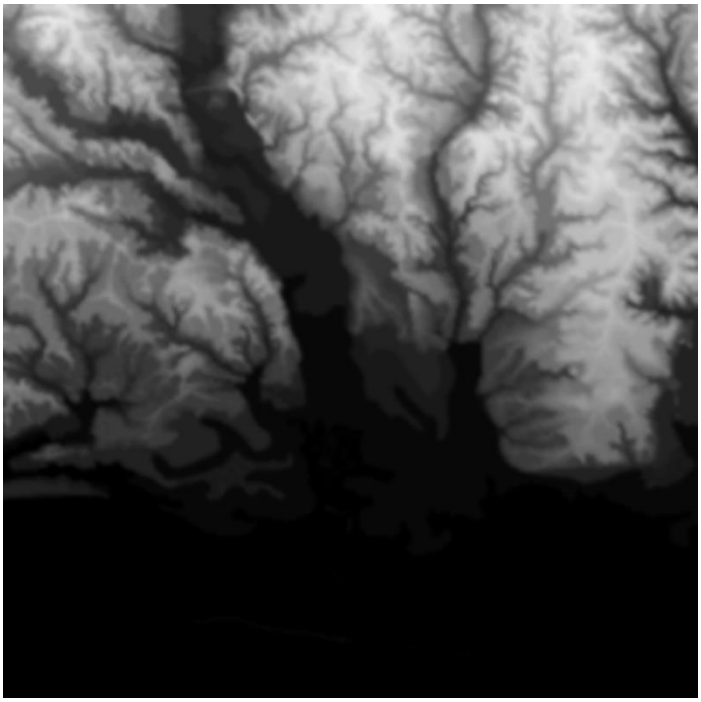

Рис. 1.1 Карта височин

- Запис даних у вигляді іррегулярної сітки:

Найчастіше такі рішення застосовуються в спеціалізованих пакетах для ігор (наприклад, редактор рівнів для Серйозного дядечка Сема) або спеціальних пакетах для роботи з тривимірною графікою (типу 3Dmax, Maya і іже з ними). І зберігаються у вигляді тривимірних моделей.

Основним плюсом такого способу, що використовується значно менше інформації для побудови ландшафту. Для цього способу необхідно зберігати тільки значення висот кожної вершини і зв'язку ці вершини з'єднують. Це надає виграш в швидкості при передачі величезних масивів інформації по AGP, в процесі візуалізації ландшафту.

Але такий спосіб має перелік недоліків. Основним недоліком є те що такий спосіб потребує оптимізації таких алгоритмів. Також одним з недоліків виникає проблема рівномірного освітлення.

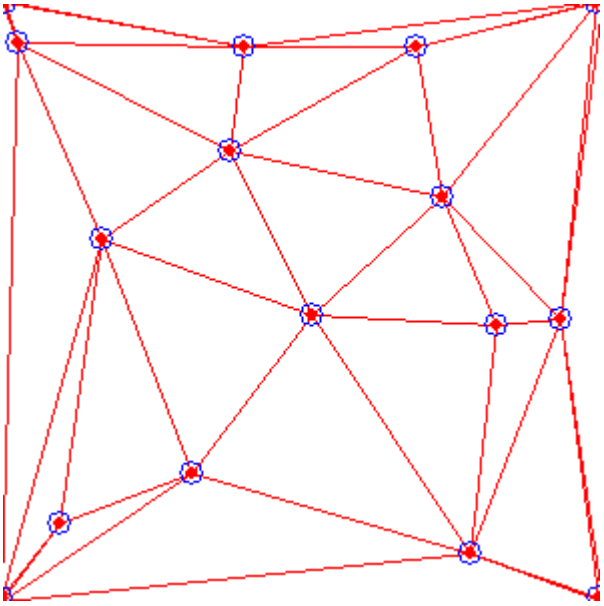

Рис. 1.2 Іррегулярна сітка

- Запис у вигляді посегментної карти висот:

В даному способі також використовуються карти висот. Тільки замість висот в ній зберігаються індекси ландшафтних сегментів. Як ці сегменти представлені, в принципі, не має значення. Вони можуть бути і регулярними, і іррегулярні.

Даний спосіб надає можливість створення великих відкритих просторів, цим способом можна зберігати не тільки дані о ландшафті, а ще о інших об’єктах. Також цей спосіб надає можливість створювати декілька варіантів одного і того ж сегменту, що надає можливість змінювати з легкістю карту та оптимізовувати.

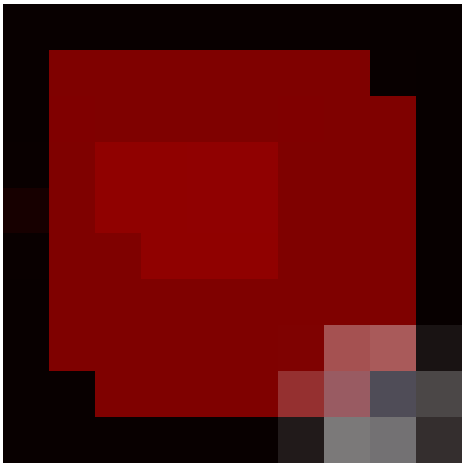

Рис. 1.3 Посегментна карта височин

Але цей спосіб має чималу кількість недоліків, один з них те що шви сегментів можуть не збігатись. При роботі з картою височин важко визначити алгоритм генерації і легше уникнути його створення, якщо є така можливість. Також дані на такій карті будуть неоднозначними, через це не можна буде попередньо сказати, як буде виглядати кінцевий результат.

1.4.Генерація ландшафтів, які мають можливість руйнуватися

При моделюванні рельєфу з кроком в один метр можна працювати з використанням методів багатокутного моделювання, але проблеми виникають, коли цей проміжок доступний на рівні дециметрів або сантиметрового рівня. В певний момент необхідно використовувати так багато полігонів, необхідних для точної моделювання даних високої точності, що традиційна багатокутна модель розвалюється через підвищення вимог до розміру файлу та його рендерингу. Таким чином створення тунелів та підземних об’єктів звичайні ландшафти не надавали можливості створювати.

Зважаючи на недоліки техніки карт висот виникла потреба у новому методі генерації ландшафту. Таким альтернативним варіантом генерації ландшафту є використання воксельної технології. Воксельні карти дозволяють подавати місцевість з набагато більшою топографічною і топологічною складністю, дозволяючи

створювати нові функції, які були неможливі при використанні полігонального моделювання.

Воксель– це елемент об'ємного зображення, що містить значення елемента растра в тривимірному просторі. Вокселі є аналогами двовимірних пікселів для тривимірного простору.

Вокселі можуть використовуватися для зберігання високо деталізованих ландшафтів, але мають відносно високі вимоги до зберігання. У порівнянні з полігональним моделюванням, таке подання може дозволити більш точне фізичне моделювання, оскільки кожен воксель може мати унікальні фізичні характеристики. Результуюча якість використання вокселів, як і пікселів, полягає в тому, що місце розташування вокселя явно не зберігається як набір координат XYZ, а скоріше визначається його відносним положенням до інших вокселів і початком координат набору даних. Це сильно відрізняється від полігонального моделювання де координатне розташування кожного кута трикутника має бути явно збережено. Єдиний розмір, як вокселів, так і пікселів на осередках дозволяє використовувати таку ефективну просторову прив'язку.

Основними плюсами використання вокселів є:

- Безперервні 3D-дані. Вокселі – це майже єдиний ефективний спосіб зберігання безперервних даних про приховані характеристиках місцевості;
- Легка модифікація. Стиснені дані вокселя можуть бути легко змінені;
- Розширені можливості рельєфу. Можна створювати печери і тунелі;
- Цікава генерація ландшафту. Наприклад, у грі «Minecraft» це відбувається шляхом накладення шумових функцій і градієнтів з зумовленими характеристиками місцевості (дерева, підземелля).

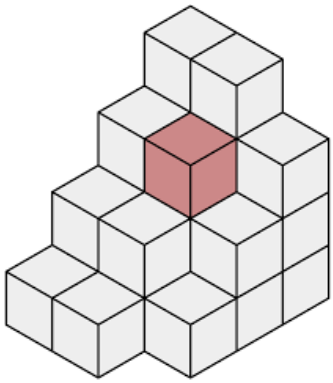

Рис. 1.4 Воксельний спосіб представлення даних у вигляді ландшафту

1.5 Опис структур воксельних даних на основі дерева

Дуже часто зустрічається архітектура, коли у воксельному октодереві суб'єкт поділяється на 8 октанів, де кожний з октанів отримує свій порядкови номер для роботи з кожним з них.

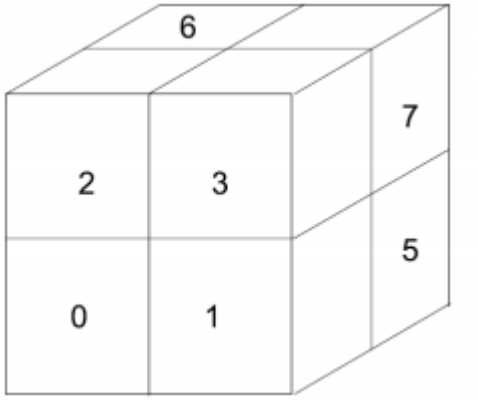

Рис. 1.5 Приклад нумерації октанів октодерева

Кожен суб'єкт може бути поділений, відносно свого розміру, який його обмежує, тому немає необхідності зберігати будь-які дані, що стосуються під-розділа; розміри кожного вузла можуть бути отримані безпосередньо з його положення в дереві. Велика кількість супервузлів може бути створена просто для подання однорідних під-виборів.

Елементи октодерева можуть поділятися відносно центру. Далі на прикладі наведено, як зберігається точка на площині, та як виглядає її запис у вигляді дерева.

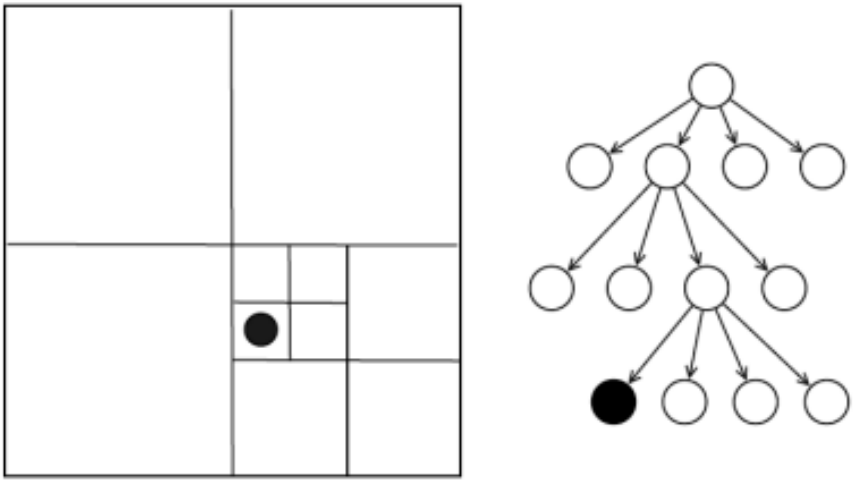

Рис. 1.6 Двовимірне квадро-представлення. Тривимірне октодерево аналогічне

Для Point-region octree область, представлена вузлом в октеті, підрозділяється через її геометричний центр незалежно від базового набору даних. Таким чином, однорідні характеристики ландшафту не завжди належним чином використовуються.

