Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping

Виконав: студент II курсу
групи 1ПІ-23м спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Ковальчук С.І.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Романюк О.В.
(прізвище та ініціали)
«06» грудня 2024 р.
Опонент: к.т.н., доц. каф. ЗІ Войтович О.П.
(прізвище та ініціали)
«06» грудня 2024 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«06» грудня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти II-й (магістерський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
Романюк О.Н. _____
18 вересня 2024 р.

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

Ковальчуку Сергію Ігоровичу

1. Тема роботи – «Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping».

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 17 вересня 2024 р. №310.

2. Строк подання студентом роботи 3 грудня 2024 року.

3. Вихідні дані до роботи: методи рельєфного текстурування – bump mapping, normal mapping, parallax mapping, steep parallax mapping, parallax occlusion mapping; метод виявлення контурів у цифровому зображенні – оператор Собеля; рушій візуалізації – jMonkeyEngine 3; середовище розробки – jMonkeyEngine SDK; мова програмування – Java; вхідні дані – текстури, параметри шейдерів, введення з клавіатури, дані про розташування джерела світла та спостерігача; вихідні дані – демонстрація роботи методів текстурування, показники продуктивності.

4. Зміст текстової частини: вступ; аналіз стану питання та постановка задач дослідження; розробка методів підвищення ефективності формування зображень рельєфних поверхонь; розробка програмного забезпечення; дослідження ефективності методів текстурування та тестування програмного забезпечення; економічна частина; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження; наукова новизна та

практична цінність отриманих результатів; метод генерації карти складності текстури; метод формування зображень рельєфних поверхонь на базі parallax occlusion mapping; реалізація програмного забезпечення; тестування програмного забезпечення; результати експериментальних досліджень розробленого методу; апробація та публікації результатів роботи; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Романюк О.В., к.т.н., доцент кафедри ПЗ	19.09.24	02.12.24
5	Причепа І.В., к.е.н., доцент кафедри ЕПВМ	23.11.24	30.11.24

7. Дата видачі завдання 19 вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та постановка задач дослідження	20.09.24 – 27.09.24	виконано
2	Проєктування архітектури та структури застосунку	28.09.24 – 07.10.24	виконано
3	Розробка методу генерації карти складності текстури	08.10.24 – 15.10.24	виконано
4	Розробка методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping	16.10.24 – 23.10.24	виконано
5	Аналіз та вибір засобів для реалізації програмного продукту	24.10.24 – 31.10.24	виконано
6	Реалізація програмного засобу на основі розроблених методів	01.11.24 – 14.11.24	виконано
7	Дослідження ефективності розроблених методів і тестування програмного забезпечення	15.11.24 – 22.11.24	виконано
8	Економічна частина	23.11.24 – 30.11.24	виконано

Студент _____ (підпис) **Ковальчук О** (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ (підпис) **Романюк О.** (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.925.4

Ковальчук С.І. Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping : магістерська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2024. 145 с.

На укр. мові. Бібліогр. : 37 назв.; рис. : 64; табл. : 16.

У магістерській кваліфікаційній роботі розроблено методи та програмний засіб для підвищення ефективності формування зображень рельєфних поверхонь на основі методу parallax occlusion mapping (POM).

Основні наукові результати дослідження включають метод генерації карти складності з використанням оператора Собеля та удосконалений метод рельєфного текстурування на базі POM. Запропонований підхід дозволяє динамічно регулювати кількість шарів дискретизації залежно від складності текстури та кута огляду, що дозволило підвищити ефективність рендерингу до 24% порівняно з традиційним підходом без погіршення візуальної якості.

Практична значущість розробки підтверджена успішною реалізацією запропонованих методів у програмному забезпеченні для демонстрації та порівняння технік рельєфного текстурування. Під час тестування застосунку було підтверджено повну відповідність функціональним вимогам та відсутність критичних помилок. Оцінка економічної ефективності свідчить про високий комерційний потенціал розробки та доцільність її реалізації.

Отримані в магістерській кваліфікаційній роботі результати можуть бути використані у системах комп’ютерної графіки, зокрема при розробці відеоігор, створенні анімаційних фільмів та вдосконаленні графічних рушіїв.

Ключові слова: parallax occlusion mapping, накладання текстур, рельєфне текстурування, програмування шейдерів, карти складності.

ABSTRACT

Kovalchuk S.I. Development of methods and software tool to improve the efficiency of forming images of relief surfaces based on parallax occlusion mapping. Vinnytsia : VNTU, 2024. 145 p.

In Ukrainian language. Bibliographer: 37 titles; fig. : 64; tabl. : 16.

In the master's thesis, methods and software tool were developed to improve the efficiency of forming images of relief surfaces based on the parallax occlusion mapping (POM) method.

The main scientific results of the study include a method for generating a complexity map using the Sobel operator and an improved method of relief mapping based on POM. The proposed approach allows to dynamically adjust the number of discretization layers depending on the texture complexity and viewing angle, which made it possible to increase rendering efficiency by up to 24% compared to the traditional approach without compromising visual quality.

The practical significance of the development is confirmed by the successful implementation of the proposed methods in software for demonstrating and comparing relief mapping techniques. During the testing of the application, full compliance with the functional requirements and the absence of critical errors were confirmed. The assessment of economic efficiency shows the high commercial potential of the development and the feasibility of its implementation.

The results obtained in the master's thesis can be used in computer graphics systems, in particular in the development of video games, the creation of animated films and the improvement of graphics engines.

Keywords: parallax occlusion mapping, texture mapping, relief mapping, shader programming, complexity maps.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сфері комп’ютерної графіки зберігається постійний попит на реалістичність зображення, що зумовлено розвитком таких галузей, як ігри, віртуальна реальність та кінематограф [1]. Сучасні програми часто вимагають високодеталізованого візуального контенту, що робить реалізм важливою метою в дизайні графічних сцен.

Однак прагнення до реалістичності часто вступає в конфлікт з не менш важливою вимогою до високої продуктивності, особливо в інтерактивних продуктах. Досягнення якісних візуальних ефектів може вимагати значних обчислювальних затрат, що призводить до зниження продуктивності, особливо коли йдеться про обробку зображення у реальному часі. Питання балансу між візуальною достовірністю та продуктивністю є фундаментальною проблемою комп’ютерної графіки, яка ще не повністю вирішена сучасними технологіями [2].

Одним із підходів до підвищення реалістичності без шкоди для продуктивності є ефективне використання технік накладання текстур. Зокрема, використовуючи методи рельєфного текстурування, можна покращити сприйняття глибини і тонкощів поверхні, зміщуючи координати текстури під час рендерингу, що дозволяє створити ілюзію деталізованої 3D-поверхні на пласкій геометрії. Однак сучасні методи здебільшого покладаються на статичні параметри у своїй роботі, що робить актуальним пошук інноваційних способів підвищення точності та ефективності.

Parallax occlusion mapping (POM) – це один з таких методів рельєфного текстурування, який покращує сприйняття глибини поверхні, зміщуючи координати текстури під час рендерингу [3]. Метод ґрунтується на вибірці декількох значень глибини вздовж напрямку погляду спостерігача, ефективно імітуючи вигляд текстурованої поверхні. Для цього діапазон глибин текстури ділиться на визначену кількість частин – шарів дискретизації.

Одним з основних недоліків згаданої техніки є використання фіксованої кількості шарів дискретизації, незалежно від особливостей текстури та

розташування камери. Хоча такий підхід забезпечує стабільну якість, він не враховує варіацій складності текстури або кут огляду поверхні. Наприклад, на ділянках з різкою зміною висоти текстури варто збільшувати частоту дискретизації, а для ділянок з меншою деталізацією використовувати менше шарів. Таким чином, POM часто виділяє більше ресурсів, ніж потрібно в ситуаціях, де високий рівень деталізації поверхні не є критичним, що може призвести до зниження продуктивності.

Описані обмеження підкреслюють доцільність застосування адаптивного підходу, який динамічно регулює кількість шарів дискретизації для POM залежно від складності текстури та кута огляду. Такий підхід дозволить підтримувати візуальну точність, яку забезпечує POM, та зменшити обчислювальні витрати, що робить його більш придатним для рендерингу у реальному часі.

Тому актуальним є питання підвищення ефективності формування зображень рельєфних поверхонь, оскільки існуючі методи не задовольняють зростаючі вимоги галузей застосування тривимірної комп’ютерної графіки. Це передбачає розробку нових методів та програмного засобу для демонстрації їхньої роботи.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення ефективності формування зображень рельєфних поверхонь за рахунок зменшення кількості обчислень.

Основними задачами дослідження є:

– провести аналіз стану питання та методів розв’язання задачі;

– здійснити архітектурне проєктування та розробити структуру програмного засобу;

– розробити метод генерації карти складності текстури;

– розробити метод формування зображень рельєфних поверхонь на базі parallax occlusion mapping;

– розробити засіб для генерації карт складності;

– здійснити розробку програмного забезпечення для демонстрації та порівняння методів рельєфного текстурування;

– провести тестування працездатності розробленого ПЗ;

– провести експериментальні дослідження методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping;

– розробити інструкцію користувача.

Об’єкт дослідження – процес кінцевої візуалізації тривимірних об’єктів у системах комп’ютерної графіки.

Предмет дослідження – методи та засоби рельєфного текстурування тривимірних об’єктів.

Методи дослідження. У процесі досліджень використовувались такі методи: теорія чисел та чисельних методів, лінійна алгебра, теорія диференціально-інтегрального числення, методи аналітичної геометрії, методи цифрової обробки зображень для розробки методу генерації карти складності текстури та методу формування зображень рельєфних поверхонь; елементи аналітичної статистики для експериментальних досліджень розроблених методів; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів:

1. Уперше запропоновано метод генерації карти складності, особливість якого полягає у визначенні рівня деталізації текстури шляхом аналізу градієнтів карти висот, що дозволяє здійснювати адаптивну дискретизацію діапазону глибин при текстуруванні.

2. Подальшого розвитку отримав метод рельєфного текстурування parallax occlusion mapping, який, на відміну від існуючих, динамічно регулює кількість шарів дискретизації залежно від складності текстури та кута огляду, що

дозволило підвищити ефективність методу до 24% за рахунок зменшення кількості обчислень без погіршення якості зображень.

Практичне значення отриманих результатів полягає в тому, що на основі отриманих у магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та програмні засоби для підвищення ефективності формування зображень рельєфних поверхонь.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У наукових працях, опублікованих у співавторстві, автору належать такі результати: аналіз методів імітації нерівностей на поверхні графічних об’єктів при накладанні текстур [4]; напрямки удосконалення методу parallax occlusion mapping [5]; шейдерна реалізація методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping [**Error! Reference source not found.**].

Апробація матеріалів магістерської кваліфікаційної роботи. Результати роботи доповідалися на:

– LIII Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2024 р.) [4];

– IV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп’ютерні ігри та мультимедіа як інноваційний підхід до комунікації» (Одеса, 2024 р.) [5];

– Міжнародній науково-практичній Інтернет-конференції 20-21 листопада 2024 р. «Електронні інформаційні ресурси: створення, використання, доступ та управління» (Суми/Вінниця, 2024 р.) [6].

Публікації. За тематикою дослідження опубліковано 3 наукові праці у збірниках матеріалів конференцій.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз методів формування зображень рельєфних поверхонь

У галузі комп’ютерної графіки та 3D-візуалізації прагнення до реалістичності є безперервним процесом. Одним із ключових питань є баланс між обчислювальною складністю та візуальною достовірністю сцени. Відображення рельєфних поверхонь є особливо ресурсомістким завданням, яке може бути оптимізоване на етапі накладання текстури [7].

Рельєфне текстурування – це техніка накладання текстур, що використовується для імітації нерівностей на пласкій поверхні. Такі методи дозволяють підвищити реалістичність текстури, оскільки дають змогу відобразити більше деталей, ніж полігональна поверхня, та водночас потребують менше обчислень для рендерингу сцени. Серед відомих сьогодні методів рельєфного текстурування можна виділити bump mapping, normal mapping, parallax mapping та його удосконалені різновиди steep parallax та parallax occlusion mapping (POM). Розглянемо кожен із цих методів детальніше.

У 1978 році Джеймс Блінн вперше представив метод рельєфного текстурування, або bump mapping [8]. Особливістю цього методу є те, що відображення нерівностей на поверхні досягається без геометричних модифікацій об’єкта. Натомість вектори нормалей поверхні збурюються відповідно до карти висот (рисунок 1.1, а) та використовуються для розрахунків освітлення (наприклад, за допомогою моделі віддзеркалення Фонга), створюючи візуальні нерівності замість гладкої поверхні (рисунок 1.1, б).

Головним недоліком bump mapping є те, що напрямок векторів нормалей необхідно рахувати кожного разу під час накладання текстури. Це створює обчислювальне навантаження, якого можна уникнути за допомогою кодування напрямку векторів нормалей у спеціальній карті нормалей. На основі цього підходу створено більш просунутий метод текстурування – normal mapping.

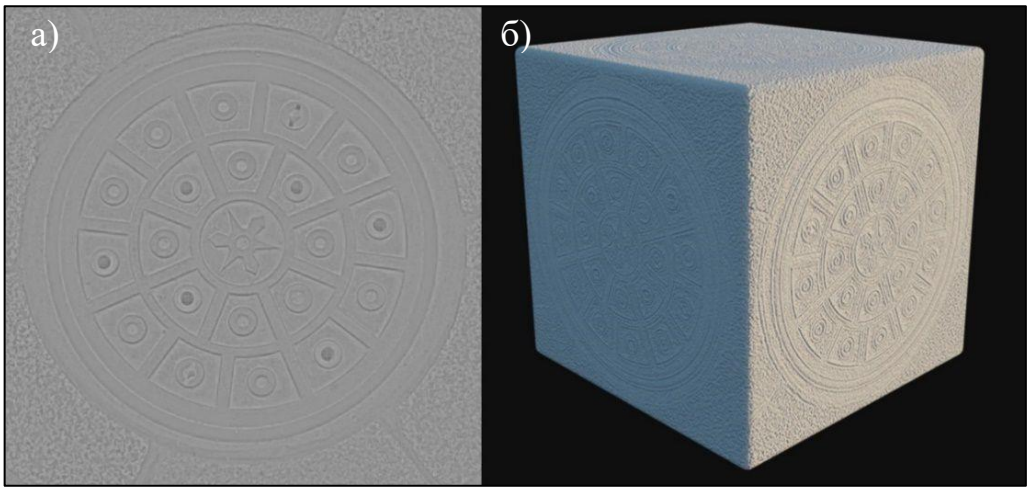


Рисунок 1.1 – Приклад застосування методу bump mapping

Normal mapping використовує значення з карти нормалей для відображення рельєфу, де кожен тексель (піксель текстури) місить закодовану у RGB-каналах інформацію про напрямок вектору нормалі поверхні у цій точці (рисунок 1.2, а). На рисунку 1.2, б наведено приклад рельєфного текстурування об’єкта із використанням цього методу.

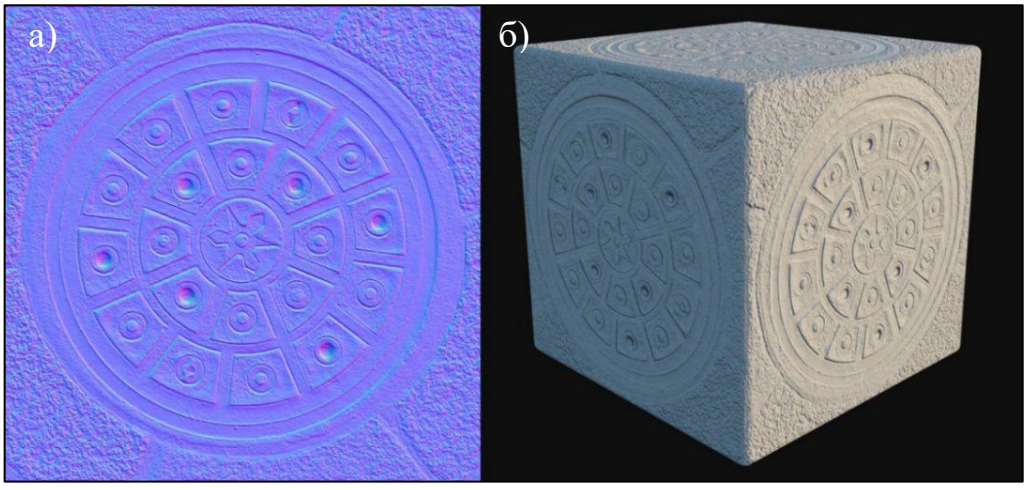


Рисунок 1.2 – Normal mapping

RGB-канали представляють компоненти X, Y та Z вектора нормалі відповідно [9]. Найчастіше значення у карті нормалей знаходяться в межах [-1; 1] у кожному каналі: значення 0 означає відсутність відхилення вектора від початкового напрямку, а додатні чи від’ємні значення вказують на відхилення вздовж відповідної осі.

Попереднє визначення напрямків нормалей дозволяє точно імітувати особливості поверхні та ефекти освітлення, чого неможливо досягти використовуючи bump mapping. Водночас обидва підходи мають суттєвий недолік: вони залежні від місця розташування спостерігача. Оскільки ефект рельєфності досягається лише за рахунок освітлення, а сам силует об'єкта залишається незмінним, під гострими кутами огляду ця ілюзія стає непереконливою.

Parallax mapping – це техніка, яка створює ілюзію глибини, імітуючи видиме зміщення текстури поверхні залежно від розташування спостерігача без змінення базової геометрії об'єкта. Цей метод полягає у коригуванні координат текстури в заданому текселі A таким чином, щоб імітувати вигляд текстури під дещо іншим кутом B (рисунок 1.3). При цьому пошук точки B виконується приблизно, оскільки точне знаходження точки перетину вектора спостереження $\overline{V}$ з текстурою є ресурсомістким процесом [10].

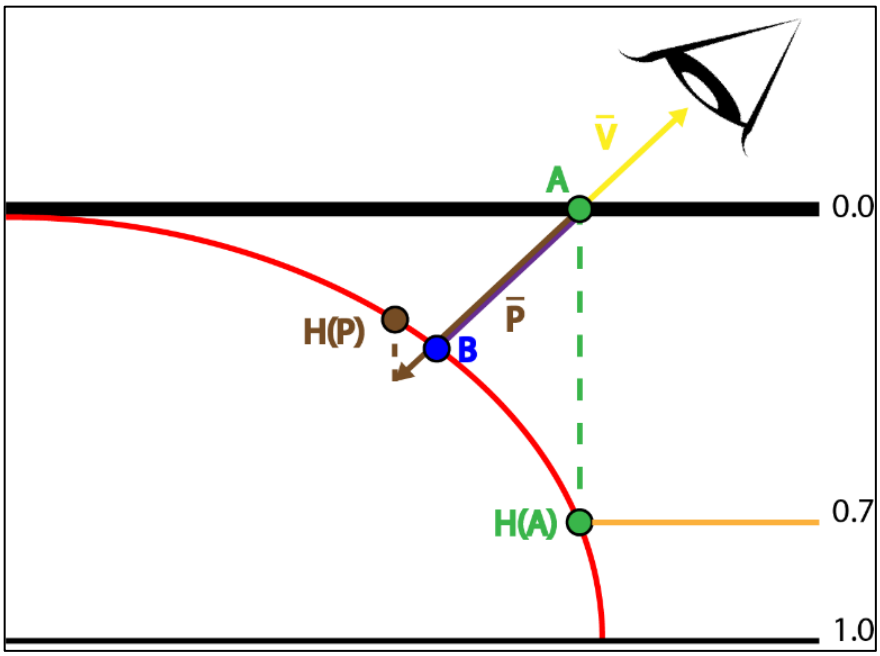


Рисунок 1.3 – Принцип роботи parallax mapping

Принцип роботи parallax mapping полягає в обчисленні вектора $\overline{P}$ шляхом віднімання вектора напрямку спостереження $\overline{V}$ від координат текстури в точці, отриманої з карти глибин (зображення, обернене до карти висот) у точці A. Цей

вектор дозволяє отримати скоригований тексель B текстури, який необхідно відобразити спостерігачеві на місці текселя A.

Створюючи ілюзію глибини за рахунок зміщення текстури, parallax mapping частково вирішує проблему непереконливості відображення нерівностей на поверхні. Однак цей підхід також не позбавлений недоліків: різка зміна висоти поверхні текстури призводить до ефектів ступінчастості та аліасингу.

Steep parallax mapping – це вдосконалення parallax mapping, яке використовує декілька точок для побудови вектора $\overline{P}$ замість однієї, що дозволяє підвищити точність методу та зменшити вираженість візуальних артефактів. Принцип роботи steep parallax mapping зображено на рисунку 1.4.

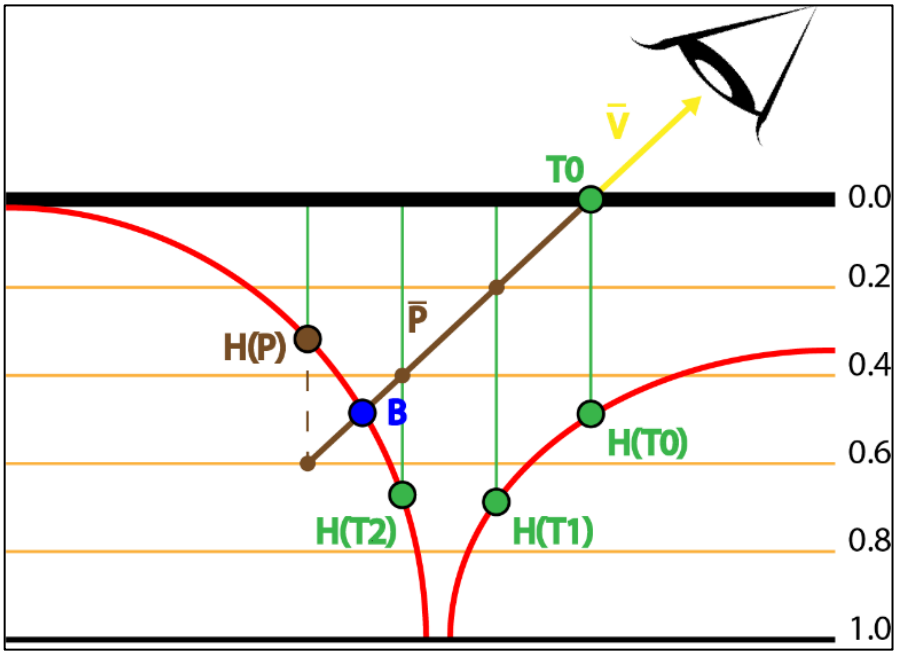


Рисунок 1.4 – Принцип роботи steep parallax mapping

Ідея цього підходу полягає у тому, що загальний діапазон глибин текстури рівномірно розділяється на декілька шарів. Рухаючись вздовж напрямку $\overline{V}$, для кожного з цих шарів необхідно знайти точки перетину з вектором $\overline{P}$, зсуваючи координати текстури, доки не буде знайдено значення глибини, яке буде меншим за значення глибини поточного шару. Це і буде скоригований тексель H(P), який необхідно відобразити.

Представлена у 2007 році техніка parallax occlusion mapping дозволяє досягнути ще більшої точності у порівнянні зі steep parallax mapping. Вона базується на тих самих принципах, але замість використання координат текстури що відповідають кінцю вектора $\overline{P}$, необхідно отримати точку між H(T2) до перетину із текстурою та H(T3) після цього, застосувавши лінійну інтерполяцію (рисунок 1.5).

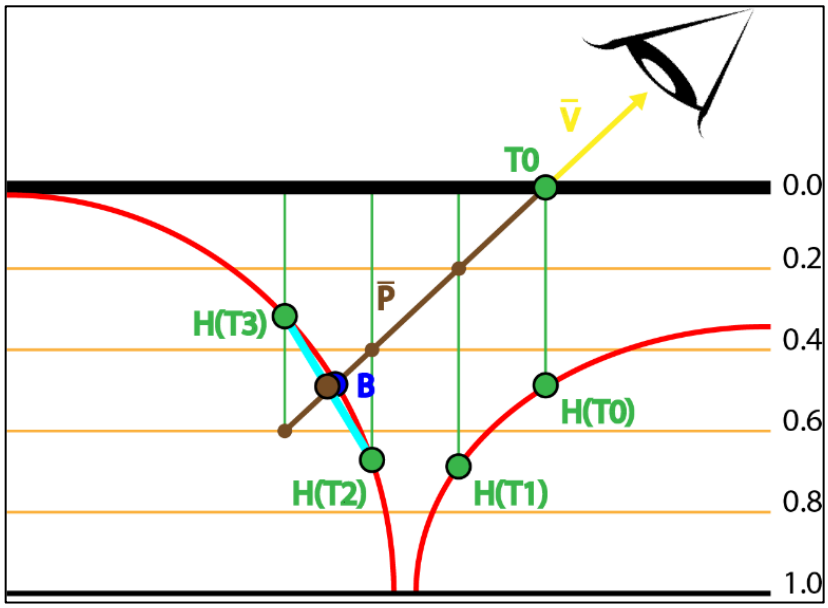


Рисунок 1.5 – Принцип роботи parallax occlusion mapping

Описаний метод став значним проривом у рельєфному текстуруванні. Він пропонує баланс між візуальною точністю, обчислювальною ефективністю та простотою реалізації. Хоча цей підхід також не позбавлений недоліків, його переваги роблять його цінним інструментом для підвищення реалістичності зображень рельєфних поверхонь.

Отже, розвиток методів рельєфного текстурування дозволив у значній мірі підвищити реалістичність комп'ютерних зображень. Починаючи з базових ілюзій bump mapping та закінчуючи просунутим parallax occlusion mapping, ці методи продовжують розширювати межі можливого та роблять свій внесок у досягнення ширшої мети – подолання розриву між комп'ютерною графікою та реальністю.

1.2 Порівняльний аналіз аналогів

У сфері комп’ютерної графіки існує безліч інструментів для накладання текстур, створення матеріалів та візуалізації різноманітних технік текстурування. Ці програми варіюються від складних систем для відображення текстур до спеціалізованих застосунків, що зосереджуються на UV-мапінгу (накладанні двовимірного зображення на тривимірні об’єкти) або процедурній генерації текстур.

Прикладами програмного забезпечення для створення матеріалів та маніпулювання текстурами можуть слугувати ArmorPaint, PixPlant та UVMapper. Порівняємо ці інструменти із власною розробкою.

ArmorPaint (рисунок 1.6) – це самостійне програмне забезпечення, що використовується для відображення текстур на основі фізики [11]. Воно має функцію створення власних матеріалів на основі вузлів та підтримує фізичний рендеринг (PBR). Попри свої переваги, ArmorPaint не має вбудованої підтримки методів рельєфного текстурування, а також не має функції їх візуального порівняння.

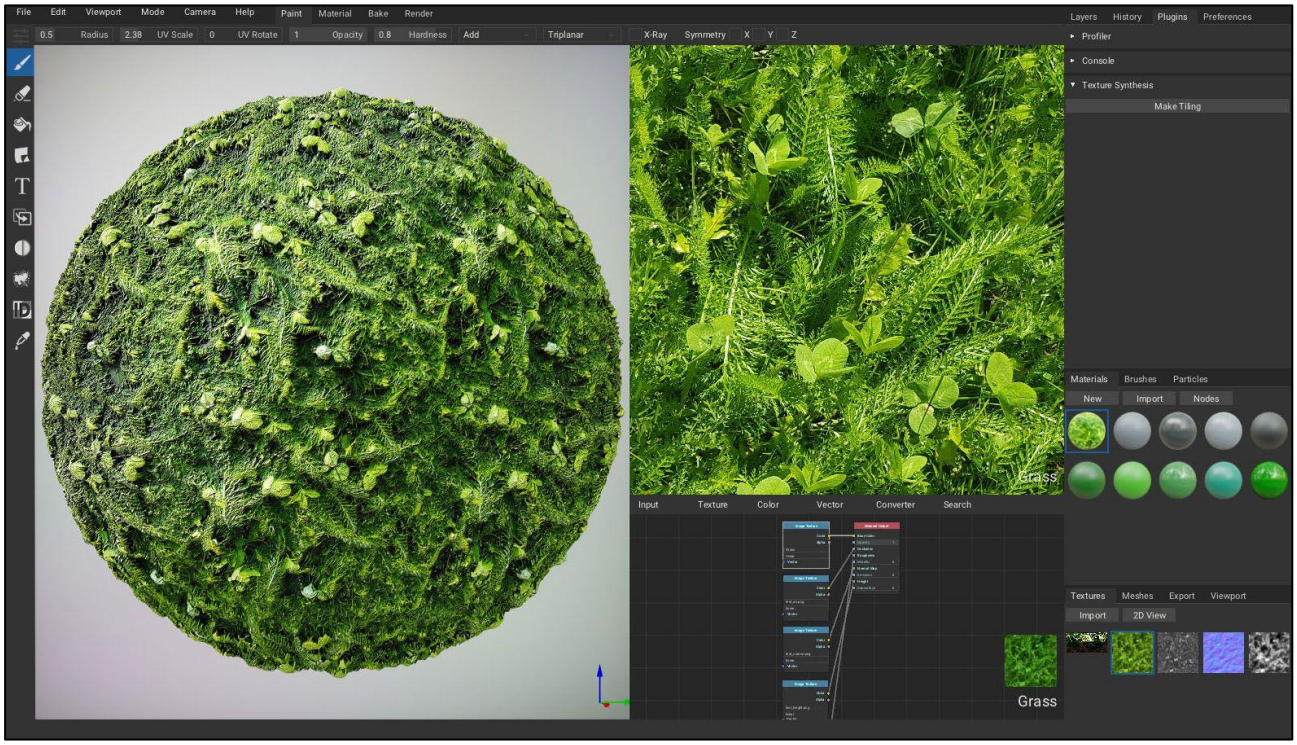


Рисунок 1.6 – ArmorPaint

PixPlant (рисунок 1.7) – це інструмент для створення текстур [12]. Він дозволяє створювати такі компоненти текстури, як карти нормалей, висот, а також карт навколишньої оклюзії (ambient occlusion). Хоча PixPlant ефективний для створення текстур, він не орієнтований на інтерактивну 3D-візуалізацію та не має функції порівняння різних методів текстурування.

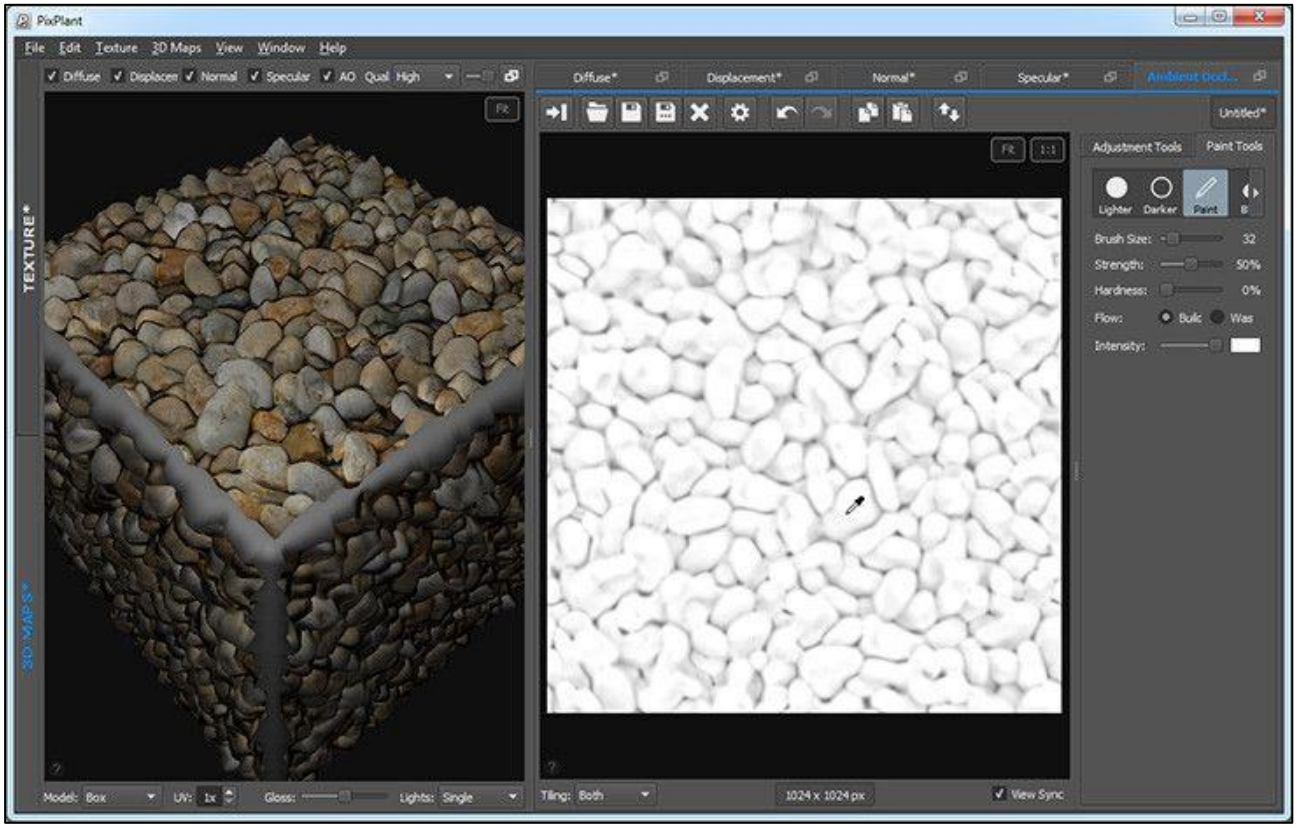


Рисунок 1.7 – PixPlant

UVMapper (рисунок 1.8) – це автономна утиліта для створення і модифікації UV-координат для n-сторонніх полігональних 3D-моделей [13]. Цей інструмент пропонує засоби для створення та налаштування UV-розгорток, але не підтримує просунуті методи текстурування, такі як POM. Функціональність UVMapper обмежена роботою з UV-розкладкою текстури, що робить його радше інструментом попередньої обробки текстур, аніж платформою для порівняння методів текстурування.

Порівняльний аналіз особливостей розглянутих аналогів із власною реалізацією подано в табл. 1.1.

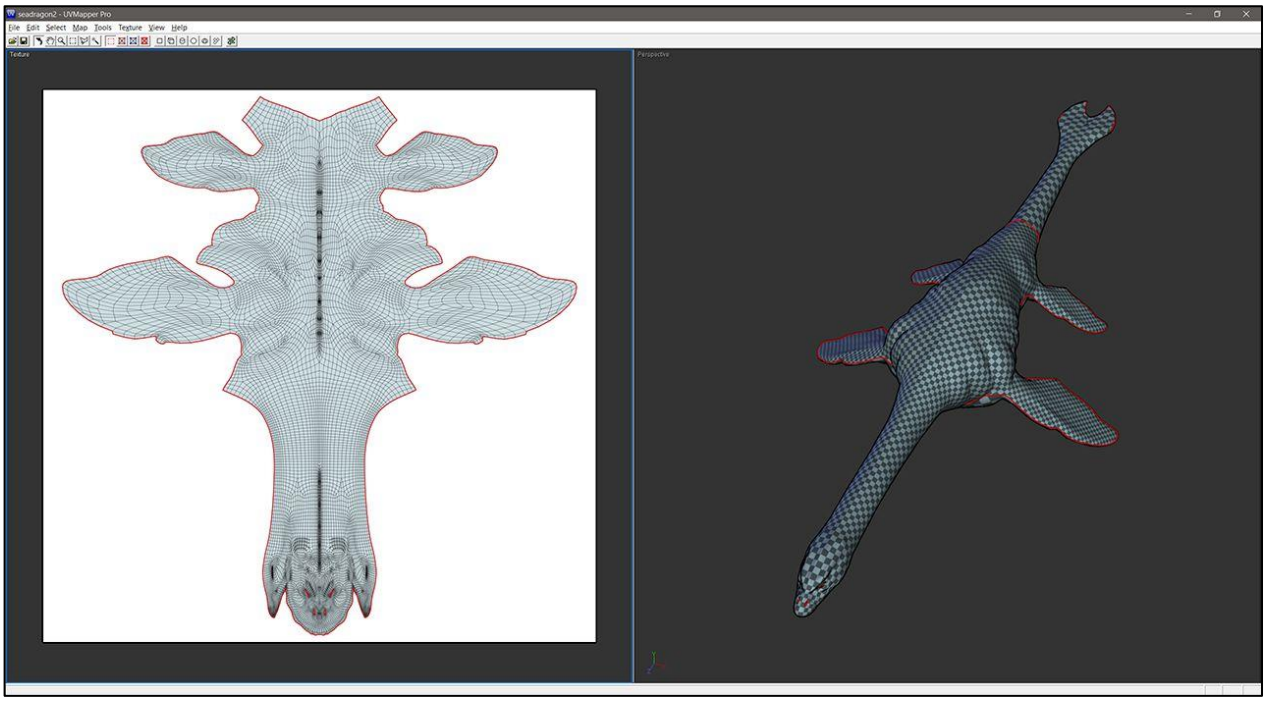


Рисунок 1.8 – UVMapper

Таблиця 1.1 – Порівняльний аналіз аналогів

Критерій	**ArmorPaint**	**PixPlant**	**UVMapper**	**Власна реалізація**
Підтримка POM	–	–	–	+
Візуалізація у реальному часі	+	+	–	+
Продуктивність та ефективність використання ресурсів	–	+	+	+
Простота використання	+	+	+	+
Можливість порівняння методів текстурування	–	–	–	+
Налаштування та гнучкість	+	+	–	+
Всього	**3**	**4**	**2**	**6**

З наведеної вище таблиці видно, що кожна програма має свої сильні сторони, проте власна розробка перевершує їх в аспектах, що стосуються демонстрації та порівняння методів накладання текстур:

1. На відміну від власної розробки, програми-аналоги не мають вбудованої підтримки методів рельєфного текстурування, що робить її ідеальним

застосунком для порівняння ефектів цього конкретного методу текстурування з іншими.

2. Порівняно з ArmorPaint, власна розробка може бути оптимізована для ефективної демонстрації методів рельєфного текстурування в реальному часі. Це дозволяє зробити застосунок більш доступним для користувачів із застарілим апаратним забезпеченням.

3. Можливість перемикання між різними методами текстурування під час виконання програми є унікальною особливістю власної розробки. Жоден з аналогів не має функціоналу безпосереднього порівняння технік, що робить розроблюваний застосунок особливо привабливим для користувачів, які хочуть візуально оцінити роботу методів рельєфного текстурування та порівняти їх між собою.

Таким чином, аналоги, що існують на ринку сьогодні, слугують для своїх цілей у процесі створення текстур та UV-розгорток, проте їм бракує спеціалізованого функціоналу, необхідного для всебічного порівняння різних технік текстурування. Власна реалізація дозволяє заповнити цю прогалину, пропонуючи користувачеві інструмент для інтерактивної демонстрації методів зображення рельєфних поверхонь у реальному часі.

1.3 Аналіз напрямків удосконалення методу parallax occlusion mapping

Техніки накладання текстур відіграють значну роль у питанні реалістичності віртуальної сцени. Однією з таких технік є parallax occlusion mapping, який дозволяє імітувати складні деталі поверхні без ускладнення геометрії 3D-об’єкта. Це особливо важливо для комп’ютерних ігор, де оптимізація продуктивності має вирішальне значення для забезпечення якісного досвіду користувача [14].

POM досягає покращення сприйняття деталей поверхні за рахунок зміщення координат текстури залежно від розташування спостерігача, створюючи ілюзію тримірної поверхні на пласкій геометрії. Це дозволяє значно покращити візуальну достовірність сцени, що сприяє більшому зануренню у

віртуальне середовище.

Ця техніка може бути значно покращена за рахунок динамічного регулювання кількості шарів дискретизації залежно від параметрів сцени чи на основі складності окремих ділянок текстури. Розглянемо можливі варіанти реалізації такого покращення детальніше.

Однією з проблем parallax occlusion mapping є пошук балансу між якістю рендерингу та продуктивністю. Цей метод дозволяє досягти кращої візуальної точності зі збільшенням кількості шарів дискретизації діапазону глибин текстури, проте це відбувається за рахунок збільшення кількості обчислень. Крім того, статичність традиційних реалізації POM обмежує їхню ефективність. Наприклад, для рендерингу текстури використовується однакова кількість шарів, незалежно від того, чи потребує певна ділянка більшої чи меншої деталізації. Це стає причиною неефективної обробки, оскільки для простих текстур використовуються зайві обчислення, у той час як більш складні поверхні виявляються недостатньо деталізованими.

Ідея полягає в тому, щоб впровадити моделі машинного навчання, зокрема нейронні мережі, для динамічного регулювання кількості шарів для POM. Ці моделі передбачатимуть оптимальну кількість шарів на основі таких факторів, як кут огляду, деталізація текстури та складність сцени.

Вхідними даними для такої нейронної мережі може слугувати розташування спостерігача, кут огляду поверхні, відстань до текстури тощо [15]. На основі цієї інформації система зможе передбачати необхідну кількість шарів дискретизації у реальному часі. Цей метод дозволяє зробити текстурування адаптивним, забезпечуючи більш якісний рендеринг там, де це необхідно, та зменшити обчислювальне навантаження на простіших ділянках.

Отже, використання нейронних мереж разом з методом POM дозволяє оптимізувати процес накладання текстур без втрати якості зображення. Такий підхід відкриває нові можливості як у підвищенні продуктивності, так і в гнучкості рельєфного текстурування.

Іншим можливим напрямком удосконалення є використання аналізу

висотної карти для динамічного налаштування кількості шарів дискретизації. Цього можна досягти за рахунок використання спеціальних карт складності на додачу до традиційного POM. Такі карти містять інформацію про локальну складність ділянок текстури, дозволяючи динамічно адаптувати кількість шарів для обчислення фінальних координат текселя текстури.

На рисунку 1.9 наведено приклад застосування запропонованого підходу. На основі карти висот текстури (рисунок 1.9, а) за допомогою оператора виявлення контурів [16] генерується карта складності поверхні (рисунок 1.9, б).

