Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ЛАНДШАФТІВ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-2
Стрюк Д. С.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник асист. Кобилін І. О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Стрюку Данилу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку для процедурної генерації ландшафтів

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 7 червня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, мова програмування Swift, рушій SceneKit, середовище розробки Xcode, фреймворк UIKit, алгоритми шуму Перліна та Diamond-Square, API SceneKit Geometry, пристрої з iOS.

4. Перелік питань, що потрібно опрацювати в роботі
1. Дослідження алгоритмів процедурної генерації ландшафтів.
2. Аналіз технологій та інструментів для розробки під iOS.
3. Реалізація генерації карти висот і полігональної сітки.
4. Візуалізація сцени та інтеграція з інтерфейсом користувача.
5. Оптимізація, тестування та підготовка застосунку до використання.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми обробки зображень, постановка задачі, тестові зображення, схема генерації ландшафту, приклади згенерованого рельєфу, карти висот, схема обробки нормалей, нормалі до сітки, ефект згладжування, схема оновлення сцени, архітектура застосунка, інтерфейс користувача, таблиці з характеристиками алгоритмів та рушіїв, графіки продуктивності.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-12.04.25	
4	Аналіз технічних засобів	13.04.25-20.04.25	
5	Проєктування алгоритму генерації ландшафту	21.04.25-22.04.25	
6	Програмна реалізація	26.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
13	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____ асист. Кобилін І. О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 110 с., 14 табл., 71 рис., 1 дод., 30 джерел.

ПРОЦЕДУРНА ГЕНЕРАЦІЯ, ТРИВИМІРНИЙ ЛАНДШАФТ, SCENEKIT, IOS, SWIFT, КАРТА ВИСОТ, ВІЗУАЛІЗАЦІЯ, ШУМ ПЕРЛІНА, DIAMOND-SQUARE.

Об’єктом роботи є алгоритми генерації тривимірного ландшафту на мобільних пристроях.

Метою роботи є створення застосунку для iOS, що реалізує генерацію тривимірного ландшафту на основі карти висот із візуалізацією у середовищі SceneKit.

У межах роботи розглянуто алгоритми процедурної генерації рельєфу, зокрема шум Перліна та Diamond-Square. Обґрунтовано вибір SceneKit як основного інструменту. Реалізовано генерацію карти висот, побудову полігональної сітки, розфарбування зон (вода, сніг, трава) та автоматичне розміщення дерев. Застосунок дозволяє повторно створювати ландшафт з новими параметрами у реальному часі.

У результаті роботи створено мобільний застосунок з процедурною генерацією тривимірного ландшафту та інтерактивною візуалізацією сцени.

PROCEDURAL GENERATION, 3D LANDSCAPE, SCENEKIT, IOS, SWIFT, HEIGHTMAP, VISUALIZATION, PERLIN NOISE, DIAMOND-SQUARE.

The object of the work is the set of algorithms for building 3D terrain models on mobile platforms.

The aim of the work is to develop an iOS application that generates 3D landscapes using a heightmap and renders them in SceneKit.

Within the scope of the work, procedural terrain generation techniques were analyzed, including Perlin noise and Diamond-Square. SceneKit was selected as the main framework. Implemented heightmap creation, polygon mesh construction, zone coloring (water, snow, grass), and automated tree placement. The app allows real-time terrain regeneration with custom parameters.

As a result of the work, a mobile application was created with procedural 3D landscape generation and interactive scene visualization.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

iOS – операційна система мобільних пристроїв Apple

macOS – пропрієтарна операційна система для комп’ютерів Apple

watchOS – операційна система годинників Apple

tvOS – операційна система пристроїв Apple TV

API – Application Programming Interface (набір інструментів та правил)

CGI – Computer Generated Imagery (тривимірна комп’ютерна графіка)

AR – технологія доповненої реальності світу

ARC – механізм керування пам’яттю

ARKit – фреймворк Apple

GCD – фреймворк Apple

ARM – Advanced RISC Machine (сімейство 32- та 64-бітних мікропроцесорних/мікроконтролерних ядер)

LLVM – Low Level Virtual Machine (універсальна система аналізу, трансформації і оптимізації програм)

Blueprints – план або креслення

UE – Unreal Engine

C – універсальна, процедурна та імперативна мова програмування

C++ – об’єктно-орієнтована мова програмування загального призначення

C# – об’єктно-орієнтована мова програмування

LOD – Level of Detail (рівнів деталізації)

.dae – формат файлу для зберігання та обміну тривимірними об’єктами та сценами

.obj – формат файлу для опису тривимірної геометрії

IDE – Integrated Development Environment (інтегроване середовище розробки)

UI – User Interface (користувацький інтерфейс)

UIKit – фреймворк, який використовується для створення графічного інтерфейсу в iOS-застосунках

ВСТУП

Розвиток тривимірної графіки суттєво вплинув на зміну парадигм у створенні цифрових застосунків. Технології, які раніше використовувались виключно у високопродуктивних системах або спеціалізованих графічних редакторах, нині стають доступними й оптимізованими для мобільних пристроїв. Тривимірна графіка проникає в усі сфери – від комп'ютерних ігор та архітектурного проєктування до навчальних застосунків, віртуальної реальності та наукових візуалізацій.

Одним із ключових напрямів у тривимірній графіці є побудова складних сцен, що включають реалістичні природні середовища. Це можуть бути ландшафти, гори, водойми, ліси, які складаються з тисяч або мільйонів об'єктів і поверхонь. Ручне створення таких сцен – трудомісткий та ресурсозатратний процес. У зв'язку з цим дедалі більшого поширення набувають методи автоматизованого побудування геометрії та об'єктів – процедурна генерація.

Процедурна генерація дозволяє формувати геометричний або візуальний контент за заданими математичними правилами. Замість того, щоб вручну моделювати кожен об'єкт, розробник задає параметри – і система автоматично створює унікальні варіанти сцени. Такий підхід економить ресурси, забезпечує гнучкість, дає змогу масштабувати середовище й створювати безліч варіацій у реальному часі.

Особливої актуальності процедурна генерація набуває в контексті мобільної розробки. Обмежені обчислювальні ресурси, вимоги до компактності застосунку, швидкий рендеринг і підтримка інтерактивності ставлять перед розробником завдання створення ефективної, але водночас візуально якісної сцени. Одним із рішень є використання фреймворку SceneKit, який надає розробникам інструменти для роботи з тривимірною графікою на платформах iOS і macOS. SceneKit підтримує побудову

геометрії, налаштування камер, освітлення, матеріалів, фізичних властивостей об’єктів та анімацій.

Актуальність роботи полягає у зростаючій потребі у динамічному, адаптивному тривимірному контенті для мобільних платформ, що поєднує високу продуктивність з гнучкістю генерації середовища в реальному часі без втрати якості візуалізації.

Реалізовано програмний застосунок, який генерує тривимірний ландшафт на основі карти висот, побудованої алгоритмічно. Для формування рельєфу використано такі алгоритми як шум Перліна та Diamond-Square. Сцена автоматично розфарбовується згідно з висотними рівнями (наприклад: трава, сніг, пісок, вода), а в зонах з відповідними параметрами висоти розміщуються спрощені тривимірні моделі дерев. Застосунок створено на мові програмування Swift, реалізовано в середовищі Xcode з використанням SceneKit.

Окрему увагу приділено інтерактивності – користувач має змогу ініціювати нову генерацію сцени одним дотиком, що дозволяє змінювати ландшафт без перезапуску програми. Це робить рішення придатним як для навчальних симуляцій, так і для ігрових або демонстраційних застосунків.

Серед основних завдань – аналіз математичних алгоритмів генерації рельєфу, побудова карти висот, формування полігональної сітки, візуалізація сцени з відповідними матеріалами й освітленням, а також забезпечення інтерактивної взаємодії користувача із застосунком.

Підсумовуючи, розроблене рішення демонструє ефективність застосування процедурної генерації у мобільному середовищі, забезпечує гнучкість і масштабованість тривимірного контенту, а також відкриває можливості для подальшого розширення функціоналу – від інтеграції з ігровими механіками до додавання елементів штучного інтелекту або доповненої реальності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ ОСНОВИ

1.1 Загальні положення

1.1.1 Актуальність теми

З розвитком цифрових технологій тривимірна графіка почала активно проникати в різні сфери людської діяльності. Вона активно використовується у відеоіграх, віртуальній та доповненій реальності, архітектурній візуалізації та навіть у наукових дослідженнях [1]. З кожним роком зростає попит на складні, реалістичні та водночас динамічні тривимірні середовища. Однак створення таких візуалізацій вручну – це надзвичайно складний і витратний процес, який вимагає багато часу та ресурсів. На допомогу приходить процедурна генерація – метод, що дозволяє створювати складні об’єкти та середовища автоматично за допомогою математичних алгоритмів [2].

Процедурна генерація – це справжня знахідка для розробників. Замість того щоб ретельно моделювати кожен елемент вручну, можна налаштувати алгоритм, який автоматично створить потрібний результат. Наприклад, у процедурній генерації ландшафтів такі алгоритми здатні формувати гори, рівнини, річки або навіть цілі ігрові світи [3]. Це не лише економить час, але й відкриває можливості для створення динамічних середовищ, які можуть змінюватися залежно від дій користувача чи інших параметрів системи.

Цей підхід став надзвичайно популярним у розробці відеоігор. Процедурна генерація дозволяє створювати масштабні світи, унікальні для кожного користувача, що значно покращує ігровий досвід. У архітектурній візуалізації вона допомагає швидко відтворювати природні ландшафти або урбаністичні середовища. У науці цей метод застосовують для симуляції природних процесів або моделювання складних явищ.

Серед інструментів для роботи з тривимірною графікою особливої уваги заслуговує SceneKit – платформа, створена Apple. SceneKit спрощує роботу з тривимірною графікою в екосистемі iOS та macOS, надаючи

розробникам інтуїтивно зрозумілі інструменти для створення сцен. Завдяки інтеграції з мовою Swift, SceneKit дозволяє швидко налаштовувати освітлення, фізику об’єктів, анімацію та багато іншого. Це робить платформу привабливою для мобільних розробників, які створюють високоякісний тривимірний контент.

Проте, незважаючи на всі переваги, документація SceneKit щодо процедурної генерації залишається обмеженою. Це може стати викликом для розробників, які хочуть використовувати платформу для складних завдань.

Розробка застосунку для процедурної генерації ландшафтів із використанням SceneKit є не лише технічно важливим завданням, але й корисним інструментом для вивчення можливостей платформи. Такий проєкт дозволить розробникам поглибити знання у сфері тривимірної графіки та створити базу для майбутніх проєктів. Це можуть бути ігрові світи, симулятори природних середовищ або навіть точні моделі реальних місцевостей.

Окрім цього, подібна розробка може сприяти вдосконаленню SceneKit. Вона надихне інших розробників на експерименти з процедурною генерацією, що дозволить розширити межі використання цієї платформи. Такий підхід не тільки відкриє нові можливості, але й допоможе зрозуміти, як автоматизовані процеси можуть оптимізувати творчу роботу, яка раніше була можлива лише вручну.

1.1.2 Концепція, тематика та напрям розробки

Концепція проєкту полягає у створенні мобільного застосунку, здатного автоматично генерувати тривимірні ландшафти з урахуванням природних висотних зон. Основна ідея полягає в тому, щоб поєднати переваги процедурної генерації з інструментами тривимірної графіки для досягнення реалістичного візуального результату в реальному часі [4].

Особливу увагу приділено алгоритмічному формуванню рельєфу, автоматичному призначенню матеріалів залежно від висоти, динамічному створенню сцени та забезпеченню базової взаємодії з користувачем. У процесі реалізації враховано обмеження мобільного середовища, такі як обчислювальні ресурси та вимоги до продуктивності.

1.1.3 Тематика та напрям розробки

У роботі розглядаються процедурні методи створення тривимірних ландшафтів, які застосовуються для візуалізації природних середовищ. Основна увага приділяється реалізації алгоритмів генерації рельєфу в середовищі SceneKit – включаючи побудову полігональної сітки, накладення матеріалів, забезпечення оптимального рендерингу та можливості інтерактивної зміни параметрів сцени на мобільному пристрої [5].

1.1.4 Підходи та засоби реалізації

У процесі виконання кваліфікаційної роботи використовувались такі підходи та практики:

– аналітичний метод – для вивчення існуючих алгоритмів процедурної генерації та їх адаптації до сучасних технологій [6];

– експериментальний метод – для реалізації прототипу генерації ландшафту та його тестування у SceneKit [7];

– моделювання – для відображення рельєфів за допомогою полігональної сітки на основі згенерованої карти висот;

– візуалізація – для наочного представлення результатів генерації з урахуванням освітлення, текстурування та камерного налаштування [8];

– порівняльний аналіз – для оцінювання ефективності реалізованого рішення порівняно з іншими інструментами [9].

1.2 Основи процедурної генерації

1.2.1 Підходи та засоби реалізації

Процедурна генерація – це підхід до створення даних (зображень, геометрії, текстур) за допомогою алгоритмів і математичних функцій, а не через ручну обробку. У випадку ландшафтів це означає, що кожен елемент рельєфу – пагорби, долини, скелі – створюється автоматично, часто з урахуванням випадкових або псевдовипадкових факторів, що додає природності результату, основні переваги показано у таблиці 1.1 [10].

Таблиця 1.1 – Основні переваги процедурної генерації ландшафтів

Переваги	**Опис**
1	**2**
Масштабованість	Створення великих відкритих світів із невеликими витратами пам’яті.
Повторюваність	Однакові вхідні дані завжди створюють той самий результат, що корисно для тестування.
Варіативність	Зміна параметрів призводить до унікальних результатів, забезпечуючи різноманітність ландшафтів.
Автоматизація	Значно скорочується участь дизайнера, що особливо важливо у великих проєктах.

Процедурна генерація може бути реалізована як на етапі розробки , так і в реальному часі. У контексті мобільних застосунків важливим аспектом є оптимальність та ефективність обчислень, що напряму впливає на вибір алгоритмів та структур даних. [11]

1.2.2 Основні алгоритми генерації

Процедурна генерація базується на низці математичних алгоритмів, кожен з яких має свої переваги, недоліки та характер застосування. Найбільш поширеними є алгоритм Diamond-Square, шум Перліна, Simplex Noise, воронієві діаграми та інші. У підпункті розглянуто ключові з них [6–8].

Шум Перліна – це один із найвідоміших і найпоширеніших методів генерації псевдовипадкових текстур і рельєфів у комп'ютерній графіці. Цей алгоритм був розроблений американським науковцем Кеном Перліном у 1983 році в контексті задач тривимірної комп'ютерної анімації [12]. Його метою було створення візуально реалістичних природних форм, таких як хмари, вогонь, вода або ландшафт, без використання громіздких та ресурсоємних моделей.

На відміну від простого випадкового шуму (такого як шум Білого), шум Перліна генерує плавно змінювані значення, що створюють органічні, реалістичні візерунки, подібні до тих, що зустрічаються у природі. Завдяки цьому він став незамінним інструментом у сферах комп'ютерної графіки, ігор, CGI-анімації, а також у генерації процедурних середовищ.

Формула для генерації висот із застосуванням шуму Перліна має вигляд:

$$f(x,y) = \sum_{i=0}^{n} a_i P(2^i x, 2^i y), \tag{1.1}$$

де $P(x,y)$ – базова функція шуму;

a_i – коефіцієнти амплітуди;

n – кількість рівнів деталізації.

Основні характеристики алгоритму шуму Перліна, що відображають його ключові властивості, переваги, недоліки та напрями застосування у комп'ютерній графіці, показано у таблиці 1.2.

Таблиця 1.2 – Основні особливості шуму Перліна

Характеристика	**Опис**
1	**2**
Тип шуму	Градієнтний фрактальний шум
Періодичність	Відсутність чіткої періодичності, що забезпечує природний вигляд текстур
Гладкість	Плавні переходи між значеннями, без різких змін
Фрактальність	Підтримка октав для створення багаторівневих текстур
Виміри	Одновимірні, двовимірні, тривимірні і вище
Переваги	Реалістичність, гладкість, можливість масштабування
Недоліки	Вища обчислювальна складність у порівнянні з іншими типами шуму
Застосування	Карти висот, біоми, природні явища (туман, хмари, вода)

Структура з плавними переходами від низьких (темні ділянки) до високих (світлі ділянки) частин рельєфу нагадує природний ландшафт, що показано на рисунку 1.1.

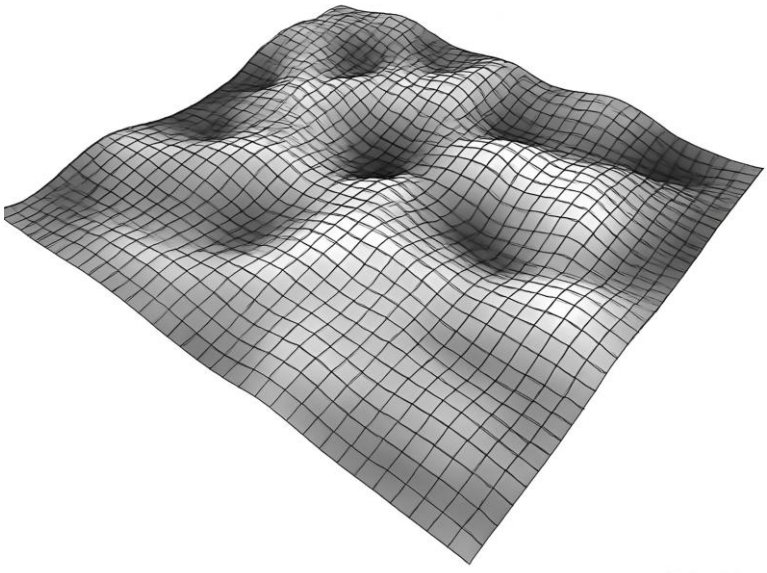

Рисунок 1.1 – Приклад карти висот, згенерованої шумом Перліна

Diamond-Square – це алгоритм, спеціально розроблений для генерації двовимірних карт висот, які широко використовуються у створенні ландшафтів, візуалізації поверхонь планет, віртуальному моделюванні місцевості та комп'ютерних іграх [13]. Його ключова ідея полягає в поступовому уточненні сітки, яка складається з квадратних блоків, за допомогою чергування двох основних кроків – ромб та квадрат. Такий підхід дозволяє генерувати складні фрактальні структури з великою кількістю деталей і варіативністю форм.

Формула для рекурсивного розрахунку висотних карт із застосуванням Diamond-Square має вигляд:

$$h_{new} = \frac{h_1 + h_2 + h_3 + h_4}{4} + R, \tag{1.2}$$

де h_1, h_2, h_3, h_4 – висоти сусідніх вершин;

R – випадковий зміщуючий фактор.

Основні характеристики алгоритму Diamond-Square, які розкривають його принципи роботи, ефективність та типові випадки використання, показано у таблиці 1.3.

Таблиця 1.3 – Характеристики алгоритму Diamond-Square

Характеристика	**Опис**
1	**2**
Тип шуму	Дискретний фрактальний шум
Виміри	Двовимірні
Переваги	Простота реалізації, масштабованість, контроль рівня деталізації
Недоліки	Менш природний вигляд, схильність симетрії
Застосування	Генерація гірських ландшафтів, моделювання терену в стратегіях

Процес створення рельєфу – від ініціалізації кутових точок до додавання центральних і крайових – демонструє, як формується деталізована карта висот, що показано на рисунку 1.2.

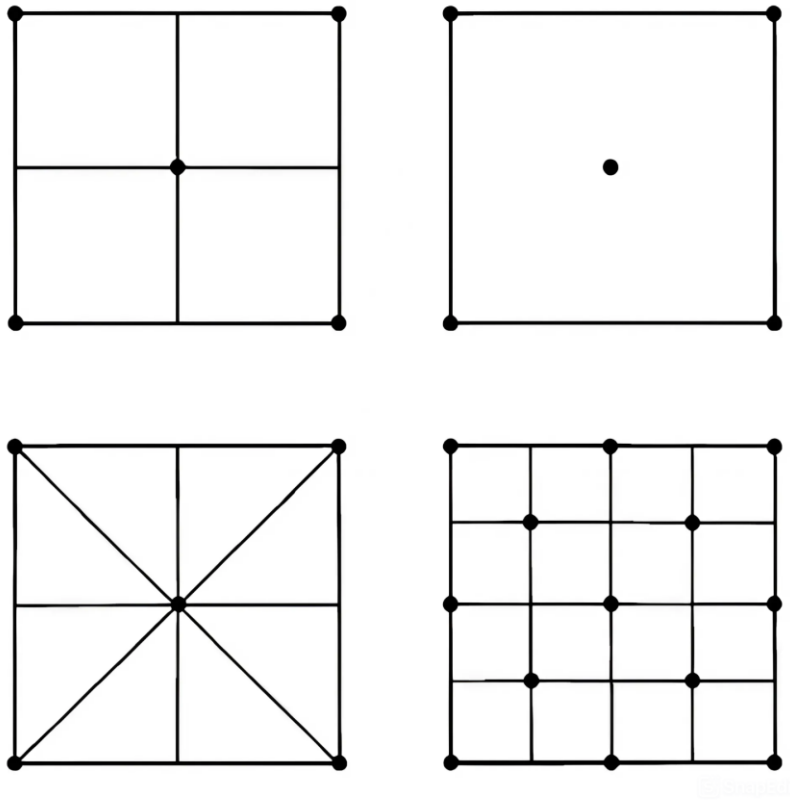

Рисунок 1.2 – Візуалізація алгоритму Diamond-Square

Simplex Noise – це удосконалений алгоритм шуму, створений Кеном Перліном у 2001 році як відповідь на недоліки класичного шуму. Його оптимізовано для багатовимірних просторів, що є критичним у сучасній графіці та симуляціях [14]. В основі алгоритму лежать симплекси – найпростіші геометричні форми в різних вимірах (наприклад, трикутник у двовимірному просторі, тетраедр – у тривимірному). Така структура дозволяє ефективно генерувати шум без характерних артефактів класичного підходу.

