Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти другий (магістерський)

**ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ТРИВИМІРНИХ МОДЕЛЕЙ ДЛЯ РОЗРОБЛЕННЯ ІГОР**
(тема)

Виконав:
здобувач 2 року навчання,
групи ІНФм-24-2
Колісниченко М. В.
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Науковий керівник доц. Кобилін О. А.
(посада, прізвище, ініціали)

Допускається до захисту

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

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти другий (магістерський)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

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

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Колісниченко Марії В'ячеславовні
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів оптимізації тривимірних моделей для розроблення ігор

затверджена наказом університету від 14 листопада 2025 року № 1045Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 24 листопада 2025 р.
3. Вихідні дані до роботи методи оптимізації тривимірних моделей, літературні джерела щодо застосування методів оптимізації, програмні засоби для реалізації вибраних методів оптимізації, тривимірна модель для проведення тестування.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз сучасних методів оптимізації тривимірних моделей для ігор.
2. Аналіз літературних джерел щодо апробації методів оптимізації тривимірних моделей.
3. Формування покрокового алгоритму оптимізації.
4. Проведення тестування вибраних методів оптимізації

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми оптимізації тривимірних моделей для розроблення ігор, об'єкт та мета дослідження, постановка задачі, приклад еталонних тривимірних моделей, ілюстрація результатів оптимізації кожним із вибраних методів, висновки, перспективи та апробація роботи.

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

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

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Строк / терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на кваліфікаційну роботу | 29.09.2025 | |
| 2 | Аналіз завдання, підбір літератури | 30.09.25-07.10.25 | |
| 3 | Аналіз літератури з досліджуваної проблеми | 08.10.25-14.10.25 | |
| 4 | Особливості методів оптимізації тривимірних моделей | 15.10.25-20.10.25 | |
| 5 | Дослідження методів оптимізації тривимірних моделей | 21.10.25-27.10.25 | |
| 6 | Програмна реалізація | 28.10.25-05.11.25 | |
| 7 | Обґрунтування отриманих результатів | 06.11.25-11.11.25 | |
| 8 | Оформлення пояснювальної записки | 12.11.25-14.11.25 | |
| 9 | Перевірка на нормоконтроль | 19.11.25-25.11.25 | |
| 10 | Перевірка на плагіат | 25.11.25-26.11.25 | |
| 11 | Рецензування | 26.11.25-27.11.25 | |
| 12 | Підготовка презентації та доповіді | 27.11.25-02.12.25 | |
| 13 | Занесення роботи в електронний архів | 03.12.25 | |
| 14 | Попередній захист кваліфікаційної роботи | 03.12.25 | |

Дата видачі завдання 29 вересня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Кобилін О. А.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 81 с., 17 табл., 28 рис., 42 джерела.

ДЕКІМАЦІЯ, ІГРОВИЙ РУШІЙ, КАРТИ КОРМАЛЕЙ, КОМП'ЮТЕРНА ГРАФІКА, ОПТИМІЗАЦІЯ, ПРОДУКТИВНІСТЬ, РЕНДЕРИНГ, РЕТОПОЛОГІЯ, СТИСНЕННЯ ДАНИХ, ТЕКСТУРНІ АТЛАСИ, ТРИВИМІРНІ МОДЕЛІ, FPS, LOD, PBR, UNITY.

Об'єктом дослідження є процес оптимізації тривимірних моделей у системах комп'ютерної графіки.

Предмет дослідження – методи оптимізації тривимірних моделей для підвищення продуктивності ігрових рушіїв.

Метою дослідження є порівняння та аналіз ефективності методів оптимізації тривимірних моделей, що застосовуються в ігровій розробці.

Використано методи зменшення полігональності, використання рівнів деталізації, оптимізація текстур і матеріалів, стиснення та потокове завантаження даних та використання нормалей і карт висот. Проведено аналіз методів оптимізації тривимірних моделей та відповідних літературних джерел.

Наукова новизна роботи полягає у проведенні порівняльного аналізу та практичному тестуванні методів оптимізації тривимірних моделей.

Результати дослідження мають взаємозв'язок із науковими роботами у сфері обробки зображень, класифікації об'єктів та підвищення продуктивності.

Рекомендації, сформульовані за результатами дослідження, дозволяють зменшити навантаження на апаратну частину без істотної втрати якості зображення, підвищити продуктивність і стабільність ігрових систем.

У результаті дослідження буде проведено системний аналіз методів оптимізації тривимірних моделей, їх математичне обґрунтування та практичне тестування у середовищі Unity.

# ABSTRACT

Explanatory note to the qualification work: 81 pages, 17 tables, 28 figures, 42 sources.

3D MODELS, COMPUTER GRAPHICS, DATA COMPRESSION, DECIMATION, FPS, GAME ENGINE, LOD, NORMAL MAPS, OPTIMIZATION, PBR, PERFORMANCE, RENDERING, RETOPOLOGY, TEXTURE ATLASES, UNITY.

The object of the research is the process of optimizing three-dimensional models in computer graphics systems.

The subject of the research is methods of optimizing 3D models to increase the performance of game engines.

The aim of the research is to compare and analyze the effectiveness of three-dimensional model optimization methods used in game development.

Methods of polygonal reduction, Levels of Detail, optimization of textures and materials, compression and streaming of data, and use of normals and height maps were used. An analysis of modern methods of optimization of 3D models and relevant literature sources was conducted.

The scientific novelty of the research lies in conducting comparative analysis and practical testing of methods for optimizing 3D models in the Unity environment.

The research results have a relationship with scientific works in the field of image processing, object classification, and improving performance.

The recommendations formulated based on the research results allow reducing the load on the hardware without significant loss of image quality, increasing the performance and stability of gaming systems.

As a result of the research, a systematic analysis of methods for optimizing three-dimensional models, their mathematical justification, and practical testing in the Unity environment will be conducted.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AR – Augmented Reality (доповнена реальність)

CPU – Central Processing Unit (центральний процесор)

Draw Call – команда рендерингу, яка передається з процесора до GPU для відображення об’єктів сцени

FPS – Frames per Second (кількість кадрів за секунду)

Frame Time – час, необхідний для рендерингу одного кадру

GPU – Graphics Processing Unit (графічний процесор)

Instancing – техніка повторного відображення одного об’єкта кілька разів із мінімальними витратами ресурсів

Material – набір властивостей, що визначає вигляд поверхні тривимірного об’єкта (колір, блиск, текстуру тощо)

Mesh – полігональна сітка, що визначає форму тривимірного об’єкта

LOD – Level of Detail (рівень деталізації)

PBR – Physically Based Rendering (фізично коректне рендеринг-зображення)

Shader – програмний модуль, який описує, як об’єкт взаємодіє зі світлом

UV-мапа – розгортка тривимірної моделі для накладання текстур

VR – Virtual Reality (віртуальна реальність)

# ВСТУП

Сучасна комп’ютерна графіка є одним із найдинамічніших напрямів інформаційних технологій, що активно розвивається в контексті ігрової індустрії, віртуальної реальності, архітектурної візуалізації та симуляцій. Вона поєднує математичне моделювання, інженерні розрахунки та художні засоби для створення реалістичних тривимірних об’єктів і сцен. Тривимірна графіка використовується для відображення геометричних моделей у віртуальному просторі, забезпечуючи взаємодію користувача із середовищем у режимі реального часу. Рівень якості тривимірних моделей безпосередньо впливає на сприйняття ігрового процесу, однак водночас є одним з основних факторів, що визначають навантаження на апаратні ресурси системи. У сучасних ігрових рушіях велика кількість деталей відтворюється завдяки складним шейдерним програмам, фізично коректному освітленню та процедурним матеріалам, що додатково підвищує вимоги до оптимізації початкових тривимірних ресурсів. Тому питання зменшення обчислювальної складності тривимірних сцен нині розглядається як один із ключових аспектів успішного проєктування ігор різних жанрів.

Зі зростанням вимог до графічної реалістичності у сучасних іграх з’являється проблема оптимізації тривимірних моделей. Чим детальнішими стають віртуальні об’єкти, тим більша кількість полігонів, текстур і матеріалів необхідна для їх відображення. Це призводить до збільшення часу рендерингу, споживання пам’яті та зниження частоти кадрів. Оптимізація тривимірних моделей дозволяє досягти балансу між якістю зображення та продуктивністю системи, що є ключовим аспектом у розробленні ігор. Особливо гостро проблема оптимізації постає у мобільних та VR-середовищах, де навіть незначне збільшення навантаження на GPU може призвести до падіння продуктивності або дискомфорту користувача. У зв’язку з цим розробники змушені впроваджувати комплексні підходи до оптимізації, які враховують особливості

цільової платформи, обмеження енергоспоживання та вимоги до стабільності FPS.

Актуальність роботи полягає у необхідності підвищення ефективності процесу створення та відображення тривимірних моделей в ігрових рушіях. У сучасних умовах більшість ігор мають відкриті світи, складну геометрію, фізику об’єктів і динамічні тіні, тому оптимізація моделей є критично важливою для забезпечення стабільної продуктивності. Водночас із розвитком технологій реального часу з’являються нові методи оптимізації, що потребують системного дослідження й узагальнення практичних підходів. Таким чином, дослідження методів оптимізації 3D моделей має як наукову, так і практичну значущість для сучасної індустрії інтерактивних додатків.

Стан проблеми досліджується в наукових роботах, присвячених підвищенню продуктивності візуалізації, зменшенню обчислювальних витрат та розробленню ефективних алгоритмів рендерингу. Зокрема, увага приділяється методам спрощення сіткових структур, застосуванню багаторівневих систем відображення, використанню PBR-матеріалів та розподіленій обробці графічних ресурсів у реальному часі. Однак більшість досліджень орієнтовані на окремі етапи оптимізації, тоді як комплексне порівняння методів у межах єдиної системи виконано недостатньо. Також значна частина публікацій зосереджується на алгоритмічних аспектах, тоді як практичні порівняння методів на прикладі конкретних інструментів, таких як Unity чи Unreal Engine, усе ще є недостатньо представленими.

# 1 ОГЛЯД ТЕХНОЛОГІЙ ОПТИМІЗАЦІЙ ТРИВИМІРНИХ МОДЕЛЕЙ

## 1.1 Поняття оптимізації в комп’ютерній графіці

Оптимізація – це процес покращення та удосконалення тривимірних моделей, мета якого полягає в зменшенні кількості полігонів, текстурних розмірів та інших параметрів що впливають на продуктивність, при цьому зберігши візуальну якість моделі. У комп’ютерній графіці цей процес є критичною стадією створення кінцевого продукту, оскільки безпосередньо впливає на своєчасне відтворення сцен, розмірі гри та її доповнень, а також на користувацький досвід.

Фундаментальна мета оптимізації полягає у досягненні оптимального балансу між візуальною якістю (Fidelity) та продуктивністю (Performance), вираженою у стабільній та високій частоті кадрів (FPS). Сучасні системи рендерингу функціонують в умовах постійного технологічного тиску, спричиненого запитом на фотореалізм, що значно ускладнює сцени та збільшує обсяг даних, які необхідно обробити за частки секунди.

Сучасний графічний конвеєр (Pipeline) вимагає оптимізації на трьох основних рівнях: CPU, GPU та пам’ять. На рівні CPU оптимізація спрямована на зменшення кількості викликів промальовування (Draw Calls), які є комунікаційним бар’єром між центральним та графічним процесорами. Надмірна кількість цих викликів може призвести до «вузького місця» (bottleneck), де CPU не встигає підготувати дані для GPU. На рівні GPU оптимізація зосереджена на ефективній обробці геометричних даних (Vertex Shader) та піксельних даних (Pixel/Fragment Shader), що досягається через зменшення геометричної складності та мінімізацію складних шейдерних інструкцій.

З розвитком технологій рендерінгу та появою складних фізично коректних шейдерів зросла потреба у методах, які дозволять досягти балансу між якістю та ефективністю. Оптимізація включає широкий спектр прийомів – від редукції

геометрії (mesh decimation) і створення рівнів деталізації (LOD) до оптимізації UV-розгорток, текстурного атласування, нормал мепінгу та baking процедур.

Ключовин чинником ефективної оптимізації є узгодження між графічною складністю сцени та можливостями апаратного забезпечення. У сучасних ігрових рушіях, таких як Unreal Engine 5 та Unity HDRP, використовуються комплексні алгоритми динамічного LOD, що адаптують рівень деталізації в залежності від відстані до камери.

Крім геометричної оптимізації, вагому роль відіграє оптимізація матеріалів та текстур. Використання texture streaming та virtual texturing може зменшити споживання пам’яті GPU на 30-40% без помітної втрати якості [1].

Також все більшого поширення набуває оптимізація через процедурні генеративні методи. Наприклад, за допомогою Houdini або Blender Geometry Nodes, які дозволяють створювати адаптивні рівні деталізації та автоматично редукувати топологію під задані критерії.