Наприклад, якщо тільки нижня третина допоміжного обсягу неоднорідна, може бути доцільним переміщати положення розщеплення вниз. Point-region octree намагається поліпшити октейн, маючи потенційно змінюване положення розщеплення. Об’єми, що обмежують, для кожного з восьми дітей вузла можуть бути однозначно визначені з цієї позиції поділу. Це означає, що необхідно зберегти тільки три значення для координат розщеплення, а не шість значень, необхідних для зберігання розмірів об’єма, що обмежує. Октанти дитини вузла нумеруються так само, як і в розрідженому воксельному октодереві. Вузли Point-region octree не завжди поділяються по центру, що потенційно дозволяє використовувати вузькі плити з одним відносно великим розміром і один, відносно невеликий, розмір. Якщо порівняти воксильне окто дерево – Point-region octree для представлення однієї точки, то структура квадрантів Point-region octree, створена під час поділу, точка зберігається в кольоровому листковому вузлі, прикладу то можна побачити, що у Point-region octree потрібна менша кількість під-розподілів, так як під-вибори можуть

бути розташовані безпосередньо поруч з точкою даних, тоді як кожен листової вузол має однакому мінімальну площу.

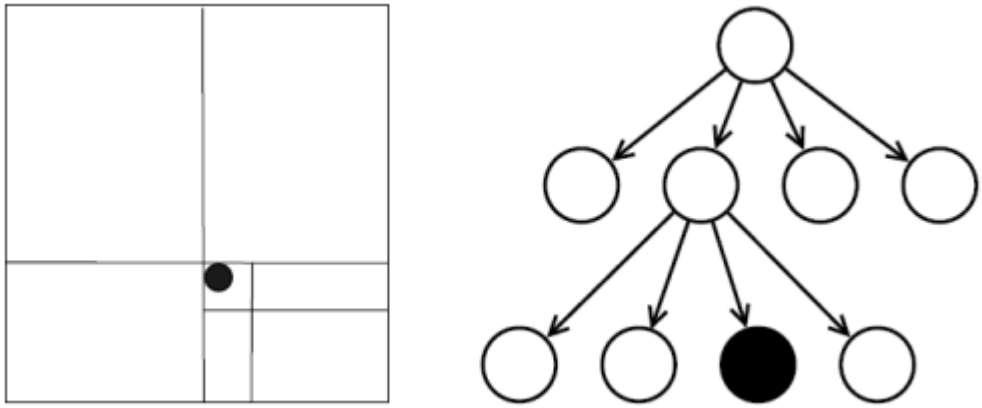

Рис. 1.7 Point-region octree квадро-представлення. Point-region octree аналогічне

Також є можливість зберігти дані у вигляді Kd-tree. Де Kd-tree виступає двійковим деревом, яке ділить обмежуючий об’єм на два підлеглих по осі згладжування. Кожен внутрішній вузол розбивається по певній осі в заданому положенні. Вісь може бути обрана як функція висоти вузла в дереві таким чином, що вісь поділу не повинна бути явно збережена для кожного вузла дерева.

Основним плюсом kd-дерева, те що у kd-дерева тільки дві дитини, потрібно менше пам'яті для кожного вузла, ніж для Sparse voxel octree або Point-region

Спочатку kd-дерево розділяється уздовж осі x, далі відбувається поділ уздовж осі y, осі x і нарешті осі y. Цей поділ триває до тих пір, поки точка даних не буде збережена як листовий вузол.

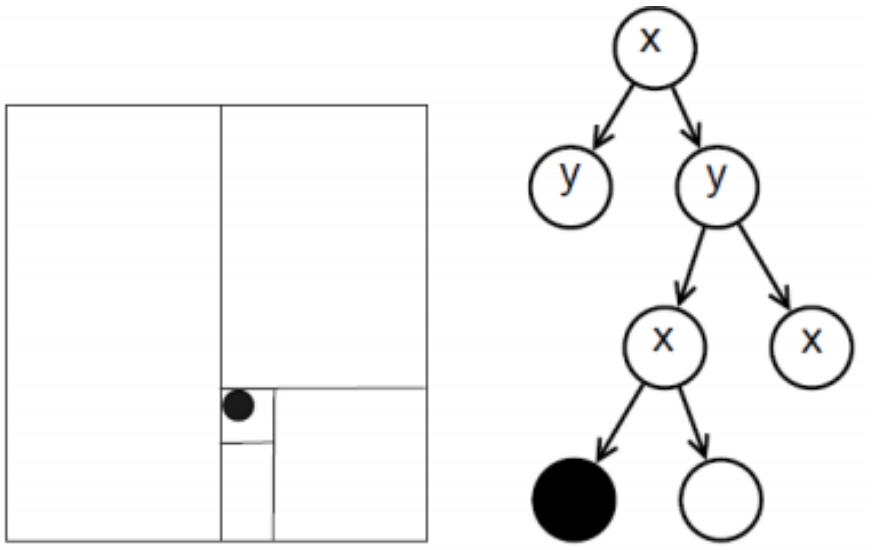

Рис. 1.8 kd-tree octree квадро-представлення. kd-tree octree аналогічне

2. АЛГОРИТМИ ГЕНЕРАЦІЇ ТА МОДЕЛЮВАННЯ ЛАНДШАФТІВ

2.1 Основи алгоритмів для генерації ландшафтів

Найбільш легким способом заповнення карти висот є генерація псевдовипадкових чисел. В їх основі як правило лежать рекурентні формули, вони генерують цілі числа від 0 до деякого максимального m.

Генерація ландшафтів передбачає собою певну кількість проблем, які повинні бути вирішені алгоритмом генерації. Основними з них виступають такі проблеми:

- неплавний перехід між значеннями даних;
- генерація даних таким чином, щоб ландшафт міг змінити свої розміри та зостався повноцінно заповненим;
- можливість генератору псевдовипадкових чисел мати алгоритм, який може бути налаштований, на те щоб мати можливість змінювати вихідні дані за допомогою вхідних параметрів;
- можливість розширення та модифікації алгоритму, де надається можливість доповнити реалізацію, щоб вносити структурні зміни в алгоритм.
- проблема реалістичності ландшафту стоїть на одному з перших місць, оскільки не кожний алгоритм передбачає можливість створити реалістичний ландшафт.

Але дана проблема є незначною оскільки на цей час було написано чимало алгоритмів підрахування ерозії ландшафту, що значить, що після виконання основного алгоритму можна підвергнути ландшафт алгоритму ерозії, що надасть реалістичності.

Список проблем вражає, але сучасна математика має алгоритми, які можуть задовольнити усі проблеми у цьому списку.

На даний момент було розроблено та адаптовано чимало алгоритмів для створення даних, на основі яких буде розроблений ландшафт. Чимало з цих алгоритмів використовуються для розробки сучасних продуктів, хоча не всі з них задовольняють список.

2.2 Білий шум, як алгоритм для генерації ландшафтів

Шуму - це серія випадкових чисел, зазвичай розташованих на лінії або в сітці. При перемиканні на канал без сигналу на старих телевізорах ми бачили на екрані випадкові чорні і білі крапки. Це шум (їх відкритого космосу!). Під час прийому радіоканал без станції ми теж чуємо шум (не впевнений, чи з'являється він з космосу, або звідкись ще).

Графічний шум - це випадкові числа, розташовані в сітці (2D). Можна створювати шум в 3D, 4D, і так далі.

Хоча в більшості випадків ми намагаємося позбутися від шуму, у проблемі створення ландшафтів – це достатньо гідні дані для використання . Хоча реальні системи і виглядають гучними, але в їх основі зазвичай закладена структура.

Основним же плюсом шуму є те що алгоритм може створювати дані для ландшафтів будь-яких розмірів. Також плюсом шуму є простота створення алгоритму генерації шуму та модифікації даного алгоритму.

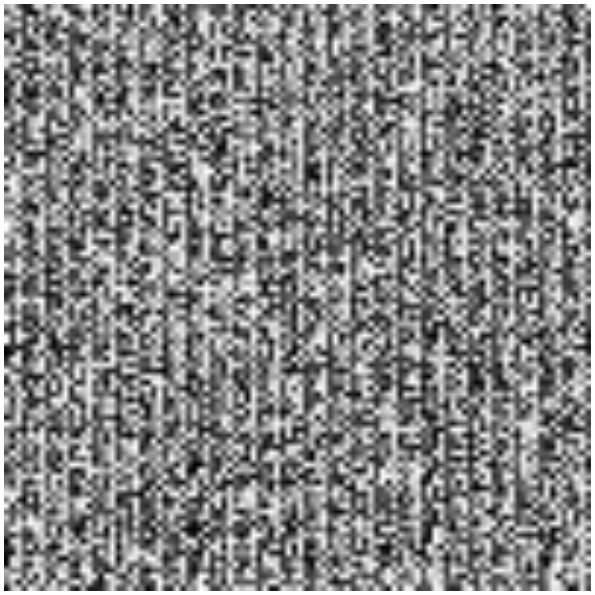

Рис. 2.1 Приклад графічного шуму.

2.3 Шум Перлина, як алгоритм для генерації ландшафтів

Шум Перлина — це примітив процедурних текстур, що належить до градієнтних шумів, які створенні певним математичним виразом. Художники використовують його, щоб зробити комп'ютерну графіку реалістичнішою. Результат алгоритму є псевдовипадковим, але всі візуальні деталі мають однаково пропорційні розміри.

Шум Перлина створений Кеном Перлином у Mathematical Applications Group для Диснеївського фільму 1982 року Трон, для покращення зображення та створення більш реалістичного зображення . У 1997 році Перлин отримав Академічну Нагороду за Технічні Досягнення від Академії кінематографічних мистецтв і наук за його внесок у CGI. Кен ніколи не подавав заявки на патенти на свій алгоритм; це просто народилося з його розчарування сучасним методам генерації комп’ютерних зображень. Однак пізніше він отримав патент на деякі випадки використання, такі як симплексний шум, що спрощує його оригінальний, більш складний алгоритм та надає дані в яких менше шансів появи непередбачених обставин(артефактів).

