ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ОЛЕСЯ ГОНЧАРА

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

Кваліфікаційна робота

перший (бакалаврський) рівень вищої освіти

спеціальність 113 Прикладна математика

освітня програма: Комп’ютерне моделювання та технології програмування

РОЗРОБКА СИСТЕМИ ПОЛІПШЕННЯ ЛАНДШАФТУ З ВИКОРИСТАННЯМ АЛГОРИТМІВ ЕРОЗІЇ ПОВЕРХОНЬ

Виконавець

студент групи ПА-19-2

\_Ільяшенко Єгор Віталійович\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(П.І.Б.) (підпис)

Керівник

доц., канд. фіз.-мат. наук

\_Степанова  Наталія Іванівна\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(П.І.Б.) (підпис)

2023 р.

# **ЗМІСТ**

[ЗМІСТ 2](#_Toc168136713)

[ВСТУП 4](#_Toc168136714)

[ПОСТАНОВКА ЗАДАЧІ 5](#_Toc168136715)

[1 АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ 6](#_Toc168136716)

[1.1 Псевдо-випадкові послідовності 6](#_Toc168136717)

[1.2 Хеш функції 7](#_Toc168136718)

[1.3 Псевдовипадкові шуми 8](#_Toc168136719)

[1.4 Шум Перліна 9](#_Toc168136720)

[1.5 Шум Симплекс 10](#_Toc168136721)

[1.6 Шум Вороний 12](#_Toc168136722)

[1.7 Комбінація рівней шуму 13](#_Toc168136723)

[1.8 Принципи формування ландшафту 13](#_Toc168136724)

[1.9 Меш. Загальні поняття 13](#_Toc168136725)

[1.10 Нормалі мешу 13](#_Toc168136726)

[2 МАТЕМАТИЧНА МОДЕЛЬ 14](#_Toc168136727)

[2.1 Шум Перліна 14](#_Toc168136728)

[2.2 Кубо-сфера 15](#_Toc168136729)

[3 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 17](#_Toc168136730)

[3.1 Платформа Unity 17](#_Toc168136731)

[3.2 Render pipeline 18](#_Toc168136732)

[3.3 Shader graph 19](#_Toc168136733)

[4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ 21](#_Toc168136734)

[ВИСНОВКИ 28](#_Toc168136735)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 29](#_Toc168136736)

[ДОДАТКИ 30](#_Toc168136737)

[Додаток А 30](#_Toc168136738)

[Додаток Б 30](#_Toc168136739)

[Додаток В 30](#_Toc168136740)

# ВСТУП

У сучасному світі тривимірна комп'ютерна графіка використовується в багатьох сферах життя. Прикладами можуть стати:

* Ігрова індустрія;
* Виготовлення фільмів;
* Віртуальна реальність;
* Різноманітні симуляції.

Одним із ключових завдань тривимірного графічного дизайну є створення реалістичних і деталізованих поверхонь. Ландшафтні поверхні найчастіше складаються з двовимірного масиву значень, які відображають висоту ландшафту в кожній точці координат. Створення таких поверхонь має велике значення для забезпечення реалістичності візуалізації у віртуальних світах.

Класичний спосіб створення ландшафтної поверхні передбачає вибір розмірності двовимірного масиву та заповнення всіх значень вручну або за допомогою спеціальних інструментів. Існує безліч програмних засобів, які спрощують цей процес, дозволяючи користувачам малювати карти висот за допомогою пензлів і інших інструментів. Такі карти висот потім зберігаються і можуть бути використані для візуалізації ландшафту.

Проте такий підхід має значні недоліки. Кожну поверхню необхідно створювати вручну, що потребує багато часу і зусиль. Крім того, для отримання високої деталізації і варіативності ландшафту потрібно генерувати велику кількість даних, що може бути ресурсоємним процесом. Зокрема, створення великих і деталізованих ландшафтів вимагає значних обчислювальних ресурсів для їх обробки та зберігання.

Альтернативний підхід до створення ландшафту, який вирішує багато з цих проблем, це процедурна генерація поверхонь. Процедурна генерація є інноваційним і ефективним методом, що дозволяє автоматично створювати складні і реалістичні ландшафти за допомогою спеціальних алгоритмів та математичних моделей.

Цей підхід ґрунтується на використанні різних алгоритмів генерації шуму, фрактальних алгоритмів та симуляцій фізичних процесів, що дозволяє досягти високого рівня деталізації та природності у створенні ландшафтів. Замість ручного моделювання кожного елемента ландшафту, процедурна генерація дозволяє створювати великі території з високою роздільною здатністю та варіативністю за допомогою визначених правил та параметрів.

Однією з головних переваг процедурної генерації є можливість збереження всього ландшафту у вигляді невеликого набору параметрів або seed, що значно зменшує вимоги до зберігання даних. Це також дозволяє швидко змінювати та відтворювати ландшафт, змінюючи лише початкові параметри або seed, що робить цей метод дуже гнучким та ефективним.

Процедурна генерація також забезпечує високу швидкість створення ландшафтів. Завдяки використанню ефективних алгоритмів, процес генерації може бути автоматизований та виконуватися в реальному часі, що особливо корисно для застосувань у ігровій індустрії та віртуальній реальності, де важлива динамічна зміна навколишнього середовища.

Окрім цього, процедурна генерація дозволяє створювати дуже великі та складні світи, які можуть бути практично нескінченними. Це особливо важливо для сучасних комп'ютерних ігор та симуляцій, де гравці або користувачі можуть досліджувати великі території з різноманітними біомами та географічними особливостями.