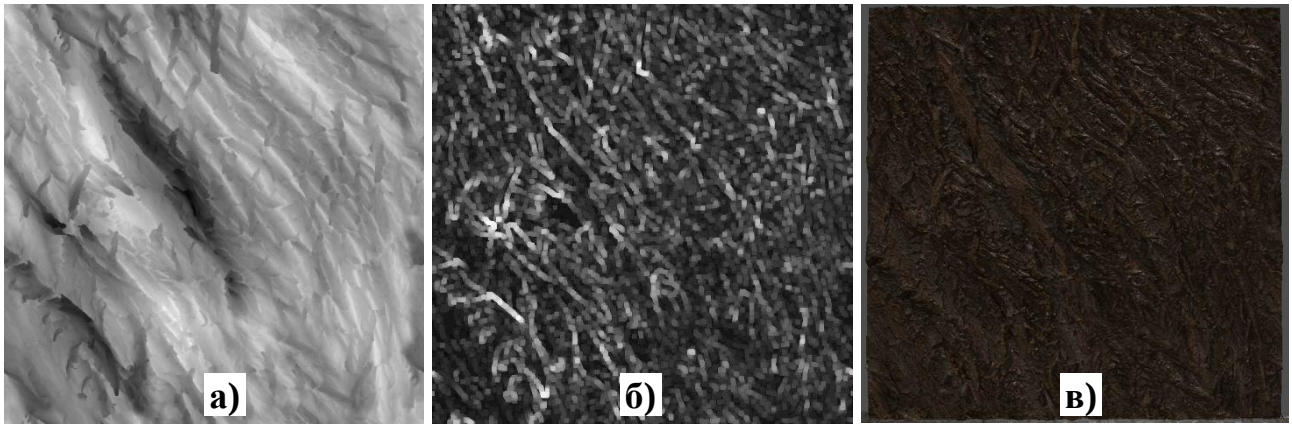


Рисунок 1.9 – POM з використанням аналізу карти висот: а) карта висот; б) карта складності; в) результат рендерингу

Використання оператора виявлення контурів дозволяє визначити градієнт яскравості ділянок зображення, вказуючи напрямок, в якому відбувається приріст від світлої області до темної, а також швидкість цієї зміни. Отримане значення градієнта у контексті карти висот можна вважати складністю зображення в даній точці. Остаточний результат текстурування наведено на рисунку 1.9, в.

Отже, використання карт складності є перспективним напрямком для оптимізації процесу рельєфного текстурування. За рахунок того, що шейдер POM може динамічно збільшувати кількість шарів дискретизації для більш складних ділянок та зменшувати для простіших, області, які потребують високої деталізації, отримують більше обчислювальної уваги, в той час як простіші обробляються більш ефективно.

1.4 Аналіз методів розробки програмного забезпечення для демонстрації та порівняння методів рельєфного текстурування

Для ефективної демонстрації та порівняння методів накладання текстур важливо правильно обрати платформу для розробки. Можливі варіанти включають десктопні, мобільні та веб-застосунки. Кожен підхід має свої переваги та обмеження в тому, що стосується продуктивності, зручності використання та відповідності поставленим вимогам.

Методи рельєфного текстурування вимагають значної обчислювальної потужності для проведення розрахунків, особливо коли мова йде про складніші техніки, такі як parallax occlusion mapping. Через це постає питання ретельної оцінки можливостей кожної платформи для виконання складних завдань з візуалізації без шкоди для продуктивності та досвіду користувача. Порівняльну характеристику платформ для розробки наведено у табл. 1.2.

Таблиця 1.2 – Порівняльна характеристика платформ для розробки

Критерій	**Веб-застосунок**	**Мобільний застосунок**	**Десктопний застосунок**
Продуктивність	Обмежено браузером та пропускною здатністю мережі	Обмежено апаратним забезпеченням мобільних пристроїв	Здатність до виконання складних обчислень
Інтеграція шейдерів	Обмежена підтримка WebGL, просунуті шейдери не підтримуються	Базова підтримка шейдерів	Повна підтримка OpenGL/DirectX та шейдерів
Обробка складних ситуацій	Підходить для простих візуалізацій	Підходить для простіших завдань	Підтримує ресурсомісткий рендеринг
Портативність	Доступний з будь-якого пристрою через браузер	Велика мобільність, але із залежністю від платформи	Залежний від платформи
Простота розповсюдження	Потрібен хостинг та доступ до Інтернету	Потрібне встановлення з пакету чи магазину застосунків	Легко пакується та розповсюджується як виконуваний файл
Час розробки	Швидше, проте з обмеженими можливостями	Середня складність; обмеженість в інструментах	Довший час розробки, але з повною підтримкою функцій
Цільовий варіант використання	Загальна, полегшена візуалізація	Демо-версії, низькопродуктивні програми	Складні, якісні візуальні демонстрації

Мобільні та веб-застосунки мають переваги з точки зору доступності та портативності, проте не відповідають ключовим вимогам для забезпечення можливості відображення та порівняння методів текстурування. Обмежена продуктивність та низький рівень підтримки шейдерів роблять ці платформи непридатними для потреб проєкту. У той же час, розробка програми для настільних ПК дозволяє досягти поставленої мети без будь-яких обмежень завдяки високій продуктивності цільових пристроїв та підтримці таких технологій, як OpenGL та DirectX.

Отже, розробку програмного забезпечення для демонстрації та порівняння методів рельєфного текстурування доцільно реалізувати у вигляді десктопного застосунку.

1.5 Постановка задач дослідження

Проаналізувавши стан питання предметної області, напрямки удосконалення методу parallax occlusion mapping та методи розробки, було визначено ряд задач, які необхідно виконати для розробки ПЗ для демонстрації та порівняння методів рельєфного текстурування:

– провести аналіз стану питання та методів розв’язання задачі;

– здійснити архітектурне проєктування та розробити структуру програмного засобу;

– розробити метод генерації карти складності текстури;

– розробити метод формування зображень рельєфних поверхонь на базі parallax occlusion mapping;

– розробити засіб для генерації карт складності;

– здійснити розробку програмного забезпечення для демонстрації та порівняння методів рельєфного текстурування;

– провести тестування працездатності розробленого ПЗ;

– провести експериментальні дослідження методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping;

– розробити інструкцію користувача.

Технічне завдання на розробку наведено в додатку А.

1.6 Висновки

У розділі було досліджено методи формування зображень рельєфних поверхонь; підтверджено доцільність створення власної реалізації на основі аналізу аналогічних продуктів на ринку; окреслено підходи до удосконалення методу parallax occlusion mapping. Було розглянуто потенційні платформи для розробки ПЗ та встановлено, що десктопне рішення є найбільш придатним для потреб дослідження. Також було здійснено постановку задач, які необхідно виконати для реалізації програмного продукту.

2 РОЗРОБКА МЕТОДІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ФОРМУВАННЯ ЗОБРАЖЕНЬ РЕЛЬЄФНИХ ПОВЕРХОНЬ

2.1 Архітектурне проєктування програмного забезпечення

Архітектурний дизайн дозволяє розробнику визначити структуру програмної системи, її компоненти та те, як вони взаємодіють між собою для досягнення запланованої функціональності. У випадку із застосунком для порівняння методів текстурування, в основі системи лежить використання рушія jMonkeyEngine 3 для 3D-візуалізації та бібліотеки Nifty GUI для створення графічного інтерфейсу для взаємодії із користувачем. Це дозволяє реалізувати у програмі можливість переміщення по тривимірній сцені, перемикаючись між різними методами текстурування та порівнюючи їх між собою.

Архітектура побудована за модульним принципом. Такий підхід забезпечує масштабованість та зручність обслуговування. На рисунку 2.1 подано графічне зображення архітектури розроблюваного застосунку.

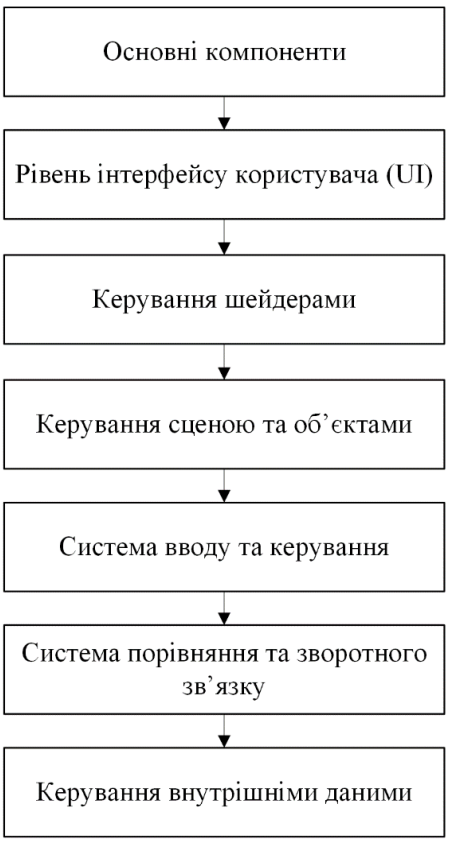


Рисунок 2.1 – Архітектура системи

Нижче наведено огляд ключових складових архітектури:

1. До основних компонентів системи входить ядро програми, яке слугує точкою входу для програми. Воно виконує первинну ініціалізацію середовища jMonkeyEngine і налаштовує сцену, розташування камери, освітлення та початкові текстури. Ядро також керує життєвим циклом програми, відповідаючи за запуск, оновлення зображення та завершення роботи програми.

2. Рівень інтерфейсу користувача включає бібліотеку Nifty GUI та обробку подій. Nifty GUI керує меню, кнопками та іншими елементами інтерфейсу, дозволяючи користувачам вибирати між різними методами текстурування, а також налаштовувати параметри, пов’язані з ними. У свою чергу, обробник подій діє як міст між графічним інтерфейсом та рушієм візуалізації. Він перехоплює вхідні дані від користувача та перетворює їх на відповідні команди для рушія.

3. jMonkeyEngine підтримує два типи шейдерів: фрагментні та вершинні шейдери (рисунок 2.2). Існує декілька часто використовуваних мов для кодування шейдерів, але оскільки JME3 базується на OpenGL, шейдери в JME використовують GLSL [17].

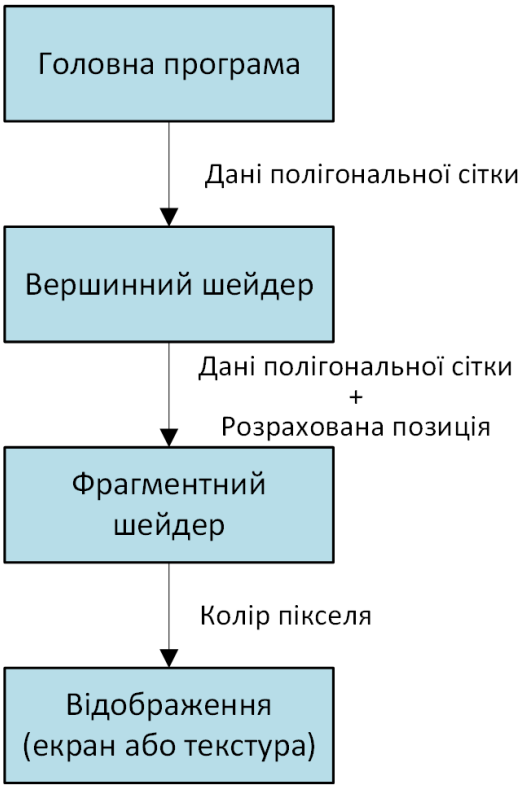


Рисунок 2.2 – Стек виклику шейдерів при текстуруванні

Вершинний шейдер виконується один раз для кожної вершини вигляду, тоді як фрагментний шейдер (також званий піксельним) виконується один раз для кожного пікселя на екрані [18]. Основним призначенням вершинного шейдера є обчислення екранної координати вершини, тобто де вона буде відображена на екрані, тоді як фрагментного шейдера – обчислення кольору пікселя.

4. Сцена у jMonkeyEngine має ієрархічну структуру. Керування об’єктами сцени здійснюється за допомогою графу сцени. Кожна платформа, стіна чи підлога представлені у вигляді вузлів, що дозволяє ефективно керувати ними. На рисунку 2.3 наведено приклад графу сцени.

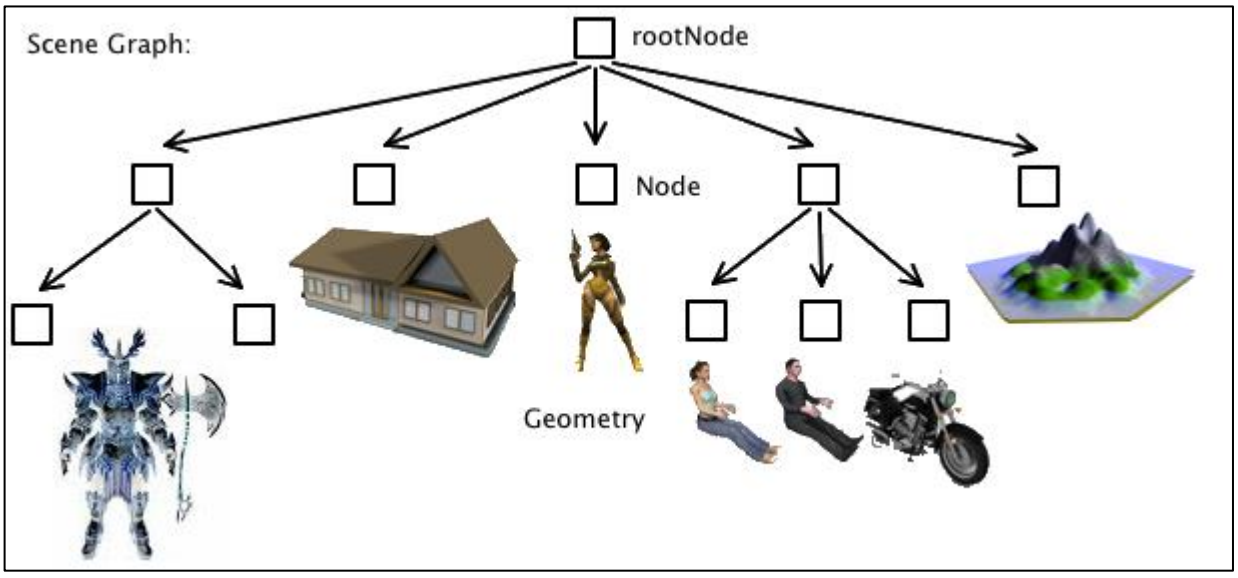


Рисунок 2.3 – Граф сцени

5. Система введення дозволяє користувачеві вільно пересуватися у віртуальному середовищі. Це реалізовано за допомогою системи керування введенням jMonkeyEngine, яка перехоплює введення з клавіатури та миші для керування рухами камери.

6. Система порівняння та зворотного зв’язку включає попередньо визначені методи текстурування, з яких користувач може вибирати. Коли користувач змінює метод, система оновлює сцену в режимі реального часу, дозволяючи йому миттєво бачити результати свого вибору. Це важливо для демонстраційної функції програми.

7. Керування внутрішніми даними включає керування завантаженням та кешуванням текстур, що забезпечує високу продуктивність програми при перемиканні між методами текстурування.

Таким чином, архітектура застосунку для демонстрації методів текстурування фокусується на ефективності, інтерактивності та гнучкості. Система забезпечує користувачеві середовище, де він має змогу досліджувати, порівнювати та отримувати краще розуміння роботи різних методів накладання текстур. Модульний підхід відкриває можливості для оновлення та розширення системи без серйозних змін у решті вихідного коду.

2.2 Розробка методу генерації карти складності текстури

При використанні методу POM якість відображення залежить від балансу між візуальною точністю та обчислювальною складністю. Ця техніка вимагає дискретизації вздовж вектора огляду, що передбачає обчислення декількох значень з карти висот для кожного текселя. Що більше шарів дискретизації, то більш точна оцінка значень та ефекти оклюзії (перекриття одних деталей текстури іншими), але це відбувається за рахунок збільшення кількості обчислень. З іншого боку, при зменшенні кількості шарів знижується якість, що стає причиною виникнення артефактів, таких як ступінчастість, розмитість чи аліасинг, особливо при гострих кутах огляду. Саме тут стає в нагоді використання карти складності.

Пропонований метод полягає в аналізі зміни значень висоти вздовж поверхні текстури та регулюванні кількості шарів дискретизації для POM на основі локальних деталей текстури. Наприклад, у регіонах, де є значні зміни висоти й з’являється ризик виникнення ступінчастості, буде застосовано більше шарів. І навпаки, на більш гладких ділянках можна використати менше шарів без погіршення якості візуалізації. Такий підхід дозволяє зберегти високу якість візуалізації без залучення надлишкових ресурсів.

Основна ідея створення карти складності полягає у визначенні ділянок з високою та низькою деталізацією шляхом аналізу градієнтів карти висот. Для

цього можна застосувати алгоритми для виявлення контурів. Вони також дозволяють виявляти у зображенні ділянки із високими змінами в інтенсивності, а у випадку з картами складності – у різкості зміни висоти деталей на карті висот. Згодом ці дані можуть бути використані для оцінки складності ділянок текстури.

2.2.1 Обґрунтування вибору оператора виявлення контурів

Серед найпоширеніших методів для виявлення контурів можна виділити оператор Собеля, оператор Прюітт, оператор Шарра та оператор Кенні [19]. У табл. 2.1 наведено порівняння цих технік на основі різних критеріїв.

Таблиця 2.1 – Порівняння операторів виявлення контурів

Критерій	**Оператор Собеля**	**Оператор Прюітт**	**Оператор Шарра**	**Оператор Кенні**
Стійкість до шумів	+	–	+	+
Обчислювальна ефективність	+	+	–	–
Згладжування країв	+	–	+	+
Чутливість до градієнтів	+	–	+	+
Всього	**4**	**1**	**3**	**3**

Оператор Прюітт є ефективним з точки зору обчислень, проте він більш чутливий до шуму і менш точний у виявленні незначних варіацій поверхні. Попри ефективність цього методу, його низька чутливість робить його менш придатним для створення карт складності.

Оператор Шарра забезпечує більшу точність у визначенні тонких градієнтів. Однак така підвищена чутливість не дає значних переваг при створенні карт складності, оскільки дрібні, менш релевантні деталі можуть бути надмірно підкреслені.

Оператор Кенні має високу чутливість до градієнтів і виявляє краї з відмінною точністю. Однак він гірше справляється поступовими змінами

градієнтів, що більш характерно для текстур. Це робить його непридатним для генерації карт складності.

Оператор Собеля дозволяє досягти оптимального балансу між завадостійкістю, обчислювальною ефективністю та градієнтною чутливістю. Крім того, він ефективно фіксує тонкі зміни висоти без накладних витрат більш просунутих алгоритмів, таких як оператор Кенні чи Шарра.

Таким чином, використання оператора Собеля для обчислення наближення градієнтів в напрямках X та Y та створення карт складності є оптимальним вибором. Отримані значення дозволять отримати уявлення про те, як швидко змінюється висота в різних регіонах текстури. Величину цих градієнтів можна вважати показником складності поверхні.

2.2.2 Математичне формулювання методу

Нехай H(x,y) відображає піксель з карти висот з координатами (x, y). Часткові похідні по осях X та Y обчислюються за допомогою оператора Собеля [20] та означають наближення градієнта G_x, G_y у відповідних напрямках:

$$G_x = \frac{\partial H}{\partial x} \approx \sum_{i=-1}^{1} \sum_{j=-1}^{1} H(x+i, y+j) * S_x(i,j)\,,$$

$$G_y = \frac{\partial H}{\partial y} \approx \sum_{i=-1}^{1} \sum_{j=-1}^{1} H(x+i, y+j) * S_y(i,j)\,,$$

де S_x, S_y – ядра Собеля для напрямків X та Y відповідно:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix},\ S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Величина градієнта $G(x, y)$ для кожного пікселя обчислюється як:

$$G(x,y) = \sqrt{G_x^2 + G_y^2} \ .$$

Отримане значення показує швидкість зміни висоти текстури, яка безпосередньо відповідає складності поверхні в цій точці. Оскільки значення можуть варіюватися в широких межах, нормалізуємо їх до діапазону [0; 1], отримавши $C(x,y)$:

$$C(x,y) = \frac{G(x,y)}{\max(G)}.$$

Цей крок гарантує, що значення складності вписуються в єдину шкалу. Крім цього, було застосовано розширення з використанням ядра 5x5. Цей процес дозволяє поширити високі значення на сусідні пікселі, щоб підкреслити області з високою складністю.

Операція розширення полягає у згортанні зображення C з деяким ядром B, яке може мати будь-яку форму або розмір, зазвичай це квадрат або коло [21]. Ядро має опорну точку, що лежить в центрі матриці. Коли ядро B сканує зображення, обчислюється максимальне значення пікселя, що перекривається B, і піксель зображення в позиції опорної точки замінюється на це максимальне значення. Математично розширення $C'(x,y)$ можна представити формулою:

$$C'(x,y) = \max_{(i,j)\in B} C(x+i-a, y+j-b) \ ,$$

де a, b – зсуви, які центрують ядро B на поточному пікселі (x, y).

Також було застосовано гамма-корекцію на деяке значення γ. Це нелінійне перетворення $C''(x,y)$ дозволяє підкреслити менші значення складності, щоб незначні зміни висоти стали помітнішими, та обчислюється за формулою:

$$C''(x,y) = C'(x,y)^{\gamma} \ .$$

Для використання у шейдері POM до отриманих значень було застосовано квантування на дискретні рівні. Це дозволяє згрупувати подібні за складністю області з метою уникнення зайвих коливань значень складності. Приведення значень до найближчого рівня $L(x, y)$ відбувається за формулою:

$$L(x,y) = round\left(\frac{C''(x,y)}{\Delta}\right) * \Delta, \qquad \Delta = \frac{1}{N-1},$$

де N – кількість рівнів квантування.

На рисунку 2.4 подано блок-схему алгоритму генерації карти складності із використанням запропонованого методу.

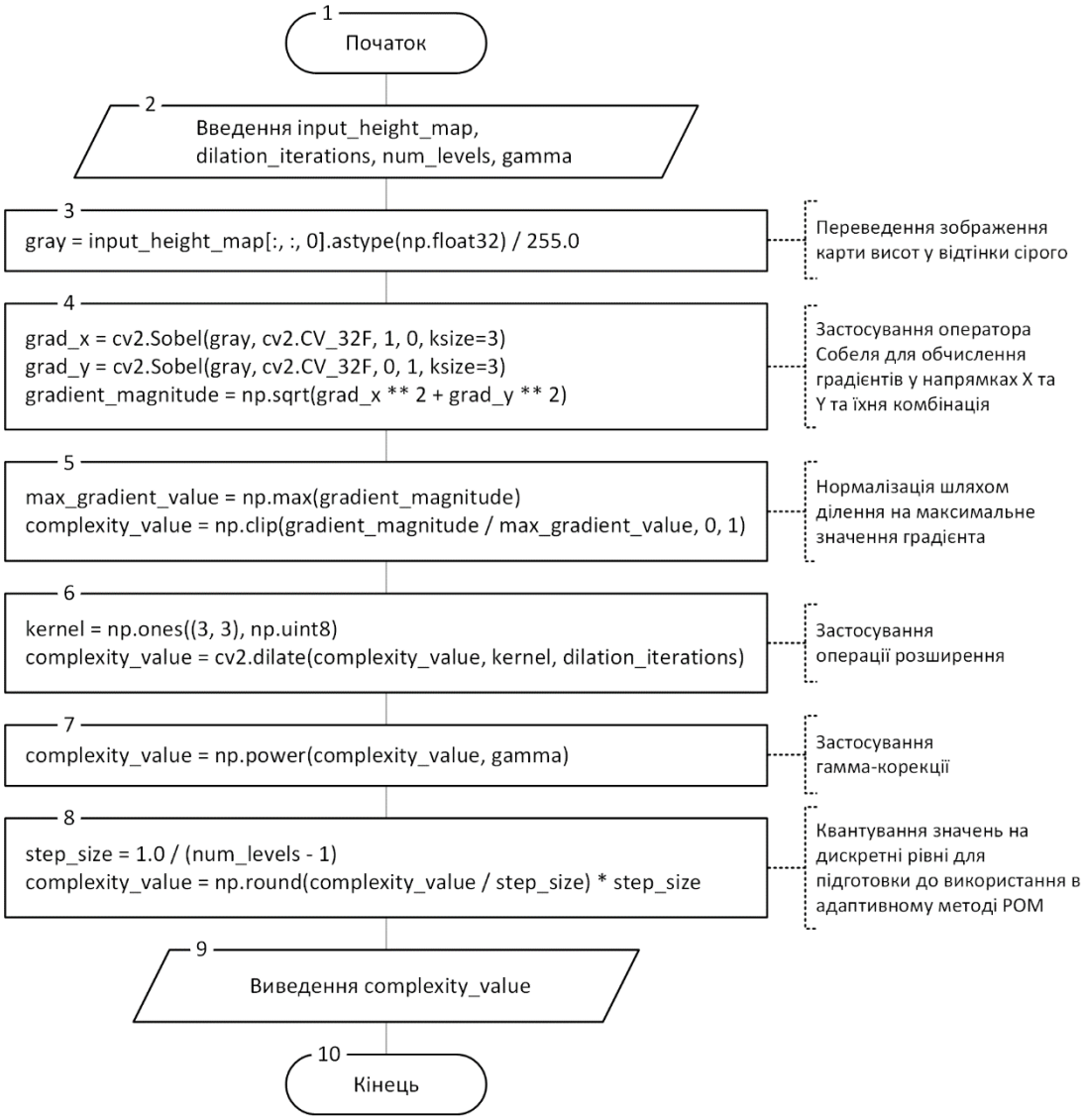


Рисунок 2.4 – Блок-схема алгоритму генерації карти складності

Виконання алгоритму розпочинається з підготовки вхідних даних – переведення зображення карти висот у відтінки сірого. Після цього до зображення застосовується оператор Собеля для обчислення градієнтів у напрямках X та Y та виконується їхня комбінація, в результаті чого отримується величина градієнта, яка відображає, наскільки сильно змінюється висота в кожній точці, тобто складність текстури. Ці значення нормалізуються шляхом ділення на максимальне значення градієнта, щоб усі значення знаходилися в діапазоні від 0 до 1. Після цього до отриманого зображення послідовно застосовується операція розширення та гамма-корекція для підкреслення деталей текстури, а також квантування значень на дискретні рівні для підготовки до використання в адаптивному методі POM.

Отже, запропонований метод дозволяє оцінити складність поверхні текстури на основі обчислення градієнтів за допомогою оператора Собеля. Згенерована карта складності слугує вхідним параметром для покращеного методу POM та гарантує, що для більш складних областей буде застосовано більше шарів дискретизації під час текстурування.

2.3 Розробка методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping

Традиційна реалізація parallax occlusion mapping, хоч і ефективна для імітації деталей поверхні, використовує статичну кількість шарів дискретизації діапазону глибин текстури, що у деяких випадках може призводити до неефективної роботи шейдера та надлишкових обчислень. Для вирішення цієї проблеми пропонується метод, який дозволяє динамічно підбирати кількість шарів для POM на основі карти складності та кута огляду. Цей підхід дозволить оптимізувати продуктивність та точність текстури, використовуючи більше шарів там, де це необхідно, і менше – на простіших ділянках.

Ключовою складовою покращеного методу є використання карти складності під час накладання текстури. Це дозволяє більш ефективно розподіляти обчислювальні ресурси, зменшуючи середню кількість шарів

дискретизації по текстурі та зберігаючи при цьому якість ділянок, де деталізація є найбільш важливою.

Крім цього, кількість шарів регулюється залежно від розташування спостерігача відносно поверхні. Оскільки POM використовує зміщення координат текстури, при гострих кутах огляду ефект паралаксу (зміщення) стає більш вираженим, що призводить до виникнення спотворень [22]. Щоб зберегти ілюзію глибини та уникнути артефактів, пропонується використовувати більшу кількість шарів для гостріших кутів огляду. Та навпаки, у випадках, коли спостерігач дивиться на поверхню прямо й ефект паралаксу мінімальний, для досягнення ефекту глибини можна застосувати меншу частоту дискретизації.

Реалізація запропонованого методу передбачає інтеграцію використання карти складності та кутової залежності у шейдер POM. Це вимагає таких кроків:

1. Обчислення фактору кута огляду.

Кут між напрямком погляду спостерігача та вектором нормалі до поверхні (приймається як (0, 0, 1)) визначає, наскільки сильним є ефект паралаксу текстури. Цей вплив виражається через кутовий фактор F:

$$F = (1 - |\bar{V} * (0, 0, 1)|)^{0.8} ,$$

де $\bar{V} = (V_x, V_y, V_z)$ – вектор напрямку погляду спостерігача в дотичному просторі.

Скалярний добуток вектору нормалі та напрямку погляду спостерігача дорівнює косинусу кута між ними та знаходиться в межах [-1;1], а використання модуля гарантує, що отримане значення буде додатним. На цьому етапі, чим ближче напрям погляду до нормалі поверхні, тим ближче значення до 1,0. Але оскільки нас цікавлять гострі кути (коли погляд майже паралельний поверхні), необхідно відняти отримане значення від одиниці. Таким чином, більші значення будуть отримані при гостріших кутах. Додаткове піднесення до степеню 0,8 допомагає коригувати криву залежності, роблячи розподіл більш плавним.

2. Визначення кількості шарів дискретизації.

Кількість шарів дискретизації залежить як від напрямку погляду

спостерігача, так і від складності поверхні. Фактор кута огляду визначає діапазон, в межах якого буде обрано конкретне значення залежно від складності текстури. Поєднання карти складності та врахування кута огляду дозволяє регулювати частоту дискретизації у двох вимірах, що забезпечує оптимальне відображення текстури незалежно від кута огляду.

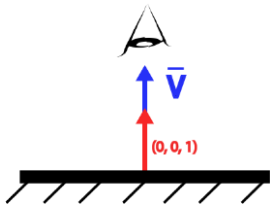
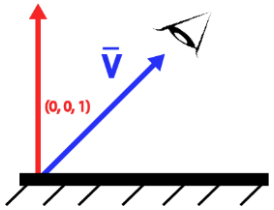
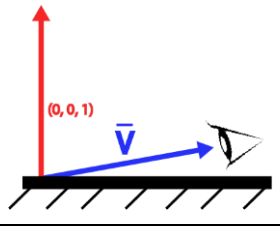
Нехай базовий діапазон можливих значень кількості шарів знаходиться в проміжку $N' \in [4, 32]$. На основі цього за допомогою лінійної інтерполяції визначається мінімальна та максимальна кількість шарів $N_{T\,min}, N_{T\,max}$ у текселі T з урахуванням обчисленого фактору кута огляду:

$$N_{T\,min} = N'_{min} + \left(\frac{N'_{max}}{2} - N'_{min}\right) * F\ ,$$

$$N_{T\,max} = N'_{min} + (N'_{max} - N'_{min}) * F\ .$$

У табл. 2.2 наведено приклади зміщення діапазону можливих значень кількості шарів дискретизації залежно від розташування спостерігача.

Таблиця 2.2 – Залежність кількості шарів від фактору кута огляду

Кут огляду	**F**	**$N_{T\,min}$**	**$N_{T\,max}$**	**Ілюстрація**
0°	0	4	4	
45°	0,3744	8	14	
80°	0,8585	14	28	

Остаточна кількість шарів дискретизації N_T в межах [$N_{T\ min}$; $N_{T\ max}$] на основі значення складності обчислюється за формулою:

$$N_T = N_{T\,min} + (N_{T\,max} - N_{T\,min}) * C_T ,$$

де $C_T \in [0; 1]$ – значення складності текстури у текселі T, що отримується з карти складності.

На рисунку 2.5 наведено приклад залежності кількості шарів дискретизації від складності текстури.

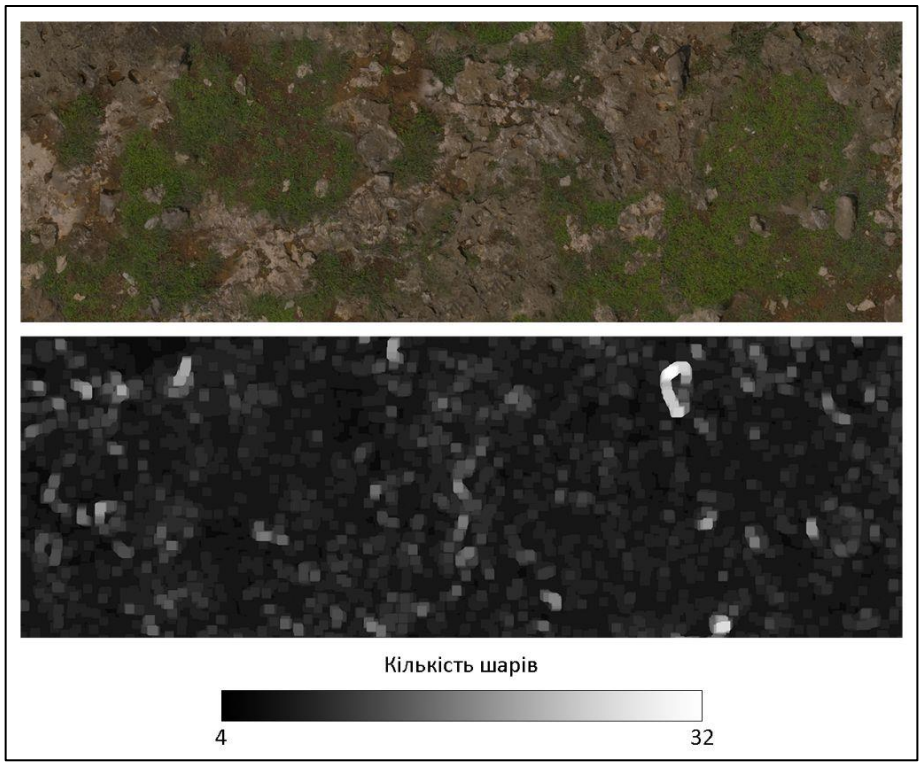


Рисунок 2.5 – Залежність кількості шарів дискретизації від складності текстури

Розглянемо детальніше алгоритм базового методу parallax occlusion mapping [23] для забезпечення основи для розуміння покращень, внесених запропонованим адаптивним підходом.

Після визначення кількості шарів координати текселя T=(u,v) поступово зміщуються вздовж напрямку паралаксу, який обчислюється за формулою:

$$\bar{P} = \frac{(V_x, V_y)}{V_z} * S \,,$$

де $\bar{P}$ – вектор зміщення паралаксу;

(V_x, V_y, V_z) – компоненти вектора напрямку погляду у дотичному просторі;

S – коефіцієнт масштабування, який визначає, наскільки сильно зміщується карта висот.

Вектор $\overline{\mathrm{P}}$ описує, наскільки потрібно змістити координати текстури для імітації ефекту паралакса. Поділивши цей вектор на загальну кількість шарів дискретизації N, отримуємо приріст зміщення ΔT на один шар:

$$\Delta T = \frac{\bar{P}}{N} \,.$$

Кожен крок вздовж вектора зміщення відповідає одному шару текстури, з приростом глибини ΔD на один шар, що обчислюється за формулою:

$$\Delta D = \frac{1}{N} \,.$$

Після цього розпочинається трасування променя вздовж вектора $\overline{\mathrm{P}}$ для пошуку перетину з текстурою. На кожній ітерації перевіряється, чи перевищує поточна висота шару значення з карти висот. На кожному кроці координати текселя та поточна висота пошуку оновлюються як:

$$T' \leftarrow T' - \Delta T \,, \qquad D' \leftarrow D' + \Delta D \,,$$

де T' – поточні координати зміщеного текселя;

D' – поточна висота, на якій перевіряється перетин променя з текстурою.

Цей цикл відбувається до тих пір, поки $\mathrm{D}' < \mathrm{H}_{\mathrm{T}'}$, де $\mathrm{H}_{\mathrm{T}'}$ – значення висоти текстури у зміщеному текселі T', що отримується з карти висот при кожній

ітерації. Коли ж поточна накопичена висота D' перевищить або буде рівною поточному значенню висоти з карти висот $H_{T'}$, це буде свідчити про перетин пошукового променя з текстурою й трасування буде завершено.

Після цього виконується лінійна інтерполяція між координатами текселя, отриманими на поточному етапі, та координатами з попереднього шару до перетину з текстурою. Це допомагає згладити перехід між сусідніми текселями та уникнути появи візуальних артефактів.

Ваговий коефіцієнт w для інтерполяції задано формулою:

$$w = \frac{H_{T'} - D'}{H_{T'+\Delta T} - D' + \Delta D} \ ,$$

де $H_{T'}$ – значення висоти текстури у текселі після перетину;

$H_{T'+\Delta T}$ – значення висоти текстури у текселі до перетину;

$(T' + \Delta T)$ – координати зміщеного текселя з попереднього шару.

Отримане відношення визначає, який вплив матиме кожна з двох точок на остаточні координати текселя.

Лінійна інтерполяція координат для визначення остаточних координат текселя для відображення проводиться за формулою:

$$T'' = w * (T' + \Delta T) + (1 - w) * T' \ ,$$

де T'' – остаточні координати текселя.

Цей крок гарантує, що перехід між шарами буде плавним, без помітних стрибків або артефактів. Пізніше за отриманими координатами зі зміщенням здійснюється накладання текстури, що дозволяє досягти ілюзії глибини текстури.

Блок-схему алгоритму накладання текстури із використанням запропонованого методу наведено на рисунку 2.6.

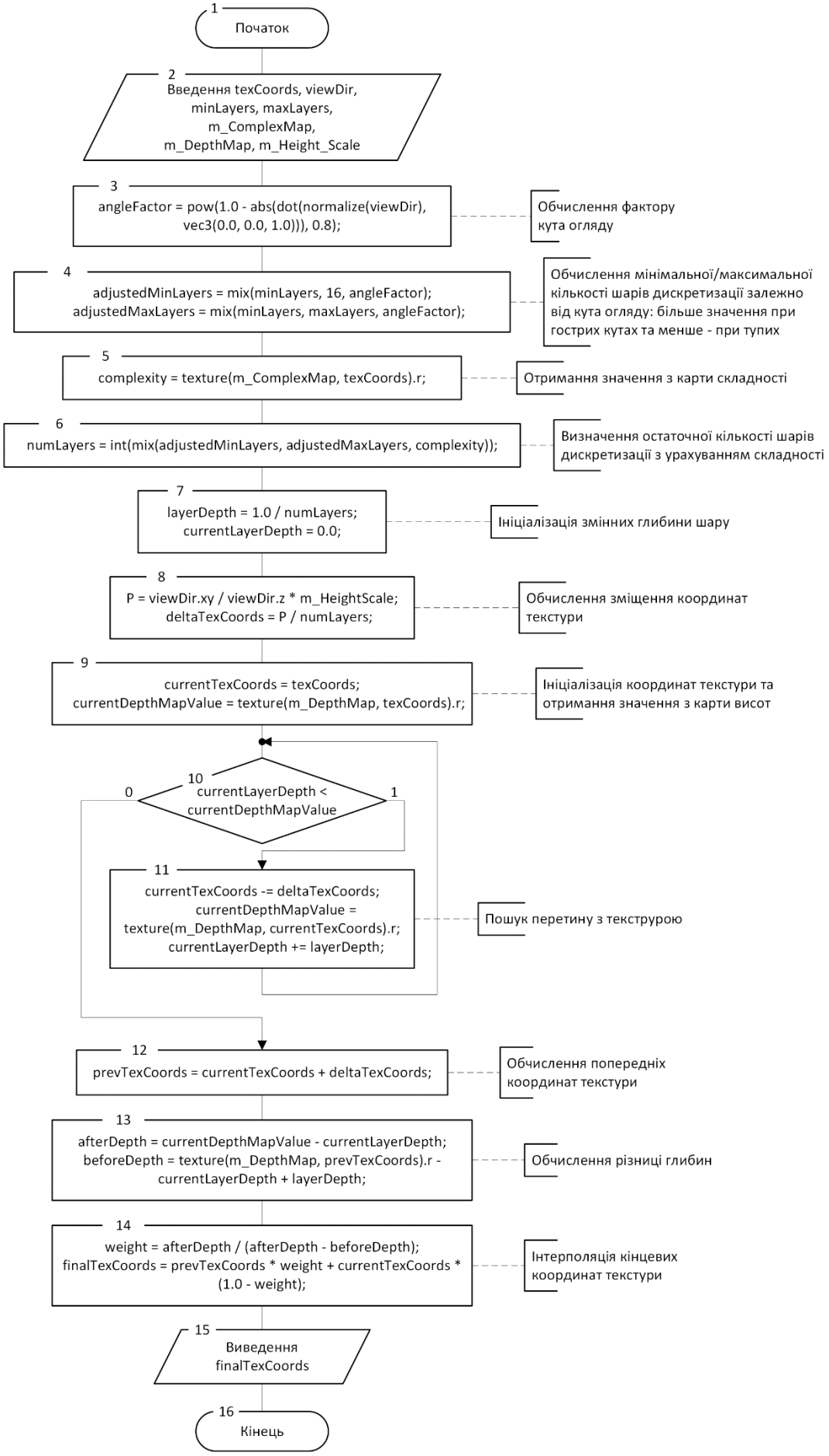


Рисунок 2.6 – Блок-схема алгоритму накладання текстури

Таким чином, вдосконалений метод на базі POM дозволяє підвищити продуктивність процесу рендерингу текстури за допомогою регулювання кількості шарів дискретизації залежно від складності поверхні та кута огляду. Таке покращення дозволяє підвищити швидкодію програм, що використовують POM, особливо в сценаріях, коли необхідно забезпечити високий рівень деталізації при обмежених обчислювальних ресурсах.

2.4 Розробка структурних схем застосунку

Уніфікована мова моделювання (UML) – це широко використовуваний стандарт для моделювання та візуалізації структури і поведінки програмних систем. Вона надає повний набір типів діаграм, кожна з яких спрямована на ілюстрацію певних аспектів системи, від високорівневої архітектури до детальних взаємодій. UML відіграє важливу роль у процесі розробки програмного забезпечення, пропонуючи візуальний спосіб представлення складних взаємозв’язків, робочих процесів і компонентів.

Однією з головних переваг UML є її здатність представляти як статичні, так і динамічні аспекти системи. Статичні елементи включають діаграми класів, які показують взаємозв’язки між класами, об’єктами та модулями, і діаграми компонентів, які описують, як організовані різні частини системи. Динамічні аспекти, з іншого боку, відображаються за допомогою діаграм, таких як діаграми послідовностей, діаграми діяльності та діаграми станів. Ці представлення корисні для ілюстрації поведінки системи у відповідь на різні події або вхідні дані, що стає вирішальним для демонстрації робочих процесів програмного забезпечення та логіки управління.

UML-діаграми корисні не лише на етапі планування, але й протягом усього життєвого циклу розробки. Вони слугують планами, які керують процесом реалізації та допомагають розробникам зрозуміти, як взаємодіють різні модулі. Крім того, UML-діаграми полегшують супровід системи, слугуючи документацією, коли система розвивається з часом і новим членам команди потрібно розуміти кодову базу.

Для демонстрації логіки роботи застосунку для демонстрації та порівняння методів рельєфного текстурування було створено діаграму діяльності (рисунок 2.7). Ця діаграма представляє дії, які може виконувати користувач, а також ілюструє, як програма реагує на різні натискання клавіш. Діаграми діяльності добре підходять для цього типу представлення, оскільки вони зосереджуються на потоці дій і рішень, полегшуючи розуміння того, як користувач взаємодіє з системою і як застосунок реагує на цю взаємодію.

