Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія генерування стилізованих 3D-ландшафтів»

Виконав: студент 2-го курсу, групи
1КН-23м спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Кушнір О. А.
(прізвище та ініціали)

Керівник: д. т. н., проф. каф. КН
Іванчук Я. В.
(прізвище та ініціали)
«05» 12 2024 р.

Опонент: д.т.н., проф. зав. каф. КСУ
Ковтун В. В.
(прізвище та ініціали)
«05» 12 2024 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«06» 12 2024 р.

Вінниця ВНТУ- 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
д.т.н., проф. Яровий А. А.
«11» 10 2024 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ

Кушніру Олександру Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Інформаційна технологія генерування стилізованих 3D-ландшафтів»
ерівник роботи д. т. н., проф., Іванчук Я. В.
тверджені наказом ВНТУ від «--17.09 2024 року № 310
2. Строк подання студентом роботи 11.11 2024 року
3. Вихідні дані до роботи: тип ландшафту – скелі, каньйони, гори, пагорби, дюни, луги, степи; ро-зміри ландшафту – 500 – 4000 м; інтенсивність шумів – 10 – 100 пунктів; тип шуму – Перлін, Вороний, Midpoint Displacement; кількість полігонів – низька, середня, висока; масштаб ландшафту – 1x, 2x, 5x; згладжування поверхні – 1- 100%.
4. Зміст текстової частини: вступ, аналіз предметної області, проєктування інформаційної технології генерування стилізованих 3D-ландшафтів, програмна реалізація системи генерування стилізованих 3D-ландшафтів, економічна частина, висновки, список використаних джерел, додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): вимоги до інформаційної системи генерування стилізованих 3D-ландшафтів загальна структура генератора ландшафтів; схема алгоритму інформаційної технології генерування стилізованих 3D-ландшафтів; UML-діаграма класів інформаційної технології генерування стилізованих 3D-ландшафтів; результати тестування розробленої інформаційної технології генерування стилізованих 3D-ландшафтів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Виконання прийняв
1-3	Іванчук Я.В., д. т. н., проф. каф. КН		
4	Лесько О.Й., д.е.н., проф. каф. ЕПВМ		

7. Дата видачі завдання 02.09 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	02.09.24 - 14.09.24	
2	Проєктування інформаційної технології генерування стилізованих 3D-ландшафтів	15.09.24 - 25.09.24	
3	Програмна реалізація системи генерування стилізованих 3D-ландшафтів	26.09.24 - 18.10.24	
4	Економічна частина	19.10.24 - 28.10.24	
5	Апробація та/або впровадження результатів дослідження	29.10.24 - 04.11.24	
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	05.11.24 - 11.11.24	

Студент _____ Кушнір О.А.
(підпис)

Керівник роботи _____ Іванчук Я.В.
(підпис)

АНОТАЦІЯ

УДК 004.92

Кушнір О. А. Інформаційна технологія генерування стилізованих 3D-ландшафтів. Магістерська кваліфікаційна робота зі спеціальності 122 «Комп'ютерні науки», освітня програма «Системи штучного інтелекту». Вінниця: ВНТУ, 2024. 126 с.

Укр. мовою. Бібліогр.: 22 назв; рис.: 31; табл. 9.

Дана магістерська кваліфікаційна робота присвячена розробці інформаційної технології генерування стилізованих 3D-ландшафтів. Проведено огляд сучасних методів процедурної генерації ландшафтів і підходів до створення стилізованих графічних об'єктів. Розроблено математичну модель і вдосконалений алгоритм генерації ландшафтів, який поєднує різні типи шумів (Перліна, Вороного, Midpoint Displacement) для створення гармонійних і виразних ландшафтів. Розроблено структуру інформаційної технології з інтерактивним інтерфейсом для вибору типів ландшафтів та їх поєднання. Реалізовано програмне забезпечення у середовищі Maya з використанням мови Python, обґрунтовано вибір середовища та інструментів розробки. Створено UML-діаграми для ключових модулів системи. Тестування підтвердило коректність роботи алгоритмів.

Графічна частина складається з 7 плакатів.

У економічному розділі оцінено та доведено ефективність виконання науково-дослідної роботи з високим науковим, технічним і економічним рівнями ($СБ_с = 38$). Визначено, що орієнтовна сума загальних витрат на проведення всіх етапів науково-дослідної роботи та оформлення її результатів складає 271926 грн.

Ключові слова: інформаційна технологія, генеруваання, 3D-ландшафти, шум Перліна, шум Вороного, стилізування, Maya.

ABSTRACT

Kushnir O.A. Information Technology for Generating Stylized 3D Landscapes. Master’s qualification thesis in the specialty 122 "Computer Science," educational program "Artificial Intelligence Systems." Vinnytsia: VNTU, 2024. 126 pages.

In Ukrainian. Bibliography: 22 sources; figures: 31; tables: 9.

This master’s qualification thesis is dedicated to developing information technology for generating stylized 3D landscapes. A review of modern procedural landscape generation methods and approaches to creating stylized graphical objects has been conducted. A mathematical model and an improved landscape generation algorithm have been developed, integrating various noise types (Perlin, Voronyi, and Midpoint Displacement) to create harmonious and expressive landscapes.

The information technology structure includes an interactive interface for selecting and combining different landscape types. The software was implemented in Maya using Python, with a rationale provided for the choice of development environment and tools. UML diagrams were created for the system’s key modules. Testing confirmed the correctness of the algorithms.

The graphical part consists of 7 posters.

The economic section evaluates and proves the efficiency of the research project, demonstrating high scientific, technical, and economic levels ($SB_c = 38$). The estimated total cost for all stages of the research and documentation amounts to 271926 UAH.

Keywords: information technology, generation, 3D landscapes, Perlin noise, Voronoi noise, stylization, Maya.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний світ технологій постійно розвивається, і однією з найбільш динамічних галузей є комп'ютерна графіка та 3D моделювання. Зокрема, велике значення набуває створення процедурних 3D-ландшафтів, які знаходять широке застосування в різних сферах: від ігрової індустрії до кіновиробництва та віртуальної реальності. З розвитком цих технологій з'являється потреба в нових підходах, які дозволяють швидко та ефективно створювати високоякісні тривимірні середовища з мінімальними затратами людських ресурсів. Процедурна генерація 3D-ландшафтів є одним із таких підходів.

Однією з ключових переваг процедурної генерації є можливість автоматизованого створення складних об'єктів і текстур на основі математичних алгоритмів. Це дозволяє не лише зменшити час на розробку, але й створювати унікальні, неповторні сцени кожного разу. Особливого значення це набуває в процесі створення стилізованих 3D-ландшафтів, що мають певний художній стиль або специфічну естетику. Така графіка активно використовується в анімаційних фільмах, мультфільмах та інді-іграх, де реалістичність не завжди є пріоритетом, а важливішим стає передача атмосфери та настрою [3].

Актуальність теми полягає в тому, що розвиток процедурних методів генерації 3D-ландшафтів значно полегшує роботу 3D-художників і дизайнерів, скорочуючи витрати часу на ручне моделювання складних об'єктів. Крім того, зростаючий попит на високоякісні стилізовані віртуальні середовища стимулює дослідження нових підходів до генерації ландшафтів, що відповідають певним стилістичним вимогам.

Зв’язок роботи з науковими програмами, планами, темами. Магістерська кваліфікаційна робота виконана відповідно до напряму наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Розробка прикладних інтелектуальних

інформаційних технологій та систем» та плану наукової та навчально-методичної роботи кафедри.

Метою дослідження є покращення візуалізації стилізованих рельєфів за допомогою розробки інформаційної технології генерування стилізованих 3D-ландшафтів, що дозволить створювати унікальні та якісні стилізовані ландшафти з використанням процедурної генерації в програмному середовищі. Такий підхід забезпечить можливість автоматизованого створення віртуальних середовищ для різних видів медіапроєктів, зокрема для анімації, комп'ютерних ігор та візуалізації.

Аби досягнути поставленої мети, потрібно сформувати та виконати наступні **завдання**:

– виконати аналіз предметної області та об’єкту дослідження, зокрема стилізованих 3D-ландшафтів та методів їх створення;

– провести аналіз існуючих методів процедурної генерації ландшафтів та визначити їх особливості і можливості для застосування у створенні стилізованих моделей;

– розробити модель інформаційної технології генерування стилізованих 3D-ландшафтів, що включає вибір алгоритмів та методів генерації рельєфу, текстурування та інтеграції різних типів ландшафтів;

– розробити алгоритм генерування стилізованих 3D-ландшафтів, який буде гнучким та налаштовуваним для створення різних видів ландшафтів (скелі, каньйони, гори, пагорби тощо);

– виконати програмну реалізацію алгоритму в середовищі Maya з використанням Python, створити інтерфейс для вибору типів ландшафтів та їх параметрів;

– провести тестування розробленої інформаційної технології для оцінки якості генерації стилізованих 3D-ландшафтів, зокрема візуальної привабливості та продуктивності генерації;

– проаналізувати результати тестування та розробити рекомендації щодо подальшого вдосконалення технології;

– провести розрахунок економічної ефективності розробки для можливого її застосування в індустрії комп’ютерної графіки та анімації.

Об’єкт дослідження – процес процедурної генерації 3D-ландшафтів.

Предмет дослідження – програмні засоби генерування стилізованих 3D-ландшафтів.

Методи дослідження – методи та підходи до розробки інформаційних технологій для процедурної генерації 3D-ландшафтів, зокрема методи використання шумів (Перліна, Вороного, Midpoint Displacement) для створення рельєфу, а також методи об’єктно-орієнтованого програмування для реалізації програмного забезпечення.

Наукова новизна одержаних результатів. Запропоновано вдосконалений метод генерування стилізованих 3D-ландшафтів, в якому, на відміну від існуючих, використовуються підхід до генерації тривимірних зображень поверхонь за рахунок використання поєднання шумів різних масштабів для досягнення природного та гармонійного вигляду стилізованих 3D-о’бєктів, що значно розширює можливості візуалізації.

Практичне значення одержаних результатів полягає у наступному:

– розроблено алгоритм функціонування інформаційної системи генерування стилізованих 3D-ландшафтів;

– розроблено програмний модуль створення стилізованих ландшафтів із можливістю вибору різних типів рельєфу (гори, пагорби, каньйони тощо) та їх поєднання, що спрощує процес налаштування та генерації налаштовуваних 3D-ландшафтів.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням математичних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими, та збіжністю результатів математичного моделювання з результатами, що отримані під час впровадження розроблених програмних засобів.

Особистий внесок магістранта. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно.

Апробація результатів роботи. Фактичні результати роботи були апробовані на такій конференції, як «LII Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)» [1].

Публікації. Результати магістерської дипломної роботи були представлені у вигляді тез на науково-технічних конференціях [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Обґрунтування доцільності створення інформаційної технології генерування стилізованих 3D-ландшафтів

Розвиток тривимірної графіки та зростання попиту на візуальні ефекти в різних галузях, таких як ігрова індустрія, кінематограф, архітектурна візуалізація та віртуальна реальність, створюють необхідність ефективних інструментів для створення 3D-середовищ. Однією з ключових складових таких середовищ є ландшафти, які можуть відображати різні типи природних або фантазійних просторів. Ландшафти відіграють важливу роль у створенні атмосфери, забезпечуючи глибину, масштаб і візуальну привабливість сцени. Вони допомагають гравцям, глядачам або користувачам зануритися у віртуальний світ і відчути себе частиною цієї реальності.

Традиційні методи створення тривимірних ландшафтів вимагають значних ресурсів, зокрема часу на ручне моделювання, текстурування та налаштування. Ці процеси є трудомісткими і потребують високої кваліфікації художників, що призводить до підвищення затрат на проектування та зниження продуктивності. В таких умовах процедурна генерація ландшафтів стає одним із найперспективніших рішень, оскільки дозволяє уникнути рутинних завдань і зосередитися на творчих аспектах розробки. Замість того, щоб вручну моделювати кожен елемент ландшафту, художники можуть використовувати алгоритми, які автоматично створюють складні та реалістичні сцени, зберігаючи художній контроль над параметрами генерації.
Процедурні методи генерації ландшафтів дозволяють автоматизувати процес створення складних тривимірних середовищ шляхом використання математичних алгоритмів, таких як шуми Перліна, Вороного, фрактали та інші. Це дозволяє отримувати унікальні, неручні геометричні форми, які можуть адаптуватися до заданих параметрів, таких як тип місцевості, висота пагорбів, структура поверхні тощо. Таким чином, користувач має можливість

швидко змінювати характеристики ландшафту відповідно до вимог проєкту, що значно підвищує гнучкість і продуктивність процесу розробки. Крім того, генерація ландшафтів на основі алгоритмів дозволяє забезпечити повторюваність результатів, що важливо для великих проєктів, де потрібно створити велику кількість схожих сцен або елементів [2].

Особливої актуальності ці методи набувають при створенні стилізованих ландшафтів, де важливими є не стільки фізична реалістичність, скільки художня естетика і виразність сцени. Стилізовані ландшафти використовуються для створення унікальних, казкових або фантазійних світів, які підкреслюють особливості графічного стилю проєкту та надають йому характерного вигляду. Створення таких ландшафтів вручну може бути дуже складним, оскільки необхідно враховувати різноманітні художні вимоги і забезпечити гармонію між всіма елементами сцени. Процедурна генерація дозволяє автоматизувати цей процес, забезпечуючи при цьому можливість контролю над стилістикою і деталями [3].

Стилізовані 3D-ландшафти широко використовуються в індустрії анімації, мультфільмів та інді-ігор, де дизайнери прагнуть досягти певного візуального стилю, що виходить за межі фотореалізму. У таких випадках важливо не тільки забезпечити унікальний вигляд кожної сцени, але й дотримуватися єдиної стилістичної концепції. Використання процедурної генерації дозволяє досягти необхідної стилізації з меншою кількістю ручної роботи, а також створювати великі і різноманітні середовища з мінімальними витратами часу. Це особливо важливо для проєктів, де час розробки є обмеженим, або для невеликих студій, які не мають ресурсів для масштабного моделювання вручну. Завдяки автоматизації процесу, команди можуть сфокусуватися на творчих аспектах проєкту, таких як дизайн персонажів, анімація та створення історії [4].

Ще одним важливим фактором є можливість інтеграції алгоритмів процедурної генерації в існуючі програмні продукти, такі як Autodesk Maya або Blender, що дозволяє художникам працювати у звичному для них

середовищі з додатковими інструментами для автоматизації процесів. Ця інтеграція не лише підвищує продуктивність роботи, але й знижує поріг входження для нових користувачів, оскільки їм не потрібно опановувати нові програми. Відомі програми для 3D-моделювання вже мають широкий набір інструментів для роботи з ландшафтами, і додавання процедурних методів дозволяє значно розширити їх можливості, забезпечуючи художникам і розробникам доступ до нових засобів автоматизації та генерації контенту.

Крім того, процедурна генерація ландшафтів забезпечує масштабованість процесу розробки. В умовах великих проєктів, таких як відкриті світи в іграх, потрібні великі території з різноманітними ландшафтами. Створення таких просторів вручну вимагало б значних ресурсів і часу, тоді як процедурні методи дозволяють генерувати величезні території автоматично, забезпечуючи при цьому необхідну деталізацію та унікальність кожної ділянки. Це робить можливим створення великих відкритих світів, які можуть адаптуватися під динамічні зміни в реальному часі, що є важливою характеристикою для ігор з відкритим світом або інтерактивних віртуальних середовищ [3].

Таким чином, доцільність створення інформаційної технології генерування стилізованих 3D-ландшафтів полягає в значному скороченні часу та ресурсів на створення складних тривимірних середовищ, підвищенні продуктивності розробників і художників, а також можливості автоматизації процесу з урахуванням стилістичних вимог. Це відкриває нові перспективи для індустрії тривимірної графіки та робить можливим створення якісних продуктів з меншими затратами. Крім того, процедурна генерація сприяє уніфікації стилю, масштабованості процесу та можливості швидкої адаптації до змінних вимог, що робить цей підхід надзвичайно привабливим для сучасних проєктів, особливо в контексті розвитку індустрії розваг і віртуальної реальності.

1.2 Аналіз відомих програмних рішень генерування стилізованих 3D-ландшафтів

Для обґрунтування доцільності розробки власної інформаційної технології генерування стилізованих 3D-ландшафтів необхідно проаналізувати існуючі програмні засоби, які вже використовуються в цій галузі. Серед відомих рішень, що пропонують можливість створення процедурних ландшафтів, можна виділити такі продукти, як World Machine, Gaea, Terragen, Vue, та інструменти для процедурного моделювання в популярних 3D-редакторах, таких як Blender та Autodesk Maya. Кожен із цих продуктів має свої сильні та слабкі сторони, а також особливості, які визначають його застосування у різних галузях [5].

World Machine є одним із найвідоміших інструментів для генерації реалістичних і стилізованих ландшафтів. Ця програма використовує процедурні алгоритми, зокрема шум Перліна та фрактальні алгоритми, що дозволяє створювати складні геометричні структури, такі як гори, річки, каньйони. World Machine пропонує гнучкий інтерфейс для налаштування різних параметрів ландшафту, що дозволяє користувачам створювати як реалістичні, так і стилізовані сцени. Однак основний недолік полягає в обмежених можливостях інтеграції з іншими програмами для 3D-моделювання, що може ускладнити робочий процес для художників, які працюють у комплексних середовищах розробки [6].

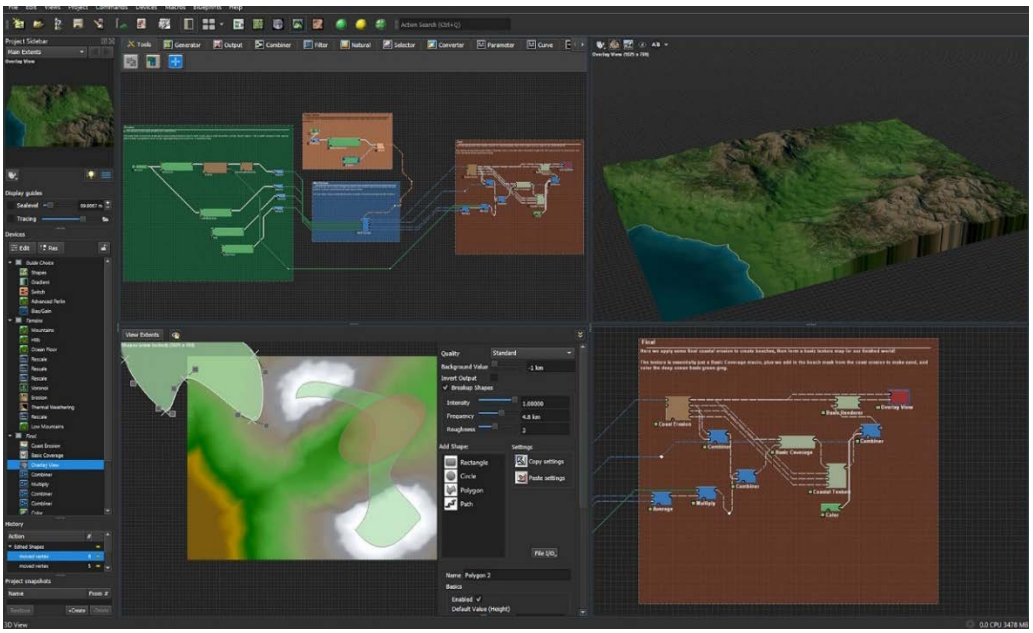

Рисунок 1.1 – Загальний вигляд нтерфейсного вікна World Machine

Gaea є більш сучасним інструментом, який також орієнтований на генерацію тривимірних ландшафтів. Gaea надає можливості для створення як реалістичних, так і стилізованих ландшафтів завдяки своїй потужній системі вузлів, яка дозволяє налаштовувати деталі генерації.

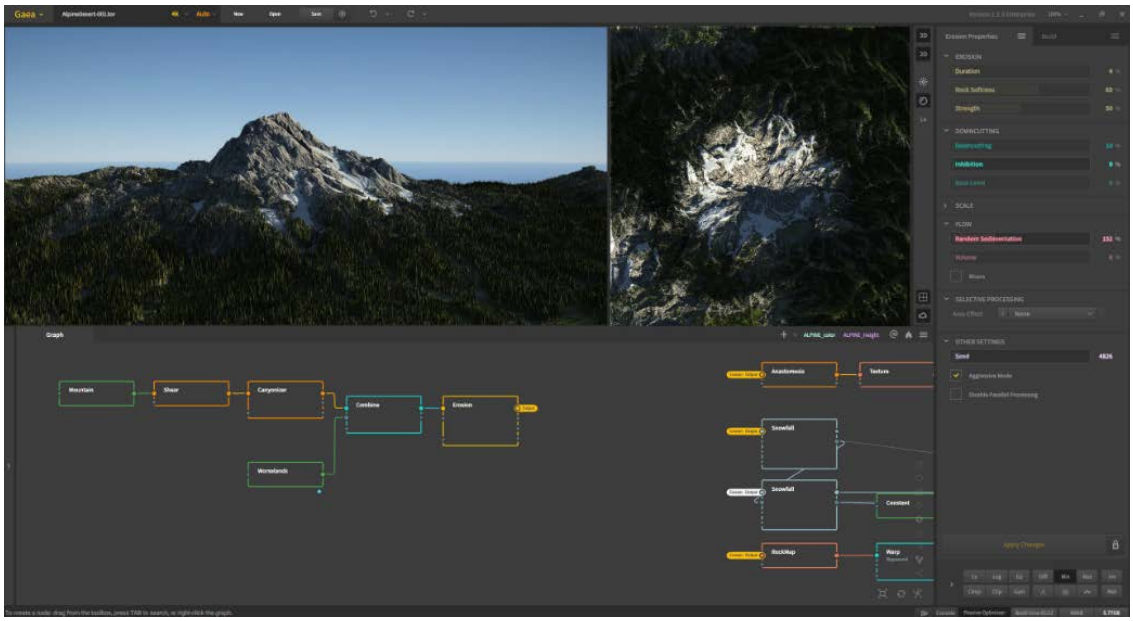

Рисунок 1.2 – Загальний вигляд нтерфейсного вікна Gaea

Terragen — це ще один популярний програмний засіб, орієнтований на створення фотореалістичних ландшафтів. Terragen використовується для створення як реальних, так і фантазійних пейзажів завдяки потужним інструментам для налаштування атмосфери, освітлення, води та інших елементів. Основною перевагою Terragen є його здатність створювати високоякісні зображення з великою кількістю деталей, що робить його популярним серед кінематографістів та художників, які працюють над візуальними ефектами. Проте, для стилізованих проєктів цей інструмент не завжди підходить, оскільки орієнтований переважно на реалістичність, і процес створення нестандартних стилізованих сцен може бути занадто складним [6].

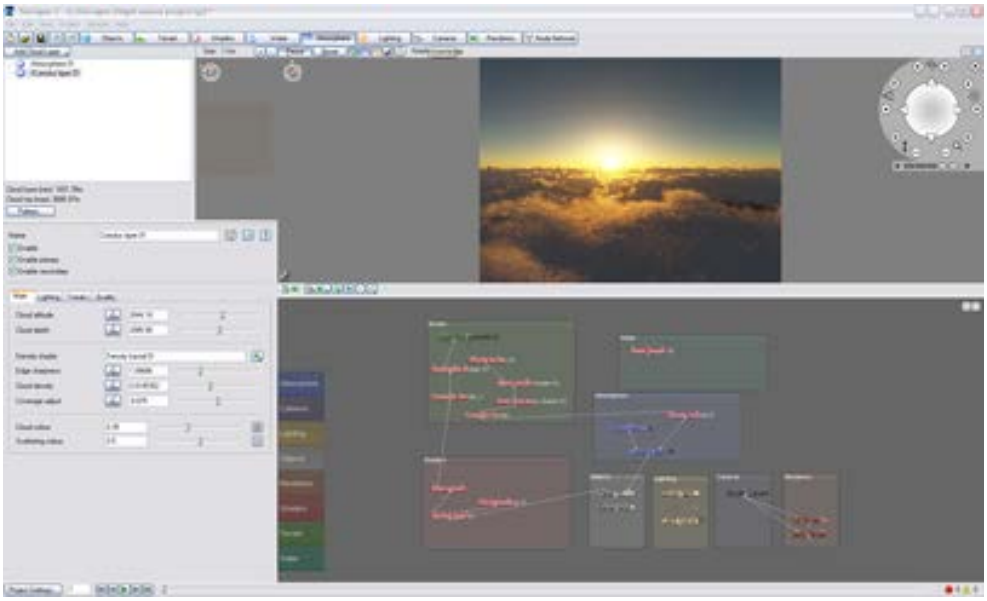

Рисунок 1.3 – Загальний вигляд нтерфейсного вікна Terragen

Vue є інструментом, орієнтованим на створення тривимірних ландшафтів та віртуальних світів, який також використовується в кінематографі та ігровій індустрії. Vue дозволяє створювати різноманітні типи ландшафтів, зокрема гори, ліси, океани, і пропонує потужні інструменти для налаштування атмосфери та освітлення. Програма надає можливість генерувати як реалістичні, так і стилізовані ландшафти, але має певні

обмеження у сфері інтеграції з іншими програмами та налаштування специфічних деталей. Крім того, використання Vue вимагає значних обчислювальних ресурсів, що може бути проблемою для невеликих студій або інді-розробників.

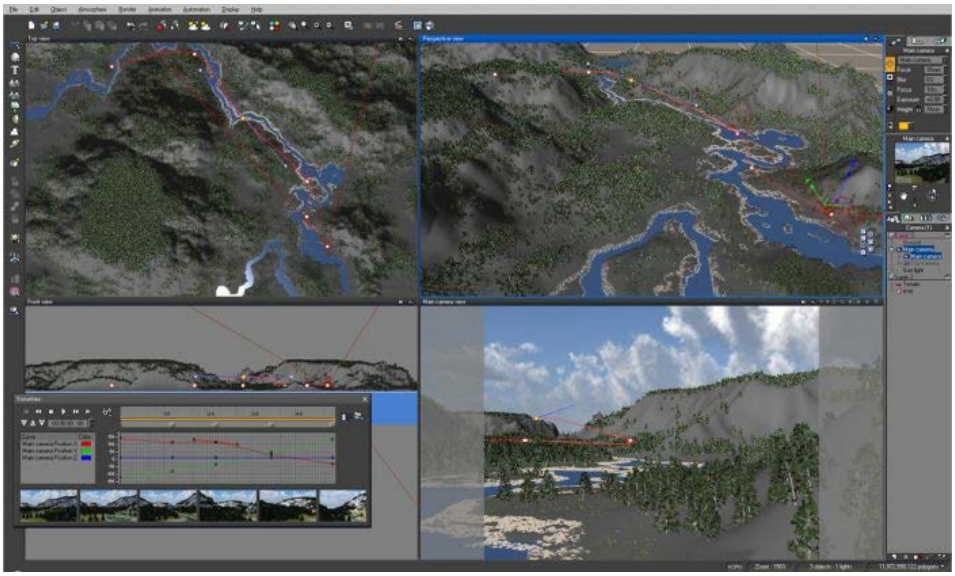

Рисунок 1.4 – Загальний вигляд нтерфейсного вікна Vue

Існують також інструменти для процедурного моделювання ландшафтів, які вбудовані в популярні 3D-редактори, такі як Blender та Autodesk Maya. Blender, завдяки своїй відкритій архітектурі та наявності великої кількості доповнень, пропонує інструменти для процедурного моделювання, такі як Add-on для шумів Перліна або генерації фрактальних ландшафтів. Ці інструменти дозволяють створювати стилізовані ландшафти з високим рівнем деталізації, а також забезпечують інтеграцію з іншими елементами сцени. Autodesk Maya також має інструменти для генерації ландшафтів і дозволяє використовувати скрипти для автоматизації процесу. Проте вбудовані інструменти мають обмежений функціонал у порівнянні з спеціалізованими програмами, такими як Gaea чи World Machine [6].

Аналізуючи відомі програмні засоби, можна зробити висновок, що кожен із них має свої переваги та недоліки, які визначають їх використання в різних галузях. Основними недоліками існуючих рішень є або надмірна

орієнтація на фотореалізм (як у Terragen чи Vue), або обмежені можливості інтеграції (як у World Machine). Крім того, багато з цих інструментів вимагають значних ресурсів та високого рівня технічної підготовки для повноцінного використання їх можливостей.

В умовах, коли потреба у створенні стилізованих ландшафтів зростає, а художники і розробники прагнуть автоматизувати процеси і знизити витрати на ручну роботу, існує необхідність розробки нового програмного засобу, який поєднував би можливості процедурної генерації, гнучкість налаштувань і зручність інтеграції з популярними програмами для 3D-моделювання. Такий підхід дозволить розширити можливості дизайнерів, підвищити продуктивність і забезпечити створення унікальних візуальних стилів, що відповідають сучасним вимогам ринку тривимірної графіки.

1.3 Аналіз методів отримання і представлення даних 3D-ландшафтів

Процес створення та представлення 3D-ландшафтів включає кілька етапів, кожен із яких вимагає використання різних методів і засобів. Основними складовими цього процесу є отримання даних про ландшафт, їх обробка і подальше представлення у вигляді тривимірної сцени. Аналіз методів і засобів, які використовуються для цього, дозволяє вибрати оптимальний підхід для розробки власного програмного забезпечення.

Методи отримання даних для 3D-ландшафтів включають кілька різних підходів. Один із найпоширеніших — це використання карт висот (heightmaps), які представляють собою двовимірні масиви даних, що зберігають висоту кожної точки ландшафту. Карта висот зазвичай зберігає значення висоти кожної точки у відтінках сірого кольору, де чим світліший піксель, тим вищою є відповідна точка поверхні, і навпаки. Коли карта висот застосовується до площини у тривимірному просторі, кожна точка площини піднімається або опускається залежно від значення висоти, яке задається на карті.

Це дозволяє швидко створювати різноманітні ландшафти, такі як гори, пагорби, долини і рівнини, просто використовуючи один двовимірний зображувальний файл. Карти висот можуть бути створені вручну за допомогою графічних редакторів, таких як Photoshop, або автоматично генеровані за допомогою процедурних алгоритмів, таких як шум Перліна або фрактальний шум [7].

Також карти висот можуть бути обмежені в деталізації, особливо якщо використовуються карти низької роздільної здатності. Це може призвести до появи артефактів або втрати реалістичності при наближенні до поверхні. Для уникнення цього використовуються додаткові карти, такі як карти нормалей або процедурне текстурування, які додають дрібні деталі без збільшення полігональної сітки.

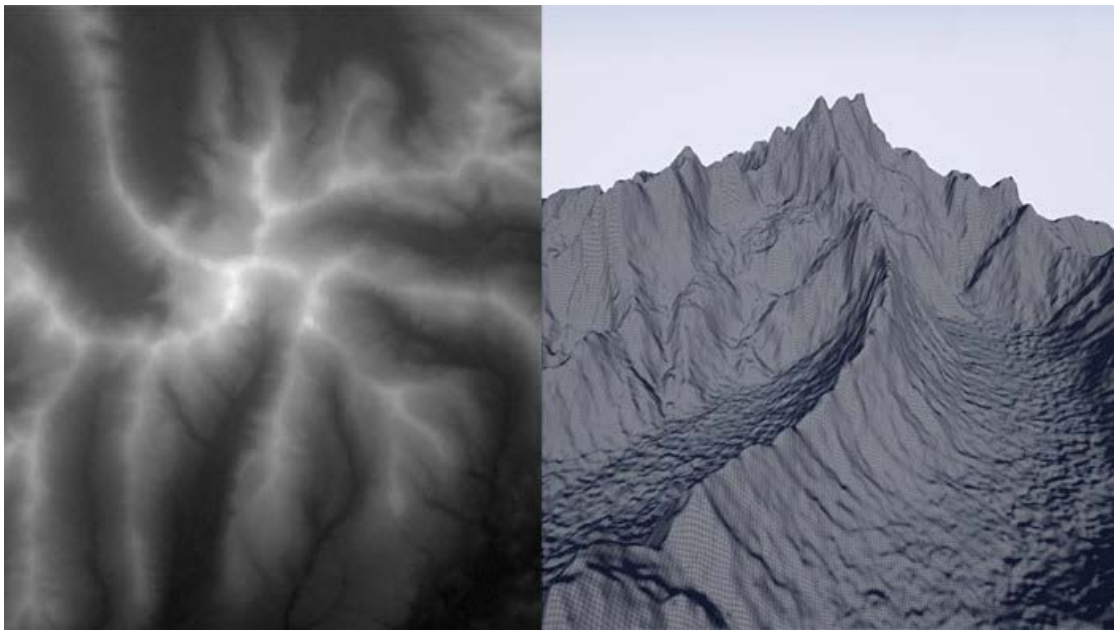

Рисунок 1.5 – Приклад генерації ландшафтів за допомогою карти висот

Процедурна генерація є ще одним методом отримання даних для 3D-ландшафтів, що набуває все більшої популярності. Вона широко застосовується для автоматизації процесів створення контенту, зокрема, в ігровій індустрії, кінематографі та архітектурній візуалізації. Основною ідеєю процедурної генерації є використання формул та алгоритмів для опису

властивостей об'єкта, що дозволяє генерувати складні і детальні структури з мінімальним залученням людини.