Важливо зазначити, що оптимізація має різний характер у контексті використання тривимірної моделі. Для ігор у реальному часі критичними є FPS, для VR/AR-застосунків – затримка та стабільність рендерингу, тоді як для кінематографу – візуальна достовірність при високій роздільній здатності.

Останні роки відзначені зростанням складності вхідних даних. Моделі, отримані через фотограмметрію або створені за допомогою процедурних генераторів, часто містять надмірну кількість полігонів, що є неефективним для рендерингу в реальному часі [2,3]. У зв’язку з цим, основними показниками ефективності оптимізації (KPIs) є: полігональний бюджет, час рендерингу кадру та використання пам’яті.

Полігональний бюджет (polygon count) визначає максимально допустиму кількость полігонів для конкретної платформи (ПК, консолі, мобільні пристрої) та типу моделі (головний персонаж, другорядний об’єкт, елемент фону). Дотримання «полігонального бюджету» є критичним, особливо для мобільних ігор, де обчислювальні ресурси є вкрай обмеженими.

Час рендерингу кадру (frame time) – це зворотна величина частоти кадрів (FPS). Для плавного геймплею, наприклад, при 60 FPS, час рендерингу кадру не має перевищувати 16,6 мс [4]. Інструменти профілювання, такі як вбудовані в Unreal Engine 5, є необхідними для виявлення ділянок, де час кадру несподівано зростає.

Використання пам’яті (memory footprint) також важливий показник, а саме оптимізація використання пам’яті GPU, особливо для текстур. Впровадження систем текстурного стрімінгу дозволяє динамічно керувати завантаженням лише необхідних рівнів деталізації (mipmaps), запобігаючи переповненню пам’яті.

Таким чином, оптимізація – це не лише технічна необхідність, а й стратегічний інструмент художнього контролю, що забезпечує ефективність рендерингу, інтерактивність та масштабність проєктів. В сучасній графіці оптимізація перейшла від простого зменшення полігонів до комплексного управління ресурсами, що охоплює всі етапи конвеєра: від створення ассета до його фінального рендерингу.

## 1.2 Особливості використання тривимірних моделей у комп’ютерних іграх

У комп’ютерних іграх тривимірні моделі є основним елементом, який формує ігровий світ, персонажів та об’єкти взаємодії. На відміну від статичного рендерингу в фільмах, ігрова графіка потребує обчислень у реальному часі, де будь яка надлишкова складність моделі може призвести до втрати продуктивності [5].

У сучасних ігрових рушіях (Unreal Engine 5, Unity, CryEngine) активно застосовуються гібридні підходи оптимізації, які включають динамічне регулювання рівня деталізації (LOD), culling – відсічення невидимих об’єктів, instancing – повторне використання однієї моделі, baking освітлення для зменшення динамічних обчислень та streaming texture та mesh compression.

Використання тривимірних моделей в інтерактивному середовищі відрізняється від статичного рендерингу тим, що моделі повинні адаптуватися до змінної точки огляду, відстані та видимості в реальному часі. Ключовою технологією, яка забезпечує цю адаптивність, є рівні деталізації (Level of Detail, LOD).

Сучасні AAA-ігри використовують системи Nanite та Virtual Shadow Maps, що дозволяють візуалізувати мільйони полігонів без традиційного LOD, але при цьому зберегти ефективність завдяки адаптивній оптимізації під час рендерингу.

Поява в 2022 році системи Nanite в Unreal Engine 5 здійснила якісний стрибок у підходах до роботи з геометрією, що є найважливішою подією в галузі за останні роки. Nanite являє собою систему віртуалізованої геометрії, яка відмовляється від класичного управління LOD-групами для статичних об’єктів.

Проте, незважаючи на революційність Nanite, традиційні методи оптимізації залишаються актуальними для анімованих (деформованих) моделей, рушіїв, які не підтримують віртуалізовану геометрію (наприклад, більшість мобільних рушіїв), та для систем, які потребують адаптивного стрімінгу геометрії у великих відкритих світах, де навіть не-Nanite підходи вимагають ефективного управління даними.

Крім того, оптимізація моделей для ігор охоплює і процес ріґінгу (спрощення скелетної структури для анімації), і оптимізацію шейдерів. Наприклад, Unreal Engine Shader Complexity View дає змогу аналізувати вплив кожного матеріалу на продуктивність сцени [6-10].

Для мобільних застосунків застосовуються додаткові методи - наприклад, texture atlases та occlusion baking, що дозволяють суттєво зменшити кількість draw calls.

З точки зору розробки, важливим аспектом є баланс між візуальною привабливістю та ефективністю рендерингу [11, 12]. Під час створення моделей для ігор, художники повинні врахувати ліміти полігонів, розмір текстур, складність матеріалів і навіть кількість кісток у скелеті.

Отже, тривимірні моделі в іграх вимагають системного підходу до оптимізації, що поєднує технічні та художні аспекти. Без належної оптимізації навіть візуально досконала модель може спричинити падіння FPS, перегрів пристроїв або крах застосунку.

## 1.3 Аналіз сучасних підходів до оптимізації тривимірних моделей

Головним завданням сучасної оптимізації тривимірних моделей є збереження балансу між продуктивністю та візуальною якістю зображення. Основними напрямами є геометрична оптимізація, оптимізація текстур, матеріалів, шейдерів та освітлення. Розглянемо детальніше кожен напрям.

Напрям геометричної оптимізації є основним та включає такі техніки, як редукція полігонів (mesh decimation), ретопологія (retopology), рівні детілазації (LOD), створення копій (instancing) та об’єднання об’єктів (mesh merging). Метод LOD передбачає створення кількох варіантів моделі з різною кількістю полігонів. Використовуючи цей метод, ігровий рушій автоматично обирає потрібну версію в залежності від відстані об’єкта до камери. Застосування даного методу дозволяє зменшити кількість обчислень на 60-80% для об’єктів, що розташовуються далеко від камери [13]. Приклад реалізації LOD оптимізації показано на рисунку 1.1.

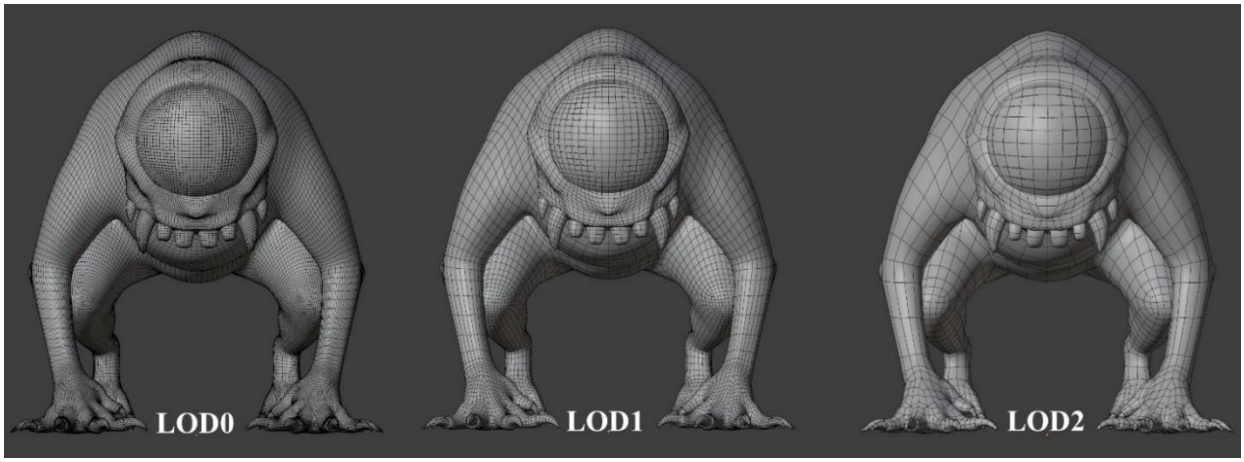



Рисунок 1.1 – Приклад реалізації LOD на тривимірному персонажі

З розвитком технологій, з’являються альтернативи традиційним LOD-системам. Наприклад, технологія Nanite (Unreal Engine 5) використовує кластерізацію полігонів у віртуальні мікромеші [14-18]. Це дозволяє працювати з великим обсягом полігонів у реальному часі без ручної редукції. Приклад реалізації наведено на рисунку 1.2 [19].


Рисунок 1.2 – Приклад реалізації Nanite на тривимірному об’єкті

Спрощення (mesh decimation) спрямоване на зниження полігонального лічильника моделі шляхом агресивного злиття вершин та видалення ребер. Сучасні алгоритми, такі як втілені в Decimation Master або в автоматичних функціях Godot 4.3 або Blender 3D, використовують метрики помилок (наприклад, Quadric Error Metrics), щоб визначити, які частини сітки можна видалити з найменшим візуальним впливом. Акцент робиться на тому, щоб важливі геометричні особливості (гострі кути, контури) були збережені навіть при агресивному спрощенні (рис. 1.3).

Ретопологія є більш керованим процесом, необхідним для створення чистої, рівномірної сітки (переважно чотирикутниками), що є ідеальною для анімації та деформації. Чиста топологія запобігає небажаним артефактам при скінінгу та деформації. Також ідеальна сітка критично важлива для запікання карт, адже рівномірність сітки забезпечує більш чисте та якісне запікання карт нормалей. Сучасні інструменти, такі як ZRemesher та процедурні підходи в Houdini, дозволяють проводити напівавтоматичну або повністю автоматичну ретопологію, суттєво прискорюючи цей процес.

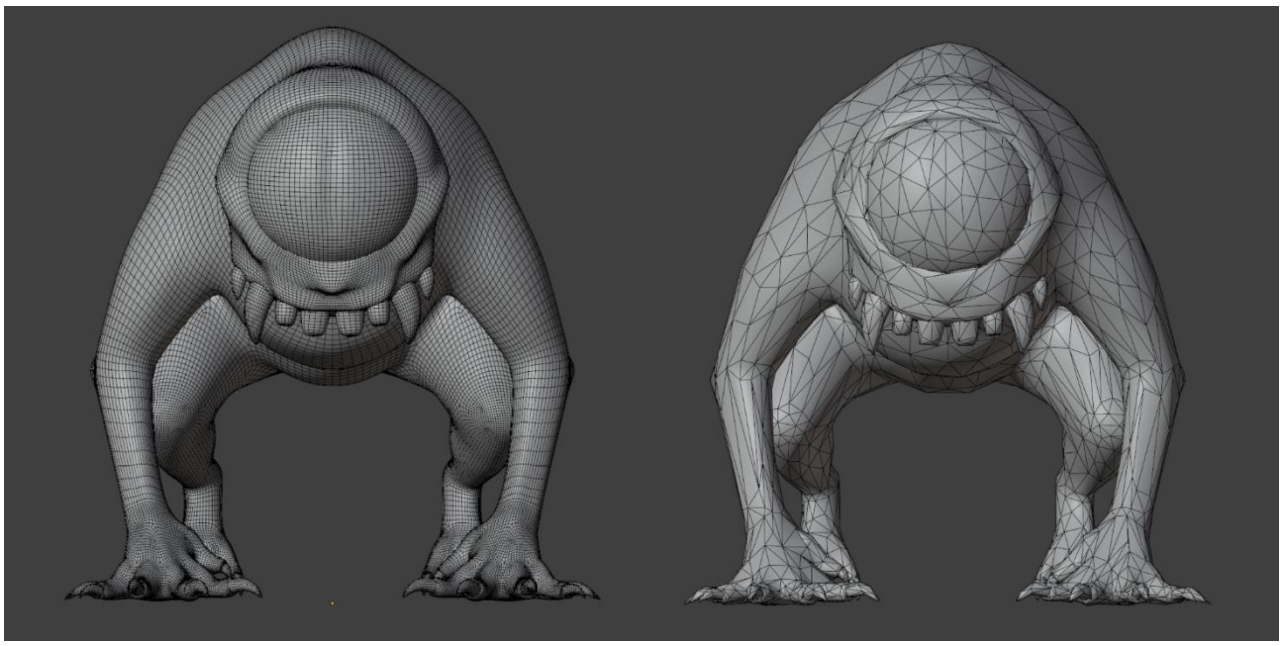


Рисунок 1.3 – Приклад реалізації Decimate на тривимірному персонажі

Такий підход частіше використовується для моделей, що не є персонажами (пропси, елементи оточенні). Такий метод може мати свої недоліки у порівнянні з традиційною ручною ретопологією. Наприклад, для ретопології ігрового персонажа майже завжди використовується метод саме ручної ретопології, адже в цьому випадку критичним аспектом є коректна деформація сітки моделі. Для таких цілей майже немоливо досягти чистої та правильної сітки за домогою автоматичних або напівавтоматичних методів ретопології (рис. 1.4).