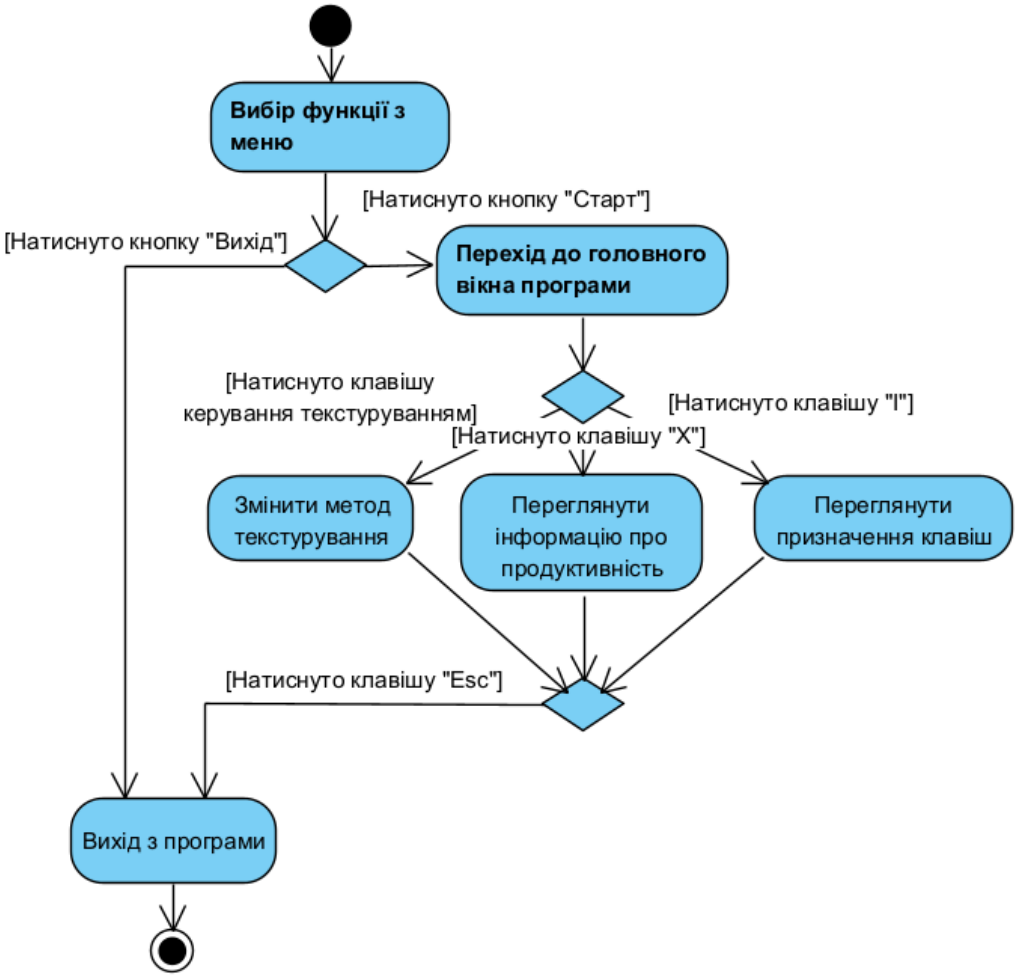


Рисунок 2.7 – Діаграма діяльності

Діаграма демонструє потік керування з моменту запуску програми. Спочатку користувачеві доступне меню, за допомогою якого він може розпочати роботу з програмою чи вийти. Опинившись на головному екрані, користувач може натискати різні клавіші, щоб перемикатися між попередньо визначеними методами текстурування, переглядати інформацію про продуктивність або

підказку щодо функціональних клавіш. Якщо користувач натискає клавішу «Esc» у будь-який момент, програма завершує роботу.

Важливо більш детально вивчити процес перемикання методів текстурування, оскільки він безпосередньо впливає на продуктивність і зручність роботи користувача. Тому було побудовано діаграму послідовності (рисунок 2.8), яка відображає ключові події, що мають місце під час застосування нового методу текстурування.

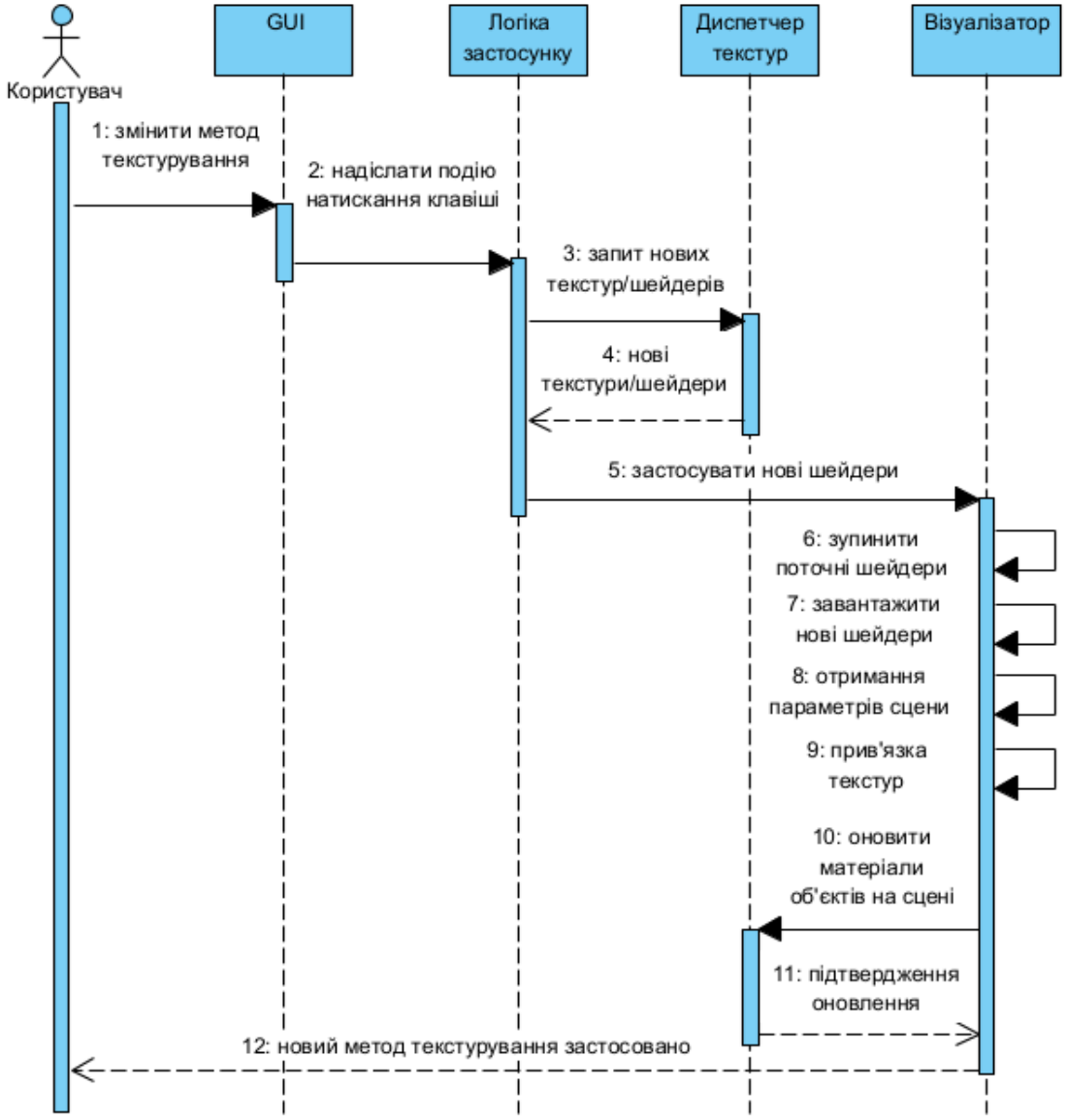


Рисунок 2.8 – Діаграма послідовності

Кожне повідомлення на діаграмі відображає важливий крок у забезпеченні належного завантаження, застосування та рендерингу нового методу

текстурування. Нижче наведено опис послідовності повідомлень, що надсилаються системою:

1. Користувач запускає процес натисканням заздалегідь визначеної клавіші, пов’язаної з певним методом текстурування. Цей ввід фіксується графічним інтерфейсом користувача.

2. Контролер графічного інтерфейсу надсилає подію натискання клавіші до керуючого класу, який обробляє вхідні дані і визначає, які дії потрібно виконати.

3. На основі події натискання клавіші надсилається повідомлення до менеджера текстур, запитуючи необхідні текстури та шейдери для обраного методу текстурування.

4. Менеджер текстур отримує ресурси і надсилає їх назад до керуючого класу для подальшої обробки.

5. Надсилається повідомлення візуалізатору застосувати нові шейдери і відповідно оновити параметри сцени.

6-9. Візуалізатор зупиняє виконання поточних активних шейдерів, щоб забезпечити перехід до нових; виконується завантаження текстур та шейдерів; отримує поточні параметри сцени, необхідні для застосування нового методу текстурування; виконує прив’язку текстур до відповідних об’єктів у сцені, щоб підготувати їх до візуалізації.

10. Матеріали, застосовані до об’єктів на сцені, оновлюються відповідно до нового методу текстурування.

11. Після повного оновлення сцени, диспетчер текстур надсилає візуалізатору повідомлення з підтвердженням, що новий метод текстурування успішно застосовано.

12. Користувач бачить результат зміни методу текстурування.

Таким чином, наведена діаграма послідовності дає чітке уявлення про послідовність операцій, які гарантують, що рушій візуалізації застосовує методи текстурування ефективно і без збоїв.

Для планування майбутніх розширень, налагодження проблем, пов’язаних із залежностями між модулями, та забезпечення тісної інтеграції між різними

частинами системи було побудовано діаграму компонентів, зображену на рисунку 2.9. Така діаграма надає високорівневе уявлення про архітектуру системи, зосереджуючись на тому, як різні програмні компоненти взаємодіють і залежать один від одного. Вона візуалізує структурні зв'язки між різними модулями, бібліотеками та зовнішніми системами, допомагаючи розробникам та зацікавленим сторонам зрозуміти організацію кодової бази.

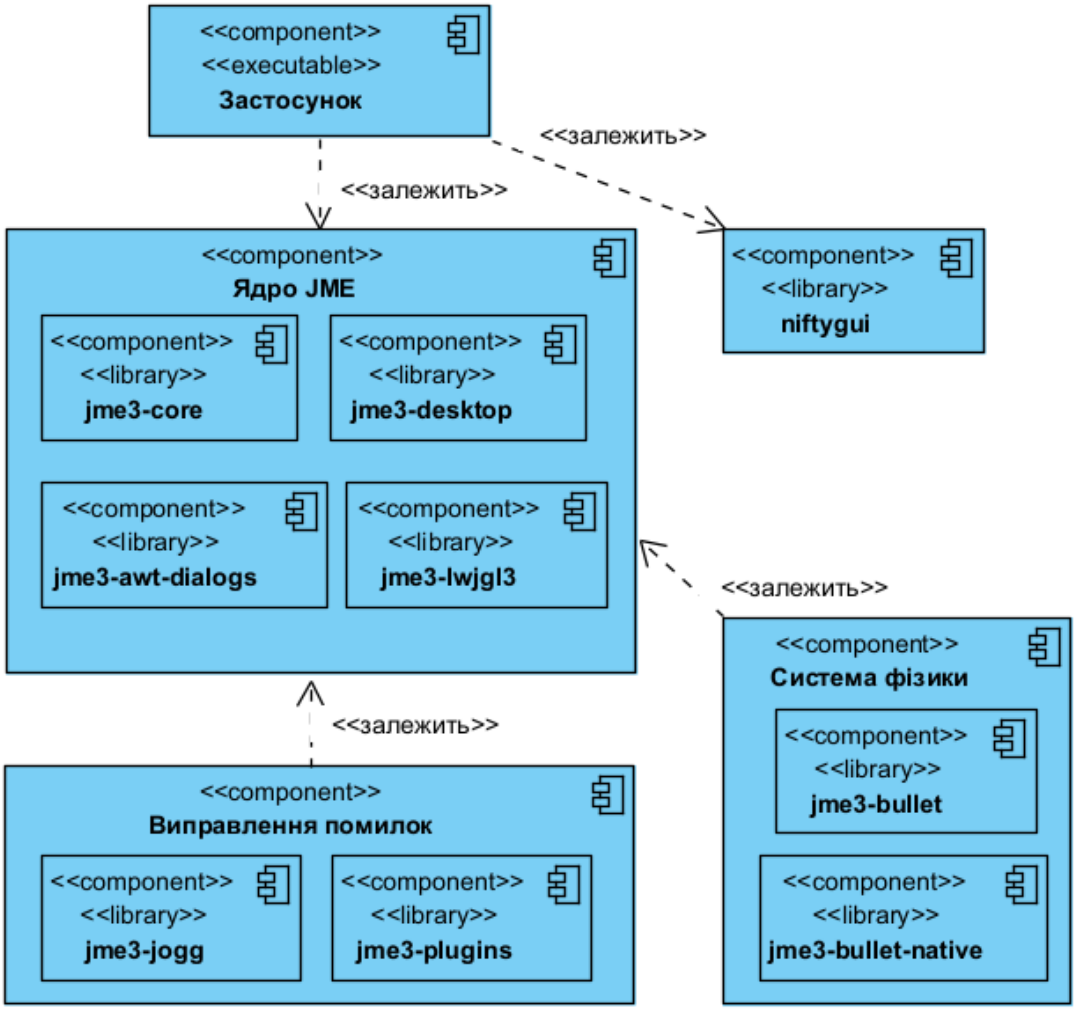


Рисунок 2.9 – Діаграма компонентів

Як бачимо, застосунок слугує виконуваним компонентом верхнього рівня, що покладається на декілька бібліотек та систем для функціонування. В основі лежить рушій jMonkeyEngine 3 (JME), який інтегрує декілька бібліотек, таких як jme3-core та jme3-desktop. Вони необхідні для базового рендерингу та специфічної для операційної системи функціональності. Інші допоміжні

бібліотеки, такі як jme3-awt-dialogs та jme3-lwjgl3, ще більше розширюють можливості рушія, надаючи специфічну для платформи підтримку вікон та інтеграції OpenGL.

Бібліотека Nifty GUI надає графічні елементи для керування програмою. Оскільки jMonkeyEngine не має розширених можливостей графічного інтерфейсу, Nifty GUI відіграє важливу роль у заповненні цієї прогалини.

Окремий компонент «Система фізики» представляє бібліотеки jme3-bullet та jme3-bullet-native, необхідні для фізичного моделювання. Вони забезпечують логіку виявлення зіткнень та динаміки об’єктів.

Компонент «Виправлення помилок» покладається на такі бібліотеки, як jme3-jogg для декодування звуку та jme3-plugins для розширеної функціональності. Це гарантує, що основний рушій і застосунок залишатимуться стабільними і функціональними, обробляючи специфічні ресурси та інші потенційні проблеми.

Отже, проєктування структури застосунку має важливе значення для забезпечення його функціональності та масштабованості. Побудовані діаграми дають цінну інформацію про структуру й поведінку програми, що слугує основою для ефективної розробки.

2.5 Розробка графічного інтерфейсу користувача

Графічний інтерфейс користувача (GUI) є важливим елементом будь-якого інтерактивного програмного забезпечення. Він дозволяє користувачам керувати системою за допомогою візуальних компонентів, таких як кнопки, меню та піктограми. Мета інтерфейсу користувача полягає у покращенні досвіду користувача та забезпеченні безперешкодної взаємодії між користувачами та цифровими продуктами чи послугами [24]. Добре продуманий UI може підвищити залученість користувачів та збільшити конверсію.

Для досягнення цієї мети існують принципи ефективного дизайну інтерфейсу, яких слід дотримуватися. Ясність і послідовність елементів інтерфейсу допомагають користувачам зрозуміти, як взаємодіяти з інтерфейсом,

а юзабіліті і доступність гарантують, що дизайн буде доступний для всіх користувачів.

Успішний інтерфейс також включає візуальний дизайн і макет, який є естетично привабливим, а також інтерактивні компоненти і функціональність, які залучають користувачів. Ці елементи працюють разом, щоб створити інтерфейс, який є одночасно візуально привабливим і функціональним.

З огляду на це, інтерфейс застосунку, що розробляється, повинен відповідати таким критеріям:

1. Графічний інтерфейс повинен чітко передавати навчальну мету програми. Кожна частина інтерфейсу повинна сприяти ознайомленню користувача із методами текстурування.

2. Усі екрани та елементи (кнопки, написи, інформаційні панелі) повинні відповідати єдиній візуальній темі та структурі. Шрифти, кольори та інтервали повинні застосовуватися послідовно по всьому інтерфейсу, щоб підтримувати узгодженість і передбачуваність.

3. Інтерфейс повинен бути зрозумілим для нових користувачів без попереднього досвіду чи навчання. Підказки, прості текстові описи та інтуїтивна навігація мають вирішальне значення для швидкого освоєння.

Структуру графічного інтерфейсу застосунку для демонстрації та порівняння методів текстурування було розроблено із дотриманням викладених вимог. Вона включає два основні компоненти: меню та головний екран.

Меню надає користувачеві основні можливості навігації та слугує для нього точкою входу у програму (рисунок 2.10). Розроблене меню має мінімалістичний дизайн, пропонуючи лише необхідні функції для початку взаємодії з програмним забезпеченням.

Кнопка «Старт» виконує перехід на головний екран, де користувач отримує доступ до основного функціоналу застосунку. Кнопка «Налаштування» дозволяє вносити зміни у параметри методів текстурування, наприклад, змінювати базову кількість шарів дискретизації для більш просунутих технік, а кнопка «Вийти» закриває програму.

Рисунок 2.10 – Структура меню застосунку

Опинившись на головному екрані (рисунок 2.11), користувач може взаємодіяти з тривимірною сценою, де до її об’єктів застосовано один із методів рельєфного текстурування.

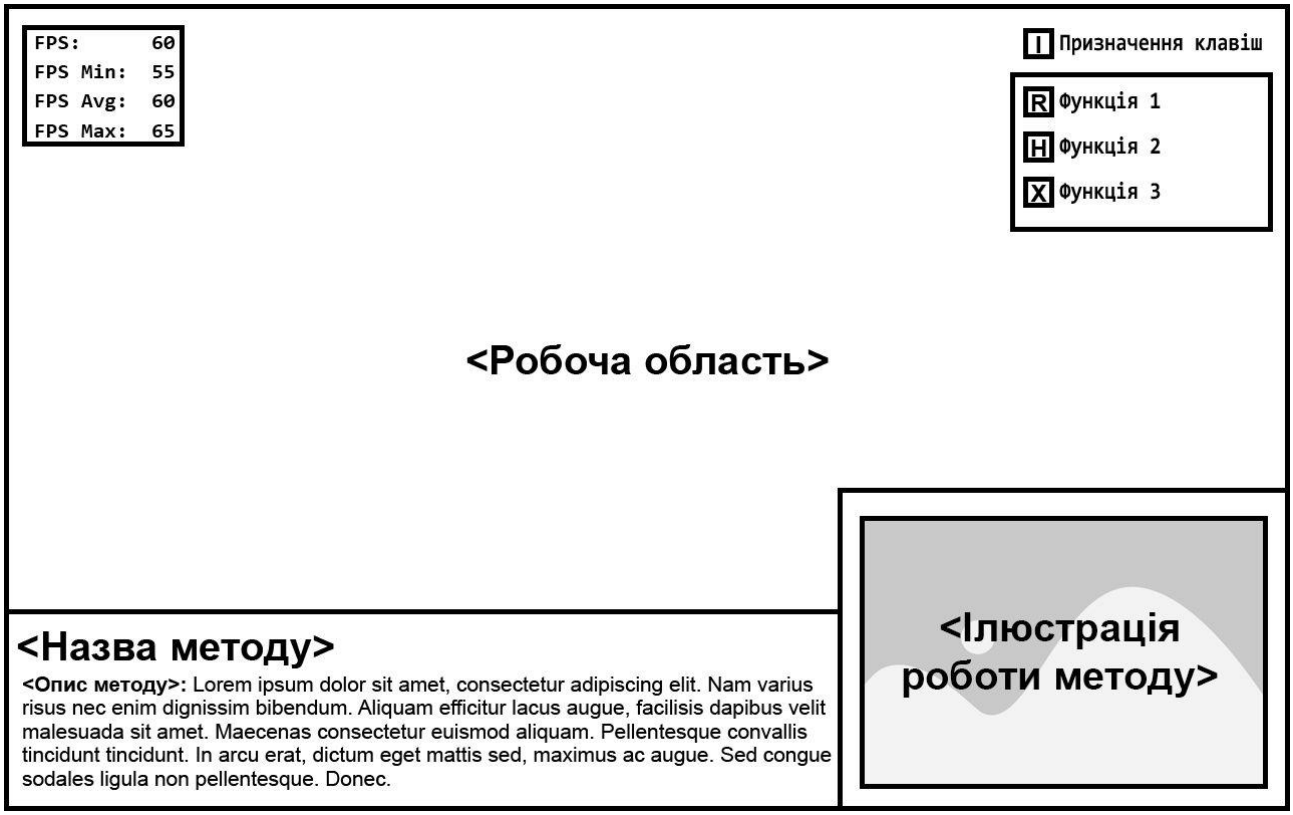


Рисунок 2.11 – Структура головного екрану застосунку

Головний екран містить такі основні компоненти:

1. Інформаційна панель у нижній частині екрана, що містить назву та короткий опис поточного методу текстурування. Цей елемент гарантує, що користувачі можуть отримати теоретичні знання поряд з візуальними уявленнями.

2. Інформаційний розділ у верхньому правому куті забезпечує швидкий доступ до інформації про клавіші керування застосунком. Це дозволить покращити доступність та швидкість навчання користувача.

3. Лічильники FPS у лівому верхньому куті для моніторингу продуктивності. Ця функція допомагає користувачам зрозуміти вплив кожного методу текстурування на продуктивність.

Отже, графічний інтерфейс застосунку розроблений відповідно до окреслених вимог, забезпечуючи чіткий та інтуїтивно зрозумілий досвід користувача. Завдяки структурі, що складається з меню та головного екрану, застосунок має просту навігацію, дозволяє в повній мірі взаємодіяти з його основною функціональністю.

2.6 Висновки

У розділі було представлено метод генерації карти складності на основі обчислення градієнтів за допомогою оператора Собеля. Також було запропоновано метод формування зображень рельєфних поверхонь на базі parallax occlusion mapping, який дозволяє оптимізувати процес рендерингу текстури за допомогою регулювання кількості шарів дискретизації на основі карти складності та кута огляду. Крім цього, було здійснено детальне архітектурне проєктування програмного засобу з використанням UML-діаграм та розроблено графічний інтерфейс користувача застосунку.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного продукту

Процес вибору правильних інструментів для розробки програмного продукту – це крок, який вимагає врахування різних факторів. Від вибору рушія та середовища розробки залежить функціональність, продуктивність та гнучкість проєкту. Це також впливає на ефективність процесу розробки, оскільки різні інструменти мають унікальні функції, робочі процеси та рівні складності.

3.1.1 Порівняльний аналіз рушіїв візуалізації

Для запланованої розробки було проаналізовано кілька рушіїв візуалізації, кожен з яких пропонує різні функції та можливості. Основна увага приділялася рушіям, здатним до 3D-рендерингу в реальному часі, інтеграції власних шейдерів та простоті використання. Було розглянуто такі інструменти:

1. Unreal Engine (рисунок 3.1) – потужний рушій, широко відомий своїми можливостями високоякісного рендерингу графіки та підтримкою складних,

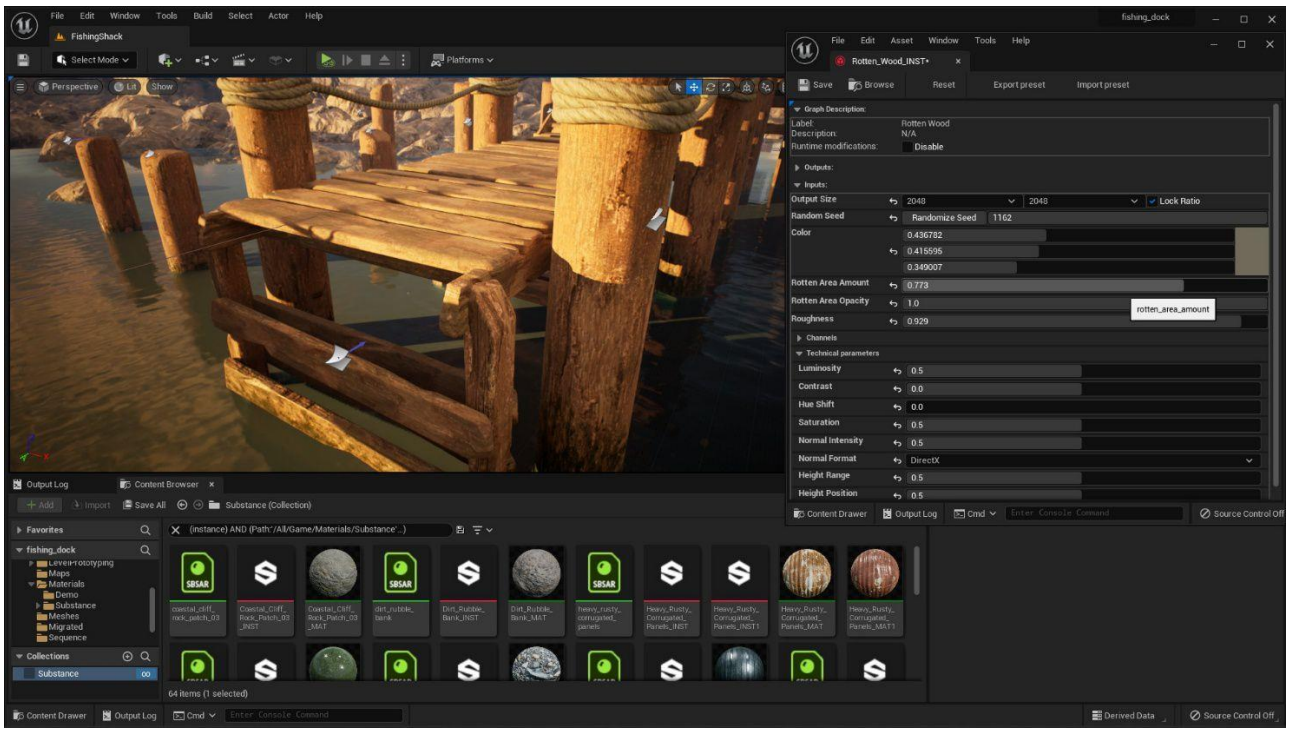


Рисунок 3.1 – Unreal Engine

масштабних проєктів [25]. Він надає візуальну систему сценаріїв під назвою Blueprint, яка полегшує розробникам створення прототипів функціональності без глибоких знань з програмування. Unreal Engine вирізняється фотореалістичним рендерингом і часто обирається для розробки AAA-ігор та застосунків віртуальної реальності. Однак, його широкі можливості приходять за рахунок більш крутої кривої навчання та високих вимог до апаратного забезпечення, що робить його менш практичним для невеликих проєктів або тих, що зосереджені на швидкому створенні прототипів.

2. Unity (рисунок 3.2) – це універсальний рушій візуалізації, відомий своєю гнучкістю та кросплатформеністю [26], що дозволяє розробникам розгортати застосунки на різних платформах, включаючи мобільні, консольні та настільні комп’ютери. Unity пропонує широку підтримку як для 2D, так і для 3D розробки, і має зручний інтерфейс, що приваблює початківців. Завдяки великому сховищу ресурсів та потужній спільноті розробників, Unity надає широкий спектр ресурсів, які полегшують розробку. Однак оптимізація може стати проблемою, особливо для проєктів, які вимагають точного контролю над графікою.

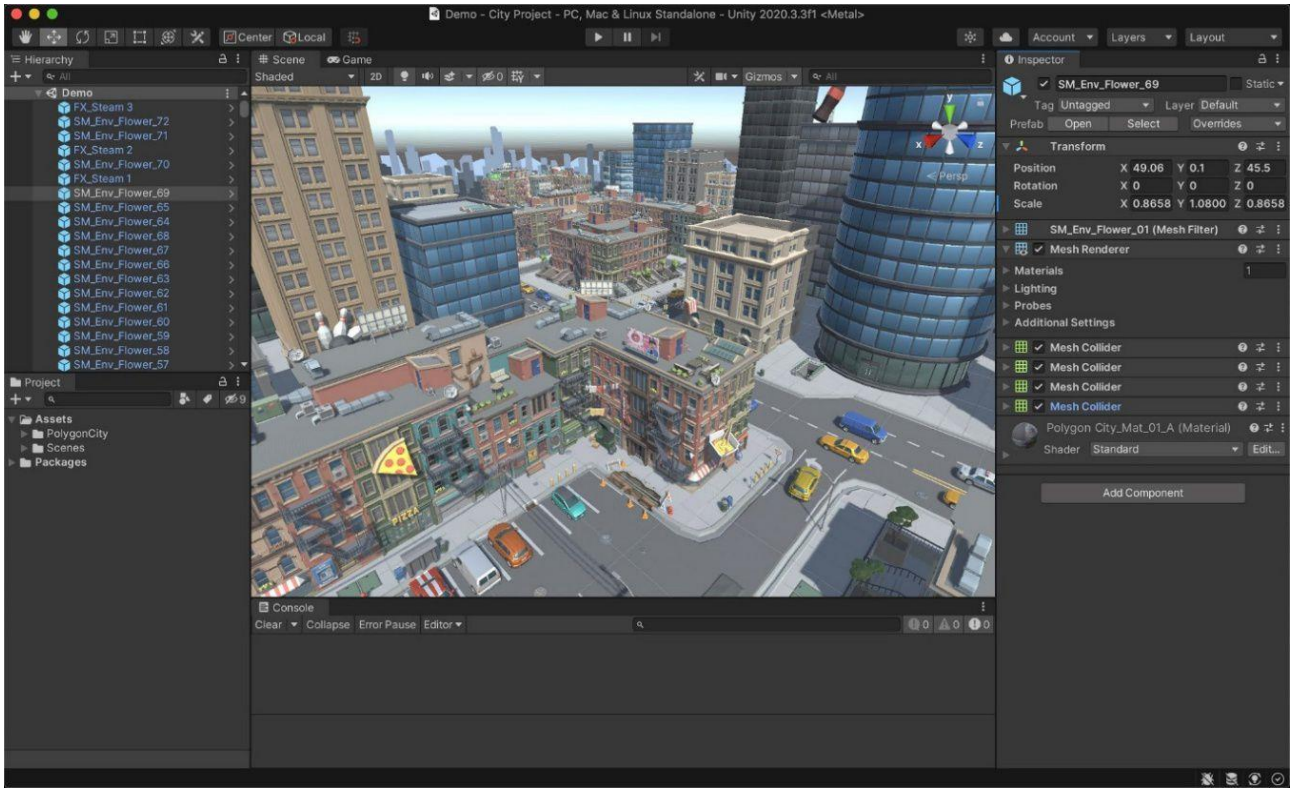


Рисунок 3.2 – Unity

3. Godot (рисунок 3.3) – це рушій з відкритим вихідним кодом, який здобув популярність завдяки своїй легкості та простоті [27]. Він надає унікальну систему сцен і підтримує як 2D, так і 3D розробку, що робить його придатним для широкого спектру проєктів. Мова сценаріїв Godot, GDScript, проста у вивченні, а сам рушій пропонує низький вхідний бар'єр для нових розробників. Хоча він все ще розвивається, Godot не має деяких розширених функцій, які є у більш зрілих рушіях, таких як Unreal та Unity, що робить його менш придатним для висококласних або дуже вимогливих проєктів.

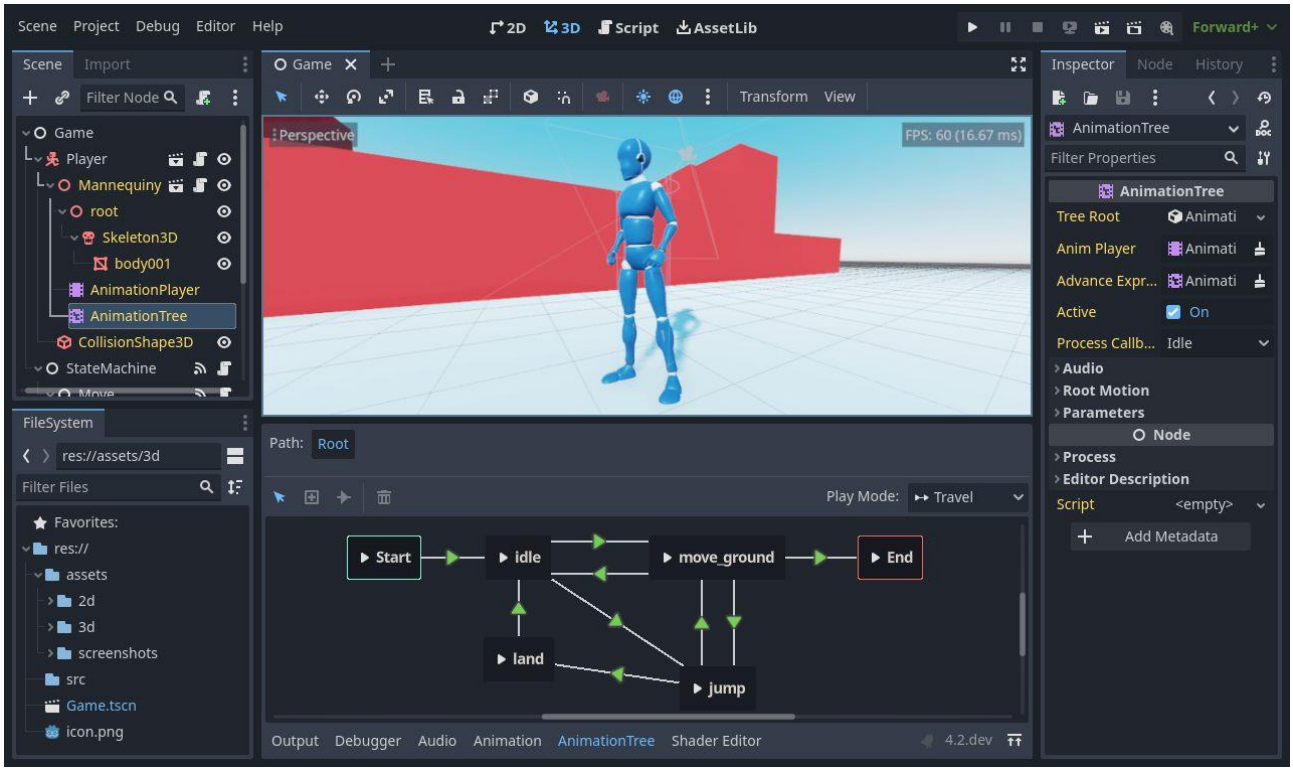


Рисунок 3.3 – Godot

4. jMonkeyEngine (рисунок 3.4) – це рушій на основі Java, розроблений з акцентом на 3D-розробку. Він пропонує модульну структуру та прямий доступ до OpenGL, що дає розробникам більше контролю над низькорівневим рендерингом. jMonkeyEngine добре підходить для застосунків, які потребують нестандартних графічних рішень, наприклад, у тому числі для демонстрації методів рельєфного текстурування.

Порівняльний аналіз описаних рушіїв наведено у табл. 3.1.

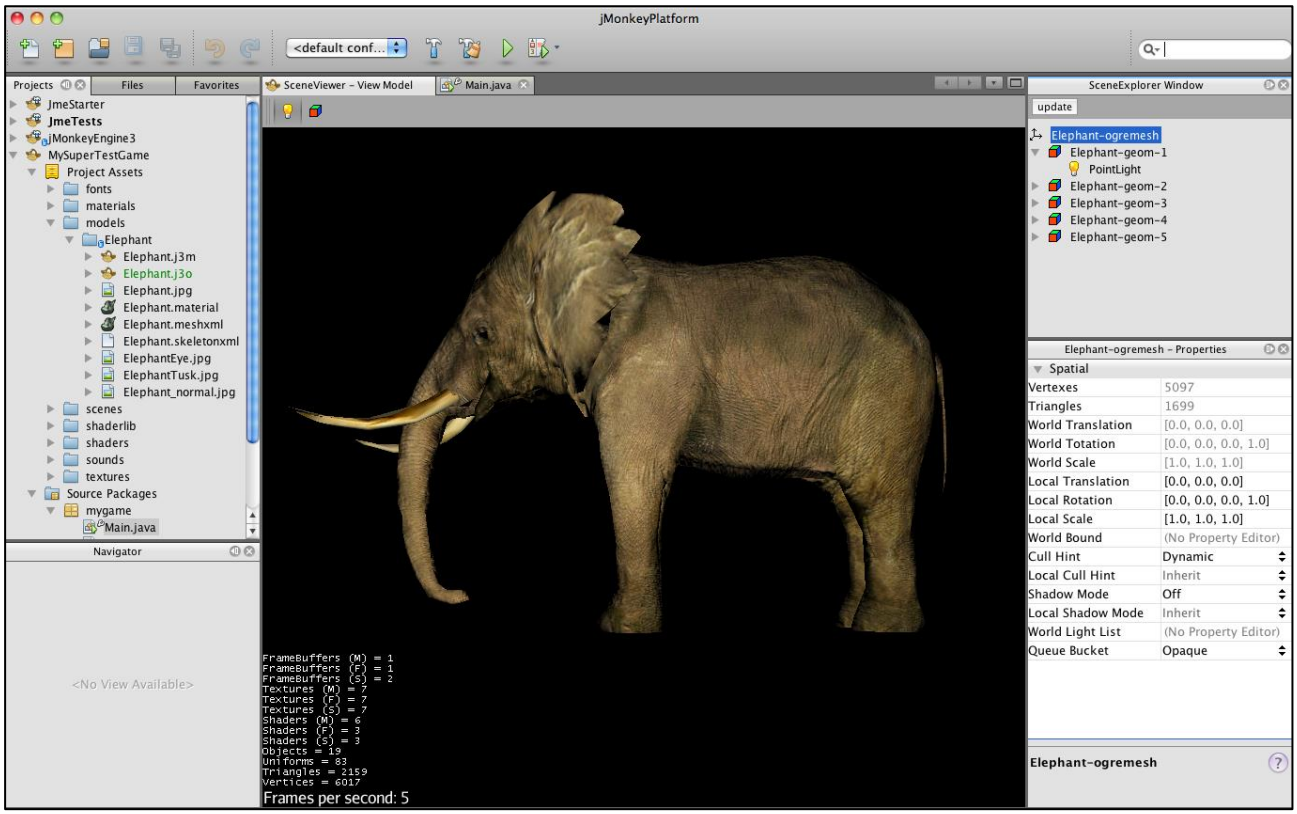


Рисунок 3.4 – jMonkeyEngine

Таблиця 3.1 – Порівняльний аналіз рушіїв візуалізації

Критерій	**Unreal Engine**	**Unity**	**Godot**	**JMonkeyEngine 3**
Відкритий вихідний код	–	–	+	+
Простота інтеграції шейдерів	+	+	–	+
Продуктивність для невеликих проєктів	–	+	+	+
Швидкість освоєння	–	+	+	+
Легкість та можливість налаштування	–	+	+	+
Спільнота та документація	+	+	+	+
Всього	2	5	5	6

jMonkeyEngine 3 було обрано насамперед через його відкритий вихідний код, малу вагу та потужну підтримку інтеграції шейдерів, що є важливим для потреб проєкту. У порівнянні з Unreal Engine, який має довшу криву навчання та

складніші набори інструментів, jME3 простіший в управлінні для невеликих проєктів, що зосереджені на рендерингу. Хоча Unity та Godot є більш гнучкими, jME3 виділяється своєю архітектурою на основі Java та відкритим вихідним кодом, що робить його вдалим вибором для розробки.

3.1.2 Порівняльний аналіз середовищ розробки

Наступним кроком є вибір правильного середовища розробки, яке полегшить реалізацію програмного продукту. Кожне середовище розробки має свої переваги і пристосоване до конкретних потреб. Було оцінено декілька IDE на предмет сумісності з jMonkeyEngine 3 та їхньої загальної придатності для керування 3D-проєктами та шейдерами [28]:

Eclipse IDE – це середовище розробки з відкритим вихідним кодом, яке підтримує широкий спектр мов програмування та фреймворків, включаючи Java, яка є основною мовою для jMonkeyEngine. Eclipse пропонує тісну інтеграцію з інструментами контролю версій та надає широку екосистему плагінів, що підвищує її гнучкість. Однак для початківців Eclipse може здатися захараченим і складним, і він вимагає ретельного налаштування, щоб максимально розкрити його можливості.

IntelliJ IDEA – потужне середовище розробки, відоме своєю інтелектуальною підтримкою коду та розширеними інструментами налагодження. Вона пропонує глибоку інтеграцію з фреймворками на основі Java і користується популярністю серед професійних розробників завдяки простоті використання та можливостям, що підвищують продуктивність. IntelliJ надає чудову підтримку для рефакторингу та завершення коду, але це ресурсномістке середовище, яке може бути не ідеальним для легких проєктів або розробників, які шукають простіший інструмент.

NetBeans – ще одне IDE з відкритим вихідним кодом, яке пропонує вбудовану підтримку Java та багатьох інших мов програмування. Воно надає зручний інтерфейс і хороші інструменти для управління проєктами. NetBeans добре підходить для навчальних цілей і невеликих проєктів, але йому не вистачає

деяких розширених функцій і оптимізації продуктивності, які є в IntelliJ. Як результат, його часто обирають розробники, які віддають перевагу простоті та легкості у використанні над складною функціональністю.

jMonkeyEngine SDK – це спеціалізоване середовище розробки, створене спеціально для роботи з проєктами jMonkeyEngine [29]. Воно надає вбудовані інструменти для 3D-моделювання, управління ресурсами та візуалізації графів сцен, спрощуючи процес розробки 3D-застосунків. SDK тісно інтегрується з рушієм, усуваючи необхідність складної конфігурації, що робить його особливо зручним для швидкого створення проєктів. Хоча jMonkeyEngine SDK не має універсальності більш просунутих IDE, його орієнтованість на розробку з jMonkeyEngine надає значні переваги для проєктів, які потребують безшовної інтеграції з цим рушієм.

Порівняння середовищ розробки наведене у табл. 3.2.