Однією з головних переваг процедурної генерації є її здатність створювати унікальний контент кожного разу при виконанні алгоритму. Завдяки цьому, навіть якщо два ландшафти будуть згенеровані за однаковими алгоритмами, вони матимуть відмінності, що додає різноманітності до тривимірного середовища. Це особливо важливо в ігровій індустрії, де важливо, щоб гравці отримували новий досвід під час повторних проходжень гри.

Крім того, процедурна генерація значно скорочує час розробки контенту, особливо для великих проєктів. Замість того, щоб моделювати кожен об'єкт вручну, розробники можуть використовувати алгоритми, які автоматично генерують потрібні елементи, економлячи ресурси і дозволяючи зосередитися на інших аспектах проєкту. Цей підхід також зменшує вимоги до зберігання даних, оскільки замість зберігання великих обсягів інформації про кожен об'єкт зберігаються лише алгоритми та початкові параметри, необхідні для його відтворення [8].

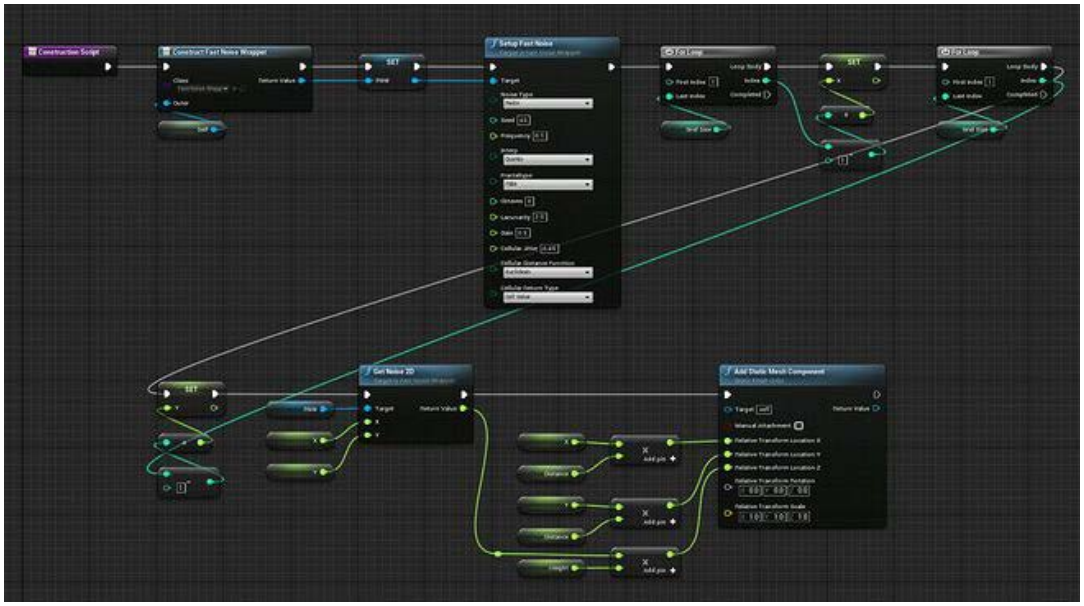

Рисунок 1.6 – Приклад генерації ландшафтів за допомогою процедурних методів

Фрактальні методи є одним із популярних підходів для генерації 3D-ландшафтів, що базується на використанні фрактальної геометрії. Основна ідея фракталів полягає в тому, що вони складаються з подібних один до одного частин, які повторюються на різних масштабах. Це властивість дозволяє створювати складні і детальні структури, які мають природний вигляд, наприклад, гірські масиви, долини або узбережжя.

Одним із найпоширеніших методів, який використовує фрактали, є метод розподілу середньої точки (midpoint displacement). Цей алгоритм починається зі створення лінії або площини та поступового розбиття її на менші частини з додаванням випадкових зміщень, що імітують природну нерівність. Завдяки повторенню цього процесу на кожному етапі генеруються деталі, що надають поверхні характерний "гірський" вигляд.

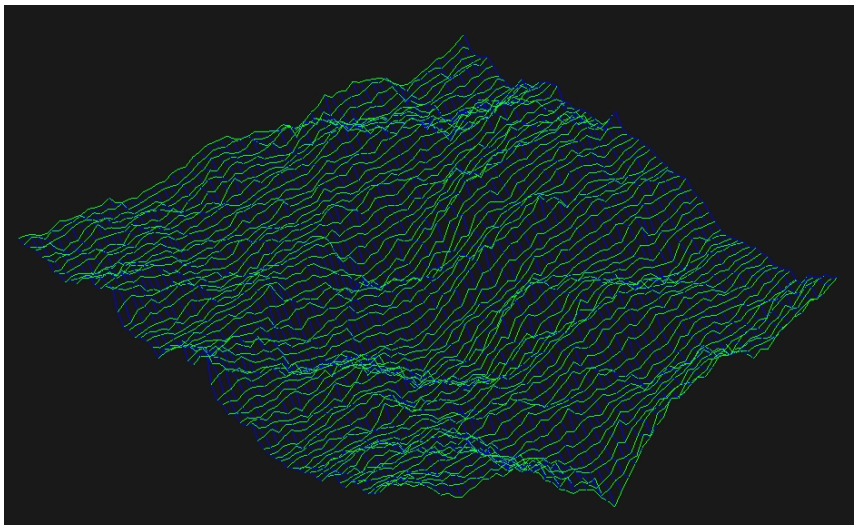

Рисунок 1.7 – Приклад генерації ландшафтів за допомогою фрактальних методів

Фотограмметрія є ще одним способом отримання даних для створення 3D-ландшафтів. Фотограмметрія — це метод отримання тривимірної інформації про об'єкт або середовище на основі аналізу фотографій, зроблених під різними кутами. Цей підхід дозволяє відтворювати точні 3D-моделі реальних об'єктів або ландшафтів, використовуючи звичайні цифрові камери,

що робить його доступним та ефективним інструментом для створення тривимірного контенту.

Основний принцип фотограмметрії полягає у використанні кількох зображень одного об'єкта, які перекриваються між собою. Алгоритми обробки зображень аналізують ключові точки та особливості на кожному знімку, визначаючи їхнє просторове розташування. Завдяки співставленню цих точок в різних кадрах, програма обчислює координати в тривимірному просторі, відтворюючи форму і текстуру об'єкта [8].

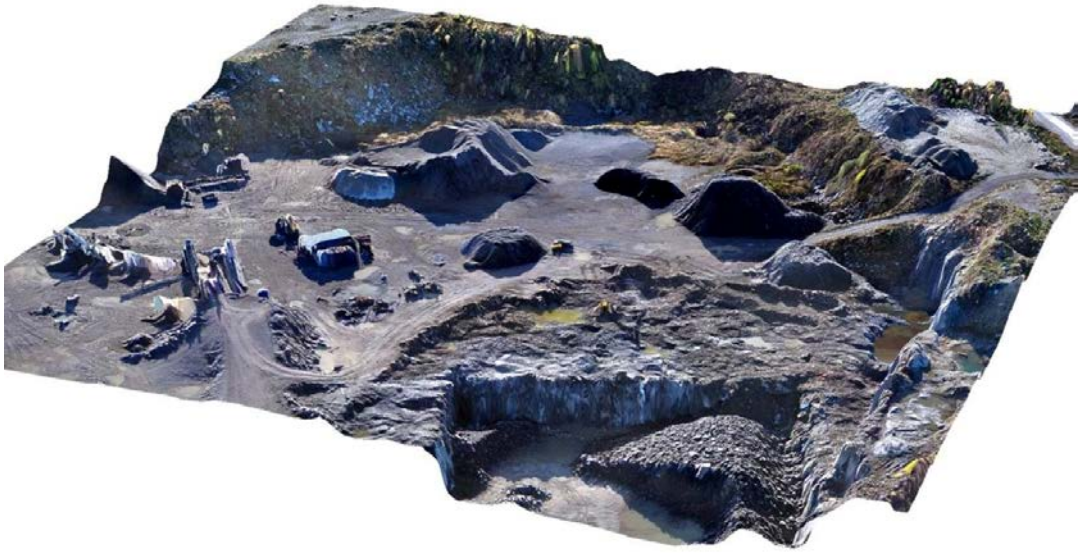

Рисунок 1.8 – Приклад генерації ландшафтів за допомогою фотограмметрії

Лідарні знімки (LiDAR) також використовуються для отримання високоточних даних про місцевість. Лідарні знімки (LiDAR) є методом отримання даних про об'єкти та поверхні за допомогою лазерного сканування. LiDAR (Light Detection and Ranging) використовує лазерний імпульс для визначення відстаней до поверхонь, дозволяючи створювати високоточні тривимірні моделі місцевості та об'єктів. Лідарні системи можуть бути встановлені на літальних апаратах (літаках, дронах), автомобілях або стаціонарних платформах, залежно від мети дослідження.

Принцип роботи LiDAR полягає у випромінюванні лазерного імпульсу, який відбивається від об'єкта і повертається до приймача. Система вимірює

час, що знадобився імпульсу на подолання відстані туди і назад, а також кут падіння променя. Використовуючи ці дані, обчислюються точні координати точки у тривимірному просторі. Велика кількість таких вимірювань дозволяє створити "хмару точок", яка представляє тривимірну форму об'єкта чи місцевості [9].

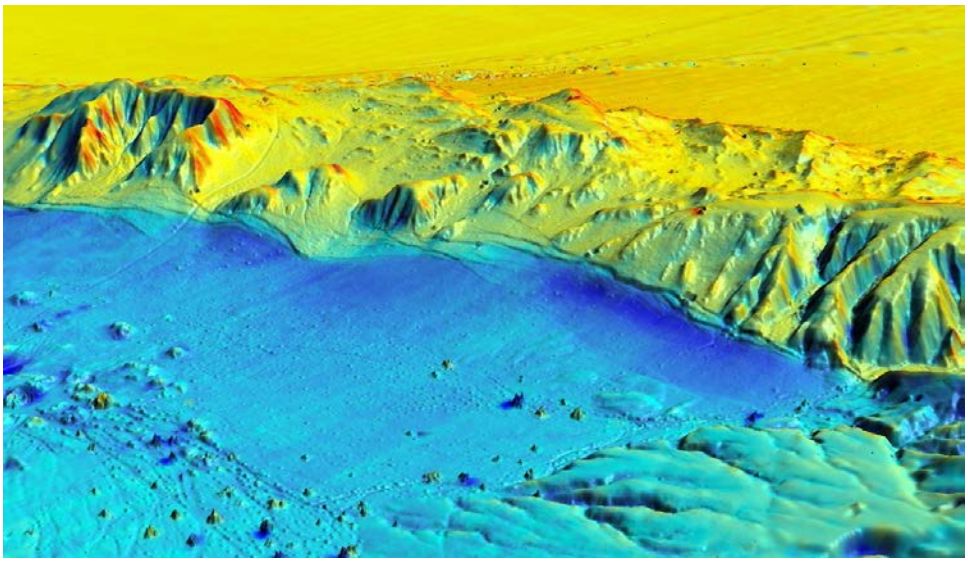

Рисунок 1.9 – Приклад генерації ландшафтів за допомогою LiDAR

Представлення даних 3D-ландшафтів здійснюється різними способами в залежності від застосування. Одним із найпоширеніших методів є використання полігональної сітки (mesh), яка представляє поверхню ландшафту як сукупність трикутників або інших багатокутників. Полігональні сітки використовуються у більшості програм для 3D-моделювання, таких як Blender, Autodesk Maya, і дозволяють легко застосовувати текстури, налаштовувати освітлення та проводити анімацію [10].

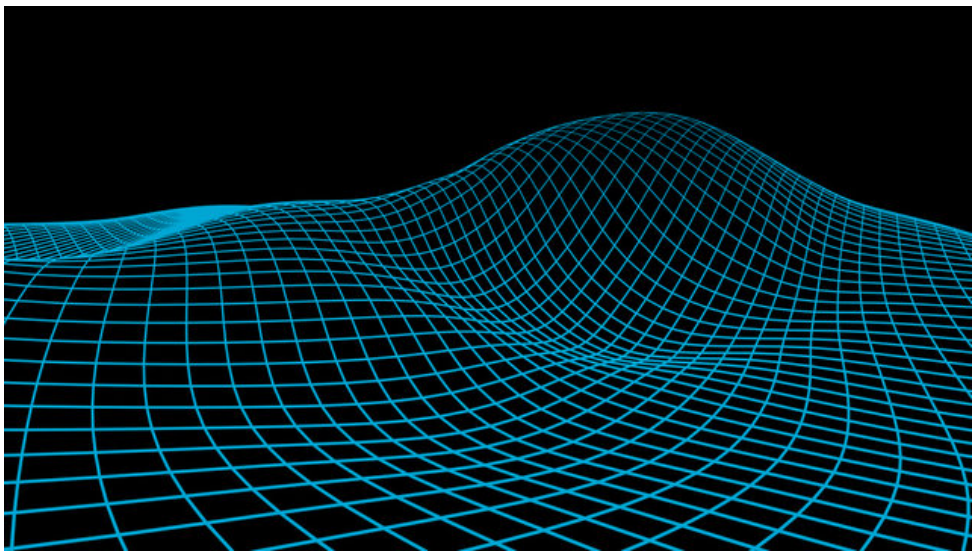

Рисунок 1.10 – Представлення 3D-ландшафтів за допомогою полігональної сітки

Іншим підходом до представлення даних є використання воксельних структур. Вокселі — це тривимірні аналоги пікселів, які представляють об'ємні елементи сцени. Воксельні методи часто застосовуються у випадках, коли необхідно моделювати внутрішню структуру ландшафту або забезпечити можливість руйнування та модифікації середовища в режимі реального часу, як це робиться у багатьох відеоіграх. Проте використання вокселів потребує значних обчислювальних ресурсів і великого обсягу пам'яті, що обмежує їх застосування для великих ландшафтів [11].

Рисунок 1.11 – Представлення 3D-ландшафтів за допомогою вокселів

Текстурування є важливою складовою представлення даних 3D-ландшафтів. Для цього використовуються карти текстур, такі як карти нормалей, карти висот, альбедо та інші, які накладаються на полігональну сітку, щоб надати поверхні ландшафту потрібного вигляду. Карти нормалей відповідають за деталізацію поверхні, додаючи дрібні нерівності та текстурні деталі без необхідності додавати додаткові полігони, що зменшує обчислювальне навантаження. Карти висот дозволяють змінювати геометрію поверхні, створюючи більш виразні форми, такі як виступи або вм'ятини, що значно покращує загальну візуальну складність моделі. Альбедо-карта визначає базове забарвлення поверхні без урахування світлових ефектів, що допомагає створити відповідний колірний тон для ландшафту [12].

Тіньове картографування (Shadow Mapping) і Ambient Occlusion є додатковими техніками, що використовуються для покращення візуальної якості 3D-ландшафтів. Тіньове картографування дозволяє відтворювати динамічні тіні, що підвищує реалістичність сцени, оскільки додає об'єктам відчуття фізичної присутності в просторі.

Таким чином, аналіз методів і засобів отримання та представлення даних 3D-ландшафтів показує, що існує широкий спектр підходів, які можна застосовувати в залежності від поставлених завдань. Процедурна генерація, фрактальні алгоритми, фотограмметрія і лідар забезпечують різні можливості для отримання даних, тоді як полігональні сітки, воксельні структури і текстурування дозволяють ефективно представляти ці дані у вигляді тривимірних сцен. Вибір конкретного методу залежить від вимог до деталізації, реалістичності, продуктивності та доступних ресурсів, що робить важливим комбінування кількох підходів для досягнення оптимальних результатів [13].

1.4 Висновок до розділу 1

У першому розділі було розглянуто та проаналізовано сферу створення стилізованих 3D-ландшафтів, а також досліджено об'єкт проектування. Було вивчено існуючі методи та підходи процедурної генерації ландшафтів, а також їхні ключові характеристики та недоліки. На основі аналізу виявлено, що існуючі технології здебільшого орієнтовані на створення реалістичних ландшафтів, що не завжди підходить для стилізованих мультяшних проектів.

Ураховуючи актуальність стилізованих рішень у сфері 3D-графіки, розроблений підхід до генерації таких ландшафтів може знайти широке застосування серед 3D-художників та розробників мультимедійного контенту.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ГЕНЕРУВАННЯ СТИЛІЗОВАНИХ 3D-ЛАНДШАФТІВ

2.1 Розробка методу генерування стилізованих 3D-ландшафтів

У цій роботі було розроблено метод для генерації стилізованих 3D-ландшафтів у середовищі Maya з використанням мови програмування Python. Метод базується на процедурній генерації, що дозволяє створювати різноманітні типи ландшафтів з мінімальним втручанням користувача. Процедурний підхід забезпечує можливість автоматичного створення ландшафтів із широким спектром параметрів, таких як форма, текстура та деталізація рельєфу. Створений метод має низку ключових особливостей, які відрізняють його від інших існуючих рішень.

Процедурна генерація ландшафтів є одним із ключових аспектів сучасного тривимірного моделювання, що забезпечує автоматичне створення складних рельєфів з мінімальними втручаннями з боку користувача. Цей метод надає можливість контролювати різноманітні параметри ландшафту, що значно розширює варіанти та гнучкість у створенні індивідуальних 3D-сцен. Основними вхідними параметрами, які використовуються для регулювання генерації, є масштаб ландшафту, тип шуму для створення текстур і рельєфу, а також кількість полігонів, яка впливає на деталізацію моделі. Ці параметри дозволяють налаштувати ландшафт як на макрорівні (наприклад, висота і форма гір), так і на мікрорівні (наприклад, текстура поверхні або дрібні деталі рельєфу) [8].

Масштаб ландшафту є критичним параметром, що визначає загальні пропорції та розміри об'єкта. З його допомогою можна створювати як невеликі пагорби чи горбки, так і величезні гірські хребти або каньйони. Завдяки можливості варіювати масштаб, художник чи дизайнер може легко адаптувати створюваний ландшафт під потреби конкретної сцени або проекту. Це

особливо корисно при роботі з великими сценами, де важливо, щоб всі об'єкти мали відповідні пропорції.

Тип шуму є ще одним важливим параметром, який значною мірою визначає форму і текстуру рельєфу. Шумові функції, такі як шум Перліна, Вороний або Midpoint Displacement, генерують складні поверхні, що додають ландшафту природної нерівності та деталізації. Кожен тип шуму має свої унікальні властивості, що дозволяє генерувати рельєфи з різними характеристиками. Наприклад, шум Перліна створює плавні, природні лінії, які чудово підходять для створення пагорбів або долин, тоді як шум Вороний генерує більш чіткі, розділені ділянки, що надає рельєфу вигляду скель або розломів. Використання таких шумів дозволяє створити реалістичний і водночас стилізований рельєф, відповідний до вимог проекту [14].

Рисунок 2.1 - Модель інформаційної системи генерування стилізованих 3D-ландшафтів

У процесі дослідження було обрано кілька методів процедурної генерації, які забезпечують гнучкість і високу якість результатів. Серед них варто виділити шум Перліна, шум Вороного і алгоритм Midpoint Displacement.

Шум Перліна дозволяє генерувати гладкі і органічні форми, що добре підходять для створення таких елементів, як пагорби і м'які ландшафтні контури. Його особливість полягає в плавних переходах між значеннями, що забезпечує природний вигляд згенерованих поверхонь.

Шум Перліна був створений в 1983 році науковцем Перліном Кеном, та названий в його честь, також іноді його називають класичним шумом Перліна, адже після нього було створено ще декілька його різновидів [8]. Цей алгоритм являється математичним алгоритмом, який в першу чергу був призначений до генерування процедурної текстури. Приклад генеруваня такої текстури ми можемо побачити на рисунок 2.2.

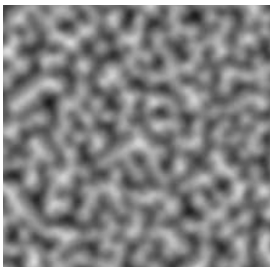

Рисунок 2.2 - Двомірний зріз тривимірного різновиду класичного шуму Перліна

Процес генерування шуму відбувається за допомогою генерування псевдо-рандомних чисел. Генерування псевдо-рандомних чисел це певний алгоритм, в результаті роботи якого ми отримуємо послідовність чисел які являються майже не залежними одне від одного, а також відповідають заданому розподіленню, частіше за все це рівномірне розподілення [11].

Класичний шум Перліна – це так званий градієнтний шум, який складається з набору псевдо-рандомних одиничних векторів, які знаходяться у певних точках в пространстві, та використовується для підвищення реалізму та графічної складності на поверхнях різноманітних геометричних об’єктів

[10]. На рисунку 2.3 ми можемо спостерігати перший етап класичного шуму Перліна, а саме генерацію псевдо-випадкових одиничних векторів, які представлені в якості точок на графіку [13].

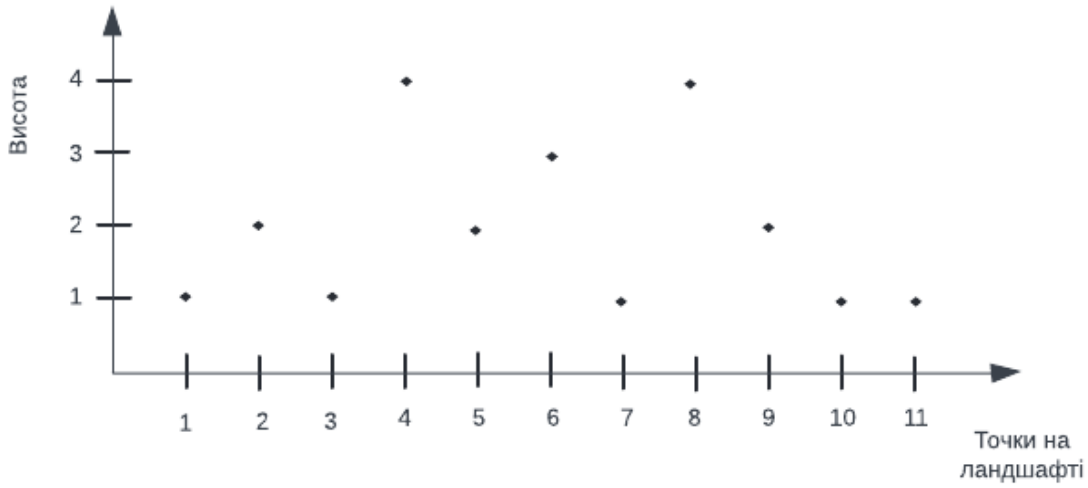

Рисунок 2.3 - Генерація псевдо-випадкових векторів класичного шуму Перліна

Наступний етап це інтерполяція отриманих значені за допомогою функції сглажуваня по цим точкам на графіці. Для того щоб згенерувати класичний шум Перліна в одномірному просторі ми повинні взяти кожну згенеровану точку та вирахувати для неї значення шумової хвилі, в цьому нам допоможе напрямок градієнта за яким була побудована точка. На рисунку 2.4 ми можемо спостерігати результат такої інтерполяції побудованої за даним рисунку 2.3.

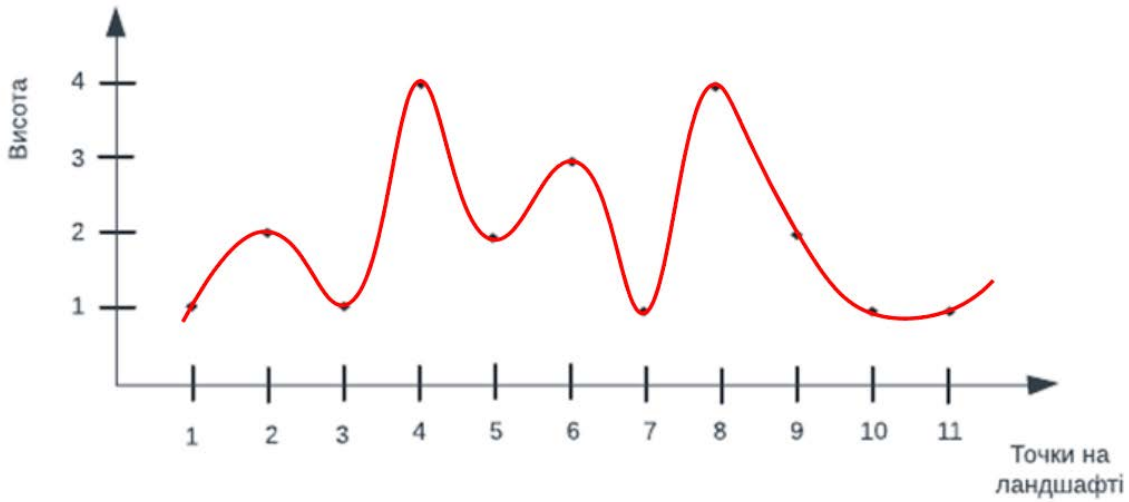

Рисунок 2.4 - Інтерполяція згенерованних данних класичного шуму Перліна

Для подальшого генерування класичного шуму Перліна використовуються такі базові математичні поняття для хвиль на графіку як амплітуда, довжина хвилі та її частота. Довжиною хвилі називають відстань від однієї вершини до наступної сусідньої з нею. А частота вираховується за формулою (2.1) як відношення одиниці до довжини хвилі [13].

$$frequency = 1 / wavelength, \quad (2.1)$$

де *frequency* – частота; *wavelength* – довжина хвилі.

Наступним етапом для створення незвичайних та більш різноманітних є створення декількох шумів з різними амплітудою та частотою. Після того як такі шуми готові, ми можемо об’єднати їх в один шум для отримання надзвичайного результату (рис. 2.5).

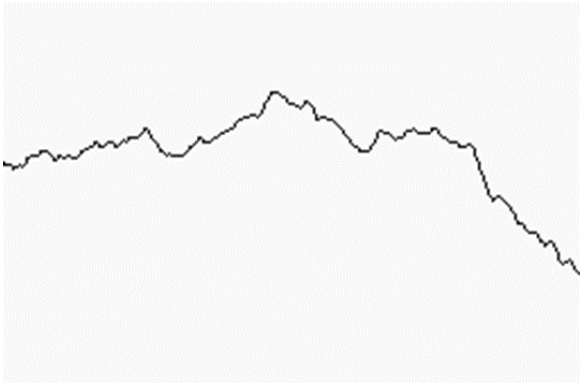

Рисунок 2.5 - Класичний шум Перліна отриманий в результаті об’єднання одиничних шумів

Для об’єднання наших шумів в один класичний шум Перліна виконується операція складання всіх отриманих класичних шумів Перліна. Але при цьому виникає складність у розрахунку частоти та амплітуди. Для таких розрахунків була введена нова величина яка називається стійкість, або Persistence. Ця величина вираховується як різниця у зміні розміру, нашого поточного шуму та наступного. Також змінюються розрахунки частоти та амплітуди за формулами (2.2-2.3) [13]:

$$frequency = 2^i, \tag{2.2}$$

де i – і-та додана функція шума; *frequency* – частота.

$$amplitude = persistence^i \tag{2.3}$$

де i – і-та додана функція шуму; *amplitude* – амплітуда; *persistence* – стійкість.

Що стосується окремих шумів які складаються в один шум, такі шуми називаються октавами. Така назва пішла з музики, адже так саме як і там кожен наступний шум має частоту вдвічі більше того, що йшов перед ним. Можна додавати скільки завгодно октав до класичного шуму Перліна, але є декілька нюансів які слід враховувати при цих діях.

По перше бувають такі моменти коли скомбінований шум досягає дуже великої частоти, через це в нас просто не вистачить пікселів на екрані для точного відображення всього шуму, а саме його дрібних деталей. По друге бажано вже не додавати багато шумових хвиль, амплітуда яких занадто мала, такі октави просто вже не зможуть оказувати достатнього впливу на кінечний результат.

Це робить шум Перліна основним компонентом для створення базових форм ландшафту, таких як пагорби та височини, що додають сценам відчуття простору і глибини.

Шум Вороного використовується для додавання детальності і розмаїття у вигляді поверхні, створюючи нерівності та текстуровані області, що дозволяє отримувати більш складні структури, наприклад, скелі, гори або каньйони [14].

Діаграма Вороного була винайдена у 1908 році Георгієм Феодосовичем Вороним, після того як він дослідив загальний n-мерний випадок цього явища. Діаграма Вороного складається с певної множини точок, ця множина є кінцевою, а точки розміщенні на поверхні у дозвільному порядку та місцях (рис. 2.6).

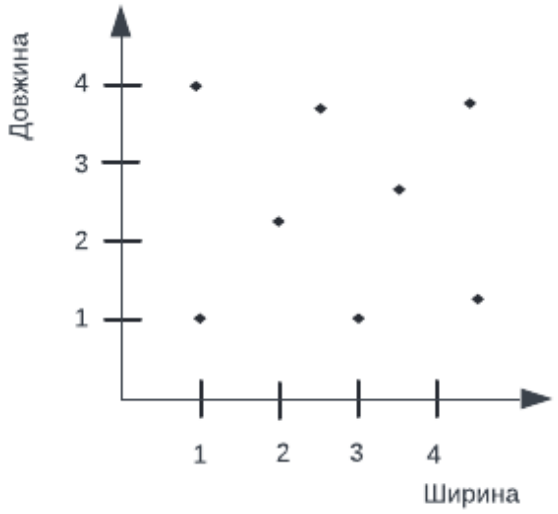

Рисунок 2.6 - Дозвільно розміщені точки до розбиття на Діаграму Вороного

Наступним кором є розбиття цієї області на зони навколо кожною з точок. Таке розбиття відбувається таким чином щоб кожна область створювала множну точок, які в свою чергу будуть більш близькими до одного з елементів множини наших основних точок на поверхні, ніж до будь якого іншого елементу цієї самої множини. Виходячи з цього ми можемо бачити на рисунку 2.7 результат розбиття множини точок на області [14].

Існують декілька способів обчислення діаграми виходячи з побажань стосовно результуючою діаграми.

Рисунок 2.7 - Розбиття множини точок та отримання Діаграми Вороного

У найпростішому випадку, який ми продемонстрували на першому малюнку, нам дано множину точок $\{p_1, \ldots, p_n\}$ яка є скінченною та знаходиться на евклідовій площині. У цьому випадку кожна ділянка pk є просто звичайною точкою, а її відповідна клітинка Вороного Rk розраховується та складається з кожної точки евклідової площини на якій вона знаходиться, відстань від якої до pk менша або дорівнює її ж відстані до будь-якого іншого pk на цій площині. Кожна така область виходить з розрахунку перетину півпросторів, а отже, виходячи з цього ми отримуємо опуклий багатогранник [14].

Формула (2.4) представляє розрахунок методом перпендикуляру:

$$V_p = \cap_{q \epsilon S/\{p\}} H_{pq} \quad (2.4)$$

де V – область Вороного; p та q – точки з кінцевої множини; S – кінцева множина точок; H – нова напівплощина.

Виходячи з цих розрахунків ми отримуємо обчислювальну складність для цього алгоритму яка становить $O(n4)$.

Наступний алгоритм – це рекурсивний алгоритм. Його основна ідея полягає у використанні метода динамічного програмування. Суть цього метода полягає у розбитті комплексної задачі на декілька під задач для більш зручних обчислень [14].