Формула базується на трикутному розбитті простору або тетраедричному розбитті, що зменшує обчислювальну складність порівняно з шумом Перліна має вигляд:

$$f(x,y) = \sum_i w_i \cdot g_i \cdot P(x - x_i, y - y_i), \quad (1.3)$$

де $P(x,y)$ – базова функція шуму, яка визначає вплив кожного симплекса;

w_i – ваговий коефіцієнт, що залежить від відстані до центру симплекса;

g_i – напрямні вектори для градієнтів, які визначають зміну значень;

x_i, y_i – координати вершин відповідного симплекса.

Основні характеристики алгоритму Simplex Noise, що охоплюють його технічні особливості, переваги, недоліки та застосування в багатовимірному просторі, становлять основу для оцінки його ефективності в задачах процедурної генерації. Усі ці аспекти узагальнено й показано у таблиці 1.4.

Таблиця 1.4 – Характеристики Simplex Noise

Характеристика	**Опис**
1	**2**
Тип шуму	Гладкий, фрактальний, із трикутним розбиттям простору
Періодичність	Неперіодичний за замовчуванням, можливе введення періодичності вручну
Гладкість	Висока – забезпечує плавні переходи між точками
Фрактальність	Підтримує октави для створення складних і деталізованих шумових структур
Виміри	Двовимірні, тривимірні і вище
Переваги	Вища продуктивність, менше артефактів, краще масштабування в багатовимірному просторі
Недоліки	Складність реалізації, менша інтуїтивність при налаштуванні параметрів
Застосування	Генерація процедурного рельєфу, симуляція природи, створення реалістичних анімацій і текстур

Рівномірна фрактальна структура рельєфу з плавними переходами між висотами, без помітних артефактів, створює природоподібні візерунки без чітких повторів, що надає зображенню органічного вигляду. Завдяки використанню компактного трикутного розбиття простору (на відміну від квадратного у Перлін-шумі), результат виглядає природно та гармонійно, що особливо важливо для тривимірних сцен і динамічних середовищ, як показано на рисунку 1.3.

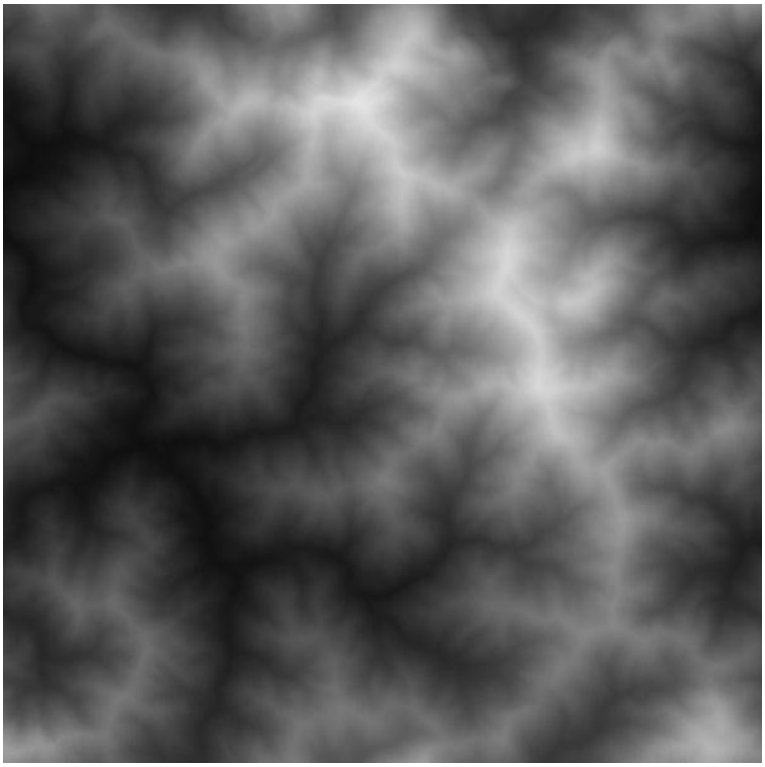

Рисунок 1.3 – Приклад карти висот, створеної за допомогою Simplex Noise

Основні відомості про тип шуму або принцип генерації, характерні переваги, а також типові недоліки кожного з розглянутих алгоритмів процедурної генерації рельєфу подано у порівняльній характеристиці. Це дозволяє здійснити більш обґрунтований вибір алгоритму залежно від цілей проєкту, вимог до візуального результату, складності реалізації та обчислювальної ефективності [15].

Порівняння алгоритмів Diamond-Square, шуму Перліна та Simplex Noise показано у таблиці 1.5.

Таблиця 1.5 – Порівняння алгоритмів шум Перліна, Diamond-Square та Simplex

Алгоритм	**Тип шуму/генерації**	**Переваги**	**Недоліки**
1	**2**	**3**	**4**
Шум Перліна	Градієнтний фрактальний шум	Плавні переходи, природний вигляд, фрактальна структура	Вища обчислювальна складність, складність реалізації
Diamond-Square	Дискретний фрактальний шум	Простота реалізації, масштабованість, контроль рівня деталізації	Менш природний вигляд, схильність до симетрії, різкі переходи
Simplex Noise	Градієнтний фрактальний шум з симплексним розбиттям	Висока продуктивність у тривимірному просторі, менше артефактів, масштабованість	Складність реалізації, менш інтуїтивне налаштування

1.2.3 Алгоритми ерозії

Щоб зробити згенеровані рельєфи більш правдоподібними, у процедурному моделюванні ландшафтів часто застосовують алгоритми ерозії. Вони повторюють вплив природних процесів, таких як вивітрювання, дія води або вітру, що формують поверхню місцевості.

Серед найпоширеніших алгоритмів можна виділити:

– термальна ерозія – цей алгоритм моделює зсуви ґрунту або кам’яних мас під впливом сили тяжіння. Якщо нахил між двома суміжними точками перевищує встановлене значення, частина матеріалу «сповзає» вниз, вирівнюючи поверхню;

– гідравлічна ерозія – цей алгоритм відображає вплив води на рельєф. Алгоритм враховує випадання опадів, формування потоків і перенесення частинок із подальшим їх відкладенням. Хоча цей метод є доволі складним з обчислювальної точки зору, він дає дуже реалістичні результати [16];

– вітрова ерозія – цей алгоритм застосовується переважно для створення пустельних чи посушливих ландшафтів. Хоча її ефект на загальну геометрію місцевості незначний, вона додає атмосферності сцені [17].

Кожен із цих підходів має свої переваги й обмеження, зокрема щодо застосування в реальному часі. Основні характеристики різних типів ерозії показано у таблиці 1.6.

Таблиця 1.6 – Порівняльна характеристика основних видів ерозії

Тип ерозії	**Переваги**	**Недоліки**	**Використання в реальному часі**
1	**2**	**3**	**4**
Термальна	Легка реалізація	Менш реалістичний ефект	Висока придатність
Гідравлічна	Висока відповідність природі	Високі витрати ресурсів	Обмежене використання
Вітрова	Придатна до пустельних умов	Незначний вплив на загальну форму рельєфу	Обмежене використання

1.3 Огляд існуючих інструментів

1.3.1 Основні рушії для тривимірної графіки

У розробці тривимірного середовища та візуалізації простору активно застосовуються спеціалізовані рушії та бібліотеки, які дають змогу як створювати високоякісні сцени вручну, так і автоматизувати цей процес за допомогою процедурних підходів. Вибір конкретного інструменту залежить від низки чинників: технічних вимог проєкту, особливостей цільової платформи, наявного досвіду розробника, а також складності графіки, яку необхідно реалізувати.

Одними з найпоширеніших платформ, які використовуються як у промисловості, так і в академічних дослідженнях, є Unity, Unreal Engine та SceneKit [18]. Окрім них, варто згадати менш масштабні, проте функціонально гнучкі рішення – Godot, Three.js, Blender, які також знаходять своє застосування в межах вузькоспеціалізованих задач.

Unity – це рушій, який забезпечує баланс між продуктивністю, функціональністю та зручністю розробки [19]. Платформа підтримує створення тривимірних сцен завдяки вбудованій системі Terrain System, яка дозволяє редагувати рельєф вручну або завантажувати карти висот ззовні. Завдяки широким можливостям програмування на C# можна реалізовувати складні алгоритми генерації, включно з реакцією на взаємодії у реальному часі.

Іншою перевагою є наявність великої кількості додаткових інструментів – як-от Gaia, MapMagic чи MicroSplat – які значно спрощують розробку процедурних ландшафтів навіть для користувачів із базовим рівнем знань. Крім того, Unity має потужну підтримку кросплатформеності: створені проєкти можна запускати на Windows, Android, iOS, WebGL та інших системах. Проте для проєктів з високим графічним навантаженням можливості рушія можуть потребувати додаткової оптимізації.

Unreal Engine – це рушій, який орієнтований передусім на розробку графічно насичених проєктів [20]. Його Landscape System дозволяє створювати великі відкриті світи з реалістичною деталізацією, а також поєднувати різні типи текстур, шарів, освітлення та матеріалів. Для реалізації процедурної логіки в UE можна використовувати як традиційне програмування на C++, так і візуальне моделювання через Blueprints, що відкриває доступ до гнучкої інтеграції процедурних функцій без потреби в глибоких знаннях програмування.

У версії UE5 впроваджено нові технології, як-от Nanite та Lumen, що дозволяють досягати надзвичайно реалістичної візуалізації. Проте варто враховувати, що розробка під мобільні платформи за допомогою UE може виявитися ресурсомісткою, особливо при реалізації динамічних процедурних сцен, що обмежує його використання в мобільних або AR-застосунках.

SceneKit – це рушій, призначений для створення тривимірних сцен переважно в межах екосистеми Apple. Він орієнтований на розробку застосунків для iOS, macOS, tvOS та watchOS і забезпечує зручне середовище для побудови інтерактивної тривимірної графіки з високим рівнем інтеграції з мовою Swift. Платформа підтримує поєднання з іншими фреймворками Apple, такими як ARKit (для доповненої реальності), Metal (низькорівневе API для рендерингу), а також CoreMotion та SpriteKit.

Попри відсутність спеціалізованої системи ландшафтів, як-от Terrain System у Unity чи Landscape System в UE, SceneKit дозволяє реалізовувати процедури генерації на рівні геометрії за допомогою SCNGeometrySource та SCNGeometryElement. Це відкриває доступ до створення складних форм рельєфу шляхом програмної модифікації сітки у реальному часі, з використанням алгоритмів шуму, ерозії або інших математичних моделей.

Окрім широко вживаних рушіїв, таких як Unity, Unreal Engine чи SceneKit, існують і менш масштабні, але гнучкі та перспективні платформи, які також можуть використовуватись у процедурній генерації тривимірних сцен.

Godot Engine – це легкий рушій із відкритим кодом, що стрімко набирає популярності завдяки інтуїтивно зрозумілому інтерфейсу та активній спільноті розробників. Підтримує власну мову програмування GDScript і забезпечує необхідний функціонал для створення тривимірного контенту.

Three.js – це JavaScript-бібліотека, яка дозволяє створювати тривимірну графіку безпосередньо у вікні браузера. Чудово підходить для реалізації інтерактивних візуалізацій або вебзастосунків із тривимірними елементами.

Blender – це універсальний інструмент для тривимірного моделювання, який, хоча й не призначений для генерації в реальному часі, має потужний Python API та систему Geometry Nodes. Завдяки цим можливостям він часто використовується для підготовки моделей або створення заготовок сцен.

Основні параметри цих інструментів, такі як підтримка мобільних платформ, гнучкість у реалізації алгоритмів, продуктивність і складність освоєння, дають змогу чітко порівняти їхні переваги та обмеження, що особливо важливо для обґрунтованого вибору відповідно до цілей конкретного проєкту. Для наочності всі ці характеристики зведено й показано у таблиці 1.7.

Таблиця 1.7 – Порівняння популярних інструментів тривимірної генерації

Платформа	**Підтримка мобільних платформ**	**Гнучкість генерації**	**Продуктивність**	**Складність**
1	**2**	**3**	**4**	**5**
Unity	Висока	Висока	Висока	Середня
Unreal Engine	Середня	Висока	Висока	Висока
SceneKit	Висока	Середня	Висока	Низька
Godot	Середня	Середня	Середня	Низька
Three.js	Висока	Середня	Середня	Середня
Blender	Немає	Висока	Висока	Висока

1.3.2 Обґрунтування вибору рушія

Під час планування реалізації генератора тривимірного ландшафту було прийнято рішення використовувати платформу SceneKit. Цей вибір став результатом детального порівняння кількох поширених рішень, серед яких були такі популярні рушії, як Unity та Unreal Engine. Визначальним фактором стала відповідність SceneKit ключовим вимогам проєкту – потребі створення компактного, стабільного та графічно привабливого застосунку для iOS, що працює з тривимірною сценою у режимі реального часу [21].

Однією з головних переваг SceneKit виявилася його повна сумісність із екосистемою Apple. Для роботи з iOS-платформами критично важливо мати інструмент, який не лише підтримує ці пристрої, а й оптимізований під їхню архітектуру. У цьому контексті SceneKit забезпечує ефективне використання ресурсів, оскільки не потребує сторонніх бібліотек або проміжних рівнів абстракції.

Ще одним вирішальним аргументом стала підтримка прямого створення та керування геометрією. Це дає змогу реалізовувати алгоритмічні методи побудови рельєфу, які базуються на обчисленнях висот та формуванні сітки поверхні. Завдяки SCNGeometrySource і SCNGeometryElement можна програмно задавати вершини, нормалі та інші параметри – без потреби у використанні готових шаблонів чи редакторів. Такий підхід дозволив зосередитися на точній математичній моделі генерації ландшафту.

Також варто згадати про тісну інтеграцію SceneKit з мовою Swift. Це значно полегшує реалізацію логіки, прискорює розробку та забезпечує хорошу читабельність коду. Крім того, використання Swift разом із Xcode спрощує тестування та налагодження проєкту в середовищі, максимально адаптованому для роботи з пристроями Apple.

SceneKit також демонструє високу продуктивність навіть на пристроях початкового рівня. При додаванні анімацій, джерел освітлення або базових

фізичних властивостей не виникає помітних затримок, що особливо важливо для мобільних застосунків. Крім того, застосунки, створені на SceneKit, мають невеликий розмір інсталяційного файлу, що сприяє швидкому завантаженню та встановленню.

У підсумку, SceneKit повністю відповідає технічним і функціональним потребам, поставленим у межах цього проєкту. Обрана технологія не лише забезпечила необхідну гнучкість і продуктивність, але й сприяла глибшому розумінню принципів процедурної генерації тривимірних сцен для мобільного середовища.

1.4 Постановка задачі

Таким чином, процедурна генерація тривимірних ландшафтів є актуальним завданням для розробки сучасних інтерактивних застосунків, зокрема для візуалізації природних сцен, створення ігрових рівнів, симуляцій навколишнього середовища та освітніх застосунків.

Особливої актуальності ця проблема набуває у контексті мобільних платформ, де існує потреба у продуктивних, оптимізованих алгоритмах генерації, які здатні працювати в умовах обмежених ресурсів пристрою. З урахуванням цього ставиться завдання розробки застосунку для iOS, що поєднує в собі алгоритми процедурної генерації ландшафту з можливістю візуалізації результату у тривимірному просторі за допомогою SceneKit.

Об’єктом роботи є алгоритми генерації тривимірного ландшафту на мобільних пристроях.

Метою роботи є створення застосунку для iOS, що реалізує генерацію тривимірного ландшафту на основі карти висот із візуалізацією у середовищі SceneKit.

Для досягнення мети необхідно вирішити такі завдання:

– аналіз сучасних методів процедурної генерації тривимірних рельєфів, зокрема шуму Перліна, Diamond-Square та їх варіацій;

– характеристика технологічних можливостей платформи iOS, що забезпечують реалізацію генеративних алгоритмів у межах мобільного застосунку;

– обґрунтування вибору переваг мови програмування Swift та середовища Xcode як основних інструментів розробки програмного забезпечення;

– вивчення функціоналу SceneKit як рушія для відображення тривимірних сцен, побудови геометричних моделей ландшафту та накладання текстур;

– реалізація алгоритму генерації карти висот, що слугує основою для побудови топографічної структури тривимірного ландшафту;

– побудова полігональної сітки для відображення геометрії ландшафту, з урахуванням відповідності між елементами сітки та значеннями карти висот;

– автоматичне розфарбування ділянок сцени відповідно до висотних рівнів (водні поверхні, рівнини, гори, снігові вершини);

– генерація та розміщення об’єктів довкілля (дерева, каміння тощо) залежно від типу місцевості;

– розробка інтерфейсу взаємодії користувача з можливістю масштабування, обертання сцени, зміни параметрів генерації та повторного створення ландшафту;

– тестування працездатності застосунку на реальних пристроях з операційною системою iOS, оптимізація рендерингу та мінімізація затримок при генерації сцени.

2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТАРІЙ РОЗРОБКИ

2.1 Основні технології

Розробка застосунку для генерації тривимірних ландшафтів вимагає вибору таких технологій, які не тільки забезпечують графічну продуктивність і підтримку апаратного прискорення, а й гарантують зручність розробки, підтримку актуального інструментарію та відповідність архітектурі сучасних мобільних пристроїв. Одним з важливих етапів було визначення стеку технологій, який найкраще відповідав би меті реалізації генерації динамічних ландшафтів на пристроях з операційною системою iOS. Основними критеріями відбору були стабільність, продуктивність, гнучкість, а також ступінь інтеграції з іншими інструментами екосистеми Apple.

Для реалізації логіки та взаємодії користувача із застосунком було обрано мову програмування Swift. Вона забезпечує високу продуктивність виконання, а також має багату стандартну бібліотеку, що дозволяє ефективно обробляти графіку, працювати з просторовими структурами даних і керувати ресурсами пристрою. У свою чергу, графічний рушій SceneKit дав змогу організувати візуалізацію тривимірних моделей ландшафту без використання низькорівневих API, таких як Metal або OpenGL, що значно спростило процес реалізації проєкту. Для побудови користувацького інтерфейсу застосовано UIKit – основний інструмент для побудови віконної структури застосунку в середовищі iOS.

В основі роботи генератора ландшафтів лежить побудова тривимірної сітки з координатами за осями *x*, *y*, *z*. За координатами *x* та *z* відбувається побудова площини, а значення *y* (висота) визначається за допомогою алгоритмів генерації висот, що можуть базуватися на шумі Перліна або інших стохастичних методах.

Для ілюстрації загальної архітектури побудови ландшафту у застосунку, доцільно звернутися до схематичного зображення взаємодії між

основними компонентами, які формують результат візуалізації. Дані висот перетворюються у геометрію, а потім виводяться на екран через SceneKit показано на рисунку 2.1.

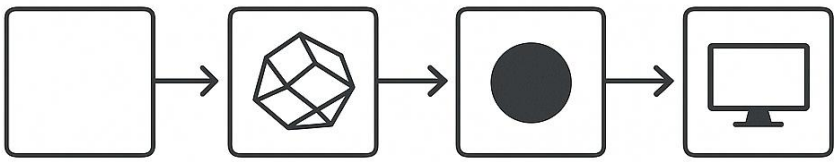

Рисунок 2.1 – Схема взаємодії компонентів генерації ландшафту

Окрім геометрії, значну роль відіграє система матеріалів. Залежно від значення висоти, для кожної точки сітки задається відповідний колір або текстура. Це дозволяє створювати візуальні ефекти гір, рівнин, води чи снігових піків.

Для більш детального уявлення про структуру даних, використаних у побудові ландшафту, де вказано різні підходи до генерації висот і їхню продуктивність при різних параметрах сітки показано у таблиці 2.1.

Таблиця 2.1 – Порівняння методів генерації висоти в залежності від продуктивності

Метод	**Час генерації (50×50), секунда**	**Візуальна складність**	**Придатність для мобільних пристроїв**
1	**2**	**3**	**4**
Шум Перліна	0,014	Висока	Висока
Diamond-Square	0,021	Середня	Висока
Simplex Noise	0,005	Низька	Висока

У результаті експериментів визначено, що саме метод на основі шуму Перліна забезпечує найкраще співвідношення між продуктивністю та якістю для потреб застосунку. Формально, обчислення висоти на основі двовимірного шуму можна виразити як:

$$h(x,z) = noise(x \cdot f, z \cdot f) \cdot a, \tag{2.1}$$

де f – коефіцієнт частоти;

a – амплітуда;

$noise()$ – функція генерації значень шуму.