Таблиця 3.2 – Порівняльний аналіз середовищ розробки

Критерій	**Eclipse**	**IntelliJ IDEA**	**NetBeans**	**jMonkeyEngine SDK**
Підтримка Java	+	+	+	+
Інтеграція з JME3	–	–	+	+
Інструменти для редагування 3D-сцен	–	–	–	+
Інструменти для налагодження шейдерів	–	–	–	+
Вбудований редактор матеріалів	–	–	–	+
Простота налаштування проєктів jME3	+	–	+	+
Всього	2	1	3	6

jMonkeyEngine SDK є доцільним вибором для розробки насамперед через його тісну інтеграцію з jME3. Цей інструмент пропонує спеціалізовані інструменти, які спрощують керування 3D-сценою, налагодження шейдерів та редагування матеріалів. На відміну від універсальних IDE, таких як Eclipse або

IntelliJ, jME3 SDK повністю відповідає потребам проєкту. Спеціальні інструменти для редагування сцен та управління шейдерами усувають необхідність у додаткових плагінах або налаштуваннях, що дозволяє зробити робочий процес більш ефективним та впорядкованим.

3.2 Реалізація утиліти для генерації карт складності

Запропонований адаптивний метод на основі POM передбачає використання спеціальних карт, які відображають рівень складності різних ділянок текстури. Для автоматизації процесу створення цих карт доцільно створити допоміжний програмний засіб, який стане важливим інструментом попередньої обробки текстур для використання із покращеною технікою.

Для розробки програми як мову програмування було обрано Python. Простий синтаксис та широка екосистема бібліотек роблять цю мову ідеальною для швидкої розробки та тестування складних алгоритмів, що було є важливим для створення ефективного інструменту для генерації карт складності.

Для виконання необхідних обчислень та операцій з обробки зображень було використано декілька спеціалізованих бібліотек:

– NumPy для ефективних числових обчислень, зокрема для обробки великих масивів і матриць. NumPy було використано для виконання попіксельних операцій із зображенням, обчислення градієнтів та математичних перетворень над текстурними даними;

– бібліотека комп’ютерного зору OpenCV з відкритим вихідним кодом, що надає широкий спектр функцій обробки зображень, включаючи оператори виявлення контурів та морфологічні операції.

Робота програми розпочинається із завантаження графічного файлу, заданого користувачем, що представляє карту висот. Карта висот зчитується в кольоровому форматі (RGB), з припущенням, що інформація про висоту кодується в червоному каналі. Якщо зображення карти висот не вдається завантажити, програма завершує роботу з повідомленням про помилку,

гарантуючи, що подальші операції відбуватимуться лише з коректними вхідними даними.

Генерація карти складності розпочинається з того, що інформація про висоту, що зберігається в червоному каналі, витягується і нормалізується до діапазону [0,1], що готує масив до обробки як зображення у відтінках сірого. Цей процес може бути представлений діаграмою на рисунку 3.5.

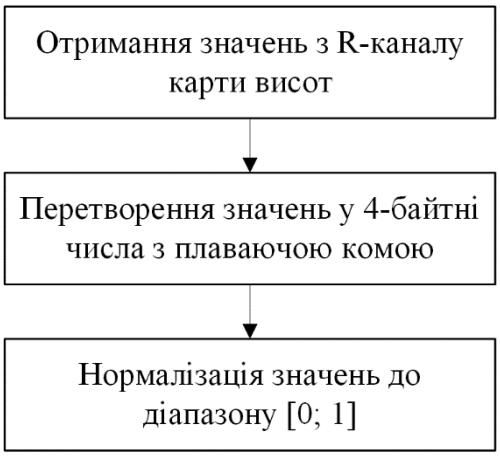


Рисунок 3.5 – Підготовка вхідних даних

Після цього обчислюються наближення градієнтів в напрямках X та Y за допомогою оператора Собеля. Тут стає в нагоді бібліотека OpenCV, яка пропонує готову реалізацію цього методу. Функції Sobel() передаються такі аргументи [30]:

– src_gray: вхідне зображення у відтінках сірого;

– ddepth: глибина вихідного зображення; використано константу CV_32F, що означає 4-байтні числа з плаваючою комою;

– x_order: порядок похідної за напрямком X;

– y_order: порядок похідної за напрямком Y;

– ksize: розмір ядра Собеля; для розрахунків використано ядро розміром 3x3.

Для обчислення градієнта в напрямку X використовується x_order = 1 та y_order = 0, а для напрямку Y встановлюються протилежні значення.

Загальна величина градієнта у конкретному пікселі обчислюється шляхом комбінування отриманих значень градієнтів у напрямку X та Y. Це значення

відображає складність текстури, де більші значення вказують на складніші області та навпаки.

Після обчислення величини градієнта наступним кроком є нормалізація значень до діапазону [0; 1] для отримання значень кольору у відтінках сірого. Для цього виконується масштабування на основі максимального градієнта, гарантуючи, що всі значення знаходяться в межах стандартного діапазону.

На цьому етапі вже сформовано початковий варіант карти складності, проте для підвищення її ефективності у відображенні деталей поверхні застосовується додаткова обробка зображення. Вона включає етапи, що наведені на рисунку 3.6.

Рисунок 3.6 – Додаткова обробка карти складності

Розширення застосовується для збільшення областей зі значними варіаціями висоти текстури. Для цього використовується ядро 3x3. Це коригування дозволяє збільшити кількість шарів дискретизації не лише на ділянках із високою складністю, а й навколо їх, що дозволяє зменшити ймовірність виникнення артефактів.

Після цього виконується гамма-корекція зі значенням гамми 0,9, щоб підкреслити менш складні ділянки текстури. Це дозволяє врахувати незначні коливання висоти текстури, які інакше могли б виглядати занадто однорідними.

Для адаптації неперервних значень складності до використання у шейдері POM до них застосовується квантування на дискретні рівні. Для цього значення

складності поділяються на 24 рівні, кожен з яких представляє певний діапазон складності. Такий крок дозволяє згрупувати подібні за складністю області та уникнути зайвих коливань, до який шейдер POM нечутливий.

Після описаних перетворень згенерована карта складності зберігається за вказаним користувачем шляхом. Приклад взаємодії користувача з програмою наведено на рисунку 3.7. Лістинг програмного засобу подано в додатку В.

```
Input path: height_map.png
Output path: complexity_map.png
Complexity map saved as complexity_map.png.

Process finished with exit code 0
```

Рисунок 3.7 – Приклад взаємодії користувача з програмою

На рисунку 3.8 наведено приклад згенерованої карти складності із використанням розробленої утиліти.

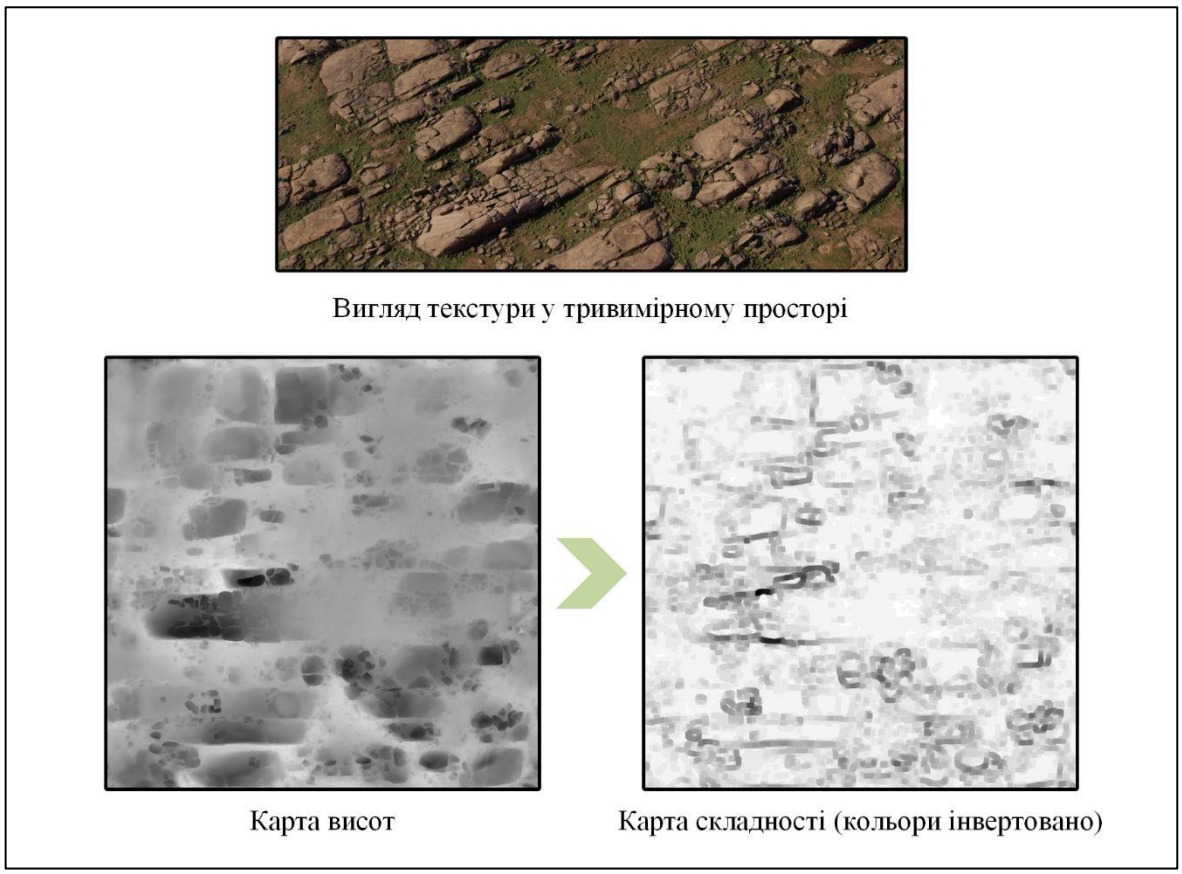


Рисунок 3.8 – Генерація карти складності для текстури

3.3 Шейдерна реалізація методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping

Для реалізації методу POM з динамічним регулюванням кількості шарів дискретизації для імітації нерівностей на пласкій поверхні необхідно створити вершинний та фрагментний шейдери.

Вершинний шейдер ініціалізує процес текстурування шляхом перетворення вершин поверхні відповідно до матриць вигляду та моделі, забезпечуючи належне позиціювання об'єкта на 3D-сцені (рисунок 3.9) [31, 32]. Шейдер отримує основні вхідні дані, зокрема положення вершин, нормалі, дотичні та UV-координати, щоб перетворити координати моделі у простір відображення. Крім того, на цьому етапі виконується обчислення матриці дотичного простору, яка буде використана для динамічного налаштування координат текстури.

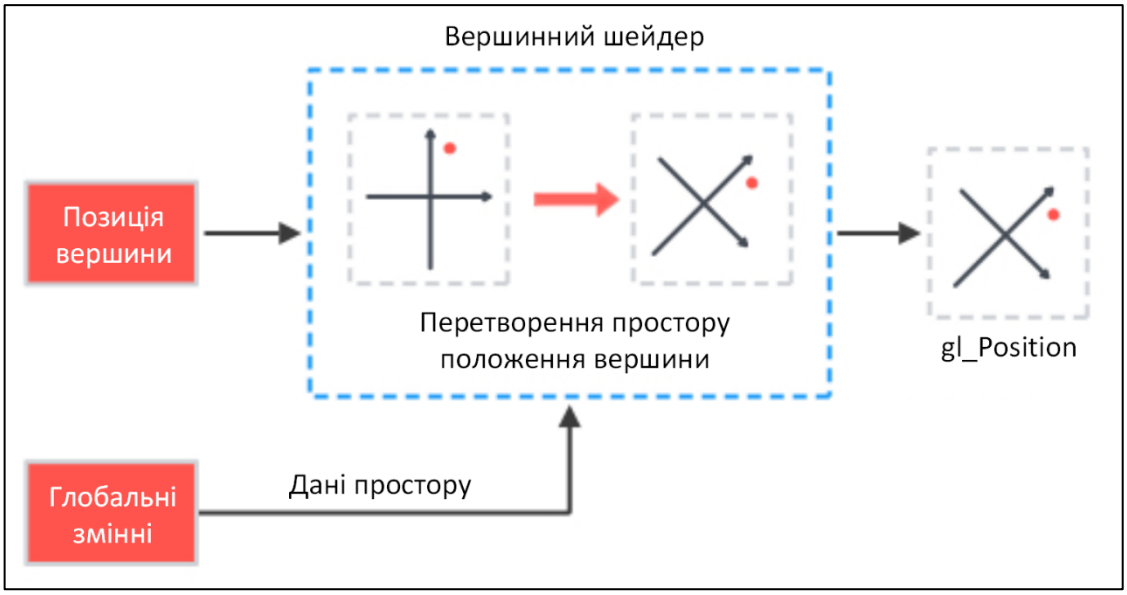


Рисунок 3.9 – Принцип роботи вершинного шейдера

Після обчислення необхідних перетворень вершинний шейдер передає координати текстури та трансформовані координати вигляду і положення світла до фрагментного шейдера (рисунок 3.10). Ці дані інтерполюються по поверхні моделі і дозволяють виконувати розрахунки для кожного фрагмента, необхідні для точного моделювання глибини та освітлення на піксельній основі.

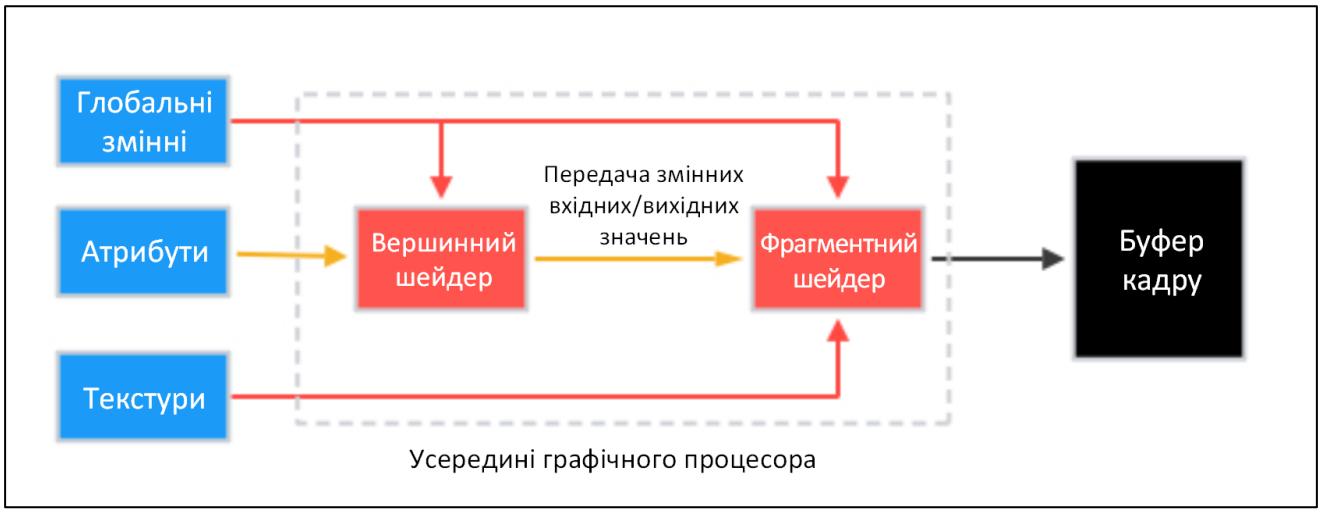


Рисунок 3.10 – Взаємодія вершинного та фрагментного шейдерів

У фрагментному шейдері реалізована основна функціональність рельєфного текстурування. Окрім отриманих значень з вершинного шейдера, тут використовується декілька глобальних параметрів, таких як масштабування глибини текстури, максимальна та мінімальна кількість шарів, а також карти висот, нормалей, складності та дифузна карта.

Карта висот слугує основним джерелом інформації про глибину, а її значення використовуються для визначення величини зсуву, що застосовується до координат текстури. Для отримання текселя з текстури за заданими координатами використовується функція texture(), що приймає такі аргументи [33]:

– sampler: вказує семплер, до якого прив’язано текстуру, з якої буде отримано текселі;

– P: координати текстури, за якими буде проведено вибірку.

Параметр HeightScale використовується для масштабування значень на карті висот, що дозволяє контролювати інтенсивність ефекту глибини.

Залежно від кута огляду, шейдер підлаштовує координати текстури для вибірки, ітераційно виконуючи трасування променю вздовж вектору погляду, поки не буде досягнуто перетину зі значенням глибини з карти висот.

Кількість ітерацій (шарів дискретизації) динамічно регулюється залежно від складності текстури та кута огляду (рисунок 3.11). Так, параметри MinLayers та MaxLayers визначають межі, в яких відбувається вибір остаточної кількості

шарів дискретизації для текселя. Після цього цей діапазон коригується із урахуванням кута огляду для отримання значень adjustedMinLayers та adjustedMaxLayers. Остаточна кількість шарів дискретизації numLayers визначається на основі значення складності у текселі, яке отримується з карти складності.

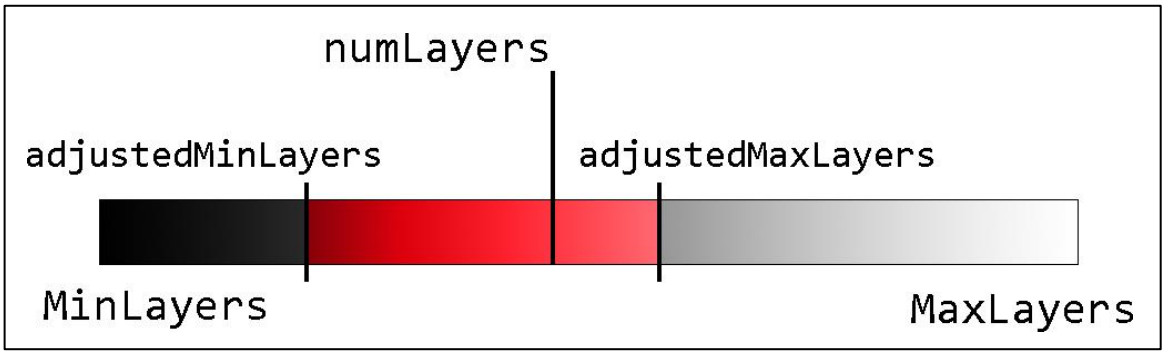


Рисунок 3.11 – Динамічне регулювання кількості шарів дискретизації

Результат роботи шейдерів після виконання описаних кроків наведено на рисунку 3.12. На цьому етапі завдяки зміщенню координат текстури досягається імітація нерівностей на поверхнях, проте ілюзія глибини не виглядає переконливо, особливо при прямому погляді на них.

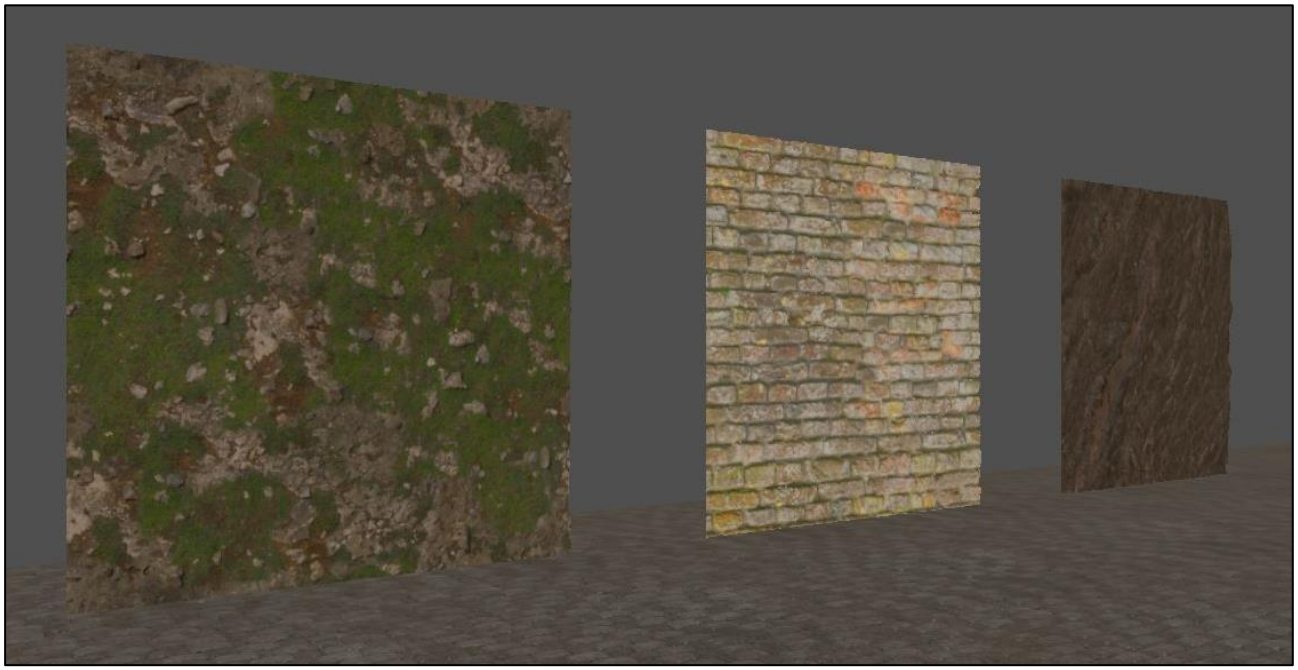


Рисунок 3.12 – Результат роботи шейдерів

Для підвищення реалістичності зображення виконується збурення напрямків векторів нормалей поверхні відповідно до карти нормалей, що дозволяє імітувати дрібні деталі поверхні. Також під час розрахунків враховується ефект від загального освітлення (ambient light) та освітлення поверхні направленим джерелом світла, якщо таке є. Остаточний результат рельєфного текстурування за допомогою методу POM із динамічним регулюванням кількості шарів дискретизації зображено на рисунку 3.13. Лістинг розроблених шейдерів наведено в додатку В.

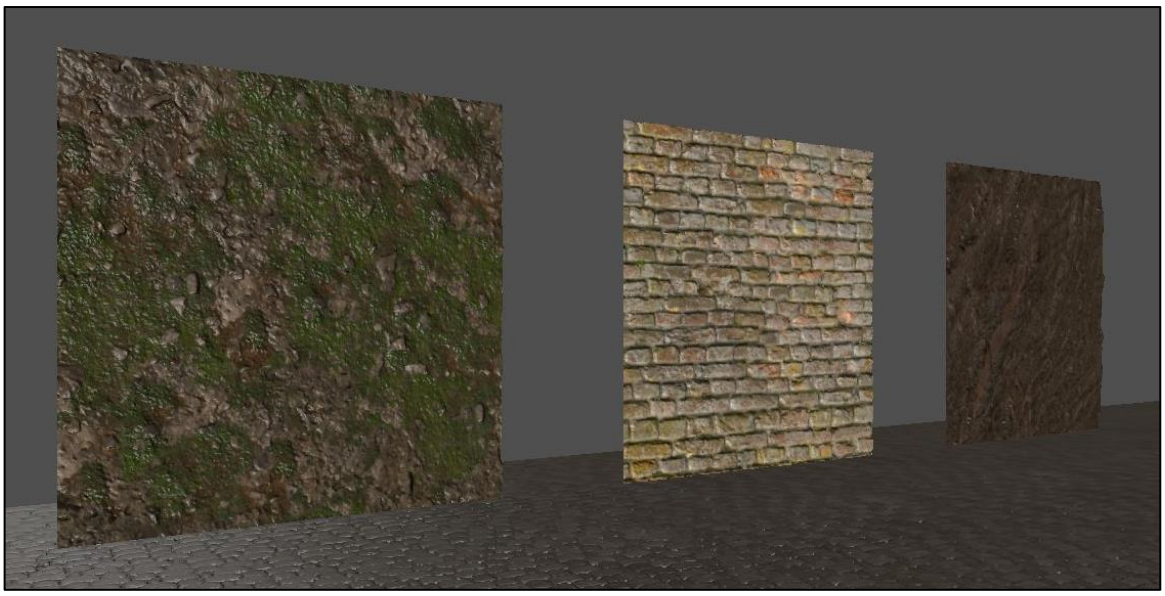


Рисунок 3.13 – Остаточний результат текстурування

Таким чином, реалізація методу рельєфного текстурування з динамічним налаштуванням дискретизації на базі POM дозволяє ефективно імітувати деталі поверхні та ефекти глибини на пласкій геометрії, зберігаючи баланс між продуктивністю та візуальною точністю. Було створено вершинний шейдер для налаштування дотичних перетворень простору та передачі необхідних даних фрагментному шейдеру. Фрагментний шейдер, у свою чергу, динамічно налаштовує кількість шарів дискретизації на основі кута огляду та складності текстури. Цей підхід забезпечує надійну основу для реалістичного текстурування поверхні в реальному часі та забезпечує переконливий ефект глибини, який може динамічно адаптуватися до умов сцени.

3.4 Реалізація програмного забезпечення для демонстрації та порівняння методів рельєфного текстурування

З метою демонстрації розроблених методів та засобів було розроблено програмний застосунок, що забезпечує практичне, інтерактивне середовище, де користувачі можуть досліджувати і візуально оцінювати різні методи текстурування на 3D-поверхнях, що дозволяє безпосередньо порівнювати продуктивність і візуальну точність. За графічну складову у розробленій програмі відповідає рушій jMonkeyEngine 3 (jME3), а графічний інтерфейс користувача було розроблено із використанням бібліотеки Nifty GUI.

Процес розробки застосунку включає такі етапи:

1. Створення логотипу. Добре розроблений логотип – це більше, ніж просто зображення; це обличчя застосунку і перше враження, яке воно справляє на користувача. Логотип програми було розроблено таким чином, щоб відобразити основну мету програми – дослідження текстур у 3D-середовищі (рисунок 3.14). Використання абстрактних елементів і чистий, мінімалістичний дизайн логотипу відображає професійний і технічний характер програми.

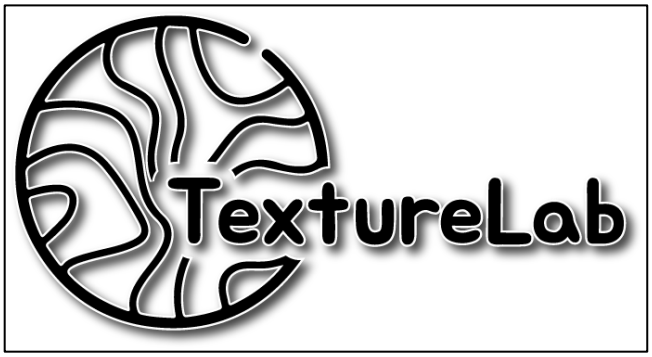


Рисунок 3.14 – Логотип програми

2. Налаштування середовища. Початкове налаштування включає створення базової 3D-сцени, налаштування освітлення та керування камерою для вільного переміщення та огляду текстурованих поверхонь. Процес налаштування параметрів сцени у інтегрованому середовищі jME3 SDK наведено на рисунку 3.15.

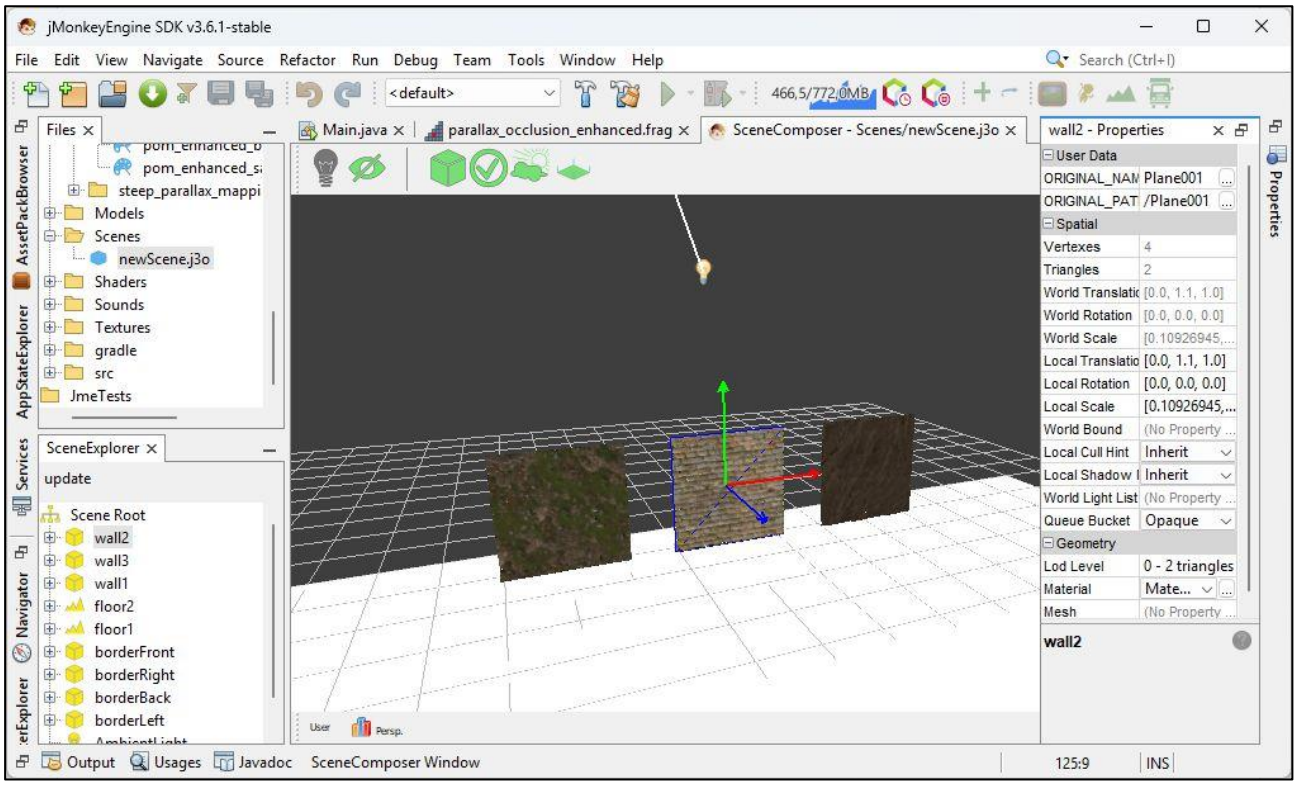


Рисунок 3.15 – Налаштування середовища

3. Інтеграція шейдерів. Для реалізації різних технік текстурування було використано векторні та вершинні шейдери. Код шейдерів було написано мовою GLSL та адаптовано до конвеєра рендерингу jME.

Для ефективного керування властивостями матеріалів було створено файли визначення матеріалів j3m, де зазначаються текстури та параметри шейдерів для кожного матеріалу, що використовується у сцені. Ці файли було відредаговано за допомогою вбудованого в середовище розробки редактора матеріалів, що дозволило спростити та візуалізувати налаштування матеріалів.

На рисунку 3.16 наведено конфігурацію матеріалу, що використовує запропонований метод на базі POM. Основні налаштування включають:

– DebugMode: перемикач режиму налагодження для візуалізації кількості використаних шарів дискретизації;

– HeightScale: контролює інтенсивність ефекту глибини при текстуруванні;

– MinLayers та MaxLayers: базовий діапазон можливих значень для динамічного регулювання кількості шарів дискретизації;

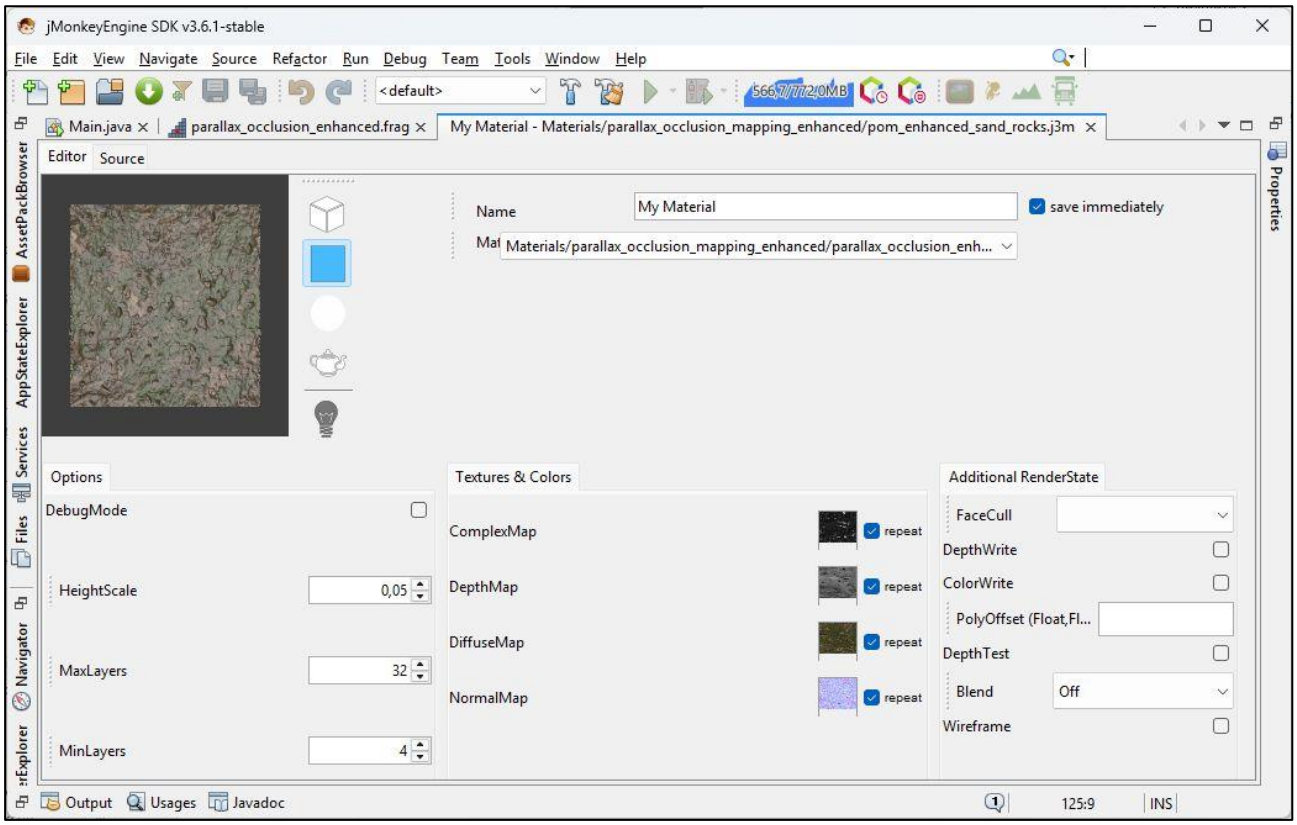


Рисунок 3.16 – Редактор матеріалів

– DiffuseMap, DepthMap та NormalMap: карти, що використовуються методом POM для імітації нерівностей на поверхні;

– ComplexMap: карта, що містить інформацію про складність ділянок текстури та використовується для визначення кількості шарів дискретизації на основі цього показника.

4. Створення графічного інтерфейсу користувача. Із використанням бібліотеки Nifty GUI було реалізовано меню, а також елементи керування в головному вікні програми.

Головне меню (рисунок 3.17) слугує точкою входу до застосунку. У верхній частині знаходиться логотип програми, а під ним розміщено функціональні кнопки:

– кнопка «Старт» запускає головне вікно застосунку;

– «Налаштування» дозволяє налаштувати параметри, пов’язані із зовнішнім виглядом застосунку та методами текстурування;

– «Вихід» закриває програму.

Рисунок 3.17 – Меню застосунку

Градієнтний фон вікна та сучасний дизайн кнопок створюють привабливий інтерфейс, зберігаючи при цьому мінімалістичний та професійний вигляд.

Натиснувши кнопку «Старт», користувач потрапляє до головного вікна програми, де має змогу дослідити різні техніки текстурування (рисунок 3.18). На сцені відображається кілька текстурованих площин, до яких застосована певна техніка рельєфного текстурування. За допомогою визначених кнопок користувач може перемикатися між ними, що дозволяє порівнювати візуальну якість кожного методу безпосередньо в 3D-середовищі.

В нижній частині екрану розташована інформаційна панель, де детально описується активний метод текстурування і надається його короткий опис. Наприклад, якщо вибрано метод POM, як на рисунку, панель пояснює, як він покращує сприйняття глибини, і вказує кількість використаних шарів вибірки.

У верхньому лівому куті лічильник FPS відображає поточну, мінімальну, середню і максимальну кількість кадрів на секунду, що дозволяє відстежувати зміни продуктивності при перемиканні між техніками.

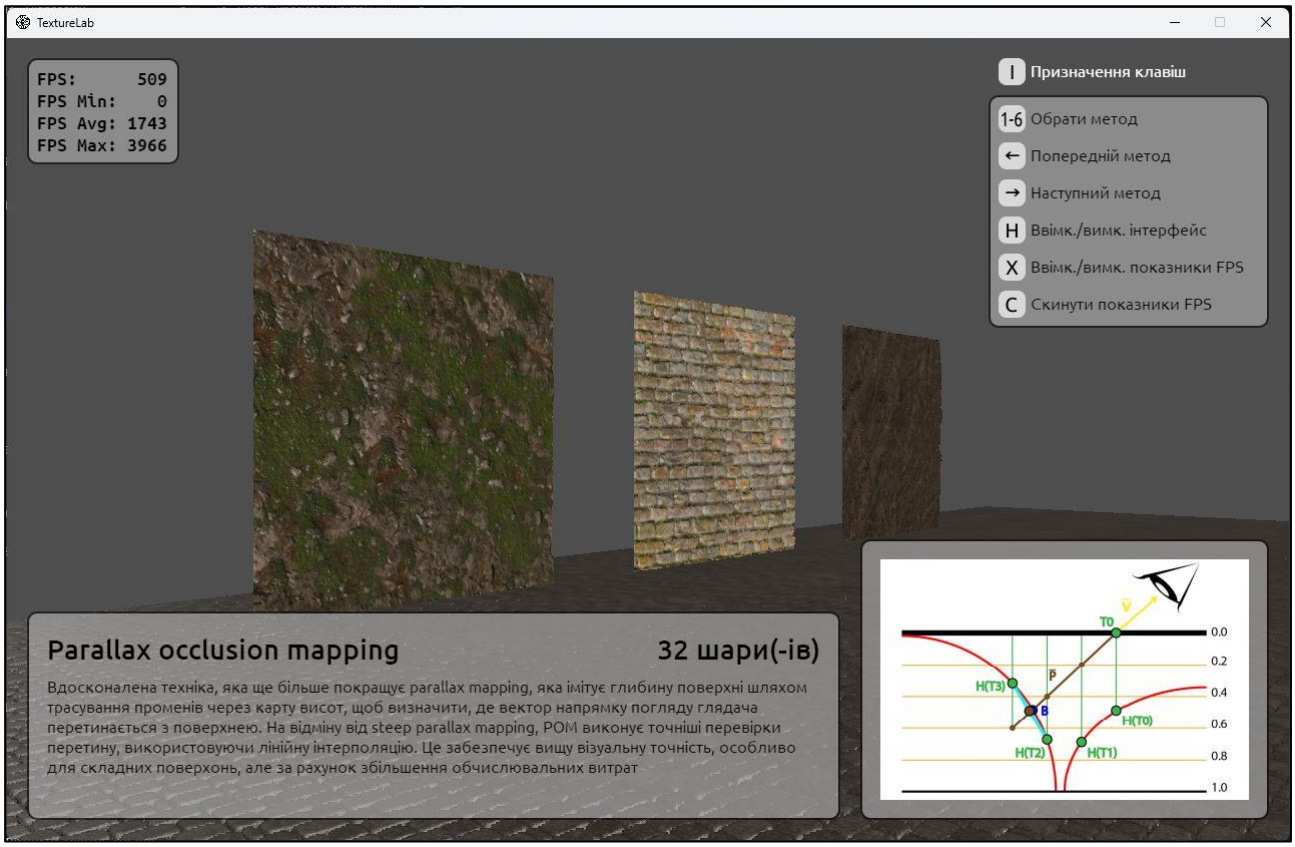


Рисунок 3.18 – Головне вікно програми

Крім цього, користувач має можливість відкрити довідку, що містить список клавіш, які дозволяють взаємодіяти зі сценою, перемикати налаштування та скидати показники продуктивності.

Отже, розроблений застосунок ефективно демонструє різні методи текстурування та надає засоби для їх порівняння в реальному часі. Було реалізовано структурований графічний інтерфейс, що полегшує взаємодію користувача із програмою. Повний лістинг програми подано у додатку В.

3.5 Висновки

У розділі було проведено порівняльний аналіз засобів для реалізації програмного продукту і встановлено доцільність використання рушія jMonkeyEngine 3 та середовища розробки jME3 SDK. Було описано процес реалізації програмного засобу для генерації карт складності та застосунку для демонстрації та порівняння методів рельєфного текстурування. Особлива увага приділена технічним аспектам впровадження розробленого методу формування зображень на базі POM.

4 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МЕТОДІВ ТЕКСТУРУВАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз методів тестування

Тестування програмного забезпечення відіграє вирішальну роль у забезпеченні надійності, точності та ефективності програмного забезпечення. Методи тестування загалом можна розділити на автоматизоване та ручне тестування, кожен з яких має свої переваги та обмеження (рисунок 4.1).