У нашому випадку ми маємо множину точок S на площині. Ця множина точок розбивається на дві підмножини S1 та S2. Для кожної з підмножин будується свої діаграма Вороного. Об'єднання двух діаграм Вороного складаючихся з множин S1 та S2 може бути виконано за час $O(n)$. Виходячи з цього ми можемо отримати обчислювальну складність алгоритму, яка в свою чергу становитиме $O(n \log n)$.

Також отриманний результат, а саме область Вороного описуємо відрізками залежить від обраної системи підрахунку відстані між точками на площині. Існує два типи відстані які ми можемо застосовувати для визначення областей Вороного [14]. Перший це евклідова відстань. Цю відстань можна обчислити з декартової системи координат, за допомогою теореми Піфагора. Отже ми маємо розрахувати її як корень квадратний сум різниць у другій степені відповідних координат за формулою (2.5):

$$L_2 = d[(a_1,a_2),(b_1,b_2)] = \sqrt{(a_1-b_1)^2+(a_2-b_2)^2} \quad (2.5)$$

де l – відстань між двома точками; d – відрізок між двома точками; a та b – точки на площині.

Через це можна підібрати такий алгоритм, який найбільше підійде до розбиття на області які ми бажаємо побачити на нашому ландшафті. Але так

само як класичний шум Перліна, діаграма Вороного не підходить для самостійного використання у формуванні ландшафту, адже навіть згенеровані таким чином гори будуть виглядати не натуральними.

Алгоритм Midpoint Displacement додає випадковість і варіативність у процес генерації, що дозволяє уникнути зайвої симетрії та повторюваності, надаючи ландшафтам природності і багатошаровості.

Алгоритм зміщення середньої точки вперше був використаним у 1982 році під час конференції siggraph, яка спеціалізується на обчислювальних методах комп'ютерної графіки, вченими Фурн'є, Фусселєм та Карпентером. Також такий алгоритм ще називають алгоритмом Diamond-Square через його особливості обчислювання [15].

Алгоритм зміщення середньої точки застосовується для змін значення мапи висот, з якої власно і складається ландшафт в ігрових двигунах. Мапа висот представляє собою двомірну матрицю яка визначає власне висоту певних точок нашого ландшафту. Беручи до уваги цю інформацію алгоритм зміщення середньої точки працює з двомірною сіткою [16].

Першим кроком ми маємо вибрати чотири початкові значення з нашої мапи висот, найчастіше беруться углові точки мапи висот, якщо ми хочемо охопити усю нашу мапу для процедурного генерування (рис. 2.8). Після того як ми обрали точки їм призначається початкова висота яка генерується випадковим чином [16].

Далі робота алгоритму зміщення середньої точки ділиться на два основні етапи, з застосуванням двох алгоритмів:

– метод діамантового кроку;
– метод квадратного кроку.

Власне через ці дві алгоритми алгоритм зміщення середньої точки і називають по іншому алгоритм Diamond-Square. Завдяки комбінації цих двох алгоритмів відкривається можливість для дуже успішної генерації різноманітних ландшафтів.

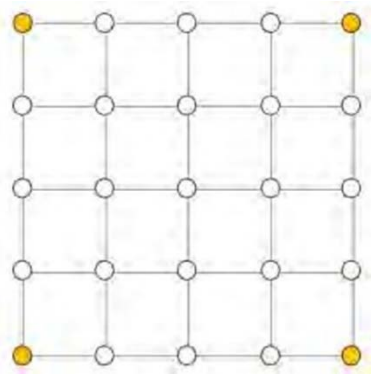

Рисунок 2.8 - Перший крок, обирання кутових точок

Наступним кроком після обирання кутових точок являється застосування одного з двох алгоритмів, а саме алгоритмау діамантового кроку. Суть застосування такого алгоритму полягає у знаходженні серединної точки, яка знаходиться між обраними чотирма. Такі розрахунки полягає на розрахунках середнього від кутових точок. Після цього серединній точці присвоюється випадково згенероване значення, яке визначає її висоту. Поглянувши на рисунок 2.9 можна побачити чому саме цей алгоритм називається діамантовим кроком, якщо ми намалюємо стрілочки, від кутових точок до їх серединної точки, то отримаємо картинку яка віддалено нагадує діамант [16].

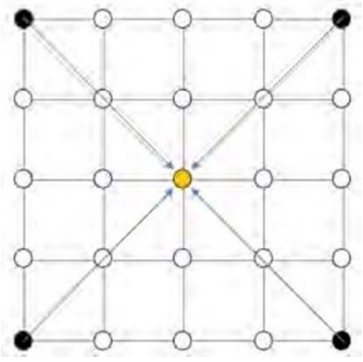

Рисунок 2.9 - Зображення виконання діамантового кроку, де чорні точки це кутові точки для яких розраховується серединна жовта точка

Після розрахунку серединної точки ми умовно отримали на нашій мапі висот один великий квадрат, який охоплює усю мапу, та складається с

чотирьох кутових точок та однієї серединної. Тепер ми повинні застосувати метод квадратного кроку. Суть цього метода полягає у подальшому розбитті нашої мапи висот на менші квадрати для більш детальної генерації ландшафту. Для цього ми беремо нашу отриману серединну точку, та знаходимо нові серединні точки для кожної пари кутових точок. У цьому випадку ми уявляємо що кожна пара кутових точок створює діамант з серединною точку, а четверта точка такого діаманта є уявною та знаходиться за межами нашої мапи висот [16].

Для знаходження нових серединних точок в квадратному кроці ми застосовуємо стандартний алгоритм, який використовували при розрахунках діамантового кроку. Після визначення нових серединних точок ми додаємо до них випадково згенеровану величину яка відповідає їх висоті. З рисунка 2.10 ми можемо побачити як за допомогою умовних стрілочок проходить розбиття на нові квадрати.

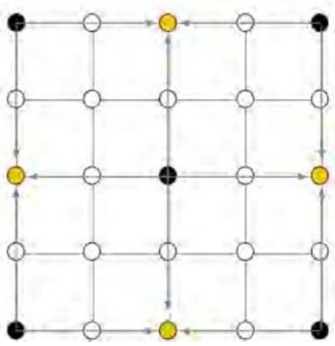

Рисунок 2.10 - Зображення роботи методу квадратного кроку, де чорними точками зображені точки отримані у результаті діамантового кроку, а жовтими – нові серединні точки які розбивають на квадрати

Результатом виконання методу квадратного кроку ми отримуємо мапу висот яка розбита вже на менші квадрати. Якщо на етапі діамантового кроку ми мали один великий квадрат, то на етапі квадратного кроку ми вже отримуємо чотири квадрати, які додають більшої деталізації.

Алгоритм продовжує свою роботи поки не розіб’є всю мапу висот на менші квадрати, але таке розбиття може регулювати з метою додати більше, або меншої точності до генерації з метою оптимізації, або натуральності сгенерованої мапи. Результат пошагової роботи розбивання мапи, яка розглядалася раніше ми можемо спостерігати на рисунках 2.11 та 2.12.

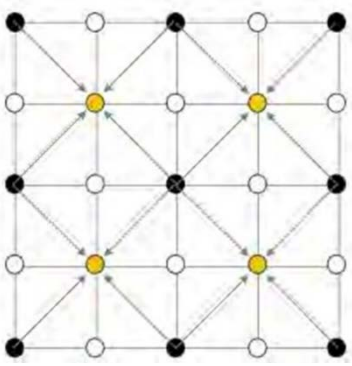

Рисунок 2.11 - Зображення розрахунку нових серединних точок для чотирьох отриманих квадратів під час квадратного кроку

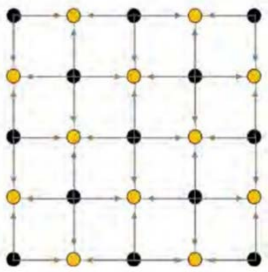

Рисунок 2.12 - Розбиття залишків мапи висот на квадрати методом квадратного кроку

Таким чином ми бачимо що кількість квадратів які будуть участувати під час процедурної генерації збільшилася в чотири рази, тобто ми отримали шістнадцять квадратів. З цього можна зробити висновок, що ми можемо вивести кількість ітерацій алгоритму за формулою (2.6):

$$n = squares\ count\ /\ 4, \tag{2.6}$$

де n – кількість ітерацій алгоритму, *squarescount* – кількість квадратів.

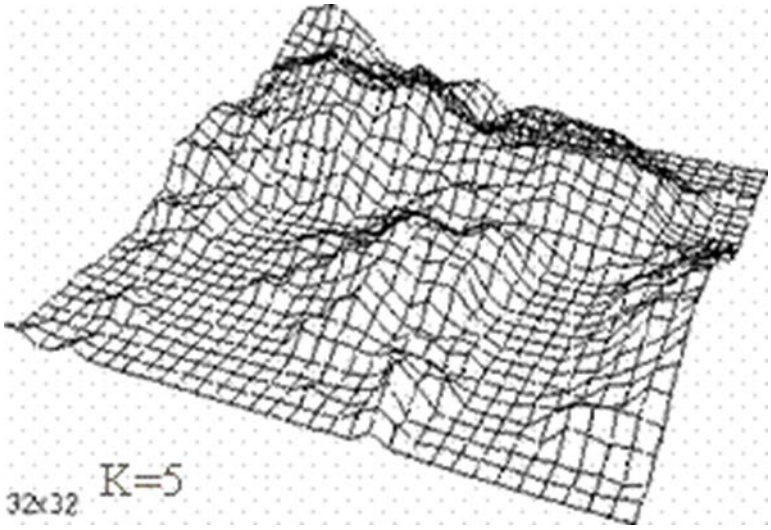

Рисунок 2.13 - Генерація висот результатом якого є ландшафт розмірністю 32x32 квадрати, та ітерацією *K*

З рисунку 2.13 ми можемо побачити, як кількість ітерацій впливає на деталізацію та унікальність ландшафту, чим вище ітерація, тим більш детальний ландшафт ми отримаємо. Або іншими словами ми отримаємо більш високі піки с крутими схилами, а долини становляться глибшими, через нестачу визначених вершин на мапі висот. Ми можемо порахувати зміщення на кожній ітерації алгоритму за формулою (2.7) [16]:

$$Displacement\ amount = average + (\frac{1}{2})^{H} * random, \tag{2.7}$$

де *Displacement amount* – параметр зміщення; *average* – середнє значення прилягаючих кутів; *random* – рандомне число яке визначає висоту; *H* – значення нерівності ландшафту.

Однією з основних величин які впливають на генерацію висот є параметр нерівності. Параметр нерівності це фактор за допомогою якого визначається з якою силою буде зменшуватися збурення на кожній ітерації алгоритму. Найчастіше стандартним значенням для нерівності використовують двійку, але це значення можна змінювати як забажає

розробник. Отже як саме воно впливає на зменшення збурення. Чим вище ми задаємо значення нерівності, тим більш рівним та гладким буде наш згенерований ландшафт. Отже чим менше буде значення величини нерівності тим більш нерівним то гористим буде наш згенерований ландшафт. За допомогою нерівності розраховується той самий масштаб який корегує висоту обраної точки за формулою (2.8) [16]:

$$scale = 1^{roughness} / 2 , \quad (2.8)$$

де s*cale* – значення коефіцієнту висоти; *roughness* – нерівність.

Виходячи з цього ми маємо що чим більше масштаб тим менше ми маємо заданий параметр нерівності, отже наш ландшафт більш нерівний. Та це працює і у діаметрально протилежному випадку.

Алгоритм зміщення середньої точки також має похідний алгоритм, який дещо інакше генерує висоти отриманих піків. Цей алгоритм розраховує положення піків на мапі висот таким самим чином як і стандартний алгоритм, але для розрахунку висоти піка використовується не проста рандомне значення, але ще й кореговане за допомогою фрактального методу[16].

Саме завдяки цим двом принципам досягається фрактальність ландшафту, та в той самий час зберігається достатня індивідуальність багатьох частих цього ландшафту.

Кількість полігонів — ще один важливий параметр, який регулює рівень деталізації ландшафту. Більша кількість полігонів дозволяє отримати більш детальну поверхню, що важливо для створення складних та деталізованих ландшафтів. Однак це також збільшує навантаження на комп'ютер, тому метод був оптимізований для роботи з низькополігональними моделями. Це дозволяє зберегти достатню деталізацію при генерації стилізованих ландшафтів, уникаючи перевантаження системи і забезпечуючи плавну роботу навіть у великих сценах.

Метод генерації ландшафтів починається з вибору типу ландшафту, який художник хоче створити. Це може бути один із кількох запропонованих варіантів: скелі, каньйони, гори, пагорби, дюни, луги або степи. Кожен із цих типів ландшафту має свої характерні особливості, які впливають на форму рельєфу та загальний вигляд. Наприклад, скелі зазвичай мають різкі, грубі форми, що можуть бути підкреслені використанням шуму Вороний для створення вертикальних структур. Пагорби, навпаки, мають плавніші контури, і для їх створення найкраще підходить шум Перліна, який забезпечує м'які переходи між вершинами та долинами.

Цей підхід до вибору типу ландшафту дозволяє швидко і легко адаптувати генерацію під різні проекти. Оскільки кожен тип рельєфу має свої особливості та характерні риси, користувач може створювати унікальні сцени, варіюючи форми і текстури відповідно до потреб проекту.Однією з важливих складових методу є створення карти висот, що генерується на основі різних алгоритмів шумів. У цьому проекті були використані класичні процедури, такі як шум Перліна, Вороний і Midpoint Displacement. Кожен із цих типів шуму дає унікальний результат, що дозволяє створювати різні види рельєфу. Наприклад, шум Перліна надає м’які і плавні форми, тоді як Вороний генерує більш різкі переходи між зонами висот, що є ідеальним для створення стилізованих ландшафтів зі скелями або каньйонами. Midpoint Displacement дозволяє створити різкі та нерегулярні форми, що робить його придатним для моделювання гірських хребтів [16].

На основі обраної карти висот, вона проєктується на площину в Maya, де формується тривимірний ландшафт. Використовується спеціальний алгоритм для згладжування ландшафтів, щоб уникнути занадто різких переходів, що є важливим для створення стилізованого вигляду, оскільки основна мета методу — отримати мультяшний ефект. Ландшафт не повинен виглядати реалістично; навпаки, він повинен бути більш м'яким і стилізованим, з плавними переходами та м’якими контурами.

Цей метод відрізняється від інших підходів генерації 3D-ландшафтів кількома ключовими моментами:

– Комбінація типів шуму. В даному методі передбачена можливість комбінування кількох типів шумів, що дозволяє створювати більш складні та варіативні ландшафти. Користувач може обрати або один шум, або комбінувати, що дає більшу свободу творчого підходу.

– Інтерактивність налаштувань. На відміну від багатьох автоматизованих систем, де кінцевий результат сильно залежить від випадковості, цей метод дозволяє користувачу інтерактивно змінювати параметри в процесі генерації. В Мауа створено інтуїтивно зрозуміле меню, де можна вибрати тип ландшафту, масштаб площини, параметри шуму та кількість полігонів.

– Стилізація. Основна мета полягає не в досягненні фотореалістичних зображень, а у створенні стилізованих, мультиплікаційних ландшафтів. Це відрізняє даний підхід від більшості існуючих рішень, які орієнтовані на реалістичність. М’які форми та приглушені текстури створюють ефект мультяшної естетики, що дозволяє використовувати цей метод у проектах, пов’язаних з анімацією або відеоіграми у мультяшному стилі.

– Оптимізація для низькополігональних моделей. У зв'язку з тим, що стилізовані ландшафти не потребують високої деталізації, код оптимізовано для роботи з низькою кількістю полігонів. Це робить метод ефективним навіть на слабких машинах і дозволяє легко працювати з великими сценами без втрати продуктивності.

– Можливість модифікації. Алгоритм спроектовано таким чином, що його можна розширювати. Наприклад, можна додати нові типи шумів або текстури для ландшафту. Це робить метод універсальним та гнучким для різних потреб.

Новизна даного методу полягає в тому, що він не лише дозволяє створювати стилізовані 3D-ландшафти, але й надає широкі можливості для їх

модифікації. Більшість існуючих генераторів ландшафтів обмежені стандартними наборами функцій і створюють моделі з фіксованою деталізацією або обмеженими параметрами шуму. У цьому ж підході ключовою особливістю є гнучкість налаштувань, можливість поєднувати різні типи шумів і отримувати як плавні, так і більш детальні структури ландшафтів.

Також варто зазначити, що більшість генераторів ландшафтів орієнтовані на реалістичні результати. Однак для багатьох творчих проектів, особливо в ігровій індустрії або в анімації, важливо мати можливість генерувати стилізовані, мультяшні моделі. Тому цей метод заповнює прогалину, пропонуючи інструмент для швидкого створення мультяшних 3D-ландшафтів із гнучкими налаштуваннями.

Таким чином, розроблений метод для Maya дозволяє швидко і ефективно створювати унікальні стилізовані ландшафти, що відрізняє його від інших підходів до генерації тривимірних рельєфів. Завдяки його можливостям налаштування і гнучкості, цей метод є потужним інструментом для 3D-художників і розробників, які працюють у сфері анімації, відеоігор та візуальних ефектів.

2.2 Розробка моделі інформаційної системи генерування стилізованих 3D-ландшафтів

Модель інформаційної системи (ІС) є абстрактним відображенням структури та функціоналу системи, що включає всі компоненти, їх взаємозв'язки та процеси, які відбуваються всередині системи. Така модель надає загальний огляд системи і є важливим інструментом для розробки, аналізу та документування інформаційних систем.

Діаграми варіантів використання дозволяють отримати відмінне візуальне уявлення про вимоги користувачів. Діаграма Use Case визначає поведінку системи з точки зору користувача і використовується для

з’ясування вимог замовника до системи, що розробляється, фіксації цих вимог у формі, яка дозволяє проводити подальшу розробку. До складу діаграм варіантів використання входять елементи Use Case, актори, а також відношення залежності, узагальнення і асоціації. Як і інші діаграми, діаграми Use Case можуть включати примітки і обмеження. Залежно від мети виконання процедури розрізняють такі варіанти використання: основні (базові), які забезпечують необхідну функціональність ПЗ, допоміжні, які забезпечують виконання необхідних налаштувань системи і її обслуговування, та додаткові, які забезпечують додаткові зручності для користувача.

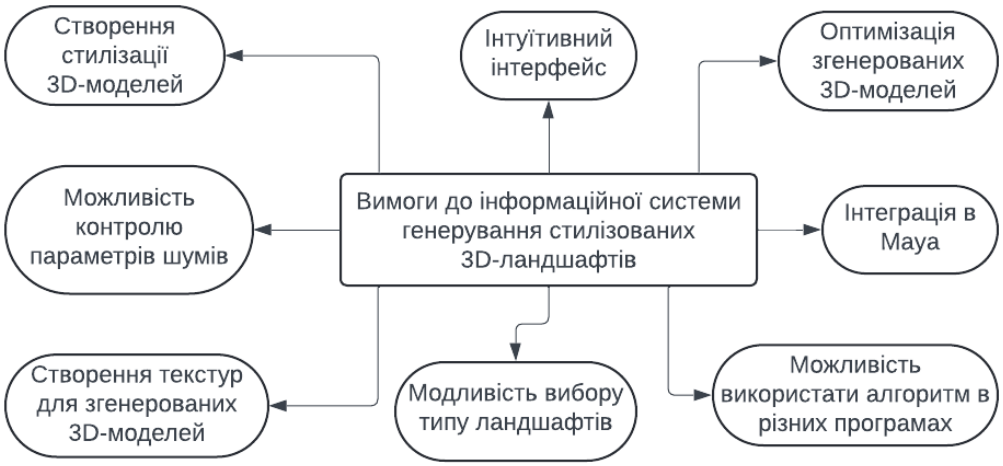

Рисунок 2.6 - Вимоги до інформаційної системи генерування стилізованих 3D-ландшафтів

При розгляді моделі використання можна виділити такі компоненти: внутрішня система у формі набору варіантів використання, зовнішнє оточення у формі набору дійових осіб, а також зв’язок між системою і зовнішнім оточенням у формі асоціацій між дійовими особами і варіантами використання. У разі необхідності можна використати спеціальну конструкцію, звану «Межі системи».

Модель інформаційної системи генератора стилізованих 3D-ландшафтів повинна включати в себе такі компоненти: система створення ландшафту, що дозволяє користувачу обирати тип ландшафту; система налаштувань, яка дозволяє користувачу адаптувати генерацію ландшафту під свої потреби;

система попереднього перегляду, що надає можливість переглянути результат генерації перед застосуванням; система збереження ландшафту, яка дозволяє користувачу зберігати результати в обраному форматі [17].

Цільовою аудиторією програми є 3D-художники, які прагнуть швидко і ефективно створювати стилізовані ландшафти для проектів у сфері анімації, ігор, віртуальної реальності тощо. Основним завданням системи є надання користувачам інтуїтивно зрозумілого інтерфейсу для генерації 3D-ландшафтів без необхідності глибоких знань програмування [17].

Створено модель вимог до інформаційної системи генератора стилізованих 3D-ландшафтів, що продемонстрована у вигляді діаграми, яка ілюструє взаємозв’язки між різними компонентами системи та вимогами користувачів. Вона включає зручний інтерфейс, можливість налаштування параметрів генерації, наявність попереднього перегляду, збереження результатів у різних форматах, а також документацію для користувачів.

Архітектура інформаційних систем для генерації 3D-ландшафтів повинна бути масштабованою та гнучкою для задоволення зростаючих потреб користувачів. Вона повинна забезпечувати надійність та ефективність роботи. Одним із підходів до розробки архітектури є використання клієнт-серверної моделі, де клієнтська частина відповідає за взаємодію з користувачем, а серверна частина – за обробку запитів та зберігання даних.

Комбінуючи ці алгоритми, ми отримали загальну структуру генерування різноманітних типів ландшафтів із різною складністю і деталізацією, рисунок 2.7.

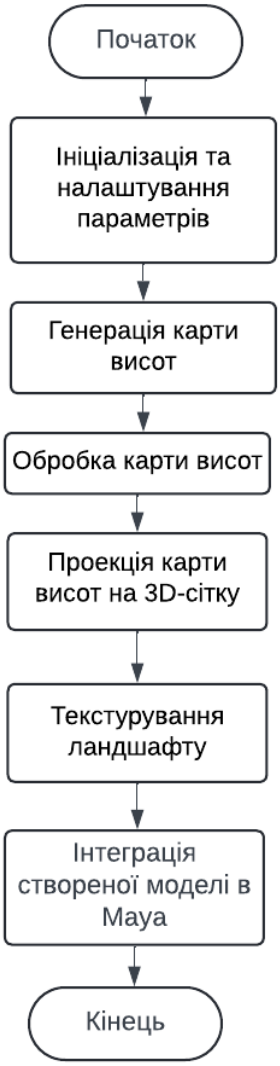

Рисунок 2.7 - Загальна структура генератора ландшафтів

Однією з ключових переваг розробленого інструменту є можливість його інтеграції в робочий процес 3D-художника. Генератор, написаний на Python, був реалізований як плагін для Maya, що забезпечує зручний інтерфейс і можливість взаємодії з іншими інструментами, які використовуються у процесі створення 3D-графіки. Користувач має змогу вибирати тип ландшафту з меню плагіну, задавати параметри, що визначають вигляд і масштаб ландшафту, а також комбінувати різні типи ландшафтів для створення більш складних і цікавих сцен. Наприклад, можна об'єднати пагорби з каньйонами, додаючи деталі в залежності від потреб конкретної сцени. Такий підхід дозволяє досягти високого рівня кастомізації, що

особливо важливо для створення індивідуальних сцен у стилізованих анімаціях або ігрових проектах, де важливо підкреслити унікальність світу.

Загалом, моделі інформаційної системи для генерації стилізованих 3D-ландшафтів включають в себе взаємозв’язки між компонентами, відповідно до яких забезпечується функціональність системи, що дозволяє користувачам легко взаємодіяти з нею та отримувати бажані результати.

2.3 Розробка алгоритму інформаційної технології генерування стилізованих 3D-ландшафтів

Алгоритм — це послідовність інструкцій, яка визначає, як виконувати певну задачу, що в нашому випадку полягає у створенні процедурних ландшафтів. Головним завданням алгоритму є забезпечення ефективності генерації різних типів ландшафтів, таких як пагорби, гори, дюни чи каньйони, які мають стилізований вигляд для використання в мультяшній графіці. Алгоритм має бути модульним, масштабованим і легко налаштовуваним, щоб користувачі могли швидко адаптувати параметри до своїх потреб.

Створення алгоритму починається з чіткого визначення його основних складових і необхідного функціоналу. Перший крок алгоритму передбачає генерацію карти висот, яка є основою для формування рельєфу. Карта висот являє собою двовимірне зображення, де кожне значення пікселя визначає висоту відповідної точки на поверхні ландшафту. Для генерування карти висот використовуються різні види процедурних шумів, такі як шум Перліна, шум Вороного та алгоритм Midpoint Displacement. Кожен з цих шумів виконує певну роль у створенні ландшафту. Шум Перліна генерує гладкі, м'які контури, що дозволяють створювати плавні пагорби, які підходять для мультяшної стилізації. Шум Вороного додає складності і створює текстуровані області, що підходять для генерації скель чи каньйонів, тоді як Midpoint Displacement вносить випадковість і варіативність, дозволяючи уникнути зайвої симетрії та забезпечити природність рельєфу.

Обробка даних в алгоритмі виконується у кілька етапів. На першому етапі генеруються базові форми ландшафту шляхом поєднання шумів Перліна та Вороного, що дозволяє створити первинну карту висот. Потім застосовуються додаткові алгоритми для підвищення деталізації та модифікації структури рельєфу. Наприклад, Midpoint Displacement використовується для додавання нерівностей і різних рівнів висоти, що робить ландшафт більш цікавим і динамічним. Усі ці обробки виконуються таким чином, щоб кінцевий результат мав стилізований вигляд, без зайвого реалізму, але з достатньою кількістю деталей для надання сцені необхідної атмосфери.

Розробка алгоритму включає також інтерактивний елемент для користувача, що дозволяє вибирати параметри генерації. У плагіні, реалізованому в Maya, користувач має змогу обирати тип ландшафту, визначати розмір, кількість деталей і використовувані алгоритми шуму. Це забезпечує гнучкість генерації, дозволяючи користувачам отримувати різні результати на основі своїх потреб і експериментувати з різними комбінаціями шумів для досягнення бажаного вигляду.

Алгоритм інформаційної технології генерування стилізованих 3D-ландшафтів має бути правильно спроєктованим, щоб забезпечити його модульність і легкість у підтримці. Кожен етап генерації ландшафту — від створення карти висот до фінальної побудови геометрії — представлений окремим модулем, що дозволяє легко вносити зміни або додавати нові функціональні можливості. Наприклад, можна додати нові види шумів або нові параметри налаштування, не порушуючи загальну структуру алгоритму. Це робить алгоритм зручним для подальшого розширення та адаптації до нових потреб користувачів.

Правильність роботи алгоритму визначається результатами генерації. Задання різних вхідних параметрів повинно приводити до очікуваних результатів, що відповідають вимогам стилізованого ландшафту. Аналіз алгоритму також включає перевірку його ефективності, тобто швидкості роботи і оптимальності використання ресурсів.

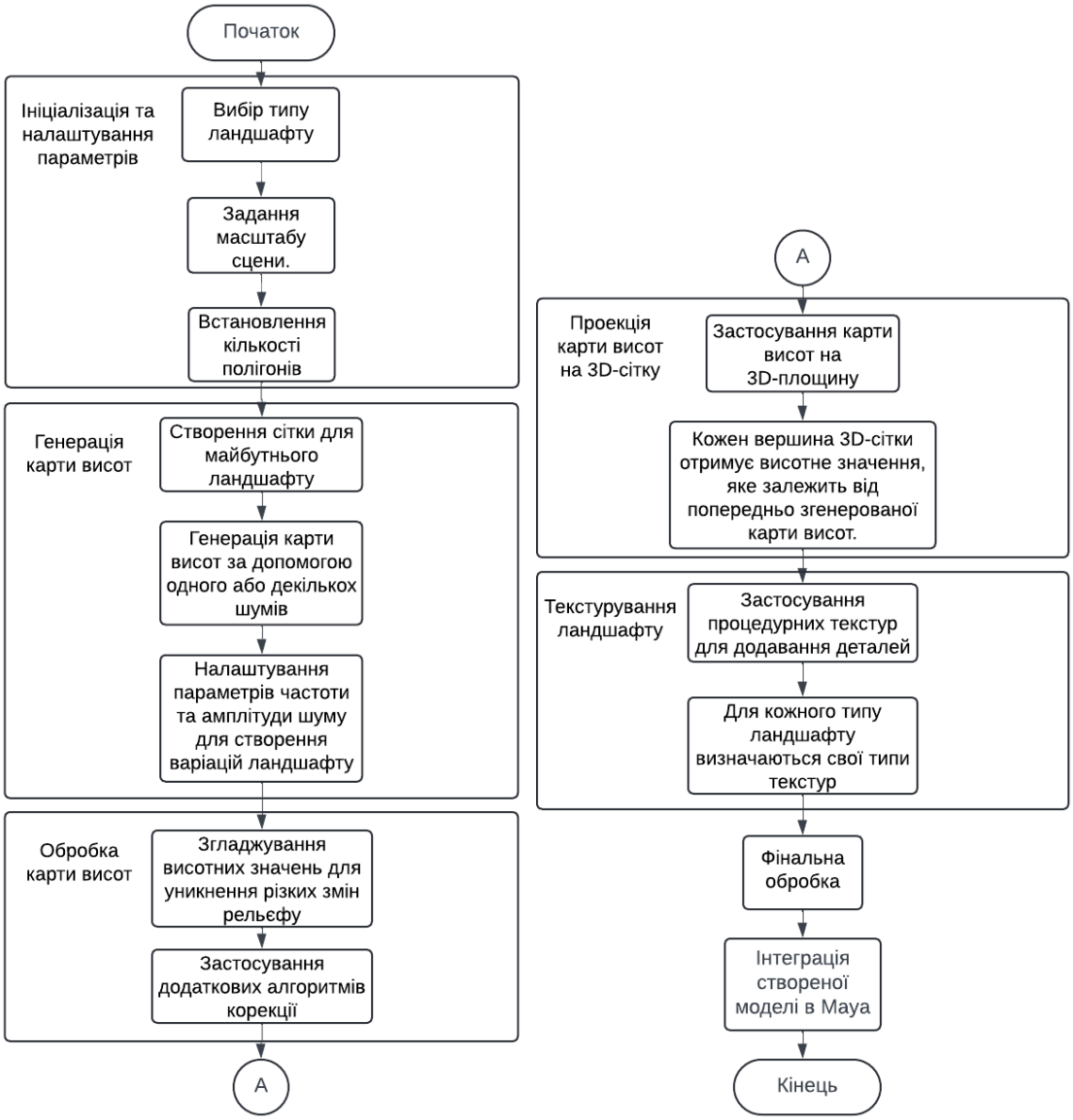

Рисунок 2.8 – Схема алгоритму інформаційної технології генерування стилізованих 3D-ландшафтів

Надійність алгоритму також є важливим фактором. Він повинен бути стійким до помилок і забезпечувати правильне генерування ландшафту навіть при введенні крайніх або нестандартних значень параметрів.

Одним із ключових аспектів розробки алгоритму є забезпечення його зручності використання. Інтерфейс для взаємодії з алгоритмом у Mаya повинен бути інтуїтивно зрозумілим, щоб навіть користувачі з мінімальним досвідом у програмуванні могли легко використовувати генератор

ландшафтів. Усі параметри повинні бути чітко позначені і доступні для налаштування, що дозволяє художникам зосередитися на творчості, а не на технічних деталях.