Шум Перлина зазвичай реалізується як дво-, три-, або чотиривимірна функція, але може бути визначений довільною кількістю вимірів. Реалізація складається з трьох кроків: визначення сітки, обчислення скалярного добутку градієнтних векторів, та інтерполяція між цими значеннями.

Зазвичай виконання алгоритму Шуму Перлина поділяють на 3 частини:

- Визначення сітки;

Визначається вимір сітки координат, за якою буде проводитись створення даних. Після визначення з виміром кожній координаті встановлюється певне значення в залежності від виміру сітки.

- Скалярний добуток;

Визначення, в яку комірку сітки потрапляє окрема точка. Для кожного вузла сітки визначаємо вектор відстані між точкою і координатами вузла. Після чого обчислюємо скалярні добутки векторів відстані та градієнтних векторів кожного вузла комірки.

- Інтерполяція.

Проводимо інтерполювання значень. Інтерполяція виконується з використанням функції, що має нульову першу похідну, можливо, другу похідну також, на обох кінцевих точках.

В основному на першому етапі генерації Шуму Перлина використовується звичайний шум, для того щоб задати випадкові значення в таблицю. Тобто алгоритм Шуму є важливим, але не необхідним для використання шумів Перлина.

Даний алгоритм генерації випадкових чисел є одним із еталонних на наш час для генерації ландшафтів, оскільки він вирішує майже всі проблеми, які виникають при створенні та роботі з даними для генерації ландшафтів. Єдиними недоліком Шуму

Перлина є його складність та кількістю операцій для виконання добутку (відносно інших алгоритмів), але цим фактором можна знехтувати, оскільки у наші дні цей алгоритм виконується достатньо швидко та тому що даний алгоритм зазвичай використовується невелику кількість разів. Також э недолік можливості з’явлення артефактів, але шанс їх появи незначний.

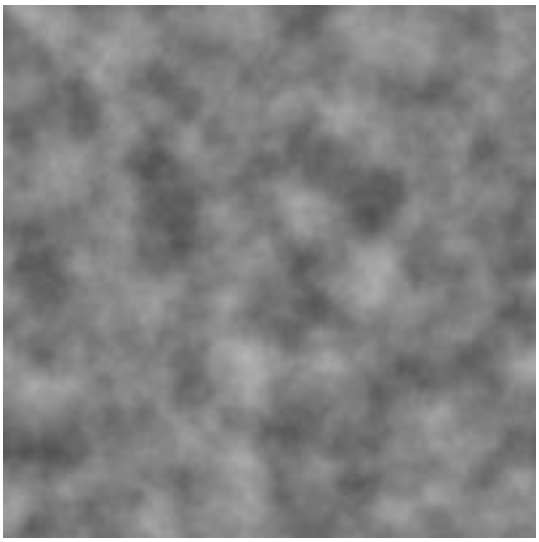

Рис. 2.2 Приклад шуму Перлина.

2.4 Інші алгоритми для генерації ландшафтів

2.4.1 Сімплекс Шум

Симплекс-шум — метод побудови *n*-вимірної функції шуму, заснований на основі Шуму Перлина. але з меншою кількістю артефактів і кращою продуктивністю. Кен Перлин розробив алгоритм у 2001 році щоб усунути обмеження класичної функції шуму, особливо за великої кількості вимірів.

Основною відмінністю Симплекс шуму від шуму Перлина є те, що шум Перлина інтерполює значення градієнтів вузлових точок сітки, коли Симплекс шум поділяє простір на симплекси та проводить інтерполяцію вже на них.

Основними перевагами Симплекс шуму над її пращуром є:

- Симплекс-шум має меншу обчислювальну складність і вимагає менше операцій множення;

- За більшої кількості вимірів (4D, 5D) симплекс-шум є продуктивнішим;
- Симплекс-шум не має помітних артефактів;
- Симплекс-шум має добре визначений неперервний градієнт, який обчислюється доволі швидко;
- Симплекс-шум легко реалізувати на машинному рівні.

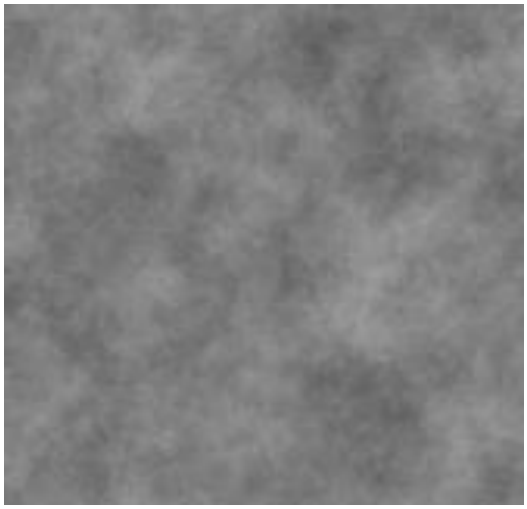

Рис. 2.2 Приклад Симплекс шуму

2.4.2 Діаграма Вороного

Діаграма Вороного — це особливий вид розбиття метричного простору, що визначається відстанями до заданої дискретної множини ізольованих точок цього простору. Діаграма виступає результатом даних та переводиться у інший вид, щоб вивести ландшафт.

Діаграма Вороного має декілька алгоритмів побудови:

- Простий алгоритм;

Між кожними сусідніми точками проводиться лінія, на цій лінії створюється рівновіддалений перпендикуляр, який розділяє плоскість. Таким чином розділяються усі сусідні точки між собою.

Даний спосіб є простим з графічної точки зору, але є достатньо складним з математичної точки зору.

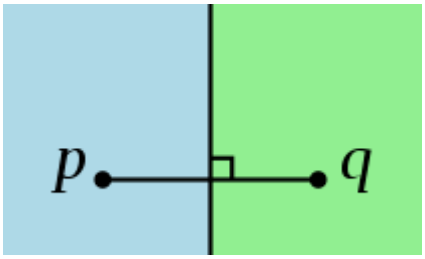

Рис. 2.4 Приклад розділення площин між точками для Діаграми Вороного, простим алгоритмом.

- Алгоритм Форчуна ;

Даний алгоритм заснований на прямій, яка є допоміжною, при кроці якого, від точки до точки, перестроюється діаграма відносно доповненої точки.

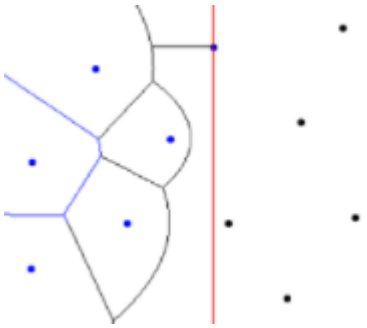

Рис. 2.4 Приклад побудови Діаграми Вороного за допомогою Алгоритму Форчуна

Даний спосіб одним з найбільш оптимальних способів для створення діаграми Вороного, оскільки його нескладно реалізувати з математичної та логічної точок зору.

- Рекурсійний алгоритм .

Основна задумка побудови діаграми на основі рекурсивного алгоритму полягає в розбитті точок на підмножина та створення діаграм Вороного для цих підмножин, після створення діаграм підмножин підмножини рекурсійно об’єднуються між собою. Даний спосіб здається більш оптимальним ніж простий спосіб, але програє алгоритму Форчуна у своїй складності.

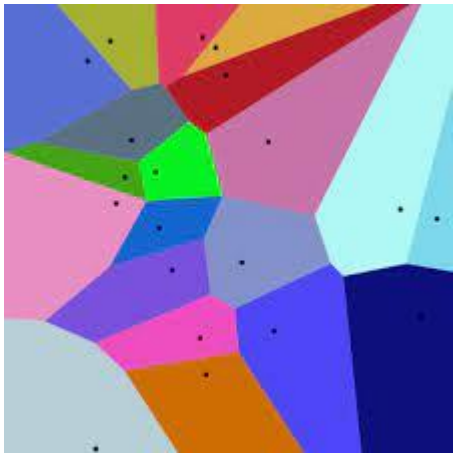

Рис. 2.5 Приклад Діаграми Вороного

2.4.3 Алгоритм Diamond-Square

Diamond-Square - це метод генерації карт висоти для комп'ютерної графіки. Це трохи кращий алгоритм, ніж тривимірна реалізація алгоритму переміщення середньої точки, який створює двовимірні пейзажі. Він також відомий як випадковий фрактал зміщення середньої точки, фрактал хмари або фрактал плазми через ефект плазми, що виникає при застосуванні.

Алгоритм генерації складається з 2 частин, і виконує їх поки не проведе по одному з цих методів з усіма точками:

- square step;

Для кожного квадрата масиву встановіть середню точку цього квадрата як середнє значення чотирьох кутових точок плюс випадкове значення.

- diamond step.

Для кожного ромбу в масиві встановіть середню точку цього ромбу як середнє значення чотирьох кутових точок плюс випадкове значення.

Після виконання обох дій процедуру потрібно повторяти, вся процедура не пройде по кожній з точок, які є на ландшафті.

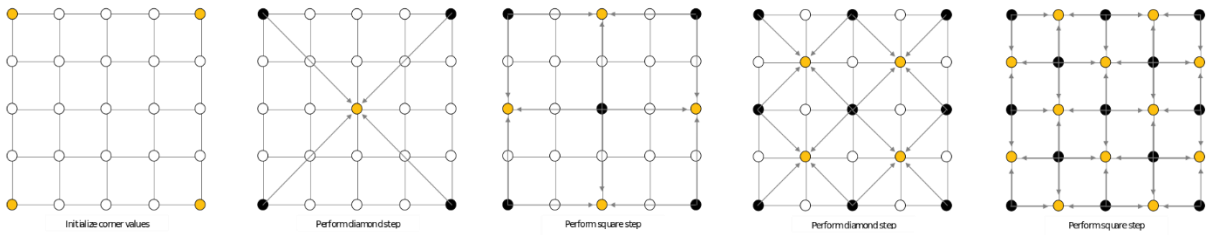

Рис. 2.6 Опис алгоритму Diamond-Square

2.4.3 На основі Інших Фракталів

Не дивлячись на те що є чимало алгоритмів для генерації ландшафтів, фрактали знайшли і у цьому місці використання. На основі багатьох фракталів можна отримати набір випадкових чисел для генерації ландшафтів. Через те що деякі фрактали повертають тільки певні числа, їх можна інтерполювати, щоб отримати більш реалістичний графік.