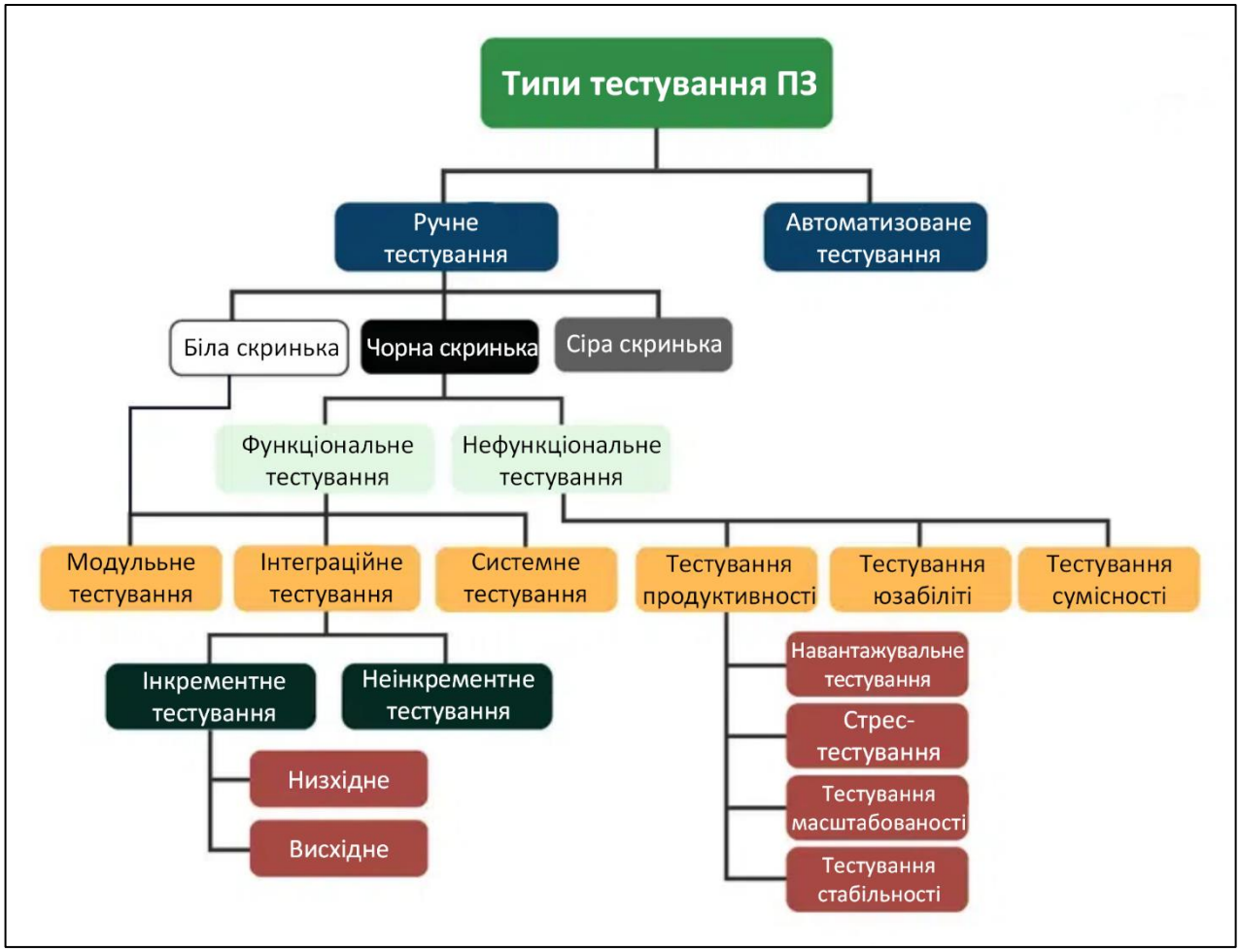


Рисунок 4.1 – Класифікація тестування програмного забезпечення

Автоматизоване тестування передбачає використання скриптів та інструментів для виконання повторюваних тестових завдань без втручання людини [34]. Автоматизовані тести ефективні для регресійного тестування, тестування продуктивності та інших сценаріїв, де основна увага приділяється

швидкості. Прикладами автоматизованого тестування є навантажувальне та стрес-тестування, які імітують високі вимоги до програми, щоб виміряти її реакцію в екстремальних умовах. Однак автоматизоване тестування часто обмежене, коли мова йде про застосунки, що використовують комп'ютерну графіку, де суб'єктивне людське судження необхідне для оцінки візуальної якості та виявлення артефактів.

Ручне тестування дозволяє оцінити візуальну коректність, зручність використання та інші суб'єктивні критерії, які автоматизовані тести можуть не врахувати. Ручному тестуванню зазвичай надають перевагу при тестуванні візуальних ефектів чи графічного інтерфейсу користувача. Воно може включати методи «чорної скриньки», «білої скриньки» та «сірої скриньки», кожен з яких надає різні точки зору на функціональність програми.

Тестування «чорної скриньки» фокусується на перевірці результатів роботи програми без необхідності розуміння її внутрішньої структури або коду. Це особливо корисно для функціонального та системного тестування, оскільки дозволяє тестувальникам підходити до програми так, як це робили б кінцеві користувачі, перевіряючи очікувану поведінку системи. Тестування методом «чорної скриньки» ідеально підходить для перевірки відповідності кожної функції своїм вимогам і може застосовуватися як вручну, так і автоматично.

Інші типи ручного тестування включають методи «білої скриньки», коли тестувальники вивчають внутрішню структуру і логіку коду програми, та «сірої скриньки» – гібридний підхід, який поєднує в собі аспекти обох описаних методів. Хоча тестування «білої скриньки» може бути цінним для модульного та інтеграційного тестування, у контексті розробленого застосунку воно менш доречне, оскільки основну увагу необхідно зосередити на перевірці візуальних результатів та користувацького досвіду, а не на внутрішній структурі коду.

Функціональне тестування дозволяє перевірити, чи відповідає програмне забезпечення очікуванням відповідно до заздалегідь визначених вимог. Для розробленого застосунку, який фокусується на рендерингу візуальних ефектів, функціональне тестування необхідне для того, щоб переконатися, що рельєфне

текстурування та елементи керування програмою працюють належним чином. Функціональне тестування може проводитися на різних рівнях, включаючи модульне тестування для окремих функцій та інтеграційне тестування для перевірки взаємодії між компонентами. Однак особливо важливим є системне тестування, яке перевіряє всю програму як єдине ціле.

Системне тестування дозволяє оцінити поведінку застосунку в середовищі, що максимально наближене до реального варіанту використання. Ця форма тестування оцінює взаємодію всіх компонентів і перевіряє, чи функціонує програма так, як очікується з точки зору кінцевого користувача. У випадку із розробленим застосунком системне тестування особливо важливе для підтвердження відповідності кінцевого результату очікуванням користувача, особливо з точки зору візуальної точності та реалістичності рендерингу.

Отже, враховуючи особливості розробленого програмного забезпечення, ручне функціональне тестування методом «чорної скриньки» є найбільш доцільним підходом. Це дозволить оцінити якість візуальних ефектів та коректність роботи функціоналу застосунку.

4.2 Тестування працездатності програмного забезпечення для демонстрації та порівняння методів рельєфного текстурування

Тестування розробленого застосунку має на меті перевірку його основної функціональності та стабільності, щоб переконатися, що система працює належним чином за типових умов використання. Кожна функція була ретельно протестована, щоб підтвердити, що вона працює правильно і безперешкодно взаємодіє з іншими компонентами програми.

Процес тестування було розпочато з меню програми, щоб переконатися, що кожна кнопка виконує свою функцію (рисунок 4.2). Після натискання кнопки «Старт» програма переходить до головного екрану застосунку, де користувач може взаємодіяти з різними методами текстурування. Кнопка «Налаштування» веде до меню конфігурації, що дозволяє налаштувати параметри програми.

Кнопка «Вихід» успішно закриває програму. Усі елементи пройшли перевірку, за результатами тестування проблем не виявлено.

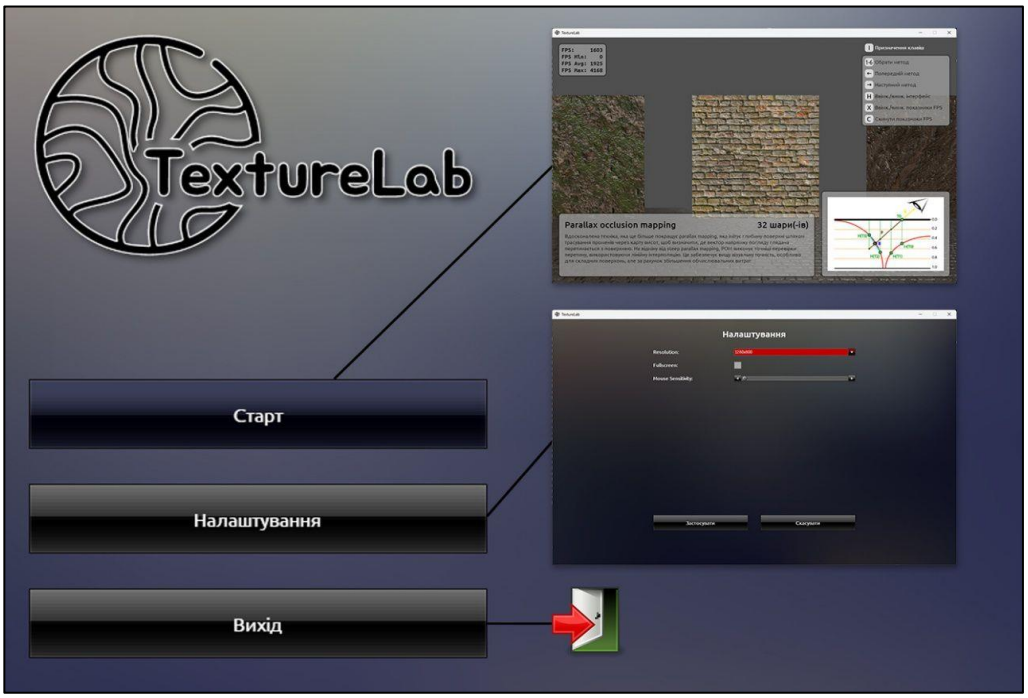


Рисунок 4.2 – Тестування меню застосунку

У меню налаштувань (рисунок 4.3) було протестовано кожну опцію, щоб підтвердити, що зміни зберігаються та застосовуються коректно. Було обрано

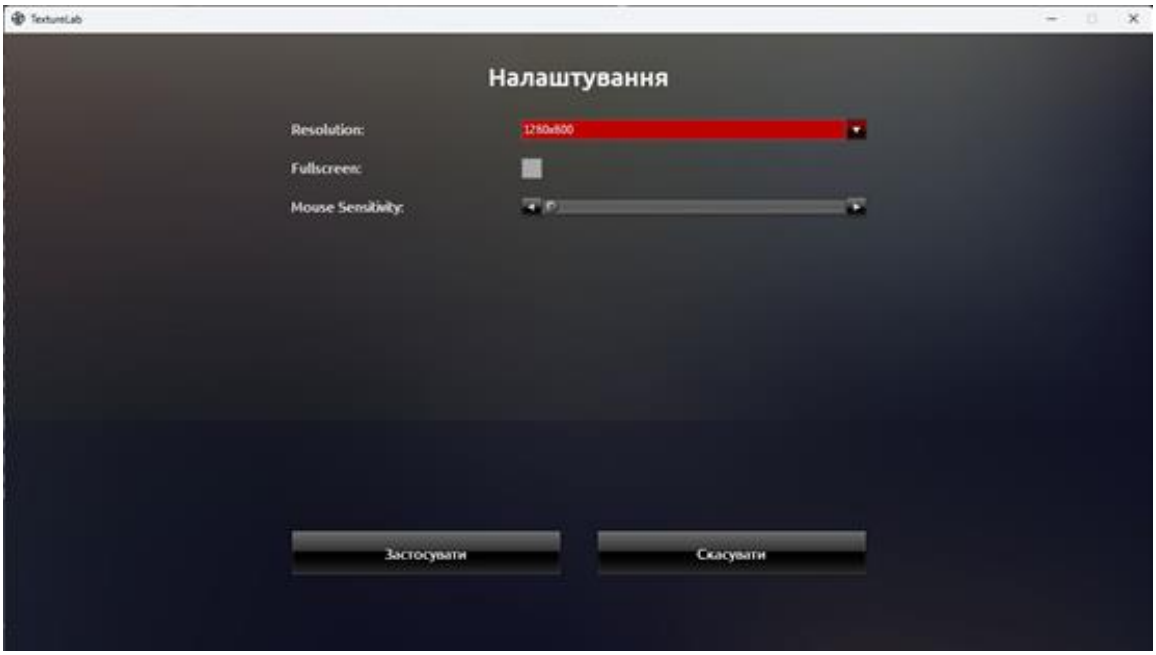


Рисунок 4.3 – Тестування меню налаштувань

різні роздільні здатності екрану, щоб перевірити, чи правильно змінюється розмір вікна програми, а також перевірено її роботу в повноекранному режимі. Після натискання кнопки «Застосувати» усі налаштування були встановлені без помилок.

Відображення методів рельєфного текстурування є основною функцією розробленого застосунку. Користувач має можливість досліджувати і порівнювати різні техніки, що включають базове текстурування, normal mapping, parallax mapping, steep parallax mapping, parallax occlusion mapping та адаптивний метод на базі POM. Під час індивідуального тестування кожного з цих методів програма працює належним чином й оновлює відображувану текстуру, супровідну інформацію про поточний метод текстурування та метрики продуктивності у реальному часі (рисунок 4.4). За результатами тестування проблем не виявлено.

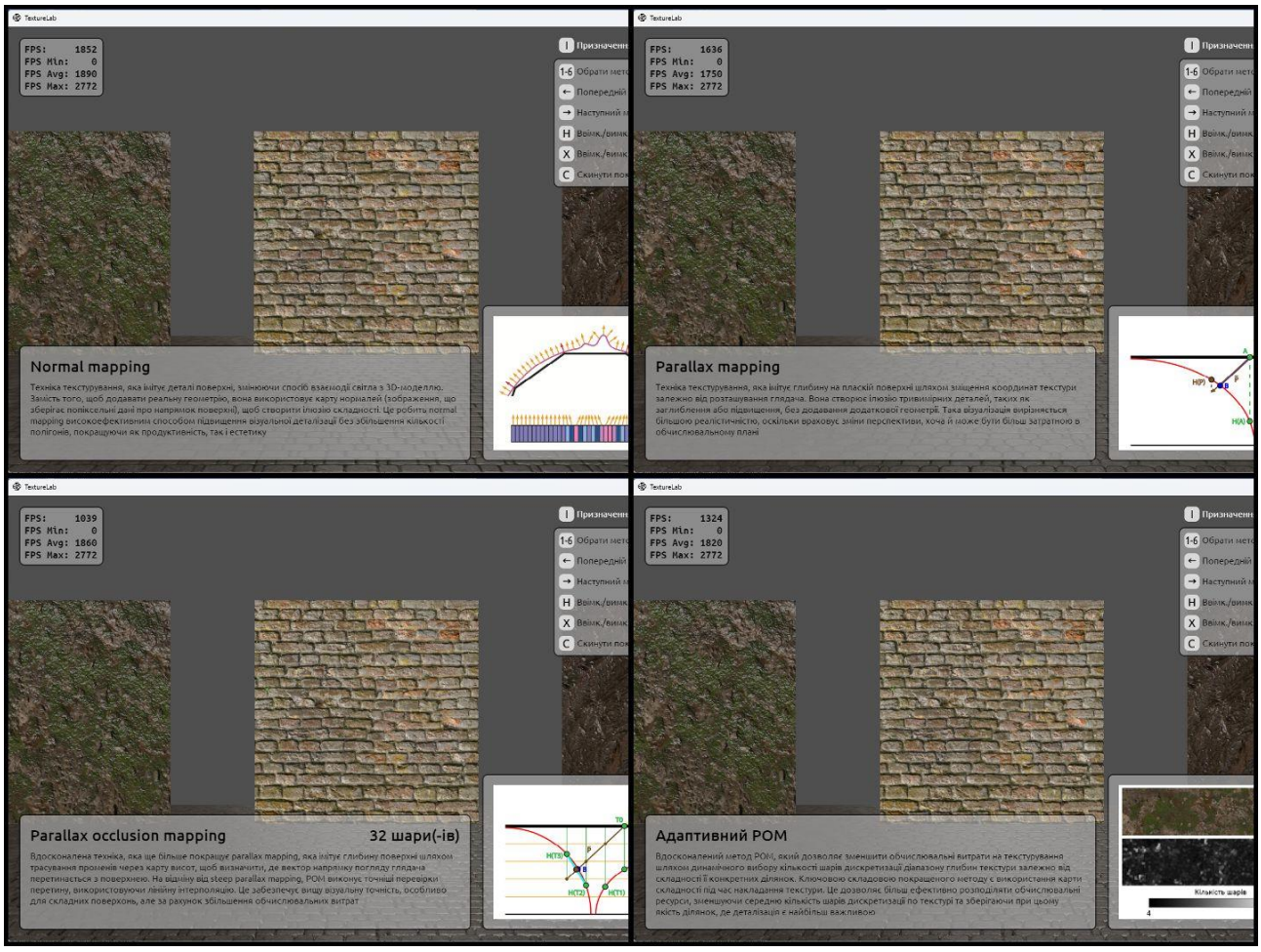


Рисунок 4.4 – Тестування головного екрану

Було перевірено функцію переміщення шляхом навігації по 3D-сцені за допомогою клавіатури, що дозволяє користувачам вивчати деталі кожної текстури під різними кутами (рисунок 4.5). Під час тестування проблем не виявлено – користувач може рухатися та змінювати напрямок погляду камери без збоїв та помилок з боку програми.

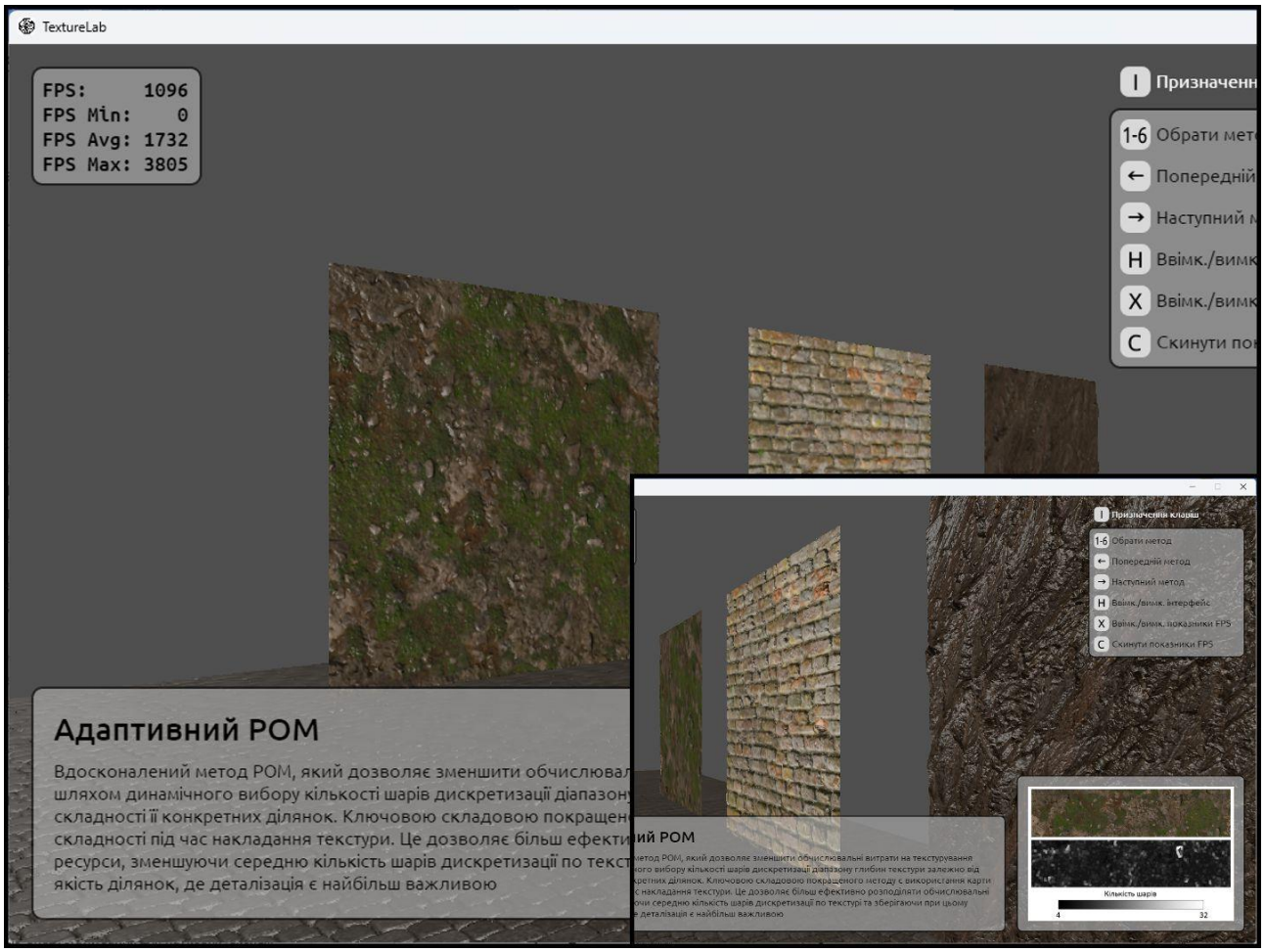


Рисунок 4.5 – Тестування переміщення по сцені

Також були протестовані комбінації клавіш для керування інтерфейсом користувача, щоб перевірити їхню функціональність та якість досвіду користувача (рисунок 4.6). Зокрема, було перевірено функцію динамічного перемикання між методами текстурування, зміну видимості інтерфейсу та індикаторів FPS, а також інформації про призначення клавіш. За результатами тестування, кожна функціональна клавіша виконує свою дію без збоїв та помилок.

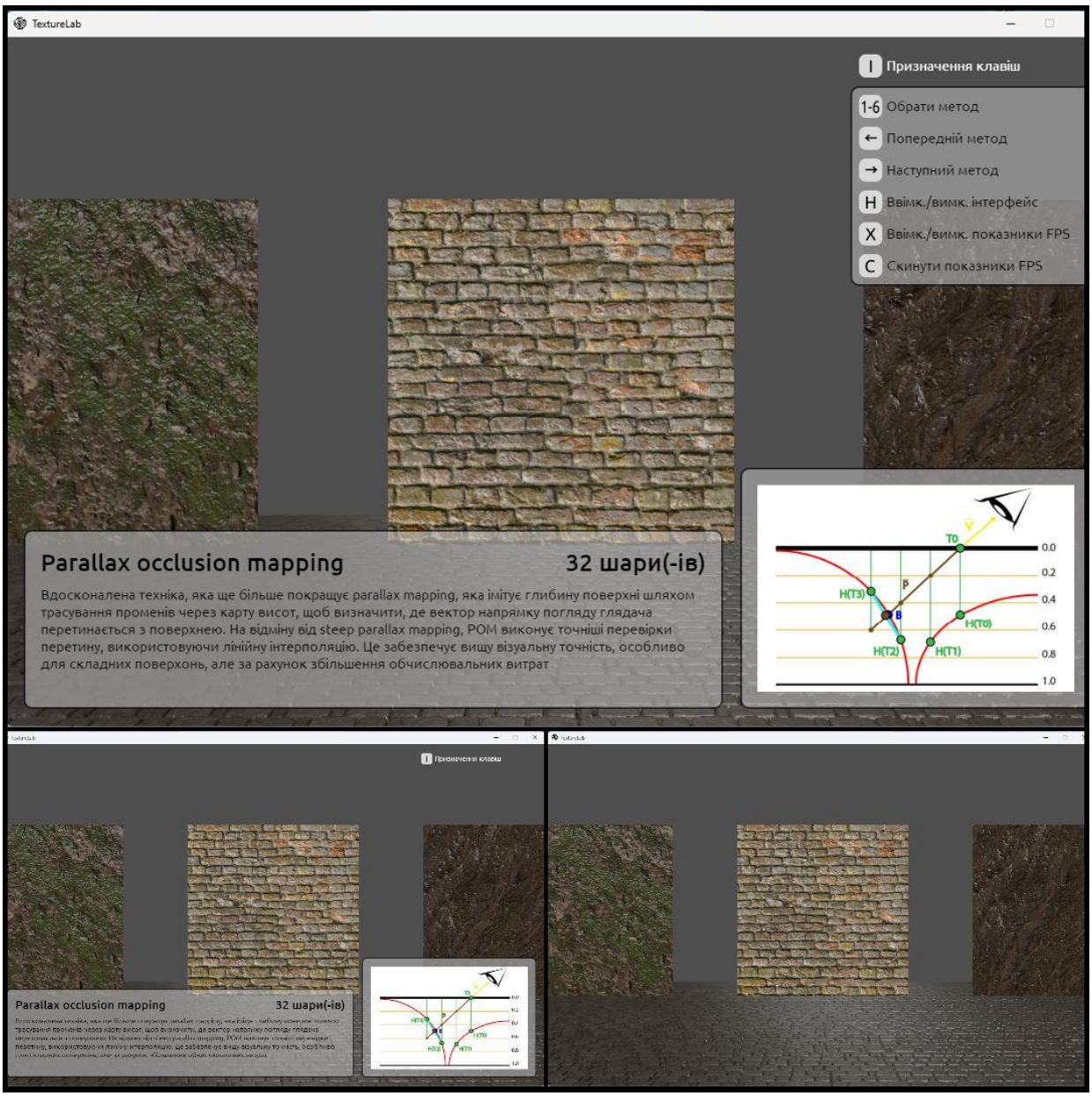


Рисунок 4.6 – Тестування комбінацій клавіш

Крім цього, було перевірено можливість вибору конкретного методу текстурування за допомогою клавіш «1-6». Функція працює належним чином, дозволяючи користувачеві візуально порівнювати між собою будь-які техніки накладання текстур.

Таким чином, проведене тестування підтвердило, що розроблений застосунок відповідає всім функціональним вимогам. Головне меню, налаштування, тривимірна сцена на головному екрані та призначення клавіш працюють належним чином, що свідчить про повну працездатність розробленого застосунку.

4.3 Експериментальні дослідження методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping

Проведення експериментальних досліджень має важливе значення для перевірки і розуміння практичної ефективності запропонованого методу. Метою цього дослідження є перевірка працездатності та демонстрація переваг запропонованого методу над базовою реалізацією POM.

Для перевірки коректності роботи розроблених шейдерів за допомогою програмного засобу для налагодження графічних застосунків RenderDoc було зафіксовано послідовність подій при накладанні текстур на одну з поверхонь. Процес тестування наведено на рисунку 4.7.

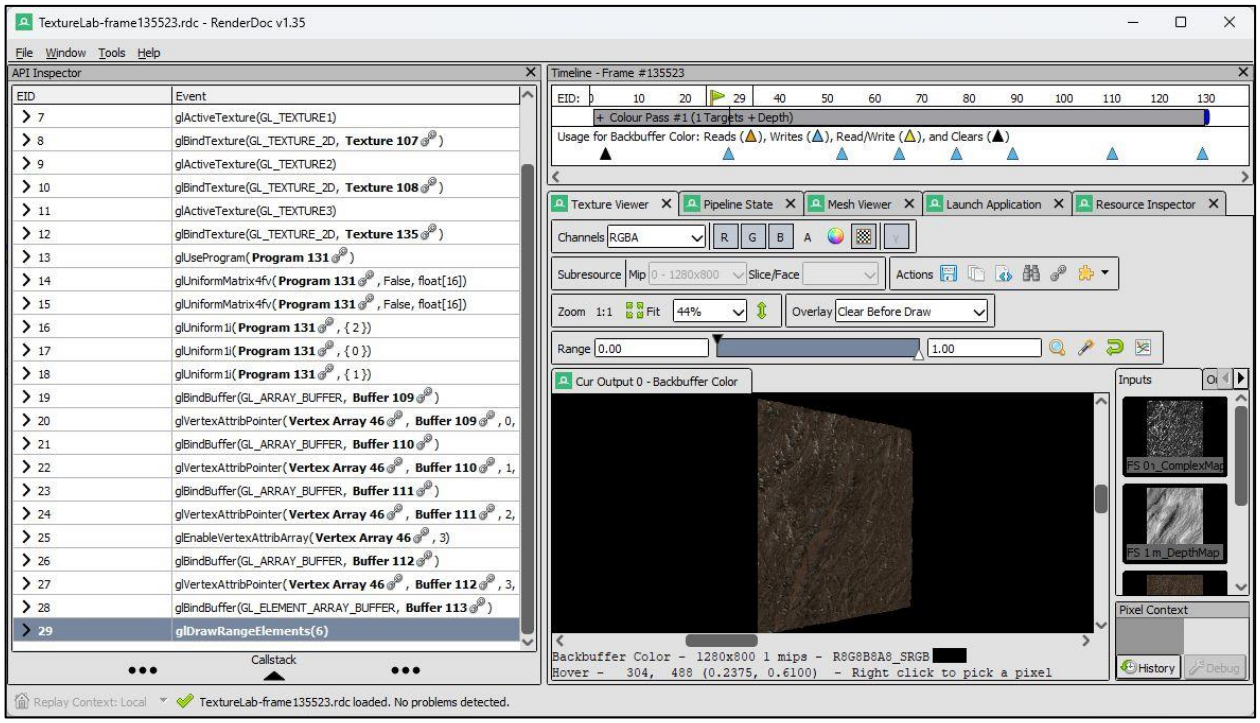


Рисунок 4.7 – Перевірка працездатності шейдера

Кожна зафіксована у RenderDoc подія демонструє, що шейдерна програма працює належним чином. За результатами тестування проблем не виявлено.

Щоб продемонструвати підвищення ефективності рендерингу рельєфних поверхонь, було проведено порівняльний аналіз базового методу POM та запропонованого адаптивного підходу. На рисунку 4.8 наведено приклад застосування базового POM у різних конфігураціях та запропонованого методу,

який динамічно змінює кількість шарів залежно від складності ділянок текстури та кута огляду.

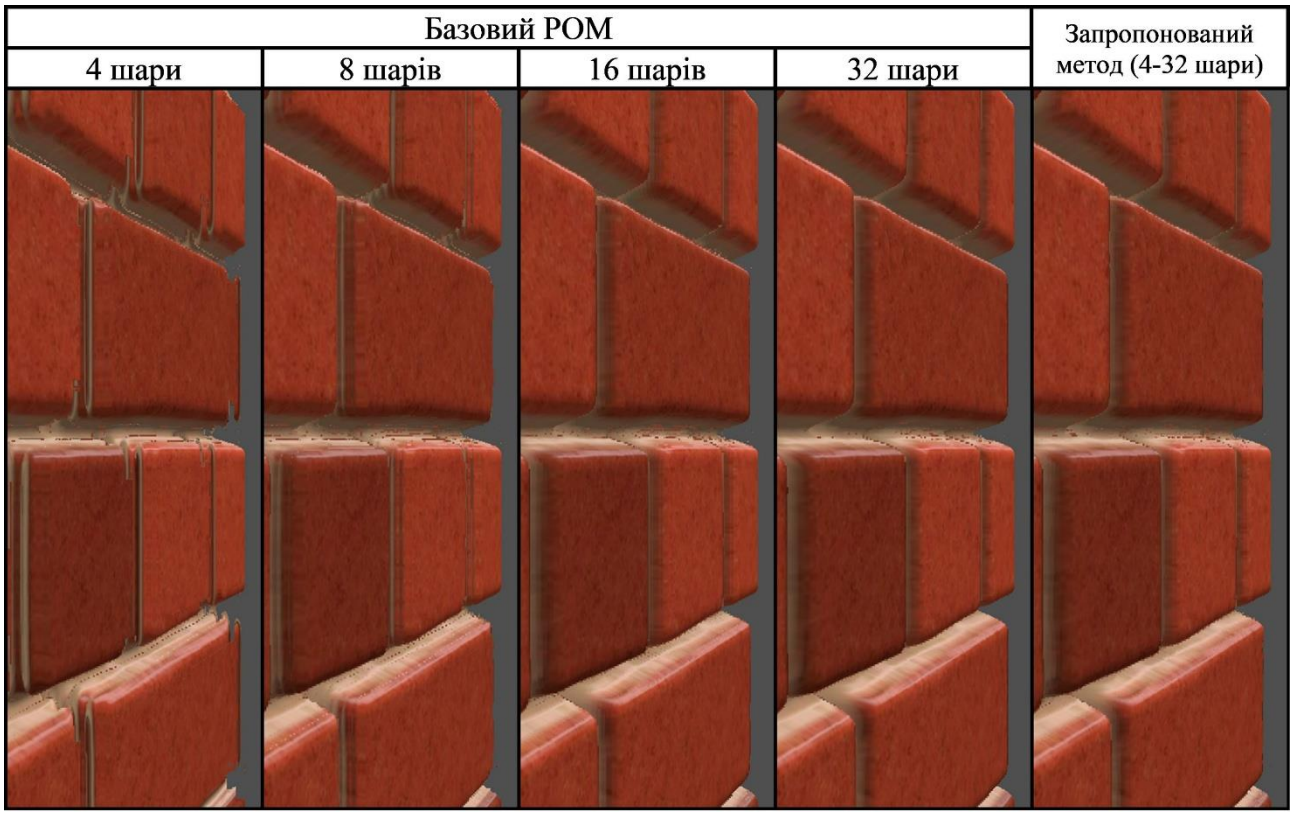


Рисунок 4.8 – Приклад застосування базового та запропонованого методу

Як бачимо, адаптивний підхід дозволяє досягти рівня деталізації, співставного з базовим POM з максимальною кількістю шарів, зберігаючи точність відображення текстури під різними кутами і на ділянках з різною складністю.

Також було проведено кількісний аналіз методів в контрольованому середовищі. Основною метрикою для порівняння було обрано частоту кадрів за секунду (FPS), яка безпосередньо відображає обчислювальне навантаження на графічний процесор і може слугувати показником продуктивності методів рельєфного текстурування.

Під час дослідження обидва методи застосовувалися за схожих умов: ідентичне обладнання, сцена та кут огляду, що забезпечує репрезентативність отриманих результатів. Були розраховані середні значення FPS для різних

конфігурацій базового методу POM і порівняні з показниками адаптивного методу на основі POM. Результати проведеного аналізу наведено у табл. 4.1.

Таблиця 4.1 – Порівняльний аналіз методів на основі показника FPS

Кут огляду	**Базовий POM**					**Запропонований метод (4-32 шари)**
	4 шари	**8 шарів**	**16 шарів**	**24 шари**	**32 шари**	
0°	1294	866	418	280	207	978
45°	571	265	121	80	62	162
60°	670	301	150	102	76	169
80°	1351	735	345	247	185	332
Середнє значення, $\bar{x}$	**972**	**542**	**259**	**177**	**133**	**410**
	331					

Враховуючи низьку візуальну точність при застосуванні 4 та 8 шарів дискретизації, загальне середнє значення FPS для базового POM було обчислено зі зменшенням ваги цих показників вдвічі порівняно з результатами для інших конфігурацій. Такий підхід дає змогу під час порівняння підкреслити сценарії, де якість відображення може конкурувати із покращеним методом:

$$\bar{x_1} = \frac{972 * 0{,}5 + 542 * 0{,}5 + 259 + 177 + 133}{4} \approx 331 \text{ кадрів/с},$$

$$\bar{x_2} = \frac{978 + 162 + 169 + 332}{4} = 410 \text{ кадрів/с},$$

де $\bar{x_1}$, $\bar{x_2}$ – середні значення показника FPS для базового та запропонованого методів відповідно. Розрахуємо досягнутий ступінь покращення ефективності $\Delta\eta$ порівняно з базовим POM:

$$\Delta\eta = \frac{\bar{x_2} - \bar{x_1}}{\bar{x_1}} * 100\% = \frac{410 - 331}{331} * 100\% \approx 24\%\,.$$

Отже, запропонований метод забезпечує середній приріст продуктивності близько 24% порівняно з традиційним підходом, досягаючи вищої ефективності рендерингу без погіршення візуальної якості.

4.4 Розробка інструкції користувача програмного застосунку

Для забезпечення ефективного використання розробленого застосунку було створено детальний посібник користувача, що описує всі основні функції та можливості програми.

При запуску TextureLab користувач потрапляє до головного меню, де відображається логотип програми та три основні кнопки керування: «Старт», «Налаштування» та «Вихід». Логотип програми, виконаний у мінімалістичному стилі, відображає спрямованість застосунку на роботу з текстурами у тривимірному просторі.

Перед початком роботи користувач може налаштувати параметри програми через меню «Налаштування». У цьому розділі доступні опції для вибору роздільної здатності екрану, перемикання повноекранного режиму та налаштування чутливості миші. Після встановлення бажаних параметрів необхідно натиснути кнопку «Застосувати» для збереження змін або «Скасувати» для повернення до попередніх налаштувань.

Натискання кнопки «Старт» відкриває головне вікно програми, де користувач може досліджувати різні методи рельєфного текстурування. У верхньому лівому куті екрану відображаються показники продуктивності: поточна кількість кадрів на секунду, а також мінімальне, середнє та максимальне значення FPS з моменту запуску програми.

Для навігації між різними методами текстурування користувач може використовувати клавіші «1-6» для прямого вибору конкретного методу або клавіші зі стрілками для послідовного перемикання між методами. При активації кожного методу на інформаційній панелі в нижній частині екрану з’являється детальний опис поточної техніки текстурування та її особливостей.

Для зручності перегляду текстур користувач може вільно рухатися по

тривимірній сцені за допомогою клавіатури. Клавіша «H» дозволяє приховати або показати інтерфейс користувача, «X» керує відображенням показників FPS, а «C» скидає статистику продуктивності. Довідка щодо призначення клавіш доступна у правому верхньому куті екрану.

У головному вікні програми представлені різні техніки текстурування, включаючи базове текстурування, normal mapping, parallax mapping, steep parallax mapping, parallax occlusion mapping та адаптивний метод на базі POM. Кожен метод демонструється на поверхнях із різними текстурами, що дозволяє користувачу візуально оцінити та порівняти різні підходи до текстурування.

Завершити роботу з програмою можна через кнопку «Вихід» у головному меню або стандартну кнопку закриття вікна. Усі налаштування зберігаються автоматично для наступного запуску програми.

4.5 Висновки

У четвертому розділі було здійснено аналіз підходів до тестування програмного забезпечення; для розробленого застосунку встановлено доцільність проведення ручного функціонального тестування методом «чорної скриньки». За результатами тестування було підтверджено повну відповідність розробленого застосунку функціональним вимогам та відсутність критичних помилок. Крім цього, було розроблено інструкцію користувача програмного забезпечення.

5 ЕКОНОМІЧНА ЧАСТИНА

Будь-яке дослідження має бути спрямоване не лише на досягнення технічної точності та інновацій, але й демонструвати економічну життєздатність в умовах конкурентного ринку. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота на тему «Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping» є науково-технічною роботою, що орієнтована на комерціалізацію розробки потенційним інвестором та виведення на ринок. Цей напрямок є пріоритетним, оскільки розроблені методи та програмне забезпечення мають потенціал для застосування в різних галузях комп’ютерної графіки. Для цього необхідно знайти потенційного інвестора, зацікавленого у підтримці проєкту, та ефективно донести до нього економічну доцільність такої інвестиції.

Таким чином, для оцінки економічної ефективності розробки мають бути виконані такі етапи робіт:

1) проведення комерційного аудиту науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розрахунок витрат на здійснення науково-технічної розробки;

3) розрахунок економічної ефективності науково-технічної розробки у випадку її впровадження та комерціалізації потенційним інвестором і обґрунтування економічної доцільності комерціалізації потенційним інвестором науково-технічної розробки.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [35]. Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів, якими можуть бути провідні викладачі випускової або спорідненої кафедри чи інші відомі фахівці.

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за п’ятибальною шкалою)					
	0	**1**	**2**	**3**	**4**
1	*2*	*3*	*4*	*5*	*6*
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижча за ціни аналогів	Ціна продукту значно нижча за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

1	*2*	*3*	*4*	*5*	*6*
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовують ся у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більший 5-ти років	Термін реалізації ідеї менший 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менший 3-х років. Термін окупності інвестицій менший 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що потребує значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту потребує незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки зведено до табл. 5.2.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	**Експерт**		
	1	**2**	**3**
	Бали:		
1. Технічна здійсненність концепції	3	4	3
2. Ринкові переваги (наявність аналогів)	4	4	2
3. Ринкові переваги (ціна продукту)	4	4	4
4. Ринкові переваги (технічні властивості)	2	2	1
5. Ринкові переваги (експлуатаційні витрати)	4	3	4
6. Ринкові перспективи (розмір ринку)	3	4	4
7. Ринкові перспективи (конкуренція)	2	2	2
8. Практична здійсненність (наявність фахівців)	4	4	3
9. Практична здійсненність (наявність фінансів)	4	3	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	**42**	**42**	**39**
Середньоарифметична сума балів	**41**		

За отриманими результатами необхідно зробити висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки, керуючись даними з табл. 5.3.

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	**Науково-технічний рівень та комерційний потенціал розробки**
41…48	Високий
31…40	Вищий середнього
21…30	Середній
11…20	Нижчий середнього
0…10	Низький

Згідно з проведеними дослідженнями рівень комерційного потенціалу розробки становить 41 бал, що, відповідно до табл. 5.3, свідчить про комерційну важливість проведення досліджень (високий рівень комерційного потенціалу

розробки). Такий рівень було досягнуто за рахунок суттєвого зростання продуктивності, ефективності нової науково-технічної розробки завдяки застосуванню розроблених методів та засобів, які дозволяють зменшити обчислювальні витрати порівняно з відомими методами.

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Важливо розрахувати витрати, пов’язані з проведенням науково-дослідної роботи, створенням дослідного зразка і здійсненням виробничих випробувань, під час планування, обліку і калькулювання собівартості роботи. Точна оцінка цих витрат дозволяє здійснити краще фінансове планування, гарантує, що розподіл ресурсів відповідає цілям проєкту, і забезпечує міцну основу для ціноутворення та аналізу прибутковості.

5.2.1 Витрати на оплату праці

До статті належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Витрати на основну заробітну плату дослідників ($З_о$) розраховуються відповідно до посадових окладів працівників за формулою:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p},$$

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

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

t_i – кількість днів роботи конкретного дослідника, дн.;

T_p – середня кількість робочих днів в місяці, T_p = 22 дні.

Розрахуємо витрати на заробітну плату керівника проєкту:

$$З_о = \frac{25\,000 * 60}{22} = 68\,182 \text{ грн.}$$

Аналогічно визначимо цей показник для інших дослідників. Проведені розрахунки зведено до табл. 5.4.

Таблиця 5.4 – Витрати на основну заробітну плату дослідників

Найменування посади	**Місячний посадовий оклад, грн**	**Оплата за робочий день, грн**	**Кількість днів роботи**	**Витрати на заробітну плату, грн**
Керівник проєкту	25 000	1 136,36	60	68 182
Інженер-розробник ПЗ	16 000	727,27	60	43 636
Тестувальник ПЗ	16 000	727,27	40	29 091
Консультант	18 000	818,18	60	49 091
Всього				**190 000**

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховуються за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i ,$$

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

t_i – час роботи робітника на виконання певної роботи, год.

Погодинна тарифна ставка робітника відповідного розряду C_i визначається за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}},$$

де M_M – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), грн;

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

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

T_p – середня кількість робочих днів в місяці, T_p = 22 дні;

$t_{зм}$ – тривалість зміни, год.