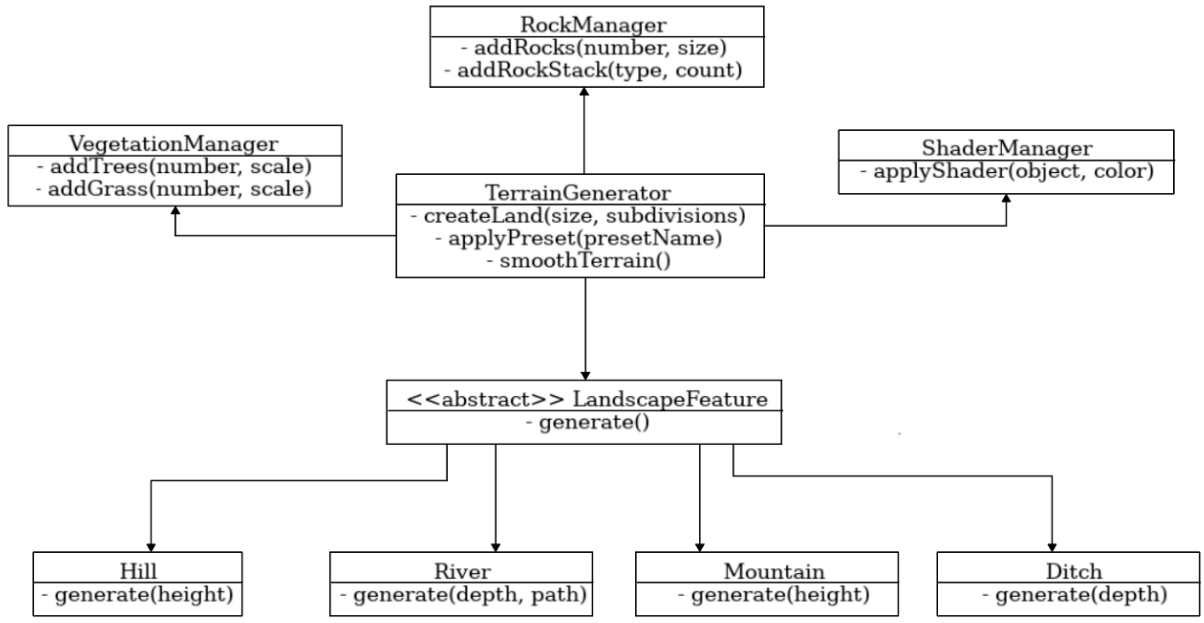

Рисунок 2.9 – UML-діаграма класів інформаційної технології генерування стилізованих 3D-ландшафтів

Таким чином, розробка алгоритму інформаційної технології генерування стилізованих 3D-ландшафтів включає в себе поєднання різних процедурних методів для створення карт висот, проєктування їх у тривимірний простір, оптимізацію геометрії та надання користувачу можливості керувати процесом генерації. Важливо, щоб алгоритм був модульним, гнучким, ефективним і надійним, що забезпечує його придатність для використання в різних проєктах і полегшує роботу 3D-художників. У результаті створюється інструмент, який дозволяє швидко генерувати стилізовані ландшафти для мультяшної графіки, забезпечуючи високий рівень кастомізації і творчої свободи для користувачів.

Алгоритм генерування стилізованих 3D-ландшафтів полягає у поетапному створенні карти висот, її обробці для отримання різноманітних форм і подальшому перетворенні на тривимірну геометрію в середовищі 3D

моделювання, як-от Maya. Нижче представлено загальний алгоритм цього процесу:

Ініціалізація параметрів:

– Встановити розміри ландшафту (ширина, висота).

– Визначити тип ландшафту (пагорби, дюни, гори, каньйони тощо).

– Визначити параметри шуму (амплітуда, частота, кількість октав тощо).

– Обрати тип процедурного шуму (Перліна, Вороного, Midpoint Displacement або комбінація).

Генерація базової карти висот:

– Створити двовимірну матрицю для зберігання висотних значень (карта висот).

– Застосувати шум Перліна для створення плавних контурів рельєфу.

– Додати шум Вороного для створення складних текстур (скелі, каньйони).

– Використати алгоритм Midpoint Displacement для додання випадкових нерівностей, що надасть природного вигляду.

Обробка карти висот:

– Нормалізувати значення висот для забезпечення плавного переходу між ділянками рельєфу.

– Застосувати згладжування для усунення різких перепадів висот, щоб створити мультяшний вигляд ландшафту.

– Внести корекції висот відповідно до обраного типу ландшафту (наприклад, зробити пагорби більш округлими або збільшити висоту гірських вершин).

– Проєктування карти висот на тривимірну площину:

– Створити площину з розділенням на сітку вершин.

– Для кожної вершини площини встановити координату Y, використовуючи значення з карти висот.

– Оптимізувати кількість полігонів, залишаючи достатньо деталізації, але зменшуючи обчислювальну складність моделі.

– Текстурування ландшафту:

– Створити карту текстур для ландшафту на основі висотних даних (наприклад, трава на низьких ділянках, скелі на високих).

– Застосувати стилізовані текстури, які відповідають мультяшному стилю.

Цей алгоритм описує послідовність дій для створення стилізованих 3D-ландшафтів, використовуючи процедурні методи генерації. Він забезпечує можливість створювати різні види рельєфів, а також надає інструменти для їх адаптації під конкретні вимоги, що робить його корисним для 3D художників та розробників ігор.

2.4 Висновок до розділу 2

У другому розділі було розроблено модель інформаційної системи для генерації стилізованих 3D-ландшафтів, визначено основні модулі та етапи її функціонування. Розглянуто ключові технології та підходи, зокрема інтеграцію шумів Перліна, Вороного та Midpoint Displacement для покращення реалізму та стилізації ландшафтів. Детально проаналізовано ефективність алгоритмів щодо оптимізації генерації для зменшення навантаження на ресурси та покращення якості результату.

Усі враховані аспекти дозволяють створити гнучку та масштабовану систему для генерації ландшафтів, що відповідає вимогам магістерської роботи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ГЕНЕРУВАННЯ СТИЛІЗОВАНИХ 3D-ЛАНДШАФТІВ

3.1 Вибір та обґрунтування засобів програмування

Для створення плагіна процедурної генерації стилізованих 3D-ландшафтів у Maya було обрано мову програмування Python та середовище розробки Maya, яке має вбудовану підтримку Python API. Таке рішення є надзвичайно виправданим через ряд причин, включаючи зручність у роботі з процедурними графічними алгоритмами, легкість у використанні об’єктно-орієнтованих підходів, а також багаті можливості Maya для роботи з тривимірною графікою та анімацією. У порівнянні з іншими мовами та середовищами, такими як C++ у 3ds Max або MEL у Maya, Python разом із Maya забезпечують найбільш збалансоване середовище для ефективної розробки 3D-ландшафтів із можливістю використання розширених процедурних фреймворків [18].

Python є мовою високого рівня, яка виділяється простим і читабельним синтаксисом. Це особливо важливо для розробників, які працюють над створенням складних графічних алгоритмів, таких як процедурна генерація ландшафтів. Завдяки інтуїтивному синтаксису Python розробники можуть швидко писати код і зосереджуватися на логіці алгоритмів, а не на низькорівневих аспектах реалізації.

Мова Python має динамічну типізацію, що дозволяє скорочувати кількість коду без втрати гнучкості. Наприклад, у процедурному моделюванні це означає, що розробник може швидко змінювати типи даних або додавати нові функціональні елементи до програми, не переписуючи значні обсяги коду. Це особливо корисно при тестуванні різних конфігурацій генерації ландшафтів, таких як комбінація шумів Перліна та Вороного або налаштування їхніх параметрів [18].

Python володіє багатою стандартною бібліотекою, яка включає інструменти для роботи з текстом, числами, графікою, геометрією та багатьма іншими задачами. Крім того, є численні додаткові пакети, які роблять Python незамінним інструментом для 3D моделювання.

PyOpenGL може використовуватися для створення більш складних графічних візуалізацій або тестування ландшафтів у реальному часі.

Python також добре інтегрується з різними програмними платформами, зокрема Autodesk Maya, Blender та Houdini, що забезпечує широку універсальність у сфері 3D моделювання. Це дозволяє розробникам легко переносити свої знання між різними інструментами. Наприклад, алгоритм, розроблений у Blender для процедурного текстурування, може бути адаптований до Maya з мінімальними змінами [18].

У сфері 3D Python також популярний через його інтерактивність. Мова підтримує інтерактивне тестування через консолі, такі як Jupyter Notebook або середовище Maya Script Editor. Це забезпечує миттєвий зворотний зв'язок, дозволяючи розробникам швидко тестувати нові ідеї, змінювати параметри шумів або перевіряти, як працює комбінація різних алгоритмів у генерації ландшафтів [18].

Порівняно з C++, який використовується в Autodesk 3ds Max, Python є набагато простішим у використанні. C++ вимагає значно більше зусиль для створення алгоритмів генерації ландшафтів, особливо через складніші структури пам'яті, вказівники та ручне керування пам'яттю. Хоча C++ має високу продуктивність і може бути використаний для високопродуктивних завдань, зокрема для рендерингу, Python є значно більш інтуїтивним для задач процедурної генерації, що вимагають постійних змін і тестування.

Також важливо відзначити, що Python підтримується більшістю популярних 3D-середовищ, зокрема Maya, Houdini та Blender. Така універсальність дозволяє розробникам легко переносити знання та навички між різними платформами, забезпечуючи гнучкість у роботі та можливість інтегрувати Python-код у різні 3D-системи без потреби в значному

переписуванні. Зокрема, у Maya Python підтримується через API та MASH, що надає прямий доступ до управління геометрією, об’єктами, матеріалами та анімаціями, дозволяючи ефективно будувати складні графічні моделі та ландшафти [19].

Функціональність Maya як платформи для процедурного моделювання Autodesk Maya є однією з найпотужніших і найпопулярніших платформ для 3D-моделювання, анімації та візуалізації. Основною перевагою Maya є її інструментарій для роботи з тривимірною графікою та анімацією, а також вбудовані можливості для створення процедурних об'єктів і контролю над їх параметрами. Зокрема, в Maya реалізований фреймворк MASH, який дозволяє програмно генерувати ландшафти, об'єкти, анімації та легко масштабувати їхню складність. MASH також дозволяє використовувати процедурний підхід для створення динамічних сцен і розташування об'єктів у просторі, що є незамінним для процедурного моделювання ландшафтів [20].

Порівняння з іншими 3D-редакторами: Maya проти 3ds Max і Blender, таблиці 3.1.

Порівнюючи Maya з 3ds Max, слід зазначити, що Maya має більш розвинену підтримку процедурної анімації та моделювання завдяки інтеграції MASH, тоді як 3ds Max більше зосереджений на архітектурному моделюванні та візуалізації. Хоча обидві платформи мають підтримку Python, в Maya цей API є більш гнучким і оптимізованим для складних процедурних сценаріїв. Інструменти MASH надають більше можливостей для налаштування складних процедурних ефектів і застосування алгоритмів генерації ландшафтів.

Blender також є потужним інструментом, і хоча він активно розвивається у напрямку процедурної графіки, його можливості для складних процедурних ландшафтів менш гнучкі у порівнянні з Maya, особливо в частині взаємодії з іншими програмами та спеціалізованими фреймворками. Крім того, в Maya є значно більше інструментів для роботи з великою кількістю об'єктів, що особливо важливо при розробці процедурного плагіна, орієнтованого на великий ландшафт з багатьма деталями.

Python у поєднанні з MASH надає ефективний інструмент для розробки процедурних ландшафтів. Зокрема, MASH дозволяє створювати об’єкти, розміщувати їх у просторі, керувати їхнім масштабом, орієнтацією та іншими параметрами. Це забезпечує різноманітність у створюваних елементах ландшафту (трава, дерева, скелі, водні об’єкти тощо), дозволяючи створювати складні сцени без необхідності вручну розміщувати кожен елемент. MASH підтримує випадковість у параметрах об’єктів, що ідеально підходить для створення природного вигляду стилізованих ландшафтів [19].

Таблиця 3.1 – Порівняльна таблиця Python + Maya проти C++ + 3ds Max і Blender (Python API)

	Python + Maya	C++ + 3ds Max	Blender (Python API)
1	2	3	4
Простота використання	Простий та зрозумілий синтаксис, легка інтеграція з 3D API	Складний синтаксис, вимагає знань управління пам'яттю	Висока, але API менш потужний для процедурної генерації
Типізація	Динамічна, не вимагає ручного управління типами	Статична, вимагає точної специфікації типів	Динамічна, підтримка Python API
Продуктивність	Достатня для процедурного моделювання та генерації сцен середньої складності	Висока, підходить для ресурсомістких задач	Достатня, але обмежена можливостями Blender
Можливості API для моделювання	Високий рівень контролю над параметрами та анімацією через MASH	Добре підходить для статичного моделювання, менше процедурних можливостей	Можливість процедурного моделювання, але з обмеженнями
Підтримка спільноти	Висока, багато ресурсів та прикладів для Python + Maya	Висока для C++ і 3ds Max, але важче знайти приклади для процедурного моделювання	Зростаюча спільнота, підтримка через Blender Foundation

Таблиця 3.1 – Продовження порівняльної таблиці Python + Maya проти C++ + 3ds Max і Blender (Python API)

	Python + Maya	C++ + 3ds Max	Blender (Python API)
1	2	3	4
Універсальність	Може використовуватись у різних 3D-середовищах (Maya, Houdini, Blender)	Більше підходить для спеціалізованих застосувань	В основному використовується лише в Blender
Автоматизація	Просте управління пам'яттю та можливість швидкої зміни параметрів	Складне ручне управління пам'яттю та складність налаштувань	Автоматизоване управління, але без потужного процедурного фреймворку

Отже, Python і Maya забезпечують оптимальне середовище для розробки процедурного плагіна, оскільки:

Простий синтаксис та легка інтеграція Python роблять його ідеальним для розробників, які створюють алгоритми для генерації ландшафтів.

Maya як середовище моделювання надає MASH API, що спрощує управління параметрами та автоматичне розміщення об’єктів, таких як дерева та трава, з можливістю їхньої легкої модифікації.

Відмінності від C++ і 3ds Max: Python значно спрощує розробку процедурних моделей завдяки автоматичному керуванню пам'яттю та динамічній типізації, що підходить для швидкого створення та тестування складних алгоритмів.

Інтеграція з іншими платформами: Python дозволяє легко переносити знання між різними 3D-системами, роблячи його універсальним вибором для розробників 3D-ландшафтів.

Загалом, вибір Python і Maya для реалізації плагіна з використанням MASH API є найдоцільнішим рішенням для ефективної роботи з процедурною графікою та автоматизацією, що дозволяє створювати складні, стилізовані 3D-ландшафти з високим рівнем налаштувань та інтерактивності.

3.2 Програмна реалізація системи генерування стилізованих 3D-ландшафтів

Основним функціоналом програмного забезпечення є створення процедурного 3D-ландшафту, який автоматично генерує сцени з різними типами ландшафтів (пагорби, скелі, гори тощо) за допомогою процедурної генерації. Цей підхід дозволяє створювати художньо стилізовані пейзажі, які вражають своїм гармонійним виглядом і деталізацією. Процес реалізовано за допомогою мови Python у середовищі Maya, що забезпечує автоматизацію генерації, гнучкість у налаштуваннях і можливість інтеграції у робочий процес 3D-художників.

Програмне забезпечення використовує шумові алгоритми, зокрема Перліна, Вороного та Midpoint Displacement, кожен з яких відіграє важливу роль у формуванні рельєфу. Шум Перліна створює плавні переходи та органічні форми, що нагадують природні рельєфи, такі як пагорби або хвилясті рівнини. Шум Вороного формує чіткі межі та гострі контури, які підходять для моделювання скель, каньйонів і урвищ. Алгоритм Midpoint Displacement забезпечує додаткову деталізацію шляхом створення дрібних нерівностей та підвищення динаміки поверхні.

Процес генерації включає кілька етапів, які забезпечують високий рівень деталізації та стилізації. Спочатку обирається тип ландшафту, його параметри, такі як розміри, висота, глибина і текстурність. Далі обраний алгоритм генерує карту висот, яка слугує основою для створення об'ємного рельєфу. Карта висот проєктується на площину в Maya, формуючи 3D-об’єкт, який можна налаштовувати та доопрацьовувати. Крім того, для досягнення більшого рівня контролю над виглядом ландшафту передбачена можливість комбінування різних типів шумів, що дозволяє створювати складні сцени з плавними переходами між різними типами рельєфу.

Структура коду побудована модульно, що дозволяє розподілити основні етапи генерації між окремими функціями. Цей підхід спрощує налаштування,

модифікацію та розширення функціоналу. Наприклад, функції генерації шумів відповідають за створення базових текстур для висотних карт, а функції формування об'єктів забезпечують їхнє перетворення у 3D-геометрію. Інші модулі реалізують обробку текстур, що дає змогу додавати до рельєфу кольори, стилізовані текстури та інші візуальні ефекти [21].

Для підвищення зручності використання програмне забезпечення оснащене графічним інтерфейсом, що дозволяє користувачам легко налаштовувати параметри генерації. У меню можна вибрати тип ландшафту, масштаб сцени, рівень деталізації та додаткові ефекти. Зміна цих параметрів у реальному часі забезпечує швидкий зворотний зв'язок, що значно спрощує створення необхідного результату.

Реалізація функціоналу здійснюється через Python API у Maya. Це дає змогу інтегрувати процедурні інструменти безпосередньо у середовище 3D-моделювання. Основні функції API використовуються для створення геометрії, зміни параметрів об'єктів, додавання текстур та автоматичного накладання висотних карт. Водночас забезпечується можливість швидкої обробки великих об'ємів даних, що є критично важливим для створення складних сцен. У коді передбачено також оптимізацію використання ресурсів Maya, що знижує ризики перевантаження системи під час генерації великих або деталізованих ландшафтів.

Окрім основної функції генерації, програмне забезпечення має кілька додаткових можливостей. Наприклад, реалізовано налаштування масштабу шуму для створення більш природного вигляду поверхонь або їхньої стилізації під мультяшний вигляд. Це особливо корисно для проєктів у стилі фентезі, мультфільмів або комп'ютерних ігор, де необхідно досягти балансу між реалістичністю та художньою виразністю.

Процедурний підхід до створення 3D-ландшафтів значно скорочує час на моделювання, забезпечуючи високу якість результату. Завдяки автоматизації, система підходить як для швидкого прототипування сцен, так і

для використання в комерційних проектах, дозволяючи художникам зосередитися на творчих аспектах.

```
import maya.cmds as cmds
import random
# Функція для генерації шуму Перліна
def perlin_noise(width, height, scale):
    # Код для створення шуму Перліна з заданими параметрами
    pass
# Функція для генерації шуму Вороного
def voronoi_noise(width, height, scale):
    # Код для створення шуму Вороного
    pass
# Функція для генерації рельєфу методом Midpoint Displacement
def midpoint_displacement(width, height, roughness):
    # Код для реалізації алгоритму Midpoint Displacement
    pass
# Основна функція для вибору типу ландшафту
def generate_landscape(landscape_type):
    if landscape_type == "hills":
        return perlin_noise(width=10, height=10, scale=5)
    elif landscape_type == "canyons":
        return voronoi_noise(width=15, height=15, scale=3)
    elif landscape_type == "mountains":
        return midpoint_displacement(width=20, height=20, roughness=1.2).
# Виклик основної функції для створення ландшафту
generate_landscape("hills").
```

Генерація шуму Перліна

Шум Перліна є основою для плавних природних рельєфів, таких як пагорби та м’які гори. Для цього використовується функція perlin_noise, яка створює карту висот на основі визначених параметрів:

```
import maya.cmds as cmds
import random
def perlin_noise(width, height, scale):
    # Створення матриці для карти висот
    height_map = [[0 for _ in range(width)] for _ in range(height)]
    # Процедурне заповнення карти шумом Перліна
    for x in range(width):
        for y in range(height):
            height_map[x][y] = random.uniform(-1, 1) * scale
    # Створення полігональної сітки для рельєфу на основі карти висот
    cmds.polyPlane(width=width, height=height, subdivisionsWidth=width-1, subdivisionsHeight=height-1)
    for x in range(width):
        for y in range(height):
            cmds.select(f"pPlane1.vtx[{x*width + y}]")
            cmds.move(height_map[x][y], y=True).
```

Генерація шуму Вороного

Шум Вороного створює різкіші форми, що добре підходять для каньйонів або крутих схилів. Цей алгоритм генерує базову структуру рельєфу, де відстань до найближчої точки впливає на висоту поверхні:

```
def voronoi_noise(width, height, scale):
    # Випадкове розташування контрольних точок
    points = [(random.randint(0, width), random.randint(0, height)) for _ in range(10)]
    height_map = [[0 for _ in range(width)] for _ in range(height)]
    # Генерація шуму Вороного
    for x in range(width):
        for y in range(height):
            closest_distance = min([((x - px)**2 + (y - py)**2)**0.5 for px, py in points])
```

```
                height_map[x][y] = closest_distance * scale
        # Створення сітки для рельєфу
        cmds.polyPlane(width=width, height=height, subdivisionsWidth=width-1, subdivisionsHeight=height-1)
        for x in range(width):
            for y in range(height):
                cmds.select(f"pPlane1.vtx[{x*width + y}]")
                cmds.move(height_map[x][y], y=True).
```

Midpoint Displacement для деталізованих гірських рельєфів. Алгоритм Midpoint Displacement розбиває діапазони значень висот на менші сегменти, створюючи більш детальні та різноманітні структури. Це підходить для гірських масивів з різними вершинами.

```
    def midpoint_displacement(width, height, roughness):
        # Ініціалізація базової сітки з крапковими висотами
        height_map = [[0 for _ in range(width)] for _ in range(height)]
        # Функція для рекурсивного розбиття і додавання випадкових значень
        def divide(x0, y0, x1, y1, offset):
            if x1 - x0 < 2:
                return
            xm, ym = (x0 + x1) // 2, (y0 + y1) // 2
            avg = (height_map[x0][y0] + height_map[x1][y0] + height_map[x0][y1] + height_map[x1][y1]) / 4
            height_map[xm][ym] = avg + random.uniform(-offset, offset)
            divide(x0, y0, xm, ym, offset * roughness)
            divide(xm, y0, x1, ym, offset * roughness)
            divide(x0, ym, xm, y1, offset * roughness)
            divide(xm, ym, x1, y1, offset * roughness)
        # Рекурсивне заповнення карти висот
        divide(0, 0, width - 1, height - 1, 10.0)
```

```
    # Створення сітки
    cmds.polyPlane(width=width, height=height, subdivisionsWidth=width-1, subdivisionsHeight=height-1)
    for x in range(width):
        for y in range(height):
            cmds.select(f"pPlane1.vtx[{x*width + y}]")
            cmds.move(height_map[x][y], y=True).
```

Комбінування різних типів ландшафтів

Використовуючи інтерфейс, користувач може обирати й комбінувати типи рельєфу для створення комплексних сцен. Ось фрагмент коду для виклику основних функцій генерації на основі вибору типу ландшафту.

```
def generate_landscape(landscape_type):
    if landscape_type == "hills":
        return perlin_noise(width=10, height=10, scale=5)
    elif landscape_type == "canyons":
        return voronoi_noise(width=15, height=15, scale=3)
    elif landscape_type == "mountains":
        return midpoint_displacement(width=20, height=20, roughness=1.2)
# Виклик основної функції для створення ландшафту
generate_landscape("hills").
```

Інтерфейс користувача та інтерактивність: Інтерфейс дозволяє користувачу вибирати тип ландшафту, що створюється, і змінювати його параметри. Завдяки доступу до API Maya, генерація здійснюється інтерактивно, і результат відображається в реальному часі, що полегшує процес налаштування і коригування параметрів.

Процедурне створення карт висот: Картографічні дані генеруються відповідно до вибраного алгоритму шуму та параметрів рельєфу. Карта висот визначає форму ландшафту і його текстуру.

Застосування текстур: На основі процедурно згенерованих даних до об'єктів сцени застосовується стилізована текстура, яка додає кольору та виразності ландшафту.

Комбінування ландшафтів: Система дозволяє поєднувати різні ландшафти в одній сцені, що створює комплексні природні середовища, як-от пагорби з каньйонами чи гірські пасма.

Переваги використання Python API у Maya: Python API в Maya надає великі можливості для гнучкої процедурної генерації 3D-ландшафтів завдяки інтеграції з 3D-об’єктами та параметрами сцени. Це дозволяє динамічно змінювати параметри без повторної генерації і швидко отримувати візуальний результат, на відміну від інших середовищ, де процес генерування більш статичний або вимагає додаткових ресурсів.

Реалізація процедурного генератора 3D-ландшафтів у Maya за допомогою Python забезпечує гнучкість, можливість детальної настройки та високу продуктивність, що дозволяє ефективно створювати стилізовані пейзажі для художніх і навчальних проектів.

3.3 Тестування системи генерування стилізованих 3D-ландшафтів

Для початку тестування було запущено плагін у середовищі Autodesk Maya. Після активації плагіна відкривається інтерактивне вікно з різноманітними параметрами, які дозволяють створювати стилізовані 3D-ландшафти. Інтерфейс поділений на кілька функціональних блоків, кожен з яких відповідає за певний етап створення ландшафту. Наприклад, є розділи для вибору готових пресетів, створення порожньої площини, додавання базових елементів (пагорбів, гір, водойм тощо), а також їх подальшої модифікації.

На рисунку 3.1 зображено загальний вигляд інтерфейсу плагіна.

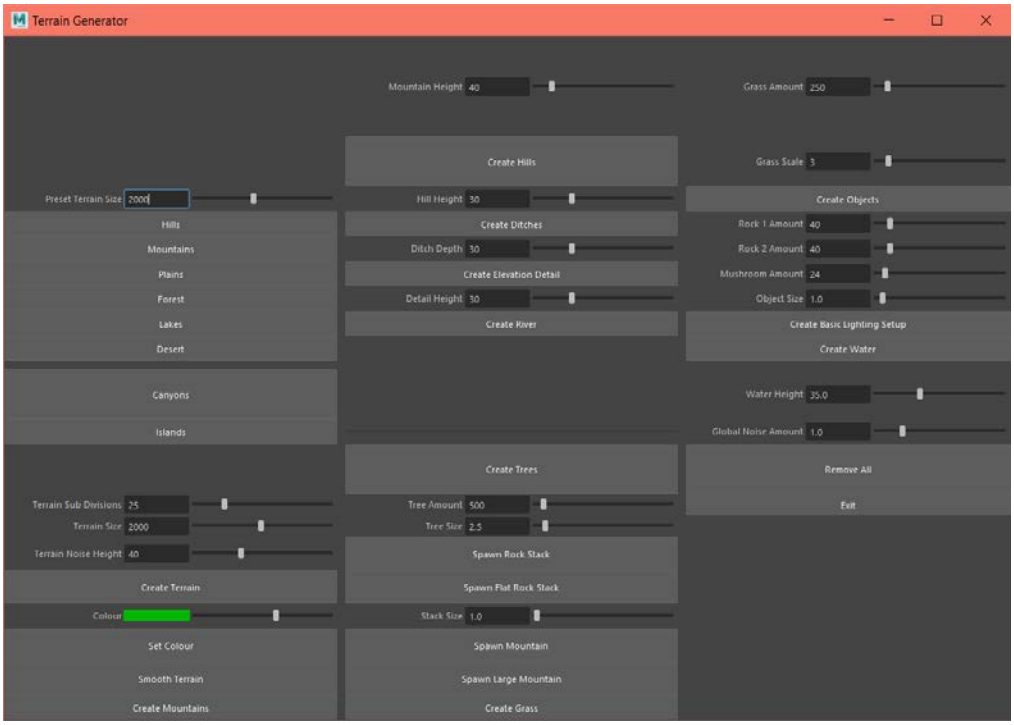

Рисунок 3.1 – Загальний вигляд інтерфейсу плагіну генерування стилізованих 3D-ландшафтів

Спочатку було протестовано генерацію готових пресетів. Кожен пресет відображає унікальний ландшафт із заданими параметрами, такими як пагорби, гори або водойми.

На рисунках 3.2–3.4 наведені приклади згенерованих ландшафтів:

– Рисунок 3.2 демонструє стилізований ландшафт із пагорбами, який характеризується плавними контурами та м'якими переходами висот.

– Рисунок 3.3 ілюструє створення ландшафту з горами, де помітно чітко виражені піки, створені за допомогою шуму Вороного.

– Рисунок 3.4 показує ландшафт із водоймами, що включає реалізацію низьких западин, заповнених "водою".

–

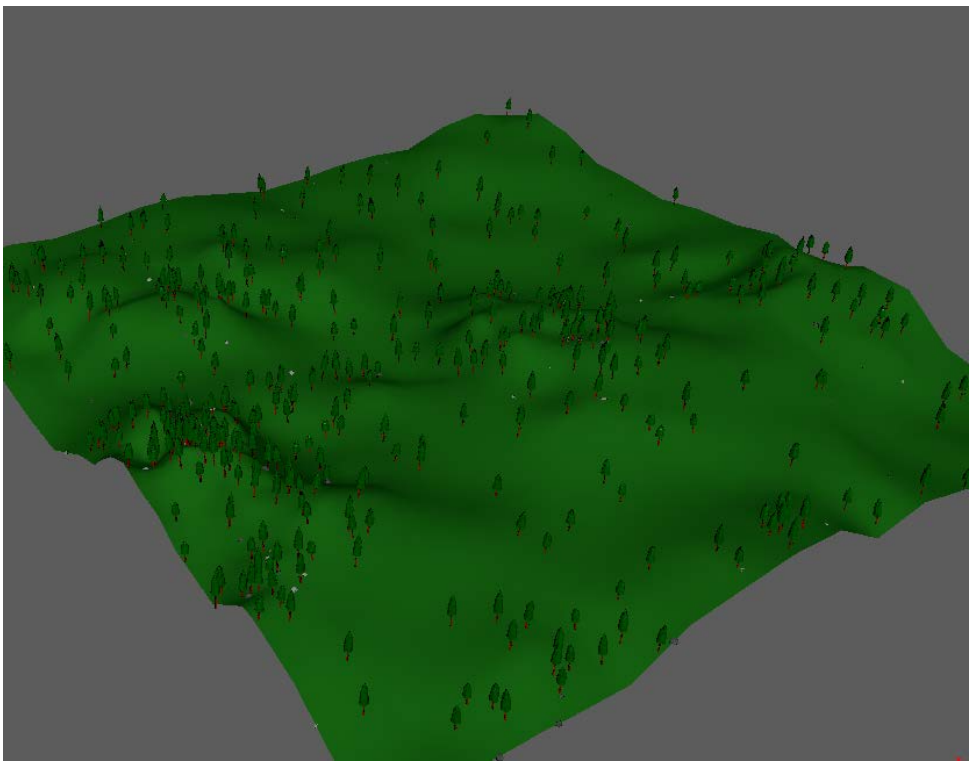

Рисунок 3.2 – Згенерований за допомогою створеного плагіну ландшафт з пагорбами

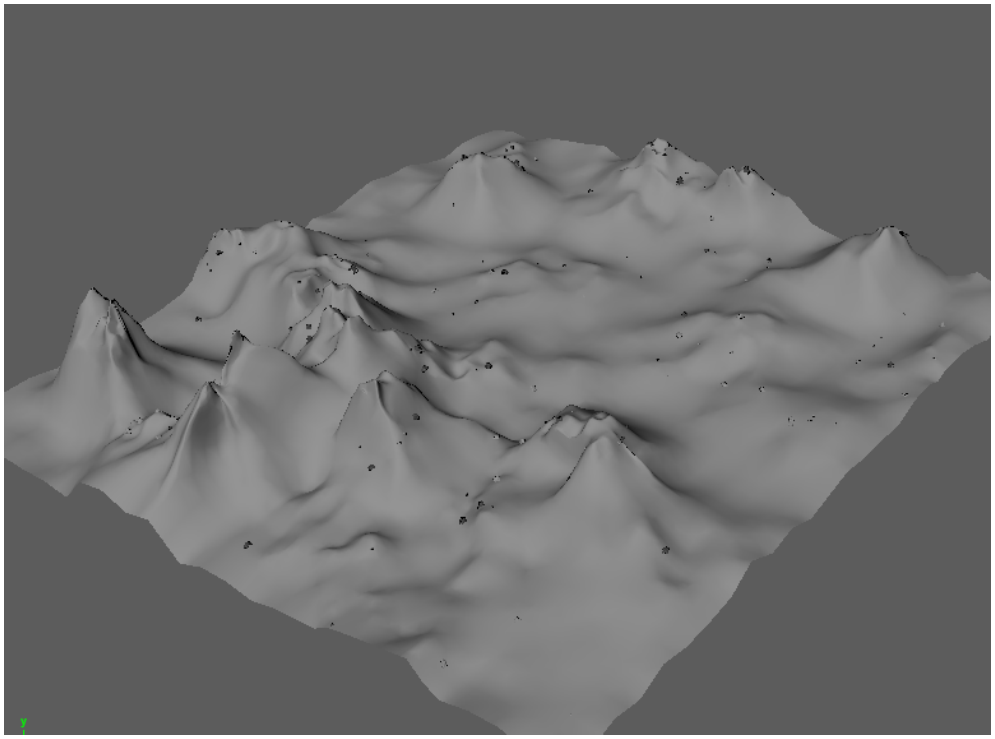

Рисунок 3.3 – Згенерований за допомогою створеного плагіну ландшафт з горами

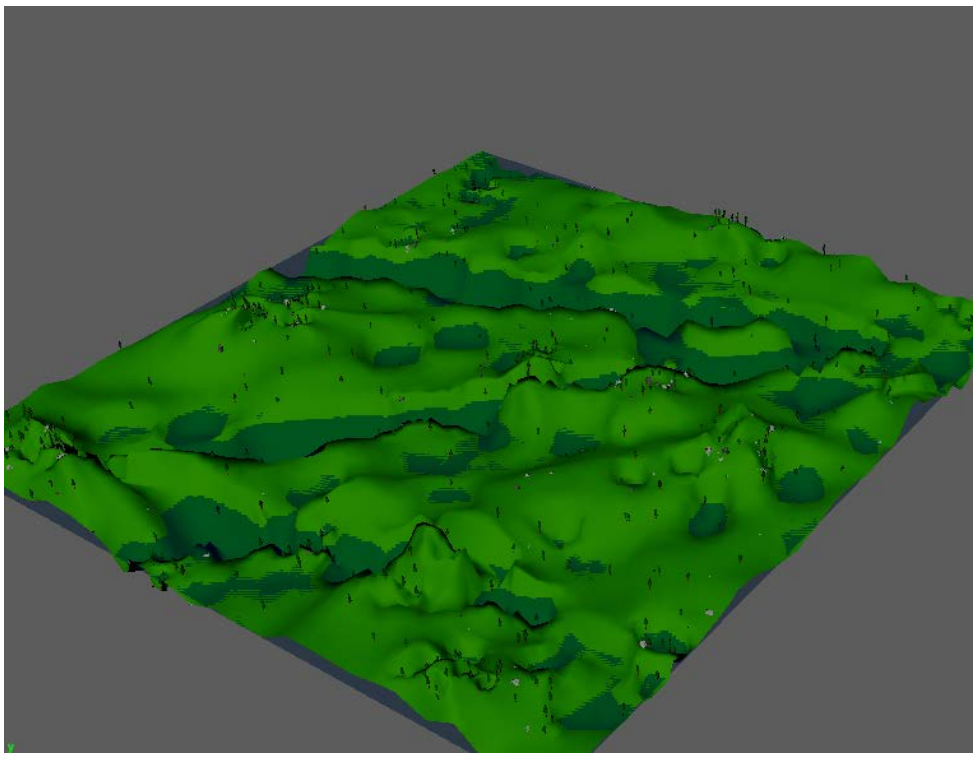

Рисунок 3.4 – Згенерований за допомогою створеного плагіну ландшафт з озерами

Всі пресети працюють коректно, причому кожна нова генерація створює унікальний результат завдяки використанню алгоритмів випадковості та процедурної генерації.