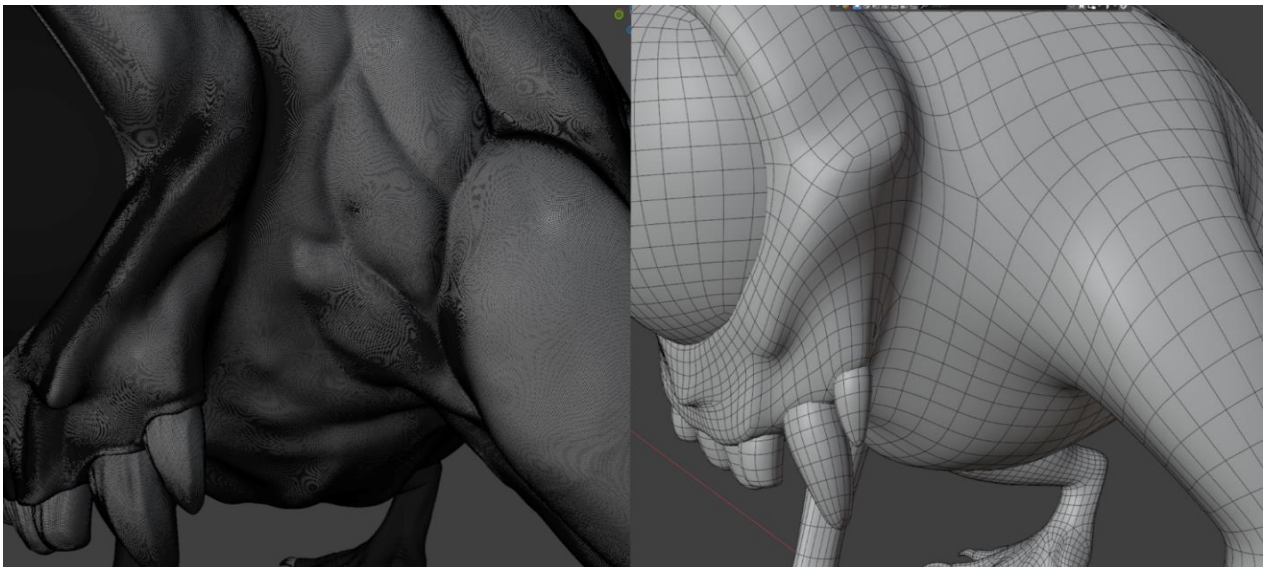


Рисунок 1.4 – Приклад ручної ретопології тривимірного персонажа

Оптимізація текстур також має ключове значення, адже текстури займають до 70% пам’яті графічної сцени [20]. Основним підходами цього напряму є зменшення роздільної здатності без втрати якості (mipmapping), об’єднання кількох текстур в одну карту (texture atlasing), стиснення текстур (compression) [21] та віртуальне текстурування (virtual texturing), що дозволяє стрімінг фрагментів лише видимих ділянок карти. Приклад використання texture atlasing у комп’ютерній грі Minecraft наведено на рисунку 1.5 [22].

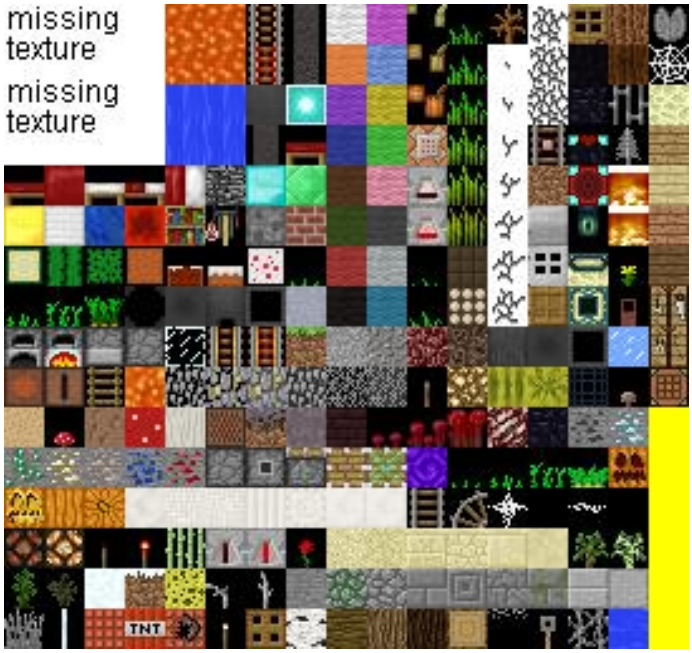



Рисунок 1.5 – Приклад використання texture atlasing у комп’ютерній грі Minecraft

Матеріали мають прямий вплив на продуктивність, адже визначають кількість текстур, світлових взаємодій та інструкцій шейдера. Для оптиміазції за цим напрямом використовують об’єднання матеріалів (material instancing), прості шейдери (simple PBR shaders) та оптимізацію за допомогою нод (node-based).

Рендерінг освітлення – один із найбільш ресурсомістких процесів. Методи оптимізації включають запікання світла (light baking), світлові проби (light probes), тіні екранного простору (screen-space shadows), поля тіней на відстані (distance field shadows) та віртуальні карти тіней (virtual shadow maps).

## 1.4 Огляд сучасного програмного забезпечення для створення та обробки тривимірних моделей

Сучасний процес створення тривимірних моделей базується на комбінуванні кількох програм та охоплює процеси моделювання, скульптингу, створення UV-розгортки, текстурування, рендерингу та тестування.

Для моделювання, скульптингу та ретопології зазвичай використовують такі програми як Maya, Blender, 3DSMax, Houdini, 3DCoat та ZBrush. Програми мають різний функціонал та можливості, тож їх також часто поєднують.

ZBrush – це основний інструмент для створення високодеталізованої геометрії, скульпту. Його критична роль у оптимізації полягає у використанні ZRemesher (інструмент для автоматичної ретопології) та Decimation Master (інструмент для створення спрощених версій моделі, які пізніше використовуються, наприклад, для запікання карти нормалей). Ці інструменти дозволяють перетворити терабайти даних скульптингу в ефективні ігрові ассети.

Houdini визначається своєю процедурністю. Це дозволяє створювати гнучкі системи для автоматичної генерації LOD-рівнів, динамічного спрощення та підготовки геометричних даних у великих масштабах. Процедурний підхід гарантує, що зміни в оригінальній моделі автоматично оновлюють усі оптимізовані версії.

Blender – потужний відкритий інструмент, який швидко став основним у сфері інді- та АА-розробки. Його перевага полягає в інтеграції повного набору інструментів, необхідних для оптимізації. Ключовим інструментом оптимізаційної частини функціоналу є модифікатор «Decimate» [23, 24]. Він пропонує ефективні алгоритми спрощення сітки (наприклад, планарний, колапс) для швидкої генерації LOD-версій. Крім того, інструменти для ретопології та UV-розгортки в Blender є основними для підготовки чистої, оптимізованої сітки під анімацію та запікання текстур. Активний розвиток інструментів (особливо Geometry Nodes) дозволяє створювати процедурні системи для оптимізації всередині самого середовища.

Autodesk Maya та 3DSMax є стандартом у великих (AAA) студіях, слугуючи головними хабами для збірки, анімації та експорту фінальних, оптимізованих ассетів. Обидва пакети пропонують вбудовані інструменти для генерації LOD та спрощення сітки, часто інтегровані з ігровими рушіями через спеціалізовані плагіни (наприклад, FBX-експорт та Data Prep для Unreal/Unity). Maya є незамінною для оптимізації анімованих моделей, як от персонажів, оскільки дозволяє ретельно налаштовувати скінінг, зменшувати кількість кісток (joints) та оптимізувати ваги (weights), що прямо впливає на продуктивність CPU під час анімації. 3DSMax історично сильний у роботі з візуалізацією великих об’єктів та має потужні інструменти для оптимізації оклюзійних карт та карт освітлення по розгортці, які є критично важливими для оптимізації освітлення.

Хоча 3DCoat може використовуватися для скульптингу, його найважливіша роль у оптимізації полягає у двох сферах: автоматизована ретопологія та UV-розгортка. 3Dcoat відомий своїми інструментами авторетопології, які часто дають кращі результати, ніж деякі вбудовані функції інших програм, створюючи чисту чотирикутну сітку, необхідну для ефективного скінінгу та запікання карт нормалей. Високоефективні алгоритми UV-розгортки дозволяють мінімізувати шви та оптимізувати простір текстури, що безпосередньо покращує якість текстурного стрімінгу та кешування GPU.

Для текстурування існують такі програми як Substance 3D Painter, Substance 3D Designer та інші. У першій реалізовано функції оптимізації UV і автоматичного створення LOD для текстур. Крім того, він інтегрований з рушіями Unity та Unreal, що значно спрощує процес експорту текстур.

Substance 3D Painter – це центральний інструмент для створення PBR-матеріалів. Його оптимізаційні функції включають запікання карт високої якості (ефективне та точне запікання нормалей, оклюзії та кривизни) та оптимізований експорт (підтримка експорту текстур у форматі, оптимізованому для пам’яті GPU та цільового рушія, наприклад, Unity або Unreal Engine).

Substance 3D Designer – дозволяє створювати матеріали з нуля за допомогою вузлових графів. Це забезпечує високу гнучкість у створенні текстур

з різною роздільною здатністю на вимогу. Ця гнучкість є ключовою для оптимізації, оскільки вона дозволяє використовувати матеріали у роздільній здатності 4K для головних об’єктів та 1K для фонових об’єктів, не зберігаючи багато статичних копій матеріалів.

Для рендерингу та тестування використовують Marmoset Toolbag, Unity та Unreal Engine. Вони дозволяють аналізувати вплив оптимізації на FPS, полігональність і GPU навантаження.

Unreal Engine 5 – ігровий рушій, який найчастіше використовується для створення комп’ютерний AAA-проєктів. Для оптимізації геометрії він має потужний інтегрований та автоматизований інструмент Nanite. Вбудовані засоби профілювання («Stat GPU», «Stat Unit») дозволяють розробникам точно ідентифікувати «вузькі місця» продуктивності [25]. Зазвичай це draw calls, складність шейдерів або навантаження на освітлення. Інструмент «Mesh Simplification Tool» дозволяє проводити традиційну ручну оптимізацію та генерування LOD-груп для моделей, які не підтримуються Nanite.

Unity – також ігровий рушій, але його частіше використовують під мобільні, VR/AR або інді-ігри. Основний компонент для ручного та автоматизованого управління рівнями деталізації – це LOD Group component. Також рушій має легкий, оптимізований конвеєр рендерингу, який є стандартом для мобільної розробки та VR/AR – Universal Render Pipeline (URP). Спеціалізовані системи для управління ассетами, які забезпечують їхнє ефективне завантаження та вивантаження з пам’яті, що є критичним для мобільної оптимізації називаються Asset Bundles та Addressables System. Функції Automatic Mesh Simplification, подібні до Godot 4.3 (Godot Engine, 2024), які автоматизують спрощення сіток також є важливими інструментами.

Marmoset Toolbag це переважно рендерер, але також він слугує еталонним інструментом для перевірки та профілювання оптимізованих ассетів перед їхнім фінальним імпортом. Він дозволяє точно виміряти навантаження на рендеринг PBR-матеріалів та коректність запікання.

Отже, сучані методи розробки тривимірної графіки стають все більше орієнтованими на інтеграцію та автоматизацію, що дозволяє оптимізовувати моделі ще на етапі створення, а не лише після завершення всього процесу.

## 1.5 Постановка задачі дослідження

Таким чином, оптимізація тривимірних моделей є актуальною проблемою, що потребує системного підходу, адже питання ефективного відображення складних тривимірних сцен без втрати якості набуває особливого значення. Прийнято рішення щодо дослідження та порівняння сучасних методів оптимізації тривимірних моделей, а також глибокого аналізу ефективності вибраних методів.

Об’єктом дослідження є процес оптимізації тривимірних моделей у середовищах реального часу.

Предметом дослідження є методи та алгоритми зменшення навантаження на графічний процесор при збереженні візуальної якості моделей.

Метою дослідження є порівняння методів оптимізації тривимірних моделей шляхом розгляду ефективності даних методів у сучасних ігрових рушіях.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз літературних джерел і визначити основні підходи до оптимізації тривимірних моделей;

– дослідити сучасні програмні засоби для реалізації оптимізації тривимірних моделей;

– порівняти ефективність різних методів оптимізації тривимірних моделей;

– провести тестування та оцінити результати за допомогою показників FPS, GPU навантаження та якості відображення.