Тривалість зміни становить 8 годин; коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду до розміру мінімальної заробітної плати приймемо рівним 1,65 для всіх робітників; розмір мінімальної місячної заробітної плати – 8000 грн. Розрахуємо витрати на заробітну плату робітника для підготовки робочого місця дослідника:

$$C_1 = \frac{8\,000 * 1{,}1 * 1{,}65}{22 * 8} = 82{,}5 \text{ грн.},$$

$$З_p = 82{,}5 * 10 = 825 \text{ грн.}$$

Аналогічно визначимо цей показник для решти робітників. Проведені розрахунки зведено до табл. 5.5.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	**Тривалість роботи, год**	**Розряд роботи**	**Тарифний коефіцієнт**	**Погодинна тарифна ставка, грн.**	**Величина оплати на робітника, грн.**
Підготовка робочого місця дослідника	10	2	1,1	82,5	825
Інсталяція програмного забезпечення	5	5	1,7	637,5	637,5
Підготовка та організація наборів даних	2	3	1,35	101,25	202,5
Всього					**1 665**

Додаткову заробітну плату розраховуємо як 11% від суми основної заробітної плати дослідників та робітників за формулою:

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

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

$$З_{дод} = (190\,000 + 1\,665) * 0{,}11 = 21\,083 \text{ грн.}$$

5.2.2 Відрахування на соціальні заходи

До статті «Відрахування на соціальні заходи» належать відрахування внеску на загальнообов'язкове державне соціальне страхування та для здійснення заходів щодо соціального захисту населення (ЄСВ – єдиний соціальний внесок).

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

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

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

$$З_н = (190\,000 + 1\,665 + 21\,083) * 0{,}22 = 46\,805 \text{ грн.}$$

5.2.3 Сировина та матеріали

До статті належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби й предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (M) у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj},$$

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

n – кількість видів матеріалів;

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

K_j – коефіцієнт транспортних витрат, K_j = 1,1;

B_j – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

Розрахуємо витрати на офісний папір для друку проміжних звітів, фінальної документації, а також на проведення презентацій, де передбачається роздавати друковані матеріали для ознайомлення з результатами дослідження:

$$M_1 = 1 * 200 * 1{,}1 - 0 = 220 \text{ грн.}$$

Аналогічно визначимо цей показник для решти матеріалів. Проведені розрахунки зведено до табл. 5.6.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	**Ціна за од., грн**	**Норма витрат, од.**	**Величина відходів, кг**	**Ціна відходів, грн/кг**	**Вартість витраченого матеріалу, грн**
Папір офісний 80 г/м^2, 500 аркушів	200	1	0	0	220
Флеш пам’ять USB Samsung Bar Plus USB 3.1 64GB	617	1	0	0	679
Картриджі для принтера Canon Pixma MG5540 з чорнилом	1 063	1	0	0	1 169
Всього					**2 068**

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі вироби для науково-технічного дослідження на тему «Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті належать витрати на виготовлення та придбання спецустаткування, необхідного для проведення досліджень, а також витрати на їх проєктування, виготовлення, транспортування, монтаж та встановлення.

Балансова вартість спецустаткування розраховується за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ ,$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ – кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, K_i = 1,11;

k – кількість найменувань устаткування.

Розрахуємо витрати на спецустаткування для проведення досліджень, що включають ноутбук Lenovo Ideapad L340-15IRH (81LK00DARA):

$$B_{спец} = 20999 * 1 * 1{,}11 = 23\ 309 \text{ грн.}$$

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, необхідних для проведення досліджень, також витрати на їх проєктування, формування та встановлення.

Балансова вартість програмного забезпечення розраховується за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ ,$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, K_i = 1,11;

k – кількість найменувань програмних засобів.

Розрахуємо балансову вартість ліцензійної копії операційної системи Windows 11 Pro:

$$B_{прг} = 7\,899 * 1 * 1{,}11 = 8\,768 \text{ грн.}$$

Аналогічно визначимо цей показник для іншого програмного забезпечення, використаного під час досліджень. Проведені розрахунки зведено до табл. 5.7.

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	**Кількість, шт**	**Ціна за одиницю, грн**	**Вартість, грн**
Операційна система Windows 11 Pro	1	7 899	8 768
Офісний пакет Microsoft Office Home & Business 2024	2	10 327	22 926
Всього			**31 694**

5.2.7 Амортизація обладнання, програмних засобів та приміщень

До статті відносять амортизаційні відрахування по кожному виду обладнання, устаткування та інших приладів і пристроїв, а також програмного забезпечення для проведення науково-дослідної роботи.

В спрощеному вигляді амортизаційні відрахування по кожному виду устаткування розраховуються з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12},$$

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

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

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

Термін використання обладнання $t_{вик}$ приймемо рівним 3 місяцям. Розрахуємо амортизаційні відрахування для ноутбука Lenovo Ideapad L340-15IRH:

$$A_{обл} = \frac{23\,309}{5} * \frac{3}{12} = 1\,165 \text{ грн.}$$

Аналогічно розрахуємо амортизацію для решти обладнання. Проведені розрахунки зведено до табл. 5.8.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	**Балансова вартість, грн**	**Строк корисного використання, років**	**Термін використання обладнання, місяців**	**Амортизаційні відрахування, грн**
Ноутбук Lenovo Ideapad L340-15IRH	23 309	5	3	1 165
Операційна система Windows 11 Pro	8 768	5	3	438
Офісний пакет Microsoft Office Home & Business 2024	22 926	3	3	1 911
Приміщення лабораторії	250 000	20	3	3 125
Всього				**6 639**

5.2.8 Паливо та енергія для науково-виробничих цілей

До статті належать витрати на придбання у сторонніх підприємств, установ і організацій будь-якого палива, що витрачається з технологічною метою на проведення досліджень.

Витрати на силову електроенергію (B_e) розраховуються за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i},$$

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

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії);

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

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Розрахуємо поточну ціну на електроенергію з урахуванням податку на додану вартість за формулою:

$$Ц_e = (Ц_{опт} + Ц_{розп} + Ц_{пост}) \cdot (1 + \frac{ПДВ}{100\%}),$$

де $Ц_{опт}$ – середня оптова ціна електроенергії, яка визначається оператором ринку (без ПДВ), грн за 1 кВт·год;

$Ц_{розп}$ – вартість розподілу електроенергії окремою енергорозподільною компанією (без ПДВ), грн за 1 кВт·год;

$Ц_{пост}$ – вартість постачання електроенергії від енергорозподільної компанії до конкретного споживача (без ПДВ), грн за 1 кВт·год;

ПДВ – величина податку на додану вартість, %. Ставка ПДВ у 2024 році – 20%.

$$Ц_e = (5{,}307 + 0{,}529 + 2{,}028) * 1{,}2 = 9{,}44 \text{ грн.}$$

Коефіцієнт, що враховує використання потужності $K_{вні}$, приймемо рівним 0,8; коефіцієнт корисної дії обладнання η_i становить 0,85. Розрахуємо витрати на електроенергію, використану під час дослідження ноутбуком Lenovo Ideapad L340-15IRH:

$$B_e = \frac{0{,}14 * 480 * 9{,}44 * 0{,}8}{0{,}85} = 597 \text{ грн.}$$

Аналогічно визначимо витрати на електроенергію для решти обладнання. Проведені розрахунки зведено до табл. 5.9.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	**Встановлена потужність, кВт**	**Тривалість роботи, год**	**Сума, грн**
Ноутбук Lenovo Ideapad L340-15IRH	0,14	480	597,05
Робоче місце дослідника	0,009	480	38,38
Принтер Canon Pixma MG5540	0,011	10	0,98
Всього			**636,41**

5.2.9 Службові відрядження

До статті належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Службові витрати розраховуються як 20-25% від суми основної заробітної плати дослідників та робітників за формулою:

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

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

Розрахуємо витрати на службові відрядження як 20% від суми основної заробітної плати дослідників та робітників:

$$В_{св} = (190\,000 + 1\,665) * 0{,}2 = 38\,333 \text{ грн.}$$

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

До статті належать витрати на проведення досліджень, що не можуть бути виконані штатними працівниками або наявним обладнанням організації, а виконуються на договірній основі іншими підприємствами, установами і організаціями незалежно від форм власності та позаштатними працівниками.

Витрати за цією статтею розраховуються як 30-45% від суми основної заробітної плати дослідників та робітників за формулою:

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

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

Розрахуємо витрати на роботи, які виконують сторонні підприємства на службові відрядження як 35% від суми основної заробітної плати дослідників та робітників:

$$В_{сп} = (190\,000 + 1\,665) * 0{,}35 = 67\,082 \text{ грн.}$$

5.2.11 Інші витрати

До статті належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за цією статтею розраховуються як 50-100% від суми основної заробітної плати дослідників та робітників за формулою:

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

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

Розрахуємо величину інших витрат як 50% від суми основної заробітної плати дослідників та робітників:

$$I_в = (190\,000 + 1\,665) * 0{,}5 = 95\,833 \text{ грн.}$$

5.2.12 Накладні (загальновиробничі) витрати

До статті належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу тощо.

Витрати за цією статтею розраховуються як 100-150% від суми основної заробітної плати дослідників та робітників за формулою:

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

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

Розрахуємо величину накладних витрат як 100% від суми основної заробітної плати дослідників та робітників:

$$В_{нзв} = (190\,000 + 1\,665) * 1{,}0 = 191\,665 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}.$$

Розрахуємо загальні витрати на проведення науково-дослідної роботи:

$$B_{заг} = 190\,000 + 1\,665 + 21\,083 + 46\,805 + 2\,068 + 0 + 23\,309 +$$
$$+31\,694 + 6\,639 + 636{,}41 + 38\,333 + 67\,083 + 95\,833 + 191\,665 =$$
$$= 716\,812{,}46 \text{ грн.}$$

Загальні витрати *ЗВ* на завершення науково-дослідної роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{B_{заг}}{\eta},$$

де η – коефіцієнт, який характеризує стадію виконання науково-дослідної роботи.

Зважаючи на те, що науково-дослідна робота знаходиться на стадії розробки промислового зразка, значення коефіцієнта η оберемо рівним 0,7. Розрахуємо загальні витрати на завершення науково-дослідної роботи:

$$ЗВ = \frac{716\,803{,}41}{0{,}7} = 1\,024\,017{,}80 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

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

величини чистого прибутку [36]. Тому важливо розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки.

Науково-технічна розробка за темою «Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping» передбачає розробку програмного засобу для використання масовим споживачем. Основні позитивні результати для потенційного інвестора очікуються протягом 3-х років після впровадження розробки.

Майбутній економічний ефект буде формуватися на основі таких даних:

– ΔN – збільшення кількості споживачів продукту в аналізовані періоди часу від покращення його певних характеристик; приймемо збільшення кількості користувачів на 500, 1000, 2000 у 1, 2 та 3 роки відповідно;

– N – кількість споживачів, які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки; приймемо 100 000 користувачів;

– $Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки; приймемо 12 000 грн. – базова ціна реалізації одиниці продукції;

– $\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки в аналізовані періоди часу; приймемо зростання ціни реалізації одиниці нової розробки на 1000 грн..

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}),$$

де $Ц_о$ – основний якісний показник, який визначає ціну реалізації нової науково-

технічної розробки в аналізованому році, $Ц_о = Ц_б \pm \Delta Ц_о$ = 12 000 + 1000 = 13 000;

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. Ставка ПДВ у 2024 році – 20%, а коефіцієнт λ = 0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги); приймемо ρ = 0,5;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, У 2024 цей показник становить 18%, тому $1 - \frac{\vartheta}{100} = 0{,}82$.

Розрахуємо можливе збільшення чистого прибутку у потенційного інвестора протягом 3-х років після впровадження науково-технічної розробки:

$$\Delta П_1 = (1\,000 * 100\,000 + 13\,000 * 500) * 0{,}83 * 0{,}5 * 0{,}82 = 36\,241\,950 \text{ грн.},$$
$$\Delta П_2 = (1\,000 * 100\,000 + 13\,000 * 1500) * 0{,}83 * 0{,}5 * 0{,}82 = 40\,665\,850 \text{ грн.},$$
$$\Delta П_3 = (1\,000 * 100\,000 + 13\,000 * 3500) * 0{,}83 * 0{,}5 * 0{,}82 = 49\,513\,650 \text{ грн..}$$

Приведена вартість збільшення всіх чистих прибутків $ПП$, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t},$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки, Т = 3 роки.

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні. За урядовим прогнозом, рівень інфляції у 2025 році очкується на рівні 9,5%, тому τ =0,095;

t – період часу від моменту початку впровадження розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{36\,241\,950}{(1+0{,}095)^1} + \frac{40\,665\,850}{(1+0{,}095)^2} + \frac{49\,513\,650}{(1+0{,}095)^3} = 104\,725\,698{,}34 \text{ грн.}$$

Величина початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки, розраховується за формулою:

$$PV = k_{інв} \cdot ЗВ,$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію; приймемо k = 3.

$$PV = 3 * 1\,024\,017{,}80 = 3\,072\,053{,}39 \text{ грн.}$$

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

$$E_{абс} = ПП - PV = 104\,725\,698{,}34 - 3\,072\,053{,}39 = 101\,653\,644{,}96 \text{ грн.}$$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1,$$

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, $T_ж$ = 3 роки.

$$E_в = \sqrt[3]{1 + \frac{101\,653\,644{,}96}{3\,072\,053{,}39}} - 1 = 2{,}24$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d =0,13;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,2.

$$\tau_{мін} = 0{,}13 + 0{,}2 = 0{,}33$$

Оскільки величина $E_в > \tau_{мін}$, то потенційний інвестор може бути зацікавлений у фінансуванні впровадження науково-технічної розробки та виведенні її на ринок, тобто в її комерціалізації. Таким чином, інвестувати в науково-дослідну роботу на тему «Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$T_{ок} = \frac{1}{E_в} = \frac{1}{2{,}24} = 0{,}45 \text{ року}$$

Оскільки $T_{ок} < 3$ років, то це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

5.4 Висновки

Згідно з результатами комерційного та технологічного аудиту науково-технічної розробки рівень комерційного потенціалу розробки становить 41 бал, що свідчить про комерційну важливість проведення досліджень та високий рівень комерційного потенціалу розробки.

Загальні витрати на завершення науково-дослідної роботи та оформлення її результатів складають 1 024 017,80 грн., а період окупності після впровадження науково-технічної розробки при її можливій комерціалізації потенційним інвестором становить 0,45 року, що свідчить про комерційну привабливість науково-технічної розробки.

Отже, на основі проведених розрахунків економічної ефективності можна зробити висновок про доцільність проведення науково-технічної розробки на тему «Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping».

ВИСНОВКИ

У магістерській кваліфікаційній роботі було розроблено методи та програмний засіб для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping. Робота оформлена відповідно до методичних вказівок [37]. Розроблене ПЗ має на меті підвищення швидкодії процесу імітації нерівностей на поверхні за рахунок зменшення кількості обчислень.

Було проведено ґрунтовне дослідження сучасних методів формування зображень рельєфних поверхонь, таких як bump mapping, normal mapping, parallax mapping, steep parallax mapping, parallax occlusion mapping. Розглянуто два перспективні напрямки удосконалення методу parallax occlusion mapping: використання нейронних мереж та застосування аналізу карти висот для динамічного регулювання кількості шарів дискретизації. За результатами аналізу існуючих програмних рішень підтверджено актуальність розробки власної реалізації, що враховує виявлені недоліки та обмеження наявних продуктів. На основі порівняльного аналізу платформ розробки обґрунтовано вибір десктопного рішення як найбільш доцільного для реалізації поставлених завдань. Здійснено постановку задач, необхідних для розробки програмного продукту.

Запропоновано метод генерації карти складності з використанням оператора Собеля для обчислення градієнтів. Представлено удосконалений метод формування зображень рельєфних поверхонь, що базується на методі parallax occlusion mapping та дозволяє підвищити продуктивність рендерингу текстури до 24% за допомогою регулювання кількості шарів дискретизації на основі карти складності та кута огляду. Крім цього, було здійснено детальне архітектурне проєктування програмного засобу з використанням UML-діаграм та розроблено графічний інтерфейс користувача застосунку.

Обґрунтовано вибір технологічного стеку для реалізації програмного продукту, зокрема рушія jMonkeyEngine 3 та середовища розробки jME3 SDK.

Описано програмну реалізацію двох ключових компонентів системи: допоміжного програмного засобу для генерації карт складності та застосунку для демонстрації методів рельєфного текстурування. Особлива увага приділена технічним аспектам впровадження розробленого методу формування зображень на базі POM.

Проведено експериментальні дослідження для перевірки розробленого методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping. Зокрема, було перевірено працездатність шейдерів за допомогою програмного засобу RenderDoc, а також проведено візуальний та кількісний порівняльний аналіз результатів застосування базового та запропонованого методів. Визначено, що запропонований метод забезпечує середній приріст продуктивності близько 24% порівняно з традиційним підходом, досягаючи вищої ефективності рендерингу без погіршення візуальної якості.

Здійснено аналіз підходів до тестування програмного забезпечення; для розробленого застосунку встановлено доцільність проведення ручного функціонального тестування методом «чорної скриньки». За результатами тестування було підтверджено повну відповідність розробленого застосунку функціональним вимогам та відсутність критичних помилок. Крім цього, було розроблено інструкцію користувача програмного забезпечення.

Оцінка економічної ефективності науково-технічної розробки показала високий рівень комерційного потенціалу розробки із оцінкою у 41 бал, що підтверджує важливість проведених досліджень. Було розраховано загальні витрати на завершення науково-дослідної роботи, які склали 1 024 017,80 грн. Період окупності розробки після її впровадження при можливій комерціалізації потенційним інвестором становить 0,45 року, що свідчить про значну комерційну привабливість проєкту та економічну доцільність його реалізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. Комп’ютерна графіка : електронний навчальний посібник. Вінниця : ВНТУ, 2023. 147 с.

2. Tomas Akenine-Möller, Eric Haines, Naty Hoffman. Real-Time Rendering. Бока-Ратон, Флорида, США : CRC Press, 2019. 1045 с.

3. Steve Marschner, Peter Shirley. Fundamentals of Computer Graphics. Бока-Ратон, Флорида, США : CRC Press, 2021. 716 с.

4. Ковальчук С. І., Романюк О. В. Методи імітації нерівностей на поверхні графічних об`єктів при накладанні текстур : зб. матеріалів LIII науково-технічної конференції підрозділів ВНТУ. Вінниця : ВНТУ, 2024. URL: https:// conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20168 (дата звернення: 17.11.2024).

5. Ковальчук С.І., Романюк О.В. Напрямки удосконалення методу parallax occlusion mapping : зб. матеріалів IV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів. Одеса : Видавництво ОНТУ, 2024. с. 383-385.

6. Ковальчук С.І., Романюк О.В. Шейдерна реалізація методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping : Електронні інформаційні ресурси: створення, використання, доступ та управління. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 20-21 листопада 2024 р. – Суми/Вінниця: НІКО/КЗВО «Вінницька академія безперервної освіти», 2022. с. 76-79.

7. V. Scott Gordon, John L. Clevenger. Computer Graphics Programming in OpenGL with C++, Second Edition. Герндон, Вірджинія, США : Mercury Learning and Information, 2020. 514 с.

8. Atlas of Digital Architecture / за ред. Ludger Hovestadt, Oliver Fritz, Urs Hirschberg. Берлін : De Gruyter, 2020. С. 267.

9. Normal Mapping : веб-сайт. URL: https://learnopengl.com/Advanced-Lighting/Normal-Mapping (дата звернення: 15.11.2024).

10. Parallax Mapping : веб-сайт. URL: https://learnopengl.com/Advanced-Lighting/Parallax-Mapping (дата звернення: 15.11.2024).

11. ArmorPaint | 3D PBR Texture Painting : веб-сайт. URL: https://armorpaint.org (дата звернення: 15.11.2024).

12. PixPlant : веб-сайт. URL: https://www.pixplant.com (дата звернення: 15.11.2024).

13. UVMapper – UV Mapping Software : веб-сайт. URL: https://www.uvmapper.com (дата звернення: 15.11.2024).

14. Penny de Byl, Mathematics for Game Programming and Computer Graphics: Explore the Essential Mathematics for Creating, Rendering, and Manipulating 3D Virtual Environments. Німеччина : Packt Publishing, 2022. 444 с.

15. Leonardo De Marchi, Laura Mitchell, Hands-On Neural Networks: Learn how to Build and Train Your First Neural Network Model Using Python. Німеччина: Packt Publishing, 2019. 280 с.

16. Wilhelm Burger, Mark J. Burge. Digital Image Processing. США : Springer International Publishing, 2022. 945 с.

17. jMonkeyEngine Documentation : веб-сайт. URL: https://wiki.jmonkeyengine.org/docs/3.4/documentation.html (дата звернення: 16.11.2024).

18. Романюк О.Н., Іваха О.А., Дудник О.О. Аналіз шейдерів : зб. матеріалів XXI Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса : Видавництво ОНАХТ, 2021. С. 237-238.

19. Naeem Akbar Channar. A Comparative Study of Edge Detection Techniques in Digital Images. Німеччина : Bod Third Party Titles, 2020. 58 с.

20. Sobel Operator : веб-сайт. URL: https://leimao.github.io/blog/Sobel-Operator (дата звернення: 16.11.2024).

21. OpenCV: Eroding and Dilating : веб-сайт. URL: https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html (дата звернення: 16.11.2024).

22. Parallax Occlusion Map : веб-сайт. URL: https://online.ts2009.com/mediaWiki/index.php/Parallax_Occlusion_Map (дата звернення: 16.11.2024).

23. Tatarchuk N. Practical Dynamic Parallax Occlusion Mapping : International Conference on Computer Graphics and Interactive Techniques. 2005. URL: https://bit.ly/3Z8qJ2i (дата звернення: 17.11.2024).

24. What is the goal of UI design? Achieve Usability & User Engagement : веб-сайт. URL: https://www.linkedin.com/pulse/what-goal-ui-design-achieve-usability-user-engagement-aminul-islam-eaqsc (дата звернення: 17.11.2024).

25. Game Development Projects with Unreal Engine / H. Fozi, G. Marques, D. Pereira, D. Sherry. Велика Британія : Packt Publishing, 2020. 822 с.

26. Jiadong Chen, Ed Price. Game Development with Unity for .NET Developers: The Ultimate Guide to Creating Games with Unity and Microsoft Game Stack. Німеччина : Packt Publishing, 2022. 584 с.

27. Chris Bradfield. Godot 4 Game Development Projects. Велика Британія : Packt Publishing, 2023. 264 с.

28. Comparing the Top Java IDEs: Which One is Right for You? : веб-сайт. URL: https://parallelstaff.com/comparing-java-ides (дата звернення: 17.11.2024).

29. jMonkeyEngine SDK Documentation : веб-сайт. URL: https://wiki.jmonkeyengine.org/docs/3.4/sdk/sdk.html (дата звернення: 17.11.2024).

30. Sobel Derivatives : веб-сайт. URL: https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html (дата звернення: 17.11.2024).

31. What is a Vertex Shader in OpenGL? : веб-сайт. URL: https://www.haroldserrano.com/blog/what-is-a-vertex-shader-in-opengl (дата звернення: 17.11.2024).

32. A Closer Look At Parallax Occlusion Mapping : веб-сайт. URL: https://bit.ly/48TGBJ3 (дата звернення: 17.11.2024).

33. Sumanta Guha. Computer Graphics Through OpenGL. США : CRC Press, 2022. 702 с.

34. Котлярчук Д.В., Романюк О.В. Аналіз методів тестування інтерфейсу користувача : зб. матеріалів XLIX науково-технічної конференції підрозділів ВНТУ. Вінниця : ВНТУ, 2020. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/view/9764 (дата звернення: 17.11.2024).

35. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В.О. Козловський, О.Й. Лесько, В.В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

36. Кавецький В.В., Причепа І.В., Нікіфорова Л.О. Економічне обґрунтування інноваційних рішень : навчальний посібник. Вінниця : ВНТУ, 2016. 136 с.

37. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення» / Уклад. : О.Н. Романюк, Г.О. Черноволик. Вінниця : ВНТУ, 2024. 48 с.

ДОДАТКИ

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф.
О.Н. Романюк _____
19 вересня 2024 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доц. О.В. Романюк
"19" вересня 2024 р.
Виконав:
_____ студент гр. 1ПІ-23м С.І. Ковальчук
"19" вересня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping».

Галузь застосування – системи комп’ютерної графіки.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №310 від 17 вересня 2024 р. ректора ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є підвищення ефективності формування зображень рельєфних поверхонь за рахунок зменшення кількості обчислень.

Призначення роботи – розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР:

1. О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. Комп’ютерна графіка : електронний навчальний посібник. Вінниця : ВНТУ, 2023. 147 с.

2. Tomas Akenine-Möller, Eric Haines, Naty Hoffman. Real-Time Rendering. Бока-Ратон, Флорида, США : CRC Press, 2019. 1045 с.

3. Steve Marschner, Peter Shirley. Fundamentals of Computer Graphics. Бока-Ратон, Флорида, США : CRC Press, 2021. 716 с.

4. Wilhelm Burger, Mark J. Burge. Digital Image Processing. США : Springer International Publishing, 2022. 945 с.

5. Sobel Derivatives: веб-сайт. URL: https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html (дата звернення: 17.11.2024).

6. jMonkeyEngine SDK Documentation : веб-сайт. URL: https://wiki.jmonkeyengine.org/docs/3.4/sdk/sdk.html (дата звернення: 17.11.2024).

5. Технічні вимоги

Методи рельєфного текстурування – bump mapping, normal mapping, parallax mapping, steep parallax mapping, parallax occlusion mapping; метод виявлення контурів у цифровому зображенні – оператор Собеля; рушій візуалізації – jMonkeyEngine 3; середовище розробки – jMonkeyEngine SDK; мова програмування – Java; вхідні дані – текстури, параметри шейдерів, введення з клавіатури, дані про розташування джерела світла та спостерігача; вихідні дані – демонстрація роботи методів текстурування, показники продуктивності.

6. Конструктивні вимоги

Розроблене програмне забезпечення повинно відповідати естетичним та ергономічним вимогам, бути зручним в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану питання та постановка задач дослідження	20.09.24 – 27.09.24
2	Проєктування архітектури та структури застосунку	28.09.24 – 07.10.24
3	Розробка методу генерації карти складності текстури	08.10.24 – 15.10.24
4	Розробка методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping	16.10.24 – 23.10.24
5	Аналіз та вибір засобів для реалізації програмного продукту	24.10.24 – 31.10.24
6	Реалізація програмного засобу на основі розроблених методів	01.11.24 – 14.11.24
7	Дослідження ефективності розроблених методів і тестування програмного забезпечення	15.11.24 – 22.11.24
8	Економічна частина	23.11.24 – 30.11.24

10. Порядок контролю та прийняття

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедри згідно з графіком.

ДОДАТОК Б

Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping.**

Тип роботи: кваліфікаційна робота
Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ-23м
Науковий керівник: к.т.н., доц. каф. ПЗ. Романюк О.В.

Turnitin	
Оригінальність	**92%**
Схожість	8%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Розробка методів та програмного засобу для підвищення ефективності формування зображень рельєфних поверхонь на базі методу parallax occlusion mapping».

Автор _____ Ковальчук Сергій Ігорович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт (за потреби) _____ _____
(підпис) (прізвище, ініціали, посада)

ДОДАТОК В

Лістинг програми

complexity_calculator.py

```
import cv2
import numpy as np


# Function to calculate the gradient magnitude (complexity) and graduate it into
discrete levels
def calculate_complexity(input_height_map, dilation_iterations=1, num_levels=10,
gamma=0.8):
    gray = input_height_map[:, :, 0].astype(np.float32) / 255.0

    grad_x = cv2.Sobel(gray, cv2.CV_32F, 1, 0, ksize=3)
    grad_y = cv2.Sobel(gray, cv2.CV_32F, 0, 1, ksize=3)
    gradient_magnitude = np.sqrt(grad_x ** 2 + grad_y ** 2)

    # Normalize the gradient magnitude to the range [0, 1]
    max_gradient_value = np.max(gradient_magnitude)  # Find the maximum gradient value
    complexity_value = np.clip(gradient_magnitude / max_gradient_value, 0, 1)

    # Apply dilation to further widen high-complexity areas
    kernel = np.ones((3, 3), np.uint8)
    complexity_value = cv2.dilate(complexity_value, kernel,
iterations=dilation_iterations)

    # Apply gamma correction to emphasize low magnitudes
    complexity_value = np.power(complexity_value, gamma)

    # Quantize the final value into discrete levels
    step_size = 1.0 / (num_levels - 1)  # Step size between levels
    complexity_value = np.round(complexity_value / step_size) * step_size  # Quantize
to nearest level

    return complexity_value


input_image_path = input("Input path: ")
output_image_path = input("Output path: ")

height_map = cv2.imread(input_image_path, cv2.IMREAD_COLOR)

if height_map is None:
    print("Error: Could not load the height map.")
    exit()

complexity = calculate_complexity(height_map, dilation_iterations=5, num_levels=24)

graduated_complexity = (complexity * 255).astype(np.uint8)
height_map[:, :, 0] = graduated_complexity
height_map[:, :, 1] = graduated_complexity
height_map[:, :, 2] = graduated_complexity

cv2.imwrite(output_image_path, height_map)
print(f"Complexity map saved as {output_image_path}.")
```

Main.java

```
package com.mygame;

import com.jme3.app.SimpleApplication;
import com.jme3.renderer.RenderManager;
import com.jme3.scene.Node;
import com.jme3.system.AppSettings;
import com.jme3.bullet.BulletAppState;
import com.jme3.bullet.control.BetterCharacterControl;
import com.jme3.bullet.control.RigidBodyControl;
import com.jme3.input.KeyInput;
import com.jme3.input.controls.ActionListener;
import com.jme3.input.controls.KeyTrigger;
import com.jme3.material.Material;
import com.jme3.math.ColorRGBA;
import com.jme3.math.FastMath;
import com.jme3.math.Quaternion;
import com.jme3.math.Vector3f;
import com.jme3.niftygui.NiftyJmeDisplay;
import com.jme3.scene.CameraNode;
import com.jme3.scene.Geometry;
import com.jme3.scene.Spatial;
import com.jme3.scene.control.CameraControl;
import com.jme3.terrain.geomipmap.TerrainQuad;
import com.jme3.util.TangentBinormalGenerator;
import com.mygame.GUI.GUIController;
import com.mygame.GUI.SettingsController;
import static com.mygame.TextureMapping.NORMAL_MAPPING;
import static com.mygame.TextureMapping.PARALLAX_OCCLUSION_BASIC;
import de.lessvoid.nifty.Nifty;
import de.lessvoid.nifty.elements.render.TextRenderer;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.EnumSet;
import javax.imageio.ImageIO;

public class Main extends SimpleApplication implements ActionListener {
    private float numLayers = 32;

    private GUIController guiController;
    private Nifty nifty;

    private BulletAppState bulletAppState;
    private RigidBodyControl scenePhy;

    private Node playerNode;
    private BetterCharacterControl playerControl;
    private CameraNode camNode;

    private Vector3f walkDirection = new Vector3f(0,0,0);
    private Vector3f viewDirection = new Vector3f(0,0,1);
    private boolean rotateLeft = false, rotateRight = false, forward = false, backward
= false;
    private float moveSpeed = 8;

    private Geometry wall1;
    private Geometry wall2;
    private Geometry wall3;

    private TerrainQuad floor1;
    private TerrainQuad floor2;
```

```
    private Vector3f targetPlayerPosition;
    private Vector3f minBounds = new Vector3f(-7, 0, 1);
    private Vector3f maxBounds = new Vector3f(7, 0, 7);

    public static void main(String[] args) {
        Main app = new Main();
        AppSettings settings = new AppSettings(true);
        settings.setRenderer(AppSettings.LWJGL_OPENGL33);
        settings.setWindowSize(1280, 800);
        settings.setVSync(false);
        settings.setTitle("TextureLab");

        try {
            BufferedImage[] icons = {
                ImageIO.read(Main.class.getClassLoader().getResource("Icons/icon-
256.png")),
                ImageIO.read(Main.class.getClassLoader().getResource("Icons/icon-
128.png")),
                ImageIO.read(Main.class.getClassLoader().getResource("Icons/icon-
64.png")),
                ImageIO.read(Main.class.getClassLoader().getResource("Icons/icon-
32.png")),
                ImageIO.read(Main.class.getClassLoader().getResource("Icons/icon-
16.png"))
            };
            settings.setIcons(icons);
        }
        catch(IOException e) {
            e.printStackTrace();
        }
        app.setSettings(settings);

        app.setDisplayStatView(false);
        app.setDisplayFps(false);
        app.setShowSettings(false);
        app.start();
    }

    @Override
    public void simpleInitApp() {
        NiftyJmeDisplay niftyDisplay = NiftyJmeDisplay.newNiftyJmeDisplay(
            assetManager, inputManager, audioRenderer, guiViewPort
        );
        nifty = niftyDisplay.getNifty();
        guiViewPort.addProcessor(niftyDisplay);

        guiController = new GUIController(this);

        // Load XML file for menu and HUD
        nifty.fromXml("Interface/nifty.xml", "menuScreen", guiController, new
SettingsController(this, guiController));

        stateManager.attach(guiController);

        flyCam.setEnabled(false);

        Node scene = (Node) assetManager.loadModel("Scenes/newScene.j3o");
        TangentBinormalGenerator.generate(scene);

        wall1 = (Geometry) scene.getChild("wall1");
        wall2 = (Geometry) scene.getChild("wall2");
```

```
        wall3 = (Geometry) scene.getChild("wall3");

        floor1 = (TerrainQuad) scene.getChild("floor1");
        floor2 = (TerrainQuad) scene.getChild("floor2");

        ((Geometry) scene.getChild("borderBack")).setCullHint(Spatial.CullHint.Always);
        ((Geometry)
scene.getChild("borderFront")).setCullHint(Spatial.CullHint.Always);
        ((Geometry) scene.getChild("borderLeft")).setCullHint(Spatial.CullHint.Always);
        ((Geometry)
scene.getChild("borderRight")).setCullHint(Spatial.CullHint.Always);


        //Benchmarking
//        scene.detachChildNamed("wall1");
//        scene.detachChildNamed("wall3");
//        floor1.setCullHint(Spatial.CullHint.Always);
//        floor2.setCullHint(Spatial.CullHint.Always);
//        wall1.setCullHint(Spatial.CullHint.Always);
//        wall3.setCullHint(Spatial.CullHint.Always);

        bulletAppState = new BulletAppState();
        stateManager.attach(bulletAppState);
        bulletAppState.getPhysicsSpace().setGravity(new Vector3f(0, -9.81f, 0));

        scenePhy = new RigidBodyControl(0f);
        scene.addControl(scenePhy);
        bulletAppState.getPhysicsSpace().add(scene);

        playerNode = new Node("player");
        playerNode.setLocalTranslation(new Vector3f(0, 0, 7f)); //0 degrees
//        playerNode.setLocalTranslation(new Vector3f(-4.24f, 0, 5.24f)); //45 degrees
//        playerNode.setLocalTranslation(new Vector3f(-5.2f, 0, 4f)); //60 degrees
//        playerNode.setLocalTranslation(new Vector3f(-5.91f, 0, 2.04f)); //80 degrees


        rootNode.attachChild(playerNode);
        rootNode.attachChild(scene);

        playerControl = new BetterCharacterControl(1.5f, 4, 30f);
        playerControl.setJumpForce(new Vector3f(0, 100, 0));
        playerControl.setGravity(new Vector3f(0, -10, 0));
        playerNode.addControl(playerControl);
        bulletAppState.getPhysicsSpace().add(playerControl);

        camNode = new CameraNode("CamNode", cam);
        camNode.setControlDir(CameraControl.ControlDirection.SpatialToCamera);
        camNode.setLocalTranslation(new Vector3f(0, 1, 0));
        Quaternion quat = new Quaternion();
        quat.lookAt(new Vector3f(0, 0, 1).subtract(playerNode.getLocalTranslation()),
Vector3f.UNIT_Y);
        camNode.setLocalRotation(quat);
        playerNode.attachChild(camNode);
        camNode.setEnabled(true);

        inputManager.addMapping("Forward", new KeyTrigger(KeyInput.KEY_W));
        inputManager.addMapping("Back", new KeyTrigger(KeyInput.KEY_S));
        inputManager.addMapping("Rotate Left", new KeyTrigger(KeyInput.KEY_A));
        inputManager.addMapping("Rotate Right", new KeyTrigger(KeyInput.KEY_D));
        inputManager.addMapping("Jump", new KeyTrigger(KeyInput.KEY_SPACE));
        inputManager.addListener(this, "Rotate Left", "Rotate Right");
        inputManager.addListener(this, "Forward", "Back", "Jump");
```

```
        inputManager.addMapping("SwitchDebug", new KeyTrigger(KeyInput.KEY_B));
        inputManager.addMapping("NumLayersUp", new KeyTrigger(KeyInput.KEY_UP));
        inputManager.addMapping("NumLayersDown", new KeyTrigger(KeyInput.KEY_DOWN));
//        inputManager.addMapping("TeleportToRandomPos", new
KeyTrigger(KeyInput.KEY_T));
        inputManager.addListener(this, "SwitchDebug", "TeleportToRandomPos",
"NumLayersUp", "NumLayersDown");

        inputManager.addMapping("PreviousMaterial", new KeyTrigger(KeyInput.KEY_LEFT));
        inputManager.addMapping("NextMaterial", new KeyTrigger(KeyInput.KEY_RIGHT));

        inputManager.addMapping("TextureMapping1", new KeyTrigger(KeyInput.KEY_1));
        inputManager.addMapping("TextureMapping2", new KeyTrigger(KeyInput.KEY_2));
        inputManager.addMapping("TextureMapping3", new KeyTrigger(KeyInput.KEY_3));
        inputManager.addMapping("TextureMapping4", new KeyTrigger(KeyInput.KEY_4));
        inputManager.addMapping("TextureMapping5", new KeyTrigger(KeyInput.KEY_5));
        inputManager.addMapping("TextureMapping6", new KeyTrigger(KeyInput.KEY_6));

        inputManager.addListener(this, "PreviousMaterial", "NextMaterial",
                "TextureMapping1", "TextureMapping2", "TextureMapping3",
                "TextureMapping4", "TextureMapping5", "TextureMapping6");

        viewPort.setBackgroundColor(ColorRGBA.fromRGBA255(20, 20, 20, 255));

        // Initial set up
        Material brickPavementBasicPOM =
assetManager.loadMaterial("Materials/parallax_occlusion_mapping_basic/pom_basic_brick_p
avement.j3m");

        floor1.setMaterial(brickPavementBasicPOM);
        floor2.setMaterial(brickPavementBasicPOM);

        guiController.updateShaderInfo(PARALLAX_OCCLUSION_BASIC);
        guiController.setNumLayers((int) numLayers);
    }

    @Override
    public void simpleUpdate(float tpf) {
        // Get current forward and left vectors of the playerNode:
        Vector3f modelForwardDir = playerNode.getWorldRotation().mult(new
Vector3f(0,0,-1));
        Vector3f modelLeftDir = playerNode.getWorldRotation().mult(new Vector3f(-
1,0,0));

        // Determine the change in direction
        walkDirection.set(0, 0, 0);
        if (forward) {
             walkDirection.addLocal(modelForwardDir.mult(moveSpeed));
        } else if (backward) {
            walkDirection.addLocal(modelForwardDir.mult(moveSpeed).negate());
        }

        playerControl.setWalkDirection(walkDirection); // walk!
        // Determine the change in rotation
        if (rotateLeft) {
            Quaternion rotateL = new Quaternion().
            fromAngleAxis(FastMath.PI * tpf, Vector3f.UNIT_Y);
            rotateL.multLocal(viewDirection);
        } else if (rotateRight) {
            Quaternion rotateR = new Quaternion().fromAngleAxis(-FastMath.PI * tpf,
Vector3f.UNIT_Y);
```

```
            rotateR.multLocal(viewDirection);
        }
        playerControl.setViewDirection(viewDirection); // turn!
    }

    @Override
    public void simpleRender(RenderManager rm) {

    }

    @Override
    public void onAction(String binding, boolean isPressed, float tpf) {
        if (nifty.getCurrentScreen().getScreenId().equals("menuScreen")) return;

        switch (binding) {
            case "Rotate Left" -> rotateLeft = isPressed;
            case "Rotate Right" -> rotateRight = isPressed;
            case "Forward" -> forward = isPressed;
            case "Back" -> backward = isPressed;
            case "Jump" -> playerControl.jump();

            case "SwitchDebug" -> {
                if (wall1.getMaterial().getMaterialDef().getName()
                        .equals(TextureMapping.PARALLAX_OCCLUSION_ENHANCED.toString())
&& isPressed) {

                    boolean debugMode = (boolean)
wall1.getMaterial().getParam("DebugMode").getValue();
                    wall1.getMaterial().setBoolean("DebugMode", !debugMode);
                    wall2.getMaterial().setBoolean("DebugMode", !debugMode);
                    wall3.getMaterial().setBoolean("DebugMode", !debugMode);
                    floor1.getMaterial().setBoolean("DebugMode", !debugMode);
                    floor2.getMaterial().setBoolean("DebugMode", !debugMode);
                }
            }
//            case "TeleportToRandomPos" -> {
//                if (!isPressed) return;
//                setRandomPosition();
//            }

            case "NumLayersUp" -> {
                if (!isPressed || numLayers >= 50) return;

                numLayers++;
                updateShaderNumLayers();
                guiController.setNumLayers((int) numLayers);
            }

            case "NumLayersDown" -> {
                if (!isPressed || numLayers <= 1) return;

                numLayers--;
                updateShaderNumLayers();
                guiController.setNumLayers((int) numLayers);
            }

            case "PreviousMaterial" -> {
                if (!isPressed) return;
                setMaterial(

TextureMapping.getPrevValue(wall1.getMaterial().getMaterialDef().getName())
                );
```

```
            }
            case "NextMaterial" -> {
                if (!isPressed) return;
                setMaterial(

TextureMapping.getNextValue(wall1.getMaterial().getMaterialDef().getName())
                );
            }
            case "TextureMapping1" -> {
                if (!isPressed) return;
                setMaterial(TextureMapping.BASIC_TEXTURING);
            }
            case "TextureMapping2" -> {
                if (!isPressed) return;
                setMaterial(TextureMapping.NORMAL_MAPPING);
            }
            case "TextureMapping3" -> {
                if (!isPressed) return;
                setMaterial(TextureMapping.PARALLAX_MAPPING);
            }
            case "TextureMapping4" -> {
                if (!isPressed) return;
                setMaterial(TextureMapping.STEEP_PARALLAX_MAPPING);
            }
            case "TextureMapping5" -> {
                if (!isPressed) return;
                setMaterial(TextureMapping.PARALLAX_OCCLUSION_BASIC);
            }
            case "TextureMapping6" -> {
                if (!isPressed) return;
                setMaterial(TextureMapping.PARALLAX_OCCLUSION_ENHANCED);
            }

            default -> {}
        }
    }

    private void updateShaderNumLayers() {
        EnumSet<TextureMapping> adjustableLayersMappings =
               EnumSet.of(
                       TextureMapping.STEEP_PARALLAX_MAPPING,
                       TextureMapping.PARALLAX_OCCLUSION_BASIC
               );

        if (!adjustableLayersMappings.contains(

TextureMapping.valueOf(wall1.getMaterial().getMaterialDef().getName())))
            return;

        wall1.getMaterial().setFloat("NumLayers", numLayers);
        wall2.getMaterial().setFloat("NumLayers", numLayers);
        wall3.getMaterial().setFloat("NumLayers", numLayers);
        floor1.getMaterial().setFloat("NumLayers", numLayers);
        floor2.getMaterial().setFloat("NumLayers", numLayers);
    }

    private void setMaterial(TextureMapping mapping) {
        switch(mapping) {
            case PARALLAX_OCCLUSION_BASIC -> {

wall1.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_basic
/pom_basic_sand_rocks.j3m"));
```

```
wall2.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_basic
/pom_basic_bricks.j3m"));

wall3.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_basic
/pom_basic_bark_willow.j3m"));


floor1.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_basi
c/pom_basic_brick_pavement.j3m"));

floor2.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_basi
c/pom_basic_brick_pavement.j3m"));
            }
            case PARALLAX_OCCLUSION_ENHANCED -> {

wall1.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_enhan
ced/pom_enhanced_sand_rocks.j3m"));

wall2.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_enhan
ced/pom_enhanced_bricks.j3m"));

wall3.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_enhan
ced/pom_enhanced_bark_willow.j3m"));


floor1.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_enha
nced/pom_enhanced_brick_pavement.j3m"));

floor2.setMaterial(assetManager.loadMaterial("Materials/parallax_occlusion_mapping_enha
nced/pom_enhanced_brick_pavement.j3m"));
            }
            case BASIC_TEXTURING -> {

wall1.setMaterial(assetManager.loadMaterial("Materials/basic_texturing/basic_texturing_
sand_rocks.j3m"));

wall2.setMaterial(assetManager.loadMaterial("Materials/basic_texturing/basic_texturing_
bricks.j3m"));

wall3.setMaterial(assetManager.loadMaterial("Materials/basic_texturing/basic_texturing_
bark_willow.j3m"));


floor1.setMaterial(assetManager.loadMaterial("Materials/basic_texturing/basic_texturing
_brick_pavement.j3m"));

floor2.setMaterial(assetManager.loadMaterial("Materials/basic_texturing/basic_texturing
_brick_pavement.j3m"));
            }
            case NORMAL_MAPPING -> {

wall1.setMaterial(assetManager.loadMaterial("Materials/normal_mapping/normal_mapping_sa
nd_rocks.j3m"));

wall2.setMaterial(assetManager.loadMaterial("Materials/normal_mapping/normal_mapping_br
icks.j3m"));

wall3.setMaterial(assetManager.loadMaterial("Materials/normal_mapping/normal_mapping_ba
rk_willow.j3m"));
```

```
floor1.setMaterial(assetManager.loadMaterial("Materials/normal_mapping/normal_mapping_b
rick_pavement.j3m"));

floor2.setMaterial(assetManager.loadMaterial("Materials/normal_mapping/normal_mapping_b
rick_pavement.j3m"));
            }
            case PARALLAX_MAPPING -> {

wall1.setMaterial(assetManager.loadMaterial("Materials/parallax_mapping/parallax_mappin
g_sand_rocks.j3m"));

wall2.setMaterial(assetManager.loadMaterial("Materials/parallax_mapping/parallax_mappin
g_bricks.j3m"));

wall3.setMaterial(assetManager.loadMaterial("Materials/parallax_mapping/parallax_mappin
g_bark_willow.j3m"));


floor1.setMaterial(assetManager.loadMaterial("Materials/parallax_mapping/parallax_mappi
ng_brick_pavement.j3m"));

floor2.setMaterial(assetManager.loadMaterial("Materials/parallax_mapping/parallax_mappi
ng_brick_pavement.j3m"));
            }
            case STEEP_PARALLAX_MAPPING -> {

wall1.setMaterial(assetManager.loadMaterial("Materials/steep_parallax_mapping/steep_par
allax_mapping_sand_rocks.j3m"));

wall2.setMaterial(assetManager.loadMaterial("Materials/steep_parallax_mapping/steep_par
allax_mapping_bricks.j3m"));

wall3.setMaterial(assetManager.loadMaterial("Materials/steep_parallax_mapping/steep_par
allax_mapping_bark_willow.j3m"));


floor1.setMaterial(assetManager.loadMaterial("Materials/steep_parallax_mapping/steep_pa
rallax_mapping_brick_pavement.j3m"));

floor2.setMaterial(assetManager.loadMaterial("Materials/steep_parallax_mapping/steep_pa
rallax_mapping_brick_pavement.j3m"));
            }
            default -> {}
        }

        guiController.updateShaderInfo(mapping);
        updateShaderNumLayers();
    }

    // Helper method to generate a random position within the world bounds
    private Vector3f generateRandomPosition() {
        float x = FastMath.nextRandomFloat() * (maxBounds.x - minBounds.x) +
minBounds.x;
        float z = FastMath.nextRandomFloat() * (maxBounds.z - minBounds.z) +
minBounds.z;
        return new Vector3f(x, 0, z); // Y is kept at 0 for flat ground
    }

    private void setRandomPosition() {
        targetPlayerPosition = generateRandomPosition();
        playerControl.warp(targetPlayerPosition);
```

```
        // Make the player look at the world's origin (0,0,0)
        Vector3f lookAtTarget = new Vector3f(0, 0,
0).subtract(targetPlayerPosition.normalizeLocal());
        Quaternion quat = new Quaternion();
        quat.lookAt(lookAtTarget, Vector3f.UNIT_Y);
        camNode.setLocalRotation(quat);
    }
}
```