Другий етап тестування полягав у ручному створенні ландшафтів із використанням доступних інструментів плагіна.Наступним кроком генеруємо власний ландшафт.

Спершу було згенеровано базову площину, на яку надалі проєктувалися елементи ландшафту. Як видно на рисунку 3.5, площина створюється швидко та коректно, забезпечуючи початкову основу для подальших маніпуляцій.

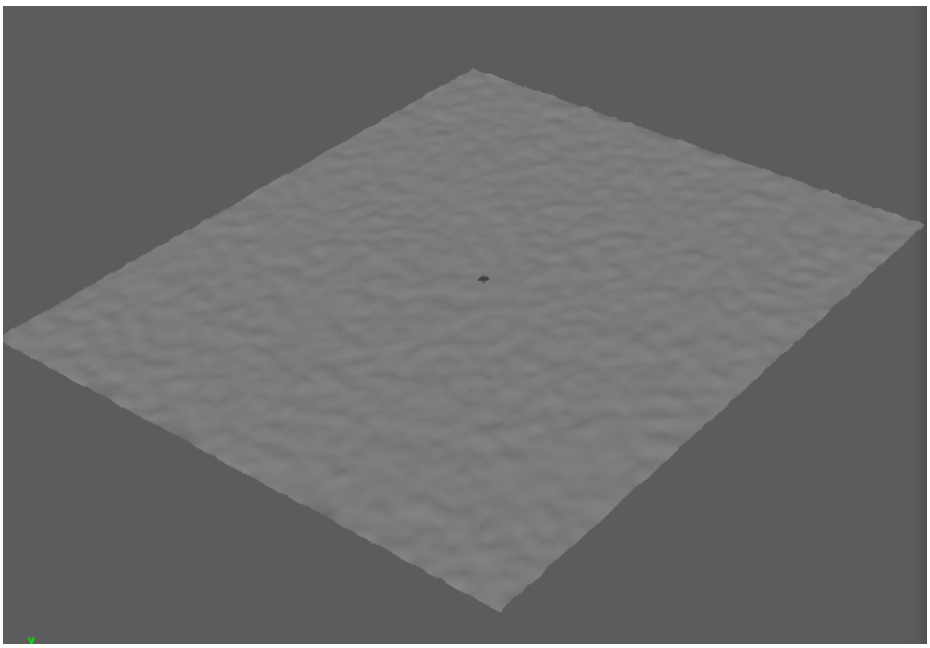

Рисунок 3.5 – Згенерована за допомогою створеного плагіну площина

На створену площину додано пагорби з використанням заданих параметрів висоти та радіусу. На рисунку 3.6 видно, що результат виглядає стилізовано, завдяки використанню шуму Перліна та методів згладжування.

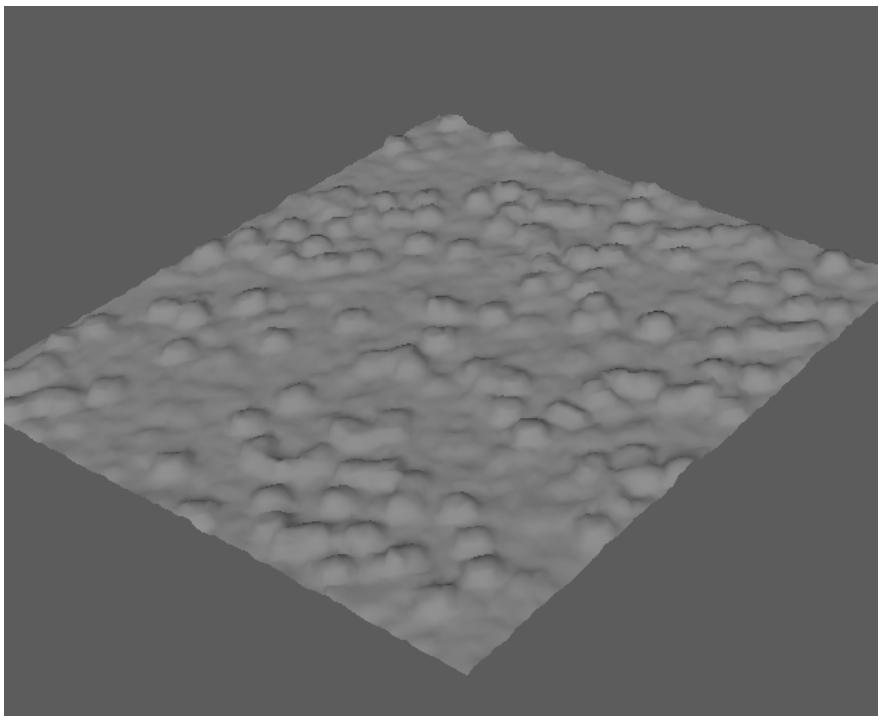

Рисунок 3.6 – Згенеровані за допомогою створеного плагіну пагорби на площині

Наступним кроком було додавання гір до ландшафту. Гори мають чітко виражені вершини, що додає сцені більше динаміки та складності. Приклад наведено на рисунку 3.7.

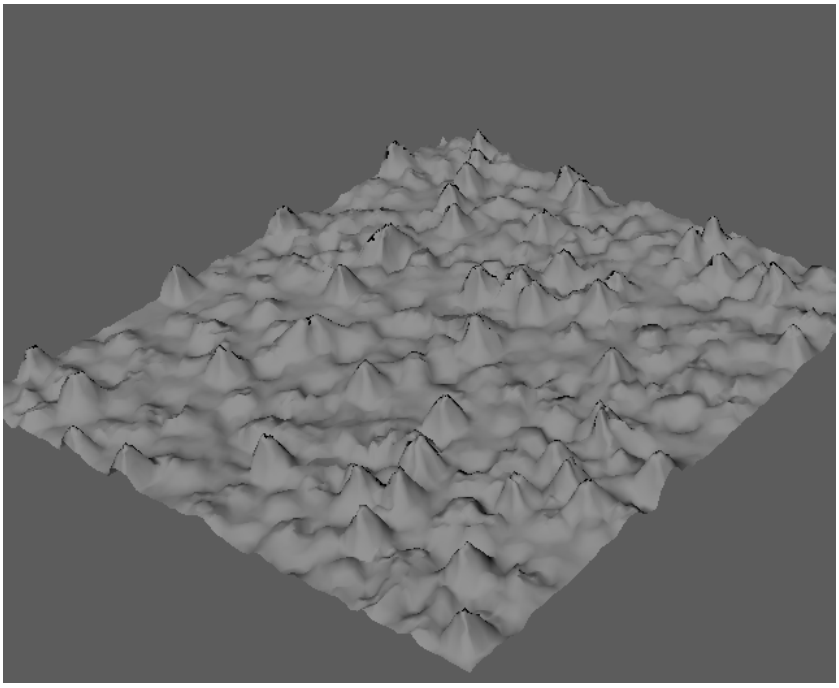

Рисунок 3.7 – Згенеровані за допомогою створеного плагіну гори на площині

Додавання русел річок стало наступним етапом. На рисунку 3.8 зображено, як річки природно інтегруються у рельєф, враховуючи ухил поверхні. Це було реалізовано за допомогою алгоритмів трасування потоків води.

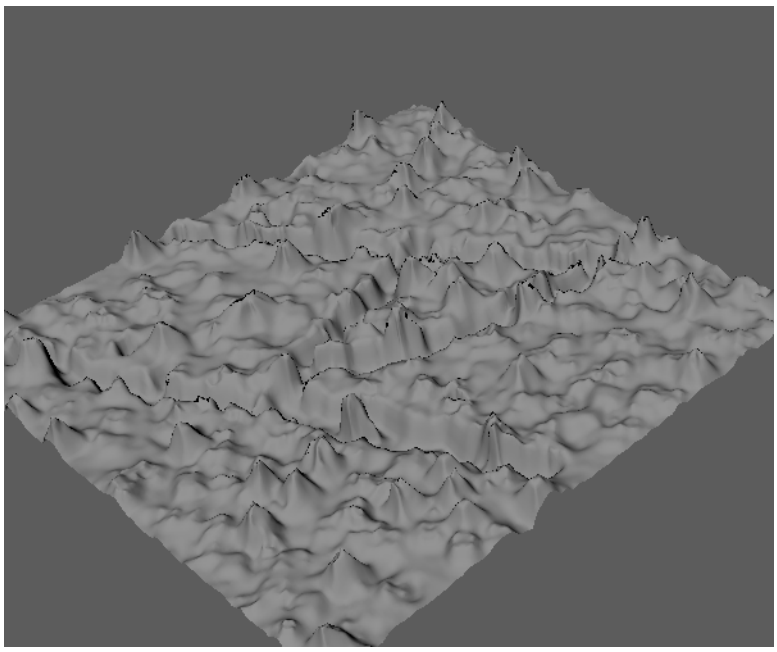

Рисунок 3.8 – Згенеровані за допомогою створеного плагіну русла річок

Ландшафт було доповнено деревами та травою, що значно підвищило деталізацію сцени. Результат зображено на рисунку 3.9. Програма дозволяє задавати щільність, висоту та радіус покриття рослинності.

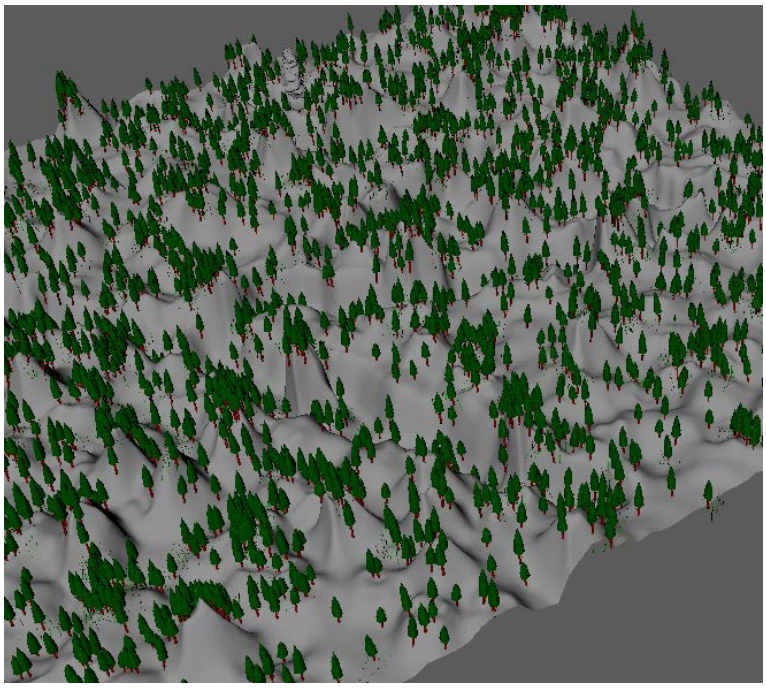

Рисунок 3.9 – Згенерована за допомогою створеного плагіну рослинність

Водойми були інтегровані у ландшафт на основі згенерованих низин. На рисунку 3.10 видно, що вода правильно відображається в заданих областях, додаючи реалістичності до сцен.

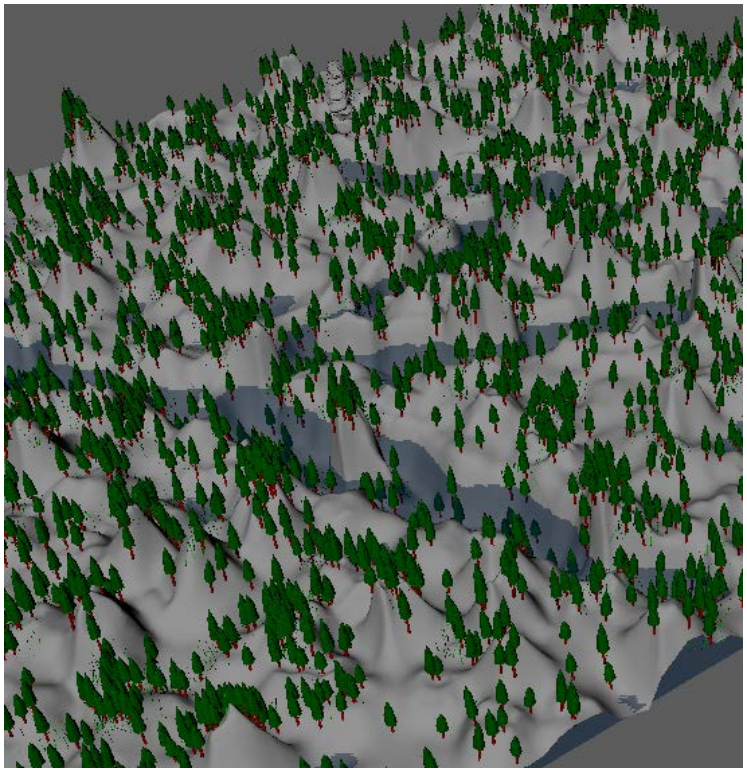

Рисунок 3.10 – Згенерована за допомогою створеного плагіну вода

Завершальним етапом стало текстурування поверхні. Як показано на рисунку 3.11, текстури були застосовані відповідно до висотної карти, що дозволило створити природні переходи між травою, камінням і землею.

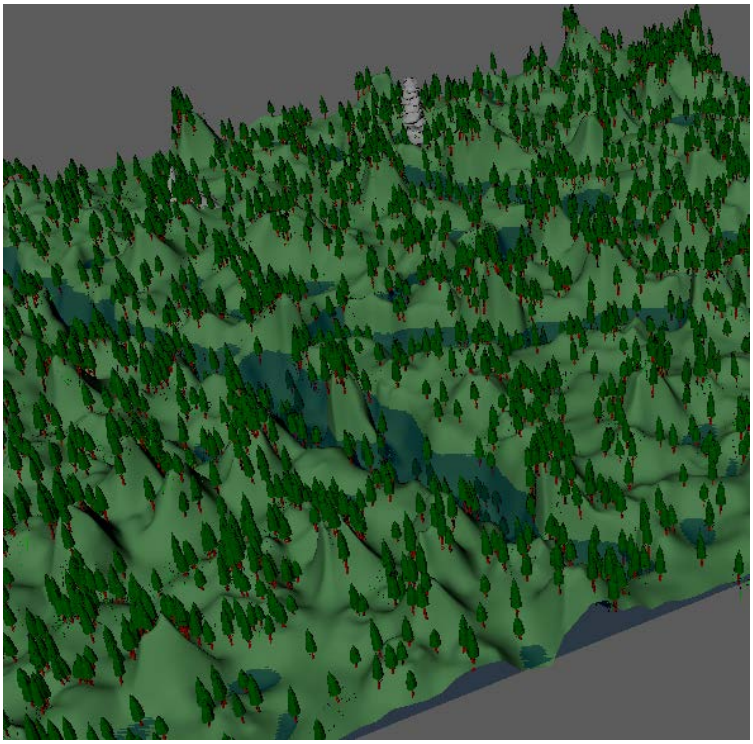

Рисунок 3.11 – Згенерована за допомогою створеного плагіну текстура ландшафту

Після проведення всіх тестів було підтверджено, що плагін працює стабільно, забезпечуючи генерацію стилізованих 3D-ландшафтів відповідно до заданих параметрів. Зокрема, перевірено:

– Готові пресети дозволяють швидко створювати унікальні ландшафти.

– Ручне налаштування параметрів забезпечує високу деталізацію та гнучкість у створенні складних сцен.

– Генерація всіх елементів ландшафту відбувається швидко, без значних затримок або збоїв.

Окрім основної функціональності, плагін також успішно пройшов тестування на сумісність із різними версіями Maya, що підтверджує його універсальність.

Розроблений плагін показав значний потенціал для подальшого вдосконалення, наприклад, додавання нових типів елементів (печери, скельні

арки), інтеграції алгоритмів автоматичного розміщення рослинності або вдосконалення текстурування.

3.4 Висновок до розділу 3

У третьому розділі проведено вибір мови програмування Python та середовища розробки Maya, обґрунтовано доцільність їх використання, а також вибір процедурних алгоритмів генерації (шум Перліна, шум Вороного та Midpoint Displacement), що є актуальним при створенні системи генерування стилізованих 3D-ландшафтів. Генератор ландшафтів проаналізовано відповідно до функціональних характеристик аналогів та проведено тестування його роботи, а також перевірку коректності комбінування різних типів рельєфу та їх візуалізації. Розроблена система повністю відповідає вимогам, забезпечуючи можливість створення різних стилізованих рельєфів з інтерактивним налаштуванням параметрів.

4 ЕКОНОМІЧНА ЧАСТИНА

Успішне впровадження науково-технічної розробки можливе лише за умови її відповідності сучасним вимогам науково-технічного прогресу та врахування економічних чинників. Важливою складовою цього процесу є оцінка економічної доцільності результатів науково-дослідної діяльності.

Магістерська кваліфікаційна робота на тему «Інформаційна технологія генерування стилізованих 3D-ландшафтів» відноситься до науково-технічних проектів з потенціалом комерціалізації. Прийняття рішення про вихід розробки на ринок може відбутися ще під час її реалізації, що створює умови для застосування отриманих результатів у практичній діяльності.

4.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту дослідження на тему «Інформаційна технологія генерування стилізованих 3D-ландшафтів» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Для проведення технологічного аудиту використано таблицю 4.1 в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу [22].

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
1	2	3	4	5	6
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах

Таблиця 4.1 – Продовження

1	2	3	4	5	6
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Таблиця 4.1 – Продовження

1	2	3	4	5	6
Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використо-вуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використо-вуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно зведені до таблиці 4.3. Експертами в опитуванні були: Дмитро Білик – інвестор та трейдер з п’ятирічним досвідом, Роман Бралатан – .NET developer, Єлізавета Голованова – Quality Assurance спеціаліст.

Таблиця 4.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Експерти		
	Дмитро Білик	Роман Бралатан	Єлізавета Голованова
	Бали, виставлені експертами:		
1. Технічна здійсненність концепції	3	4	3
2. Ринкові переваги (наявність аналогів)	4	3	2
3. Ринкові переваги (ціна продукту)	4	3	4
4. Ринкові переваги (технічні властивості)	3	2	2
5. Ринкові переваги (експлуатаційні витрати)	3	4	4
6. Ринкові перспективи (розмір ринку)	4	3	3
7. Ринкові перспективи (конкуренція)	4	4	4
8. Практична здійсненність (наявність фахівців)	2	1	2
9. Практична здійсненність (наявність фінансів)	2	2	3
10. Практична здійсненність (необхідність нових матеріалів)	2	4	3
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	3	4
Сума балів	$СБ_1$=39	$СБ_2$=37	$СБ_3$=38
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{39+37+38}{3} = 38$		

За результатами розрахунків, наведених в таблиці 4.3, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в таблиці 4.2.

Згідно проведених досліджень рівень комерційного потенціалу розробки, розрахований на основі висновків експертів склав 38 балів, що,

відповідно до таблиці 4.2, свідчить про рівень комерційного потенціалу розробки вище середнього.

На початковому етапі розробка може бути представлена шляхом завантаження програмного файлу в репозиторій GitHub з подальшим поширенням посилання та опису через професійні соціальні мережі, зокрема LinkedIn. Після отримання перших відгуків від користувачів планується вдосконалити рішення та розмістити його на спеціалізованому сервері.

Детальний опис розробки разом із доступом до сервера буде оформлено у вигляді статті, яку можна опублікувати на спеціалізованих платформах, таких як DOU.

Основна цільова аудиторія — 3D-художники, дизайнери ігор, а також розробники програмного забезпечення у розробці ігор.

4.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Інформаційна технологія генерування стилізованих 3D-ландшафтів», під час планування, обліку і обчислення собівартості науково-дослідної роботи групуються за певними статтями.

4.2.1 Витрати на оплату праці

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

Основна заробітна плата кожного з дослідників $З_о$ розраховуємо у відповідності до посадових окладів працівників, за формулою 4.1:

$$З_о = \sum_{i=1}^{k} \frac{M_{mi} \cdot t_i}{Т_р}, \quad (4.1)$$

де k – кількість посад дослідників, залучених до процесу досліджень;

M_{mi} – місячний посадовий оклад конкретного дослідника, грн;

t_i – кількість днів роботи конкретного дослідника, дн.;

$Т_р$ – середня кількість робочих днів в місяці, $Т_р$ = 21 дні.

$$З_о = 36000 \cdot \frac{25}{21} = 42857 \text{ } грн.$$

Проведені розрахунки зведено до таблиці 4.4.

Таблиця 4.4 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник проекту	36000	1666,6	25	42857
Інженер-програміст	20000	952,3	35	33333
Всього				76190

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників (Зр) за відповідними найменуваннями робіт НДР розраховуємо за формулою 4.2:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (4.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою 4.3:

$$C_i = \frac{M_m \cdot K_i \cdot K_c}{T_p \cdot t_{\text{зм}}}, \quad (4.3)$$

де M_m – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо M_m = 8000 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати;

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 дн;

$t_{\text{зм}}$ – тривалість зміни, год.

$$C_i = \frac{8000 \cdot 1 \cdot 1{,}65}{21 \cdot 8} = 78{,}6 \; грн.$$

$$З_{p1} = 78{,}6 \cdot 2 = 157{,}2 \; грн.$$

Таблиця 4.5 – Величина витрат на основну заробітню плату робітників

Найменування робіт	Тривалість, год.	Розряд роботи	Погодинна тарифна ставка, грн.	Величина оплати на робітника, грн.
Збір вимог	5	1	65,6	328
Проектування	10	3	79,5	795
Розробка	26	5	115,2	2995,2
Тестування	7	2	68,1	476,7
Реалізація	10	4	60,9	609
Всього				5203,9

Додаткова заробітна плата дослідників та робітників. Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою 4.4:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$$З_{дод} = (76190 + 5203{,}9) \cdot \frac{11}{100\%} = 8953{,}3 \text{ грн.}$$

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою 4.5:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (4.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймемо 22%.

$$З_н = (76190 + 5203{,}9 + 8953{,}3) \cdot \frac{22}{100\%} = 19876{,}384 \text{ грн.}$$

4.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою "Інформаційної технології

організації заходів". Витрати на матеріали (М), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою 4.6:

$$\text{М} = \sum_{j=1}^{n} H_j \cdot \text{Ц}_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot \text{Ц}_{\text{в}j}, \quad (4.6)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

Ц_j – вартість матеріалу j-го найменування, грн/кг;

K_j– коефіцієнт транспортних витрат, ($K_j = 1{,}1 \ldots 1{,}15$);

B_j – маса відходів j-го найменування, кг;

$\text{Ц}_{\text{в}j}$ – вартість відходів j-го найменування, грн/кг.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 шт, грн.	Норма витрат, шт.	Вартість витраченого матеріалу, грн.
Пачка паперу А4	216	1	180
Ручка	56	2	80
Флеш пам'ять USB Kingston DataTraveler Exodia Onyx 128GB	399	1	319
Всього			727
З врахуванням коефіцієнта транспортування			796,5

4.2.4 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою 4.5:

$$\text{А}_{\text{обл}} = \frac{\text{Ц}_{\text{б}}}{T_{\text{в}}} \cdot \frac{t_{\text{вик}}}{12}, \quad (4.7)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

K_c – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = \frac{68000}{2} \cdot \frac{2}{12} = 5666.6 \text{ грн.}$$

Таблиця 4.7 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн.
Ноутбук	68000	2	2	5666.6
Монітор	8500	2	1	354.17
Приміщення	250000	20	1	791,67
Всього				6782.44

4.2.5 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію $В_е$ розраховуємо за формулою 4.8:

$$В_е = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_е \cdot K_{впі}}{n_i}, \quad (4.8)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е$ = 10 грн;

$K_{впі}$ – коефіцієнт, що враховує використання потужності, $K_{впі} < 1$;

n_i – коефіцієнт корисної дії обладнання, $n_i < 1$.

$$В_е = \frac{0{,}25 \cdot 250{,}0 \cdot 10 \cdot 0{,}5}{0{,}8} = 390{,}62 \text{ грн.}$$

4.2.6 Службові відрядження

До статті «Службові відрядження» належать витрати на відрядження штатних працівників, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень. Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 4.9:

$$З_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.9)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження». Приймемо 22%.

$$З_{св} = (76190 + 5203{,}9) \cdot \frac{22}{100\%} = 17906.66 \text{ грн.}$$

4.2.7 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою 4.10:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.10)$$

де $I_в$ – норма нарахування за статтею «Інші витрати». Приймемо 60%.

$$I_в = (76190 + 5203{,}9) \cdot \frac{50}{100\%} = 48836.34 \text{ грн.}$$

4.2.8 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науковотехнічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою 4.11:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \qquad (4.11)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати». Приймемо 100%.

$$Н_{нзв} = (76190 + 5203{,}9) \cdot \frac{100}{100\%} = 81323.3 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи на тему «Інформаційна технологія генерування стилізованих 3D-ландшафтів», розраховуємо як суму всіх попередніх статей витрат за формулою 4.12:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв} \quad (4.12)$$

$$В_{заг} = 76190 + 5203{,}9 + 8953{,}3 + 19876{,}384 + 796{,}5 + 5666.6 + 6782.44 + 390{,}62 + 17906.66 + 48836.34 + 81323.3 = 271926.044 \text{ грн.}$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою 4.13:

$$ЗВ = \frac{В_{заг}}{n}, \quad (4.11)$$

де n – коефіцієнт, який характеризує етап (стадію) виконання науководослідної роботи. Приймемо 0,7.

$$ЗВ = \frac{271926.044}{0{,}7} = 388465 \textit{ грн.}$$

4.3 Розрахунок економічної ефективності науково-технічної розробки

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи.

Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із трьох років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою 4.12:

$$\Delta П_{\text{і}} = \sum_1^n (\Delta Ц_{\text{о}} * N * Ц_{\text{о}} * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\upsilon}{100}\right), \quad (4.12)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_{\text{о}}$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

ρ – коефіцієнт, який враховує рентабельність продукту. ρ = 0,25;

υ – ставка податку на прибуток. У 2024 році – 18%.

Збільшення чистого прибутку 1-го року:

$$\Delta П_1 = (88 \cdot 4500 + 338 \cdot 5000) \cdot 0{,}83 \cdot 0{,}4 \cdot \left(1 - \frac{0{,}18}{100}\%\right)$$

$= 1550060{,}5$ грн.

Збільшення чистого прибутку 2-го року:

$$\Delta П_2 = \text{П2}\ (88 \cdot 4500 + 338 \cdot (5000 + 8000)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1 - 0{,}18/100\%)$$

$= 2288109{,}14$ грн.

$\Delta П_3 = (88 \cdot 4500 + 338 \cdot (500 + 8000 + 10000)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1 - 0{,}18 /100\%) = 3210662{,}44$ грн.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \tag{4.13}$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо 2;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 3282746,34 грн.

$$PV = 2 \cdot 388465 = 776930 \text{ грн.}$$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$\text{Е}_{\text{абс}} = (\text{ПП} - PV), \tag{4.14}$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 3282746,34 грн;

$$\text{ПП} = \sum_1^T \frac{\Delta \text{П}_\text{і}}{(1+\tau)^t}\ , \tag{4.13}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$Е_{абс} = 388465 - 77693 = 310772 \text{ грн.}$$

Оскільки $Е_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$. Для цього користуються формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1 , \quad (4.14)$$

де $Т_ж$ – життєвий цикл наукової розробки, роки.

$$Е_в = \sqrt[3]{1 + \frac{310772}{77693}} - 1 = 1{,}03.$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f , \quad (4.15)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,14…0,2);

f – показник, що характеризує ризикованість вкладень, приймемо 0,25.

$$\tau_{min} = 0{,}1 + 0{,}25 = 0{,}35.$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$\mathrm{T_{ок}} = \frac{1}{\mathrm{E_в}}. \quad (4.16)$$

$$\mathrm{T_{ок}} = \frac{1}{1{,}03} = 0{,}97 \text{ (роки)}.$$

$\mathrm{T_{ок}} \leq 3$ років, що свідчить про комерційну привабливість науковотехнічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

4.5 Висновок до розділу 4

Згідно з проведеними дослідженнями, рівень комерційного потенціалу розробки «Інформаційна технологія генерування стилізованих 3D-ландшафтів» становить 38 балів, що свідчить про її вище середню комерційну значущість. За результатами оцінки конкурентоспроможності, узагальнений коефіцієнт показує, що ця науково-технічна розробка перевершує існуючі аналоги приблизно на 18%. Крім того, термін окупності складає близько одного року, що значно менше трирічного порогу, і підтверджує її інвестиційну привабливість. Це може стимулювати потенційних інвесторів до фінансування впровадження та виведення розробки на ринок. Отже, проведення науково-дослідної роботи на тему «Інформаційна технологія генерування стилізованих 3D-ландшафтів» є доцільним.

ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи розв’язано задачу розробки інформаційної технології для генерування стилізованих 3D-ландшафтів.

Під час виконання поставленої задачі було виконано аналіз предметної області та об’єкту дослідження, зокрема стилізованих 3D-ландшафтів та методів їх створення.

Також було проведено аналіз існуючих методів процедурної генерації ландшафтів та визначити їх особливості і можливості для застосування у створенні стилізованих моделей.