Приклад виведеного ландшафту у SceneKit із різними рівнями деталізації, показано на рисунку 2.2.

Рисунок 2.2 – Візуалізація згенерованого ландшафту у SceneKit із застосованими матеріалами на основі висоти

Поєднання Swift, SceneKit та UIKit дозволяє реалізувати ефективне створення та рендеринг процедурно згенерованих тривимірних ландшафтів з високою продуктивністю, інтуїтивним інтерфейсом та масштабованістю. Цей підхід забезпечує не тільки функціональність, але й простоту модифікації в майбутньому – як на рівні графіки, так і логіки.

2.1.1 Swift (або Objective-C)

На етапі розвитку мобільного програмування перед розробниками стоїть вибір між кількома мовами, які дозволяють створювати застосунки під операційну систему iOS. Найпоширенішими серед них є Swift та Objective-C. Враховуючи особливості задачі генерації тривимірних ландшафтів, важливо було обрати мову, яка б одночасно забезпечувала високу продуктивність, безпечне управління пам’яттю, зручну роботу з математичними структурами, графікою, і в той же час – легкість у розробці інтерфейсу.

Swift, як сучасна мова програмування, створена Apple спеціально для розробки під платформи iOS, macOS, watchOS та tvOS, значно перевершує свого попередника Objective-C за багатьма параметрами [22]. Основною перевагою є її синтаксична лаконічність, що підвищує швидкість розробки й знижує вірогідність помилок. Порівнюючи Swift із Objective-C, можна зробити висновок, що перша забезпечує більшу читабельність, простішу підтримку коду, кращу безпеку типів та значно ширшу підтримку сучасних парадигм програмування.

Однією з головних причин вибору Swift стало активне супроводження та розвиток мови самою компанією Apple, а також відкритий вихідний код, що дозволяє залучати розробників із різних платформ [23]. У 2023 році Apple представила вже п’яте велике оновлення мови, в якому було вдосконалено роботу з паралельним виконанням, підтримку одночасності та впроваджено нові можливості для роботи з C API. У контексті тривимірної генерації, де важлива обробка великих обсягів даних та відтворення графіки в реальному часі, такі покращення відіграють критичну роль [24].

Варто зазначити, що Swift значно ефективніше працює з арифметичними операціями на масивах, що часто використовуються у процедурній генерації ландшафтів. Зокрема, Swift дозволяє здійснювати паралельну обробку масивів із використанням GCD (Grand Central Dispatch), що підвищує продуктивність. На відміну від Objective-C, Swift має прозору й

сучасну систему керування пам’яттю, що базується на ARC (Automatic Reference Counting), проте є менш схильною до помилок.

Порівняння ключових параметрів мов Swift та Objective-C у контексті розробки застосунків з інтенсивною графічною складовою показано у таблиці 2.2.

Таблиця 2.2 – Порівняння Swift та Objective-C для графічних застосунків

Параметр	**Swift**	**Objective-C**
1	**2**	**3**
Безпека типів	Статична, з підтримкою опціоналу	Динамічна, менш безпечна
Синтаксис	Сучасний і лаконічний	Вербозний та застарілий
Продуктивність	Висока, оптимізована під iOS	Трохи нижча при однакових умовах
Паралельність	Вбудована підтримка асинхронності (async)/очікування (await)	Через GCD, складніше реалізувати
Підтримка Apple	Основна мова розробки	Лише зворотна сумісність

Однією з особливостей Swift є гнучкість у роботі з математичними виразами та підтримка операторів високого рівня. Завдяки цьому можна ефективно реалізовувати алгоритми генерації ландшафтів. Наприклад, при реалізації процедурної генерації часто застосовується нормалізація значень, яка в Swift може бути реалізована просто і виразно.

У реальних умовах генерації ландшафтів важливо нормалізувати висоти до діапазону від 0 до 1 – це необхідно для коректного застосування текстур або кольорових градієнтів. Крім того, Swift забезпечує зручну роботу з векторними структурами, що спрощує оперування точками у тривимірному просторі під час створення рельєфу.

Із точки зору обробки графіки, Swift забезпечує зручність у роботі з тривимірними даними та легко інтегрується у типові обчислювальні процеси. Завдяки підтримці перевантаження операторів мова дозволяє створювати читабельний та інтуїтивно зрозумілий код. Це особливо корисно в застосунках, де активно відбувається обчислення напрямків, нормалей та градієнтів – усіх тих характеристик, які потрібні при побудові ландшафтної сітки. Приклад структурної схеми роботи одного з модулів генерації висоти у Swift показано на рисунку 2.3.

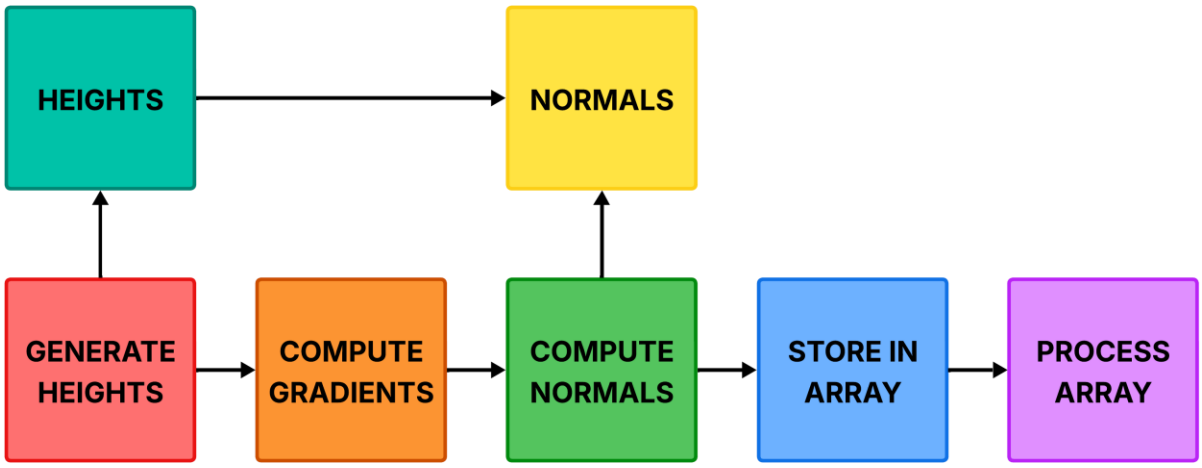

Рисунок 2.3 – Схема обробки масиву висот та нормалей

Swift забезпечує ефективнішу інтеграцію з модульною архітектурою застосунку. Завдяки сучасним засобам інкапсуляції та дженерікам, більшість частин логіки генерації могли бути повторно використані або протестовані ізольовано. Наприклад, функції генерації шуму або структури геометрії реалізовано у вигляді окремих сервісів, що спрощує як тестування, так і супровід.

Приклад візуалізації результату обчислення векторів напрямку (нормалей) до кожної вершини сітки, ці нормалі потім використовуються для освітлення поверхні у тривимірному просторі показано на рисунку 2.4.

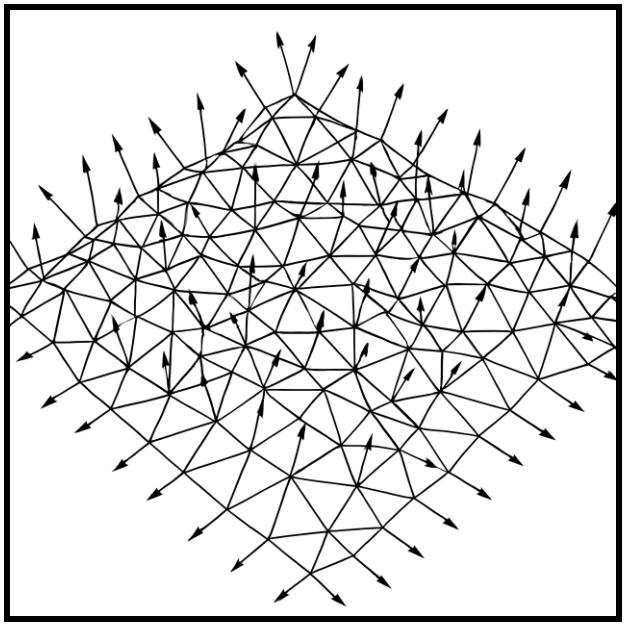

Рисунок 2.4 – Нормалі, обчислені до вершин сітки, що використовуються у SceneKit для освітлення поверхні

У Swift подібні обчислення здійснюються зручно і швидко завдяки вбудованим структурам та методам обчислень. У контексті оптимізації, застосунок використовує можливість попереднього кешування частин обчислень у структурі, що значно знижує навантаження при повторному рендерингу.

У контексті обчислювальної ефективності Swift демонструє високі показники оптимізації під ARM-архітектуру, що є особливо важливим для iOS-пристроїв. Завдяки цьому Swift-код, який компілюється з використанням LLVM-компілятора, здатен створювати машинні інструкції, які ефективніше використовують кеш-пам’ять, регістри процесора та паралельні обчислення. Під час аналізу застосунку було встановлено, що основні обчислення координат та їх трансформацій займають не більше 5% загального часу кадру, що вказує на високу продуктивність Swift у графічних задачах.

Swift забезпечує стабільно вищу продуктивність, що пов’язано як із сучасною структурою мови, так і з її оптимізацією під нові покоління процесорів. Тим не менш, Objective-C і далі залишається актуальним у великих проєктах, де важлива зворотна сумісність або існує потреба в

інтеграції з бібліотеками на мові C. У випадку побудови тривимірного ландшафту, де вся логіка розробляється з нуля, Swift виступає оптимальним рішенням завдяки кращому балансу між читабельністю, швидкодією та сучасними можливостями фреймворків Apple.

Ще однією перевагою Swift є система безпеки. Використання опціональних типів дозволяє уникати багатьох поширених помилок, пов'язаних із нульовими посиланнями. Це особливо важливо у графічних застосунках, де некоректне звернення до ресурсів може призвести до аварійного завершення. У Swift в таких випадках виконується явна перевірка на відсутність, що дозволяє створювати надійніші застосунки.

Також варто згадати концепцію структури у Swift, яка підтримує значущу семантику. Для геометрії, координат, кольорів – це дозволяє уникати зайвих витрат на копіювання та передачу даних.

Завдяки значення-тип, кожна така структура копіюється швидко і без необхідності управління посиланнями, що позитивно впливає на швидкість роботи при обробці великих масивів даних.

Крім цього, Swift дозволяє використовувати математичні функції з розширеної бібліотеки Foundation або через підключення модулів із C. Логарифмічна нормалізація висоти визначається формулою:

$$h'(x,z) = \frac{\log(1+h(x,z))}{\log(1+H)}, \tag{2.2}$$

де $h(x,z)$ – висота на координаті;

H – максимальна висота на ділянці.

Застосування подібного підходу дозволяє візуально вирівняти різкі перепади висоти, роблячи рельєф більш плавним.

Порівняння виводів сцени до та після застосування логарифмічного згладжування, показано на рисунку 2.5.

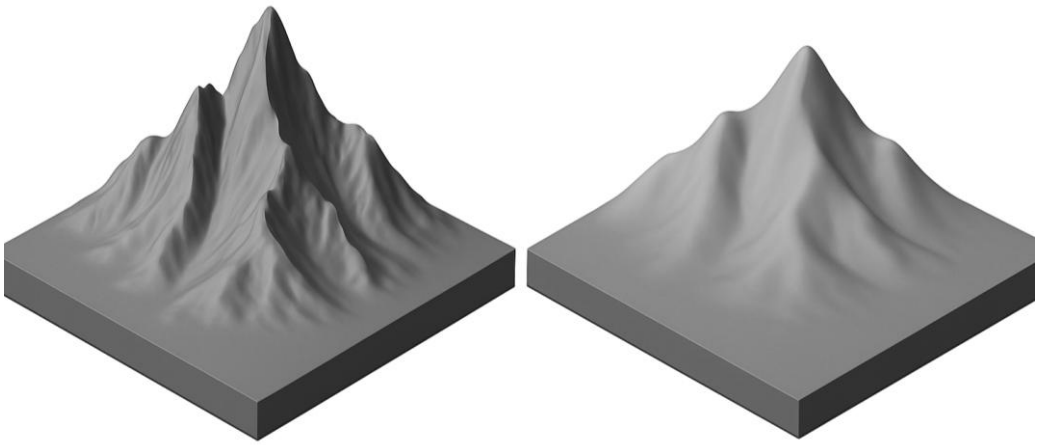

Рисунок 2.5 – Ефект логарифмічного згладжування

Swift забезпечує кращу підтримку багатопоточності завдяки новим ключовим словам асинхронно, очікування, а також структурам завдання і актор. Це дозволяє розділити обробку генерації, обчислення матеріалів та рендеринг на паралельні потоки. Завдяки цьому візуальне оновлення сцени не переривається, а користувач не відчуває затримок під час обробки нових ділянок ландшафту.

Для кращого розуміння структури асинхронного оновлення сцени доцільно подати схематичне представлення, яке демонструє порядок виконання паралельних задач, показано на рисунку 2.6.

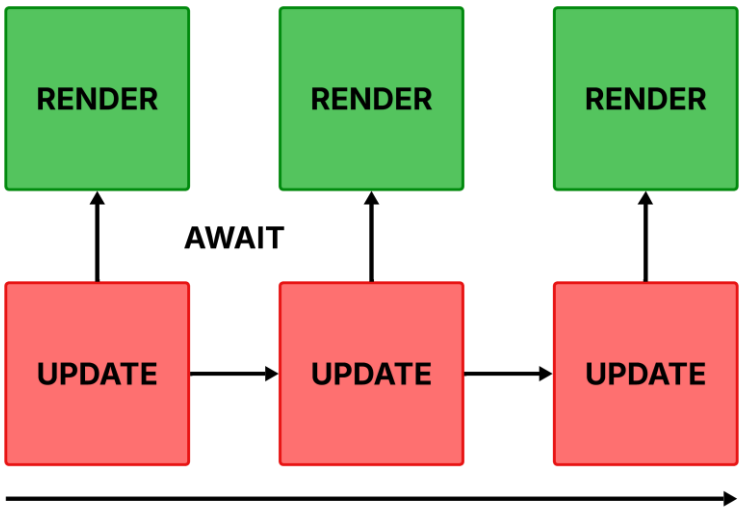

Рисунок 2.6 – Схематичне представлення оновлення сцени з використанням Swift асинхронності/очікування механізму

Такий підхід дає змогу не тільки ефективно використовувати апаратні ресурси пристрою, а й досягти високої плавності графіки, що критично для застосунків із динамічною тривимірною візуалізацією.

Усі наведені аргументи демонструють, що Swift – це не лише сучасна, безпечна та ефективна мова програмування, а й практичний інструмент для реалізації застосунків з високою графічною складністю, таких як генерація тривимірних ландшафтів [25].

Swift має активну спільноту, велику кількість актуальної документації та широку підтримку з боку Apple, що робить її стратегічно правильним вибором для розробників під iOS. Важливо також, що Swift добре інтегрується з існуючими фреймворками, полегшуючи створення масштабованих і надійних рішень.

2.1.2 SceneKit: можливості для тривимірної візуалізації

SceneKit є одним із провідних високорівневих графічних фреймворків, розроблених Apple для створення тривимірної графіки. Він дозволяє зосередитись на побудові сцени, анімаціях, освітленні, камерах і фізиці без потреби звертатися до низькорівневого коду на основі Metal або OpenGL [26]. Завдяки цьому SceneKit став ефективним інструментом для реалізації застосунку з генерації тривимірних ландшафтів.

Сцена в SceneKit представлена у вигляді ієрархії вузлів, кожен із яких може містити геометрію, матеріали, трансформації та фізичні властивості. Це дає змогу створювати складні структури, де дочірні елементи автоматично наслідують трансформації батьківських вузлів – наприклад, поєднати рельєф, воду, рослинність і освітлення в єдину композицію.

Перевагою SceneKit є підтримка кількох платформ – сцена, створена для iOS, може бути використана й на macOS. Також фреймворк тісно

інтегрується з іншими технологіями Apple, зокрема ARKit, що відкриває можливості доповненої реальності.

Візуальний вигляд об’єктів у SceneKit формується за допомогою матеріалів. Геометрія може мати кілька матеріалів, які задають параметри кольору, відбивання, шорсткості, карти нормалей тощо. У реалізованому застосунку матеріали створюються динамічно відповідно до висоти точок ландшафту – це дозволяє варіювати кольори або текстури залежно від умов середовища.

Приклад сцени ландшафту з кольоровим кодуванням висоти наведено на рисунку 2.7.

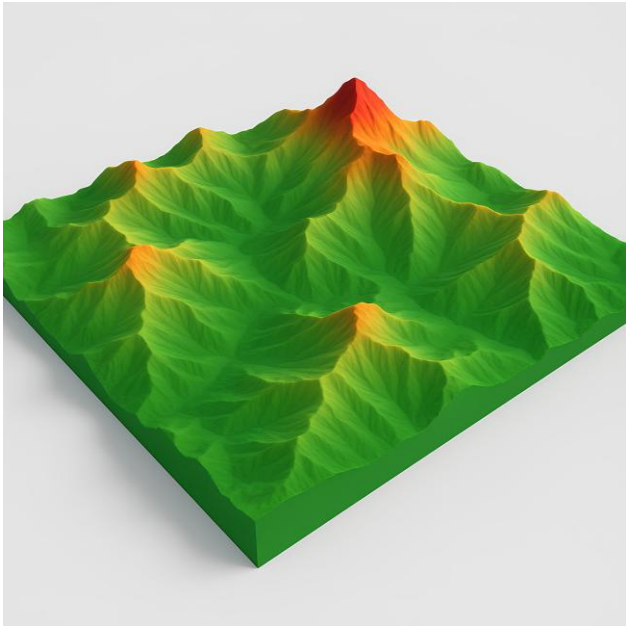

Рисунок 2.7 – Сцена генерації ландшафту в SceneKit з кольоровим кодуванням висоти

Для досягнення реалістичності сцени використовуються різні типи освітлення: фонове (рівномірне), точкове (імітація сонця) і напрямлене світло, яке створює тіні та підкреслює форму рельєфу. SceneKit має підтримку тіней, що значно покращує сприйняття простору. Напрямлене світло можна орієнтувати разом із вузлом, до якого воно прикріплене, змінюючи кут падіння відносно камери.

Для обчислення інтенсивності світла використовується модель Ламберта. Вона базується на косинусі кута між напрямком світла та нормаллю до поверхні, як показано у формулі:

$$I = I_0 \cdot \max(0, \vec{L} \cdot \vec{N}), \tag{2.3}$$

де I_0 – інтенсивність світла;

$\vec{L}$ – напрямок світла;

$\vec{N}$ – нормаль до поверхні.

Камера у SceneKit виконує ключову роль, дозволяючи фокусуватися на потрібній області сцени. Вона може автоматично орієнтуватися на об’єкти, що спрощує навігацію або візуалізацію центру ландшафту.

Структуру сцени з основними вузлами (камера, джерела світла, геометрія) зображено на рисунку 2.8.

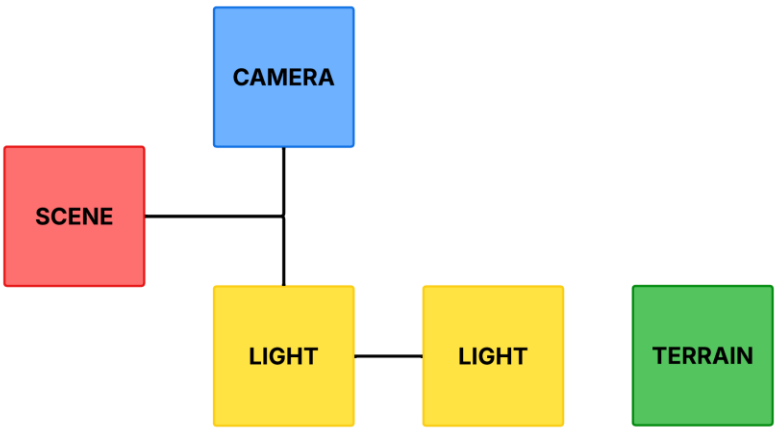

Рисунок 2.8 – Дерево вузлів SceneKit: камера, джерела світла та геометрія ландшафту

SceneKit також підтримує фізику об’єктів – можна задавати масу, тертя, прискорення. У рамках реалізованої сцени ці функції не використовувались активно, проте вони є перспективними для розширення застосунку, наприклад, для взаємодії рухомих елементів із рельєфом.

Фреймворк дозволяє імпортувати зовнішні тривимірні моделі у форматах .dae, .obj та інших, що відкриває можливості для додавання складніших об’єктів – дерев, будівель, скель тощо.