# 2 МАТЕМАТИЧНІ МОДЕЛІ МЕТОДІВ ТА ТЕХНОЛОГІЙ ОПТИМІЗАЦІЇ ТРИВИМІРНИХ МОДЕЛЕЙ

## 2.1 Метод оптимізації за допомогою зменшення полігональності

Зменшення полігональності є одним із найважливіших і найефективніших методів оптимізації тривимірних моделей. Це дозволяє суттєво знизити обсяг обчислень, необхідних для рендерингу сцени, без втрати візуальної якості. Спрощення геометрії є життєво важливим для забезпечення продуктивності та стабільної роботи рушіїв у сучасних умовах комп'ютерної графіки, де скульптовані або скановані об'єкти можуть містити мільйони полігонів.

Полігональність тривимірної моделі визначається як кількість елементарних геометричних одиниць, полігонів, зазвичай трикутників, з яких складається поверхня об'єкта. Під час візуалізації кожен полігон проходить різні етапи, включаючи трансформацію, освітлення, відсікання, текстурування та інші, тому загальне навантаження на графічний процесор прямо залежить від кількості полігонів $M$:

$$M = \sum_{i=1}^{n} P_i, \tag{2.1}$$

де $P_i$ – кількість полігонів і-го об'єкта;

$n$ – кількість об'єктів сцени.

Збільшення $M$ пропорційно зменшує обсяг геометричних обчислень, одночасно збільшуючи швидкість рендерингу [26]. Щоб уникнути спотворення форми або силуету моделі, це спрощення має відбуватися під контролем.

### 2.1.1 Метод ручної ретопології

Ретопологія (retopology) – це процес створення нової, оптимізованої сітки поверхні. Цей процес повторює форму вихідної високополімерної моделі, але має значно меншу кількість полігонів і правильну топологію. Її мета полягає в тому, щоб отримати чисту, рівномірну сітку, яка може легко деформуватися під час анімації та гарантує якість запікання карт нормалей.

Коли об’єкт має надлишкову геометрію (до декількох мільйонів трикутників), ретопологія зазвичай проводиться після скульптингу або тривимірного сканування. Художник зберігає ключові форми і зменшує кількість полігонів у десятки або сотні разів, створюючи ручну версію low-poly.

Коли об’єкт має надлишкову геометрію (до декількох мільйонів трикутників), ретопологію зазвичай проводять після скульптингу або тривимірного сканування. Художник зберігає ключові форми і зменшує кількість полігонів у десятки або сотні разів, створюючи ручну версію low-poly.

Зменшення кількості полігонів без втрати форми, створення логічного потоку ребер (edge flow), підготовка моделі до анімації, покращення запікання карт нормалей і замикання навколишнього середовища є основними цілями ретопології.

Доступні ручні та автоматизовані інструменти ретопології в програмах Blender, ZBrush, 3DCoat і Maya. Наприклад, ZRemesher у ZBrush і QuadRemesher у Blender використовують алгоритми аналізу кривизни та напрямків поверхні, щоб автоматично створювати топології.

На рисунку 2.1 зображено порівняння силуетів high-poly моделі (12,5 млн полігонів) і її ретопологізованої версії (14,5 тис. полігонів).

Візуальна різниця не дуже помітна, хоча кількість полігонів зменшилась більше ніж у 1000 разів.

Рисунок 2.1 – Силуети high-poly та low-poly моделей

### 2.1.2 Метод декімації та редукції полігонів

Декімація – автоматичне спрощення сітки на основі геометричного аналізу. Цей метод використовується, коли модель не потребує анімації або складної топології. Намагаючись зменшити середньоквадратичне відхилення між новою та початковою поверхнею, алгоритм поступово видаляє або об’єднує ребра та вершини.

Математично це можна подати як задачу мінімізації функції похибки форми.

$$E = \sum_{v_i \in V} (d(v_i, S))^2, \tag{2.2}$$

де $v_i$ – вершини нової сітки;

$S$ – початкова поверхня;

$d(v_i, S)$ – відстань між вершиною та поверхнею.

Quadric Error Metrics (QEM), розроблений Garland і Heckbert у 1997 році,

зараз є найпоширенішим підходом [27]. Використовуючи матрицю похибки, він визначає «вартість» видалення ребра.

$$Q_i = \sum_{p \in P(v_i)}^{n} (a_p x + b_p y + c_p z + d_p)^2, \quad (2.3)$$

де коефіцієнти $a_p$, $b_p$, $c_p$, $d_p$ описують площину трикутника $p$, до якої належить вершина $v_i$.

Цей метод може зменшити кількість полігонів на 70–90%, зберігаючи загальний профіль моделі [28].

Він використовується в Blender як Decimate Modifier, в Unreal Engine як компонент LOD Generator і в Simplygon як окремий пакет із розширеними параметрами збереження нормалей, UV і матеріалів [29].

Методи ретопології дещо схожі, але мають свої відмінності, які представлено у таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика методів ретопології та декіміації.

| **Параметр** | **Ретопологія** | **Декімація** |
|---|---|---|
| Тип процесу | Ручний або напівавтоматичний | Повністю автоматичний |
| Контроль над сіткою | Високий | Обмежений |
| Якість деформації | Висока | Низька |
| Придатність до LOD | Середня | Висока |
| Час виконання | Високий | Низький |
| Ризик артефактів | Мінімальний | Високий |

Ретопологія частіше використовують для оптимізації персонажів та динамічних об’єктів, тоді як декіміація працює добре з фоновими або статичними моделями (будівлі, реквізит, ландшафти). У реальних робочих процесах їх часто поєднують. Локальна ручна ретопологія проблемних областей виконується після автоматичного спрощення.

Метод зменшення полігональності є фундаментальним підходом до оптимізації тривимірних моделей. Ретопологія забезпечує високу якість сітки, придатну до анімації, тоді як декіміація – швидке автоматичне спрощення геометрії. Комбіноване використання цих методів дозволяє досягти оптимального співвідношення між якістю та продуктивністю, що є особливо важливим для ігор, віртуальної реальності та мобільних додатків.

## 2.2 Метод оптимізації за допомогою рівнів деталізації

Технологія рівнів деталізації (LOD) є однією з найважливіших стратегій оптимізації тривимірних моделей у реальному часі. Він передбачає автоматичне або напівавтоматичне зниження складності шейдерів, геометрії, текстури або відстані до камери залежно від кута огляду, важливості елемента у сцені або відстані до камери. Цей метод має на меті зменшити навантаження на графічний процесор, одночасно не втрачаючи якості зображення для користувача.

Вперше ідея рівнів деталізації була запропонована Кларком у 1976 році, але лише з появою сучасних графічних рушіїв (Unreal Engine, Unity, Godot) вона стала широко використана в реальному часі [30]. У методі використовується одна версія тривимірної моделі, починаючи з найдетальнішого LOD0 і закінчуючи найпростішими LOD3 і LOD4, між якими відбувається динамічне перемикання залежно від відстані до камери (рис. 2.2).

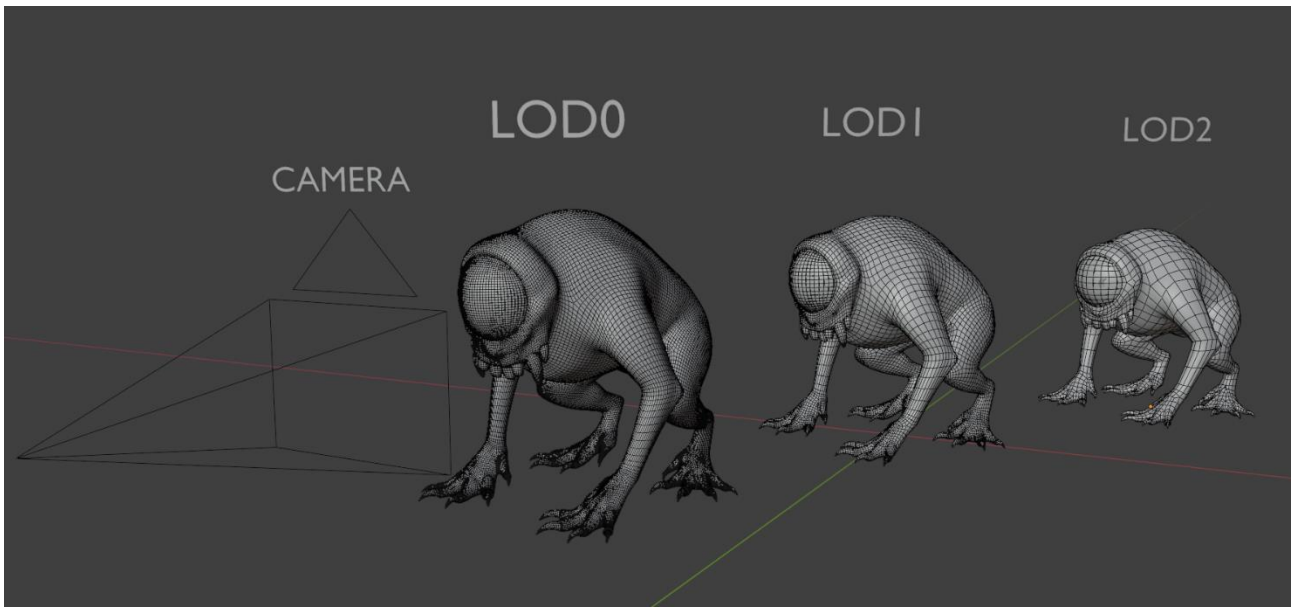



Рисунок 2.2 – Динамічне перемикання між LOD в залежності від відстані до камери

Для кожного рівня визначається поріг видимості $D$, за яким застосовується відповідна модель:

$$M = \begin{cases} LOD0, & \text{якщо } d < D_1, \\ LOD1, & \text{якщо } D_1 \leq d < D_2,, \\ LOD2, & \text{якщо } d \geq D_2. \end{cases} \quad (2.4)$$

де $d$ – поточна відстань від камери до об’єкта [31].

Використання LOD дозволяє значно скоротити кількість відтворюваних трикутників у сцені. Наприклад, у великій відкритій локації лише 10–15% моделей одночасно відображаються у високій деталізації [32].

Сучасні системи LOD розділяють на кілька основних типів: геометричний, текстурний, шейдерний або матеріальний та імпостерний.

Геометричний LOD – це спрощення самої сітки тривимірної моделі. Зазвичай використовується декіміація або заміна моделі на іншу, створену вручну. Цей тип найбільш поширений для персонажів, техніки, архітектури.

Текстурний LOD (MIP-mapping) означає зменшення роздільної здатності текстур при віддаленні об’єкта. У рушіях це реалізується через завантаження

менш детальних MIP-рівнів текстур, що зменшує споживання відеопам’яті.

Шейдерний або матеріальний LOD означає спрощення шейдерів (менше джерел світла, відключення нормалей або парлаксу). Застосовується в іграх ААА-класу, де складні матеріали використовуються лише поблизу камери.

Імпостерний LOD – це заміна тривимірного об’єкта на плоске 2D-зображення (білборд). Це радикальний, але надзвичайно ефективний спосіб економії ресурсів, часто використовується для віддалених дерев, натовпу або архітектури.

Алгоритми LOD використовують метрику помилки для визначення, коли саме потрібно знизити рівень деталізації. Найпоширенішим підходом є Quadric Error Metrics (QEM), який вимірює геометричну різницю між оригінальною поверхнею та спрощеною моделлю. Однак у реальному часі використовують спрощені евристики.

$$LODIndex = \left\lfloor \frac{d - D_{min}}{D_{max} - D_{min}} \times N_{LOD} \right\rfloor, \tag{2.5}$$

де $N_{LOD}$ – кількість рівнів деталізації.

У сучасних рушіях (Unreal Engine, Unity, Godot) перемикання відбувається гладко (cross-fade LOD), адже система змішує два сусідніх рівні протягом кількох кадрів, запобігаючи ефекту «підстрибування» (popping).

У Unreal Engine 5.3 є вбудований LOD Generator, який автоматично створює кілька рівнів спрощення для кожної моделі. У процесі експорту low-poly версій у Blender або Maya використовується назва правил (наприклад, object_LOD0 або object_LOD1), що дозволяє рушію автоматично підвантажувати відповідний рівень.

У Unity 2022+ реалізується компонент LOD Group, який підтримує різноманітні рівні геометрії та імпостери. Користувач може вказати відстань як процентне співвідношення до поля зору камери. Для забезпечення плавного переходу між рівнями додано підтримку поєднання crossfade з Unity 2021.3.

У системах оптимізації на рівні рушія LOD інтегрується з іншими

підходами, такими як «Culling» (відсікання невидимих об’єктів), «Instancing» (інстанціювання однакових об’єктів) та «Streaming» (потокове завантаження даних).

Ефективність використання LOD визначається співвідношенням між кількістю полігонів і відстанню до камери. У середньому, залежність FPS можна апроксимувати функцією:

$$FPS(d) = k\frac{1}{1+\alpha M(d)}, \tag{2.6}$$

де $M(d)$ – кількість полігонів, які відображаються при відстані $d$,

$\alpha$ – коефіцієнт залежності між геометрією та навантаженням,

$k$ – базовий FPS при мінімальному навантаженні [33].