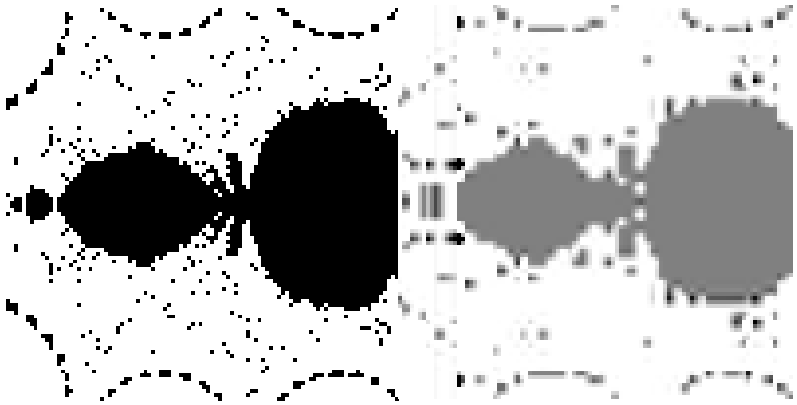

Рис. 2.7 Інтерпольований та Не інтерпольований фрактали

2.4.4 Сучасні інструменти для моделювання та генерації ландшафтів

2.4.4.1 Програми для моделювання ландшафтів

Є безліч програмних продуктів для моделювання ландшафтів та набору чисел. В основному для моделювання є 2 типи інструментів:

- Інструменти моделювання даних у двовимірному просторі;

До таких інструментів можна віднести багато програм, від звичайного тестового редактору до графічного редактору. В основному ці інструменти можуть бути використані для моделювання значень ландшафту. Даний тип інструментів працює на пряму з даними і щоб зберигти дані не потрібно експортувати їх чи проводи інші махінації.

Зазвичай, через ці інструменти важко передбачити результат, оскільки він відображується у значенні, але можливо.

- Інструменти моделювання даних у тривимірному просторі.

Інструменти моделювання даних у тривимірному просторі, більш рідко зустрічаються ніж двовимірні, оскільки вони є вузько спеціалізованими, або додатком до програми. Наприклад Unity може виступати, як даний інструмент, адже воно надає можливість створити ландшафт та змінювати його за допомогою пензля та вбудованими функціями.

Обидва типу інструменту мають місце бути у такому вигляді як є, адже у кожного є свої плюси та недоліки і по більшій частині дані інструменти не завжди розроблені під задачу працювати з ландшафтами.

2.4.4.2 Програми для генерації ландшафтів

Є чимало програм для генерації ландшафтів. Зазвичай дані програми являють собою перелік математичних функцій, як можна перенести на множину даних, які представляють собою террейн. Більшість цих функцій взята із звичайної математики та перероблена під ці задачі, але деякі з представлених там функцій спеціально розроблялися під такі задачі.

Для ознайомлення з такими програмами найкраще всього взяти ПЗ World Machine, даний інструментарій є одним з найпопулярніших для генерації ландшафтів. World Machine надає можливість створити схему на основі блоків, які представляють собою деякі функції, за допомогою цих блоків можна виконувати математичні

функції, генерувати псевдо випадкові числа для виводу ландшафту, надавати текстури террейну та багато чого іншого.

Через подібні інструменти можна подивитись, як террейн буде виглядати при певному вірні води, ящо потрібно.

World Machine надає не тільки можливість згенерувати террейн, а ще надає можливість подивитись у реальному часі, як повпливає певна математична формула коли буде використана на ландшафт.

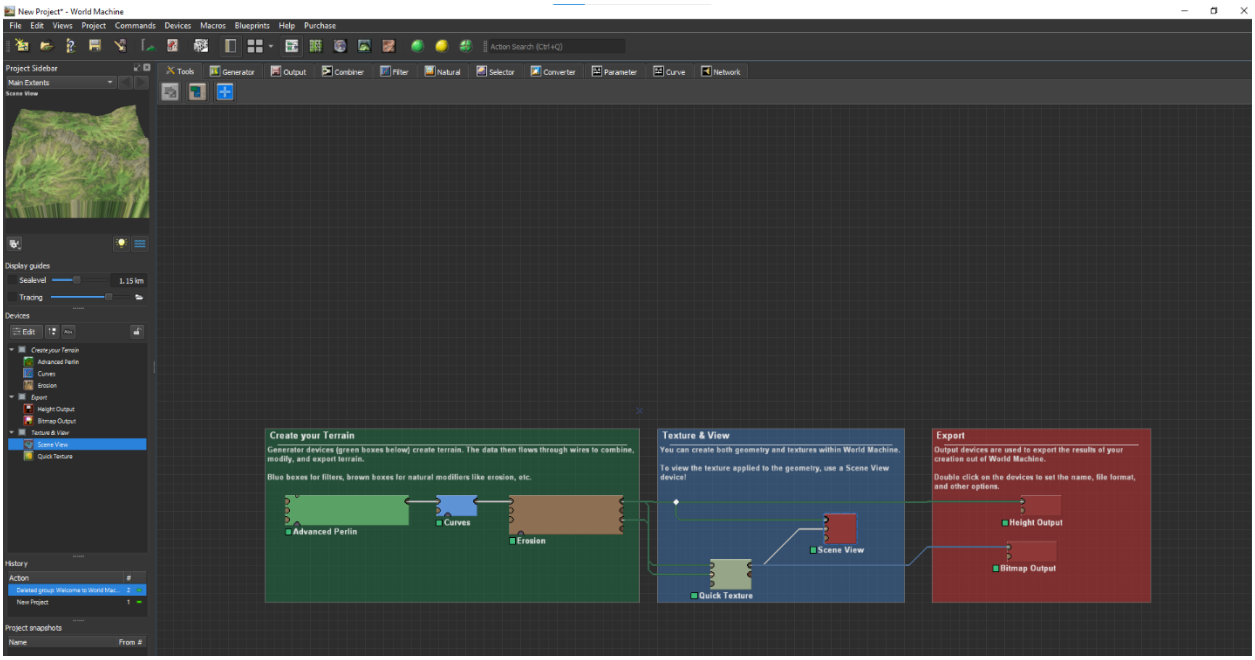

Рис. 2.8 World Machine та її інструментарій

Отож інструментарії подібні World Machine, мають велику цінність та популярність через можливість полегшити процес генерації даних для створення ландшафтів на основі математичних формул.

3. ГЕНЕРАЦІЯ ТА ВИВЕДЕННЯ ЛАНДШАФТІВ НА ПРАКТИЦІ

3.1 Генерація випадкових даних на практиці

Генерація шумів буде проходити за допомогою мови програмування python, яка дозволить швидко та з легкістю виконати усі математичні задачі. Щоб використовувати математичні функції, які знадобляться для реалізації програми потрібно підключити бібліотеку numpy. Також потрібна можливість записувати дані у правильному форматі, тому потрібно підключити бібліотеку PIL та використовувати об’єкт з неї, який має назву Image, за допомогою даного об’єкту можна з легкістю зберігати дані в файл у вигляді картинки.

3.1.1 Генерація шуму

Як було сказано вище шум є одним із примітивніших алгоритмів, оскільки має немало недоліків, а розробити такий алгоритм буде достатньо легко. Для створення шуму буде використана функція noise2d з бібліотеки OpenSimplex. Також для використання математичних функцій приєднаємо бібліотеку numpy.

Отож поділимо на 2 частини задачу:

- Генерації даних
- Запис даних

Для «Генерації даних» потрібно описати функцію generateNoise, яка буде виконувати генерацію даних за допомогою даної функції будуть створені дані та повернуті, для подальшого запису.

У пункті «Запис даних» записано дані у вигляді картинки за допомогою вищесказаною бібліотеки Image. Для цього записуємо в буфер дані та викликаємо функцію Save.

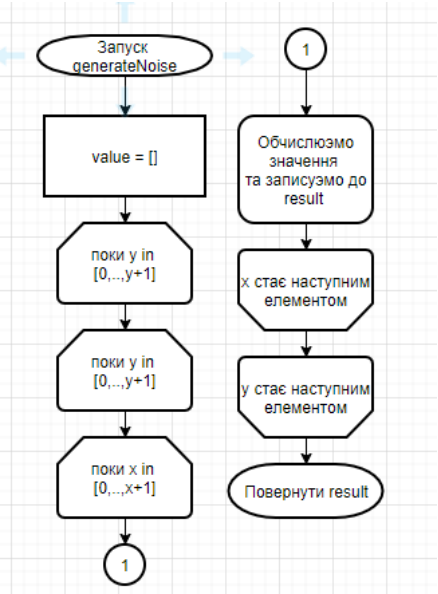

Рис. 3.1 Блок схема функції generateNoise

3.1.2 Генерація шуму Перлина

Шум Перлина дуже потужний алгоритм, в плані генерації террейну, але є дуже складним алгоритмом генерації. Для створення алгоритму буде використовуватись. При розробці алгоритму потрібно підключити до файлу функцію product бібліотеки intertools.

Для створення бібліотеки буде створена фабрика, яка буде генерувати дані для кожної заданої точки. Фабрика буде поділена на декілька методів:

- get_plain_noise

Функція, яка буде підраховувати значення в певній точці, де використовується функція _generate_gradient.

- _generate_gradient

Функція, яка буде проводити векторні градієнти між вершинами.

- _call_

Функція, котра отримує координати точки та використовує, щоб отримати дані з функції get_plain_noise та проводить редагування отриманого значення за допомогою октав.

Також для запису буде використаний спосіб, який використовувався для запису звичайного Шуму.