Таким чином, процедурна генерація поверхонь є потужним інструментом для створення реалістичних та деталізованих ландшафтів. Використання алгоритмів та математичних моделей дозволяє автоматизувати процес створення, забезпечуючи високу ефективність, гнучкість та якість кінцевого результату. Цей підхід значно спрощує роботу розробників та художників, дозволяючи їм зосередитися на творчих аспектах проектування, залишаючи технічні деталі генерації алгоритмам.

Але такі поверхні не завжди є реалістичними. Реальні ландшафти генеруються за допомогою фізичних процесів, таких як ерозія, зсув літосферних плит. Тож альтернативний шлях для генерації поверхонь – зробити симуляцію на основі фізичного процесу. Один із прикладів такого генерації - це симуляції ерозії на поверхні карти висот.

Алгоритми ерозії поверхонь імітують природні процеси ерозії, такі як вплив води, вітру і температури, на формування ландшафту. Вони дозволяють створювати реалістичні і динамічні поверхні, які виглядають природно і переконливо. Ерозія поверхонь додає деталізацію і складність до процедурно згенерованих ландшафтів, роблячи їх більш реалістичними.

Цей принцип генерації поверхонь має обмежену кількість сфер, де він може застосовуватися. Тоді як при генерації через випадкові шуми ми можемо перегенерувати будь-яку точку у будь-який час, при симуляції ми повинні мати готову базову карту висот, на які вже накладається симуляція. Оскільки час, необхідний на симуляцію, сильно залежить від розміру поверхні, та самої симуляції і, зазвичай, не виконується у межах 2мс, цей принцип не може використовуватися для миттєвої генерації поверхні і повинен бути підготовлений і збережений у пам'яті заздалегідь.

# ПОСТАНОВКА ЗАДАЧІ

Метою цієї роботи є розробка та реалізація ефективної системи поліпшення ландшафту з використанням алгоритмів ерозії поверхонь. У рамках цієї роботи будуть вивчені існуючі методи ерозії, проведено їх порівняльний аналіз та вибрано найбільш оптимальні підходи для розробки системи.

Основними завданнями цієї роботи є:

* Вивчення теоретичних основ і існуючих підходів до ерозії поверхонь.
* Розробка та реалізація системи поліпшення ландшафту з використанням алгоритмів ерозії поверхонь.
* Проведення експериментів для оцінки ефективності розробленої системи та порівняння з іншими методами.

Результати цієї роботи можуть мати практичне застосування в галузі комп'ютерної графіки, допомагаючи створювати більш реалістичні та деталізовані ландшафти для ігор, фільмів, віртуальної реальності та інших застосувань. Крім того, отримані знання і розроблені методи можуть бути використані для подальшого вдосконалення систем генерації поверхонь і створення ще більш вражаючих візуальних ефектів.

# 1 АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ

## 1.1 Псевдо-випадкові послідовності

Псевдо-випадкові послідовності є послідовностями чисел, які здавалося би мають випадковий характер, але фактично генеруються за допомогою алгоритмів псевдовипадкових генераторів. Ці алгоритми базуються на математичних формулах та початкових значеннях, відомих як seed або насіння.

Використовуючи однакове насіння, псевдо-випадковий генератор завжди буде генерувати однакову послідовність чисел. Однак, якщо змінити насіння, буде отримана інша послідовність чисел. Це дозволяє контролювати генерацію псевдо-випадкових послідовностей і забезпечує відтворюваність результатів.

Псевдо-випадкові послідовності широко застосовуються у різних галузях. У криптографії, вони використовуються для генерації випадкових ключів і векторів ініціалізації, що забезпечує безпеку і криптографічну стійкість. В моделюванні випадкових явищ, псевдо-випадкові послідовності допомагають створити віртуальні експерименти та симуляції, що дозволяють досліджувати та передбачати поведінку систем.

У багатьох програмах та алгоритмах, які вимагають випадковості, псевдо-випадкові послідовності можуть бути прийнятним рішенням через свою ефективність та передбачуваність. Однак, в криптографічних задачах, де безпека є критичною, використання справжньо випадкових послідовностей є необхідним.

Застосування псевдо-випадкових послідовностей включають генерацію градієнт-векторів для побудови псевдовипадкових шумів, що використовуються у візуальних ефектах, комп'ютерній графіці та моделюванні природних явищ. Ці шуми можуть створити враження реалістичної текстури, нерівномірного освітлення або вібрацій віддалених об'єктів.

В цілому, псевдо-випадкові послідовності є корисним інструментом для великої кількості застосувань, де випадковість або недетермінованість є важливою властивістю. Розуміння їх принципів та обмежень допомагає забезпечити правильне використання і відповідну оцінку їхньої випадковості в конкретних ситуаціях.

## 1.2 Хеш функції

Хеш-функція - це функція, яка приймає на вхід послідовність даних будь-якої довжини і обчислює фіксований вихідний код, відомий як хеш-код. Одна з основних властивостей хеш-функцій - це важкість знаходження двох різних вхідних послідовностей, які мають один і той самий хеш-код. Це називається колізією і в хороших хеш-функціях вони виникають дуже рідко.

Хеш-функції мають широке застосування в різних галузях. Одним з найпоширеніших використань є зберігання паролів. Замість зберігання самих паролів, хеш-функція обчислює хеш-код пароля і зберігається лише цей код. При перевірці пароля, введені дані обчислюються хеш-функцією, і їх хеш-код порівнюється зі збереженим. Це дозволяє перевірити правильність пароля без зберігання самого пароля, що забезпечує безпеку в разі витоку бази даних.