У багатокористувацьких середовищах LOD має додаткову складність, так як необхідно уникати розсинхронізації між клієнтами. Рушії часто використовують серверний LOD, коли саме сервер визначає, які об’єкти повинні бути у високій або спрощеній якості для кожного клієнта. Цей метод значно прискорює синхронізацію та значно зменшує навантаження на мережу.

Збереження чесності геймплею залишається важливою проблемою. Один гравець може бачити більше деталей, ніж інший, що може змінити змагальний баланс. Таким чином, у сучасних онлайн-іграх використовується адаптивний, але синхронізований LOD, який обмежує діапазон деталізації сервером.

Одним із найкращих методів оптимізації тривимірної графіки є використання рівнів деталізації (LOD). Залежно від умов відображення ця технологія дозволяє зменшити кількість полігонів, обсяг текстурних даних і складність шейдерів. LOD, у поєднанні з ретопологією, текстурною оптимізацією та стрімінгом ресурсів, гарантує стабільну частоту кадрів, особливо у великих відкритих світах і мережевих іграх.

Таким чином, LOD є не лише технічним, а й стратегічним інструментом оптимізації у сучасній ігровій графіці.

## 2.3 Методи оптимізації тектур та матеріалів

Одним із основних напрямків підвищення продуктивності тривимірних сцен є оптимізація текстур і матеріалів. Текстурна оптимізація спрямована на зменшення навантаження на відеопам’ять (VRAM) і скорочення часу вибірки даних під час рендерингу. Це відрізняє геометричні методи, такі як декіміація чи LOD, які зменшують кількість полігонів.

Завдяки технології PBR (Physically Based Rendering) сучасні ігрові рушії, такі як Unreal Engine 5, Unity HDRP і CryEngine V, можуть забезпечувати майже фотореалістичне відтворення поверхонь. Однак для досягнення цієї якості потрібна ефективна система управління текстурами.

Для того, щоб повністю зрозуміти, як працюють текстури на тривимірних моделях, треба розібратися з поняттям UV-мапінгу.

UV-мапінг – це процес проєціювання тривимірної геометрії на двовимірну площину для подальшого накладання текстури. Кожна вершина тривимірної моделі має координати (u,v), які відповідають позиції на площині текстури. Математично UV-мапування визначається функцією:

$$f : (x, y, z) \rightarrow (u, v), \tag{2.7}$$

де $(x, y, z)$ – координати точки в просторі моделі;

$(u, v)$ – відповідні координати на текстурній площині.

Завдання оптимізації полягає у мінімізації спотворень між геометрією об’єкта та текстурною площиною. Цю задачу можна описати через енергетичну функцію розтягнення (stretch energy):

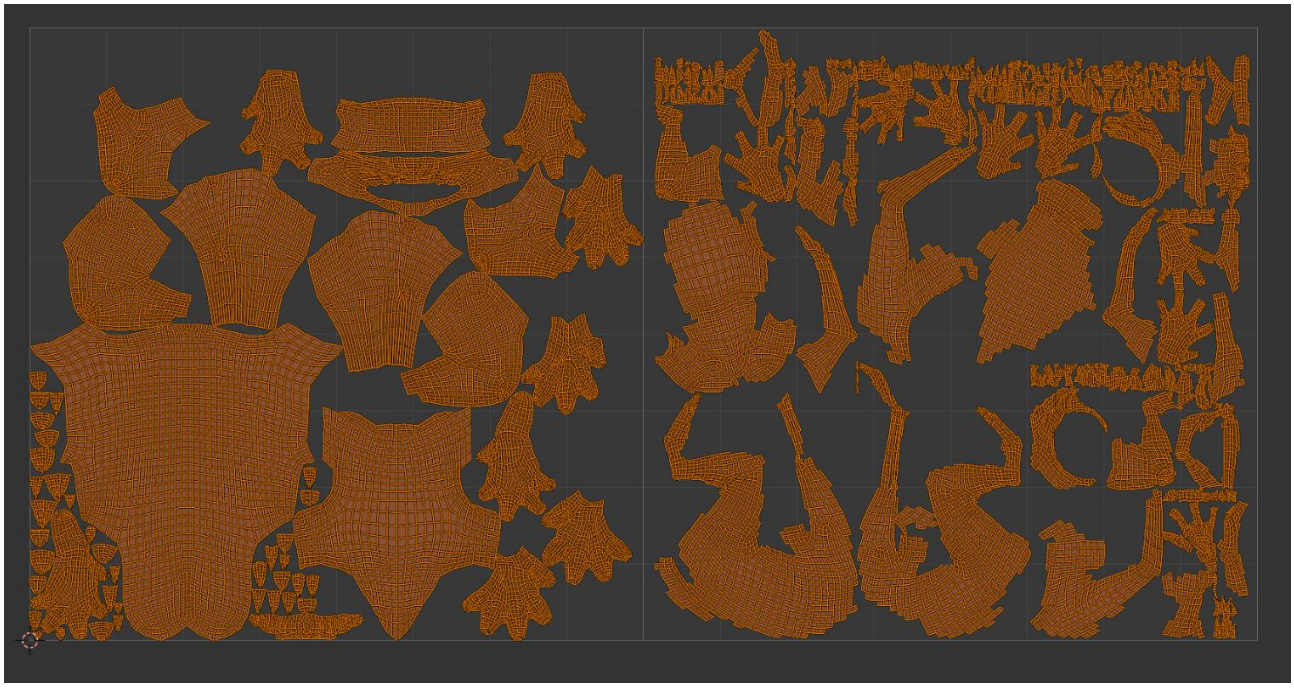
$$E_s = \sum_{i=1}^{n} ||J_i - I||^2, \tag{2.8}$$

де $J_i$ – матриця Якобі для локального відображення,

$I$ – одинична матриця.

Чим менше значення $E_s$, тим рівномірніше текстура розподіляється по поверхні.

Сучасні алгоритми UV-розгортки (наприклад, Angle Based Flattening або Least Squares Conformal Maps) дозволяють мінімізувати ці спотворення. Інструменти, такі як RizomUV, Blender Smart UV Project, Maya UV Editor, використовують подібні принципи з автоматичним визначенням «швів» і відступів (margins), що є важливим для оптимізації текстурної пам’яті. На рисунку 2.3 (а) наведено приклад ручної UV-розгортки. На рисунку 2.3 (б) наведено приклад автоматичної UV-розгортки за допомогою вбудованого у Blender інструменту Smart UV Project.



(а) (б)

Рисунок 2.3 – Приклади UV-розгортки одного об’єкта різними методами (а) приклад ручної UV-розгортки; (б) приклад автоматичної UV-розгортки за допомогою вбудованого у Blender інструменту Smart UV Project

Розглянемо тепер різні методи оптимізації текстур та матеріалів. Занадто великі текстури часто є основною причиною високого часу завантаження сцени та низької частоти кадрів. 4K-текстури не потрібні для об’єктів, які займають

невелику частину екрану. У цьому випадку застосовуються MIP-мапи або динамічне масштабування текстур.

MIP-мапи, також відомі як Multum in Parvo, забезпечують багаторівневе збереження текстур різної роздільної здатності. Залежно від відстані до камери рушій автоматично вибирає рівень деталізації під час рендерингу. Це дозволяє зменшити використання пам’яті та запобігти «мерехтіння».

Динамічне масштабування текстур, також відоме як texture streaming – це технологія, яка дозволяє рушію додавати текстури вищої якості лише тоді, коли це необхідно для кадру. Наприклад, використання Virtual Texture Streaming дозволяє Unreal Engine обробляти сцени з тисячами об’єктів без збільшення обсягу VRAM.

Система PBR (Physically Based Rendering) базується на фізичних моделях взаємодії світла з матеріалами. Основна мета – створити матеріали, які поводяться однаково в будь-якому освітленні. PBR використовує кілька текстурних карт:

– albedo (diffuse) – базовий колір без тіней;

– metallic – визначає, чи поверхня металева;

– roughness – описує мікрогеометрію відбивання;

– normal – імітує деталі без додаткових полігонів;

– AO (ambient occlusion) – моделює м’які тіні у заглибленнях.

Більшість сучасних рушіїв підтримують цю систему, яка є загальноприйнятою в галузі, однак навіть у цьому випадку оптимізація можлива. Наприклад, можна поєднати кілька карт у один файл (наприклад, поєднання каналів, R канал = AO, G канал = Roughness, B канал = Metallic) і зменшити динамічний діапазон для неважливих об’єктів, таких як фон.

Як приклад можна привести гру «Horizon Forbidden West» (2022), де використовуються комбіновані PBR-текстури для фонового оточення, що дозволило зменшити обсяг використовуваної пам’яті на 30% [34].

Створення атласів, тобто великих текстур, які складаються з кількох менших текстурних елементів, є ще одним методом оптимізації. Атласи

зменшують кількість перемикань матеріалів (draw calls) і підвищують ефективність GPU при відтворенні сцен із великою кількістю дрібних об’єктів.

Алгоритм створення атласу можна розглядати як задачу упаковки прямокутників (rectangle packing) у межах фіксованої площі текстури. Хоча ця задача NP-складна з математичної точки зору, існують евристичні методи, такі як MaxRects, Guillotine або Shelf Algorithm, які можуть забезпечити високий коефіцієнт заповнення площі (більше 90%) [35]. Завдяки атласам рушій може відрендерити десятки об’єктів за один виклик (рис. 2.4).

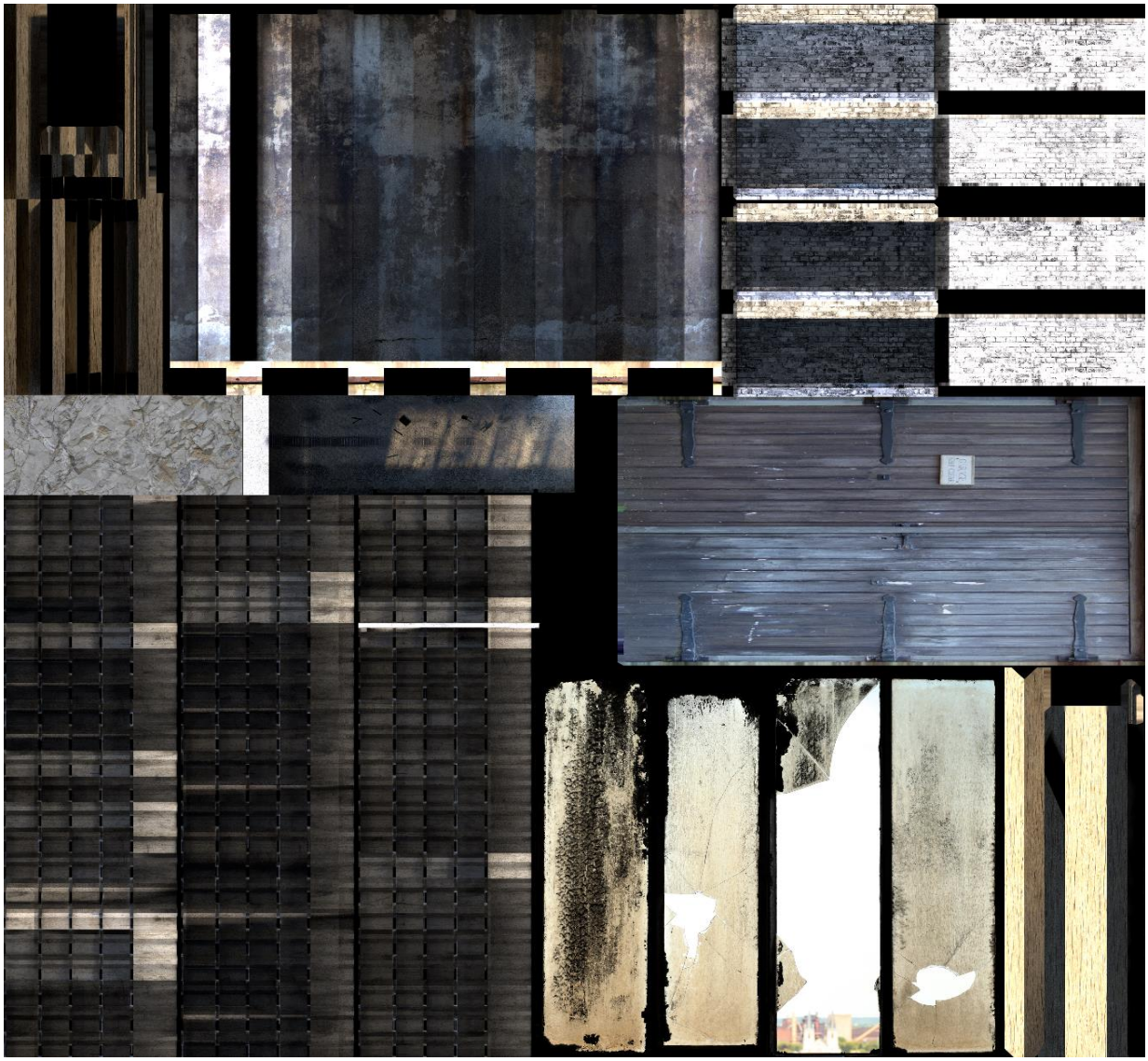


Рисунок 2.4 – Приклад тексутрного атласу для локації

У практичних тестах Unreal Engine 5 продемонстрував зростання FPS на 25–40% після переходу з індивідуальних текстур на атласи [36].