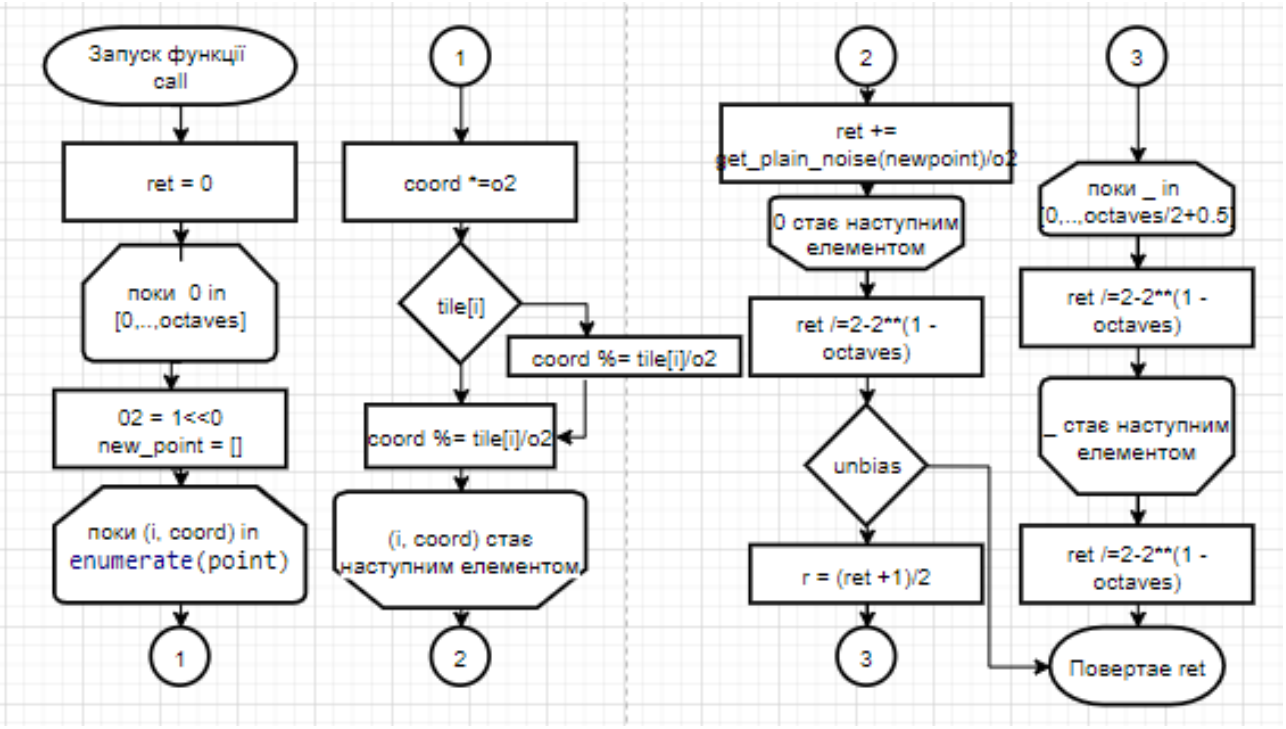

Рис. 3.2 Блок схема функції _call_

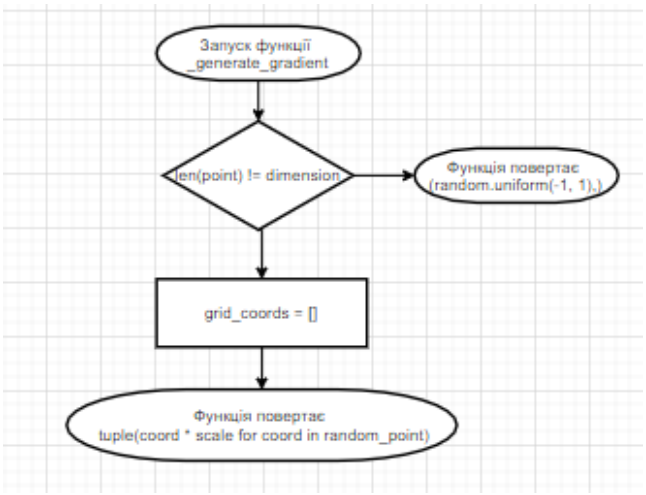

Рис. 3.3 Блок схема функції _ generate_gradient

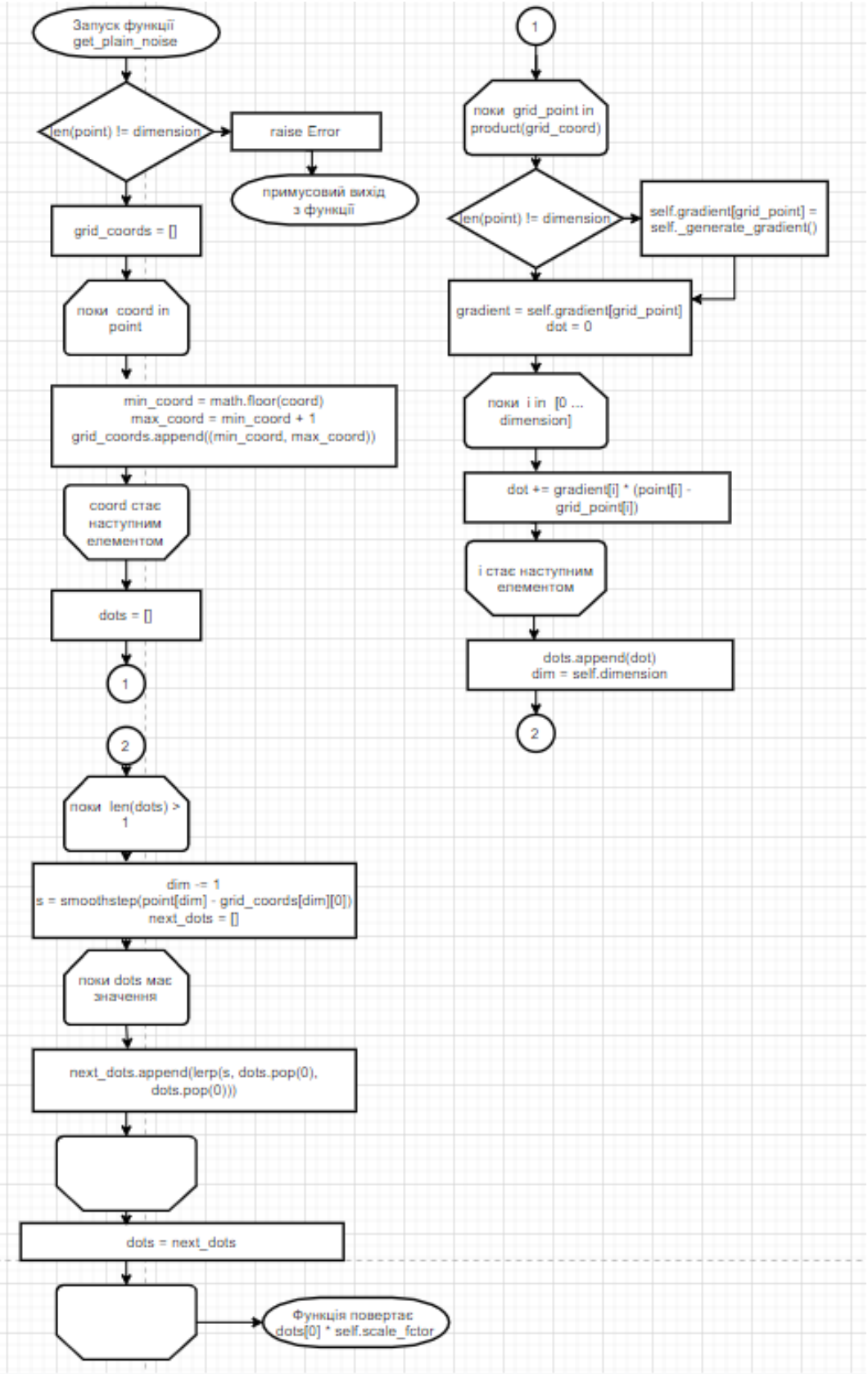

Рис. 3.4 Блок схема функції get_plain_noise

При використанні функцій, описаних вище в вигляді блок схеми, буде отриманий наступний результат:

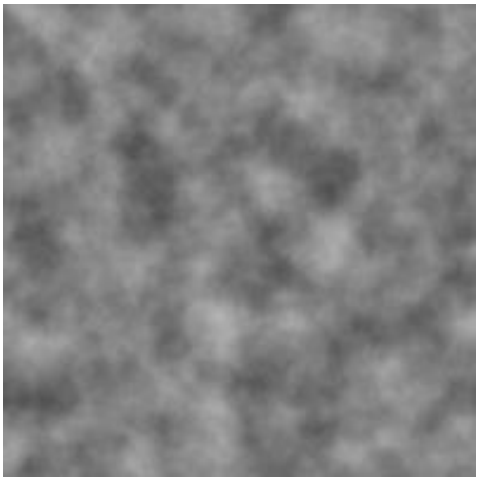

Рис. 3.5 Дані записані отримані
при використанні шуму Перлина

3.1.3 Генерація фракталу Мандельброта

На основі багатьох фракталів можна створити ландшафт, фрактал Мандельброта не є виключенням. Даний фрактал буде генеруватися за допомогою класу генератору, який може згенерувати набір чисел як і інтерпольований, так і не інтерпольований. Спосіб збереження даних буде майже таким же, як у попередніх випадках.

Для того щоб згладити фрактал буде використовуватися функція interp2d з бібліотеки scipy.interpolate. При інтерполяції потрібно обрати спосіб за яким буде проводитися інтерполяція, для цього методу буде обрана лінійна інтерполяція.

Отож буде використано три метода:

- ZFunc – функція, яка допомагає отримати дані фракталу;
- GenerateImage – функція яка знаходить дані фракталу та повертає їх у 2-вимірному масиві;
- GenerateImageSmoothedImage – функція яка знаходить дані фракталу, інтерполює дані та повертає їх у 2-вимірному масиві.

При використанні описаних функцій буде отримано два типи фракталів:

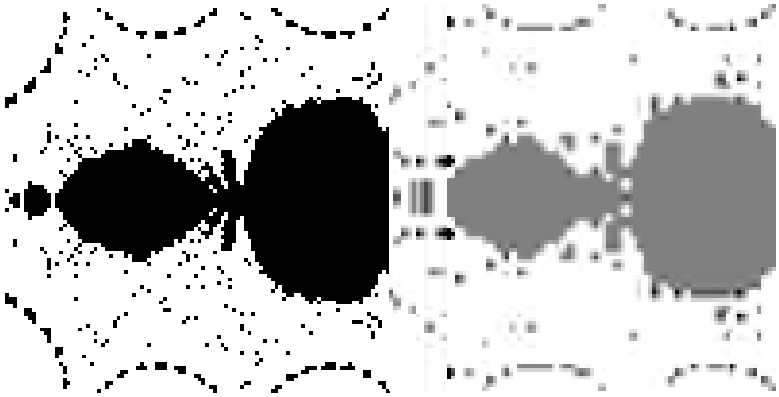

Рис. 3.6 Не інтерпольовані та інтерпольовані фрактали Мандельброта

3.1.3 Генерація фракталу Жулиа

Фрактал Жулиа має можливість також бути відображеним у вигляді ландшафту. Даний фрактал буде розроблений на основі вже створеного фракталу Мандельброта, для цього потрібно змінити реалізацію методів попереднього фракталу, а архітектурних змін не передбачається.

При виконанні змін реалізації методів генерації фракталу Мандельбрта на Жулиа будуть отримані наступні графіки:

Рис. 3.8 Не інтерпольовані та інтерпольовані фрактали Жулиа

3.2 Вивід даних у вигляді террейну

Для виводу террейнів буде використовуватися програмне забезпечення за назвою Unity. Unity – це інструментарій для розробки комп’ютерних ігор, створення

симуляцій та створення CGI медіа. У Unity встроена велика кількість інструментів для роботи з терренами, інструментарій цього продукту надає можливість не тільки виводити ландшафт, а ще редагувати його, моделювати та навіть генерувати.