TextureMapping.java

```
package com.mygame;


public enum TextureMapping {
    PARALLAX_OCCLUSION_BASIC("Parallax occlusion mapping", "Вдосконалена техніка, яка
ще більше покращує parallax mapping, яка імітує глибину поверхні шляхом трасування
променів через карту висот, щоб визначити, де вектор напрямку погляду глядача
перетинається з поверхнею. На відміну від steep parallax mapping, POM виконує точніші
перевірки перетину, використовуючи лінійну інтерполяцію. Це забезпечує вищу візуальну
точність, особливо для складних поверхонь, але за рахунок збільшення обчислювальних
витрат", "Interface/mappings/parallax_occlusion_basic.png"),
    PARALLAX_OCCLUSION_ENHANCED("Адаптивний POM", "Вдосконалений метод POM, який
дозволяє зменшити обчислювальні витрати на текстурування шляхом динамічного вибору
кількості шарів дискретизації діапазону глибин текстури залежно від складності її
конкретних ділянок. Ключовою складовою покращеного методу є використання карти
складності під час накладання текстури. Це дозволяє більш ефективно розподіляти
обчислювальні ресурси, зменшуючи середню кількість шарів дискретизації по текстурі та
зберігаючи при цьому якість ділянок, де деталізація є найбільш важливою",
"Interface/mappings/parallax_occlusion_enhanced.png"),
    BASIC_TEXTURING("Базове текстурування", "Процес накладання 2D-зображення (текстури)
на поверхню 3D-моделі, щоб визначити її зовнішній вигляд. Текстура накладається на
геометрію моделі за допомогою UV-координат. Хоча базове текстурування додає
реалістичності, йому бракує глибини та взаємодії з освітленням, які надають просунуті
методи", "Interface/mappings/basic_texturing.png"),
    NORMAL_MAPPING("Normal mapping", "Техніка текстурування, яка імітує деталі
поверхні, змінюючи спосіб взаємодії світла з 3D-моделлю. Замість того, щоб додавати
реальну геометрію, вона використовує карту нормалей (зображення, що зберігає
попіксельні дані про напрямок поверхні), щоб створити ілюзію складності. Це робить
normal mapping високоефективним способом підвищення візуальної деталізації без
збільшення кількості полігонів, покращуючи як продуктивність, так і естетику",
"Interface/mappings/normal_mapping.png"),
    PARALLAX_MAPPING("Parallax mapping", "Техніка текстурування, яка імітує глибину на
пласкій поверхні шляхом зміщення координат текстури залежно від розташування глядача.
Вона створює ілюзію тривимірних деталей, таких як заглиблення або підвищення, без
додавання додаткової геометрії. Така візуалізація вирізняється більшою реалістичністю,
оскільки враховує зміни перспективи, хоча й може бути більш затратною в обчислювальному
плані", "Interface/mappings/parallax_mapping.png"),
    STEEP_PARALLAX_MAPPING("Steep parallax mapping", "Вдосконалена версія parallax
mapping, яка використовує ітеративну вибірку вздовж напрямку погляду для створення
більш вираженого ефекту глибини. Більша реалістичність забезпечується завдяки більш
точному моделюванню глибоких деталей поверхні, особливо під крутими кутами огляду,
зменшуючи візуальні артефакти, характерні для простіших методів, однак це
супроводжується збільшенням обчислювальних витрат",
"Interface/mappings/steep_parallax_mapping.png");

    private final String name;
    private final String description;
    private final String imagePath;

    private TextureMapping(String name, String description, String imagePath) {
        this.name = name;
```

```
        this.description = description;
        this.imagePath = imagePath;
    }

    public static TextureMapping getNextValue(String currentValue) {
        TextureMapping[] values = TextureMapping.values();
        TextureMapping value = TextureMapping.valueOf(currentValue);
        int nextIndex = (value.ordinal() + 1) % values.length;
        return values[nextIndex];
    }

    public static TextureMapping getPrevValue(String currentValue) {
        TextureMapping[] values = TextureMapping.values();
        TextureMapping value = TextureMapping.valueOf(currentValue);
        int previousIndex = (value.ordinal() - 1 + values.length) % values.length;
        return values[previousIndex];
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return description;
    }

    public String getImagePath() {
        return imagePath;
    }
}
```

FpsTracker.java

```
package com.mygame;

public class FpsTracker {
    private int minFps = Integer.MAX_VALUE;
    private int maxFps = 0;
    private int totalFps = 0;
    private int frameCount = 0;

    public void update(int fps) {
        frameCount++;
        totalFps += fps;

        if (fps < minFps) minFps = fps;
        if (fps > maxFps) maxFps = fps;
    }

    public int getMinFps() { return minFps; }
    public int getMaxFps() { return maxFps; }
    public int getAvgFps() { return frameCount == 0 ? 0 : totalFps / frameCount; }

    public void reset() {
        minFps = Integer.MAX_VALUE;
        maxFps = 0;
        totalFps = 0;
        frameCount = 0;
    }
}
```

parallax_occlusion_enhanced.vert

```
in vec3 inPosition;
in vec3 inNormal;
in vec2 inTexCoord;
in vec3 inTangent;

uniform mat4 g_WorldViewProjectionMatrix;
uniform mat4 g_WorldMatrix;
uniform mat4 g_ViewMatrix;

uniform vec3 g_LightPosition;
uniform vec3 g_CameraPosition;

out vec3 vPosition;
out vec2 vTexCoord;
// For non-directional lights
// out vec3 vTangentLightPos;
// out vec3 vTangentFragPos;
out vec3 vTangentViewPos;

void main() {
    vec3 worldPosition = (g_WorldMatrix * vec4(inPosition, 1.0)).xyz;
    vPosition = worldPosition;
    vTexCoord = inTexCoord;

    vec3 T = normalize((g_WorldMatrix * vec4(inTangent, 0.0)).xyz);
    vec3 B = normalize(cross(inNormal, T));
    vec3 N = normalize((g_WorldMatrix * vec4(inNormal, 0.0)).xyz);
    mat3 TBN = transpose(mat3(T, B, N));

    // For non-directional lights
    // vTangentLightPos = TBN * (g_LightPosition - worldPosition);
    // vTangentFragPos = TBN * worldPosition;
    vTangentViewPos = TBN * (g_CameraPosition - worldPosition);

    gl_Position = g_WorldViewProjectionMatrix * vec4(inPosition, 1.0);
}
```

parallax_occlusion_mapping.frag

```
out vec4 fragColor;

uniform sampler2D m_DiffuseMap;
uniform sampler2D m_NormalMap;
uniform sampler2D m_DepthMap;
uniform sampler2D m_ComplexMap;

uniform float m_HeightScale;
uniform bool m_DebugMode;
uniform float m_TextureScale;

uniform float m_MinLayers;
uniform float m_MaxLayers;

in vec2 vTexCoord;
// For non-directional lights
// in vec3 vTangentLightPos;
// in vec3 vTangentFragPos;
in vec3 vTangentViewPos;
in vec3 vTangentFragPos;
```

```
uniform vec4 g_LightPosition; // Directional light direction

float numLayers = 32;

vec2 ParallaxMapping(vec2 texCoords, vec3 viewDir) {
    float angleFactor = pow(1.0 - abs(dot(normalize(viewDir), vec3(0.0, 0.0, 1.0))),
0.8);

    // Збільшення кількості шарів при гострих кутах
    float adjustedMinLayers = mix(m_MinLayers, m_MaxLayers / 2, angleFactor);
    float adjustedMaxLayers = mix(m_MinLayers, m_MaxLayers, angleFactor);

    float complexity = texture(m_ComplexMap, texCoords).r;

    numLayers = int(mix(adjustedMinLayers, adjustedMaxLayers, complexity)); //
Кількість шарів з урахуванням складності

    float layerDepth = 1.0 / numLayers;
    float currentLayerDepth = 0.0;

    vec2 P = viewDir.xy / viewDir.z * m_HeightScale;
    vec2 deltaTexCoords = P / numLayers;

    vec2 currentTexCoords = texCoords;
    float currentDepthMapValue = texture(m_DepthMap, texCoords).r;

    while (currentLayerDepth < currentDepthMapValue) {
        currentTexCoords -= deltaTexCoords;
        currentDepthMapValue = texture(m_DepthMap, currentTexCoords).r;
        currentLayerDepth += layerDepth;
    }

    vec2 prevTexCoords = currentTexCoords + deltaTexCoords;

    float afterDepth = currentDepthMapValue - currentLayerDepth;
    float beforeDepth = texture(m_DepthMap, prevTexCoords).r - currentLayerDepth +
layerDepth;

    float weight = afterDepth / (afterDepth - beforeDepth);
    vec2 finalTexCoords = prevTexCoords * weight + currentTexCoords * (1.0 - weight);

    return finalTexCoords;
}

void main() {
    vec3 viewDir = normalize(vTangentViewPos - vTangentFragPos);

    vec2 texCoords;
    if (m_TextureScale == 1.0) {
        texCoords = ParallaxMapping(vTexCoord, viewDir);

        if (texCoords.x > 1.0 || texCoords.y > 1.0 || texCoords.x < 0.0 || texCoords.y
< 0.0)
        discard;
    }
    else {
        vec2 scaledTexCoords = vTexCoord * m_TextureScale;
        texCoords = fract(ParallaxMapping(scaledTexCoords, viewDir));
    }

    vec3 normal = texture(m_NormalMap, texCoords).rgb;
    normal = normalize(normal * 2.0 - 1.0);
```

```
    vec3 color = texture(m_DiffuseMap, texCoords).rgb;
    vec3 ambient = 0.2 * color;
    // For non-directional lights
    // vec3 lightDir = normalize(vTangentLightPos - vTangentFragPos);
    vec3 lightDir = normalize(g_LightPosition.xyz);
    float diff = max(dot(lightDir, normal), 0.0);
    vec3 diffuse = diff * color;

    vec3 reflectDir = reflect(-lightDir, normal);
    vec3 halfwayDir = normalize(lightDir + viewDir);
    float spec = pow(max(dot(normal, halfwayDir), 0.0), 32.0);
    vec3 specular = vec3(0.2) * spec;

    if (m_DebugMode) {
        float numLayersNormalized = float(numLayers - m_MinLayers) / float(m_MaxLayers
-  m_MinLayers);
        fragColor = vec4(vec3(numLayersNormalized), 1.0);
    }
    else {
        fragColor = vec4(ambient + diffuse + specular, 1.0);
    }
}
```

parallax_occlusion_enhanced.j3m

```
MaterialDef PARALLAX_OCCLUSION_ENHANCED {
    MaterialParameters {
        Texture2D DiffuseMap
        Texture2D NormalMap
        Texture2D DepthMap
        Texture2D ComplexMap
        Float HeightScale
        Boolean DebugMode: false
        Float TextureScale: 1.0
        Float MinLayers: 4.0
        Float MaxLayers: 32.0
    }

    Technique {
        LightMode MultiPass

        VertexShader   GLSL330:
Materials/parallax_occlusion_mapping_enhanced/parallax_occlusion_enhanced.vert
        FragmentShader GLSL330:
Materials/parallax_occlusion_mapping_enhanced/parallax_occlusion_enhanced.frag

        WorldParameters {
            WorldViewProjectionMatrix
            WorldMatrix
            ViewMatrix
            LightPosition
            CameraPosition
        }
    }
}
```

parallax_occlusion_basic.vert

```
in vec3 inPosition;
in vec3 inNormal;
in vec2 inTexCoord;
```

```
in vec3 inTangent;

uniform mat4 g_WorldViewProjectionMatrix;
uniform mat4 g_WorldMatrix;
uniform mat4 g_ViewMatrix;

uniform vec3 g_LightPosition;
uniform vec3 g_CameraPosition;

out vec3 vPosition;
out vec2 vTexCoord;
// For non-directional lights
// out vec3 vTangentLightPos;
// out vec3 vTangentFragPos;
out vec3 vTangentViewPos;

void main() {
    vec3 worldPosition = (g_WorldMatrix * vec4(inPosition, 1.0)).xyz;
    vPosition = worldPosition;
    vTexCoord = inTexCoord;

    vec3 T = normalize((g_WorldMatrix * vec4(inTangent, 0.0)).xyz);
    vec3 B = normalize(cross(inNormal, T));
    vec3 N = normalize((g_WorldMatrix * vec4(inNormal, 0.0)).xyz);
    mat3 TBN = transpose(mat3(T, B, N));

    // For non-directional lights
    // vTangentLightPos = TBN * (g_LightPosition - worldPosition);
    // vTangentFragPos = TBN * worldPosition;
    vTangentViewPos = TBN * (g_CameraPosition - worldPosition);

    gl_Position = g_WorldViewProjectionMatrix * vec4(inPosition, 1.0);
}
```

parallax_occlusion_basic.frag

```
out vec4 fragColor;

uniform sampler2D m_DiffuseMap;
uniform sampler2D m_NormalMap;
uniform sampler2D m_DepthMap;

uniform float m_HeightScale;
uniform float m_NumLayers;
uniform float m_TextureScale;

in vec2 vTexCoord;
// For non-directional lights
// in vec3 vTangentLightPos;
// in vec3 vTangentFragPos;
in vec3 vTangentViewPos;
in vec3 vTangentFragPos;

uniform vec4 g_LightPosition; // Directional light direction

vec2 ParallaxMapping(vec2 texCoords, vec3 viewDir) {
    float layerDepth = 1.0 / m_NumLayers;
    float currentLayerDepth = 0.0;

    vec2 P = viewDir.xy / viewDir.z * m_HeightScale;
    vec2 deltaTexCoords = P / m_NumLayers;
```

```
    vec2 currentTexCoords = texCoords;
    float currentDepthMapValue = texture(m_DepthMap, currentTexCoords).r;

    while (currentLayerDepth < currentDepthMapValue) {
        currentTexCoords -= deltaTexCoords;
        currentDepthMapValue = texture(m_DepthMap, currentTexCoords).r;
        currentLayerDepth += layerDepth;
    }

    vec2 prevTexCoords = currentTexCoords + deltaTexCoords;

    float afterDepth = currentDepthMapValue - currentLayerDepth;
    float beforeDepth = texture(m_DepthMap, prevTexCoords).r - currentLayerDepth +
layerDepth;

    float weight = afterDepth / (afterDepth - beforeDepth);
    vec2 finalTexCoords = prevTexCoords * weight + currentTexCoords * (1.0 - weight);

    return finalTexCoords;
}

void main() {
    vec3 viewDir = normalize(vTangentViewPos - vTangentFragPos);

    vec2 texCoords;
    if (m_TextureScale == 1.0) {
        texCoords = ParallaxMapping(vTexCoord, viewDir);

        if (texCoords.x > 1.0 || texCoords.y > 1.0 || texCoords.x < 0.0 || texCoords.y
< 0.0)
        discard;
    }
    else {
        vec2 scaledTexCoords = vTexCoord * m_TextureScale;
        texCoords = fract(ParallaxMapping(scaledTexCoords, viewDir));
    }

    vec3 normal = texture(m_NormalMap, texCoords).rgb;
    normal = normalize(normal * 2.0 - 1.0);

    vec3 color = texture(m_DiffuseMap, texCoords).rgb;
    vec3 ambient = 0.2 * color;
    // For non-directional lights
    // vec3 lightDir = normalize(vTangentLightPos - vTangentFragPos);
    vec3 lightDir = normalize(g_LightPosition.xyz);
    float diff = max(dot(lightDir, normal), 0.0);
    vec3 diffuse = diff * color;

    vec3 reflectDir = reflect(-lightDir, normal);
    vec3 halfwayDir = normalize(lightDir + viewDir);
    float spec = pow(max(dot(normal, halfwayDir), 0.0), 32.0);
    vec3 specular = vec3(0.2) * spec;

    fragColor = vec4(ambient + diffuse + specular, 1.0);
}
```

parallax_occlusion_basic.j3m

```
MaterialDef PARALLAX_OCCLUSION_BASIC {
    MaterialParameters {
        Texture2D DiffuseMap
        Texture2D NormalMap
```

```
        Texture2D DepthMap
        Float HeightScale
        Float NumLayers
        Float TextureScale: 1.0
    }

    Technique {
        LightMode MultiPass

        VertexShader   GLSL330:
Materials/parallax_occlusion_mapping_basic/parallax_occlusion_basic.vert
        FragmentShader GLSL330:
Materials/parallax_occlusion_mapping_basic/parallax_occlusion_basic.frag

        WorldParameters {
            WorldViewProjectionMatrix
            WorldMatrix
            ViewMatrix
            LightPosition
            CameraPosition
        }
    }
}
```

steep_parallax_mapping.vert

```
in vec3 inPosition;
in vec3 inNormal;
in vec2 inTexCoord;
in vec3 inTangent;

uniform mat4 g_WorldViewProjectionMatrix;
uniform mat4 g_WorldMatrix;
uniform mat4 g_ViewMatrix;

uniform vec3 g_LightPosition;
uniform vec3 g_CameraPosition;

out vec3 vPosition;
out vec2 vTexCoord;
// For non-directional lights
// out vec3 vTangentLightPos;
// out vec3 vTangentFragPos;
out vec3 vTangentViewPos;

void main() {
    vec3 worldPosition = (g_WorldMatrix * vec4(inPosition, 1.0)).xyz;
    vPosition = worldPosition;
    vTexCoord = inTexCoord;

    vec3 T = normalize((g_WorldMatrix * vec4(inTangent, 0.0)).xyz);
    vec3 B = normalize(cross(inNormal, T));
    vec3 N = normalize((g_WorldMatrix * vec4(inNormal, 0.0)).xyz);
    mat3 TBN = transpose(mat3(T, B, N));

    // For non-directional lights
    // vTangentLightPos = TBN * (g_LightPosition - worldPosition);
    // vTangentFragPos = TBN * worldPosition;
    vTangentViewPos = TBN * (g_CameraPosition - worldPosition);

    gl_Position = g_WorldViewProjectionMatrix * vec4(inPosition, 1.0);
}
```

steep_parallax_mapping.frag

```
out vec4 fragColor;

uniform sampler2D m_DiffuseMap;
uniform sampler2D m_NormalMap;
uniform sampler2D m_DepthMap;

uniform float m_HeightScale;
uniform float m_NumLayers;
uniform float m_TextureScale;

in vec2 vTexCoord;
// For non-directional lights
// in vec3 vTangentLightPos;
// in vec3 vTangentFragPos;
in vec3 vTangentViewPos;
in vec3 vTangentFragPos;

uniform vec4 g_LightPosition; // Directional light direction

vec2 ParallaxMapping(vec2 texCoords, vec3 viewDir) {
    float layerDepth = 1.0 / m_NumLayers;
    float currentLayerDepth = 0.0;

    vec2 P = viewDir.xy / viewDir.z * m_HeightScale;
    vec2 deltaTexCoords = P / m_NumLayers;

    vec2 currentTexCoords = texCoords;
    float currentDepthMapValue = texture(m_DepthMap, currentTexCoords).r;

    while(currentLayerDepth < currentDepthMapValue) {
        currentTexCoords -= deltaTexCoords;
        currentDepthMapValue = texture(m_DepthMap, currentTexCoords).r;
        currentLayerDepth += layerDepth;
    }

    return currentTexCoords;
}

void main() {
    vec3 viewDir = normalize(vTangentViewPos - vTangentFragPos);

    vec2 texCoords;
    if (m_TextureScale == 1.0) {
        texCoords = ParallaxMapping(vTexCoord, viewDir);

        if (texCoords.x > 1.0 || texCoords.y > 1.0 || texCoords.x < 0.0 || texCoords.y
< 0.0)
        discard;
    }
    else {
        vec2 scaledTexCoords = vTexCoord * m_TextureScale;
        texCoords = fract(ParallaxMapping(scaledTexCoords, viewDir));
    }

    vec3 normal = texture(m_NormalMap, texCoords).rgb;
    normal = normalize(normal * 2.0 - 1.0);

    vec3 color = texture(m_DiffuseMap, texCoords).rgb;
    vec3 ambient = 0.2 * color;
    // For non-directional lights
```

```
    // vec3 lightDir = normalize(vTangentLightPos - vTangentFragPos);
    vec3 lightDir = normalize(g_LightPosition.xyz);
    float diff = max(dot(lightDir, normal), 0.0);
    vec3 diffuse = diff * color;

    vec3 reflectDir = reflect(-lightDir, normal);
    vec3 halfwayDir = normalize(lightDir + viewDir);
    float spec = pow(max(dot(normal, halfwayDir), 0.0), 32.0);
    vec3 specular = vec3(0.2) * spec;

    fragColor = vec4(ambient + diffuse + specular, 1.0);
}
```

steep_parallax_mapping.j3m

```
MaterialDef STEEP_PARALLAX_MAPPING {
    MaterialParameters {
        Texture2D DiffuseMap
        Texture2D NormalMap
        Texture2D DepthMap
        Float HeightScale
        Float NumLayers
        Float TextureScale: 1.0
    }

    Technique {
        LightMode MultiPass

        VertexShader   GLSL330:
Materials/steep_parallax_mapping/steep_parallax_mapping.vert
        FragmentShader GLSL330:
Materials/steep_parallax_mapping/steep_parallax_mapping.frag

        WorldParameters {
            WorldViewProjectionMatrix
            WorldMatrix
            ViewMatrix
            LightPosition
            CameraPosition
        }
    }
}
```

parallax_mapping.vert

```
in vec3 inPosition;
in vec3 inNormal;
in vec2 inTexCoord;
in vec3 inTangent;

uniform mat4 g_WorldViewProjectionMatrix;
uniform mat4 g_WorldMatrix;
uniform mat4 g_ViewMatrix;

uniform vec3 g_LightPosition;
uniform vec3 g_CameraPosition;

out vec3 vPosition;
out vec2 vTexCoord;
// For non-directional lights
// out vec3 vTangentLightPos;
// out vec3 vTangentFragPos;
```

```
out vec3 vTangentViewPos;

void main() {
    vec3 worldPosition = (g_WorldMatrix * vec4(inPosition, 1.0)).xyz;
    vPosition = worldPosition;
    vTexCoord = inTexCoord;

    vec3 T = normalize((g_WorldMatrix * vec4(inTangent, 0.0)).xyz);
    vec3 B = normalize(cross(inNormal, T));
    vec3 N = normalize((g_WorldMatrix * vec4(inNormal, 0.0)).xyz);
    mat3 TBN = transpose(mat3(T, B, N));

    // For non-directional lights
    // vTangentLightPos = TBN * (g_LightPosition - worldPosition);
    // vTangentFragPos = TBN * worldPosition;
    vTangentViewPos = TBN * (g_CameraPosition - worldPosition);

    gl_Position = g_WorldViewProjectionMatrix * vec4(inPosition, 1.0);
}
```

parallax_mapping.frag

```
out vec4 fragColor;

uniform sampler2D m_DiffuseMap;
uniform sampler2D m_NormalMap;
uniform sampler2D m_DepthMap;

uniform float m_HeightScale;
uniform float m_TextureScale;

in vec2 vTexCoord;
// For non-directional lights
// in vec3 vTangentLightPos;
// in vec3 vTangentFragPos;
in vec3 vTangentViewPos;
in vec3 vTangentFragPos;

uniform vec4 g_LightPosition; // Directional light direction

vec2 ParallaxMapping(vec2 texCoords, vec3 viewDir) {
    float height =  texture(m_DepthMap, texCoords).r;
    vec2 p = viewDir.xy / viewDir.z * (height * m_HeightScale);
    return texCoords - p;
}

void main() {
    vec3 viewDir = normalize(vTangentViewPos - vTangentFragPos);

    vec2 texCoords;
    if (m_TextureScale == 1.0) {
        texCoords = ParallaxMapping(vTexCoord, viewDir);

        if (texCoords.x > 1.0 || texCoords.y > 1.0 || texCoords.x < 0.0 || texCoords.y
< 0.0)
        discard;
    }
    else {
        vec2 scaledTexCoords = vTexCoord * m_TextureScale;
        texCoords = fract(ParallaxMapping(scaledTexCoords, viewDir));
    }
```

```
    vec3 normal = texture(m_NormalMap, texCoords).rgb;
    normal = normalize(normal * 2.0 - 1.0);

    vec3 color = texture(m_DiffuseMap, texCoords).rgb;
    vec3 ambient = 0.2 * color;
    // For non-directional lights
    // vec3 lightDir = normalize(vTangentLightPos - vTangentFragPos);
    vec3 lightDir = normalize(g_LightPosition.xyz);
    float diff = max(dot(lightDir, normal), 0.0);
    vec3 diffuse = diff * color;

    vec3 reflectDir = reflect(-lightDir, normal);
    vec3 halfwayDir = normalize(lightDir + viewDir);
    float spec = pow(max(dot(normal, halfwayDir), 0.0), 32.0);
    vec3 specular = vec3(0.2) * spec;

    fragColor = vec4(ambient + diffuse + specular, 1.0);
}
```

parallax_mapping.j3m

```
MaterialDef PARALLAX_MAPPING {
    MaterialParameters {
        Texture2D DiffuseMap
        Texture2D NormalMap
        Texture2D DepthMap
        Float HeightScale
        Float TextureScale: 1.0
    }

    Technique {
        LightMode MultiPass

        VertexShader   GLSL330: Materials/parallax_mapping/parallax_mapping.vert
        FragmentShader GLSL330: Materials/parallax_mapping/parallax_mapping.frag

        WorldParameters {
            WorldViewProjectionMatrix
            WorldMatrix
            ViewMatrix
            LightPosition
            CameraPosition
        }
    }
}
```

normal_mapping.vert

```
in vec3 inPosition;
in vec3 inNormal;
in vec2 inTexCoord;
in vec3 inTangent;

uniform mat4 g_WorldViewProjectionMatrix;
uniform mat4 g_WorldMatrix;

uniform vec3 g_LightPosition;
uniform vec3 g_CameraPosition;

out vec3 vPosition;
out vec2 vTexCoord;
// For non-directional lights
```

```
// out vec3 vTangentLightPos;
// out vec3 vTangentFragPos;
out vec3 vTangentViewPos;

void main() {
    vec3 worldPosition = (g_WorldMatrix * vec4(inPosition, 1.0)).xyz;
    vPosition = worldPosition;
    vTexCoord = inTexCoord;

    vec3 T = normalize((g_WorldMatrix * vec4(inTangent, 0.0)).xyz);
    vec3 B = normalize(cross(inNormal, T));
    vec3 N = normalize((g_WorldMatrix * vec4(inNormal, 0.0)).xyz);
    mat3 TBN = transpose(mat3(T, B, N));

    // For non-directional lights
    // vTangentLightPos = TBN * (g_LightPosition - worldPosition);
    // vTangentFragPos = TBN * worldPosition;
    vTangentViewPos = TBN * (g_CameraPosition - worldPosition);

    gl_Position = g_WorldViewProjectionMatrix * vec4(inPosition, 1.0);
}
```

normal_mapping.frag

```
out vec4 fragColor;

uniform sampler2D m_DiffuseMap;
uniform sampler2D m_NormalMap;
uniform float m_TextureScale;

in vec2 vTexCoord;
// For non-directional lights
// in vec3 vTangentLightPos;
// in vec3 vTangentFragPos;
in vec3 vTangentViewPos;
in vec3 vTangentFragPos;

uniform vec4 g_LightPosition; // Directional light direction

void main() {
    vec3 viewDir = normalize(vTangentViewPos - vTangentFragPos);

    vec2 texCoords = vTexCoord;
    if (m_TextureScale > 1.0) {
        vec2 scaledTexCoords = vTexCoord * m_TextureScale;
        texCoords = fract(scaledTexCoords);
    }

    vec3 normal = texture(m_NormalMap, texCoords).rgb;
    normal = normalize(normal * 2.0 - 1.0);

    vec3 color = texture(m_DiffuseMap, texCoords).rgb;
    vec3 ambient = 0.2 * color;
    // For non-directional lights
    // vec3 lightDir = normalize(vTangentLightPos - vTangentFragPos);
    vec3 lightDir = normalize(g_LightPosition.xyz);
    float diff = max(dot(lightDir, normal), 0.0);
    vec3 diffuse = diff * color;

    vec3 reflectDir = reflect(-lightDir, normal);
    vec3 halfwayDir = normalize(lightDir + viewDir);
    float spec = pow(max(dot(normal, halfwayDir), 0.0), 32.0);
```

```
    vec3 specular = vec3(0.2) * spec;

    fragColor = vec4(ambient + diffuse + specular, 1.0);
}
```

normal_mapping.j3m

```
MaterialDef NORMAL_MAPPING {
    MaterialParameters {
        Texture2D DiffuseMap
        Texture2D NormalMap
        Float TextureScale: 1.0
    }

    Technique {
        LightMode MultiPass

        VertexShader   GLSL330: Materials/normal_mapping/normal_mapping.vert
        FragmentShader GLSL330: Materials/normal_mapping/normal_mapping.frag

        WorldParameters {
            WorldViewProjectionMatrix
            WorldMatrix
            LightPosition
            CameraPosition
        }
    }
}
```

basic_texturing.vert

```
in vec3 inPosition;    // Vertex position
in vec2 inTexCoord;    // Texture coordinates

// Outputs to fragment shader
out vec2 vTexCoord;

// Uniforms
uniform mat4 g_WorldViewProjectionMatrix; // Transformation matrix

void main() {
    // Pass texture coordinates to fragment shader
    vTexCoord = inTexCoord;

    // Transform vertex position and output to OpenGL
    gl_Position = g_WorldViewProjectionMatrix * vec4(inPosition, 1.0);
}
```

basic_texturing.frag

```
// Inputs from vertex shader
in vec2 vTexCoord;

// Output color
out vec4 fragColor;

// Uniforms
uniform sampler2D m_DiffuseMap; // Texture sampler
uniform float m_TextureScale;

void main() {
```

```
    vec2 texCoords = vTexCoord;
    if (m_TextureScale > 1.0) {
        vec2 scaledTexCoords = vTexCoord * m_TextureScale;
        texCoords = fract(scaledTexCoords);
    }

    // Sample the texture using the texture coordinates
    vec4 color = texture(m_DiffuseMap, texCoords);

    // Output the texture color as the fragment color
    fragColor = color;
}
```

basic_texturing.j3m

```
MaterialDef BASIC_TEXTURING {
    MaterialParameters {
        Texture2D DiffuseMap
        Float TextureScale: 1.0
    }

    Technique {
        LightMode MultiPass

        VertexShader   GLSL330: Materials/basic_texturing/basic_texturing.vert
        FragmentShader GLSL330: Materials/basic_texturing/basic_texturing.frag

        WorldParameters {
            WorldViewProjectionMatrix
        }
    }
}
```

ДОДАТОК Г

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ ТА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ФОРМУВАННЯ ЗОБРАЖЕНЬ РЕЛЬЄФНИХ ПОВЕРХОНЬ НА БАЗІ МЕТОДУ PARALLAX OCCLUSION MAPPING