SceneKit має розвинену систему анімації, яка дає змогу використовувати часові інтервали, ключові кадри, а також інтеграцію з фізичним рушієм. Наприклад, можливо реалізувати поступове зміщення камери або зміну матеріалу з часом. У застосунку реалізовано автоматичне коливання камери вгору-вниз, що імітує огляд рельєфу – завдяки системному графічному рушію це не знижує продуктивність.

Важливою функцією, яка впливає на продуктивність, є підтримка рівнів деталізації (Level of Detail, LOD). SceneKit дозволяє завантажувати моделі з різним рівнем деталізації залежно від відстані до камери. Це дозволяє зменшити навантаження на графічний процесор без помітної втрати якості візуалізації.

Обчислення рівня деталізації ґрунтується на функції відстані до камери, як показано у формулах:

$$LOD = f(d), \tag{2.4}$$

$$d = \left\| \vec{P}_{камера} - \vec{P}_{об'єкт} \right\|, \tag{2.5}$$

де $\vec{P}_{камера}$ – інтенсивність світла;

$\vec{P}_{об'єкт}$ – нормаль до поверхні.

Це дозволяє змінювати рівень деталізації у реальному часі залежно від місця розташування камери. Такий підхід дає змогу зменшити використання оперативної пам’яті та запобігти зайвому навантаженню на графічний процесор. Крім того, це створює більш плавний користувацький досвід, особливо під час переміщення камери по складному ландшафту, як показано на рисунку 2.9.

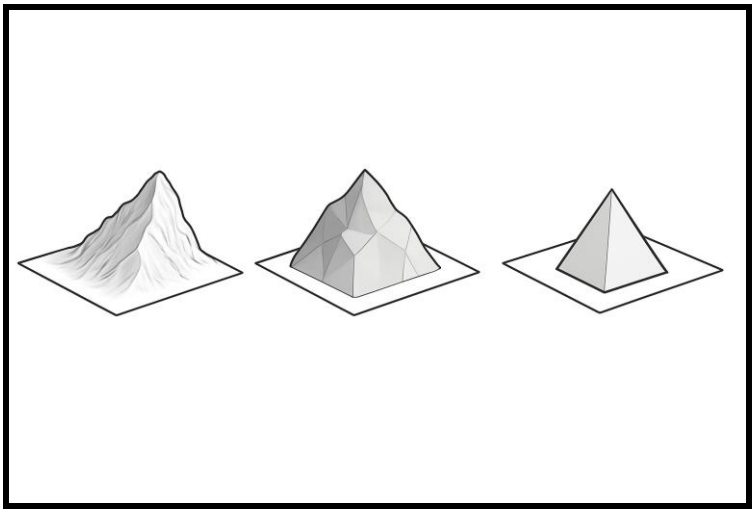

Рисунок 2.9 – Рівні деталізації у SceneKit

Щоб уникнути помітних змін у вигляді об’єктів при перемиканні рівнів деталізації, важливо правильно налаштувати порогові значення – це забезпечує плавний перехід між LOD-моделями.

SceneKit також підтримує відображення тіней у реальному часі, що позитивно впливає на реалістичність сцени, проте потребує додаткових ресурсів.

Для забезпечення якісного рендерингу у застосунку використано такі налаштування:

– активація мультисемплінгу;

– встановлення високої точності рендерингу;

– використання Metal, як рендеринг-бекенду.

SceneKit автоматично адаптує рендеринг до можливостей пристрою, що забезпечує сумісність з різними моделями iPhone або iPad.

Інтеграція з UIKit дозволяє вбудовувати тривимірну сцену безпосередньо в інтерфейс застосунку разом із класичними елементами управління – кнопками, слайдерами та текстовими полями.

Таким чином, SceneKit у поєднанні зі Swift забезпечує швидку розробку інтерактивних тривимірних середовищ із високим рівнем абстракції, що дозволяє зосередитись на логіці побудови сцени, зберігаючи контроль над деталями рендерингу.

2.1.3 Короткий огляд UIKit

UIKit є базовим фреймворком для створення інтерфейсів користувача у програмному забезпеченні під iOS. Він надає розробникам набір класів, структур і механізмів для створення екранів, контролерів, переходів та обробки взаємодії користувача [27]. Основною перевагою UIKit є гнучкість його архітектури, що дозволяє масштабувати інтерфейс від простих одновіконних застосунків до складних багатовіконних структур із великою кількістю динамічних елементів [28].

Фреймворк UIKit побудований навколо ієрархії елементів інтерфейсу та керуючих об’єктів. Кожен компонент інтерфейсу – кнопка, текстове поле, слайдер або панель – є частиною цієї структури та може бути позиціонований, масштабований або стилізований. Окремі об’єкти відповідають за логіку керування відображенням, життєвим циклом інтерфейсу та обробкою подій.

Для побудови адаптивного інтерфейсу, що коректно працює на різних діагоналях екранів iPhone та iPad, UIKit надає інструменти для опису взаємозв’язків між елементами інтерфейсу. Завдяки цьому компоненти можуть автоматично змінювати розміри та положення під час адаптації до різних екранів. Крім того, доступні засоби для зручного групування елементів у вертикальні чи горизонтальні контейнери.

У межах реалізації інтерфейсу UIKit забезпечує зручний спосіб реагування на дії користувача за допомогою системи обробників подій, що пов’язані з відповідними елементами. Це дозволяє, наприклад, реалізовувати виклик певної функції у відповідь на натискання кнопки чи зміну значення елемента керування.

Інтерфейси, створені з використанням UIKit, можуть бути організовані як у коді, так і за допомогою візуальних інструментів у середовищі розробки Xcode. Розробник має змогу компонувати екрани, визначати переходи між ними, а також керувати логікою навігації між частинами застосунку.

Ще однією перевагою UIKit є підтримка адаптивного дизайну, що дозволяє враховувати особливості пристроїв, зокрема розмір екрана, його орієнтацію, доступні функції та налаштування середовища. Це дає змогу реалізовувати, наприклад, підтримку темної теми або створення альтернативних інтерфейсів для різних типів пристроїв.

Окрім статичних елементів, UIKit підтримує роботу з розпізнаванням жестів, які дозволяють реагувати на натискання, свайпи, масштабування та інші дії, що часто застосовуються для перегляду вмісту, навігації або зміни властивостей об’єктів. Така взаємодія робить інтерфейс більш інтуїтивним і природним для користувача.

Цей підхід особливо корисний у застосунках, де важлива швидка і точна взаємодія з графічними елементами.

Він значно підвищує зручність користування програмами, що працюють з візуальним контентом. Крім того, його можна легко адаптувати для різних категорій пристроїв – від смартфонів до планшетів.

Масштабування елемента інтерфейсу може бути реалізоване через обробку жесту зведення та розведення пальців, як показано на рисунку 2.10.

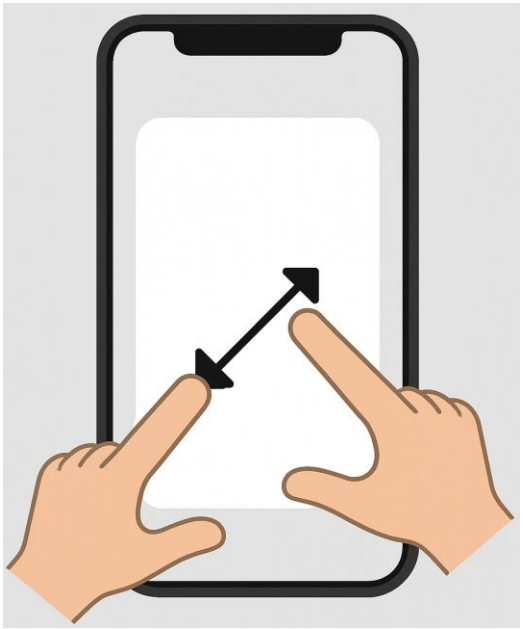

Рисунок 2.10 – Реалізація масштабування елемента інтерфейсу через жест зведення та розведення пальців у UIKit

Щоб пов’язати змінні параметри користувача з візуальними елементами інтерфейсу, фреймворк підтримує можливість передачі даних між елементами керування. Наприклад, значення, яке змінюється за допомогою слайдера, може одразу відображатися в текстовому полі або використовуватись у внутрішніх розрахунках.

Також UIKit забезпечує підтримку анімаційних ефектів, які застосовуються до властивостей елементів інтерфейсу. Це дає змогу створювати плавні переходи, змінювати розмір, положення або зовнішній вигляд об’єктів, що значно покращує загальне сприйняття програми з боку користувача.

Завдяки архітектурі та стабільності, UIKit залишається ключовим фреймворком для створення інтерфейсів у середовищі iOS. Його можливості охоплюють як прості, так і складні сценарії, включаючи адаптивний дизайн, взаємодію з жестами, візуальні ефекти, гнучку кастомізацію та підтримку широкого спектра пристроїв.

2.2 Інструменти розробки

У сучасному середовищі розробки тривимірних візуалізацій застосовується широкий спектр інструментів, які охоплюють усі етапи створення – від побудови сцени до оптимізації рендерингу та взаємодії з користувачем. Вибір програмного забезпечення залежить від завдань проєкту, технічних характеристик цільової платформи, продуктивності та типу графічного контенту. Особливо цінними є засоби, що підтримують тривимірну графіку на мобільних пристроях і дозволяють реалізовувати генерацію вмісту в реальному часі.

Однією з ключових частин процесу є інтегроване середовище розробки (IDE), яке забезпечує написання, компіляцію, налагодження та тестування коду. До інструментарію також належать графічні рушії, фреймворки,

математичні бібліотеки й засоби створення візуального контенту. Їхній вибір визначається не лише вимогами до якості зображення, але й потребою оптимізувати використання ресурсів пристрою.

Для моделювання сцен використовуються рушії, які підтримують тривимірну графіку, анімацію, освітлення та матеріали. Деякі з них дозволяють динамічно створювати або змінювати геометрію, що важливо для процедурної генерації. У складніших проєктах активно застосовуються бібліотеки для роботи з векторами, матрицями, трансформаціями та графічними шейдерами, що дає змогу реалізовувати точні алгоритми геометричної обробки та симуляції природних процесів.

Крім цього, інструменти для створення текстур і матеріалів, зокрема Blender, GIMP чи Substance, допомагають досягти реалістичності візуального оформлення. Загалом, розробка тривимірних застосунків потребує поєднання інструментів різного рівня – від низькорівневих бібліотек до комплексних платформ для побудови та візуалізації сцен. Раціональне використання цих засобів дозволяє збалансувати продуктивність, якість графіки та масштабованість застосунку.

Середовище Xcode є одним із найпотужніших засобів для створення програмного забезпечення в межах екосистеми Apple. Воно поєднує в собі текстовий редактор, компілятор, засоби налагодження, профілювання, проєктування інтерфейсу, тестування, а також емуляцію застосунків [29]. Розробка у цьому середовищі передбачає гнучке поєднання різних підходів – від архітектурного моделювання до створення складних графічних інтерфейсів.

Xcode підтримує використання кількох мов програмування, зокрема Swift, Objective-C та C++. У випадках, коли проєкт вимагає обробки складних графічних структур, поєднання Swift з відповідними фреймворками дозволяє досягти балансу між швидкодією та зручністю розробки. Початкове створення застосунку виконується на основі шаблонів, які визначають тип інтерфейсу, структуру файлів та загальні параметри проєкту.

Приклад робочого інтерфейсу Xcode зі сценою, побудованою засобами тривимірної графіки показано на рисунку 2.11.

Рисунок 2.11 – Робоче середовище Xcode під час створення сцени з використанням SceneKit

Однією з важливих особливостей середовища є підтримка модульної організації: функціональні частини застосунку, зокрема інтерфейс користувача, логіка, візуалізація та обчислення, можуть бути реалізовані як окремі компоненти. Упорядкування структури здійснюється за допомогою панелі навігації проєкту, де файли можуть бути згруповані за логікою без зміни їх фізичного розташування.

Інтерфейс користувача в Xcode можна створювати як візуально, так і програмним шляхом. Візуальне проєктування виконується засобами графічного редактора, а для динамічного підходу застосовується декларативна модель опису інтерфейсу. В обох випадках забезпечується інтеграція з основною архітектурою застосунку.

Середовище Xcode також надає повну підтримку тривимірної графіки. Це дає змогу створювати сцени з об’єктами, освітленням, камерами та матеріалами, які можна редагувати як вручну, так і програмно. Візуальний компонент, відповідальний за відображення сцени, інтегрується в основне вікно застосунку як стандартний елемент інтерфейсу.

Важливим етапом під час створення застосунків є перевірка продуктивності. У Xcode реалізовано набір інструментів для виявлення надмірного споживання ресурсів. Це дає змогу проаналізувати навантаження на систему та виявити компоненти, що потребують оптимізації. Особливу увагу при цьому слід приділяти не лише ефективності алгоритмів, а й правильній організації їх викликів у межах застосунку.

Для забезпечення стабільності та коректності обчислень передбачено засоби тестування, які дозволяють перевіряти логіку функцій на різних рівнях. Завдяки цьому можливо гарантувати, що зміни у програмному коді не вплинуть на загальну працездатність застосунку.

Ще одним важливим інструментом розробника є віртуальні пристрої, які дозволяють запускати застосунки без фізичного обладнання. Це дає змогу протестувати масштабованість інтерфейсу, поведінку рендерингу та адаптацію до різних розмірів екранів. Станом на 2025 рік підтримується емуляція сучасних моделей мобільних пристроїв, що дозволяє оцінювати застосунок у наближених до реальності умовах.

Для зручності перегляду результатів під час розробки Xcode також дозволяє створювати інтерактивні перегляди інтерфейсу. Такий підхід особливо корисний при використанні декларативної моделі опису інтерфейсу, яка активно впроваджується в останні роки..

Загальна архітектура проєкту в Xcode зазвичай включає логічні групи файлів і модулів, що показано на рисунку 2.12.

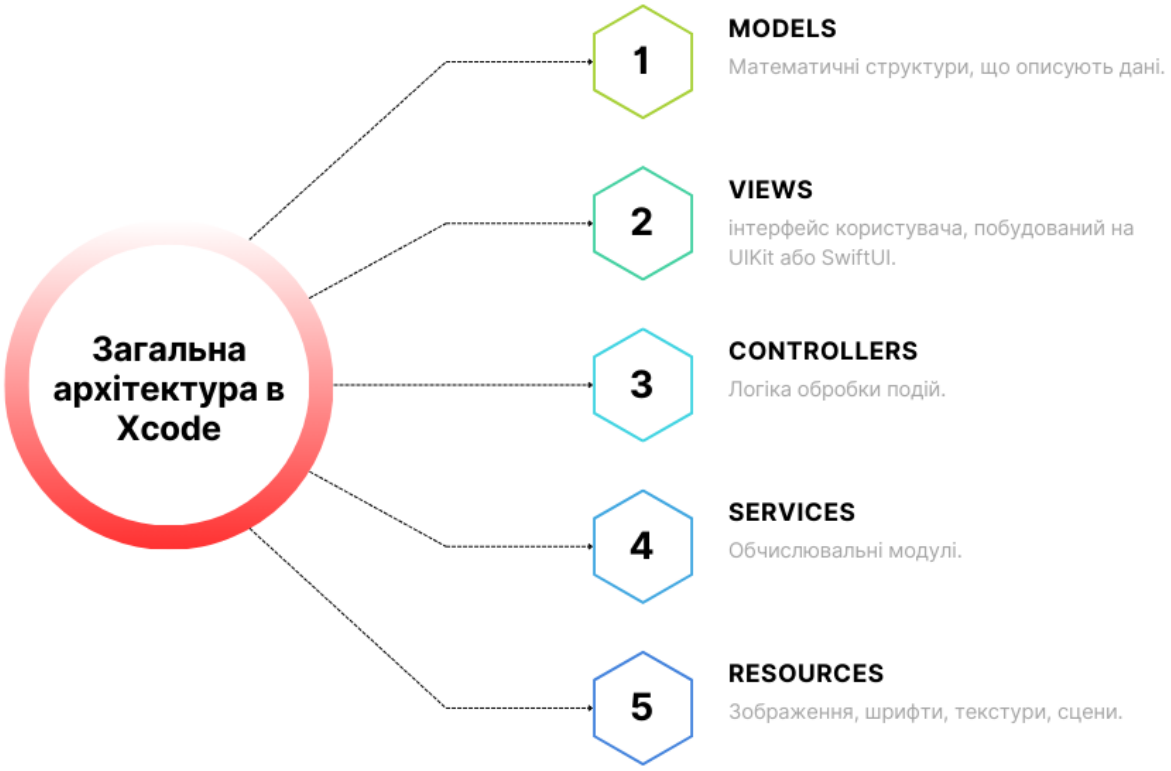

Рисунок 2.12 – Загальна архітектура проєкту в Xcode

Середовище розробки взаємодіє з вбудованими засобами контролю версій, що дає змогу зручно відстежувати зміни у коді, зберігати попередні стани проєкту та працювати над окремими функціями паралельно. Це особливо корисно під час командної розробки, де важливо швидко координувати зміни, поєднувати результати й керувати історією модифікацій. Можливість створювати ізольовані гілки для експериментів дозволяє уникати конфліктів і зберігати стабільність основної версії. Усі основні дії доступні безпосередньо з інтерфейсу середовища, без потреби у зовнішніх утилітах.

Xcode також дозволяє налаштовувати параметри збирання, що впливають на якість і стабільність застосунку. Серед них – вибір архітектури пристрою, рівень оптимізації, обмеження на використання ресурсів і конфігурації запуску. Це забезпечує гнучкість у тестуванні, налагодженні та розгортанні застосунків на різні пристрої, що сприяє підвищенню їх продуктивності..

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Архітектура застосунку

3.1.1 Загальна архітектура

Архітектура застосунку Landscape Generation побудована на принципах модульності та розділення відповідальності (Single Responsibility Principle). Такий підхід дозволяє забезпечити гнучкість системи, легкість підтримки та можливість подальшого розширення функціональності.

Основні принципи архітектури:

– модульність – модульність є ключовим принципом архітектури системи. Кожен компонент системи є самостійним модулем з чітко визначеними межами відповідальності. Це дозволяє розробляти та тестувати кожен модуль незалежно від інших, що значно спрощує процес розробки та підтримки. Модулі мають мінімальну залежність один від одного, що забезпечує гнучкість системи та можливість легкої заміни окремих компонентів. Взаємодія між модулями відбувається через чітко визначені інтерфейси, що забезпечує прозорість та передбачуваність поведінки системи;

– рівні абстракції – рівні абстракції в системі чітко розділені на три основні рівні. Високий рівень представлений компонентами TerrainScene та LandscapeViewController, які відповідають за управління сценою та користувацьким інтерфейсом. Ці компоненти забезпечують взаємодію з користувачем та керують загальним потоком даних у системі. Середній рівень представлений TerrainGenerator, який відповідає за генерацію ландшафту та координацію процесу створення рельєфу. Низький рівень включає TerrainMeshBuilder та NoiseGenerator, які відповідають за технічні аспекти побудови геометрії та генерації шуму відповідно;

– потік даних – потік даних в системі організований послідовно, починаючи з NoiseGenerator, який генерує базовий шум для створення

рельєфу. Ці дані передаються в TerrainGenerator, який формує карту висот та додає додаткові характеристики ландшафту. TerrainMeshBuilder використовує цю карту для створення полігональної сітки, яка потім передається в TerrainScene для візуалізації за допомогою SceneKit.

Переваги обраної архітектури:

– масштабованість системи забезпечується можливістю легкого додавання нових алгоритмів генерації та розширення функціональності. Система дозволяє просто інтегрувати нові типи візуалізації та адаптуватися до змінних вимог;

– підтримка системи спрощується завдяки ізольованим компонентам, які легше тестувати та налагоджувати. Локалізовані зміни не впливають на інші частини системи, що забезпечує стабільність роботи;

– гнучкість архітектури дозволяє легко замінювати окремі компоненти та адаптувати систему до нових вимог. Це забезпечує можливість простого додавання нових функцій та модифікації існуючих;

– продуктивність системи оптимізована завдяки ефективній генерації геометрії та раціональному використанню ресурсів. Система мінімізує навантаження на пам’ять та забезпечує плавну роботу навіть при створенні складних ландшафті;

– взаємодія компонентів організована таким чином, що забезпечує чіткий потік даних та ефективну комунікацію між різними частинами системи. Це дозволяє створювати складні ландшафти з високою продуктивністю та забезпечує зручність підтримки та розширення системи в майбутньому.

У системі реалізовано послідовну передачу даних між модулями, починаючи з генерації шуму і закінчуючи візуалізацією сцени, як показано на рисунку 3.1.