Хеш-функції також використовуються для перевірки цілісності даних. Наприклад, при завантаженні файлу з Інтернету можна перевірити його цілісність, порівнявши хеш-код файлу, обчислений на локальному комп'ютері, з вказаним хеш-кодом на веб-сайті. Якщо хеш-коди співпадають, це означає, що файл не був пошкоджений під час передачі.

Хеш-функції також застосовуються для швидкого доступу до даних. Наприклад, в хеш-таблицях ключі перетворюються в хеш-коди, які використовуються для швидкого пошуку значень. Це особливо корисно для пошуку великих наборів даних.

У вашому випадку, для генерації градієнт-векторів хеш-функція використовується для перетворення тривимірних координат і додаткових даних (seed) у фіксований хеш-код. Цей хеш-код може служити як початкова точка для псевдо-випадкової послідовності, яка буде використовуватися для генерації псевдовипадкових шумів. Таким чином, ми отримуємо контрольований і передбачуваний спосіб генерації шумів залежно від вхідних даних.

## 1.3 Псевдовипадкові шуми

Псевдовипадкові шуми є шумами, що генеруються за допомогою псевдовипадкових алгоритмів. Ці шуми можуть бути будь якої вимірності і можуть бути використані для моделювання природних або випадкових явищ, таких як вітрові коливання, хвилі в океані або деталі пейзажу. Вони генеруються шляхом використання математичних формул та псевдовипадкових послідовностей, що надають їм властивості випадковості, але при цьому є детермінованими. Псевдовипадкові шуми є корисним інструментом у багатьох галузях, включаючи комп'ютерну графіку, візуальні ефекти, моделювання природних явищ, аудіо та багато інших.

Один з найпопулярніших видів псевдовипадкових шумів - це шум Перліна, який названий на честь Кена Перліна, який його вперше запропонував. Шум Перліна генерується за допомогою градієнтного вектора, що призводить до створення нерегулярних і плавних змін яскравості або висоти. Цей вид шуму часто використовується для створення текстур, рельєфів і природних ландшафтів.

Інший популярний вид псевдовипадкових шумів - це симплекс-шум (simplex noise), який є вдосконаленням шуму Перліна. Симплекс-шум забезпечує більш гладкі переходи і вищу якість результату. Він часто використовується у графічних програмах для створення природних ефектів, таких як хвилі, хмари і трава.

Ще один поширений вид псевдовипадкових шумів - це шум Вороного (Voronoi noise). Він базується на теорії Вороного і створює мозаїчні ефекти з утворенням клітинок, які заповнюють простір. Шум Вороного використовується для створення вигляду кристалів, каміння або інших нерегулярних структур.

Кожен вид псевдовипадкових шумів має свої особливості і застосування в конкретних областях. Вони дозволяють створити візуально привабливі і реалістичні ефекти, що додають деталізацію та реалізм до комп'ютерних сцен і моделей.

## 1.4 Шум Перліна

Шум Перліна (Perlin noise) - це вид псевдовипадкового шуму, що використовується для моделювання нерегулярних текстур, таких як поверхні, хмари, вогні та інше. Був винайдений Кеном Перліном, створює плавні переходи між значеннями, створюючи ілюзію природного шуму. Шум Перліна генерується за допомогою інтерполяції градієнтів у випадкових точках сітки. Шум Перліна може бути будь якої розмірності, але зазвичай реалізується, як двовимірна, або тривимірна функція. У моєму випадку я буду використовувати тривимірний шум Перліна, бо генерації поверхні буде виконуватися у тривимірному просторі і двовимірна площина буде мати артефакти при накладанні на сферу.

Реалізації шуму Перліна проводиться в декілька етапів:

1. Спочатку ми визначаємо n-вимірну сітку, де кожній координаті сітки присвоюється n-вимірний вектор (градієнт) [рис. 1.1](#Рисунок_1_1). Цей вектор рандомізують за допомогою псевдо-випадкової послідовності, використовуючи координату сітки – як seed. Оскільки я реалізовував тривимірний шум Перліна, у моєму випадку це тривимірний вектор. Надалі, цей вектор потрібно нормалізувати.
2. Другий крок – це визначити, до якої комірки у нашій сітці потрапляє точка, значення якої треба знайти.
3. Для кожного вузла комірки визначаємо вектор відстані між точкою та координатами вузла.
4. Надалі обчислюємо скалярні добутки векторів відстані та градієнтних векторів кожного вузла комірки.
5. Значення скалярних добутків інтерполюють і ми отримуємо фінальну точку у потрібній координаті.

A picture containing diagram, line, colorfulness, plot

Description automatically generated

Рисунок 1.1 - Градієнт-вектори у вершинах сітки та їх інтерполяція

Така процедура повторюється для кожної точки, яку нам потрібно визначити. В результаті виконання двовимірного варіанту алгоритму, ми отримуємо набір точок, який можна перевести у зображення [рис. 1.2](#Рисунок_1_2).

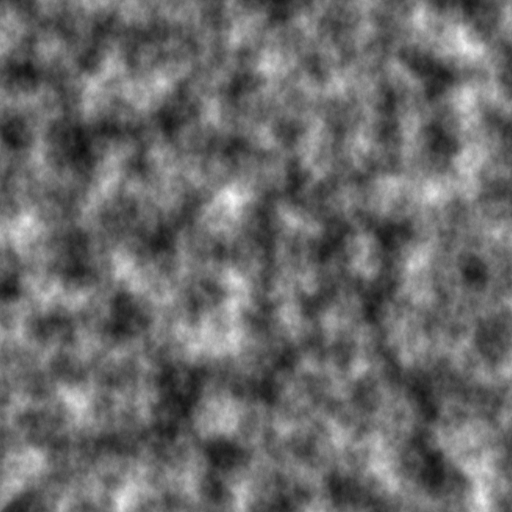


Рисунок 1.2 – Результат виконання алгоритму шуму Перліна

## 1.5 Шум Симплекс

Шум Симплекс (Simplex noise) є одним з популярних методів генерації псевдовипадкових шумів, який знаходить широке застосування у комп'ютерній графіці, симуляції природних явищ та інших областях [рис 1.3](#Рисунок_1_3).