Розроблено модель інформаційної технології генерування стилізованих 3D-ландшафтів, що включає вибір алгоритмів та методів генерації рельєфу, текстурування та інтеграції різних типів ландшафтів.

Також було розроблено алгоритм генерування стилізованих 3D-ландшафтів, який буде гнучким та налаштовуваним для створення різних видів ландшафтів (скелі, каньйони, гори, пагорби тощо).

Виконано програмну реалізацію алгоритму в середовищі Maya з використанням Python, створити інтерфейс для вибору типів ландшафтів та їх параметрів.

Також було проведено тестування розробленої інформаційної технології для оцінки якості генерації стилізованих 3D-ландшафтів, зокрема візуальної привабливості та продуктивності генерації.

Проаналізовано результати тестування та розробити рекомендації щодо подальшого вдосконалення технології.

У результаті виконання останнього розділу проведено розрахунок економічної ефективності науково-технічної розробки за можливого її застосування в індустрії комп’ютерної графіки та анімації.

Готовий плагін може генерувати різні типи ландшафту: скелі, каньйони, гори, пагорби, дюни, луги, степи; розміри ландшафту є налаштовуваними: 500 – 4000 м; інтенсивність шумів варіюється від 10 до 100 пунктів; комбінуються різні типи шумів: Перлін, Вороний, Midpoint Displacement; кількість полігонів можна налаштувати; згладжування поверхні можна зробити в залежності від ступею стилізації від 1 до 100%.

Розроблена інформаційна технологія та програмне забезпечення забезпечують ефективне генерування стилізованих 3D-ландшафтів, відповідаючи сучасним вимогам індустрії. Отримані результати свідчать про перспективність подальшого розвитку цього напрямку.

Програмна реалізація додатку відбувалась у середовищі створення 3D-графіки Autodesk Maya й за допомогою мови програмування Python.

Мету роботи досягнуто та поставлені задачі виконані у повному обсязі.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Кушнір О. А., Іванчук Я. В./ Методи генерування стилізованих 3D-ландшафтів // Тези LII науково-технічної конференції підрозділів ВНТУ (НТКП ВНТУ-2024). [Електронний ресурс] - Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/22317
2. Методичні вказівки до виконання магістерських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. К. Колесницький. – Вінниця : ВНТУ, 2023. – 58 с.
3. Визначення терміну «Процедурна генерація» [Електрон. ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/Процедурна_генерація.
4. Heightmaps in Unity [Electronic resource]. – Access mode: https://docs.unity3d.com/Manual/terrain-Heightmaps.html.
5. Working with Terrain in Unity [Electronic resource]. – Access mode: https://docs.unity3d.com/Manual/terrain-Heightmaps.html.
6. Working with Meshes in unity [Electronic resource]. – Access mode: https://docs.unity3d.com/ScriptReference/Mesh.html.
7. Shaker, N. Procedural Content Generation in Games / N. Shaker, J. Togelius, M. J. Nelson. - Springer, 18.10.2016. - 218 p.
8. Development report about using procedural generation in The Witcher 3 [Electronic resource]. – Access mode: https://www.gdcvault.com/play/1020197/ Landscape-Creation-and-Rendering-in.
9. Development report about using procedural generation in Horizon Zero Down [Electronic resource]. – Access mode: https://www.guerrilla-games.com/read/gpu-based-procedural-placement-in-horizon-zero-dawn.
10. Giffard, P. V. Perlin Noise / P.V. Giffard. - Tort, 2011. – 72 p.
11. Golossary for Perlin noise [Electronic resource]. – Access mode: http://libnoise.sourceforge.net/glossary/.

12. Sunil Arya, Sunil; Malamatos, Theocharis; Mount, David M. (2002). "Space-efficient approximate Voronoi diagrams". Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. P. 721–730.
13. Liberti, Leo; Lavor, Carlile (2017), Euclidean Distance Geometry: An Introduction, Springer Undergraduate Texts in Mathematics and Technology, Springer.
14. Krause, E. F. Taxicab Geometry [Electronic resource] / E. F. Krause. – Courier Corporation, 01.01.1986. – 88 p.
15. Theory of Midpoint Displacement algorithm [Electronic resource]. – Access mode: https://www.sfu.ca/~rpyke/335/projects/tsai/report1.htm.
16. Implementation of Midpoint Displacement [Electronic resource]. – Access mode: https://bitesofcode.wordpress.com/2016/12/23/landscape- generation-using-midpoint-displacement/.
17. Визначення терміну «Фрактал» [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Фрактал.
18. Hamilton, Naomi (5 August 2008). "The A-Z of Programming Languages: Python". Computerworld. Archived from the original on 29 December 2008. Retrieved 31 March 2010.
19. Downey, Allen B. (May 2012). *Think Python: How to Think Like a Computer Scientist* (version 1.6.6 ed.)
20. Official website Autodesk Maya [Electronic resource]. – Access mode: https://www.autodesk.com/maya
21. Сучасні інформаційні технології у сфері штучного інтелекту : електронний навчальний посібник комбінованого (локального та мережного) використання / Яровий А. А., Крилик Л. В. , Козловський А. В. – Вінниця : ВНТУ, 2023. – 145 с.
22. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В.О. Козловський, О.Й. Лесько, В.В. Кавецький. Вінниця : ВНТУ, 2021.

Додаток А (обов'язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна технологія генерування стилізованих 3D-ландшафтів

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Turnitin

Оригінальність 96,00% Схожість 4,00%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Ознайомлені з повним звітом подібності, який був згенерований системою Turnitin щодо роботи.

Автор роботи Кушнір О.А.

Керівник роботи Іванчук Я.В.

Опис прийнятого рішення

Магістерську кваліфікаційну роботу допущено до захисту.

Особа, відповідальна за перевірку Озеранський В.С.

Додаток Б (обов'язковий)