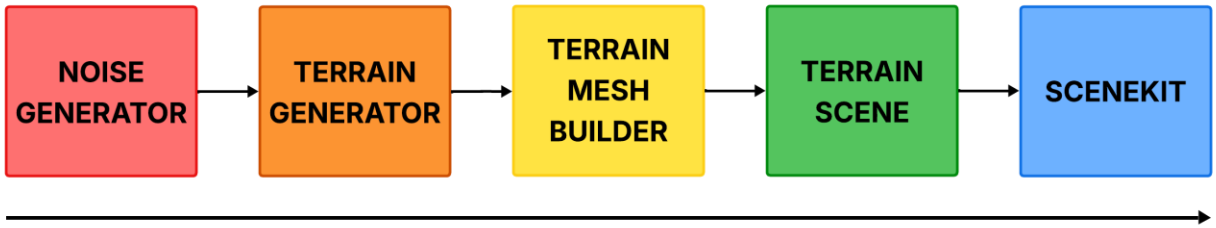

Рисунок 3.1 – Загальний потік даних у системі Landscape Generation

Детальний опис компонентів:

– TerrainScene є центральним компонентом системи, який відповідає за управління тривимірною сценою. Він ініціалізує та налаштовує камеру, керує освітленням та матеріалами, а також координує взаємодію між різними компонентами системи. Цей компонент забезпечує ефективну візуалізацію ландшафту та взаємодію з користувачем;

– TerrainGenerator відповідає за генерацію карти висот та визначення параметрів ландшафту. Він створює водну поверхню та керує процесом текстурування, забезпечуючи реалістичний вигляд ландшафту. Цей компонент є ключовим у процесі створення рельєфу, оскільки він об’єднує дані з різних джерел та формує кінцевий результат;

– TerrainMeshBuilder відповідає за створення геометрії на основі карти висот. Він розраховує нормалі, оптимізує полігональну сітку та формує дані для SceneKit. Цей компонент забезпечує ефективне представлення ландшафту в тривимірному просторі;

– NoiseGenerator реалізує алгоритм шуму Перліна та генерує багатооктавний шум для створення природного рельєфу. Він забезпечує параметризацію генерації, що дозволяє створювати різноманітні типи ландшафту;

– LandscapeViewController управляє користувацьким інтерфейсом, обробляє взаємодію користувача та координує процес генерації нового ландшафту. Цей компонент забезпечує зручний інтерфейс для користувача та ефективну взаємодію з системою.

Процес ініціалізації починається з LandscapeViewController, який є точкою входу в систему. Він створює та ініціалізує TerrainScene, який у свою чергу ініціалізує TerrainGenerator. TerrainGenerator створює TerrainMeshBuilder, який нарешті ініціалізує NoiseGenerator. Така послідовність забезпечує правильне налаштування всіх компонентів системи перед початком роботи, як показано на рисунку 3.2.

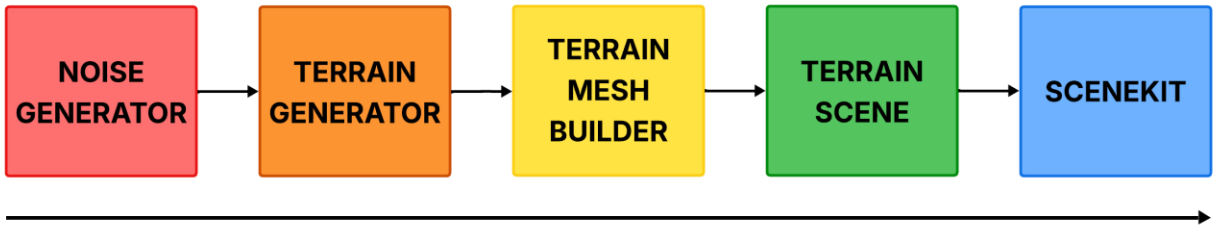

Рисунок 3.2 – Послідовність ініціалізації компонентів системи Landscape Generation

Коли користувач ініціює генерацію нового ландшафту, LandscapeViewController передає команду в TerrainScene. TerrainScene звертається до TerrainGenerator для створення карти висот. TerrainGenerator використовує NoiseGenerator для генерації базового шуму, а потім передає дані в TerrainMeshBuilder для створення геометрії. Після завершення генерації, TerrainMeshBuilder повертає готову геометрію в TerrainScene, який інтегрує її в тривимірну сцену за допомогою SceneKit.

Процедура створення ландшафту активується взаємодією користувача з інтерфейсом і виконується у вигляді серії послідовних викликів компонентів, як показано на рисунку 3.3.

Дані проходять через систему за наступною схемою:

Крок 1. NoiseGenerator створює базовий шум, який використовується для формування рельєфу.

Крок 2. Цей шум перетворюється в карту висот, яка представляє собою двовимірний масив значень висот.

Крок 3. На основі карти висот створюється тривимірна геометрія, яка включає вершини, нормалі та індекси.

Крок 4. Готова геометрія інтегрується в тривимірну сцену для візуалізації.

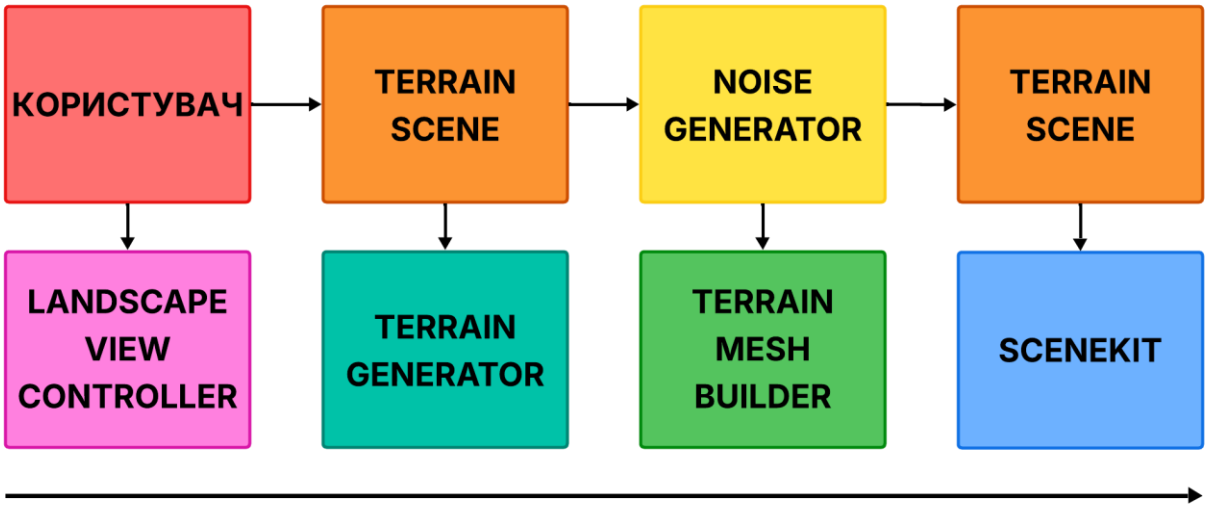

Рисунок 3.3 – Послідовність викликів під час генерації нового ландшафту

Структура обробки даних в архітектурі системи організована у вигляді поетапного перетворення шуму у тривимірну геометрію, як показано на рисунку 3.4.

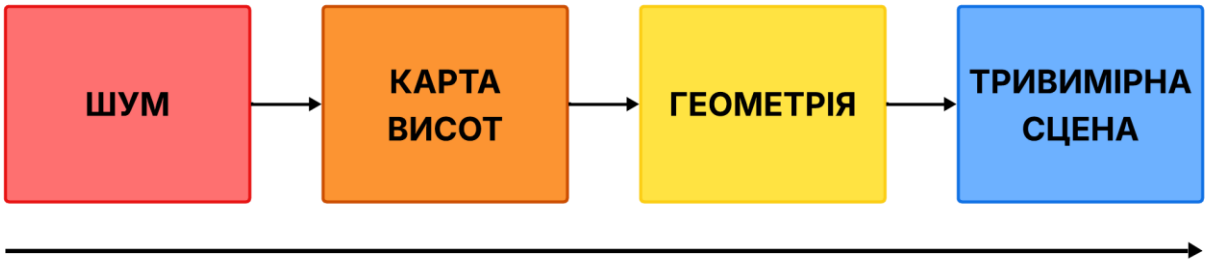

Рисунок 3.4 – Потік даних у процесі генерації тривимірного ландшафту

Така архітектура забезпечує:

– чітке розділення відповідальності;

– ефективну взаємодію компонентів;

– можливість подальшого розвитку;

– оптимальне використання ресурсів.

Кожен компонент має чітко визначені вхідні та вихідні дані, що забезпечує надійність та передбачуваність роботи системи. Така структура дозволяє легко модифікувати окремі компоненти без впливу на інші частини системи, що забезпечує гнучкість та масштабованість архітектури.

3.1.2 Основні модулі

Система Landscape Generation складається з п’яти основних модулів, кожен з яких відповідає за окрему функціональну частину. Така структура забезпечує чітке розмежування відповідальності та дозволяє ефективно реалізувати всі етапи генерації тривимірного ландшафту. Взаємодія між модулями організована через чітко визначені інтерфейси, що гарантує стабільність і передбачуваність роботи всієї системи.

Кожен із цих компонентів працює автономно, але водночас тісно взаємодіє з іншими модулями через чітко визначені інтерфейси. Така архітектура дозволяє досягти високої стабільності системи, забезпечити передбачувану поведінку при виконанні типових сценаріїв, а також створює можливості для подальшого розширення функціональності без суттєвого перепроєктування системи.

Перший модуль TerrainScene – є центральним модулем системи, що відповідає за управління тривимірною сценою та візуалізацію ландшафту. Цей модуль інтегрується з фреймворком SceneKit, забезпечуючи ефективне представлення тривимірного простору.

Функціональність:

– управління тривимірною сценою SceneKit:

1) модуль створює та керує тривимірною сценою;

2) налаштовує параметри відображення та забезпечує плавну анімацію;

– налаштування камери та освітлення:

1) реалізує різні режими камери;

2) налаштовує освітлення для реалістичного відображення рельєфу;

– координація процесу генерації ландшафту: керує послідовністю виконання операцій генерації та обробки даних;

– інтеграція згенерованої геометрії: забезпечує ефективне додавання нової геометрії до сцени.

Ключові компоненти модуля TerrainScene показано у таблиці 3.1.

Таблиця 3.1 – Ключові компоненти модуля TerrainScene

Назва компонента	**Призначення**
1	**2**
SCNScene	Основна сцена для відображення ландшафту
SCNCamera	Налаштування камери та параметрів відображення
SCNLight	Управління освітленням сцени
SCNNode	Контейнери для геометрії та матеріалів

У класі TerrainScene реалізовано управління тривимірною сценою, включаючи налаштування камери та генерацію нових елементів ландшафту за допомогою компонента TerrainGenerator (рис. А.1).

Другий модуль TerrainGenerator – відповідає за створення та обробку карти висот, яка є основою для генерації ландшафту. Цей модуль об’єднує дані з різних джерел та формує кінцевий результат.

Функціональність:

– генерація карти висот: створення двовимірного масиву висот на основі шуму та параметрів;

– створення водної поверхні: визначення рівня води та формування водних об’єктів;

– текстурування ландшафту: налаштування матеріалів та текстур для різних типів поверхні;

– управління параметрами генерації: контроль над процесом створення ландшафту.

Ключові компоненти TerrainGenerator показано у таблиці 3.2.

Таблиця 3.2 – Ключові компоненти модуля TerrainGenerator

Назва компонента	**Призначення**
1	**2**
HeightMap	Структура для зберігання карти висот
WaterLevel	Управління водною поверхнею
TerrainParameters	Налаштування параметрів генерації
TextureManager	Керування текстурами та матеріалами

У класі TerrainGenerator реалізовано генерацію ландшафту та водної поверхні на основі карти висот, яка створюється за допомогою генератора шуму та перетворюється на тривимірну геометрію через компонент TerrainMeshBuilder (рис. А.2).

Третій модуль TerrainMeshBuilder – відповідає за створення тривимірної геометрії на основі карти висот. Цей модуль забезпечує ефективне представлення рельєфу у вигляді полігональної сітки.

Функціональність:

– створення геометрії на основі карти висот: перетворення двовимірних даних у тривимірну модель;

– розрахунок нормалей: обчислення векторів нормалей для реалістичного освітлення;

– оптимізація полігональної сітки: зменшення кількості полігонів при збереженні якості;

– формування даних для SceneKit: підготовка геометрії для відображення.

Ключові компоненти модуля TerrainMeshBuilder показано у таблиці 3.3.

Таблиця 3.3 – Ключові компоненти модуля TerrainMeshBuilder

Назва компонента	**Призначення**
1	**2**
VertexBuffer	Буфер для зберігання вершин
IndexBuffer	Буфер для зберігання індексів
NormalCalculator	Обчислення нормалей
GeometryOptimizer	Оптимізація геометрії

У класі TerrainMeshBuilder реалізовано побудову тривимірної геометрії на основі карти висот. Метод createGeometry створює сітку з вершин, індексів і нормалей, що необхідні для коректного відображення освітлення (рис. А.3).

Четвертий модуль NoiseGenerator – реалізує алгоритми генерації шуму для створення природного рельєфу. Цей модуль забезпечує базові дані для формування ландшафту.

Функціональність:

– реалізація алгоритму шуму Перліна: створення плавного шуму для природного рельєфу;

– генерація багатооктавного шуму: комбінація різних частот шуму для деталізації;

– параметризація генерації шуму: налаштування параметрів для різних типів рельєфу;

– оптимізація генерації: ефективне обчислення значень шуму.

Ключові компоненти модуля NoiseGenerator показано у таблиці 3.4.

Таблиця 3.4 – Ключові компоненти модуля NoiseGenerator

Назва компонента	**Призначення**
1	**2**
PerlinNoise	Реалізація алгортиму шуму Перліна
OctaveNoise	Генерація багатооктавного шуму
NoiseParameters	Параметри генерації шуму
NoiseCache	Кешування обчислених значень

У класі NoiseGenerator реалізовано алгоритм генерації згладженого шуму Перліна, який використовується для створення природних варіацій висот у ландшафті. Основна функція повертає значення шуму для заданих координат (рис. А.4).

П’ятий модуль LandscapeViewController – керує користувацьким інтерфейсом та взаємодією з користувачем. Цей модуль забезпечує зручний доступ до функціональності системи.

Функціональність:

– управління користувацьким інтерфейсом: створення та оновлення UI елементів;

– обробка взаємодії користувача: реагування на жести та натискання;

– інтеграція SceneKit з UIKit: поєднання тривимірної сцени з інтерфейсом;

– координація процесу генерації: керування послідовністю операцій.

Ключові компоненти модуля LandscapeViewController показано у таблиці 3.5.

Таблиця 3.5 – Ключові компоненти модуля LandscapeViewController

Назва компонента	**Призначення**
1	**2**
UIViewController	Базовий клас для управління інтерфейсом

Продовження таблиці 3.5

1	2
GestureRecognizer	Обробка жестів користувача
SceneView	Відображення тривимірної сцени
ControlPanel	Панель керування параметрами

У класі LandscapeViewController реалізовано основну логіку взаємодії користувача з інтерфейсом, а також відображення тривимірної сцени та запуск генерації ландшафту. Компонент працює з візуалізацією у SceneKit через SCNView і керує ініціалізацією сцени (рис. А.5).

Взаємодія між модулями організована таким чином, що забезпечує чіткий потік даних та ефективну комунікацію між різними частинами системи. Кожен модуль має чітко визначені вхідні та вихідні дані, що забезпечує надійність та передбачуваність роботи системи.

Загальний процес генерації ландшафту починається з взаємодії користувача через контролер інтерфейсу та проходить усі етапи створення та відображення тривимірної сцени, як показано на рисунку 3.5.

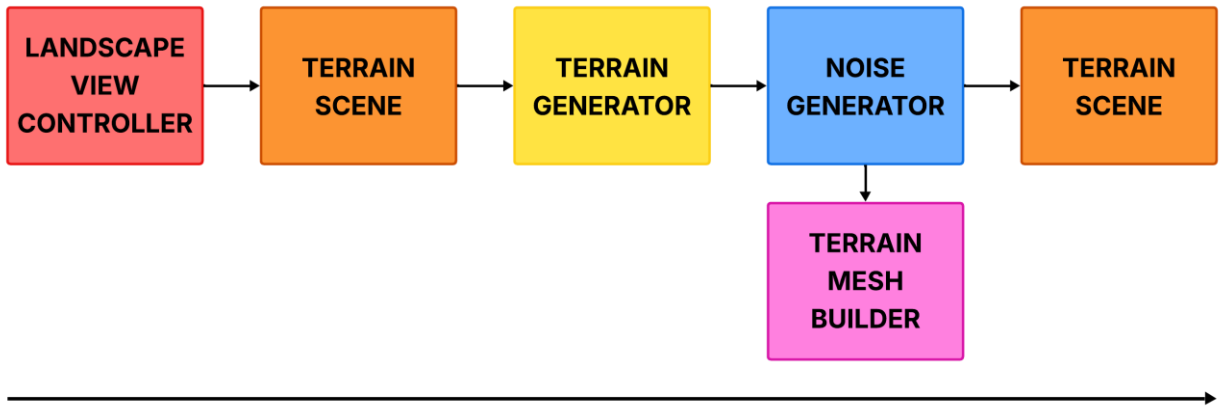

Рисунок 3.5 – Потік генерації ландшафту від LandscapeViewController до TerrainScene

Процес формування візуального зображення ландшафту включає проходження декількох етапів – від генерації шуму до відображення в інтерфейсі користувача, як показано на рисунку 3.6.

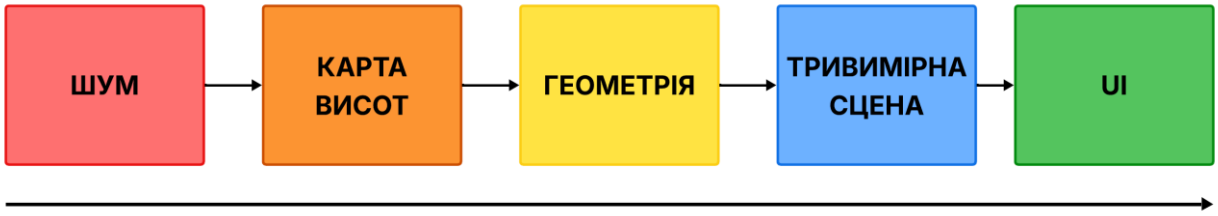

Рисунок 3.6 – Етапи формування тривимірного ландшафту від шуму до інтерфейсу користувача

Переваги модульної структури:

– незалежність модулів дозволяє розробляти та тестувати кожен компонент окремо;

– масштабованість системи забезпечує можливість легкого додавання нових функцій;

– чіткі межі відповідальності спрощують підтримку та налагодження;

– гнучкість архітектури дозволяє адаптувати систему до нових вимог.

Така модульна структура забезпечує ефективну роботу системи та спрощує її подальший розвиток, дозволяючи створювати складні ландшафти з високою продуктивністю та забезпечує зручність підтримки та розширення системи в майбутньому [30].

3.2 Генерація ландшафту

3.2.1 Реалізація алгоритмів шуму

Реалізація алгоритмів шуму у системі генерації ландшафтів Landscape Generation ґрунтується на використанні спеціалізованого класу NoiseGenerator, який відіграє ключову роль у створенні процедурного шуму,

необхідного для формування реалістичного природного рельєфу. Цей компонент системи відповідає за обчислення значень шуму, які далі використовуються як основа для побудови карти висот. Саме на основі цих значень визначається висота кожної точки майбутнього ландшафту, що дозволяє досягти ефекту природної нерівномірності та різноманітності поверхні.

У якості основного методу генерації шуму обрано алгоритм Перліна. Це рішення обґрунтоване його здатністю генерувати згладжені, неперіодичні коливання, що імітують природні форми рельєфу – такі як пагорби, долини, нерівності тощо. Шум Перліна широко використовується у комп'ютерній графіці, геопросторовому моделюванні, а також в ігровій індустрії завдяки своїй ефективності, простоті налаштування та можливості масштабування.

Однією з головних переваг цього алгоритму є плавність переходів між значеннями, що дозволяє уникати різких змін висоти та створити більш реалістичне середовище. Крім того, шум Перліна має детермінований характер, тобто при однакових параметрах він завжди генерує однакову структуру, що забезпечує відтворюваність результатів і є критично важливим у контексті генерації ландшафтів. Такий підхід забезпечує не лише візуальну достовірність рельєфу, але й стабільність та контрольованість усієї системи генерації.

Базовий клас NoiseGenerator містить основні компоненти для генерації шуму. Ключовим елементом є масив перестановок, який визначає характер шуму. Цей масив ініціалізується випадковими значеннями, але зберігає свою структуру протягом усього життєвого циклу генератора, що забезпечує детермінованість генерації (рис. А.6).

Загальний вигляд згенерованого тривимірного рельєфу, сформованого на основі шуму Перліна, показано на рисунку 3.7.

Основна функція генерації шуму реалізує алгоритм шуму Перліна, який комбінує градієнти в кожній точці для створення плавних переходів.

Алгоритм використовує інтерполяцію між градієнтами для створення безперервного шуму (рис. А.7).