Окрім текстур, велику роль відіграє оптимізація матеріальних шейдерів. Матеріали можуть мати безліч параметрів, таких як прозорість, емісія та підповерхневе розсіювання, і кожен із цих параметрів включає інструкції до GPU. Для зменшення навантаження використовують такі методи:

– об’єднання матеріалів для схожих об’єктів;

– використання матеріальних інстансів (material instances) у Unreal Engine;

– замороження параметрів (bake values), щоб прибрати непотрібні обчислення;

– створення lightweight shaders для дрібних або далеких об’єктів.

Одним із найкращих способів зменшити навантаження на графічну систему без значного зниження якості візуалізації є оптимізація текстур і матеріалів. Завдяки правильній UV-розгортці, використанню атласів, багаторівневих MIP-текстур і раціональному підходу до PBR-матеріалів можливо скоротити споживання відеопам’яті у 2–4 рази, що є критично важливим для ігрових рушіїв, VR-проєктів, інтерактивних архітектурних візуалізацій тощо.

Таким чином, ці методи не лише забезпечують баланс між продуктивністю та якістю, але й служать основою для подальшої оптимізації сцени на рівні потокового завантаження та LOD.

## 2.4 Методи оптимізації за допомогою текстурних карт

Одним із ключових напрямів оптимізації тривимірної графіки є імітація геометричних деталей без збільшення кількості полігонів. Це досягається завдяки використанню спеціальних текстур, таких як карти нормалей (normal maps), карти висот (height maps) та карти навколишнього затемнення (AO maps). Такі карти дозволяють передавати дрібні деталі (рельєф, подряпини, текстуру матеріалу) без потреби у високополігональній геометрії. Основний принцип

полягає в тому, що візуальні властивості поверхні описуються векторним полем нормалей, яке впливає на взаємодію світла з поверхнею, створюючи ілюзію рельєфу. На рисунку 2.5 наведено приклад вигляду карти нормалей.

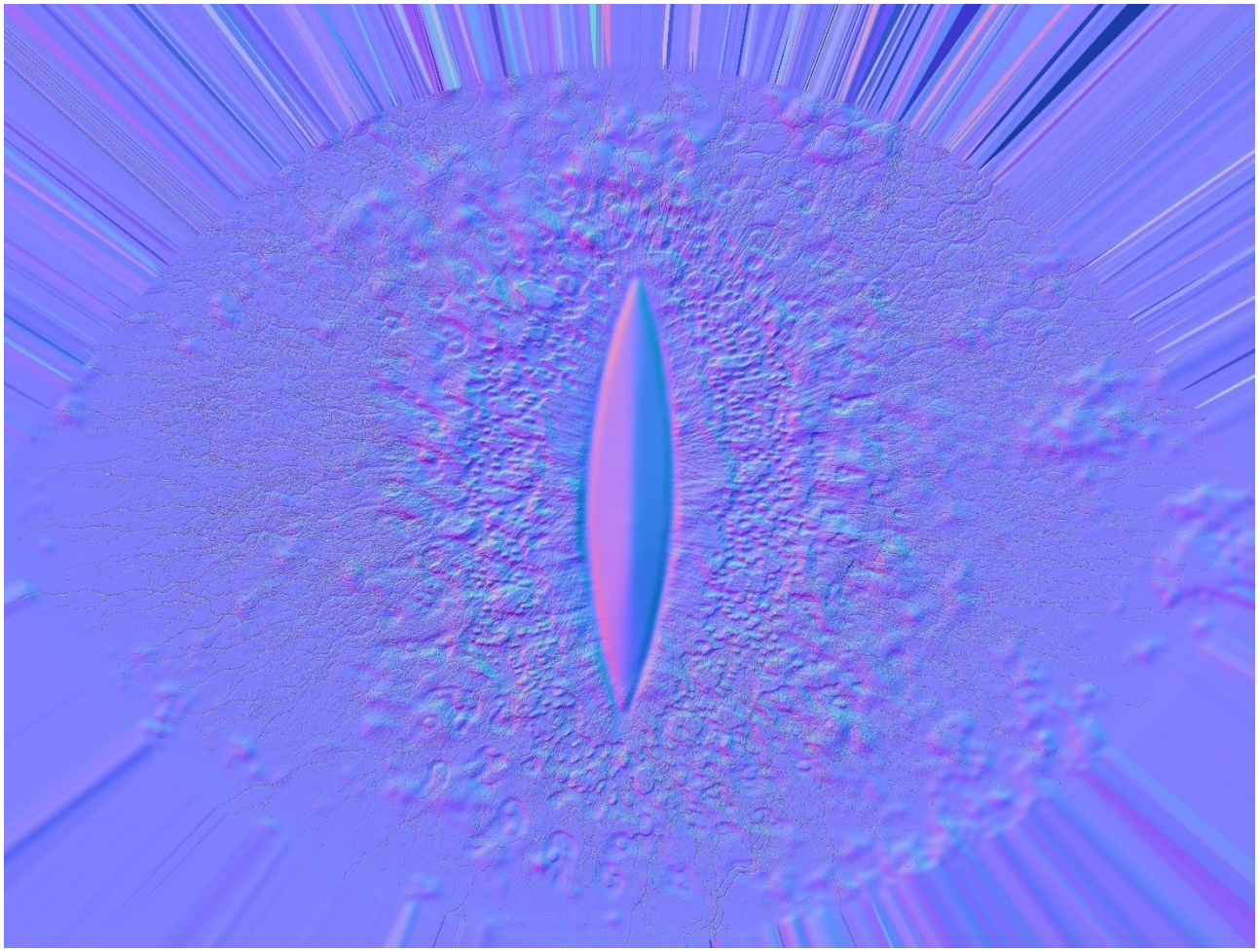


Рисунок 2.5 – Приклад вигляду карти нормалей

Математично це можна подати як зміну вектора нормалі $\vec{N}_{pixel}$ у просторі пікселів.

$$\vec{N}_{pixel} = T\vec{n}_t + B\vec{n}_b + N\vec{n}_n, \tag{2.9}$$

де $T$, $B$, $N$ – ортонормований базис (tangent, bitangent, normal),

$\vec{n}_t, \vec{n}_b, \vec{n}_n$ – компоненти вектора з карти нормалей у просторі тангентів.

Таким чином, для кожного пікселя нормаль може бути змінена незалежно від геометрії моделі, створюючи деталізацію візуально, але не геометрично [37]. На рисунку 2.6 наведено приклад застосування карти нормалей, запеченої з high-poly (12,5 млн полігонів) на low-poly модель (14,5 тис полігонів).

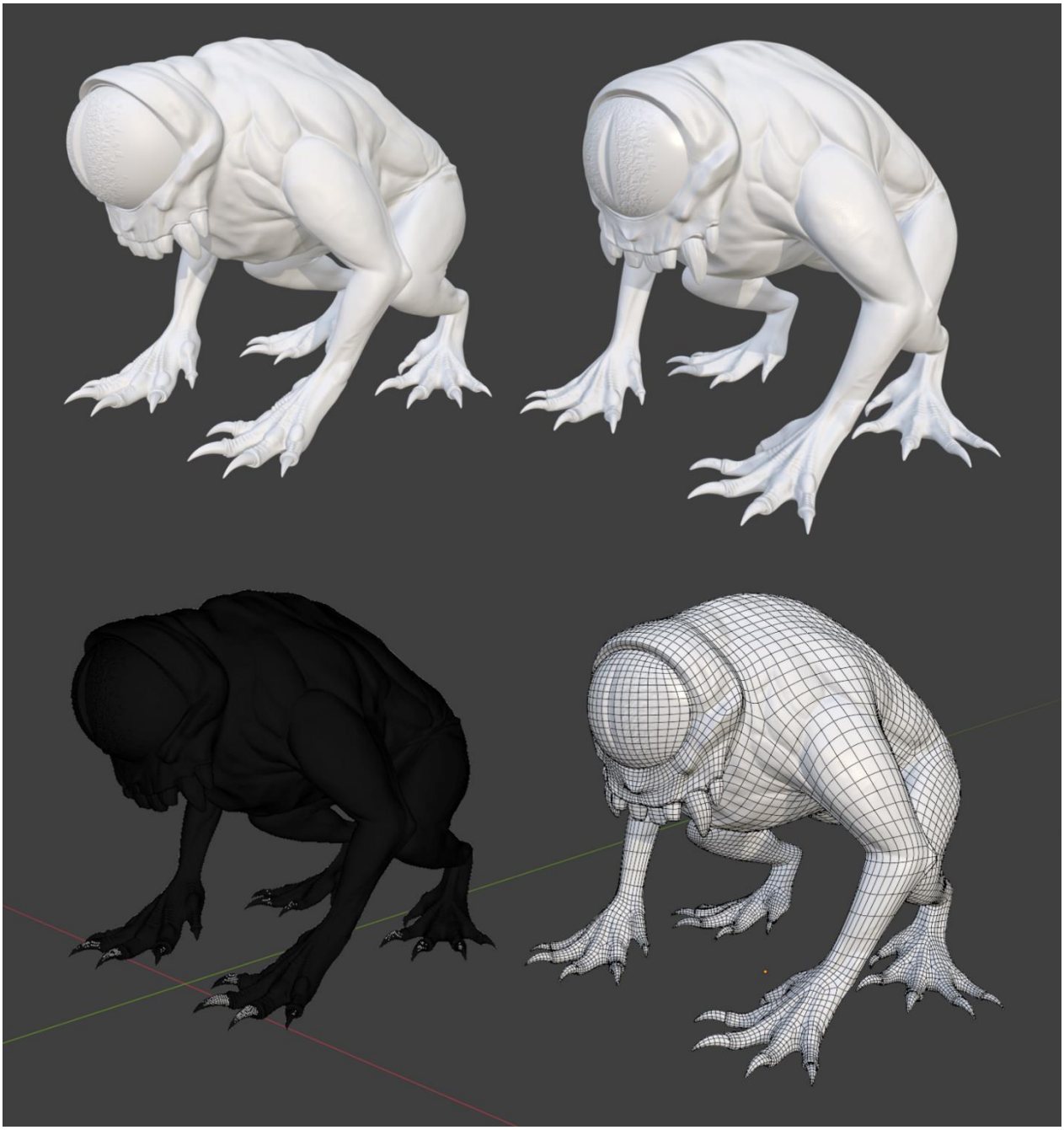


Рисунок 2.6 – Застосування карти нормалей на моделі

З точки зору рендерингу, карта нормалей змінює напрямок відбиття світла в моделі освітлення, наприклад, моделі Фонга або моделі Кука-Торренса.

Це дозволяє досягати високої візуальної деталізації навіть при низькому полігональному бюджеті. У рівнянні відбиття Фонга вектор $N$ замінюється локальним вектором з карти нормалей.

$$I = k_a I_a + k_d (L \times N) I_d + k_s (R \times V)^{n_s} I_s. \quad (2.10)$$

Таким чином, освітлення реагує на дрібні рельєфні зміни, хоча фактична геометрія не змінюється. Це дозволяє досягати високої візуальної деталізації навіть при низькому полігональному бюджеті.

Карти висот – це текстури у градаціях сірого, де яскравість визначає відстань точки поверхні від базової площини.

Двома основними способами їх використання є displacement mapping, яке фізично змінює геометрію поверхні під час рендерингу або препроцесингу та parallax (relief) mapping, яке створює оптичну ілюзію рельєфу без фактичного переміщення вершин. У класичному displacement mapping зміна координати вершини $P$ задається як:

$$P` = P + h(u, v) \times N, \quad (2.11)$$

де $h(u, v)$ – значення висоти з карти у координатах UV,

$N$ – нормаль поверхні.

Хоча displacement mapping забезпечує найвищу реалістичність, він є обчислювально затратним. У реальному часі частіше використовується Parallax Occlusion Mapping (POM), який модифікує координати UV замість вершин, забезпечуючи ілюзію об’єму без геометричних витрат.

За результатами тестів NVIDIA у 2021 році завдяки displacement mapping вдалось досягти зниження FPS на 45% про високій частоті деталей, а використовуючи normal mapping втрата продуктивності склала лише 5-7% за аналогічної візуальної якості.