Основною мовою розробки скриптів для продукту Unity є мова програмування C#, яку буде використано для роботи з ландшафтами.

3.1.4 Вивід звичайного тривимірного ландшафту

Сучасні ігрові двигуни передбачають виведення ландшафту у вигляді плоскості, за допомогою вбудованих функцій. Unity не став виключенням та надає можливість вивести звичайних тривимірний ландшафт. Для того щоб вивести ландшафт потрібно створити мапу височин та перевести її в формат RAW та імпортувати переведену карту височин.

Після імпорту карти височин в проект потрібно створити ландшафт на обраній сцені та імпортувати RAW файл до створеного ландшафту.

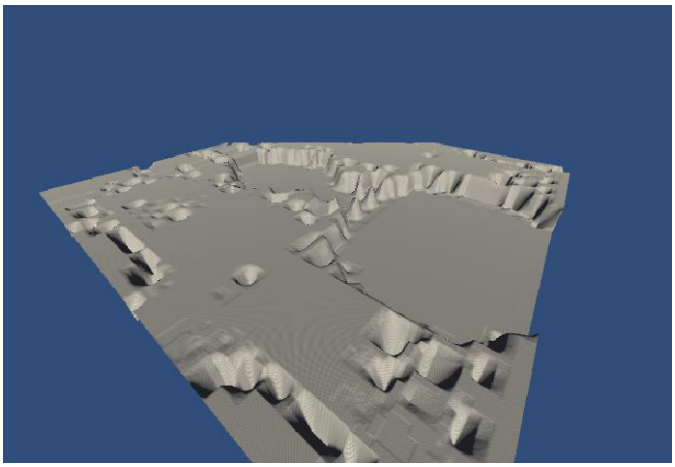

Рис. 3.8 Виведення фракталу Мандельброта

3.1.5 Вивід тривимірного воксельного ландшафту

У теоритичній частині було пояснено, що таке воксель та як з ним працювати. Для того щоб вивести вокселі за допомогою інструментів Unity потрібно написати скрипт, який буде отримувати карту височин та будувати на основі неї воксельні

масиви. Для розробки створюємо файл скрипту в проекті, який буде загружати масив та пустий об’єкт створити на сцені. Прив’язати скрипт виведення воксельного ландшафту до створеного пустого об’єкту та передати в нього наступні значення:

- Масив елементів;
- Карту височин;
- Висоту;
- Розмір.

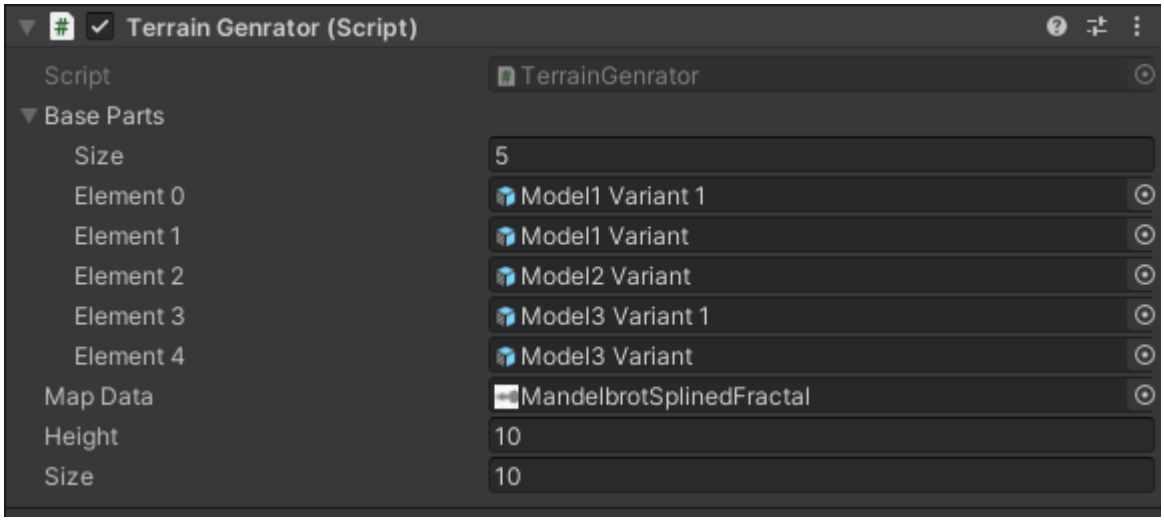

Рис. 3.9 Введення даних для генерації воксельного террейну

Код буде працювати дуже простим чином, він буде ділити на частини карту височин та зберігати у двовимірний масив середнє значення помножене на висоту отриманих даних відносно даної частини. Далі код буде створювати до кожної координати певну кількість вокселів, яка вказана в кожній координаті масиву. Останнім вокселем можна вказати певний об’єкт, щоб поверхня мала шар однакових вокселів(як земля, яка покриває інші породи).

Після запуску програми пройде ініціалізація, та буде створений ландшафт на основі переданих об’єктів. Перший переданий об’єкт буде використаний для генерації випадкових вокселів, та вокселів котрі будуть виступати верхнім пластом ландшафту.

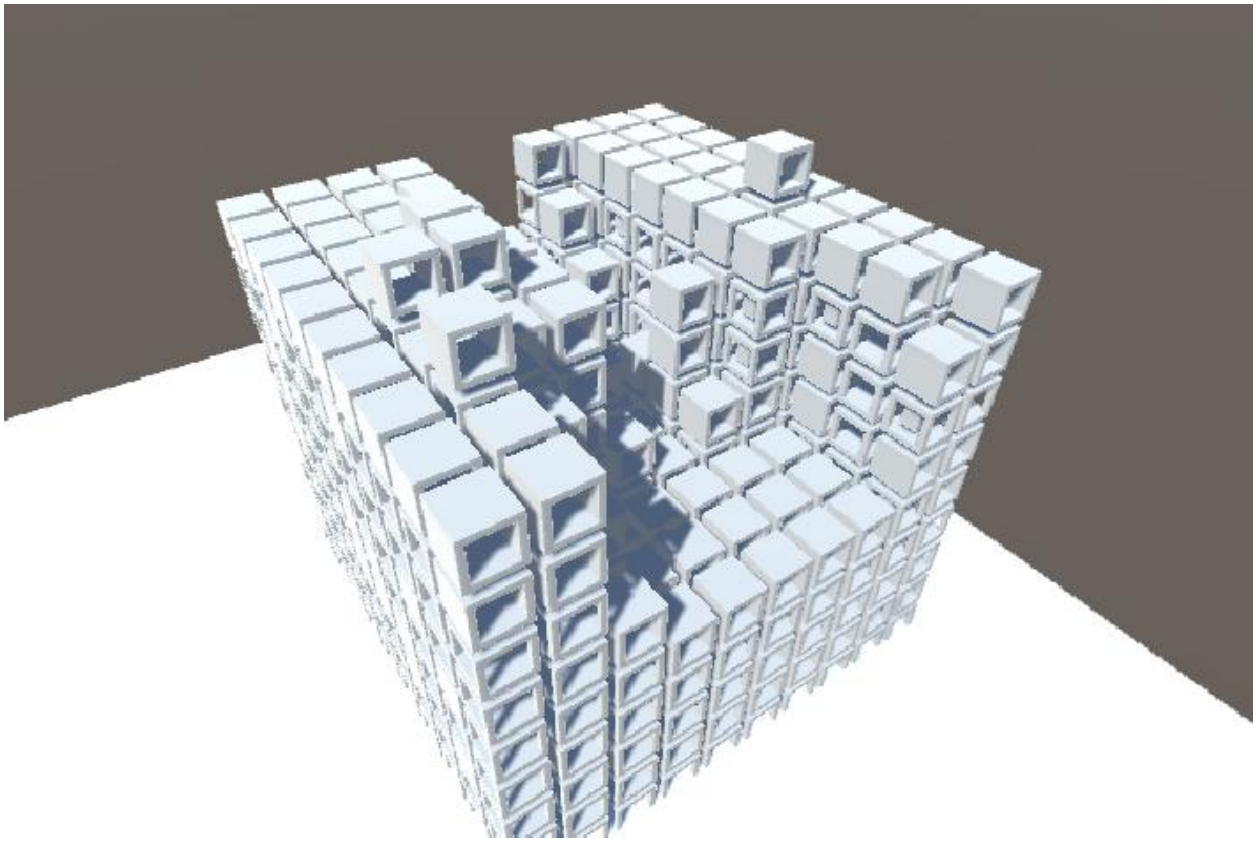

Рис. 3.10 Воксельний ландшафт виведений
на основі даних фракталу мандельброта

3.2.1 Вивід тривимірного на основі сегментної карти височин

Як було сказано вище сегментна карта височин може зберігати інформацію не тільки про рельєф ландшафту, а також багато іншої інформації. Але для того щоб створити ландшафт на основі сегментної карти височин потрібно дуже обережно її створювати. Я вирішив створити сегментну карту в Paint. У цій карті червоний канал зберігає у собі височину, зелений тип покриття ландшафту, синій – об'єкт, який стоїть на обраній частині, альфа канал не був потрібний, але його також можна використовувати, якщо потрібно буде зберегти ще якісь дані.

Після побудови сегментованої карти висот, її потрібно загрузити до проекту, після цього карту височин потрібно передати в метод генерації карт, який генерує ландшафт на основі сегментних карт.