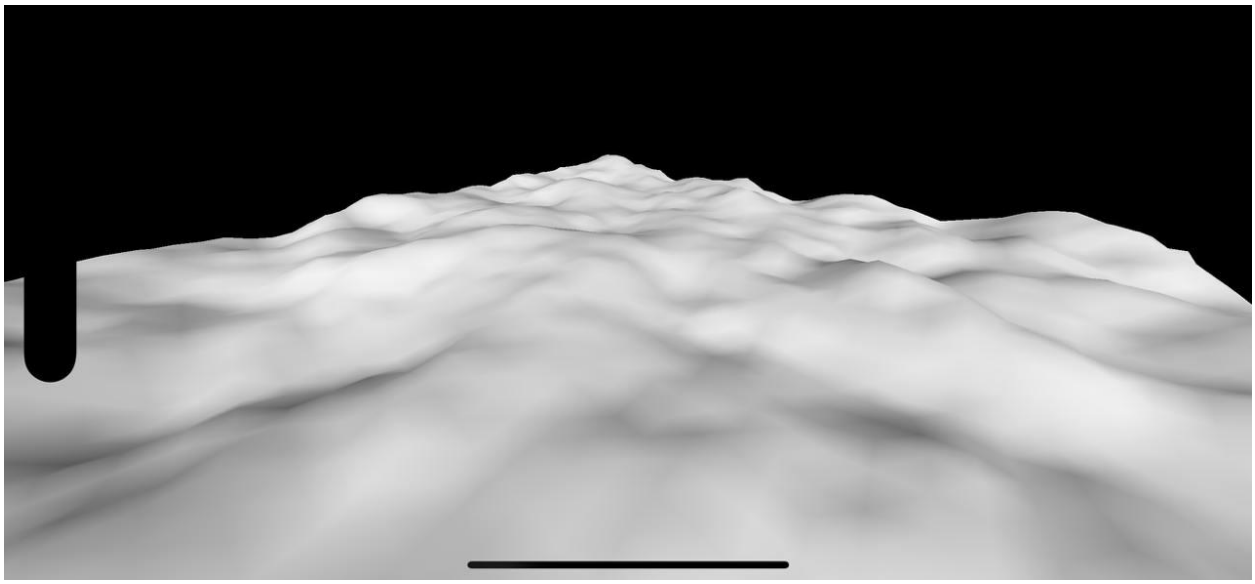

Рисунок 3.7 – Тривимірна карта висот, згенерована на основі шуму Перліна

Допоміжні функції для обчислення шуму включають функцію затухання, яка забезпечує плавні переходи, та функцію градієнта, яка визначає напрямок зміни шуму в кожній точці (рис. А.8).

Для створення більш природного та деталізованого рельєфу використовується багатооктавний шум. Цей підхід комбінує кілька октав шуму Перліна з різними частотами та амплітудами. Структура параметрів для налаштування генерації включає всі необхідні параметри для контролю над процесом створення рельєфу (рис. А.9).

Функція генерації багатооктавного шуму комбінує кілька октав шуму, кожна з яких має свою частоту та амплітуду. Це створює більш природний та деталізований рельєф, ніж простий шум Перліна. Обчислення висоти для конкретної точки враховує всі октави шуму, що дозволяє створити рельєф з різноманітними масштабами деталей, від загальних форм до дрібних деталей (рис. А.10).

Для ілюстрації тривимірної структури такого рельєфу, створеного багатооктавним шумом, його зображення з іншого ракурсу показано на рисунку 3.8.

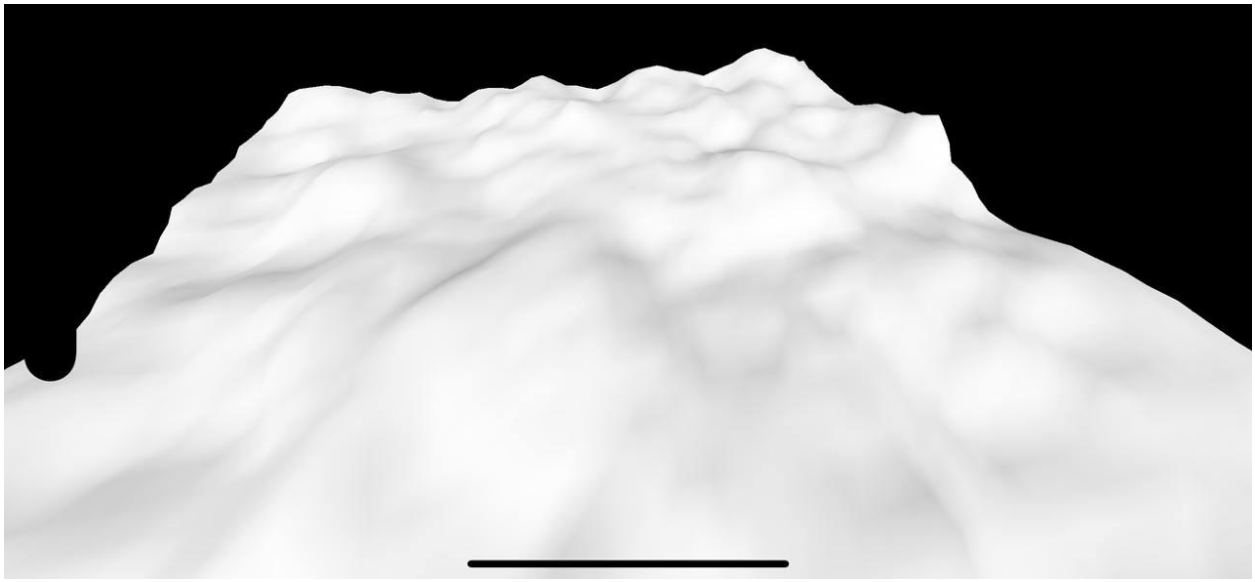

Рисунок 3.8 – Тривимірний ландшафт, зображений з іншого ракурсу для демонстрації просторової глибини і варіативності рельєфу

Обчислення висоти для конкретної точки виконується з урахуванням внеску кожної октави шуму з відповідною амплітудою та частотою (рис. А.11).

Вплив основних параметрів на генерацію ландшафту:

– baseFrequency (базова частота):

1) високі значення – створюють дрібні деталі рельєфу, що добре підходить для гірської місцевості;

2) низькі значення – формують плавні форми рельєфу, ідеальні для рівнин;

– baseAmplitude (базова амплітуда):

1) високі значення – створюють різкі перепади висот, характерні для гірських масивів;

2) низькі значення – формують м’які переходи, властиві рівнинній місцевості;

– octaves (кількість октав):

1) більше октав – додає більше деталей до рельєфу, створюючи більш реалістичний вигляд;

2) менше октав – створює простіший рельєф з меншою кількістю деталей;

– persistence (персистентність):

1) високі значення – зберігають більше деталей у високих частотах створюючи більш деталізований рельєф;

2) низькі значення – швидше зникають деталі, створюючи більш плавний рельєф;

– lacunarity (лакунарність):

1) високі значення – створюють більше варіацій у високих частотах, додаючи різноманітність до рельєфу;

2) низькі значення – формують більш рівномірний рельєф з меншою кількістю варіацій.

Приклад налаштування параметрів для генерації різних типів ландшафту, таких як гори та пагорби (рис. А.12).

Така реалізація дозволяє створювати різноманітні типи ландшафту шляхом налаштування параметрів генерації, забезпечуючи гнучкість та контроль над процесом створення рельєфу.

3.2.2 Формування карти висот

Після завершення етапу генерації шуму наступним кроком є формування карти висот – процес, що трансформує згенеровані шумові дані у структуровану основу для побудови рельєфу.

Формування карти висот є ключовим етапом у створенні ландшафту, який визначає основні характеристики рельєфу. Цей процес включає генерацію початкових висот, їх нормалізацію, масштабування та визначення

рівня води для створення реалістичного ландшафту. Якість та реалістичність кінцевого результату значною мірою залежать від правильного виконання цього етапу. Процес формування карти висот є основою для створення реалістичного ландшафту, який буде візуально привабливим та природним.

Саме карта висот формує базу, на якій збудовано подальшу геометрію ландшафту, що має бути як візуально привабливою, так і природною.

Щоб забезпечити високий рівень деталізації та достовірності, необхідно враховувати особливості розподілу висот у різних регіонах ландшафту, баланс між рівнинними та гористими ділянками, а також плавність переходів між ними. Сучасні алгоритми генерації дозволяють досягати вражаючої глибини та різноманіття рельєфу, що особливо важливо для візуалізації великих територій. Важливим аспектом також є дотримання масштабних співвідношень, що дає змогу створити топографічно коректну модель.

В основі побудови карти лежить використання шумової функції, яка задає базові висотні значення у кожній точці сітки.

Процес генерації висот починається з використання шуму Перліна для створення початкової карти висот. Це забезпечує природний вигляд рельєфу з плавними переходами та реалістичними формами. Шум Перліна використовується як основа, оскільки він забезпечує плавні переходи між різними висотами та створює природний вигляд рельєфу. Генерація висот відбувається для кожної точки сітки, створюючи двовимірний масив значень, який представляє висоту в кожній точці ландшафту.

Процес генерації висот включає кілька етапів:

– створення початкової карти висот з використанням шуму Перліна;

– застосування багатооктавного шуму для створення більш деталізованого рельєфу;

– корекція висот для створення більш природного рельєфу;

– застосування додаткових ефектів для покращення якості рельєфу.

Генерація карти висот здійснюється класом TerrainGenerator, який використовує задані параметри та шумову функцію для обчислення значень висоти в кожній точці сітки (рис. А.13).

Щоб уникнути надмірних перепадів і забезпечити відповідність висот заданому масштабу, виконується нормалізація.

Нормалізація висот є важливим етапом, який забезпечує приведення всіх значень до єдиного діапазону. Це дозволяє уникнути екстремальних значень та забезпечити рівномірний розподіл висот по всьому ландшафту. Процес нормалізації включає знаходження мінімального та максимального значень висот та приведення всіх значень до діапазону 0 та 1.

Обробка карти висот включає нормалізацію значень до стандартного діапазону та масштабування для відповідності заданим межам рельєфу (рис. А.14).

Наступним важливим кроком є визначення рівня води – це впливає не лише на візуальне сприйняття, а й на структуру всієї сцени.

Визначення рівня води є критичним етапом для створення реалістичного ландшафту. Цей процес враховує відсоток водної поверхні та забезпечує природний перехід між сушею та водою. Рівень води визначається на основі заданого відсотка водної поверхні, що дозволяє контролювати кількість водних об’єктів у ландшафті. Після визначення рівня води відбувається розділення ландшафту на сушу та воду, що дозволяє окремо обробляти ці частини ландшафту.

На практиці результат поділу території на водні та сухопутні області з урахуванням рівня води можна побачити на рисунку 3.9.

Для більш повного уявлення про просторову конфігурацію водної поверхні, подано той самий ландшафт з іншого ракурсу, який показано на рисунку 3.10.

Процес визначення рівня води включає:

- розрахунок оптимального рівня води на основі заданого відсотка;
- розділення ландшафту на сушу та воду;

– корекція рівня води для створення природних берегових ліній;

– застосування додаткових ефектів для покращення вигляду водної поверхні.

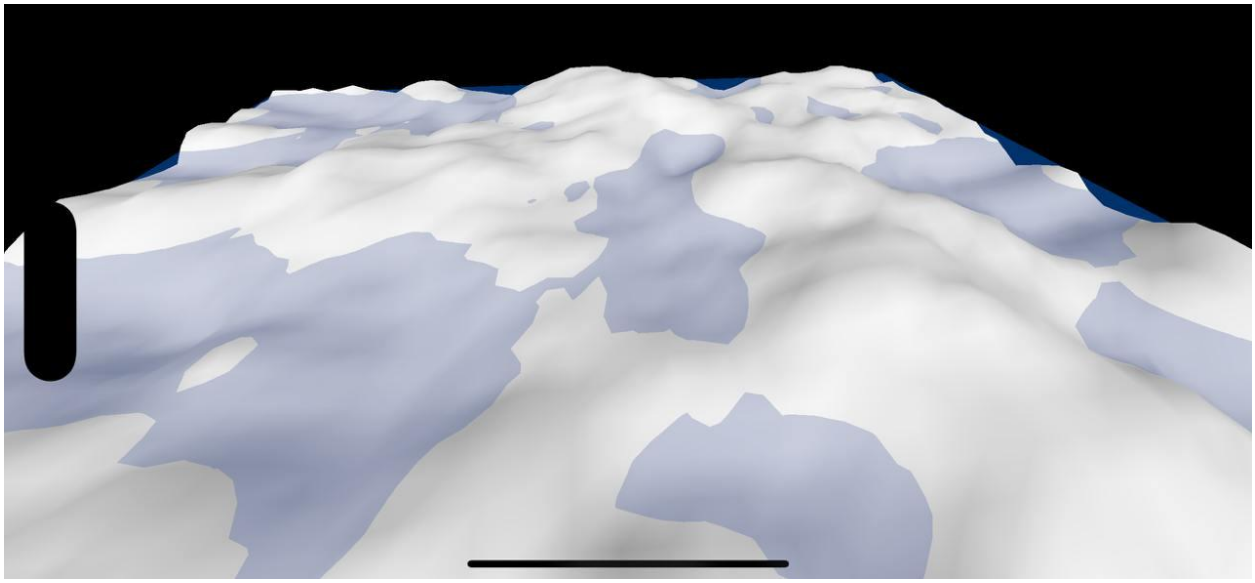

Рисунок 3.9 – Візуалізація водної поверхні після обробки рівня води у сцені

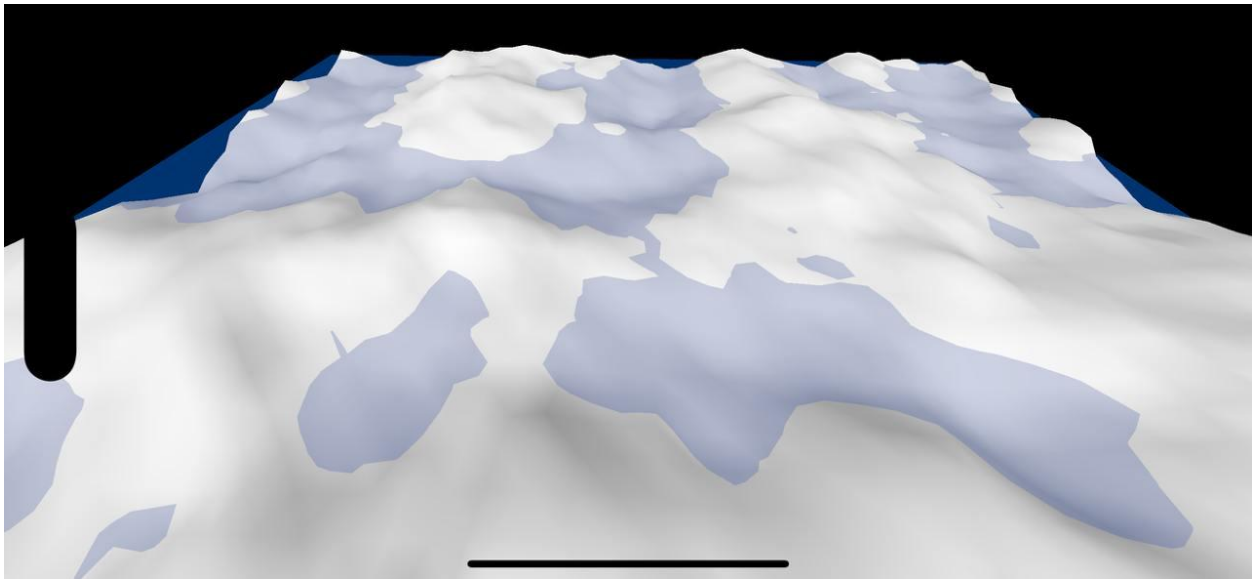

Рисунок 3.10 – Альтернативний вигляд водної поверхні після встановлення рівня води

Визначення рівня води та розділення ландшафту на водні й сухопутні області виконується за допомогою класу WaterLevelProcessor (рис. А.15).

Після виконання всіх окремих операцій вони об’єднуються у єдиний процес, що забезпечує завершене формування карти висот.

Інтеграція всіх процесів формування карти висот забезпечує послідовну обробку даних та створення кінцевого результату. Цей процес включає всі попередні етапи та додаткову обробку для покращення якості ландшафту. Кожен етап обробки впливає на кінцевий результат, тому важливо правильно налаштувати параметри кожного етапу.

Інтеграція всіх етапів обробки висотної інформації виконується у класі TerrainGenerator, який формує фінальну карту висот з урахуванням нормалізації, масштабування та рівня води (рис. А.16).

Подальше використання згенерованих даних потребує їх збереження у зручній структурі, що дозволяє швидко отримувати доступ до характеристик ландшафту.

Для ефективного зберігання та подальшого використання оброблених даних створюється спеціальна структура, яка містить всі необхідні характеристики ландшафту. Ця структура включає інформацію про висоти, рівень води, розділення на сушу та воду, а також додаткові характеристики, такі як середня висота, максимальна та мінімальна висоти.

Результати генерації та обробки зберігаються у структурі ProcessedHeightMap, яка містить не лише основні дані, але й додаткові статистичні характеристики рельєфу (рис. А.17).

Щоб зберегти контроль над параметрами генерації карти висот, що визначають її якість і вигляд, використовується окрема конфігураційна структура.

Для гнучкого налаштування процесу формування карти висот використовується структура параметрів, яка дозволяє контролювати всі аспекти генерації ландшафту.

Ключові параметри містяться у HeightMapParameters (рис. А.18).

Усі ці етапи взаємопов’язані та забезпечують побудову якісного ландшафту, готового до подальшої обробки.

Процес формування карти висот є основою для створення реалістичного ландшафту.

Він забезпечує:

– генерацію природного рельєфу з використанням шуму Перліна;

– нормалізацію та масштабування висот для створення рівномірного розподілу;

– визначення оптимального рівня води для створення реалістичних водних об’єктів;

– розділення ландшафту на сушу та воду для подальшої обробки;

– збереження оброблених даних для використання в різних частинах системи.

Ця реалізація дозволяє створювати різноманітні типи ландшафту, від рівнин до гірських масивів, з реалістичними водними об’єктами та природними переходами між різними типами рельєфу.

3.2.3 Побудова полігональної сітки

Побудова полігональної сітки є одним із ключових етапів у процесі створення тривимірної моделі ландшафту в системі генерації. На цьому етапі відбувається трансформація двовимірної карти висот, яка містить цифрові значення висоти кожної точки сітки, у повноцінну тривимірну полігональну структуру. Це дозволяє представити ландшафт у вигляді сітки з багатокутників (зазвичай трикутників або чотирикутників), яка потім може бути передана графічному рушію для візуалізації.

Процес побудови сітки виконується автоматично на основі алгоритмів, які обчислюють положення вершин у тривимірному просторі, визначають порядок їх з’єднання в полігони та обчислюють нормалі для коректного освітлення. Саме на цьому етапі визначається, наскільки реалістично буде виглядати рельєф після його виведення на екран. Висока щільність сітки

дозволяє досягти більш точного відображення нерівностей, але також потребує більше ресурсів для обробки.

Якість та ефективність побудови полігональної сітки безпосередньо впливають на загальну продуктивність системи. Якщо сітка створена занадто деталізовано, це може призвести до зниження частоти кадрів і перевантаження графічного процесора. З іншого боку, надто груба сітка може спростити рельєф, погіршуючи візуальне враження. Тому важливо дотримуватися балансу між точністю візуалізації та швидкістю обробки, що досягається шляхом налаштування параметрів генерації та застосування оптимізованих алгоритмів побудови геометрії.

Процес створення полігональної сітки починається з формування вершин на основі карти висот. Кожна вершина представляє точку в тривимірному просторі, яка визначається своїми координатами (x, y, z). Координата y відповідає за висоту точки, яка береться з карти висот, а координати x та z визначають положення точки в горизонтальній площині. Для створення трикутників, які формують поверхню ландшафту, використовуються індекси. Індекси визначають, які вершини пов’язані між собою для утворення трикутників. Це дозволяє ефективно використовувати пам’ять, оскільки кожна вершина використовується в кількох трикутниках.

Цей процес виконується за допомогою функції createVerticesAndIndices, яка генерує вершини та індекси з використанням висотної карти як вхідних даних (рис. А.19).

Розрахунок нормалей є важливим етапом для створення реалістичного освітлення ландшафту. Нормаль – це вектор, перпендикулярний до поверхні в даній точці. Він визначає, як світло взаємодіє з поверхнею, що впливає на її видимий вигляд. Для кожного трикутника обчислюється нормаль на основі його вершин. Ця нормаль використовується для розрахунку освітлення в кожній точці поверхні. Правильний розрахунок нормалей забезпечує плавні переходи освітлення та створює реалістичний вигляд ландшафту.

Функція calculateNormals відповідає за виконання цих обчислень на основі сітки, що сформована раніше (рис. А.20).

Оптимізація геометрії є критичним етапом для забезпечення ефективної роботи застосунку. Вона включає кілька аспектів:

– згладжування різких переходів:

1) застосування алгоритмів згладжування для створення більш природного рельєфу;

2) зменшення різких перепадів висот;

3) створення плавних переходів між різними типами рельєфу;

– зменшення кількості полігонів:

1) видалення надлишкових вершин та трикутників;

2) збереження важливих деталей рельєфу;

3) оптимізація співвідношення якості та продуктивності;

– оптимізація кешу вершин:

1) ефективне використання кешу GPU;

2) зменшення кількості переривань при рендерингу;

3) покращення загальної продуктивності;

– рівні деталізації:

1) створення різних рівнів деталізації для різних відстаней;

2) динамічна зміна деталізації залежно від відстані до камери;

3) оптимізація рендерингу для віддалених частин ландшафту.