Текстури, створені алгоритмічно без використання зображень, називають процедурними картами. Вони дозволяють забезпечувати нескінченну повторюваність і варіативність, не збільшуючи обсяг пам’яті. У більшості рушіїв і редакторів, таких як Substance Designer, Blender Shader Editor та Houdini, вони описуються вузловими графами (node-based) або математичними функціями шуму [38].

Одним із найвідоміших прикладів є шум Перліна (Perlin Noise), який використовується для створення природних текстур, таких як камень, хмари, шкіра, пісок. Його основна формула:

$$f(x, y, z) = \sum_{i=1}^{n} a_i \times noise(b_i x, b_i y, b_i z), \quad (2.12)$$

де $a_i$ – амплітуда,

$b_i$ – частота для і-го октаву.

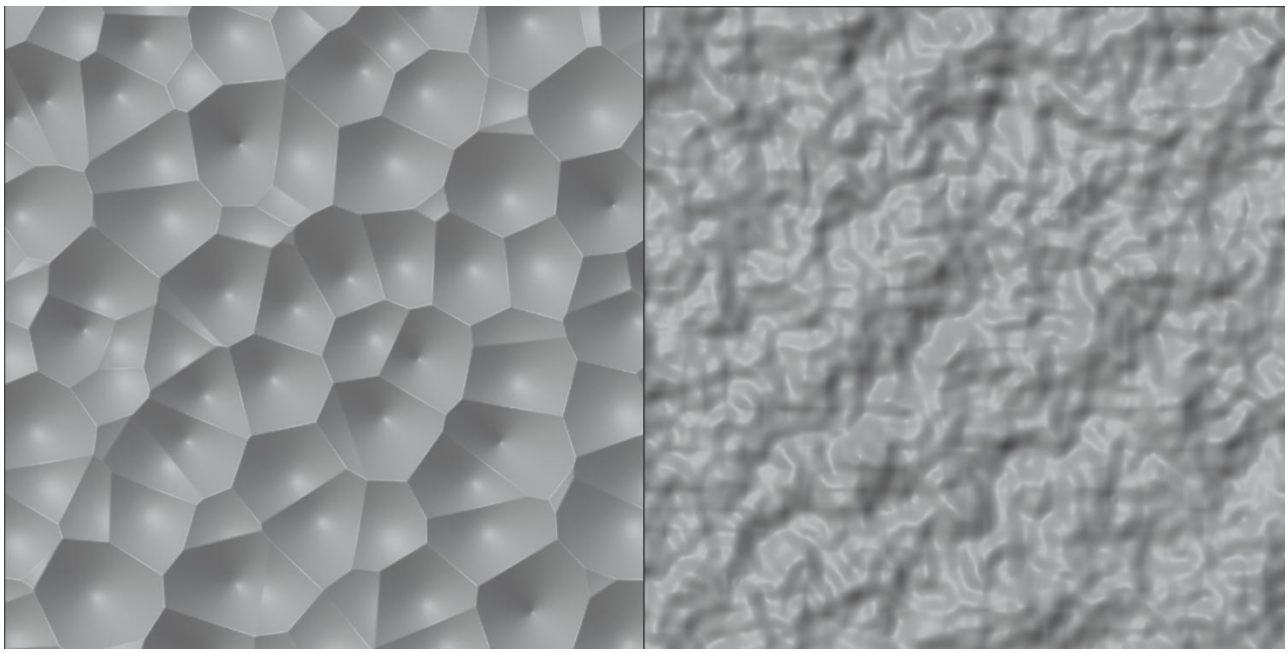
Для більш реалістичних результатів використовується шум Worley, який генерує текстури типу «клітин» (рис. 2.7 (а)), а також фрактальний шум, який додає детальність на різних масштабах (рис. 2.7 (б)).

Процедурні карти мають багато переваг, таких як мінімальна кількість пам’яті (зберігається формула, а не піксельні дані), нескінченна роздільна здатність (можна масштабувати без артефактів) і динамічна адаптація (може змінюватися залежно від умов сцени).

З недоліків можна виділити високе навантаження на CPU/GPU при генерації в реальному часі, тому карти зазвичай попередньо запікаються у звичайні тексутри.

У сучасних пайплайнах створення тривимірного контенту існують численні інструменти для генерації карт нормалей і висот. Найбільш популярні з них Substance 3D Painter / Designer, Marmoset Toolbag та Blender.

У рушіях (Unity, Unreal) існують вбудовані системи Tangent Space Normal Mapping, Virtual Heightfields, Dynamic Normal Blending, що дозволяють комбінувати карти у реальному часі.



(а) (б)

Рисунок 2.7 – Приклади різних типів шумів

(а) шум Worley; (б) фрактальний шум

Крім того, напрям генерації текстури на основі штучного інтелекту активно розвивається. Цей напрям використовує нейронні мережі для створення карт нормалей або висот із звичайних зображень. Модель Stable Diffusion + Normal Map Converter або AI Texture Generator є прикладами цього. Такі підходи дозволяють суттєво скоротити час розробки матеріалів та автоматизувати частину оптимізаційного процесу.

Отже, методи оптимізації за допомогою тексутрних карт дозволяють передавати високий рівень деталізації без необхідності збільшення полігональності, що значно зменшує навантаження на систему рендерингу. Ці методи становлять основу сучасного підходу до оптимізації моделей у реальному часі, за умови використання разом з правильно підібраними PBR-текстурами та UV-мапами. У свою чергу, процедурні карти дозволяють створювати динамічні середовища та адаптивні матеріали, що є особливо важливим для великих ігрових сцен та віртуальних світів.

## 2.5 Методи оптимізації у ігрових рушіях

Сучасні ігрові рушії обробляють велику кількість даних, включаючи моделі, текстури, анімацію, звукові ефекти, скрипти та інші ресурси. Складні ігрові сцени можуть потребувати більше 50–100 ГБ активів, що створює значне навантаження на оперативну пам’ять і сховище. Таким чином, основними напрямами оптимізації у рушіях є стиснення даних (compression), що означає зменшення розміру файлів без значної втрати якості, та потокове завантаження (streaming) – динамічне підвантаження лише тих ресурсів, які потрібні в конкретний момент гри. Ці підходи забезпечують плавність геймплею, зменшення часу завантаження та зниження споживання оперативної пам’яті.

Існує декілька видів стиснення тривимірних моделей. Найпопулярнішими з них є геометричне та текстурне стиснення.

Основною метою геометричного стиснення є зменшення кількості даних, необхідних для опису вершин, нормалей та текстурних координат. Типовими підходами для цього виду є: квантоване кодування координат, стиснення з прогнозуванням та компресія топології.

Квантоване кодування координат означає зменшення точності з float (32-біт) до short (16-біт), що дозволяє знизити обсяг пам’яті вдвічі без помітної втрати якості [39].

Стиснення з прогнозуванням (predictive encoding) – це підхід, де замість збереження абсолютних координат зберігаються різниці між послідовними вершинами (Δ-компресія). Такий підхід ефективно стискає топологічно послідовні сітки.

Компресія топології (topological compression) означає кодування зв’язків між трикутниками як послідовності команд, що відновлюють сітку. Алгоритм «EdgeBreaker» зменшує обсяг топологічних даних до ~2,5 біт на трикутник [40].

Текстури займають до 70% пам’яті у типовій грі, тому їх оптимізація є критичною (табл. 2.2).

Таблиця 2.2 Формати текстурного стиснення

| **Формат** | **Біт/піксель** | **Особливості** | **Рівень якості** |
|---|---|---|---|
| DXT1 / BC1 | 4 | Без альфа-каналу | Середня |
| DXT5 / BC3 | 8 | З альфа-каналом | Добра |
| BC7 | 8 | Висока якість, HDR | Висока |
| ASTC | 4-8 | Адаптивне стиснення | Дуже висока |

Формат ASTC (Adaptive Scalable Texture Compression) є сучасним стандартом, який підтримує блоки 4×4 до 12×12 пікселів і дозволяє балансувати між якістю та розміром. Математично квантування в ASTC виконується через векторне кодування.

$$C = B \times Q^{-1}(I), \tag{2.13}$$

де $C$ – відновлені кольори,

$B$ – базові кольори блоку,

$Q^{-1}$ – інверсія квантування для індексів $I$.

Потокове завантаження – це техніка, при якій ресурси сцени підвантажуються асинхронно з накопичувача у пам’ять залежно від позиції гравця або камери. Ідея полягає в тому, щоб ніколи не завантажувати повну сцену цілком, а лише ті об’єкти, які потрапляють у зону видимості. Основною умовою є:

$$R_{stream} \geq R_{load}, \tag{2.14}$$

де $R_{stream}$ – швидкість потокового завантаження (МБ/с),

$R_{load}$ – середня швидкість споживання даних рушієм.

Якщо умова не виконується, то відбувається «stutter» або затримка кадрів.

Для контролю пам’яті використовується алгоритм найменш нещодавно використаних даних (Least Recently Used — LRU). При досягненні ліміту пам’яті

відбувається видалення об’єктів, які не використовувалися найтриваліший час.

$$C_t = \{r_i \in R : age(r_i) < T_{max}\}, \tag{2.15}$$

де $R$ – множина ресурсів,

$age(r_i)$ – час з моменту останнього використання,

$T_{max}$ – поріг часу зберігання.

Цей принцип лежить в основі texture streaming і geometry streaming у рушіях Unity, Unreal Engine та Frostbite.

Unreal використовує систему Virtual Texturing (VT), яка поділяє текстури на «плитки» (tiles) розміром 128×128 пікселів. Під час рендерингу рушій вибирає лише ті плитки, які потрібні в полі зору, і завантажує їх у пам’ять GPU. Математична модель індексації плиток:

$$T(u, v) = tileIndex\left(\left\lfloor \frac{u}{s} \right\rfloor, \left\lfloor \frac{v}{s} \right\rfloor\right), \tag{2.16}$$

де $s$ – розмір плитки у UV-просторі.

У UE5 також застосовується Nanite, який динамічно стрімить геометрію на основі важливості (importance culling), що дозволяє відображати мільярди трикутників у реальному часі [41].

Unity має систему Addressables & Asset Bundles, яка дозволяє розділяти ресурси на окремі пакети, що підвантажуються за потреби. Потокове завантаження здійснюється асинхронно через API. Це зменшує RAM-використання на 30–50% для великих сцен [42].

Лістинг 2.1 Використання рушієм Unity Addressables & Asset Bundles:

```
Addressables.LoadAssetAsync<GameObject>("Character_LowPoly")
```

Розроблений EA DICE рушій Frostbite використовує «Background Streaming

Threads», які розподіляють ресурси між CPU та GPU потоками. Формула продуктивності визначається як:

$$P_{stream} = \frac{D_{loaded}}{t_{frame}}, \quad (2.17)$$

де $D_{loaded}$ – обсяг даних, завантажених за кадр,

$t_{frame}$ – час одного кадру.

Система динамічно підлаштовує швидкість потоків, щоб уникнути перевантаження диску або GPU.

Оптимальні результати досягаються при комбінуванні стиснення та стрімінгу. Покроково типовий пайплайн виглядає так:

Крок 1. High-poly модель → ретопологія → low-poly.

Крок 2. Запікання карт нормалей і текстур.

Крок 3. Експорт у формат glTF/Draco.

Крок 4. Streaming у рушії з адаптивним рівнем LOD.

Формат glTF 2.0 із Draco Compression дозволяє стискати геометрію в 5–10 разів без візуальної втрати якості (табл. 2.3) [43].

Таблиця 2.3 Найпоширеніше застосування методів стиснення

| **Метод** | **Застосування** |
|---|---|
| Geometry Compression (Draco) | WebGL, мобільні ігри |
| Texture Compression (ASTC) | Всі платформи |
| Virtual Texturing (UE5) | AAA ігри, VR |
| Streaming (LRU) | Open World |
| Hybrid (Compression + Streaming) | Великі проекти |

Методи стиснення та потокового завантаження ресурсів є важливими компонентами сучасної оптимізації тривимірних моделей. Вони підвищують стабільність кадрів у реальному часі, зменшують час завантаження та ефективно

використовують пам’ять. Для рушіїв нового покоління, таких як Unreal Engine 5 і Unity, стандартом є поєднання форматів glTF + Draco, ASTC текстур і Virtual Texturing. Ці методи дозволяють створювати високодеталізовані світи без перевантажування системи, що є основною метою оптимізації у тривимірному виробництві.

# 3 ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ТРИВИМІРНИХ МОДЕЛЕЙ

## 3.1 Обґрунтування вибору середовища дослідження

У процесі побудови експерименту з оптимізації тривимірних моделей вибір середовища дослідження є важливим етапом. Для ефективного аналізу методів оптимізації необхідно мати інструмент, який забезпечує якісну візуалізацію, а також гнучкі інструменти для вимірювання продуктивності, управління ресурсами та керування рівнями деталізації. У рамках кваліфікаційної роботи як основне середовище було обрано Unity Engine, оскільки він є одним із найпоширеніших інструментів для створення інтерактивного тривимірного контенту.