У порівнянні з шумом Перліна, шум Симплекс пропонує поліпшення у якості та властивостях згладженості шумової поверхні. Цей метод використовує багатовимірні симплекси замість градієнтів на сітці, що дозволяє краще розподіляти значення шуму та уникати артефактів.

Шум Симплекс має деякі переваги порівняно з шумом Перліна, зокрема:

* Краща згладженість: Завдяки використанню симплексів замість градієнтів, шум Симплекс забезпечує більш плавні та згладжені переходи між значеннями шуму. Це дозволяє отримувати більш природні та реалістичні результати.
* Більша ефективність: Алгоритм генерації шуму Симплекс є більш ефективним, оскільки він використовує менше обчислювальних операцій для отримання значення шуму.
* Менше артефактів: Шум Симплекс має менше артефактів порівняно з шумом Перліна, таких як видимі сітчасті структури. Це дозволяє отримувати більш рівномірний та природній вигляд шумової поверхні.

Вибір між шумом Перліна та шумом Симплекс залежить від конкретної задачі і потреб у якості та ефективності. Обидва методи є корисними інструментами для моделювання природних та випадкових явищ, і вибір одного чи іншого може бути залежний від особливостей конкретного використання.

A close-up of a black and white speckled surface

Description automatically generated with low confidence

Рисунок 1.3 – Результат виконання шуму Сиплекс.

Хоч шум Симплекс і являється покращеною версією шума Перліна, при генеруванні поверхні вибір між цими основними шумами не грає великої ролі у кінцевому результаті.

## 1.6 Шум Вороний

Шум Вороного (Voronoi noise) є цікавим методом генерації псевдовипадкових шумів, який базується на діаграмі Вороного. Діаграма Вороного розбиває простір на області, використовуючи набір точок як базис. Кожна точка у шумі Вороного належить до своєї найближчої точки в цій діаграмі, і це надає шуму його характерний вигляд [рис. 1.4](#Рисунок_1_4).

Генерація шуму Вороного зазвичай включає наступні кроки:

* Вибір початкового набору точок: Генерується набір точок випадковим чином або за допомогою іншого псевдовипадкового алгоритму. Ці точки виступають як базис для діаграми Вороного.
* Обчислення найближчих точок: Кожній точці у просторі присвоюється значення, відповідне найближчій точці з початкового набору. Це може бути відстань або інше значення, яке відображає відношення до найближчої точки.
* Застосування інтерполяції: Для отримання плавних переходів між точками використовується інтерполяція значень шуму.
* Модифікація значень: Застосування додаткових операцій, таких як згладжування, масштабування або перетворення значень, може змінити вигляд та характеристики шуму Вороного.

Шум Вороного може бути використаний для створення випадкових паттернів, таких як плями, фрактальні структури або границі між областями. Його хаотичний та непередбачуваний вигляд робить його корисним інструментом для моделювання природних та випадкових явищ, а також для генерації художніх ефектів у комп'ютерній графіці. У нашому випадку, шум Вороного може використовуватися задля генерації варіацій у біомах поверхні, або для генерації тектонічних плит, якщо планета генерується принципом симуляції.

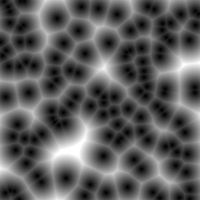


Рисунок 1.4 – Шум Вороного

## 1.7 Комбінація рівней шуму

Комбінація рівнів шуму є важливим прийомом у генерації складних текстур та ландшафтів. Зазвичай для цього використовують різні види псевдовипадкових шумів, такі як шум Перліна або шум Симплекс. Комбінуючи кілька рівнів шуму з різними частотами та амплітудами, можна створити складні, багатошарові текстури, які виглядають більш природно.

Изображение выглядит как шаблон, дизайн, серый, ткань

Автоматически созданное описаниеИзображение выглядит как шаблон, ткань

Автоматически созданное описание

Рисунок 1.5 – Три рівні шуму Симплекс, окремо один від одного

Основний принцип полягає в додаванні шумів з різними параметрами. Наприклад, можна додати високочастотний шум для створення дрібних деталей і низькочастотний шум для великих структур. Це створює ефект фрактальності, коли дрібні деталі накладаються на великі форми, що імітує природні ландшафти.

Процедура комбінації шумів включає:

1. Генерацію базового шару низькочастотного шуму.
2. Додавання середньочастотного шуму з меншою амплітудою.
3. Накладення високочастотного шуму для деталізації.

Кінцевий результат - це багатошаровий ландшафт, що виглядає реалістично завдяки поєднанню різних рівнів шуму.



Рисунок 1.6 – Комбінований шум

## 1.8 Принципи формування ландшафту

Формування ландшафту - це процес створення реалістичних тривимірних моделей поверхні землі або іншої планети. Основні принципи формування ландшафту включають використання псевдовипадкових шумів, фрактальних алгоритмів та процедурного моделювання. Ці методи дозволяють створити складні та реалістичні ландшафти з мінімальними витратами ресурсів.