Лістинг програми

```
import maya.cmds as cmds
import random
import math
import MASH.api as mapi

   #Create Trees
def mashTrees(winID, numberTrees, scaleTrees, *pArgs):

   Landscape01="Landscape01"
   Tree()

   cmds.select("TreeOriginal")
   cmds.scale(scaleTrees,scaleTrees,scaleTrees)

   # create MASH network
   mashNetwork = mapi.Network()
   mashNetwork.createNetwork(name="Trees")
   mashNetwork.meshDistribute(Landscape01)

   # set MASH points
   mashNetwork.setPointCount(numberTrees)
   cmds.setAttr ("Trees_Distribute.calcRotation", 0)
   cmds.setAttr ("Trees_Distribute.distanceAlongNormal", 8)

   mashNetwork.addNode("MASH_Random")
   cmds.setAttr ("Trees_Random.rotationY", 360)
   cmds.setAttr ("Trees_Random.rotationX", 5)
   cmds.setAttr ("Trees_Random.rotationZ", 5)
   cmds.setAttr ("Trees_Random.scaleX", 0.2)
   cmds.setAttr ("Trees_Random.scaleY", 1)
   cmds.setAttr ("Trees_Random.scaleZ", 0.2)

   nodes = mashNetwork.getAllNodesInNetwork()

   for node in nodes:
      mashNode = mapi.Node(node)
      falloffs = mashNode.getFalloffs()

   # Create grass
def mashGrass(winID, numberGrass, scaleGrass, *pArgs):

   Landscape01="Landscape01"
   grass()
   cmds.select("grassBlade")
   cmds.scale(scaleGrass,scaleGrass,scaleGrass)
   # create MASH network
```

```
    mashNetwork = mapi.Network()
    mashNetwork.createNetwork(name="Grass")
    mashNetwork.meshDistribute(Landscape01)
    cmds.setAttr ("Grass_Distribute.meshType", 1)
    cmds.setAttr ("Grass_Distribute.pointCount", 400)

    # set MASH points
    mashNetwork.setPointCount(numberGrass)
    cmds.setAttr ("Grass_Distribute.calcRotation", 0)

    # add nodes
    mashNetwork.addNode("MASH_World")
    cmds.setAttr ("Grass_World.randomPointsPerCluster", 10)
    cmds.setAttr ("Grass_World.clusterMode", 4)
    cmds.setAttr ("Grass_World.pointsPerCluster", 14)
    cmds.setAttr ("Grass_World.radius", 4)
    cmds.setAttr ("Grass_World.clusterRadius", 3)

    mashNetwork.addNode("MASH_Random")
    cmds.setAttr ("Grass_Random.positionZ", 20)
    cmds.setAttr ("Grass_Random.positionX", 20)
    cmds.setAttr ("Grass_Random.rotationX", 20)
    cmds.setAttr ("Grass_Random.rotationY", 360)
    cmds.setAttr ("Grass_Random.rotationZ", 20)
    cmds.setAttr ("Grass_Random.scaleX", 10)
    cmds.setAttr ("Grass_Random.scaleY", 25)
    cmds.setAttr ("Grass_Random.scaleZ", 10)

    nodes = mashNetwork.getAllNodesInNetwork()

    for node in nodes:
        mashNode = mapi.Node(node)
        falloffs = mashNode.getFalloffs()

    #tree group
def treeGroup():
    newObj = Tree()
    cmds.move(random.uniform(-50,50),random.random()/2,random.uniform(-
50,50),r=True,os=True,wd=True)
    cmds.rotate(0, random.uniform(0,360), 0, r=True)
    cmds.select(newObj)

    #Create river
def createRiver(winID, *pArgs):
    Landscape01="Landscape01"
    percent = 0.002
    vertex = []
    cmds.select(Landscape01)

    for obj in cmds.ls(sl=1, long=1):
        indexes = list(range(cmds.polyEvaluate(obj, vertex=True)))
        random.shuffle(indexes)
```

```
        indexes = indexes[:int(4)]
        for i in range(len(indexes)):
            indexes[i] = obj+'.vtx['+str(indexes[i])+']'
        vertex.extend(indexes)
    cmds.select(vertex, r=1)
    a=indexes[0]
    b=indexes[1]
    c=indexes[2]
    d=indexes[3]

    a=a.strip("|Landscape01.vtx[]")
    b=b.strip("|Landscape01.vtx[]")
    c=c.strip("|Landscape01.vtx[]")
    d=d.strip("|Landscape01.vtx[]")

    a=int(a)
    b=int(b)
    c=int(c)
    d=int(d)

    cmds.polySelect( 'Landscape01', shortestEdgePath=(a, b))
    cmds.polySelect( 'Landscape01', shortestEdgePath=(b, c))
    cmds.polySelect( 'Landscape01', shortestEdgePath=(c, d))

    cmds.ConvertSelectionToFaces()
    cmds.move(0.0, -80, 0.0, r=True)

    #Set Colour
def setColour(winID, *pArgs):

    rgb = cmds.colorSliderGrp( 'polygonColour', query = True, rgbValue = True )
    myShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( myShader + '.color', rgb[ 0 ], rgb[ 1 ], rgb[ 2 ], type = 'double3' )
    cmds.select('Landscape01')
    cmds.hyperShade(assign = myShader)

    #Creates land
def createLand (subControl, sizeControl, heightControl ):

    cmds.select(all=True)
    cmds.delete()

    landsub = subControl
    landsize = sizeControl
    maxheight = heightControl
    land = cmds.polyPlane( sx=subControl, sy=subControl, w=landsize, h=landsize)
    vtxCount = list(range(cmds.polyEvaluate(v=True)))
    random.shuffle(vtxCount)
    values = [random.triangular(0,1,0) for i in range(10)]
    values_count = len(values)
    optimize_setter = []
```

```
    for x in vtxCount:
        mod = x % values_count
        optimize_setter += [float(0),values[mod-1]*maxheight,float(0)]
    cmds.setAttr('pPlane1.vtx[:]', *optimize_setter)
    cmds.select(land[0])
    cmds.rename ("Landscape01")

        #Create Terrain
def actionProc(winID, subControl, sizeControl, heightControl, *pArgs):

    createLand (subControl, sizeControl, heightControl)

        #Close UI
def cancelProc(winID, *pArgs):
    cmds.deleteUI(winID)

        # Smoothing
def smoothing(winID, *pArgs):
    cmds.select("Landscape01")
    cmds.polySmooth(c=1,dv=1,kb=True,ro=1)
    cmds.pause( sec=1 )
    return True
    cmds.select(clear=True)

    #Hills preset
def hillPreset (winID, subControl, sizeControl, heightControl, hillHeight, ditchDepth,
rock1amountControl, rock2amountControl, rockscaleControl, *pArgs):

    createLand (15, sizeControl, heightControl)
    for i in range (6):
        createHill(winID, 60)
    for j in range (4):
        createDetail(winID, 60)
    for k in range (4):
        createDitch(winID, 30)

    smoothing(winID)
    rockSpawn(winID, 25, 25, 0.5, 0)
    rockSpawn(winID, 25, 25, 0.8, 0)
    mashTrees(winID, 400, 2.0)
    hillShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( hillShader + '.color', 0.13, 0.25, 0.057, type = 'double3' )
    cmds.select('Landscape01')
    cmds.hyperShade(assign = hillShader)
    cmds.select(clear=True)

    #Plains preset
def plainPreset (winID, subControl, sizeControl, heightControl, hillHeight, ditchDepth,
rock1amountControl, rock2amountControl, rockscaleControl, *pArgs):

    createLand (30, sizeControl, heightControl)
    for i in range (12):
```

```
        createHill(winID, 20)

    smoothing(winID)
    mashGrass(winID, 200, 9, 30, 3, 3)
    rockSpawn(winID, 50, 50, 1, 0)
    rockSpawn(winID, 50, 50, 0.5, 0)
    plainShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( plainShader + '.color', 0.3, 0.42, 0.0, type = 'double3' )
    cmds.select('Landscape01')
    cmds.hyperShade(assign = plainShader)
    cmds.select(clear=True)

    #Forest preset
def forestPreset (winID, subControl, sizeControl, heightControl, hillHeight, ditchDepth,
rock1amountControl, rock2amountControl, rockscaleControl, *pArgs):

    createLand (30, sizeControl, heightControl)
    for i in range (6):
        createHill(winID, 40)

    smoothing(winID)
    mashTrees(winID, 3000, 1.5)

    forestShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( forestShader + '.color', 0.2, 0.6, 0.06, type = 'double3' )
    cmds.select('Landscape01')
    cmds.hyperShade(assign = forestShader)
    cmds.select(clear=True)

    #Mountains preset
def mountainPreset (winID, subControl, sizeControl, heightControl, mountainHeight, hillHeight,
ditchDepth, rock1amountControl, rock2amountControl, rockscaleControl, *pArgs):

    createLand (25, sizeControl, heightControl)
    for i in range (4):
        createMountins(winID, 60)
    for j in range (4):
        createHill(winID, 60)
    for k in range (4):
        createDetail(winID, 30)

    smoothing(winID)
    rockSpawn(winID, 50, 50, 1, 0)
    rockSpawn(winID, 50, 50, 0.5, 0)
    cmds.select(clear=True)

    #Lakes preset
def riverPreset (winID, waterHeight, subControl, sizeControl, heightControl, mountainHeight,
hillHeight, rock1amountControl, rock2amountControl, rockscaleControl, *pArgs):

    createLand (25, sizeControl, heightControl)
    for i in range (2):
```

```
        createMountins(winID, mountainHeight)
    for j in range (6):
        createHill(winID, 60)
    for k in range (2):
        createDetail(winID, 30)

    smoothing(winID)
    createRiver(winID)
    for k in range (2):
        createDitch(winID, 60)
    mashTrees(winID, 400, 1)
    rockSpawn(winID, 50, 50, 1, 0)
    rockSpawn(winID, 50, 50, 0.5, 0)
    Water(winID, 20)
    riverShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( riverShader + '.color', 0.2, 0.4, 0.03, type = 'double3' )
    cmds.select('Landscape01')
    cmds.hyperShade(assign = riverShader)
    cmds.select(clear=True)

    #Desert preset
def desertPreset (winID, subControl, sizeControl, heightControl, mountainHeight, hillHeight,
rock1amountControl, rock2amountControl, rockscaleControl,  *pArgs):

    createLand (25, sizeControl, heightControl)

    for j in range (8):
        createHill(winID, 30)
    smoothing(winID)
    rockSpawn(winID, 20, 20, 5, 0)
    rockSpawn(winID, 40, 40, 2, 0)
    rockSpawn(winID, 30, 30, 3, 0)
    desertShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( desertShader + '.color', 0.67, 0.44, 0.18, type = 'double3' )
    cmds.select('Landscape01')
    cmds.hyperShade(assign = desertShader)
    cmds.select(clear=True)

    #Canyons preset
def canyonsPreset (winID, subControl, sizeControl, heightControl, mountainHeight, hillHeight,
rock1amountControl, rock2amountControl, rockscaleControl,  *pArgs):

    createLand (35, sizeControl, heightControl)

    for k in range (5):
        createDetail(winID, 60)

    smoothing(winID)
    createRiver(winID)
    createRiver(winID)
    createRiver(winID)
    createRiver(winID)
```

```
    rockSpawn(winID, 50, 50, 1, 0)
    rockSpawn(winID, 50, 50, 2, 0)
    cmds.select(clear=True)

    #Islands preset
def islandsPreset (winID, waterHeight, subControl, sizeControl, heightControl, mountainHeight,
hillHeight, rock1amountControl, rock2amountControl, rockscaleControl, *pArgs):

    createLand (25, sizeControl, heightControl)
    for i in range (4):
        createMountins(winID, 20)
    for j in range (8):
        createHill(winID, 60)
    for k in range (2):
        createDetail(winID, 30)

    smoothing(winID)
    Water(winID, 120)
    islandShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( islandShader + '.color', 0.2, 0.4, 0.03, type = 'double3' )
    cmds.select('Landscape01')
    cmds.hyperShade(assign = islandShader)
    cmds.select(clear=True)

    #Create Water
def Water(winID, waterHeight, *pArgs):

    if cmds.objExists('water'):
        cmds.delete('water')
    waterShader = cmds.shadingNode( 'aiStandardSurface', asShader = True)
    cmds.setAttr( waterShader + '.baseColor', 0.0, 0.104, 0.34, type = 'double3' )
    cmds.setAttr ( waterShader + '.specularRoughness',  0.027436)
    cmds.setAttr ( waterShader + '.transmission', 0.45)

    cmds.polyPlane(w=2000,h=2000,name='water')
    cmds.ls('water*')

    cmds.hyperShade(assign = waterShader)
    cmds.select('water')
    cmds.move (0, waterHeight, 0, a=True)

    #Creates a rock
def Rock():

    #Make rock
    Cube=cmds.polyCube(h=8,w=8,d=8,sx=3,sy=3,sz=3,name='RockOriginal')
    cubeFaces = (Cube[0]+'.f[25]',Cube[0]+'.f[37]',Cube[0]+'.f[1]',Cube[0]+'.f[46]')
    #Make adjust
    for i in range (1):
        cmds.select(cubeFaces)
```

```
        cmds.scale
(random.uniform(1.5,2.5),random.uniform(1,1.2),random.uniform(1.5,2.5),r=True)

    cubeFaces2 = (Cube[0]+'.f[45]',Cube[0]+'.f[0]',Cube[0]+'.f[38]',Cube[0]+'.f[36]')
    #Make adjust
    for i in range (1):
        cmds.select(cubeFaces2)
        cmds.scale
(random.uniform(0.5,1.5),random.uniform(1,1.2),random.uniform(0.5,1.5),r=True)

    cubeFaces3 = (Cube[0]+'.f[43]',Cube[0]+'.f[19]',Cube[0]+'.f[7]',Cube[0]+'.f[52]')
    #Make adjust
    for i in range (1):
        cmds.select(cubeFaces3)
        cmds.scale (random.uniform(1.5,2),random.uniform(0.5,2.2),random.uniform(1.5,2),r=True)

    cmds.select(Cube[0]+'.f[13]')
    cmds.move (1,random.uniform(2.5,5.2),1,r=True)
    cmds.select(Cube[0]+'.f[31]')
    cmds.move (1,random.uniform(-2.5,-5.2),1,r=True)

    cmds.select(cl=True)
    cmds.polySelect('RockOriginal',el=22)
    cmds.move (0,random.uniform(1.1,1.8),0,r=True)


cmds.select(Cube[0]+'.vtx[36]',Cube[0]+'.vtx[32]',Cube[0]+'.vtx[39]',Cube[0]+'.vtx[35]',Cube[0]+
'.vtx[3]',Cube[0]+'.vtx[7]',Cube[0]+'.vtx[0]',Cube[0]+'.vtx[4]')

    cmds.scale (random.uniform(1.4,2.6),random.uniform(1,2),random.uniform(1.4,2.6),r=True)
    cmds.select(Cube)
    cmds.polySmooth(c=1,dv=1,kb=True,ro=1)
    rockShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( rockShader + '.color', 0.185, 0.185, 0.185, type = 'double3' )
    cmds.select(Cube)
    #cmds.polySmooth(c=1,dv=2,kb=True,ro=1)
    addNoise(2)
    cmds.move(random.uniform(-100,100),0,random.uniform(-100,100))
    cmds.hyperShade(assign = rockShader)

def smallRock():

    randomFloat = random.uniform(0.01,2.5)
    #Make rock
    Cube=cmds.polyCube(h=8,w=8,d=8,sx=3,sy=3,sz=3,name='RockOriginal')
    cubeFaces = (Cube[0]+'.f[25]',Cube[0]+'.f[37]',Cube[0]+'.f[1]',Cube[0]+'.f[46]')
    #Make adjust
    cmds.select(cubeFaces)
    cmds.scale (random.uniform(1.5,2.5),random.uniform(1,1.2),random.uniform(1.5,2.5),r=True)

    cubeFaces2 = (Cube[0]+'.f[45]',Cube[0]+'.f[0]',Cube[0]+'.f[38]',Cube[0]+'.f[36]')
    cmds.select(cubeFaces2)
```

```
    cmds.scale (random.uniform(0.5,1.5),random.uniform(1,1.2),random.uniform(0.5,1.5),r=True)

    cubeFaces3 = (Cube[0]+'.f[43]',Cube[0]+'.f[19]',Cube[0]+'.f[7]',Cube[0]+'.f[52]')
    cmds.select(cubeFaces3)
    cmds.scale (random.uniform(1.5,2),random.uniform(0.5,2.2),random.uniform(1.5,2),r=True)

    cmds.select(Cube[0]+'.f[13]')
    cmds.move (1,random.uniform(2.5,5.2),1,r=True)
    cmds.select(Cube[0]+'.f[31]')
    cmds.move (1,random.uniform(-2.5,-5.2),1,r=True)

    cmds.select(cl=True)
    cmds.polySelect('RockOriginal',el=22)
    cmds.move (0,random.uniform(1.1,1.8),0,r=True)


cmds.select(Cube[0]+'.vtx[36]',Cube[0]+'.vtx[32]',Cube[0]+'.vtx[39]',Cube[0]+'.vtx[35]',Cube[0]+
'.vtx[3]',Cube[0]+'.vtx[7]',Cube[0]+'.vtx[0]',Cube[0]+'.vtx[4]')
    cmds.scale (random.uniform(1.4,2.6),random.uniform(1,2),random.uniform(1.4,2.6),r=True)

    cmds.select(Cube)
    cmds.scale
(random.uniform(0.5,2.0),random.uniform(0.5,2.0),random.uniform(0.5,2.0),a=True)
    cmds.scale (randomFloat,randomFloat,randomFloat,r=True)
    cmds.polySmooth(c=1,dv=2,kb=True,ro=1)
    addNoise(1)
    cmds.select('RockOriginal', r=True )

cmds.rotate(random.uniform(10,120),random.uniform(10,120),random.uniform(10,1200),a=True)
    cmds.move(random.uniform(-100,100),0,random.uniform(-100,100))
    cmds.rename('Rock_Single_01')
    #cmds.group( em=True, name='Rock_Singles')
    #cmds.group('Rock_Single_*', name='Rock_Singles', a=True)
    #cmds.select('Rock_Singles', r=True )
    #cmds.rename('Rock_Single_Group')

    #Creates a tree
def Tree():

    leafShader = cmds.shadingNode( 'blinn', asShader = True)
    cmds.setAttr( leafShader + '.color', 0.0440, 0.203, 0, type = 'double3' )

    cmds.setAttr ( leafShader + '.eccentricity', 0.25)
    cmds.setAttr ( leafShader + '.specularRollOff', 0.3)

    barkShader = cmds.shadingNode( 'blinn', asShader = True)
    cmds.setAttr( barkShader + '.color', 0.4, 0.08, 0, type = 'double3' )
    cmds.setAttr ( barkShader + '.eccentricity', 0.45)
    cmds.setAttr ( barkShader + '.specularRollOff', 0.1)

    randomFloat = random.uniform(1,1.2)
    randomFloat2 = random.uniform(0.5,0.8)
```

```
    randomFloat3 = random.uniform(1.3,1.5)
    randomFloat4 = random.uniform(-0.2,-0.4)

    randomFloat5 = random.uniform(0.6,1)
    randomFloat6 = random.uniform(0.6,1)
    randomFloat7 = random.uniform(0.6,1)
    randomFloat8 = random.uniform(0.6,1)

    #Make Trunk
    ''' Makes the tree trunk '''
    Cylinder=cmds.polyCylinder (h=8,r=0.5,sx=8,sy=3,name='TreeOriginal')

    #Make Bush
    cmds.select(Cylinder[0]+'.f[25]')

cmds.polyExtrudeFacet(s=(random.uniform(1,1.2),random.uniform(0.8,1.1),random.uniform(1,1.
2)), t=(0, randomFloat2, 0))
    cmds.scale (2,randomFloat,2)
    cmds.polyExtrudeFacet(s=(randomFloat,randomFloat,randomFloat), t=(0, randomFloat2, 0))
    cmds.scale (randomFloat3,randomFloat,randomFloat3)

    cmds.polyExtrudeFacet(s=(randomFloat,randomFloat,randomFloat), t=(0, randomFloat2, 0))
    cmds.scale (randomFloat3,randomFloat,randomFloat3)
    cmds.polyExtrudeFacet(s=(randomFloat,randomFloat,randomFloat), t=(0, randomFloat2, 0))
    cmds.scale (1.1,randomFloat,1.1)

    #Make Tree Top
    ''' Makes the tree top '''
    cmds.polyExtrudeFacet(s=(1,1,1), t=(0, random.uniform(1.2,2.6), 0))
    cmds.scale (random.uniform(0.8,0.9),1,random.uniform(0.8,0.9))
    cmds.polyExtrudeFacet(s=(1,1,1), t=(0, random.uniform(1.2,1.6), 0))
    cmds.scale (random.uniform(0.7,0.8),1,random.uniform(0.7,0.8))
    cmds.polyExtrudeFacet(s=(1,1,1), t=(0, random.uniform(1.2,1.6), 0))
    cmds.scale (random.uniform(0.6,0.7),1,random.uniform(0.6,0.7))
    cmds.polyExtrudeFacet(s=(1,1,1), t=(0, 2, 0))
    cmds.scale (0.05,1,0.05)

    #Adjust Trunk
    ''' Adjusts tree trunk '''
    cmds.select(Cylinder[0]+'.e[8:15]')
    cmds.scale (1.4,randomFloat3,1.4)
    cmds.select(Cylinder[0]+'.e[0:7]')
    cmds.scale (1.6,randomFloat3,1.6)

    #Make bottom Edge
    ''' Adjusts bottom edge '''
    cmds.select(Cylinder[0]+'.f[42:49]')
    cmds.move (0,-2,0,r=True,os=True,wd=True)
    cmds.scale (randomFloat3,1,randomFloat3)

    #Extra Branch 1
    ''' Makes a branch '''
```

```
    cmds.select(Cylinder[0]+'.f[21]')
    cmds.polyExtrudeFacet(s=(1,1,1), t=(0, 0, 0))
    cmds.scale (0.3,0.1,0.3, r=True, ls=True)
    cmds.polyCircularize(divisions=1)
    cmds.select(Cylinder[0]+'.f[21]')
    cmds.move(0.2,2,0.4,r=True,os=True,wd=True)
    cmds.rotate(0, -35, 0, r=True, cs=True)

    for i in range (2):
        cmds.polyExtrudeFacet (s = (1,1,1), t=(0, 0, 0))
        cmds.rotate(0, -15, 0, r=True, cs=True)
        cmds.move (random.uniform(0.4,1.2), random.uniform(0.8,1.8), random.uniform(0.4,1.2),
r=True, os=True, wd=True)
        cmds.scale (random.uniform(0.8,1.5),random.uniform(0.8,1.5),random.uniform(0.8,1.5),
r=True, cs=True)

    #Extra Branch 2
    ''' Makes a branch '''
    cmds.select(Cylinder[0]+'.f[11]')
    cmds.polyExtrudeFacet(s=(1,1,1), t=(0, 0, 0))
    cmds.scale (0.3,0.1,0.3, r=True, ls=True)
    cmds.polyCircularize(divisions=1)
    cmds.select(Cylinder[0]+'.f[11]')
    cmds.move(-0.5, 0, 0.2, r=True, os=True, wd=True)
    cmds.rotate(0, -35, 0, relative=True, componentSpace=True)

    for i in range (2):
        cmds.polyExtrudeFacet (s = (1,1,1), t=(0, 0, 0))
        cmds.rotate(0, -15, 0, r=True, cs=True)
        cmds.move (-random.uniform(0.4,1.2), random.uniform(0.8,1.1), random.uniform(0.4,1.2),
r=True, os=True, wd=True)
        cmds.scale (random.uniform(0.8,1.1),random.uniform(0.8,1.1),random.uniform(0.8,1.1),
r=True, cs=True)

    #Extra Branch 3
    ''' Makes a branch '''
    cmds.select(Cylinder[0]+'.f[15]')
    cmds.polyExtrudeFacet(s=(1,1,1), t=(0, 0, 0))
    cmds.scale (0.3,0.1,0.3, r=True, ls=True)
    cmds.polyCircularize(divisions=1)
    cmds.select(Cylinder[0]+'.f[15]')
    cmds.move(0.5, 0, -0.2, r=True, os=True, wd=True)
    cmds.rotate(0, -35, 0, relative=True, componentSpace=True)

    for i in range (1):
        cmds.polyExtrudeFacet (s = (1,1,1), t=(0, 0, 0))
        cmds.rotate(0, -15, 0, r=True, cs=True)
        cmds.move (random.uniform(0.5,0.6), random.uniform(0.8,1.2), random.uniform(-0.2,-0.5),
r=True, os=True, wd=True)
        cmds.scale (random.uniform(0.8,1.5),random.uniform(0.8,1.5),random.uniform(0.8,1.5),
r=True, cs=True)
```

```
    #Extra Branch 4
    ''' Makes a branch '''
    cmds.select(Cylinder[0]+'.f[17]')
    cmds.polyExtrudeFacet(s=(1,1,1), t=(0, 0, 0))
    cmds.scale (0.3,0.1,0.3, r=True, ls=True)
    cmds.polyCircularize(divisions=1)
    cmds.select(Cylinder[0]+'.f[17]')
    cmds.move(-0.2, 3, -0.5, r=True, os=True, wd=True)
    cmds.rotate(0, -35, 0, relative=True, componentSpace=True)

    for i in range (3):
        cmds.polyExtrudeFacet (s = (1,1,1), t=(0, 0, 0))
        cmds.rotate(0, -15, 0, r=True, cs=True)
        cmds.move (-random.uniform(0.5,0.6), random.uniform(0.8,1.2), random.uniform(-0.6,-0.9),
r=True, os=True, wd=True)
        cmds.scale (random.uniform(0.8,1),random.uniform(0.8,1),random.uniform(0.8,1), r=True,