Усі перелічені підходи поєднані в класі GeometryOptimizer, який реалізує алгоритми покроково згідно з заданими параметрами (рис. А.21).

Процес побудови полігональної сітки забезпечує створення високоякісної тривимірної моделі ландшафту з оптимальною продуктивністю.

Ключові аспекти цього процесу включають:

– ефективне використання пам’яті через індексовану геометрію;

– реалістичне освітлення завдяки точним нормалям;

– оптимізацію продуктивності через різні методи оптимізації;

– адаптивну деталізацію для різних відстаней;

– збереження важливих деталей рельєфу при оптимізації.

Ця реалізація дає змогу створювати реалістичні ландшафти, що продуктивно рендеряться на мобільних пристроях, гарантуючи відмінний баланс між візуальною якістю та продуктивністю.

3.3 Візуалізація та інтеграція в SceneKit

3.3.1 Налаштування сцени

Налаштування сцени в SceneKit є фундаментальним етапом для створення реалістичного тривимірного ландшафту. Цей процес включає три основні компоненти: ініціалізацію сцени, налаштування камери та освітлення, а також конфігурацію матеріалів. Кожен з цих елементів відіграє вирішальну роль у створенні якісного візуального результату.

Ініціалізація сцени в TerrainScene починається зі створення базової структури, яка охоплює головні компоненти тривимірного середовища. Клас TerrainScene відповідає за управління всіма тривимірними елементами та їхню взаємодію. Чітка ієрархія вузлів дозволяє ефективно керувати трансформаціями та забезпечує узгоджену поведінку усіх візуальних компонентів.

Цей підхід реалізовано через клас TerrainScene, що об’єднує основні елементи сцени – камеру, освітлення та сам ландшафтний вузол (рис. А.22).

При ініціалізації сцени важливо правильно структурувати ієрархію вузлів. Кореневий вузол містить всі інші елементи сцени, що дозволяє ефективно керувати трансформаціями та взаємодією між компонентами. Кожен вузол має своє призначення: cameraNode відповідає за точку огляду, lightNode керує освітленням, а terrainNode містить геометрію ландшафту.

Початкове налаштування сцени здійснюється за допомогою методу setupInitialState, який визначає фон, туман і параметри фізичного середовища (рис. А.23).

Налаштування початкового стану включає встановлення фону, туману та фізичного середовища. Фон створює атмосферу сцени, туман додає глибину та реалістичність, а фізичне середовище забезпечує правильну поведінку об’єктів у сцені.

Налаштування камери та освітлення є наступним важливим кроком у створенні повноцінної сцени. Камера є ключовим елементом для сприйняття тривимірного простору. Правильне її налаштування забезпечує оптимальний кут огляду ландшафту та коректну перспективу. При цьому важливо враховувати такі параметри, як поле зору, відстань видимості та початкову позицію камери.

Усі ці характеристики задаються в методі setupCamera, що визначає положення, напрямок і параметри огляду сцени (рис. А.24).

Система освітлення відіграє вирішальну роль у створенні реалістичного зображення. Вона включає основне джерело світла (сонце) та додаткове освітлення для заповнення тіней. Правильне налаштування освітлення забезпечує правильне відображення форм та текстур ландшафту.

Для цього використовується метод setupLighting, який встановлює спрямоване освітлення з урахуванням інтенсивності, тіней та розташування джерела світла (рис. А.25).

Конфігурація матеріалів є завершальним етапом налаштування сцени. Матеріали визначають візуальні властивості поверхні ландшафту, такі як колір, блиск, прозорість та текстура. Кожен тип поверхні потребує специфічних налаштувань для досягнення реалістичного вигляду. Правильне застосування текстур, мап нормалей і додаткових ефектів дозволяє створити деталізовану та візуально привабливу модель ландшафту.

Цей етап реалізовано в методі setupMaterials, який відповідає за кольорові параметри, шорсткість, металевість і нормал-мапінг об’єктів сцени (рис. А.26).

Налаштування текстур є критично важливим для створення реалістичної поверхні. Нормал-маппінг додає деталізацію без збільшення геометричної складності, а правильне налаштування повторення текстур забезпечує безшовне покриття поверхні.

Текстурування з нормал-мапінгом здійснюється методом setupNormalMapping, де задаються карта нормалей та параметри повторення (рис. А.27).

Додаткові ефекти матеріалів, такі як прозорість та відбиття, додають глибину та реалістичність зображенню. Важливо правильно налаштувати ці ефекти, щоб вони не впливали негативно на продуктивність.

Їх реалізацію забезпечує метод setupMaterialEffects, що встановлює параметри фізично коректного освітлення та візуальні ефекти (рис. А.28).

У результаті таких налаштувань формується деталізована сцена з виразним рельєфом і реалістичною взаємодією світла та матеріалів.

Візуалізацію сцени з водною поверхнею та текстурованим рельєфом показано на рисунку 3.11.

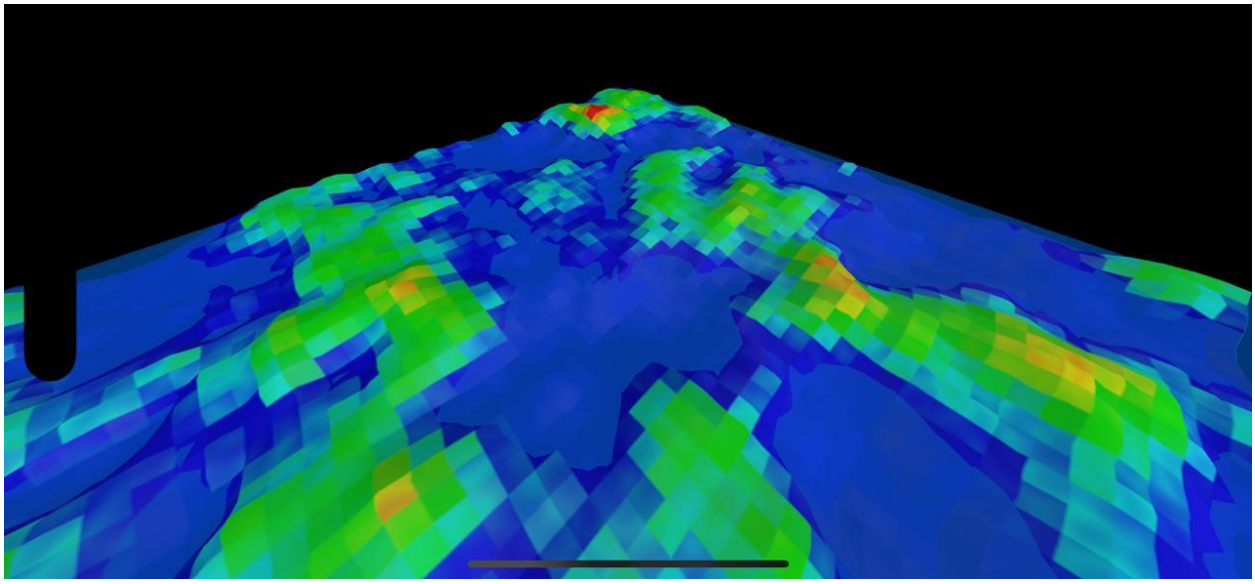

Рисунок 3.11 – Вигляд сцени з водною поверхнею та рельєфом

Зміна ракурсу дозволяє детальніше оцінити ефекти прозорості та освітлення, що показано на рисунку 3.12.

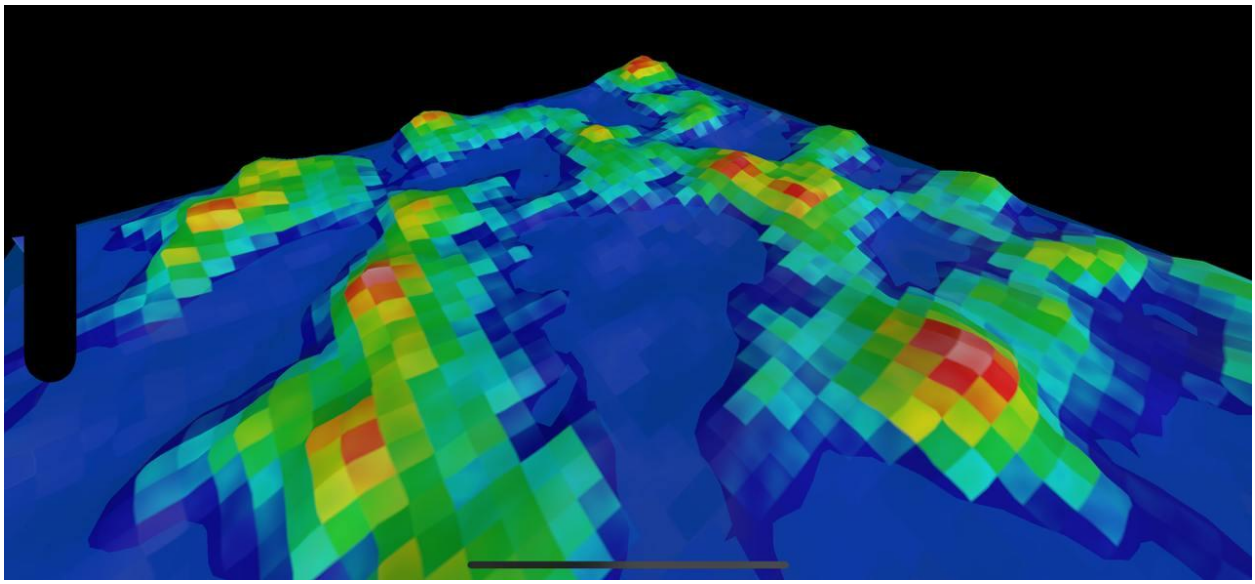

Рисунок 3.12 – Інший ракурс ландшафту з акцентом на освітлення і прозорість матеріалів

Ще один приклад із перспективною проєкцією показує глибину рельєфу та ефекти відбиття, які підсилюють реалістичність сцени показано на рисунку 3.13.

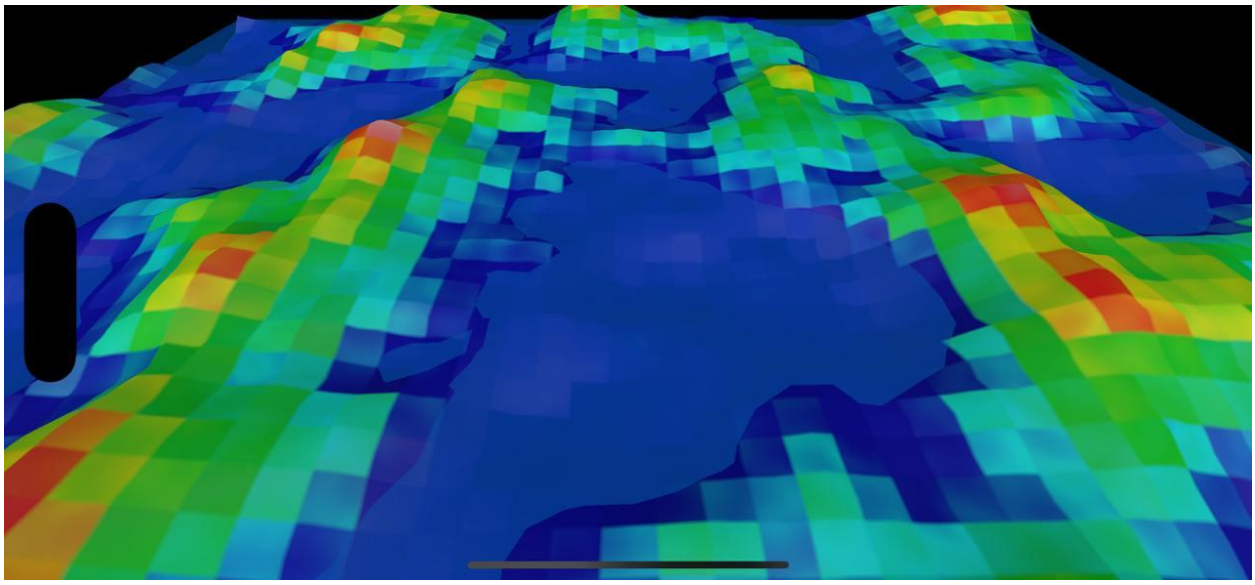

Рисунок 3.13 – Перспективне зображення з виразним рельєфом і ефектами відбиття

Реалізація налаштування сцени включає комплексний підхід до створення реалістичного тривимірного ландшафту. Кожен компонент системи реалізується з урахуванням специфічних вимог та взаємодії з іншими елементами.

Архітектура системи побудована на основі компонентного підходу, де кожен елемент сцени є незалежним модулем з чітко визначеними відповідальностями. TerrainScene виступає як центральний компонент, який координує роботу всіх підсистем. Така архітектура забезпечує гнучкість системи та можливість її розширення.

Реалізація освітлення базується на фізично-коректному підході, де кожен джерело світла моделюється відповідно до реальних фізичних законів. Основне джерело світла (сонце) реалізується як напрямлене світло з налаштуваннями тіней, що забезпечує реалістичне освітлення сцени. Додаткові джерела світла використовуються для заповнення тіней та підсвічування деталей.

Система матеріалів реалізується з використанням фізично-коректного рендерингу (PBR), що забезпечує реалістичне відображення поверхонь. Кожен матеріал включає набір параметрів, таких як шорсткість, металевість та відбиття, які визначають його візуальні властивості. Текстури застосовуються з урахуванням їх фізичних характеристик та взаємодії з освітленням.

Інтерактивність сцени реалізується через систему обробки подій, яка дозволяє користувачу взаємодіяти з ландшафтом. Камера налаштовується з урахуванням потреб користувача, забезпечуючи плавне переміщення та масштабування. Система також включає механізми обробки жестів для зручного керування.

Візуальна якість забезпечується через систему пост-обробки, яка включає такі ефекти як туман, глибина різкості та кольорова корекція. Ці ефекти додають глибину та атмосферу сцені, покращуючи загальне сприйняття ландшафту.

Система також включає механізми для обробки помилок та відновлення після збоїв. Це забезпечує стабільну роботу застосунку навіть при несприятливих умовах. Додатково реалізовано систему логування для відстеження проблем та аналізу продуктивності.

Інтеграція з іншими компонентами системи реалізується через чітко визначені інтерфейси, що дозволяє легко додавати нові функції та модифікувати існуючі. Система спроектована з урахуванням можливого розширення функціоналу та інтеграції з новими технологіями.

Ця реалізація забезпечує створення реалістичного та інтерактивного тривимірного ландшафту з високою продуктивністю та якістю візуалізації. Система є масштабованою та може бути легко розширена для підтримки нових функцій та технологій.

3.3.2 Інтеграція згенерованої геометрії

Інтеграція згенерованої геометрії в SceneKit є критично важливим етапом для візуалізації тривимірного ландшафту. Цей процес включає перетворення згенерованих даних у формат, який може бути використаний SceneKit, та правильне налаштування всіх необхідних компонентів для реалістичного відображення. Успішна інтеграція залежить від правильного форматування даних, ефективного використання ресурсів та точного налаштування візуальних параметрів.

Перетворення згенерованих даних у геометрію SceneKit вимагає ретельного підходу до форматування вершин, нормалей та текстурних координат. Цей процес включає створення буферів даних та їх правильне індексування. Важливо забезпечити оптимальну структуру даних для ефективного використання пам’яті та швидкого рендерингу.

Створення геометрії починається з аналізу карти висот та перетворення її у набір вершин. Кожна вершина повинна містити інформацію про свою

позицію в просторі, нормаль для правильного освітлення та текстурні координати для застосування текстур. Ці дані організуються в буфери, які оптимізовані для швидкого доступу та ефективного використання пам’яті.

Цю функціональність реалізує клас TerrainGeometryBuilder, що створює тривимірну геометрію на основі карти висот, включаючи вершини, нормалі, текстурні координати та індекси (рис. А.29).

У результаті застосування цього класу формується візуальне представлення рельєфу з використанням генерації геометрії за картою висот, як показано на рисунку 3.14.

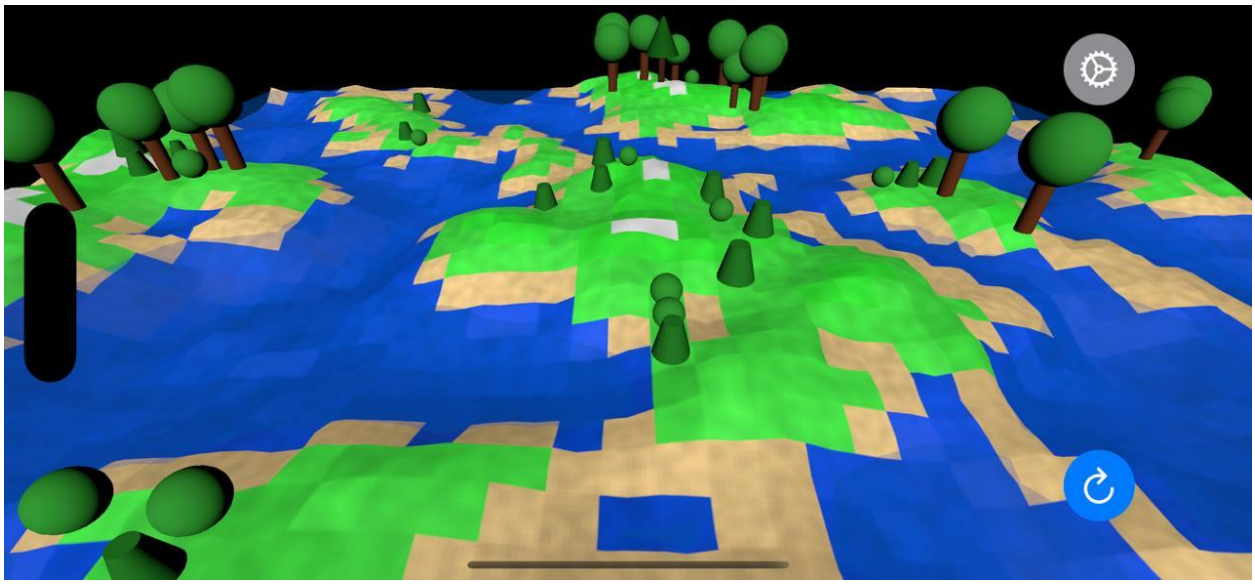

Рисунок 3.14 – Візуалізація тривимірного рельєфу з застосуванням карти висот і розміщенням об’єктів на поверхні

Розрахунок нормалей є критично важливим етапом для створення реалістичного освітлення поверхні. Нормалі визначають, як світло взаємодіє з поверхнею, і впливають на загальний вигляд ландшафту. Система аналізує сусідні вершини та розраховує середнє значення нормалі для кожної точки поверхні.

Текстурні координати генеруються з урахуванням розміру геометрії та необхідності повторення текстур. Правильне налаштування текстурних

координат забезпечує безшовне покриття поверхні текстурами та уникнення артефактів при відображенні.

Після створення геометрії необхідно правильно інтегрувати її в сцену. Це включає створення вузла сцени, налаштування його трансформацій та додавання до ієрархії сцени. Правильне позиціонування та масштабування геометрії є критично важливим для створення реалістичного ландшафту.

Це реалізовано через метод addTerrainGeometry класу TerrainScene, який відповідає за додавання геометрії до сцени та налаштування її властивостей (рис. А.30).

Приклад інтеграції геометрії у сцену з масштабуванням та позиціюванням об’єктів показано на рисунку 3.15.

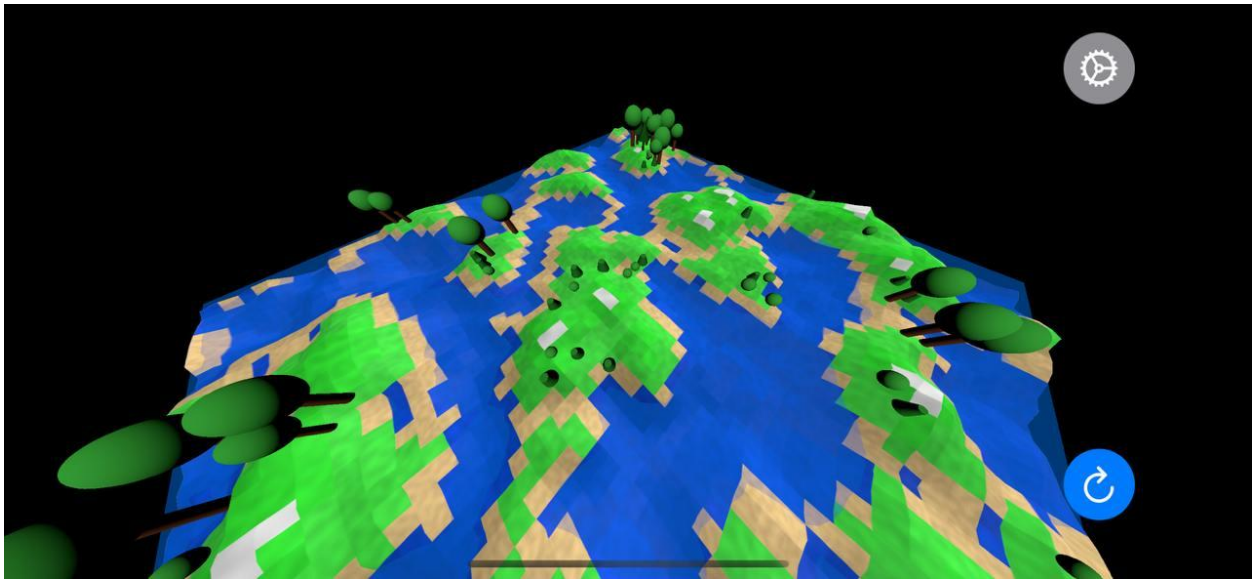

Рисунок 3.15 – Загальний вигляд сцени з інтегрованою геометрією, деревами та водними ділянками, згенерованими у межах сцени SceneKit

Налаштування фізики для геометрії включає визначення колізій та фізичних властивостей поверхні. Це дозволяє забезпечити правильну взаємодію з іншими об’єктами сцени та створює основу для реалістичної поведінки в ігровому середовищі.