Основні етапи формування ландшафту:

1. Генерація висотної мапи: використання псевдовипадкових шумів для створення основної структури ландшафту.
2. Застосування ерозійних алгоритмів: симуляція водної та вітрової ерозії для створення природних форм.
3. Додавання деталей: використання додаткових шарів шуму для створення дрібних деталей, таких як скелі та западини.
4. Текстурування: накладання текстур на поверхню для створення реалістичного вигляду.

Ці принципи дозволяють створювати різноманітні ландшафти, від гірських хребтів до пустель та рівнин, забезпечуючи високу якість та реалістичність результату.

## 1.9 Меш. Загальні поняття

Меш (mesh) - це базовий елемент тривимірної графіки, що представляє поверхню об'єкта за допомогою вершин, ребер та граней. Кожен меш складається з набору вершин (точок у просторі), з'єднаних ребрами, які утворюють грані (трикутники або полігони). Меші використовуються для моделювання будь-яких тривимірних об'єктів, від простих форм до складних моделей.

Основні компоненти мешу:

1. Вершини (vertices): точки в тривимірному просторі, що визначають форму мешу.
2. Ребра (edges): лінії, що з'єднують вершини.
3. Грані (faces): полігони, утворені з ребер. Найчастіше використовуються трикутники через їх простоту і стабільність у рендерингу.

Меші можуть бути створені вручну або за допомогою процедурних алгоритмів. Вони є основою для текстурування, освітлення та анімації в тривимірних сценах.

## 1.10 Нормалі мешу

Нормалі мешу - це вектори, перпендикулярні до поверхні граней мешу. Вони використовуються для визначення освітлення та тіней на поверхні об'єкта. Кожна вершина та грань мешу має свою нормаль, яка впливає на те, як світло взаємодіє з поверхнею.

Основні аспекти нормалей:

1. Нормалі граней (face normals): перпендикулярні до кожної грані. Використовуються для плоского затінення.
2. Нормалі вершин (vertex normals): обчислюються як середнє значення нормалей суміжних граней. Використовуються для згладженого затінення.

Правильне визначення нормалей є критично важливим для реалістичного відображення освітлення на поверхні об'єкта. Вони допомагають створювати гладкі переходи між гранями, забезпечуючи високу якість візуалізації.

## 1.11 Ерозія поверхні

Ерозія поверхні - це ключовий процес у моделюванні природних ландшафтів, який додає реалістичні деталі та динаміку поверхні. Ерозія імітує вплив природних явищ, таких як дощ, вітер та текуча вода, на формування та зміну поверхні. Ваш алгоритм, який генерує одну каплю за раз, є прикладом гідравлічної ерозії, що є однією з найпоширеніших методів ерозії у комп'ютерному моделюванні.



Рисунок 1.6 – Карта висот після виконання алгоритму ерозії

**Основні концепції гідравлічної ерозії**

Гідравлічна ерозія моделює взаємодію води з поверхнею, включаючи процеси депонування та транспортування матеріалів. Основні етапи гідравлічної ерозії включають:

1. **Генерація каплі**: Капля води падає на поверхню в певній точці.
2. **Ерозія**: Капля збирає частинки матеріалу з поверхні, що викликає ерозію в цій точці.
3. **Транспортування**: Капля рухається поверхнею під дією гравітації, збираючи та депонуючи матеріал на своєму шляху.
4. **Депонування**: Коли капля більше не може транспортувати матеріал, вона депонує його на новому місці.

**Алгоритм гідравлічної ерозії**

Алгоритм гідравлічної ерозії виконується наступними кроками:

1. **Генерація каплі**: На початку кожної ітерації створюється капля в випадковій точці на поверхні.
2. **Рух каплі**:
   * Визначення напрямку руху каплі на основі градієнту висоти поверхні.
   * Рух каплі до наступної точки у напрямку максимального зниження висоти.
3. **Ерозія матеріалу**:
   * Капля збирає матеріал з поточної точки пропорційно до нахилу поверхні та кількості води в каплі.
   * Втрата матеріалу з поверхні моделюється шляхом зменшення висоти в точці.
4. **Транспортування матеріалу**:
   * Капля переносить зібраний матеріал до нової точки.
   * Транспортування залежить від швидкості руху каплі, яка зменшується зі збільшенням ваги каплі.
5. **Депонування матеріалу**:
   * Капля депонує матеріал у точках зниження швидкості або коли вода в каплі випаровується.
   * Збільшення висоти в точці депонування моделює накопичення матеріалу.
6. **Повторення процесу**: Процес повторюється для заданої кількості ітерацій або поки капля не втратить всю воду.

**Параметри ерозії**

Основні параметри, що впливають на процес ерозії:

* **Кількість капель**: Визначає, скільки разів процес буде повторюватися.
* **Інтенсивність ерозії**: Визначає, скільки матеріалу капля може зібрати з поверхні.
* **Швидкість випаровування**: Визначає, як швидко вода випаровується з каплі.
* **Максимальна кількість матеріалу**: Обмежує кількість матеріалу, яку капля може транспортувати.

**Переваги та недоліки**

Гідравлічна ерозія має декілька переваг та недоліків:

* **Переваги**:
  + Реалістичність: Імітує природні процеси формування ландшафту.
  + Деталізація: Додає деталі та складність поверхні.
* **Недоліки**:
  + Обчислювальна складність: Вимагає значних обчислювальних ресурсів.
  + Параметризація: Потребує точного налаштування параметрів для досягнення бажаних результатів.