cs=True)

    #Make Ridges
    ''' Makes Ridges '''
    cmds.polySelect(el = 94)
    cmds.polyExtrudeEdge()
    cmds.scale (1.1,1.1,1.1, r=True, ls=True)
    cmds.move (0, -2, 0, r=True, os=True, wd=True)
    cmds.polyNormal()

    cmds.polySelect(el = 119)
    cmds.polyExtrudeEdge()
    cmds.move (0, -2, 0, r=True, os=True, wd=True)
    cmds.scale (1.2,1,1.2, r=True, ls=True)
    cmds.polyNormal()

    cmds.polySelect(el = 122)
    cmds.polyExtrudeEdge()
    cmds.scale (1.2,1,1.2, r=True, ls=True)
    cmds.move (0, -1, 0, r=True, os=True, wd=True)
    cmds.polyNormal()

    cmds.polySelect(el = 144)
    cmds.polyExtrudeEdge()
    cmds.scale (1.4,1,1.4, r=True, ls=True)
    cmds.move (0, -1, 0, r=True, os=True, wd=True)
    cmds.polyNormal()

    cmds.polySelect(el = 166)
    cmds.polyExtrudeEdge()
    cmds.scale (1.4,1,1.4, r=True, ls=True)
    cmds.move (0, -4.5, 0, r=True, os=True, wd=True)
    cmds.polyNormal()

    #Move Edges and Verts
    ''' Moves Edges and Verts '''
```

```
    cmds.select(Cylinder[0]+'.vtx[40]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[42]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[44]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[46]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)

    cmds.select(Cylinder[0]+'.vtx[193]')
    cmds.move(random.random()/4,-
random.uniform(0.2,0.6),random.random()/4,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[199]')
    cmds.move(random.random()/4,-
random.uniform(0.2,0.6),random.random()/4,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[197]')
    cmds.move(random.random()/4,-
random.uniform(0.2,0.6),random.random()/4,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[195]')
    cmds.move(random.random()/4,-
random.uniform(0.2,0.6),random.random()/4,r=True,os=True,wd=True)

    cmds.select(Cylinder[0]+'.vtx[200]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[202]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[204]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[206]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)

    cmds.select(Cylinder[0]+'.vtx[210]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[212]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[214]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[208]')
    cmds.move(0,-random.uniform(0.2,0.6),0,r=True,os=True,wd=True)

    cmds.select(Cylinder[0]+'.vtx[220]')
    cmds.move(0,-random.uniform(0.2,0.4),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[222]')
    cmds.move(0,-random.uniform(0.2,0.4),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[216]')
    cmds.move(0,-random.uniform(0.2,0.4),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[218]')
    cmds.move(0,-random.uniform(0.2,0.4),0,r=True,os=True,wd=True)

    cmds.select(Cylinder[0]+'.vtx[228]')
    cmds.move(0,-random.uniform(0.2,0.4),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[230]')
```

```
    cmds.move(0,-random.uniform(0.2,0.4),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[224]')
    cmds.move(0,-random.uniform(0.2,0.4),0,r=True,os=True,wd=True)
    cmds.select(Cylinder[0]+'.vtx[226]')
    cmds.move(0,-random.uniform(0.2,0.4),0,r=True,os=True,wd=True)

    cmds.select(Cylinder)
    cmds.hyperShade(assign = barkShader)

    cmds.select(Cylinder[0]+'.f[42:89]')
    cmds.hyperShade(assign = leafShader)
    cmds.select(Cylinder[0]+'.f[170:209]')
    cmds.hyperShade(assign = leafShader)
    cmds.select(Cylinder[0]+'.f[25]')
    cmds.hyperShade(assign = leafShader)

    cmds.select(Cylinder)

    #Distributes Objects on terrain
def rockSpawn(winID, rock1amountControl, rock2amountControl, rockscaleControl, mushroomControl, *pArgs):

    RockOrg1=smallRock()
    cmds.rename('RockOrg1')
    RockOrg2=smallRock()
    cmds.rename('RockOrg2')
    RockOrg3=Rock()
    cmds.rename('RockOrg3')
    RockOrg4=Rock()
    cmds.rename('RockOrg4')

    MushroomOrg1=mushroom()
    cmds.rename('MushroomOrg1')
    MushroomOrg2=mushroom()
    cmds.rename('MushroomOrg2')
    MushroomOrg3=mushroom()
    cmds.rename('MushroomOrg3')
    MushroomOrg4=mushroom()
    cmds.rename('MushroomOrg4')

    terrainShape ="Landscape01"
    #rockNames = ["RockOrg1", "RockOrg2"]
    rockData = {"RockOrg1": rock1amountControl//2, "RockOrg2": rock1amountControl//2, "RockOrg3": rock2amountControl//2, "RockOrg4": rock2amountControl//2,"MushroomOrg1": mushroomControl//4, "MushroomOrg2": mushroomControl//4, "MushroomOrg3": mushroomControl//4, "MushroomOrg4": mushroomControl//4}

    numVertex = cmds.polyEvaluate(terrainShape, vertex=True)
    selectedVertices = random.sample(range(numVertex), numVertex)

    currentIndex = 0
    for pair in rockData.items():
```

```
        for i in range(pair[1]):
            if currentIndex>numVertex-1:
                break
            pos = cmds.pointPosition (terrainShape+".vtx["+str(selectedVertices[currentIndex])+"]",
world=True)

            newobj = cmds.instance(pair[0])

            cmds.move(pos[0],pos[1],pos[2],newobj)
            #cmds.scale (rockscaleControl/random.uniform(0.5,2.0),
rockscaleControl/random.uniform(0.5,2), rockscaleControl/random.uniform(0.5,2),newobj)
            cmds.scale (rockscaleControl/random.uniform(0.8,1.2),
rockscaleControl/random.uniform(0.5,2.0), rockscaleControl/random.uniform(0.8,1.2),newobj)
            cmds.rotate(0, random.randint(0,360),0,newobj)
            if pos[1] > 200:
                cmds.delete(newobj)

            currentIndex+=1

    cmds.delete("RockOrg1", "RockOrg2", "RockOrg3", "RockOrg4", "MushroomOrg1",
"MushroomOrg2", "MushroomOrg3", "MushroomOrg4")

    if cmds.objExists('Rock*'):
        cmds.select("Rock*")
        cmds.group(name="Rocks")
    if cmds.objExists('MushroomOrg*'):
        cmds.select("MushroomOrg*")
        cmds.group(name="Mushrooms")

def mushroom():

    StemShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( StemShader + '.color', 0.497, 0.426, 0.312, type = 'double3' )

    CapShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( CapShader + '.color', 0.358, 0.092, 0.049, type = 'double3' )

    #Random Numbers
    randomFloatScaleZXY = random.uniform(-0.1,0.1)
    randomTrunk = random.randint(6,10)

    #Make Stalk primitive
    Cylinder=cmds.polyCylinder (h=0.5,r=1,sx=42,sy=1,name='MushroomOriginal')
    cmds.hyperShade(assign = StemShader)

    #Adjust Stalk
    for i in range (randomTrunk):
        randomFloatScale = random.uniform(0.95,1.1)
        randomFloatZX = random.uniform(-0.3,0.3)
        randomFloatY = random.uniform(0.8,1.2)
        randomFloatTopY = random.uniform(0.8,3.2)
        randomFloatTopYsmall = random.uniform(0.1,0.3)
```

```
        cmds.select(Cylinder[0]+'.f[43]')
        cmds.polyExtrudeFacet(s=(randomFloatScale,randomFloatScale,randomFloatScale),
t=(randomFloatZX, randomFloatY, randomFloatZX), d=1)

    #Scale the top stalk extrusion
    cmds.scale(1.2,1.0,1.2, r=True)

    #Create Volva
    cmds.select(Cylinder[0]+'.f[254:295]')
    cmds.polyExtrudeFacet( d=1)
    cmds.scale (1.3,0.95,1.3, r=True, ws=True)


    #Create Gills
    morphUnder = random.uniform(0.01, 0.05)
    cmds.select(Cylinder[0]+'.f[43]')

cmds.polyExtrudeFacet(s=(1.4+randomFloatScaleZXY,1.4+randomFloatScaleZXY,1.4+randomF
loatScaleZXY+morphUnder), t=(0, randomFloatTopYsmall, 0), d=1)

cmds.polyExtrudeFacet(s=(1.5+randomFloatScaleZXY,1.5+randomFloatScaleZXY,1.5+randomF
loatScaleZXY+morphUnder), t=(0, randomFloatTopYsmall, 0), d=1)

cmds.polyExtrudeFacet(s=(1.6+randomFloatScaleZXY,1.6+randomFloatScaleZXY,1.6+randomF
loatScaleZXY+morphUnder), t=(0, randomFloatTopYsmall, 0), d=1)

cmds.polyExtrudeFacet(s=(1.3+randomFloatScaleZXY,1.3+randomFloatScaleZXY,1.3+randomF
loatScaleZXY+morphUnder), t=(0, randomFloatTopYsmall-0.4, 0), d=1)

cmds.polyExtrudeFacet(s=(1.2+randomFloatScaleZXY,1.2+randomFloatScaleZXY,1.2+randomF
loatScaleZXY+morphUnder), t=(0, randomFloatTopYsmall-0.5, 0), d=1)
    cmds.hyperShade(assign = CapShader)

    #Create Cap
    step = 0.9
    topStep=0.05

    for i in range (5):
      morph = random.uniform(-0.15,-0.05)
      cmds.polyExtrudeFacet(s=(step+morph,step,step), t=(0, randomFloatTopY-topStep, 0), d=3)
      step-=0.1
      topStep+=0.2
      cmds.hyperShade(assign = CapShader)

    #Scale top of Cap
    cmds.scale(0.5,1.0,0.5, r=True)
    cmds.select(Cylinder)

    #Create Base
    cmds.select(Cylinder[0]+'.f[42]')
    cmds.scale(4.5,6.0,4.5, r=True)
```

```
    cmds.polySubdivideFacet (dv=1)
    cmds.select(clear=True)
    cmds.select('MushroomOriginal')

    #Add Noise
    vtxCount = list(range(cmds.polyEvaluate(v=True)))
    random.shuffle(vtxCount)
    values = [random.triangular(0,0.3,0) for i in range(10)]
    values_count = len(values)
    optimize_setter = []

    for x in vtxCount:
        mod = x % values_count
        optimize_setter += [values[mod-1]*1,values[mod-1]*1,values[mod-1]*1]
    cmds.setAttr('MushroomOriginal.vtx[:]', *optimize_setter)
    cmds.polySmooth(c=1,dv=2,kb=True,ro=1)

    #Smooth
    #cmds.polySmooth(divisions=1)

    #Creates Grass
def grass():

    #Shader
    myShader = cmds.shadingNode( 'lambert', asShader = True)
    cmds.setAttr( myShader + '.color', 0.25, 0.65, 0.06, type = 'double3' )

    #Create Main Shape
    polycone=cmds.polyCone (h=0.5,r=0.5,sx=4,sy=4,name='grassBlade')
    cmds.select(polycone[0]+'.f[12]')
    cmds.scale (0.05,1,0.05)
    cmds.select(polycone[0]+'.f[4:7]')
    cmds.scale (0.05,1,0.05)
    cmds.select(polycone[0]+'.f[13:16]')
    cmds.scale (0.1,1,0.1)

    #Move Edges and Verts
    ''' Moves Edges and Verts '''
    cmds.select(polycone[0]+'.e[4:7]')
    cmds.move(random.random()/20,0,random.random()/20,r=True,os=True,wd=True)

    cmds.select(polycone[0]+'.e[8:11]')
    cmds.move(random.random()/20,0,random.random()/20,r=True,os=True,wd=True)

    cmds.select(polycone[0]+'.e[12:15]')
    cmds.move(random.random()/20,0,random.random()/20,r=True,os=True,wd=True)

    cmds.select(polycone[0]+'.vtx[16]')

cmds.move(random.random()/20,random.random()/5,random.random()/20,r=True,os=True,wd=Tr
ue)
```

```
    cmds.select(polycone)

    cmds.ls('grassBlade*')
    cmds.hyperShade(assign = myShader)

    cmds.select(polycone)
    cmds.polySmooth(c=1,dv=1,kb=True,ro=1)

    #Creates a Hill
def createHill(winID, hillHeight, *pArgs):

    Landscape01="Landscape01"
    percent = 0.02
    faces = []
    cmds.select(Landscape01)
    for obj in cmds.ls(sl=1, long=1):
        indexes = list(range(cmds.polyEvaluate(obj, face=1)))
        random.shuffle(indexes)
        indexes = indexes[:int(percent*len(indexes))]
        for i in range(len(indexes)):
            indexes[i] = obj+'.f['+str(indexes[i])+']'
        faces.extend(indexes)
    cmds.select(faces, r=1)
    cmds.polySelectConstraint( pp=5 )
    rn = random.random()
    tmpValueY = hillHeight*(rn)

    if(tmpValueY < 0):
        tmpValueY = 0.0

    cmds.move(0.0, tmpValueY, 0.0, r=True)

    #Creates elevation detail
def createDetail(winID, detailHeight, *pArgs):

    Landscape01="Landscape01"
    percent = 0.02
    faces = []
    cmds.select(Landscape01)
    for obj in cmds.ls(sl=1, long=1):
        indexes = list(range(cmds.polyEvaluate(obj, face=1)))
        random.shuffle(indexes)
        indexes = indexes[:int(percent*len(indexes))]
        for i in range(len(indexes)):
            indexes[i] = obj+'.f['+str(indexes[i])+']'
        faces.extend(indexes)
    cmds.select(faces, r=1)
    cmds.polySelectConstraint( pp=5 )
    rn = random.random()
    tmpValueY = detailHeight*(rn)

    if(tmpValueY < 0):
```

```
        tmpValueY = 0.0

    cmds.move(0.0, tmpValueY, 0.0, r=True)
    cmds.polySubdivideFacet()

    #Creates a ditch
def createDitch(winID, ditchDepth, *pArgs):

    Landscape01="Landscape01"
    percent = 0.02
    faces = []
    cmds.select(Landscape01)
    for obj in cmds.ls(sl=1, long=1):
        indexes = list(range(cmds.polyEvaluate(obj, face=1)))
        random.shuffle(indexes)
        indexes = indexes[:int(percent*len(indexes))]
        for i in range(len(indexes)):
            indexes[i] = obj+'.f['+str(indexes[i])+']'
        faces.extend(indexes)
    cmds.select(faces, r=1)
    cmds.polySelectConstraint( pp=5 )
    rn = random.random()
    tmpValueY = ditchDepth*(rn)

    if(tmpValueY < 0):
        tmpValueY = 0.0

    cmds.move(0.0, -tmpValueY, 0.0, r=True)

    #Creates Mountains
def createMountins(winID, mountainHeight, *pArgs):

    Landscape01="Landscape01"
    percent = 0.008
    faces = []
    cmds.select(Landscape01)
    for obj in cmds.ls(sl=1, long=1):
        indexes = list(range(cmds.polyEvaluate(obj, face=1)))
        random.shuffle(indexes)
        indexes = indexes[:int(percent*len(indexes))]
        for i in range(len(indexes)):
            indexes[i] = obj+'.f['+str(indexes[i])+']'
        faces.extend(indexes)
    cmds.select(faces, r=1)
    cmds.polySelectConstraint( pp=1 )
    rn = random.randint(mountainHeight,mountainHeight)
    tmpValueY = rn

    if(tmpValueY < 0):
        tmpValueY = 10

    cmds.move(0.0, tmpValueY, 0.0, r=True)
```

```
    cmds.polyExtrudeFacet(s=(1,1,1), t=(1, 1, 1))
    cmds.scale(0.62,0.64,0.66,cs=True,r=True)
    cmds.move(0.0, tmpValueY/1.5, 0.0, r=True)
    cmds.polyExtrudeFacet(s=(1,1,1), t=(1, 1, 1))
    cmds.scale(0.66,0.62,0.68,cs=True,r=True)
    cmds.move(0.0, tmpValueY/2, 0.0, r=True)
    cmds.polySelectConstraint( pp=2 )
    cmds.move(0.0, tmpValueY/10, 0.0, r=True)
    cmds.select(Landscape01)
    cmds.polySoftEdge(a=180, ch=1)

def smallRockStack(noiseControl):

    randomFloat = random.uniform(0.1,0.2)
    #Make rock
    Cube=cmds.polyCube(h=8,w=8,d=8,sx=3,sy=3,sz=3,name='RockOriginal')
    cubeFaces = (Cube[0]+'.f[25]',Cube[0]+'.f[37]',Cube[0]+'.f[1]',Cube[0]+'.f[46]')
    #Make adjust
    for i in range (1):
        cmds.select(cubeFaces)
        cmds.scale
(random.uniform(1.5,2.5),random.uniform(1,1.2),random.uniform(1.5,2.5),r=True)

    cubeFaces2 = (Cube[0]+'.f[45]',Cube[0]+'.f[0]',Cube[0]+'.f[38]',Cube[0]+'.f[36]')
    #Make adjust
    for i in range (1):
        cmds.select(cubeFaces2)
        cmds.scale
(random.uniform(0.5,1.5),random.uniform(1,1.2),random.uniform(0.5,1.5),r=True)

    cubeFaces3 = (Cube[0]+'.f[43]',Cube[0]+'.f[19]',Cube[0]+'.f[7]',Cube[0]+'.f[52]')
    #Make adjust
    for i in range (1):
        cmds.select(cubeFaces3)
        cmds.scale (random.uniform(1.5,2),random.uniform(0.5,2.2),random.uniform(1.5,2),r=True)

    cmds.select(Cube[0]+'.f[13]')
    cmds.move (1,random.uniform(2.5,5.2),1,r=True)
    cmds.select(Cube[0]+'.f[31]')
    cmds.move (1,random.uniform(-2.5,-5.2),1,r=True)

    cmds.select(cl=True)
    cmds.polySelect('RockOriginal',el=22)
    cmds.move (0,random.uniform(1.1,1.8),0,r=True)


cmds.select(Cube[0]+'.vtx[36]',Cube[0]+'.vtx[32]',Cube[0]+'.vtx[39]',Cube[0]+'.vtx[35]',Cube[0]+
'.vtx[3]',Cube[0]+'.vtx[7]',Cube[0]+'.vtx[0]',Cube[0]+'.vtx[4]')
    cmds.scale (random.uniform(1.4,1.6),random.uniform(1,2),random.uniform(1.4,1.6),r=True)

    cmds.select(Cube)
```

```
    cmds.scale
(random.uniform(0.8,1.0),random.uniform(0.8,1.0),random.uniform(0.8,1.0),a=True)
    cmds.scale (randomFloat,randomFloat,randomFloat,r=True)

    cmds.polySmooth(c=1,dv=2,kb=True,ro=1)
    addNoise(noiseControl)
    cmds.select('RockOriginal', r=True )

#cmds.rotate(random.uniform(10,120),random.uniform(10,120),random.uniform(10,1200),a=True
)
    cmds.rename('Rock_Small_01')

def rockFlatStack(winID, noiseControl, sizeControl, stackSizeControl):

    randomStackSize = random.uniform(stackSizeControl+1,stackSizeControl+2.5)
    randomFloatStack = random.uniform(0.5,1.0)
    randomFloatX = random.uniform(-sizeControl/2,sizeControl/2)
    randomFloatY = random.uniform(-sizeControl/2,sizeControl/2)

    moveY=11
    scaleUni=1.5
    scaleUniY=1.1
    step=1

    for i in range (10):
        RockOrg=smallRockStack(noiseControl)
        cmds.select()
        cmds.move(randomFloatX,moveY,randomFloatY)
        cmds.rotate(random.uniform(1,15),random.uniform(10,360),random.uniform(1,15),a=True)
        cmds.scale(scaleUni,scaleUniY,scaleUni)

        scaleUni-=0.1
        scaleUniY-=0.1
        moveY+=11-step
        step+=1

    cmds.select('Rock_Small*')
    cmds.group(name="Rock_Flat_Stack_01",r=True)
    cmds.select(clear=True)

    cmds.select('Rock_Flat_Stack_*')
    cmds.move(0, 100, 0, absolute=True)
    cmds.move(0,    0,    0,    "Rock_Flat_Stack_*.scalePivot","Rock_Flat_Stack_*.rotatePivot",
absolute=True)
    cmds.scale(randomFloatStack,randomFloatStack,randomFloatStack)

    cmds.select(clear=True)
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

Інформаційна технологія генерування стилізованих 3D-ландшафтів

Виконав: студент 2 курсу, групи 1КН-23м
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Кушнір О.А.
(прізвище та ініціали)
Керівник: д. т. н., проф. каф. КН
Іванчук Я.В.
(прізвище та ініціали)
«05» 12 2024 р.

Вінниця ВНТУ - 2024 рік

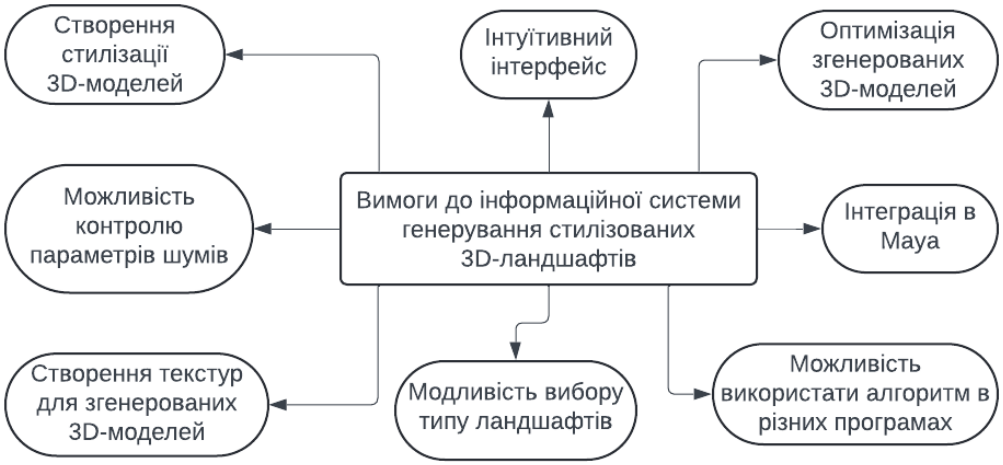

Рисунок В.1 - Вимоги до інформаційної системи генерування стилізованих 3D-ландшафтів

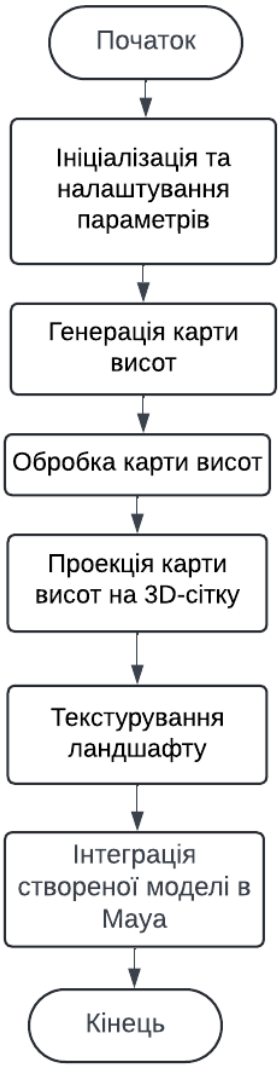

Рисунок В.2 - Загальна структура генератора ландшафтів

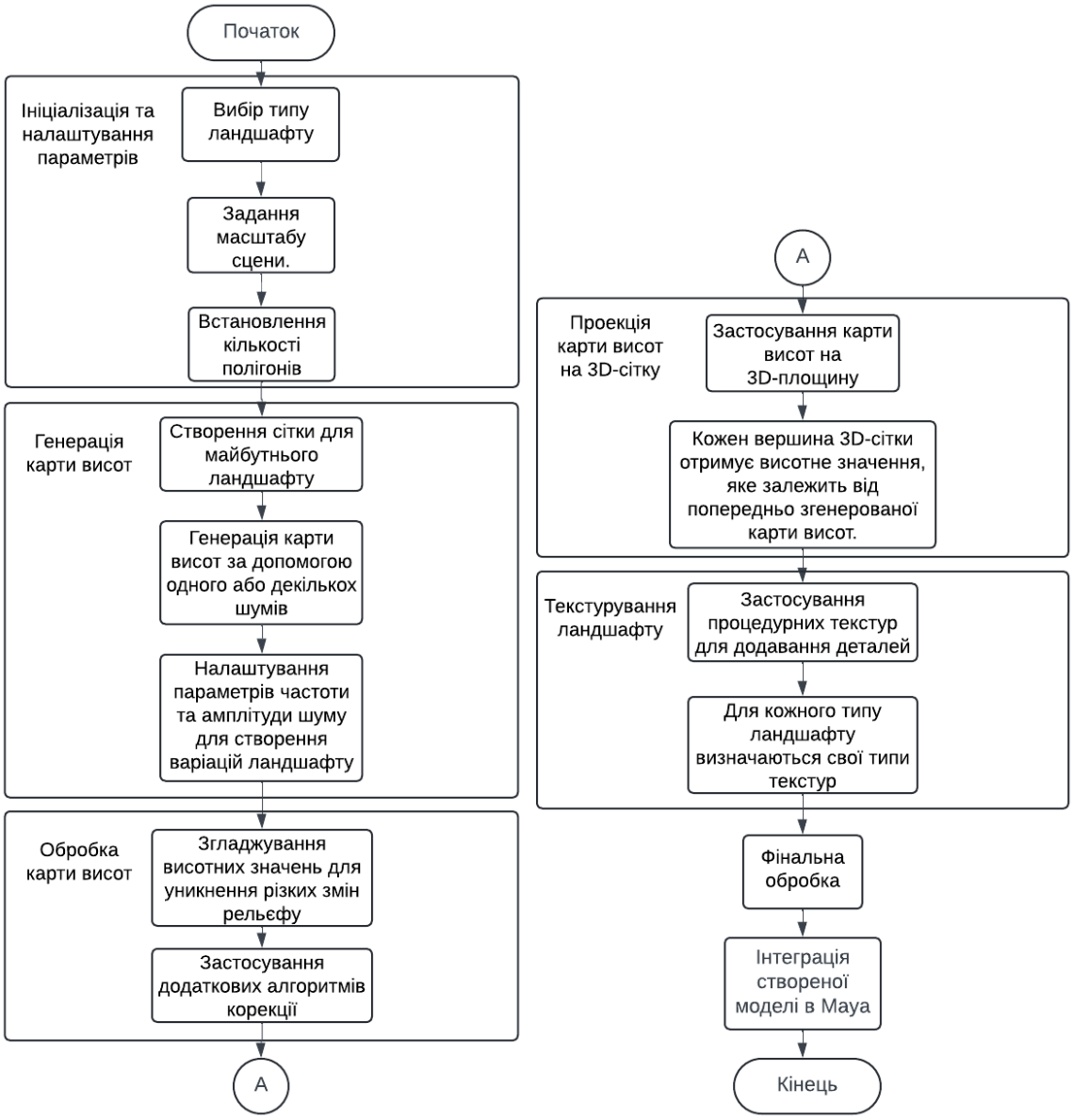

Рисунок В.3 – Схема алгоритму інформаційної технології генерування стилізованих 3D-ландшафтів

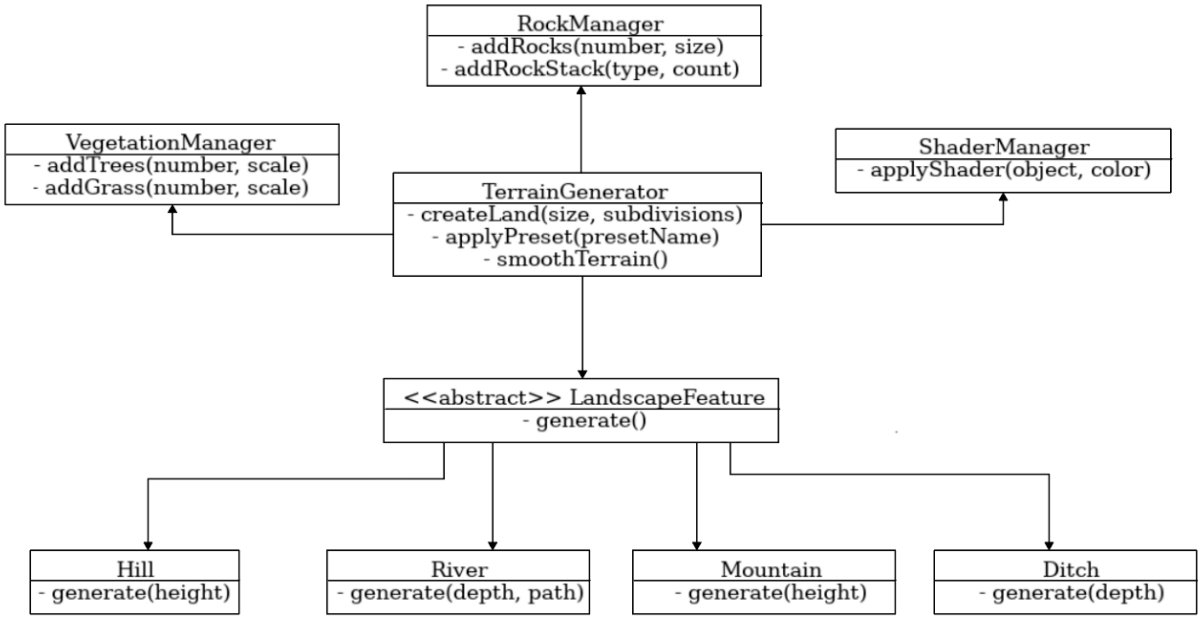

Рисунок В.4 – UML-діаграма класів інформаційної технології генерування стилізованих 3D-ландшафтів

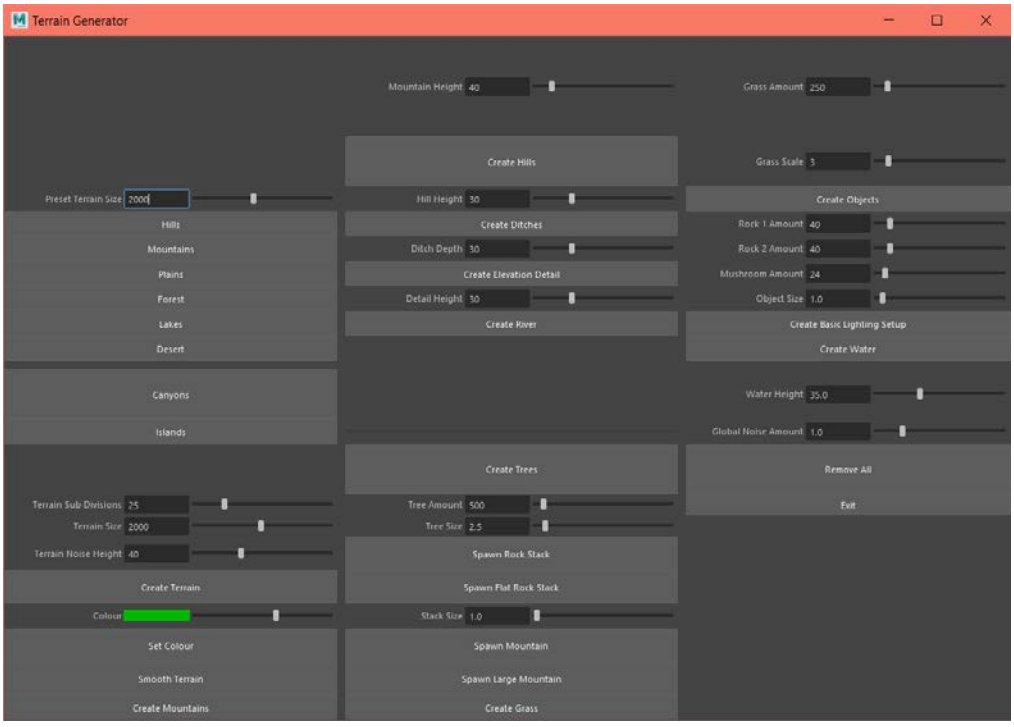

Рисунок В.5 – Загальний вигляд інтерфейсу плагіну генерування стилізованих 3D-ландшафтів

–

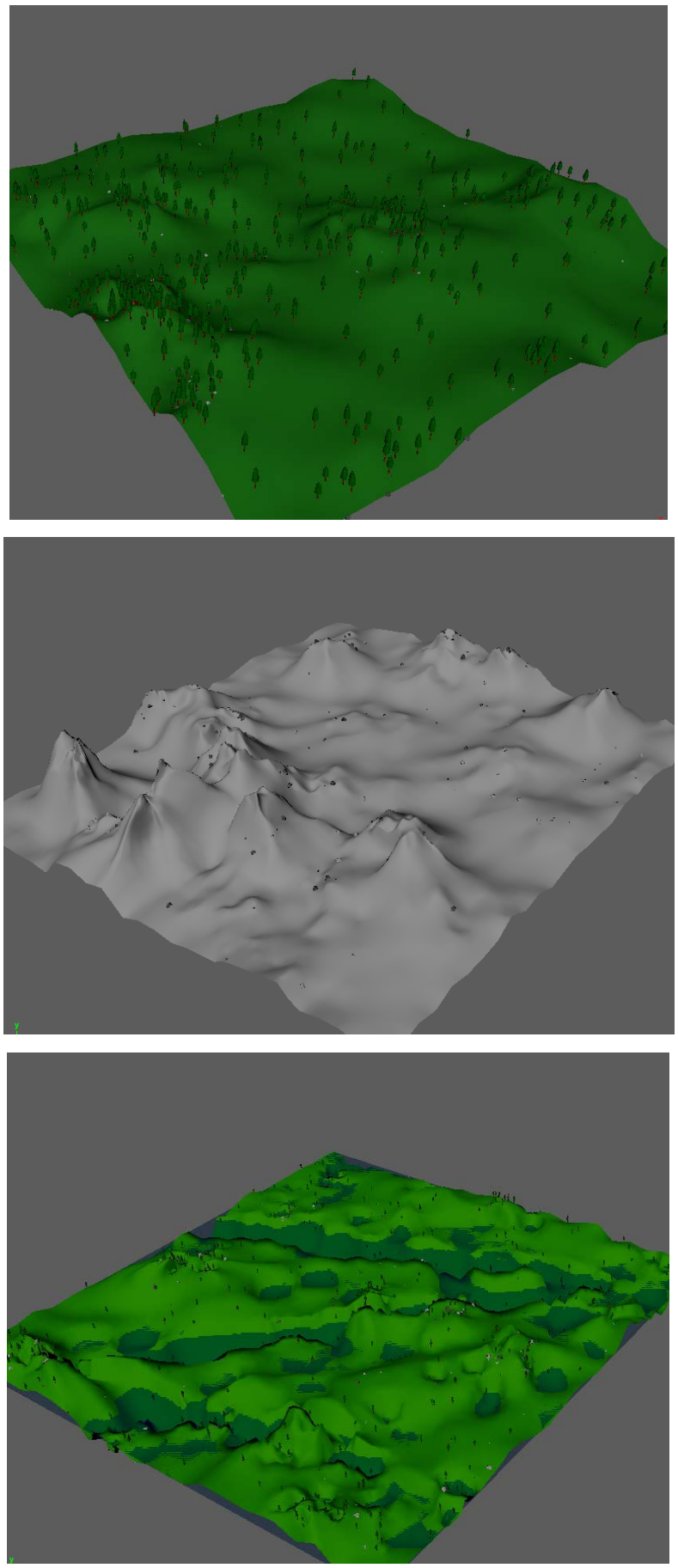

Рисунок В.6 – Генерування готових ландшафтів за допомогою плагіну

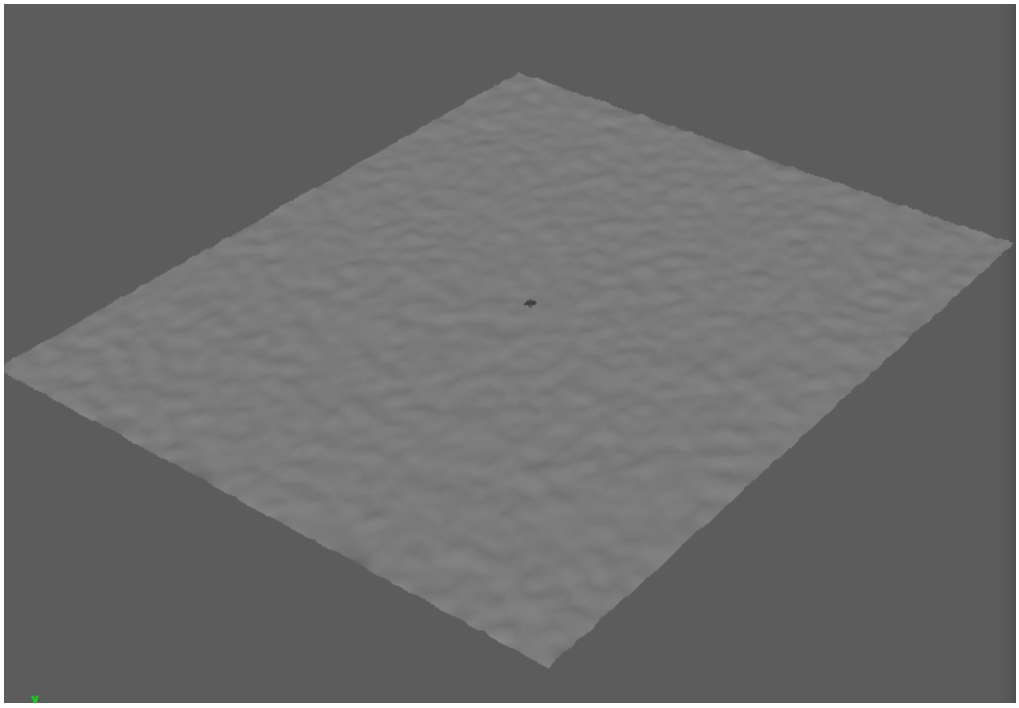

Рисунок В.7 – Згенерована за допомогою створеного плагіну площина

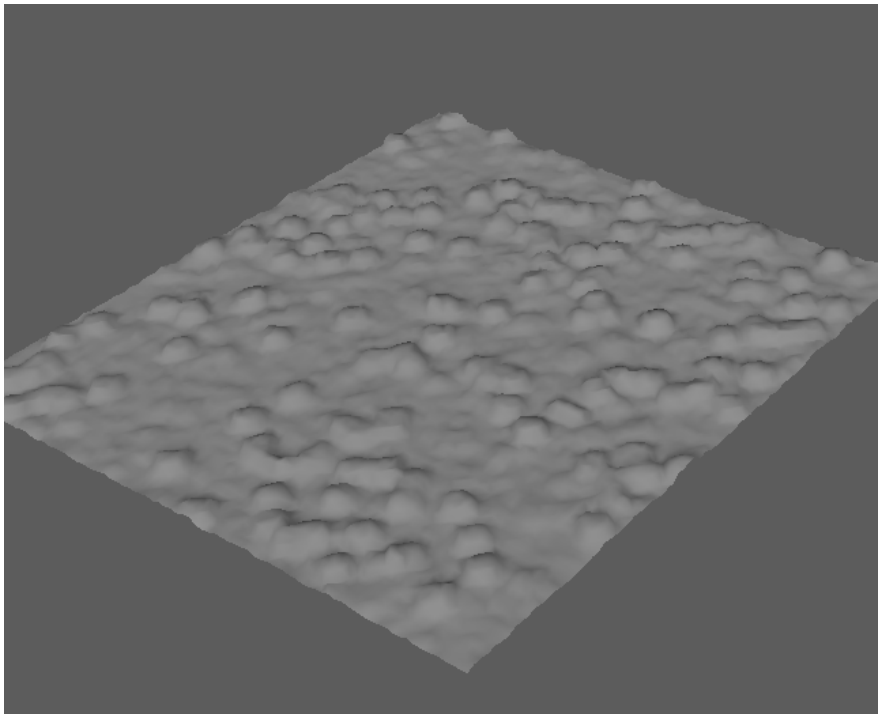

Рисунок В.8 – Згенеровані за допомогою створеного плагіну пагорби на площині

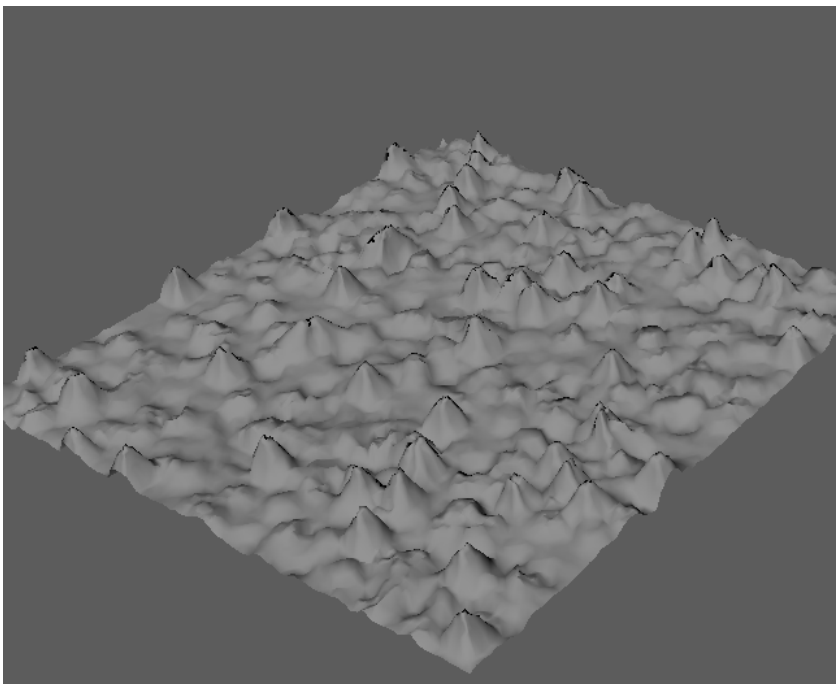

Рисунок В.9 – Згенеровані за допомогою створеного плагіну гори на площині

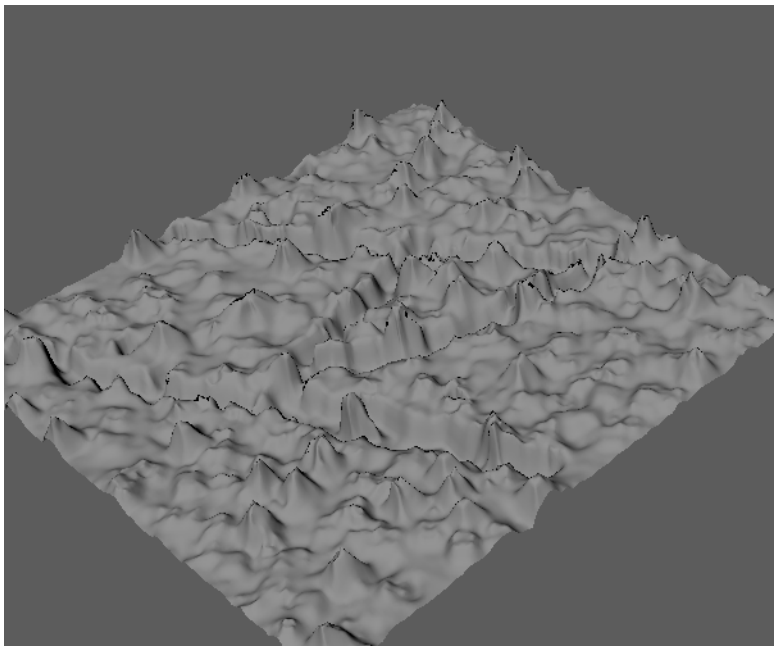

Рисунок В.10 – Згенеровані за допомогою створеного плагіну русла річок

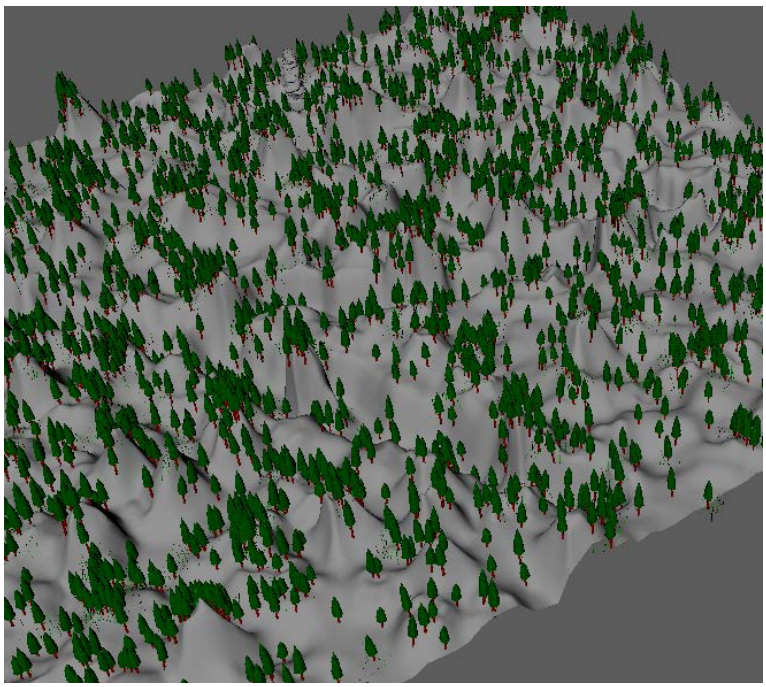

Рисунок В.11 – Згенерована за допомогою створеного плагіну рослинність

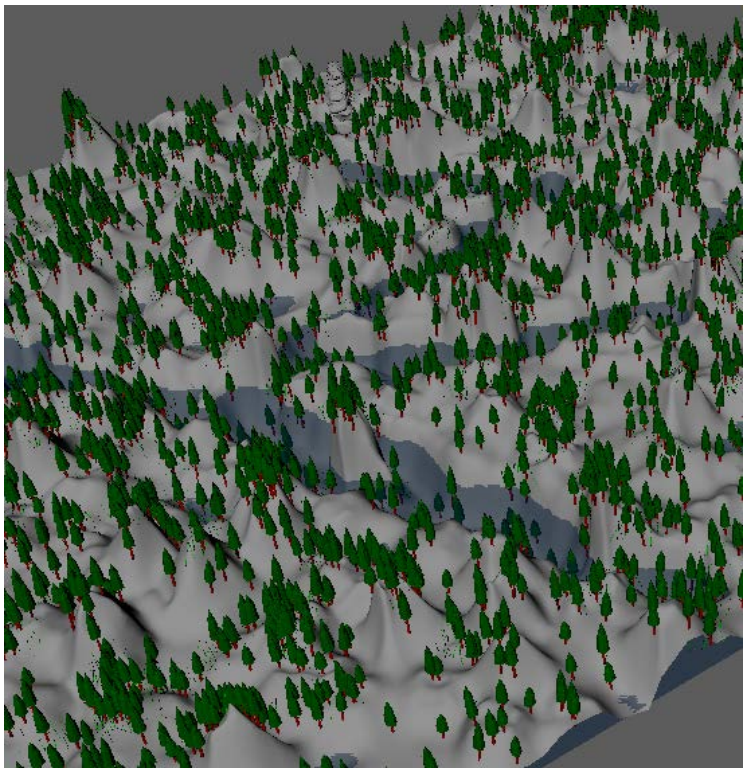

Рисунок В.12 – Згенерована за допомогою створеного плагіну вода

Додаток Г (довідниковий)

Інструкція користувача

Щоб додати скрипт до середовища розробки Autodesk Maya, потрібно зайти в меню Windows, яке знаходиться у верхній панелі, обрати General Editors, і в ньому відкрити Script Editor

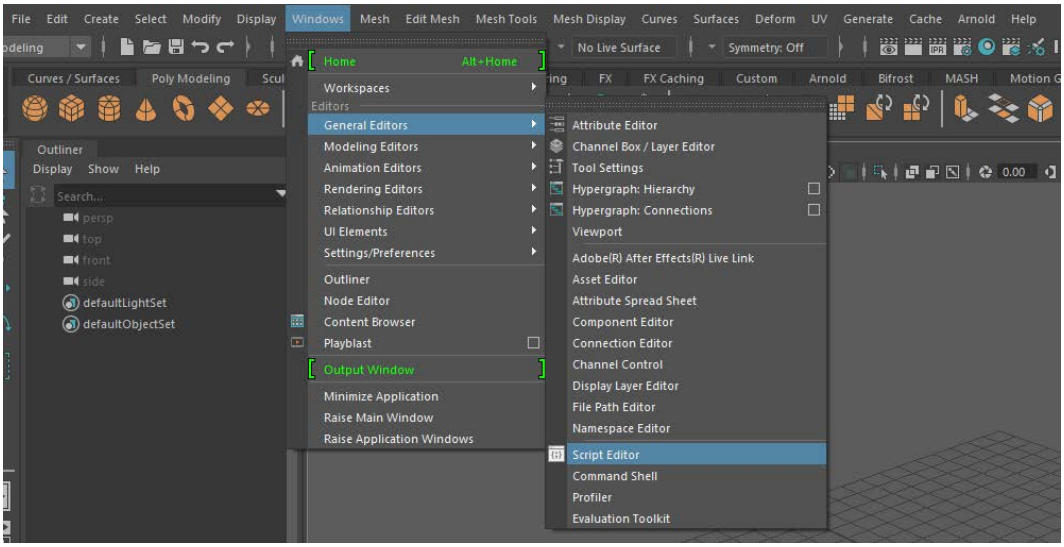

Рисунок Г.1 – Вікно Autodesk Maya, шляк до вікна Script Editor

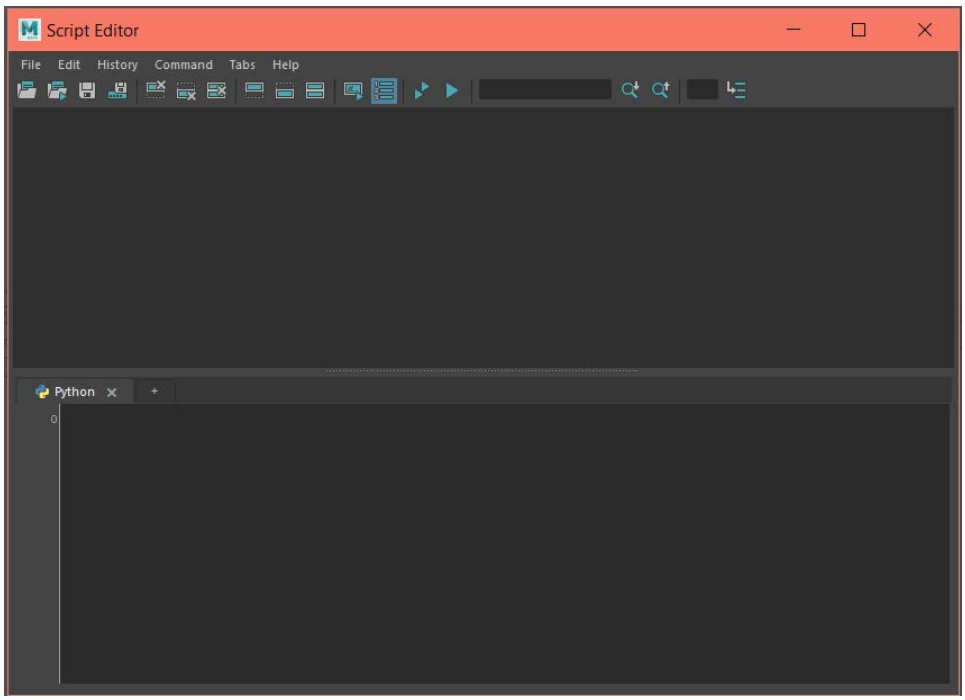

Рисунок Г.2 – Вікно Script Editor

Пля відкриття Script Editor потрібно підгрузити файл зі скриптом. Для цього треба нажати верхню ліву кнопку, і обрати потрібний файл. Код буде відображатись у нижній частині Script Editor. Після цього ми можемо запустити плагін.

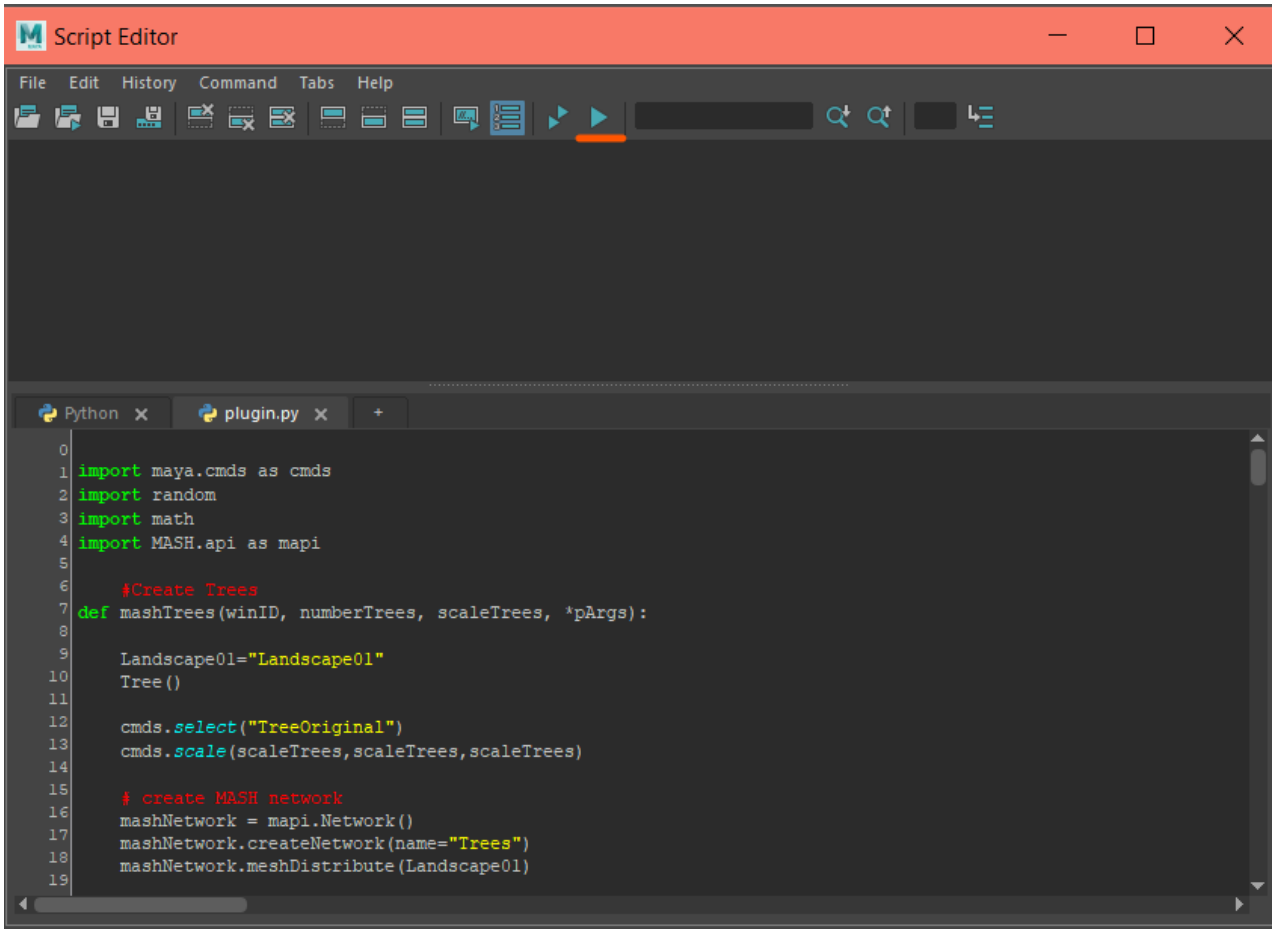

Рисунок Г.3 – Вікно Script Editor з імпортованим скриптом

Після запуску відкривається вікно для створення стилізованих 3D-ландшафтів. Воно розділяється на 4 частини:

– створення готових присетів ландшафтів;
– створення площини і додавання індивідуальних елементів ландшафту;
– додавання деталей (дерева, трава, камння, вода);
– видалення створених ландшафтів і вихід з плагіну.

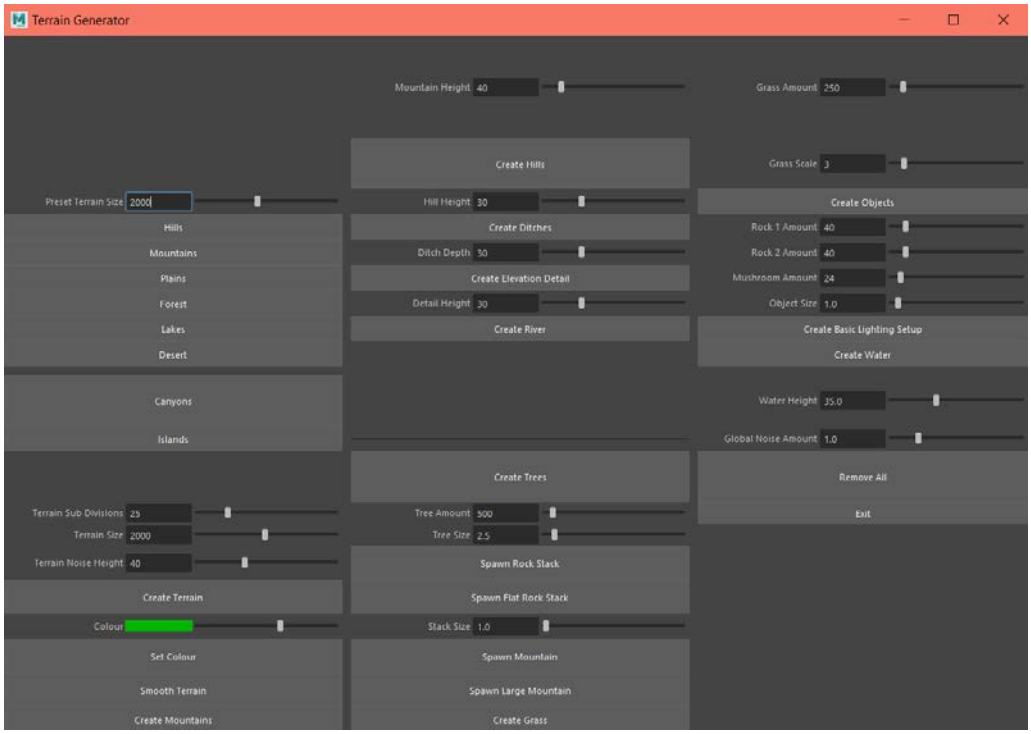

Рисунок Г.4 – Вікно для створення стилізованих 3D-ландшафтів

Щоб створити готовий ландшафт потрібно вибрати потрібний розмір і тип ландшафту.

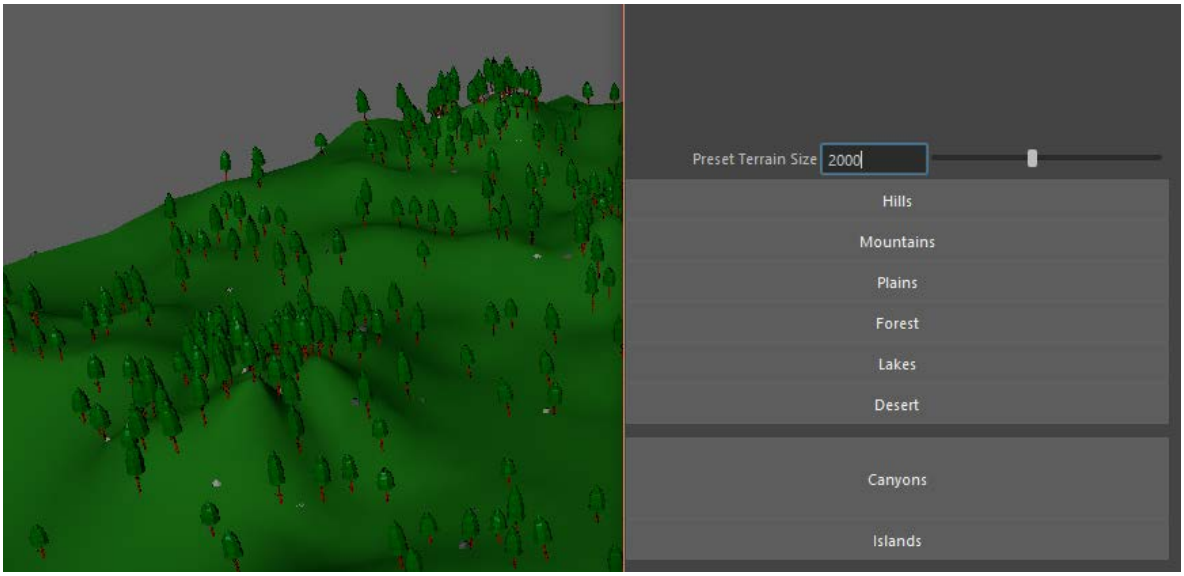

Рисунок Г.5 – Створення готових ландшафтів

Щоб створити свій налаштовуваний ландшафт, потрібно спершу обрати потрібні параметри та створити площину

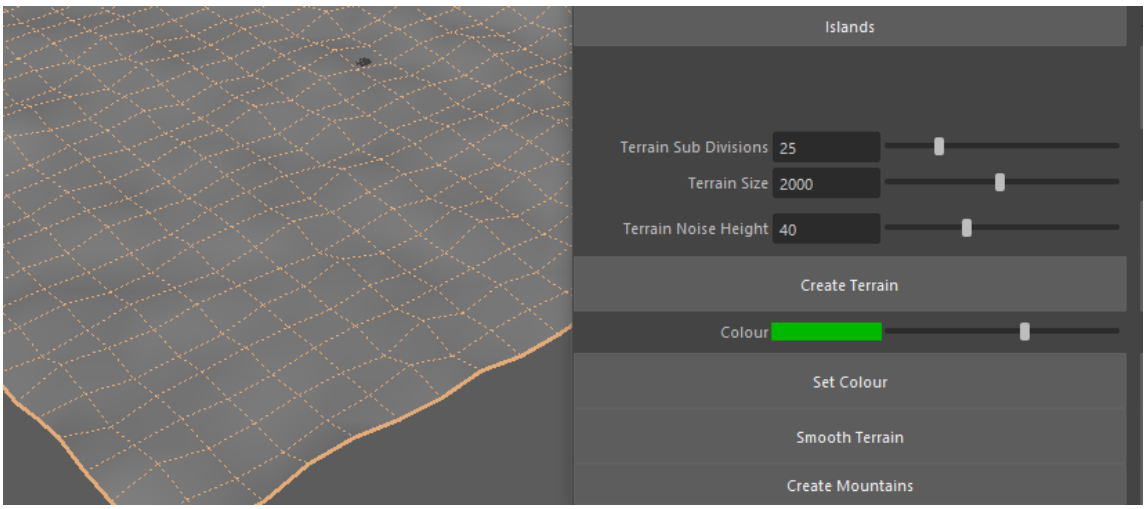

Рисунок Г.6 – Створення площини для подальшого додавання елементів і деталей ландшафту

Після цього можна добавляти по черзі на площину різні елементи ландшафту та йогодеталі. Для цього спершу потрібно виставити потрібні параметри і натиснути на відповідну кнопку.

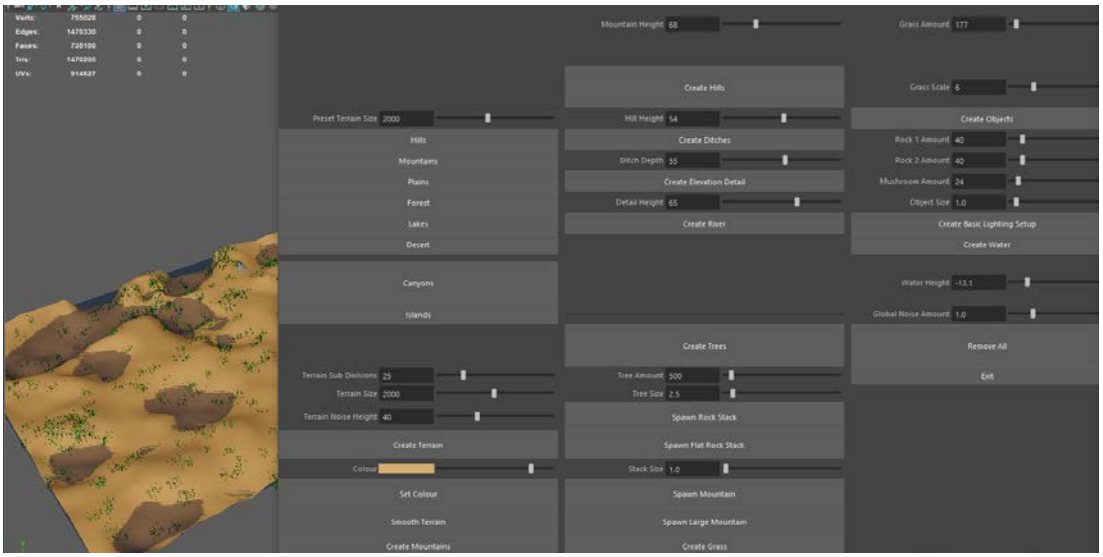

Рисунок Г.7 – Створення власних ландшафтів з площини

Протестувавши програмне забезпечення, робимо висновок про коректність його роботи у відповідності до поставлених вимог.