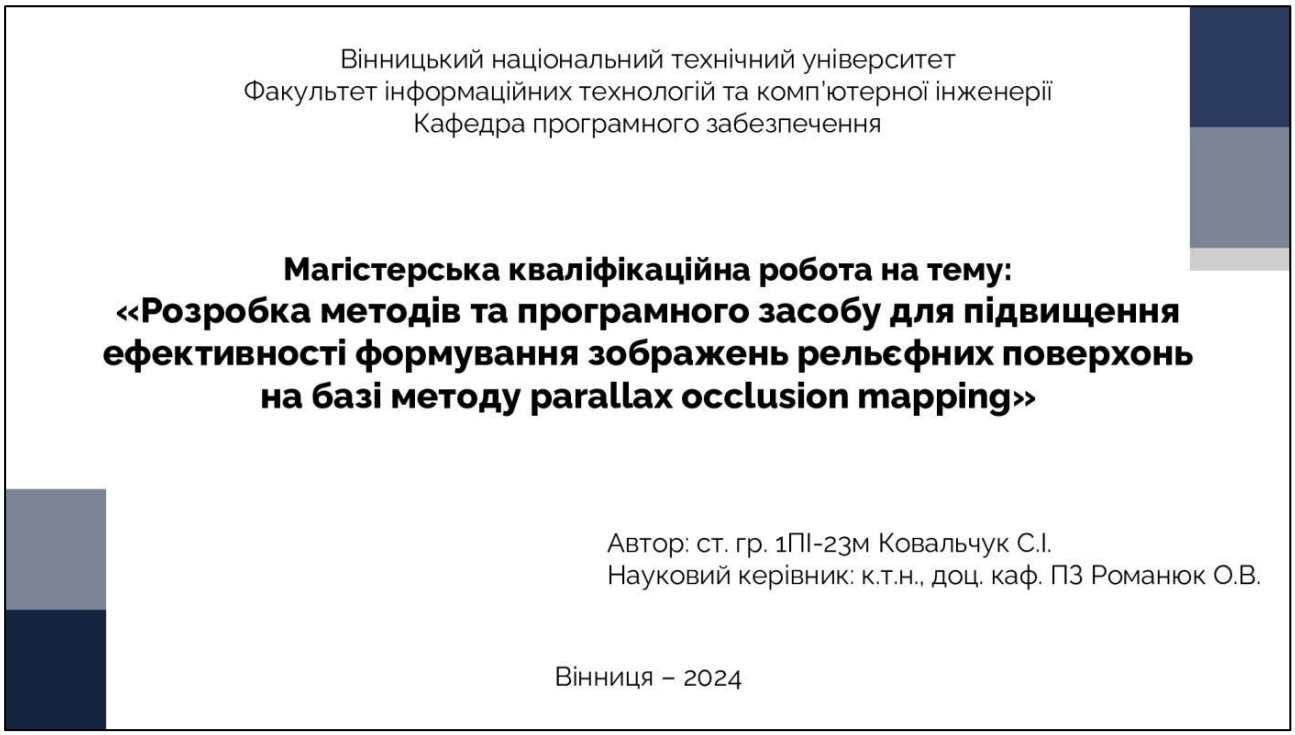


Рисунок Г.1 – Титульний слайд

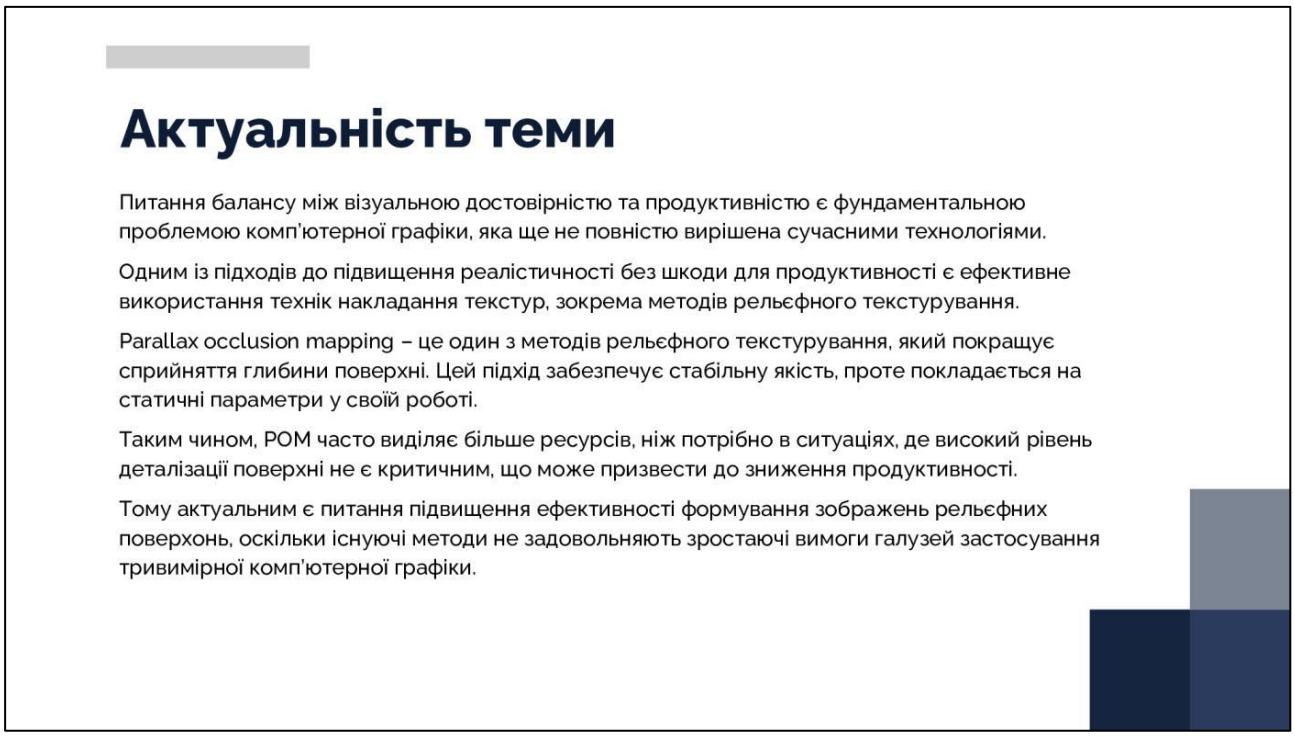


Рисунок Г.2 – Актуальність теми

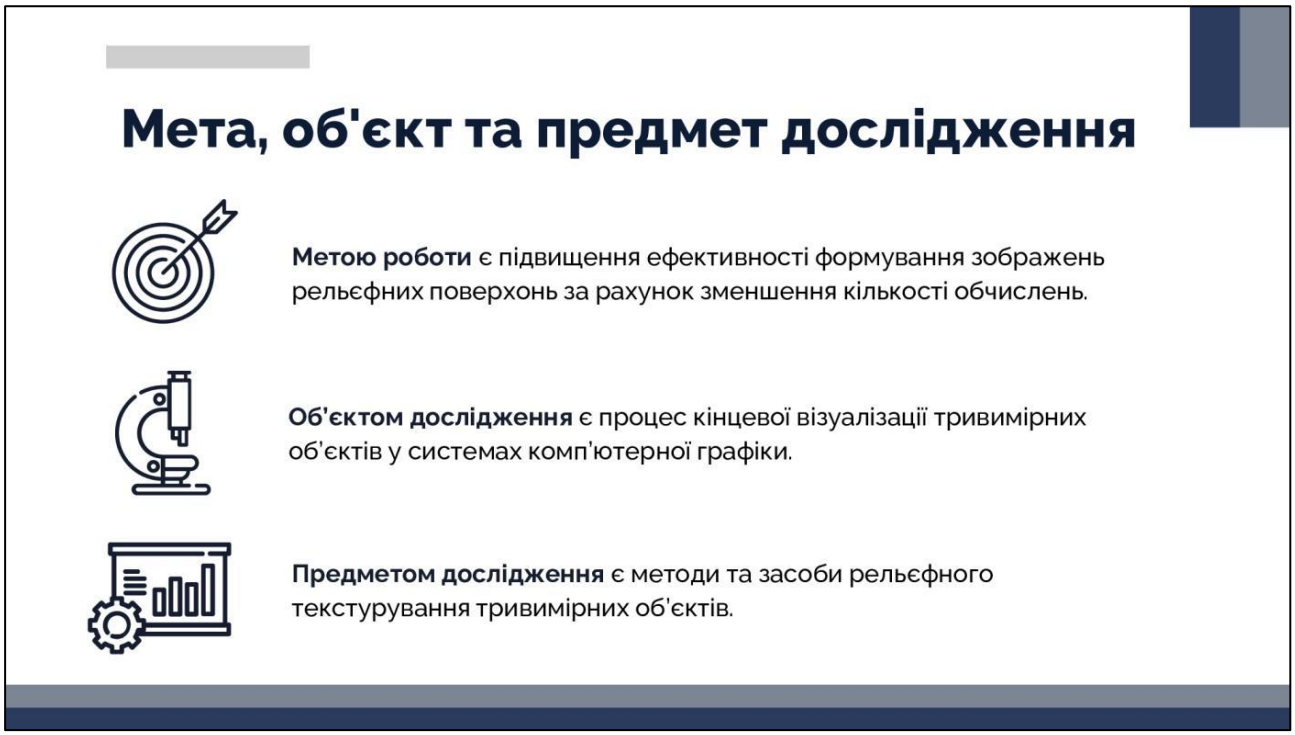


Рисунок Г.3 – Мета, об’єкт та предмет дослідження

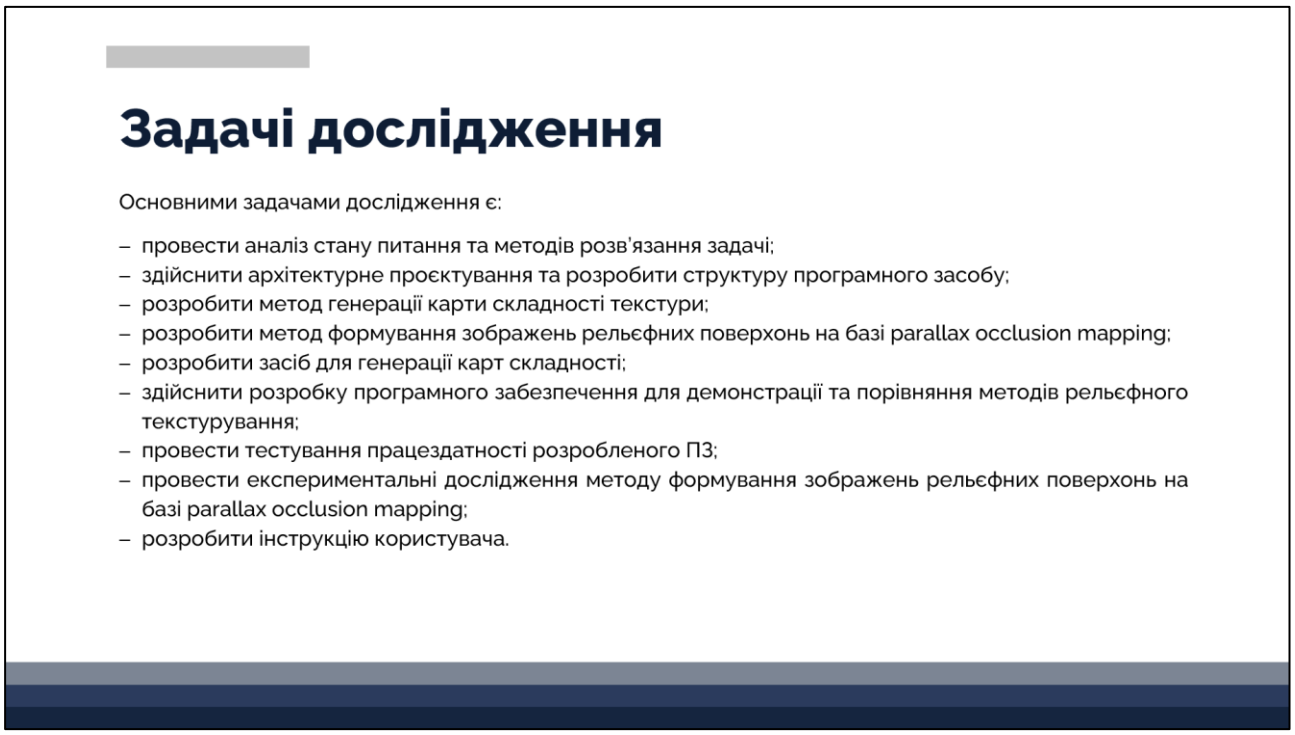


Рисунок Г.4 – Задачі дослідження

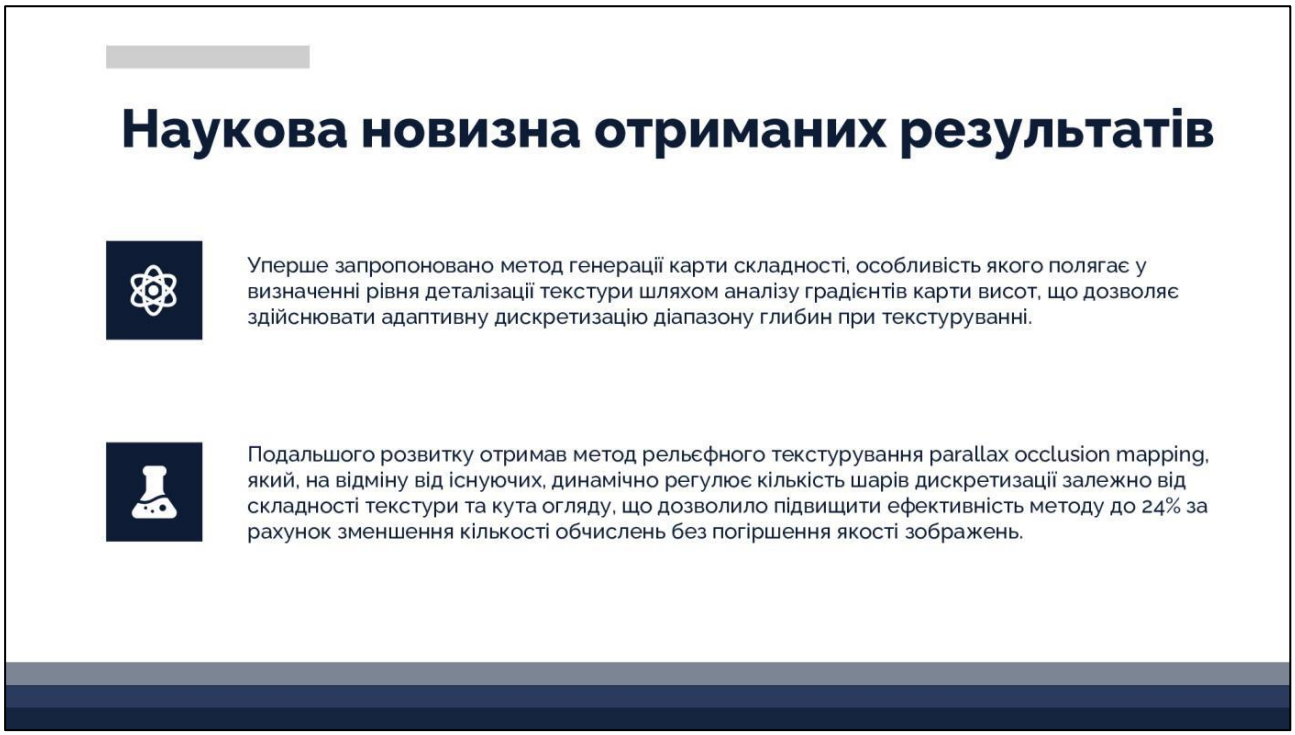


Рисунок Г.5 – Наукова новизна отриманих результатів

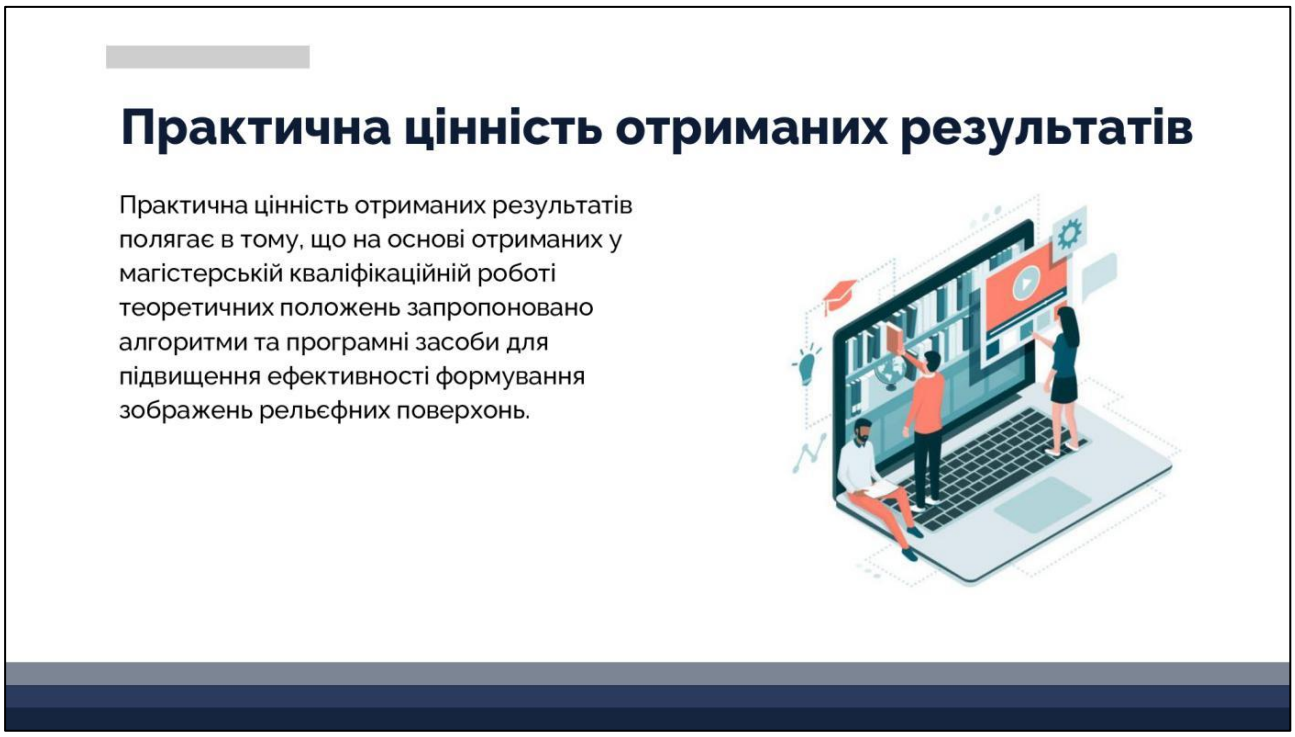


Рисунок Г.6 – Практична цінність отриманих результатів

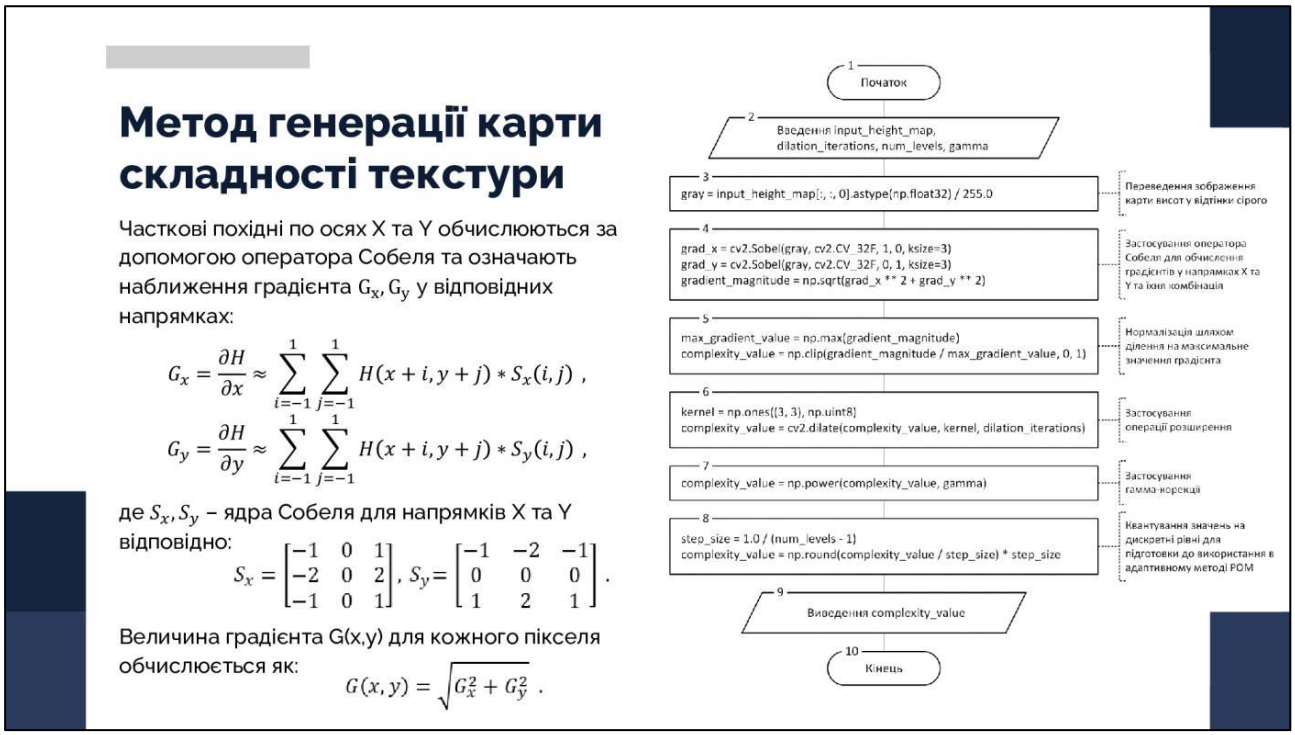


Рисунок Г.7 – Метод генерації карти складності текстури

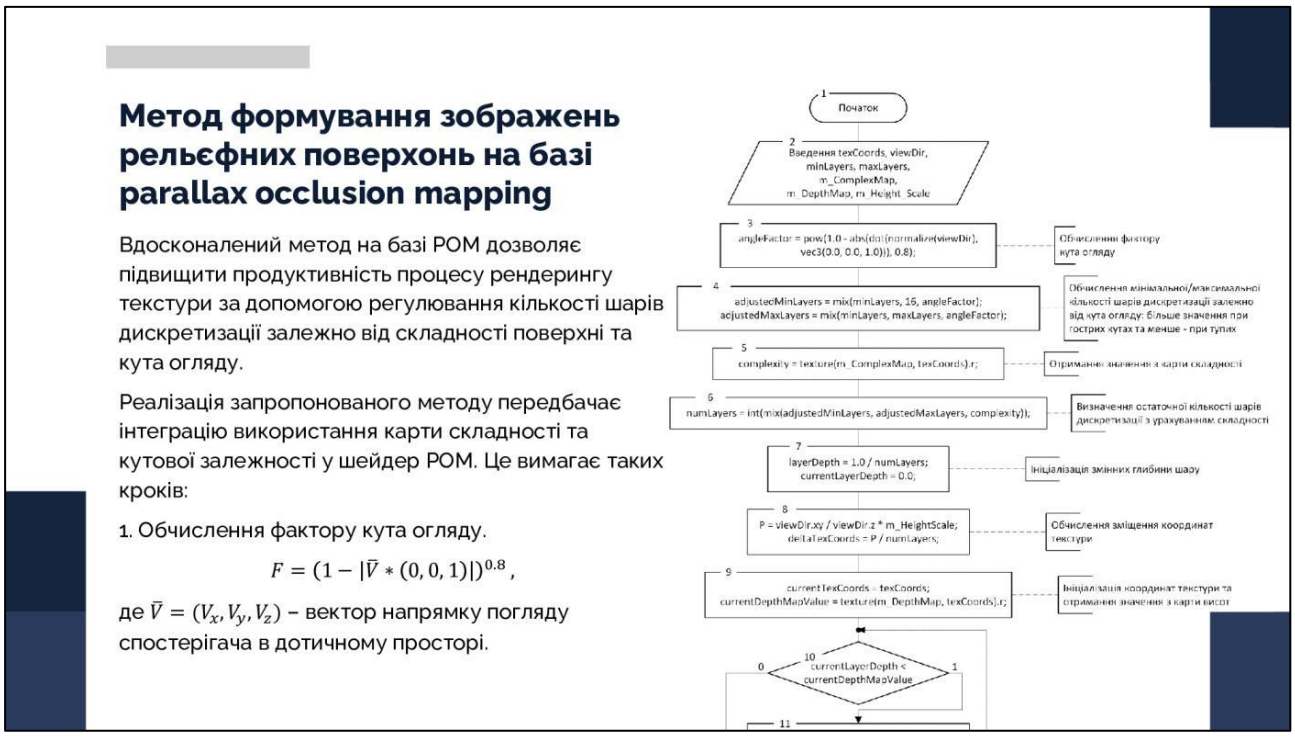


Рисунок Г.8 – Метод формування зображень рельєфних поверхонь на базі parallax occlusion mapping

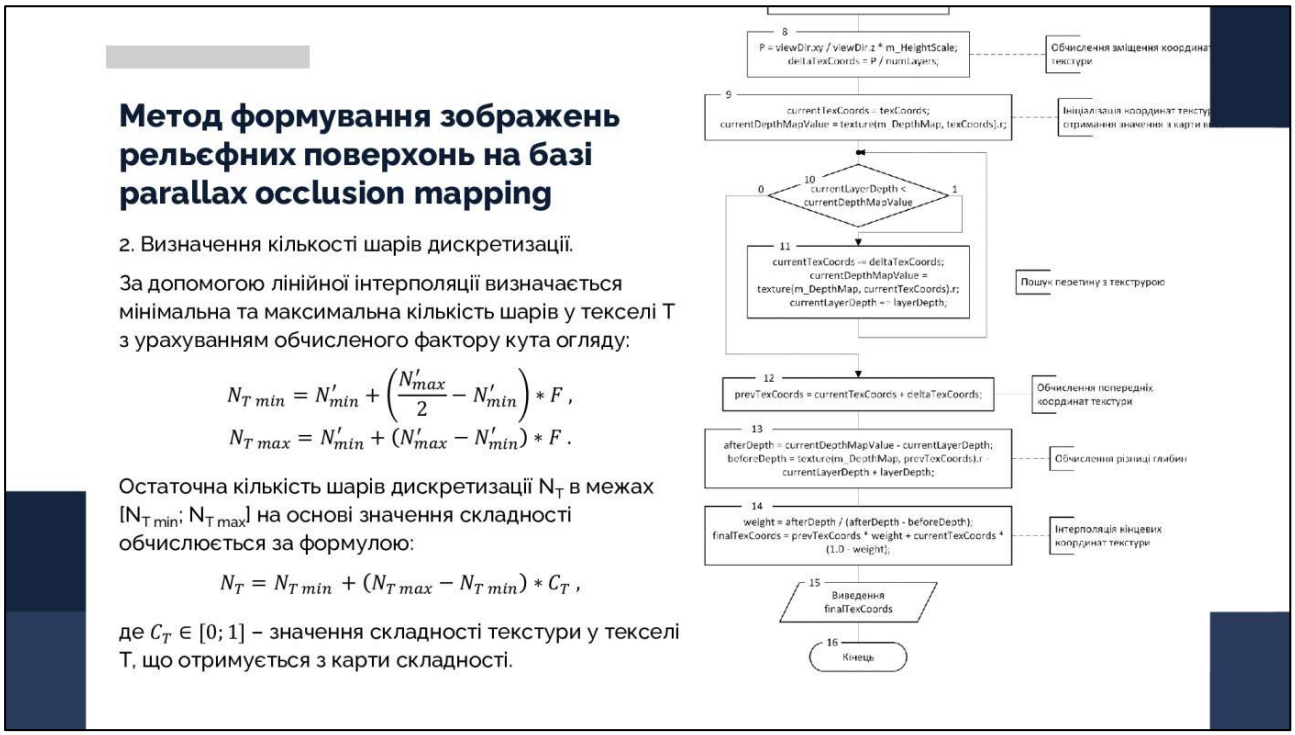


Рисунок Г.9 – Метод формування зображень рельєфних поверхонь на базі parallax occlusion mapping (продовження)

Рисунок Г.10 – Реалізація методу генерації карт складності

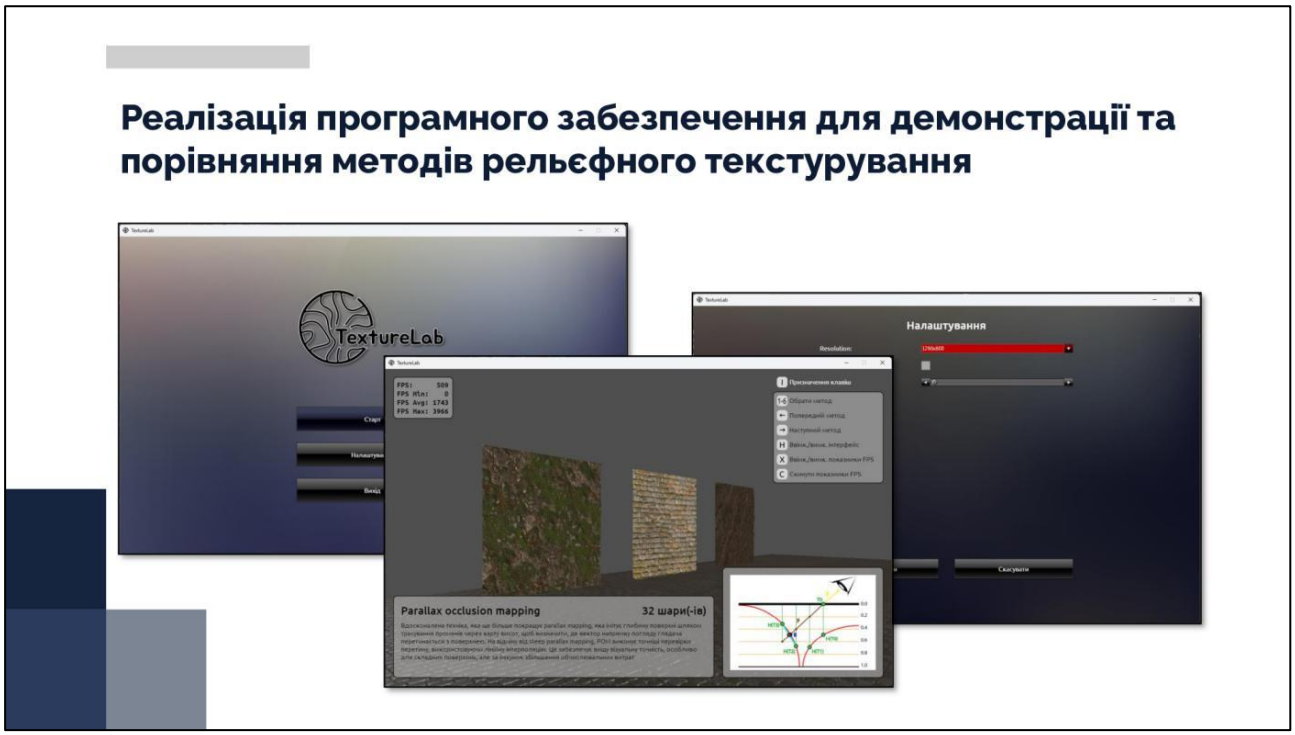


Рисунок Г.11 – Реалізація програмного забезпечення для демонстрації та порівняння методів рельєфного текстурування

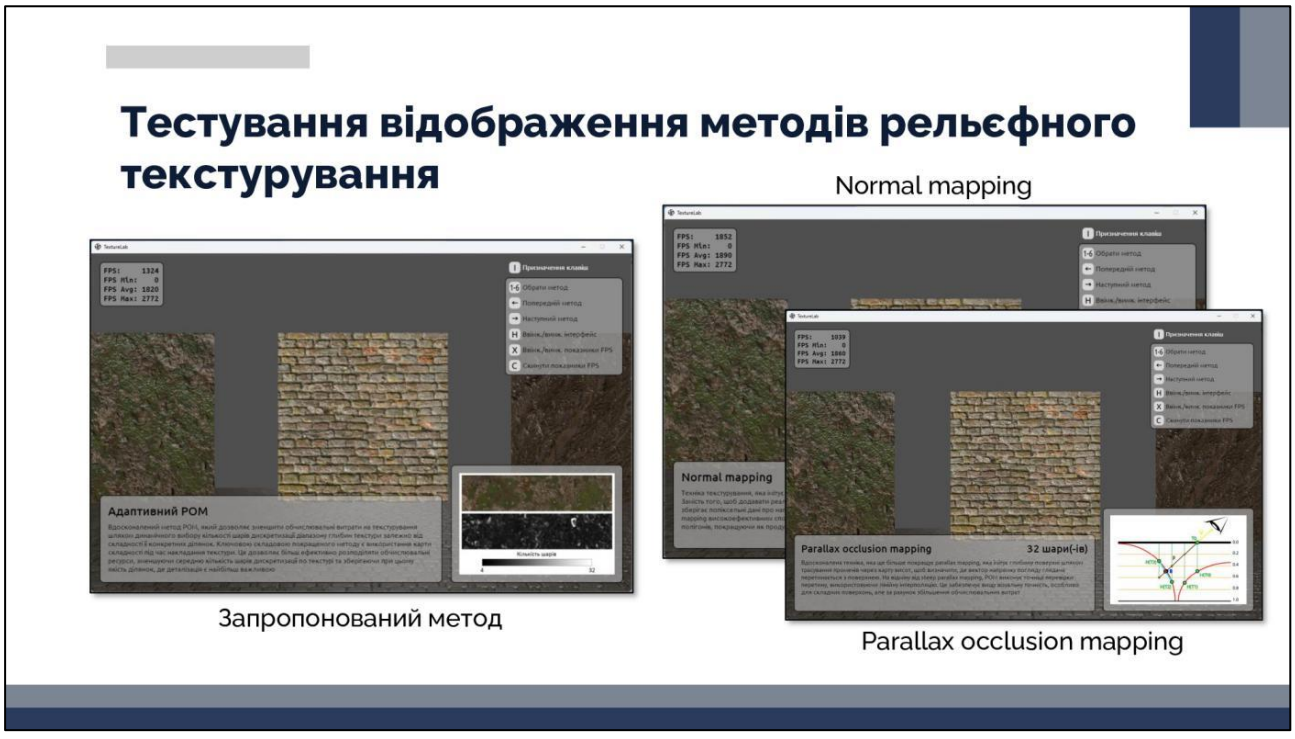


Рисунок Г.12 – Тестування відображення методів рельєфного текстурування

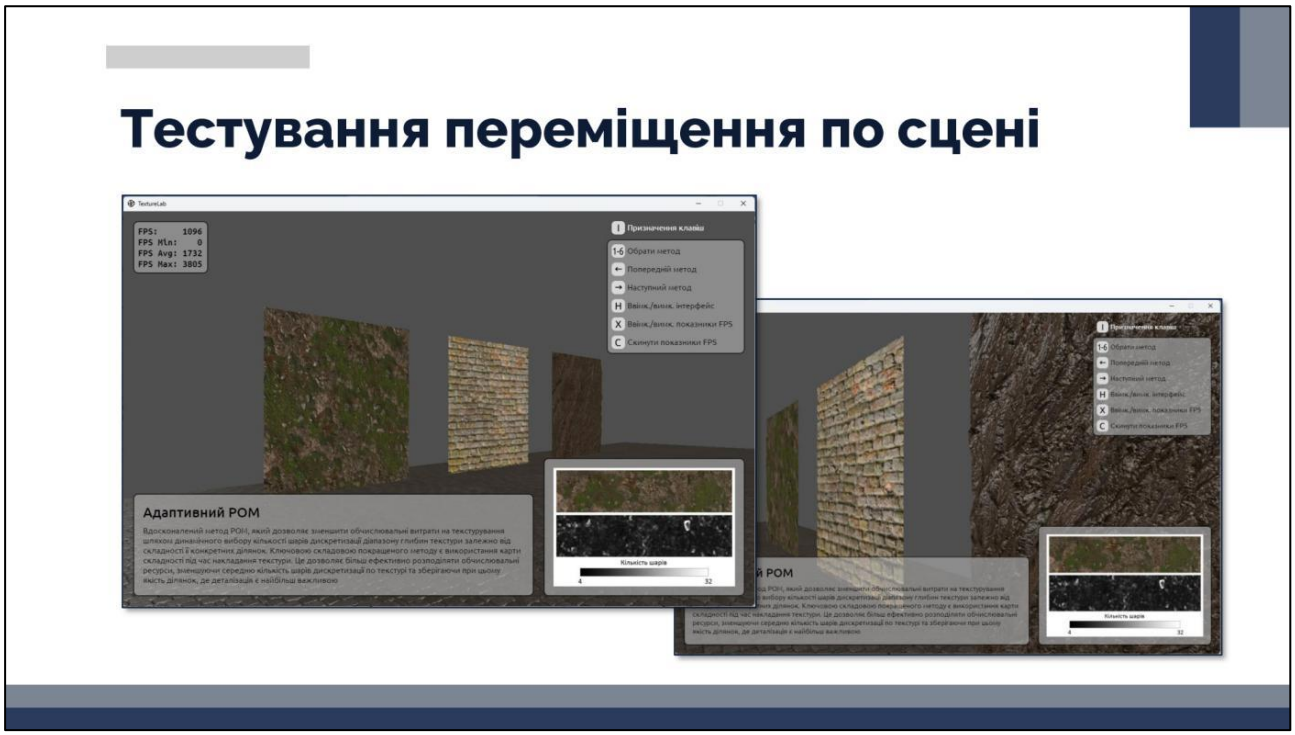


Рисунок Г.13 – Тестування переміщення по сцені

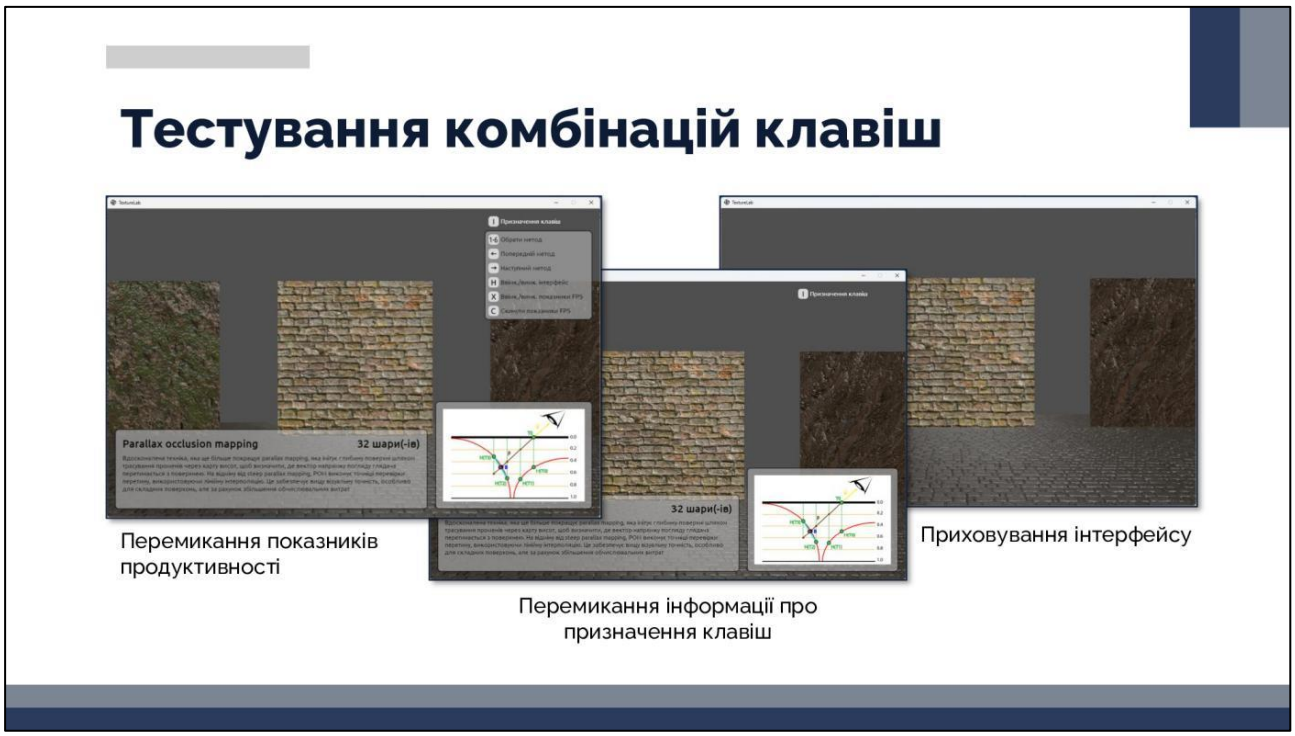
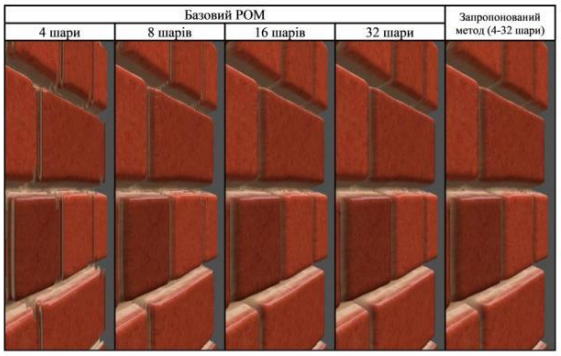


Рисунок Г.14 – Тестування комбінацій клавіш

Результати експериментальних досліджень методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping

Кут огляду	Базовий POM					Запропонований метод (4-32 шари)
	4 шари	8 шарів	16 шарів	24 шари	32 шари	
0°	1294	866	418	280	207	978
45°	571	265	121	80	62	162
60°	670	301	150	102	76	169
80°	1351	735	345	247	185	332
Середнє значення, x̄	**972**	**542**	**259**	**177**	**133**	**410**
	331					



Під час дослідження обидва методи застосовувалися за схожих умов: ідентичне обладнання, сцена та кут огляду, що забезпечує репрезентативність отриманих результатів. Запропонований метод забезпечує середній приріст продуктивності близько 24% порівняно з традиційним підходом, досягаючи вищої ефективності рендерингу без погіршення візуальної якості.

Рисунок Г.15 – Результати експериментальних досліджень методу формування зображень рельєфних поверхонь на базі parallax occlusion mapping

Апробація та публікації результатів роботи

Результати магістерської кваліфікаційної роботи доповідалися на:

- LIII Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2024 р.)
- IV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації» (Одеса, 2024 р.)
- Міжнародній науково-практичній Інтернет-конференції 20-21 листопада 2024 р. «Електронні інформаційні ресурси: створення, використання, доступ та управління» (Суми/Вінниця, 2024 р.)

За тематикою дослідження опубліковано 3 наукові праці у збірниках матеріалів конференцій.

Рисунок Г.16 – Апробація та публікації результатів роботи

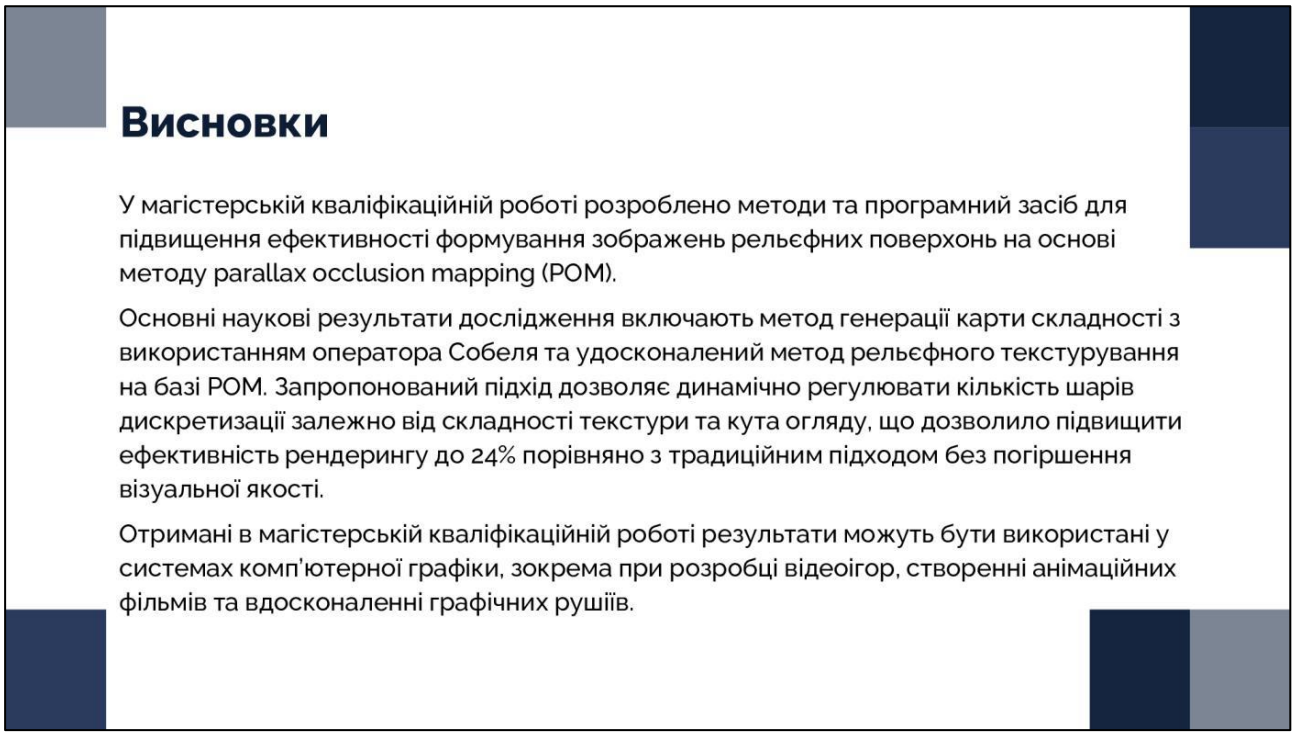


Рисунок Г.17 – Висновки

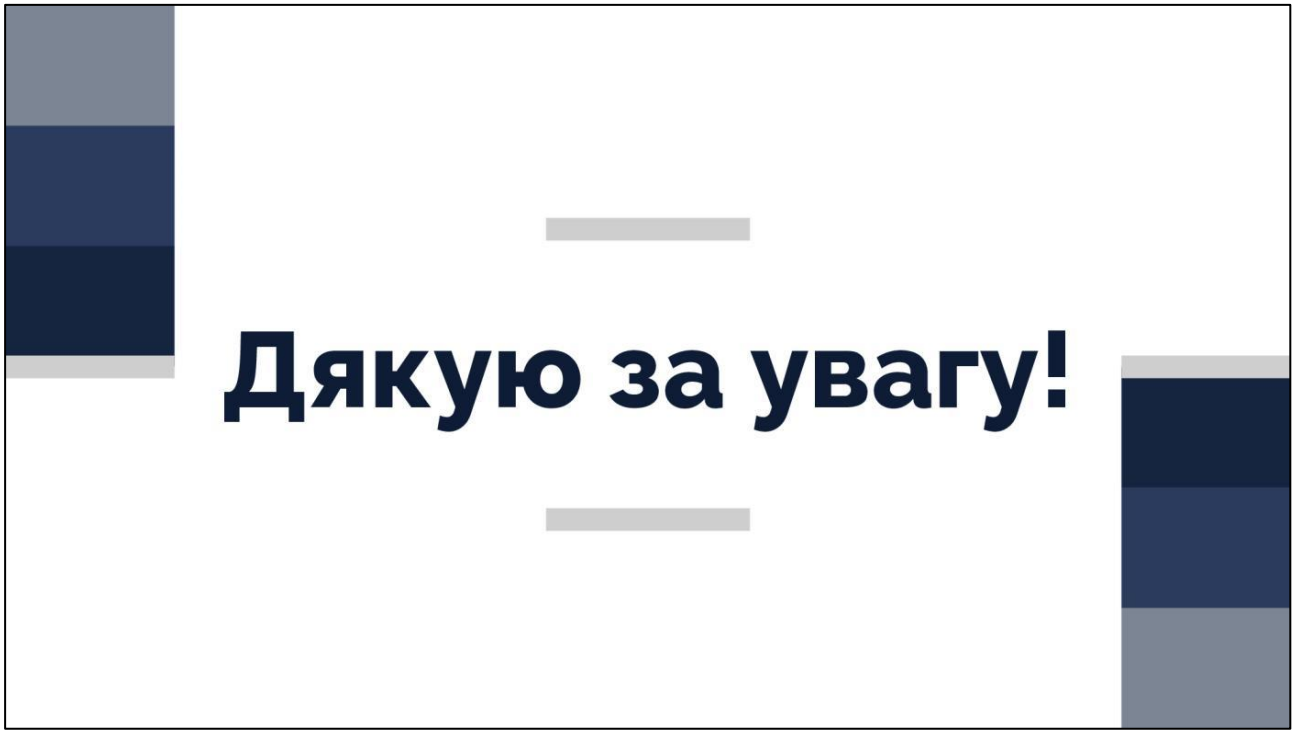


Рисунок Г.18 – Фінальний слайд