**Застосування в моделюванні**

Ерозія поверхні знаходить широке застосування в різних галузях:

* **Ігрова індустрія**: Використовується для створення реалістичних і динамічних ландшафтів.
* **Анімація та візуальні ефекти**: Створення природних текстур і динамічних середовищ.
* **Наукові дослідження**: Моделювання геологічних процесів та їх впливу на навколишнє середовище.

# 2 МАТЕМАТИЧНА МОДЕЛЬ

## 2.1 Шум Перліна

Пошук точки у шумі Перліна зводиться до пошуку найближчих градієнт векторів до даної точки. У випадку тривимірного шуму Перліну наша точка попадає у так названу комірку, яка представляє собою куб. Полотно шуму складається з багатьох таких кубів, які являють собою сітку. У кожній вершині цієї комірки ми знаходимо значення градієнт вектора. Градієнт-вектор – це вектор однакової з шумом розмірності, де когне зі значень вектору рандомізується, використовуючи псевдо-випаднові послідовності. Сам вектор обов'язково повинен бути нормалізований перед його подальшим використанням. Одна комірка шуму Перліна визначається шляхом віднімання від поточної точки остачі від ділення на розмір комірки [(2.1)](#Формула_2_1), де мінімальна точка у комірці по всім осям, – точка, в якій нам потрібно знайти значення шуму, S – розмір комірки шуму, ‘%’ – операція остачі від ділення.

(2.1)

Остання точка комірки визначається шляхом додавання розміру комірки до першої точки [(2.2)](#Формула_2_2).

(2.2)

Таким чином ми отримали мінімальну та максимальну точку у кубі. Шляхом змішування координат цих двох точок, ми можемо знайти усі 8 вершин куба, оскільки сторони куба паралельні осям координат у просторі шуму. Наступна задача – це у кожній з цих точок знайти значення градієнт-вектора.

Градієнт-вектор у точці , де функція-рандомізатор, яка приймає координату точки, як seed, визначається за формулою [(2.3)](#Формула_2_3).

(2.3)

Після того, як ми отримали вісім градієнт-векторів, нам потрібно їх інтерполювати. Оскільки ми працюємо з векторами, доречно використовувати скалярний добуток. Робити скалярний добуток ми будемо між градієнт-вектором у цій точці та вектором від бажаної точки до вершини комірки.

Скалярне значення у точці після скалярного добутку між градієнт-вектором та відстанню між визначається за формулою [(2.4)](#Формула_2_4).

(2.4)

Після цієї операції в нас залишаються вісім скалярних значень. Значення інтерполюються між собою за законом , де V – результуюче значення шуму у точці .

Таким чином ми можемо заповнити текстуру будь-якого розміру, а саму комірну можна згенерувати лише один раз і змінювати тільки тоді, коли бажана точка виходить з цього проміжку.

## 2.2 Кубо-сфера

Щоб побудувати поверхню, нам спочатку потрібно знайти точки, в яких ми будемо знаходити точки поверхні. Оскільки ми намагаємося отримати поверхню сфери, доречно використовувати принцип побудови сферичної поверхні під назвою «кубосфера». Для кожної з шости сторін куба буде використовуватися одий і той самий алгоритм. Знаходження точок одніїє сторони куба за допомогою лінійної комбінації векторів виконується за формулою [(2.5)](#Формула_2_5), де - точка на стороні куба, одиничні вектори, розташовані увздовж локальних вісей сторони куба, центр сторони куба, r – радіус сфери, кількість точок уздовж однієї осі.

(2.5)

В результаті ми отримуємо 6 сторін куба, кожна з яких складається з n\*n точок. – це фінальна координата точки на сфері, яка знаходження шляхом нормалізації точок Кубо-сфери [(2.6)](#Формула_2_6). Після помноження на її радіус, ми отримуємо точки ідеальної сфери, які надалі можна використовувати для накладання октав шуму Перліна.

(2.6)

Для отримання фінальної точки поверхні, ми беремо значення шуму у кожній з цих точок. Оскільки ці точки представлені у вигляді векторів, ми помножаємо модифікований результат шуму на наш вектор і отримуємо фінальну точку поверхні. При використанні багатьох октав шуму, кожна наступна октава має вдвічі менший вплив на поверхню, ніж попередня.

# 3 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Платформа Unity

Для реалізації програми я обрав платформу Unity3D, яке є одним з найпопулярніших і потужних програмних забезпечень для розробки ігор, симуляцій і візуальних додатків [рис. 3.1](#Рисунок_3_1). Воно надає інтегроване середовище розробки, яке дозволяє створювати інтерактивні досвіди для різних платформ, включаючи комп'ютери, мобільні пристрої, віртуальну реальність та інші.

A black and white logo

Description automatically generated with medium confidence

Рисунок 3.1 – Логотип Unity3D

Основні функції Unity3D:

1. Графічний редактор: Unity3D має потужний графічний редактор, який дозволяє візуально створювати та редагувати об'єкти, сцени, ефекти та інші компоненти гри. Це дозволяє розробникам швидко створювати та налаштовувати візуальний вигляд своїх проектів.
2. Скриптування: Unity3D підтримує різні мови програмування, такі як C#, JavaScript і Boo. Це дає розробникам велику гнучкість у виборі мови програмування та створенні власної логіки гри або додатку.
3. Фізика та колізії: Unity3D має вбудовану систему фізики, яка дозволяє реалістично моделювати рух об'єктів, колізії та взаємодії між ними. Це робить гру більш живою та реалістичною.
4. Імпорт активів: Unity3D підтримує широкий спектр форматів активів, таких як зображення, звук, відео, 3D-моделі, анімації тощо. Це дозволяє розробникам використовувати свої власні ресурси або завантажувати готові активи з онлайн-бібліотек або ринків активів.
5. Мультиплатформена розробка: Unity3D підтримує розробку для різних платформ, включаючи Windows, macOS, iOS, Android, Xbox, PlayStation і багато інших. Це дозволяє розробникам створювати свої проекти для багатьох різних пристроїв безпосередньо з Unity3D.