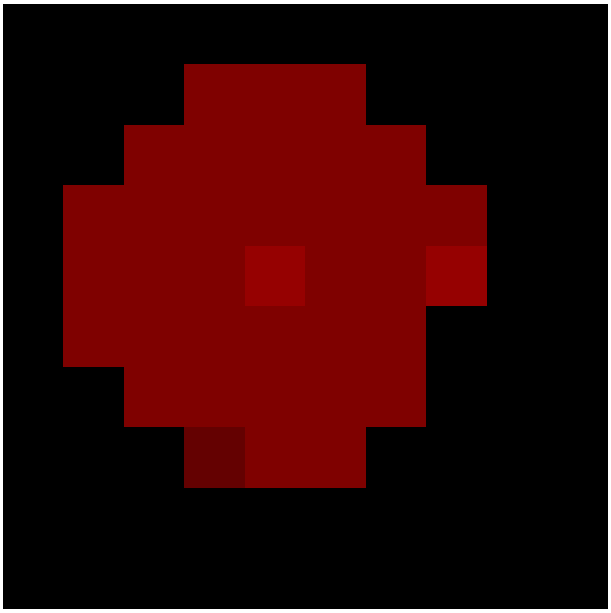

Рис. 3.11 Сегментна карта височин

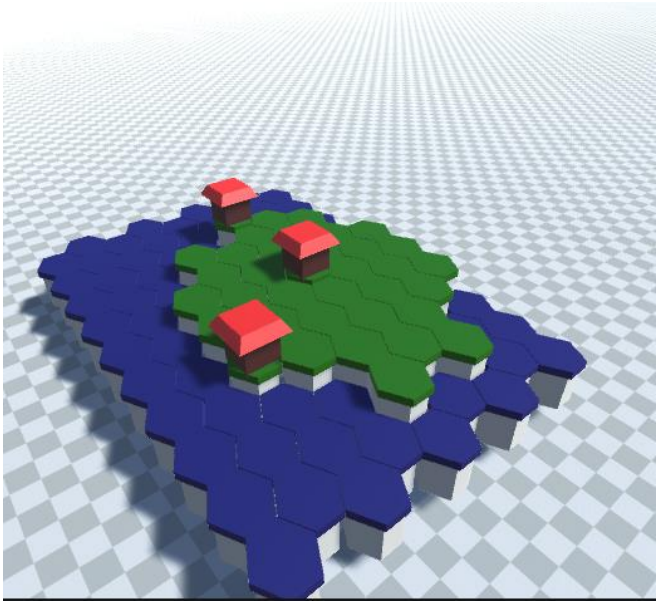

Рис. 3.12 Вивединий результат на основі сегментної карти височин

ВИСНОВОК

В ході курсової роботи було ознайомлено з террейнами та основними технологіями роботи з ними. Були розглянуті способи зберігання даних терейнів при рботі з даними способами було продемонстровано, як зберігати дані та як їх зчитувати з формату, у якому вони можуть бути збережені. Також були розглянуті способи виведення террейнів, та їх зовнішній вигляд у якості виведеного ландшафту для користувача. Наповнення террейну(дерева, машини) також було застосовано, при використанні сегментної карти височин було змодельовано на основі даних ландшафт, на якому автоматично розмістилися об’єкти.

При виконанні роботи було розглянуті алгоритми процедурної ландшафту оскільки генерація є невід’ємним пунктом для створення ландшафту на основі випадкових даних. Були розглянуті фрактальні алгоритми для отримання переліку псевдовипадкові дані, котрі можна представити у вигляді ландшафту. Також був розглянутий алгоритм шуму та алгоритм шуму Перлина.

На практиці було повноцінно пройдено шлях генерації ландшафту від створення даних до виведення готового терену користувачу. Тестування розробленого ПЗ показало, що швидкодія модулів відповідає середній швидкодії існуючих алгоритмів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Карты из шестиугольников в Unity: сохранение и загрузка, текстуры, расстояния [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/425919
2. Procedural generation: a primer for game devs [Електронний ресурс] – Режим доступу до ресурсу: https://www.gamasutra.com/blogs/ScottBeca/20170223/292255/Procedural_generation_a_primer_for_game_devs.php.
3. An Algorithm for Automated Fractal Terrain Deformation [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/244337304_An_Algorithm_for_Automated_Fractal_Terrain_Deformation.
4. Grome 3.1: генератор ландшафтов [Електронний ресурс] – Режим доступу до ресурсу: https://3dnews.ru/618577.
5. ИТ-ландшафт современных компаний [Електронний ресурс] – Режим доступу до ресурсу: https://www.it-world.ru/tech/practice/149564.html.
6. Дипломна робота «Процедурна генерація 3D геометрії ігрового контенту» [Електронний ресурс] – Режим доступу до ресурсу: https://ela.kpi.ua/bitstream/123456789/28611/1/Pasichnyk_bakalavr.pdf.
7. Наукова робота «Побудова фрактальних поверхонь в комп’ютерній графіці» [Електронний ресурс] – Режим доступу до ресурсу: https://refdb.ru/look/1482946-pall.html.

ДОДАТОК А. ГЕНЕРАЦІЯ ВИПАДКОВИХ ДАНИХ

Додаток 1 Генерація шуму:

```
import numpy as np
from opensimplex import OpenSimplex
from PIL import Image
#Функція генерації шуму
def generateNoise( height, width):
    gen = OpenSimplex()
    def noise(nx, ny):
        return gen.noise2d(nx, ny) / 2.0 + 0.5
    value = []
    for y in range(height):
        value.append([0] * width)
        for x in range(width):
            nx = x/width - 0.5
            ny = y/height - 0.5
            value[y][x] = noise(nx, ny)
    return value

height = 100
width = 100

#генерація даних шуму
noiseMap  =  generateNoise(height,width)
#Функція збереження даних у вигляді png файлу
im = Image.frombuffer('L', (height, width), np.array(noiseMap).reshape(-1), 'raw',
'L', 0, 1)
im.save('result/noise.png')
```

Додаток 2 Генерація шуму Перлина:

```
from PIL import Image
import PIL
import math
import random
from itertools import product

def smoothstep(t):
    return t * t * (3. - 2. * t)

def lerp(t, a, b):
    return a + t * (b - a)

class PerlinNoiseFactory(object):
    #Функція ініціалізації інтерпольованих даних
    def __init__(self, dimension, octaves=1, tile=(), unbias=False):
        self.dimension = dimension
        self.octaves = octaves
```

```
        self.tile = tile + (0,) * dimension
        self.unbias = unbias
        self.scale_factor = 2 * dimension ** -0.5
        self.gradient = {}

    #Функція створення векторів градієнту
    def _generate_gradient(self):
        if self.dimension == 1:
            return (random.uniform(-1, 1),)
        random_point = [random.gauss(0, 1) for _ in range(self.dimension)]
        scale = sum(n * n for n in random_point) ** -0.5
        return tuple(coord * scale for coord in random_point)

    #Функція генерації шуму
    def get_plain_noise(self, *point):
        if len(point) != self.dimension:
            raise ValueError("Expected {} values, got {}".format(
                self.dimension, len(point)))

        grid_coords = []
        for coord in point:
            min_coord = math.floor(coord)
            max_coord = min_coord + 1
            grid_coords.append((min_coord, max_coord))

        dots = []
        for grid_point in product(*grid_coords):
            if grid_point not in self.gradient:
                self.gradient[grid_point] = self._generate_gradient()
            gradient = self.gradient[grid_point]

            dot = 0
            for i in range(self.dimension):
                dot += gradient[i] * (point[i] - grid_point[i])
            dots.append(dot)
            dim = self.dimension
        while len(dots) > 1:
            dim -= 1
            s = smoothstep(point[dim] - grid_coords[dim][0])

            next_dots = []
            while dots:
                next_dots.append(lerp(s, dots.pop(0), dots.pop(0)))

            dots = next_dots
        return dots[0] * self.scale_factor

    #Функція генерації шуму Перлина
    def __call__(self, *point):
        ret = 0
        for o in range(self.octaves):
            o2 = 1 << o
            new_point = []
            for i, coord in enumerate(point):
                coord *= o2
                if self.tile[i]:
                    coord %= self.tile[i] * o2
                new_point.append(coord)
```

```
            ret += self.get_plain_noise(*new_point) / o2

        ret /= 2 - 2 ** (1 - self.octaves)

        if self.unbias:
            r = (ret + 1) / 2
            for _ in range(int(self.octaves / 2 + 0.5)):
                r = smoothstep(r)
            ret = r * 2 - 1
        return ret

size = 200
res = 25
frames = 50
frameres = 50
space_range = size//res
frame_range = frames//frameres

#генерація шуму Перлина
pnf = PerlinNoiseFactory(3, octaves=4, tile=(space_range, space_range, frame_range))

#Функція збереження даних у вигляді png файлу
img = PIL.Image.new('L', (size, size))
for x in range(size):
    for y in range(size):
        n = pnf(x/res, y/res, 1/frameres)
        img.putpixel((x, y), int((n + 1) / 2 * 255 + 0.5))
img.save("result/perlinnoise.png")
```

Додаток 3 Генерація карти височни на основі фракталу Мандельброта:

```
import math

import PIL
from scipy import interpolate
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interpn, interp2d

pmin, pmax, qmin, qmax = -2.5, 1.5, -2, 2
ppoints, qpoints = 100, 100
max_iterations = 100
infinity_border = 2000

def smoothstep(t):
    return t * t * (3. - 2. * t)

def lerp(t, a, b):
    return a + t * (b - a)

class MandelbrotFractalGenerator(object):
    #Функція ініціалізації інтерпольованих даних
    def __init__(self, pmin, pmax, qmin, qmax, ppoints, qpoints,
max_iterations,infinity_border, smoothdim =3):
        self.pmin = pmin
        self.pmax = pmax
```

```
        self.qmin = qmin
        self.qmax = qmax
        self.ppoints = ppoints
        self.qpoints = qpoints
        self.max_iterations = max_iterations
        self.infinity_border = infinity_border
        self.dimension = smoothdim
        self.gradient = {}

    #Функція генерації інтерполованих даних
    def GenerateImage(self):
        image = np.zeros((self.ppoints, self.qpoints))

        for ip, p in enumerate(np.linspace(self.pmin, self.pmax, self.ppoints)):
            for iq, q in enumerate(np.linspace(self.qmin, self.qmax, self.qpoints)):
                c = p + 1j * q

                z = 0
                for k in range(self.max_iterations):
                    z = z ** 2 * np.cos(z) / 2 - z + c
                    if abs(z) > infinity_border:
                        image[iq, ip] = 1
                        break
        return image

    #Функція створення фрактлу Мандельброта
    def ZFunc(self, X, Y):
        image = np.zeros((X.shape[0], X.shape[1]))

        for ip, p in enumerate(np.linspace(self.pmin, self.pmax,X.shape[0])):
            for iq, q in enumerate(np.linspace(self.qmin, self.qmax, X.shape[1])):
                c = p + 1j * q

                z = 0
                for k in range(self.max_iterations):
                    z = z ** 2 * np.cos(z) / 2 - z + c
                    if abs(z) > infinity_border:
                        image[iq, ip] = 1
                        break
        return image

    #Функція генерації інтерпольованих даних
    def GenerateImageSmoothedImage(self):
        X = np.linspace(1,  self.ppoints,  int(self.ppoints/2))
        Y = np.linspace(1,  self.qpoints,  int(self.qpoints/2))

        x, y = np.meshgrid(X, Y)
        values = np.array(self.ZFunc(x, y))
        a = (interp2d(x,y , values))
        Xnew = np.linspace(1, self.ppoints, self.ppoints)
        Ynew = np.linspace(1, self.ppoints, self.ppoints)

        test8x8 = a(Xnew,Ynew)
        return test8x8


plt.xticks([])
```

```
plt.yticks([])
fr =  MandelbrotFractalGenerator(pmin, pmax, qmin, qmax,ppoints, qpoints,100,2000)

#Генерація інтерпольованих даних
z = fr.GenerateImageSmoothedImage()
#Збереження даних до png файлу
img = PIL.Image.new('L', (ppoints, ppoints))
max = max(z.ravel())
min = min(z.ravel())
for y in range(ppoints):
    for x in range(ppoints):
        value = 1 if z[y,x]>1 else z[y,x]
        value = -1 if z[y,x]<-1 else z[y,x]
        img.putpixel((x, y), int((value + 1) / 2 * 255 + 0.5))
img.save("result/MandelbrotSplinedFractal.png")

#Генерація даних

fr =  MandelbrotFractalGenerator(pmin, pmax, qmin, qmax,ppoints, qpoints,100,2000)

z = fr.GenerateImage()
#Збереження даних до png файлу
a = z.ravel()
max = np.max(a)
min = np.min(a)
img = PIL.Image.new('L', (ppoints, ppoints))
for y in range(ppoints):
    for x in range(ppoints):
        value =(z[y,x]-min)/ ( max - min) * 255
        img.putpixel((x, y), int(value))
img.save("result/MandelbrotFractal.png")
```