Налаштування матеріалів та текстур є ключовим етапом для створення реалістичного вигляду ландшафту. Це включає створення матеріалів,

застосування текстур та налаштування їх параметрів. Правильне налаштування матеріалів забезпечує реалістичне відображення поверхні та взаємодію з освітленням.

Відповідальність за ці налаштування покладено на клас TerrainMaterialManager, який конфігурує базові параметри, текстури та візуальні ефекти (рис. А.31).

Візуальний приклад застосування матеріалів і текстур до різних ділянок поверхні наведено на рисунку 3.16.

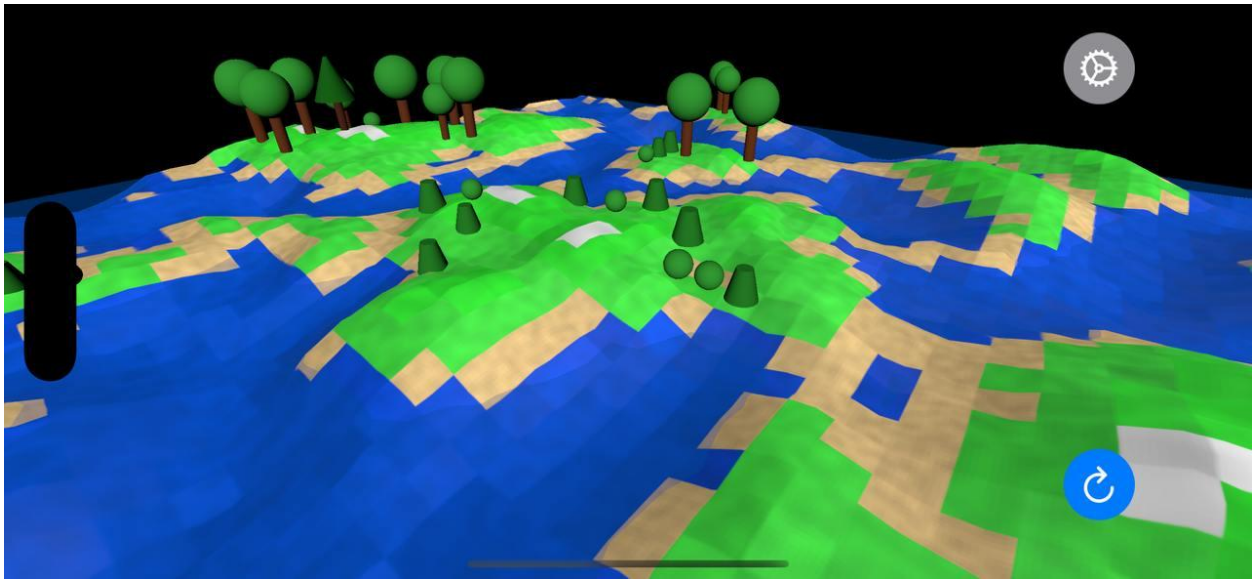

Рисунок 3.16 – Приклад застосування текстур до рельєфу місцевості з відображенням лісових масивів, води та відкритих ділянок

Застосування текстур включає вибір відповідних текстур для різних типів поверхні та їх правильне налаштування. Текстури повинні відповідати фізичним характеристикам поверхні та забезпечувати реалістичне відображення деталей. Система також включає налаштування додаткових ефектів, таких як відбиття та прозорість, для покращення візуальної якості.

Інтеграція з іншими компонентами системи забезпечується через чітко визначені інтерфейси. Це дозволяє легко додавати нові функції та модифікувати існуючі. Система спроектована з урахуванням можливого розширення функціоналу та інтеграції з новими технологіями.

Ця реалізація забезпечує створення реалістичного та інтерактивного тривимірного ландшафту з високою продуктивністю та якістю візуалізації. Система є масштабованою та може бути легко розширена для підтримки нових функцій та технологій.

3.3.3 Реалізація UI

Реалізація користувацького інтерфейсу є важливим аспектом розробки застосунку для генерації ландшафтів. UI повинен забезпечувати зручний доступ до всіх функцій генерації та налаштування ландшафту, а також надавати інтуїтивно зрозумілий спосіб взаємодії з тривимірною сценою. Розробка UI включає створення структури контролера, інтеграцію SceneKit з UIKit та реалізацію елементів керування.

LandscapeViewController є центральним компонентом, який відповідає за управління користувацьким інтерфейсом та взаємодію з тривимірною сценою. Він об’єднує функціональність UIKit та SceneKit, забезпечуючи плавну взаємодію між ними.

Цю функціональність реалізовано в класі LandscapeViewController, який виконує роль основного контролера інтерфейсу та керує всіма елементами, пов’язаними зі сценою (рис. А.32).

Контролер включає різні компоненти UI, кожен з яких відповідає за певний аспект взаємодії з користувачем. SceneView відображає тривимірну сцену, controlPanel містить елементи керування, а generationControls та terrainSettings надають доступ до налаштувань генерації та параметрів ландшафту.

Інтеграція SceneKit з UIKit вимагає ретельного налаштування взаємодії між двома фреймворками. Це включає правильне розміщення SCNView у ієрархії UIKit, налаштування делегатів та обробку подій.

Ці операції реалізуються у методі configureSceneView, який встановлює параметри відображення сцени та забезпечує накладення елементів інтерфейсу поверх неї (рис. А.33).

Інтеграція включає налаштування параметрів рендерингу, таких як антиаліасинг та частота кадрів, для забезпечення плавного відображення. Також налаштовується оверлей-сцена для відображення додаткових елементів UI поверх тривимірної сцени.

Елементи керування забезпечують зручний доступ до функцій генерації та налаштування ландшафту. Вони включають слайдери, кнопки та інші елементи UI, які дозволяють користувачу взаємодіяти з системою.

Їх реалізація відбувається в межах класу GenerationControlsView, який об’єднує всі необхідні елементи – від слайдерів до кнопок керування (рис. А.34).

Інтерфейс дозволяє змінювати основні параметри ландшафту за допомогою зручних слайдерів, що згруповані у відповідному блоці керування, як показано на рисунку 3.17.

Елементи керування налаштовуються з урахуванням потреб користувача та забезпечують інтуїтивно зрозумілу взаємодію. Слайдери дозволяють налаштовувати параметри генерації, а кнопки надають швидкий доступ до основних функцій.

Реалізація жестів та взаємодії з тривимірною сценою включає налаштування розпізнавачів жестів для масштабування, обертання та переміщення камери. Це забезпечує зручну навігацію по сцені та детальний огляд ландшафту.

Ці дії зосереджено в методі setupGestureRecognizers, який відповідає за обробку користувацьких жестів у просторі сцени (рис. А.35).

Система також включає обробку подій для оновлення UI при зміні параметрів генерації та налаштувань ландшафту. Це забезпечує синхронізацію між різними компонентами UI та тривимірною сценою.

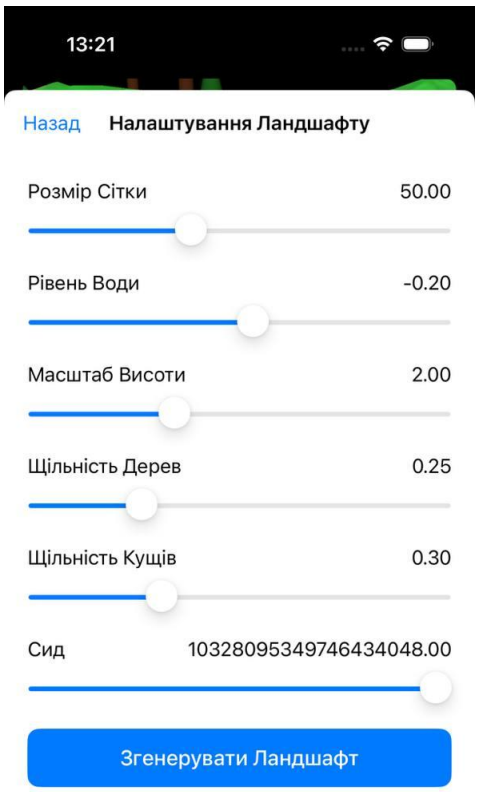

Рисунок 3.17 – Інтерфейс користувача з елементами налаштування параметрів ландшафту

Оптимізація продуктивності UI досягається через ефективне використання ресурсів та правильне налаштування анімацій. Система забезпечує плавне оновлення UI при зміні параметрів та генерації нового ландшафту.

Інтеграція з іншими компонентами системи забезпечується через чітко визначені інтерфейси. Це дозволяє легко додавати нові функції та модифікувати існуючі. Система спроектована з урахуванням можливого розширення функціоналу та інтеграції з новими технологіями.

Ця реалізація забезпечує створення зручного та інтуїтивно зрозумілого користувацького інтерфейсу для генерації та налаштування тривимірного

ландшафту. Система є масштабованою та може бути легко розширена для підтримки нових функцій та технологій.

3.3.4 Оптимізація рендерингу

Оптимізація рендерингу є критично важливим аспектом розробки тривимірного застосунку для генерації ландшафтів. Ефективне використання ресурсів, оптимізація геометрії та плавне оновлення сцени забезпечують стабільну роботу застосунку на різних пристроях. Розробка системи оптимізації включає кілька ключових компонентів, кожен з яких відповідає за певний аспект продуктивності.

Ефективне управління пам’яттю є фундаментальним для стабільної роботи застосунку. Система повинна оптимізувати використання пам’яті для текстур, геометрії та інших ресурсів, забезпечуючи плавну роботу без витоків пам’яті.

Цей процес реалізується в межах класу MemoryManager, який здійснює кешування текстур і геометрії тривимірної сцени (рис. А.36).

Система кешування текстур та геометрії дозволяє зменшити навантаження на пам’ять та прискорити завантаження ресурсів. Кеш автоматично очищається при досягненні максимального розміру, забезпечуючи ефективне використання пам’яті.

Оптимізація полігональної сітки є ключовим аспектом для забезпечення плавного рендерингу. Система повинна автоматично адаптувати складність геометрії залежно від відстані до камери та важливості деталей.

Механізм реалізовано в класі TerrainOptimizer, що застосовує рівні деталізації (LOD) відповідно до просторової відстані до об’єкта (рис. А.37).

Система рівнів деталізації (LOD) автоматично змінює складність геометрії залежно від відстані до камери. Це дозволяє зменшити

навантаження на GPU при відображенні віддалених частин ландшафту, зберігаючи високу деталізацію для близьких об’єктів.

Плавна генерація та оновлення сцени забезпечують комфортну взаємодію користувача з застосунком. Система повинна ефективно оновлювати сцену при зміні параметрів генерації та забезпечувати плавні переходи між станами.

Реалізацію цих задач забезпечує клас SceneUpdateManager, який виконує асинхронне оновлення сцени на основі нових параметрів генерації (рис. А.38).

Система оновлення сцени використовує чергу завдань для асинхронної генерації та оновлення геометрії. Це забезпечує плавну роботу UI при генерації нового ландшафту та уникнення блокування основного потоку.

Оптимізація рендерингу включає також налаштування параметрів рендерингу, таких як антиаліасинг та тіні, для забезпечення балансу між якістю зображення та продуктивністю.

Всі ці параметри конфігуруються через метод configureRendering у класі TerrainScene, який відповідає за узгодження налаштувань сцени з можливостями пристрою (рис. А.39).

Система також включає моніторинг продуктивності для виявлення проблем та оптимізації роботи застосунку. Це дозволяє адаптувати налаштування рендерингу залежно від потужності пристрою та поточного навантаження.

Інтеграція з іншими компонентами системи забезпечується через чітко визначені інтерфейси. Це дозволяє легко додавати нові функції оптимізації та модифікувати існуючі. Система спроектована з урахуванням можливого розширення функціоналу та інтеграції з новими технологіями.

Ця реалізація забезпечує ефективну оптимізацію рендерингу для створення реалістичного та інтерактивного тривимірного ландшафту. Система є масштабованою та може бути легко розширена для підтримки нових функцій оптимізації та технологій.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований мобільний застосунок для iOS, який здійснює процедурну генерацію тривимірного ландшафту з використанням карти висот. Рішення орієнтоване на створення адаптивного тривимірного середовища у реальному часі з використанням математичних методів моделювання.

Побудова рельєфу реалізується на основі поєднання відомих алгоритмів, зокрема шуму Перліна та Diamond-Square, що дає змогу створювати реалістичні топографічні поверхні з різноманітними варіаціями висоти. Усі компоненти сцени генеруються динамічно: тривимірна геометрія формується згідно з числовими параметрами карти висот, текстури та матеріали накладаються автоматично залежно від рельєфу, а в межах визначених висотних діапазонів додаються об’єкти, що імітують природні елементи – зокрема дерева, водойми та снігові ділянки.

У технічній реалізації застосовано мову Swift, яка завдяки сучасній архітектурі, високій продуктивності та підтримці Apple є оптимальним вибором для реалізації логіки генерації. SceneKit виступає основним інструментом візуалізації, що дозволяє будувати ієрархічні сцени, керувати об’єктами, налаштовувати освітлення, застосовувати камери та анімаційні ефекти. Інтеграція з UIKit забезпечує взаємодію з користувачем через інтерфейс, що дає змогу виконувати масштабування, обертання сцени, змінювати параметри генерації та повторно створювати ландшафт без потреби перезапуску застосунку.

Особливий акцент зроблено на досягненні високої швидкодії та плавної візуалізації. У процесі тестування застосунок продемонстрував стабільну роботу, підтримку високої частоти оновлення кадру, ефективне використання оперативної пам’яті та мінімальне навантаження на процесор мобільного пристрою. Це дозволяє використовувати розроблене рішення навіть на

моделях із базовими апаратними характеристиками, зберігаючи якість графіки та чутливість взаємодії.

Отримані результати підтверджують доцільність застосування процедурної генерації для створення візуально насичених і водночас технічно оптимізованих тривимірних середовищ у мобільних застосунках. Розроблене рішення має високий потенціал до масштабування та адаптації під різні сфери – від навчальних візуалізацій до створення інтерактивних ігрових світів або симуляцій реального середовища. Подальший розвиток може включати розширення бібліотеки природних елементів, впровадження алгоритмів ерозії для більшої правдоподібності рельєфу, інтеграцію з технологіями доповненої реальності та застосування елементів штучного інтелекту для модифікації або персоналізації ландшафту в реальному часі.

Таким чином, реалізоване програмне рішення є прикладом ефективного поєднання теоретичних засад процедурного моделювання з практичними підходами до мобільної розробки, що відповідає сучасним вимогам до інтерактивного тривимірного контенту, який потребує гнучкості, продуктивності та варіативності.

Результати роботи апробовано у вигляді тез доповіді під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [15].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Smelik, R. M., Tutenel, T., Bidarra, R., & Benes, B. (2014, September). A survey on procedural modelling for virtual worlds. In *Computer graphics forum* (Vol. 33, No. 6, pp. 31-50).

2. Parish, Y. I., & Müller, P. (2001, August). Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (pp. 301-308).

3. Real-Time Procedural Generation with GPU Work Graphs. URL: https://gpuopen.com/download/publications/Real-Time_Procedural_Generation_with_GPU_Work_Graphs-GPUOpen_preprint.pdf (дата звернення 14.04.2025).

4. Building iOS Apps with SceneKit and 3D Graphics. URL: https://reintech.io/blog/building-ios-apps-with-scenekit-and-3d-graphics (дата звернення 14.04.2025).

5. SceneKit. URL: https://developer.apple.com/documentation/scenekit/ (дата звернення 14.04.2025).

6. Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., & Worley, S. (2002). *Texturing and modeling: a procedural approach*. Elsevier.

7. Bernardi, A., Gadia, D., Maggiorini, D., Palazzi, C. E., & Ripamonti, L. A. (2021). Procedural generation of materials for real-time rendering. *Multimedia Tools and Applications*, 80(9), 12969-12990.

8. Gustavson, S. (2005). Simplex noise demystified. *Linköping University, Linköping, Sweden, Research Report*, *1*(2), 6.

9. Cook, R. L. (1984, January). Shade trees. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (pp. 223-231).

10. Merizzi, F. (2024). Procedural terrain generation with style transfer. *arXiv preprint arXiv:2403.08782*.

11. Dajkhosh, S. (2024). Utilizing WaveFunctionCollapse Algorithm for Procedural Generation of Terrains using Remotely Sensed Elevation Data. *arXiv preprint arXiv:2412.04688*.

12. Perlin, K. (1985). An image synthesizer. *ACM Siggraph Computer Graphics, 19*(3), 287-296.

13. Musgrave, F. K., Kolb, C. E., & Mace, R. S. (1989). The synthesis and rendering of eroded fractal terrains. *ACM Siggraph Computer Graphics, 23*(3), 41-50.

14. The Graphics Codex. URL: https://graphicscodex.com/app/app.html? (дата звернення 14.04.2025).

15. Striuk D.S. (2025) Procedural generation of landscapes for multimedia applications. *Радіоелектроніка і молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків 16-19 квітня 2025 р.).* Харків: ХНУРЕ, 2025. Т. 7. С. 131-133.

16. Häggström, H. (2006). *Real-time generation and rendering of realistic landscapes* (Doctoral dissertation, Master's thesis, University of Helsinki).

17. Bodyanskiy, Y., Vynokurova, O., Kobylin, I., & Kobylin, O. (2016). Adaptive fuzzy clustering of short time series with unevenly distributed observations in Data Stream Mining tasks. *Information Technology and Management Science, 19*(1), 23-28.

18. Scene Kit Tutorial: Getting Started. URL: https://www.kodeco.com/2243-scene-kit-tutorial-getting-started (дата звернення 14.04.2025).

19. Unity Manual. URL: https://docs.unity3d.com/Manual/index.html (дата звернення 14.04.2025).

20. Unreal Engine documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-4-documentation?application_version=5.4 (дата звернення 14.04.2025).

21. Creating a Widget with a 3D View Using SceneKit in iOS Development with Swift. URL: https://medium.com/@ios_guru/creating-a-widget-with-a-3d-view-using-scenekit-dea978f3c70d (дата звернення 14.04.2025).

22. Multithreading techniques for 3D rendering with SceneKit in Swift. URL: https://colinchswift.github.io/ (дата звернення 14.04.2025).

23. Basics of 3D Rendering with SceneKit. URL: https://byteandcloud.com/swift/basics_of_3d_rendering.php (дата звернення 14.04.2025).

24. Бодянський, Є., Винокурова, О., Кобилін, І., & Мулеса, П. (2017). Адаптивна матрична нейро-фаззі самоорганізовна мережа для кластеризації багатовимірних потоків даних. *Вісник НУ "Львівська політехніка"*, (864), 314–319.

25. Бодянський, Є. В., Винокурова, О. А., Ізонін, І. В., Кобилін, І. О., & Мулеса, П. П. (2017). Кластеризація багатовимірних часових рядів на основі адаптивної матричної нейро-фаззі самоорганізовної мережі. *Intellectual Systems for Decision Making and Problems of Computational Intelligence*, (247).

26. Кобилін, І. О., & Ніколайчук, А. І. (2024). Monitoring and diagnosing faults in online mode using time series data. *Системи обробки інформації*, (3(178)), 27–32.

27. Кобилін, І. О., & Харченко, А. І. (2024). Classification techniques for computer vision. *Системи обробки інформації*, (3(178)), 33–41.

28. Kharchenko, A. I., & Kobylin, I. O. (2023). Performance analysis of support vector machine for vehicle classification. *V. N. Karazin Kharkiv National University*, 101.

29. Bodyanskiy, Y., Vynokurova, O., Szymański, Z., Kobylin, I., & Kobylin, O. (2016, August). Adaptive robust models for identification of nonstationary systems in data stream mining tasks. In *2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP)* (pp. 263-268). IEEE.

30. Xcode. URL: https://developer.apple.com/xcode/ (дата звернення 14.04.2025).