Unity3D використовується як професійними розробниками, так і студентами для створення ігор, симуляцій, тренажерів, візуалізаційних додатків та багато іншого. Його потужність, гнучкість та активна спільнота розробників роблять Unity3D популярним вибором для будь-якої проектної потреби.

## 3.2 Render pipeline

Render pipeline - це технологія, яка визначає послідовність кроків, необхідних для відтворення графічного вмісту на екрані. Ця технологія виконує різні обчислення та операції для кожного кадру, включаючи обробку геометрії, освітлення, тіней, текстури та інших ефектів, що дозволяють створити реалістичне зображення.

У Unity, за замовчуванням використовується Build-in Render Pipeline, який є застарілим і менш гнучким. Проте, Unity також пропонує альтернативні варіанти рендер-пайплайнів для поліпшення графічної якості та продуктивності.

Одним з варіантів є High Definition Render Pipeline (HDRP), який спрямований на створення найреалістичніших графічних сцен. Він підтримує фізично-коректні матеріали, реалістичне освітлення, тіні, рефлексії та інші передові ефекти. HDRP використовує сучасні техніки рендерингу, такі як Global Illumination (GI) та Screen Space Reflections (SSR), для досягнення високої якості зображення.

Іншим варіантом є Universal Render Pipeline (URP), який покликаний оптимізувати продуктивність і підтримувати більш широкий спектр платформ. URP використовує сучасні техніки рендерингу, такі як Deferred Shading та Screen Space Ambient Occlusion (SSAO), а також підтримує можливість використання ShaderGraph для створення шейдерів за допомогою візуального редагування графів.

Universal Render Pipeline надає гнучкість та оптимізацію, дозволяючи розробникам досягти графічної якості, яка відповідає їх потребам, одночасно забезпечуючи хорошу продуктивність на різних платформах. Використання ShaderGraph у поєднанні з URP дозволяє створювати шейдери за допомогою візуального редагування графів, що спрощує процес розробки шейдерів і дозволяє швидко досягти бажаного візуального ефекту.

Вибір між HDRP і URP залежить від конкретних потреб проекту, де HDRP надає найбільшу графічну якість, а URP забезпечує кращу продуктивність і ширшу підтримку платформ.

Також він надає можливість написання шейдерів шляхом складання графу з різних методів за допомогою ShaderGraph.

## 3.3 Shader graph

Shader graph - це інструмент, який дозволяє візуально створювати і редагувати шейдерні графи. Він забезпечує графічне середовище, де ви можете складати графи з вузлів, що представляють шейдерні ефекти та операції. Це дає можливість програмувати візуально, без необхідності писати код вручну.

Shader graph надає велику кількість готових вузлів, що представляють різноманітні шейдерні операції, наприклад, вузли для обчислення освітлення, кольору, текстурного зчитування, альфа-змішування та багато інших. Ви можете з'єднувати ці вузли між собою, створюючи складні графи, які описують поведінку шейдера.

Після того, як ви скомпонували шейдерний граф, його необхідно компілювати в шейдерний код, що може бути використаний у програмі. Компілятор shader graph перетворює граф у два окремі файли: один для вершинного шейдера (vertex shader) інший для фрагментного шейдера (fragment shader). Вершинний шейдер обробляє кожен вершину геометрії, встановлюючи її позицію та інші властивості. Фрагментний шейдер працює на рівні фрагментів пікселів, обчислюючи кольори та інші атрибути, які відображаються на відображенні екрану.

Shader graph дозволяє швидко створювати та налаштовувати шейдерні ефекти без необхідності в глибокому розумінні шейдерного програмування. Це робить його корисним інструментом для розробки графічних програм, ігор, візуалізації даних та багатьох інших застосувань, де необхідно використовувати шейдери.

# 4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Результатом виконання цієї роботи став алгоритм побудови карти висот планети, з можливістю вибору величини октави шуму, накладання різних октав одне на одну для отримання більш реалістичних результатів та розмалюванням поверхні по висотам.

Алгоритм використовує структуру Quad-tree для збільшення резолюції поверхні поруч із спостерігачем [рис. 4.1](#Рисунок_4_1). Інша частина мешу, яка знаходиться далеко від спостерігача, залишається з більш простою поверхнею, що дозволяє зосередити ресурси комп'ютера на ділянці поверхні поруч із камерою спостерігача.

A screenshot of a video game

Description automatically generated with low confidence

Рисунок 4.1 – Карта висот на сфері із зображенням Quad-tree розподілу

Розмальовка поверхні виконується за критерієм дистанції точки поверхні від центра планети. Ми можемо виділити декілька точок висот, за допомогою яких буде виконуватись інтерполяція кольору. Наприклад, точка океану та точка континенту [рис 4.2](#Рисунок_4_2), [рис 4.3](#Рисунок_4_3).

A green and blue planet

Description automatically generated with low confidence

Рисунок 4.2 – Кольорове розділення поверхні на океан та континент.

A green and blue planet

Description automatically generated with low confidence

Рисунок 4.3 – Генерація поверхні окремими плоскостями, які не розділяють між собою вершини.