Додаток 4 Генерація карти височни на основі фракталу Жулиа:

```
import math

import PIL
from scipy import interpolate
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interpn, interp2d
from numba import jit

def smoothstep(t):
    return t * t * (3. - 2. * t)

def lerp(t, a, b):
    return a + t * (b - a)

class JuliaFractalGenerator(object):

    #Функція ініціалізації інтерпольованих даних
    def __init__(self, w, h, movex,movey,cX,cY,zoom, maxIter):
        self.w, self.h, self.movex, self.movey, self.cX, self.cY, self.zoom,
self.maxIter = w, h, movex,movey,cX,cY,zoom, maxIter

    #Функція генерації даних
    def GenerateImage(self):
```

```
        pix = np.zeros((self.w,self.h))
        for x in range(self.w):
            for y in range(self.h):
                zx = 1.5 * (x - self.w / 2) / (0.5 * self.zoom * self.w) + self.movex
                zy = 1.0 * (y - self.h / 2) / (0.5 * self.zoom * self.h) + self.movey
                i = self.maxIter
                while zx * zx + zy * zy < 4 and i > 1:
                    tmp = zx * zx - zy * zy + self.cX
                    zy, zx = 2.0 * zx * zy + self.cY, tmp
                    i -= 1
                    pix[x, y] = i
        return  pix

    #Функція створення фрактлу Жулиа
    def ZFunc(self, X, Y):
        pix = np.zeros((X.shape[0], X.shape[1]))
        for x in range(X.shape[0]):
            for y in range(Y.shape[1]):
                zx = 1.5 * (x - self.w / 2) / (0.5 * self.zoom * X[x,y]) + self.movex
                zy = 1.0 * (y - self.h / 2) / (0.5 * self.zoom * Y[x,y]) + self.movey
                i = self.maxIter
                while zx * zx + zy * zy < 4 and i > 1:
                    tmp = zx * zx - zy * zy + self.cX
                    zy, zx = 2.0 * zx * zy + self.cY, tmp
                    i -= 1
                    pix[x, y] = i
        return pix

    #Функція генерації інтерпольованих даних
    def GenerateImageSmoothedImage(self):
        X = np.linspace(1,  self.w,  int(self.w/2))
        Y = np.linspace(1,  self.h,  int(self.h/2))

        x, y = np.meshgrid(X, Y)
        values = np.array(self.ZFunc(x, y))
        a = (interp2d(x,y , values))

        Xnew = np.linspace(1, self.w, self.w)
        Ynew = np.linspace(1, self.h, self.h)

        test8x8 = a(Xnew,Ynew)
        return test8x8
w =100
h =100
#Генерація даних
gen = JuliaFractalGenerator(w, h,0,0,-0.7,0.27,1,255)
imgData =gen.GenerateImage()

#Збереження даних до png файлу

img = PIL.Image.new('L', (w, h))
max = max(imgData.ravel())
min = min(imgData.ravel())
for y in range(h):
    for x in range(w):

        value =(imgData[y,x]-min)/ ( max - min) * 255
        img.putpixel((x, y), int(value))
```

```
img.save("Result/juliaFractal.png")
#Генерація інтерпольованих даних
imgData =gen.GenerateImageSmoothedImage()
#Збереження интерпольованих даних до png файлу
img = PIL.Image.new('L', (w, h))
a = imgData.ravel()
max = np.max(a)
min = np.min(a)
for y in range(h):
    for x in range(w):
        value =(imgData[y,x]-min)/ ( max - min) * 255
        img.putpixel((x, y), int(value))

img.save("Result/juliaSplinedFractal.png")
```

ДОДАТОК Б. ГЕНЕРАЦЯ ВИВІД ДАНИХ У ВИГЛЯДІ ЛАНДШАФТУ

Додаток 1 Виведення ландшафту на основі сегментної карти височин:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TerrainBySegmentMapGenerator : MonoBehaviour
{

    public Texture2D mapData;
    public GameObject[] HexTop;
    public GameObject[] buildings;
    // Start is called before the first frame update



    void Start()
    {
        if (mapData.width != mapData.height)
        {
            throw new Exception();
        }
        Color32[,] heightArray = ReadHeightArray();
        BuildTerrain(heightArray);
    }

    // Update is called once per frame
    void Update()
    {

    }

     //Функція побудови террейну
    public void BuildTerrain(Color32[,] HeightArray)
    {
        GameObject clone;
        Transform randomTransform;
        GameObject cloneBuilding;
        Transform randomTransformBuilding; float height;
        for (int i = 0; i < HeightArray.GetLength(0) - 1; i++)
        {
            for (int j = 0; j < HeightArray.GetLength(1); j++)
            {
                 //Створення ландшафту певного вигляду та встановлення його до певної
височини

                int a = 0;
                if (HeightArray[i, j].g > 0 & HeightArray[i, j].g < HexTop.Length)
                {
                    a = 1;
                }
                randomTransform = HexTop[a].transform;
```

```
                height = float.Parse(HeightArray[i, j].r.ToString()) * 2 / 255;
                clone = Instantiate(randomTransform.gameObject, this.transform.position
                + new Vector3((i * 2 - (float)0.5 * i), height, j * 2 - (i % 2 == 0 ? 1 : 0))
, transform.rotation) as GameObject;
                clone.transform.SetParent(this.transform);
                 //Створення об’єкту ландшафту та встановлення його до певної  височини

                int buildingNum = HeightArray[i, j].b;
                if (buildingNum > 0 && buildingNum - 1 < buildings.Length)
                {
                    randomTransformBuilding = buildings[buildingNum - 1].transform;
                    height = float.Parse(HeightArray[i, j].r.ToString()) * 2 / 255;

                    cloneBuilding = Instantiate(randomTransformBuilding.gameObject, this.tran
sform.position + new Vector3((i * 2 - (float)0.5 * i), height + (float)1.75, j * 2 - (i % 2 =
= 0 ? 1 : 0)), transform.rotation) as GameObject;
                    cloneBuilding.transform.SetParent(clone.transform);
                }

            }

        }
    }
    //Функція Зчитування даних

    public Color32[,] ReadHeightArray()
    {
        int imageSize = mapData.width;
        Color32[,] heightArray = new Color32[imageSize, imageSize];
        for (int i = 0; i < imageSize; i++)
        {
            for (int j = 0; j < imageSize; j++)
            {
                Color colors = mapData.GetPixels32()[i * imageSize + j];

                heightArray[i, j] = colors;
            }
        }
        return heightArray;
    }
}
```

Додаток 2 Виведення воксельного ландшафту на основі карти височин:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TerrainGenrator : MonoBehaviour
{
    public GameObject[] baseParts;
    public Texture2D mapData;
    public int height;
    public int size;
```

```
// Start is called before the first frame update
void Start()
{
    if (mapData.width != mapData.height)
    {
        throw new Exception();
    }
    int[,] heightArray = ReadHeightArray();
    BuildTerrain(heightArray);
}

// Update is called once per frame
void Update()
{

}

 //Функція Зчитування даних
public int[,] ReadHeightArray()
{
    int imageSize = mapData.width;
    int Step = imageSize / size;
    int[,] heightArray = new int[size, size];
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            Color[] colors = mapData.GetPixels(i * Step, j * Step, Step, Step);
            float avg = GetAvg(colors);
            heightArray[i, j] = (int)(avg * height);
        }
    }
    return heightArray;
}


//Функція отримання середнього значення кольорів
public float GetAvg(Color[] array)
{
    float avg = 0;
    for (int i = 0; i < array.Length; i++)
    {
        avg += (array[i].r + array[i].g + array[i].b) / 3;
    }
    return avg / array.Length;
}

//Функція побудови террейну
public void BuildTerrain(int[,] ReadHeightArray)
{
    GameObject clone;
    Transform randomTransform;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
```

```
            {

                 //Функція побудови террейну
                for (int g = 0; g < ReadHeightArray[i, j]; g++)
                {
                    if (g + 1 == ReadHeightArray[i, j])
                    {
                        randomTransform = baseParts[0].transform;
                        clone = Instantiate(randomTransform.gameObject, this.transform.positi
on
                            + new Vector3(i * 2, g * 2, j * 2), transform.rotation) as GameOb
ject;
                        clone.transform.SetParent(this.transform);
                        continue;
                    }

                    randomTransform = baseParts[UnityEngine.Random.Range(1, baseParts.Length)
].transform;
                        clone = Instantiate(randomTransform.gameObject, this.transform.positi
on
                        + new Vector3(i*2, g*2, j*2), transform.rotation) as GameObject;
                    clone.transform.SetParent(this.transform);
                }


            }
        }

    }
}
```

ДОДАТОК В. ІНСТРУКЦІЯ КОРИСТУВАЧА

Інструкція до додатку А.

- Імпортуйте файли проекту;
- Оберіть алгоритм, який ви захочете використати по назві файлу;
- Замініть константні дані в файлі, на ті що вам потрібні;
- Запустіть python скрипт за допомогою інтерпритатору python версії 3.8 або вище;
- Дочекайтесь завершення роботи скрипту;
- Отримайте результат у папці в вигляді png файлу, назва якого відповідає назві обраного вами методу.

Інструкція до додатку Б.

- Імпортуйте файли проекту;
- Створіть сцену в проекті та відкрийте її;
- Створіть пустий об’єкт на сцені;
- Зв’яжіть обраний вами скрипт з пустим об’єктом;
- Імпортуйте в проект карту височин;
- Дозвольте, в властивостях карти височин, зчитувати/записувати та встановіть в властивість “Non Power of 2” значення None;
- Встановіть в вхідні дані скрипту карту височин;
- Передайте в скрипт об’єкти, які були заготовлені для ландшафту, з яких буде складатись ландшафт;
- Передайте розміри в скрипт, якщо він потребує їх;
- Запустіть програму;
- Отримайте шуканий результат.