Управління генерацією планети виконується за допомогою компонента Sphere Builder [рис. 4.4](#Рисунок_4_4), у якому можна вибрати seed, найбільшу октаву шуму, кількість октав та інше.

A screenshot of a computer program

Description automatically generated with medium confidence

Рисунок 4.4 – Компонент для генерації планети

Згенерована планета з малою октавою шуму виглядає наступним чином. На [рис 4.5](#Рисунок_4_5) ми бачимо сам результат. Вибір кольору по висоті з двох кольорів: темно-зеленого, який відповідає за висоти близько рівня моря та сірого, який відповідає за скали. На [рис 4.6](#Рисунок_4_6) видно той самий ландшафт, але з демонстрацією Quad-tree.

A close-up of a ball

Description automatically generated with low confidence

Рисунок 4.5 – Ландшафт із шумом малої октави

A picture containing text, screenshot, pc game, 3d modeling

Description automatically generated

Рисунок 4.6 – Та сама поверхня, але з демонстрацією Quad-tree

Структура Quad-tree допомагає генерувати поверхню поступово, що дає можливість швидко генерувати більш детальні ділянки замість того, щоб генерувати усю поверхню зі збільшеною деталізацією. На [рис 4.7](#Рисунок_4_7) ми бачимо планету, згенеровану без використання Quad-tree, на [рис 4.8](#Рисунок_4_8) демонстрацію деталізації. Далі, на [рис 4.9](#Рисунок_4_9), ми бачимо відмінність поверхні при використанні Quad-tree структури. Чим темніша поверхня – тип щільніше один до одного генерувалися точки висоти. Тобто деталізація у таких ділянках найвища. Протиположна сторона планети генерується з мінімальною деталізацією, що видно з [рис 4.10](#Рисунок_4_10). Таке розподілення дозволяє генерувати дуже велику щільність висот поряд з камерою спостерігача.

A picture containing screenshot, 3d modeling, sphere

Description automatically generated

Рисунок 4.7 – Поверхня планети із однаковою деталізацією (без Quad-tree)

A picture containing screenshot

Description automatically generated

Рисунок 4.8 – Поверхня планети без Quad-tree з демонстрацією мешу

A picture containing screenshot, sphere

Description automatically generated

Рисунок 4.9 – Поверхня планети з використанням Quad-tree структури. Розмір однієї ділянки – 16х16 точок висоти

A picture containing sphere

Description automatically generated

Рисунок 4.10 – Протилежна до камери сторона планети.

Використовуючи цей підхід, ми можемо значно зменшити кількість точок висот у пам'яті, а зі збільшенням максимальної кількості поділів, різниця між статичною резолюцією та поділом по Quad-tree сильно збільшується. На [рис. 4.11](#Рисунок_4_11) ми бачимо два графіки, один з яких відповідає кількість точок при заповнені площини максимальною резолюцією, інший за використання Quad-tree. По горизонталі ми відміряємо кількість точок у одній площині максимальної деталізації.

A picture containing text, screenshot, line, plot

Description automatically generated

Рисунок 4.11 – Різниця між статичною резолюцією та Quad-tree

Із графіку видно, що для досягнення точності ділянки поверхні під камерою у 1024 точки, загальна кількість витрачених точок при використанні Quad-tree менше у 3.5 рази, ніж при рендеру усієї поверхні з однаковою точністю. При подальшому збільшенню точності поверхні, використання звичайного метода стає все більш неможливим, доки Quad-tree все ще зберігає робочий стан. Ефективність метода Quad-tree можна підвищити, змінивши критерій поділу поверхні. Але при цьому буде погіршуватись якість поділу.

# ВИСНОВКИ

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт Міністерства освіти та науки України: <http://mon.gov.ua/>
2. Закон України про вище освіту [Електронний ресурс] // Верховна Рада України: [сайт]- про вищу освіту Верховна рада України; Закон від 01.07.2014№1556-VII.- Режим доступу: [http://zakon2.rada.gov.ua/ laws/show/1156-18](http://zakon2.rada.gov.ua/%20laws/show/1156-18%20(дата%20з) (Дата звернення 03.04.2020).
3. <http://www.dnu.dp.ua/docs/obgovorennya/Polozhennya_Antiplagiat_2016.doc>
4. Хеш-функції - Режим доступу: [https://uk.wikipedia.org/wiki/Hash-Function](https://uk.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F#:~:text=%D0%A5%D0%B5%D1%88%2D%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F%2C%20%D0%B0%D0%B1%D0%BE%20%D0%B3%D0%B5%D1%88%2D,%D1%80%D0%BE%D0%B7%D0%BC%D1%96%D1%80%D1%83%20%D0%B2%20%D0%B4%D0%B0%D0%BD%D1%96%20%D1%84%D1%96%D0%BA%D1%81%D0%BE%D0%B2%D0%B0%D0%BD%D0%BE%D0%B3%D0%BE%20%D1%80%D0%BE%D0%B7%D0%BC%D1%96%D1%80%D1%83) - Hash Function
5. Шум Перліна - Режим доступу: <https://en.wikipedia.org/wiki/Perlin_noise> - Perlin noise
6. Шум симплекс - Режим доступу: <https://en.wikipedia.org/wiki/Simplex_noise> - Simplex noise
7. Процедурна генерація ландшафту - Режим доступу: <https://www.simondev.io/> - SimonDev
8. Псевдовипадкові шуми - Режим доступу: <https://catlikecoding.com/unity/tutorials/pseudorandom-noise/> - Unity Pseudorandom Noise Tutorials

# ДОДАТКИ

## Додаток А

## Додаток Б

## Додаток В