Кросплатформний рушій Unity дозволяє створювати двовимірні, тривимірні, VR- та AR-додатки для більш ніж 25 платформ, включаючи Windows, macOS, Android, iOS, WebGL та консолі. У сфері оптимізації поєднання гнучкого рендерингу та зручних інструментів для аналізу продуктивності в реальному часі є основною перевагою Unity. Згідно з офіційними даними Unity Technologies [44], понад 70% мобільних ігор у світі створено саме на цьому рушії. Це свідчить про його оптимізаційну ефективність. Unity забезпечує декілька ключових можливостей для проведення досліджень оптимізації. Розглянемо основні з них.

Інструмент «LOD Group System» дозволяє автоматичне перемикання між рівнями деталізації залежно від відстані до камери, що дозволяє зменшувати навантаження на GPU (рис. 3.1).

Інструмент «Profiler» призначений для аналізу продуктивності, який відображає споживання CPU, GPU, пам’яті, кількість draw calls, frame time та інші показники у реальному часі (рис. 3.2).

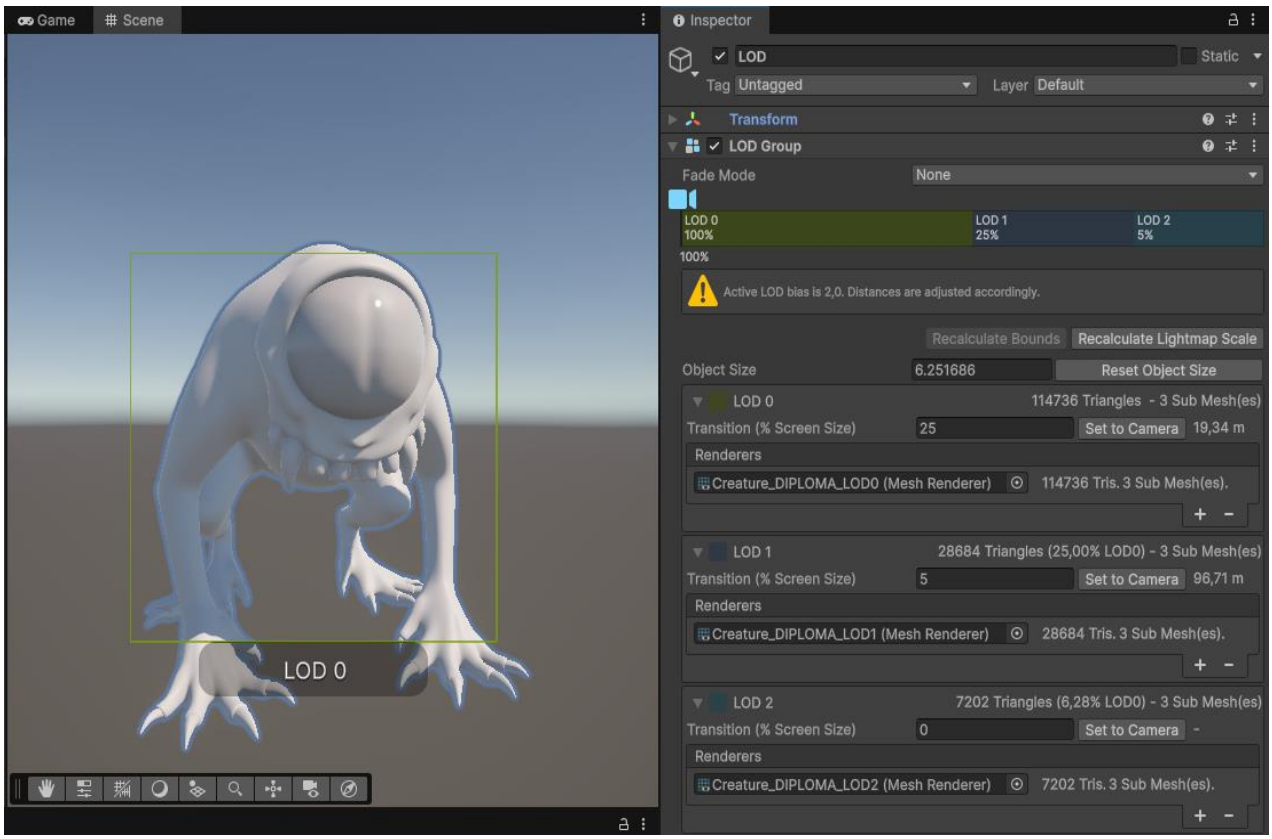



Рисунок 3.1 – LOD Group System у Unity

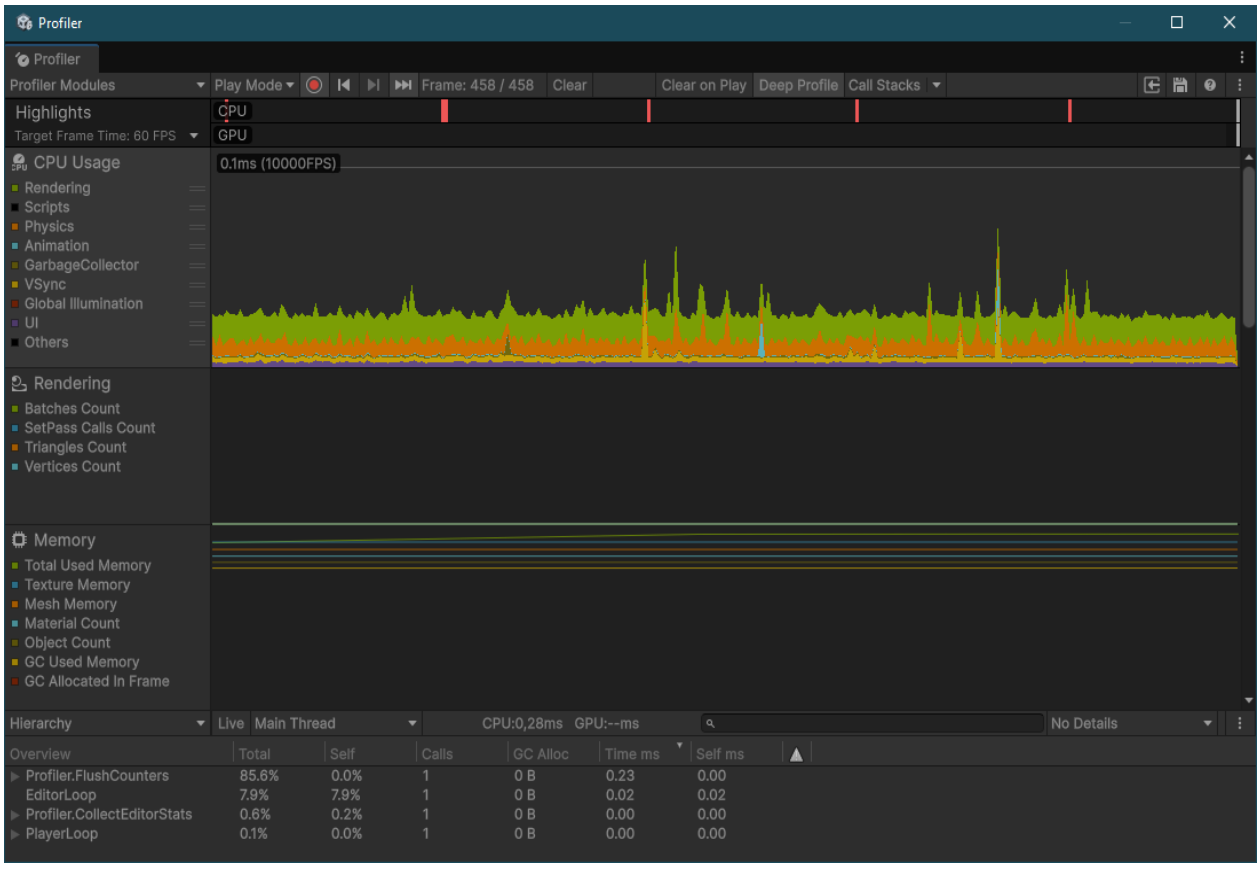



Рисунок 3.2 – Unity Profiler

Інструмент «Texture Import Settings» дає можливість керування розміром, форматом стиснення та MIP-рівнями текстур. Це дозволяє досліджувати вплив якості текстур на продуктивність сцени.

Інструмент «Frame Debugger» призначений для відстеження рендеринг-процесу, який дозволяє визначати, які саме об’єкти чи шейдери створюють надмірне навантаження (рис. 3.3).

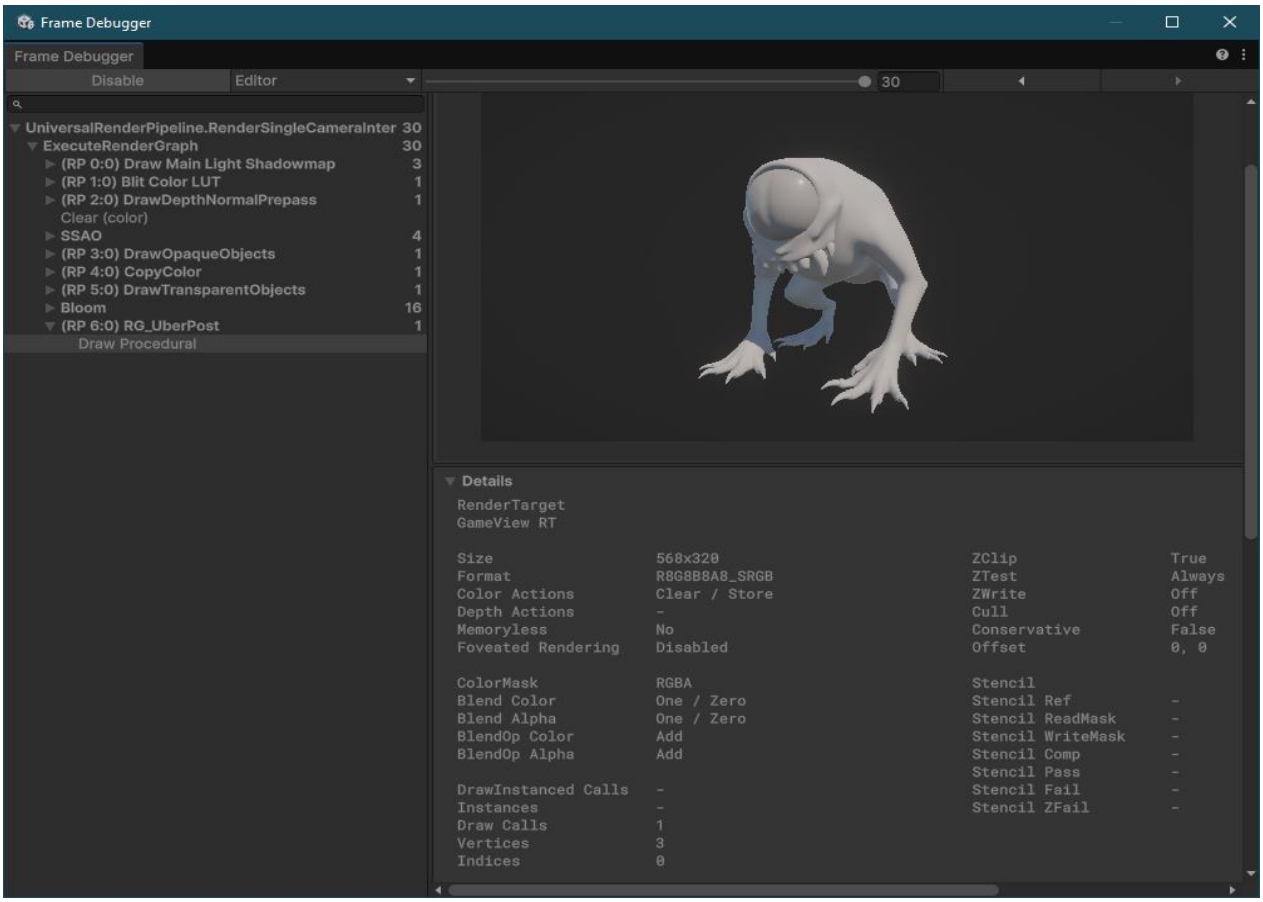



Рисунок 3.3 – Unity Frame Debugger

Інструмент «Statistics Panel» – це базова панель статистики, що показує FPS, кількість полігонів, вершин, draw calls та використання пам’яті на сцені (рис. 3.4).

Інструменти «Occlusion Culling» та «Light Baking» – це допоміжні методи, які дозволяють дослідити вплив попереднього освітлення та приховування об’єктів поза камерою на загальну продуктивність.

Для обґрунтованого вибору рушія доцільно порівняти Unity з іншими

поширеними середовищами, такими як Unreal Engine та Blender (табл. 3.1).

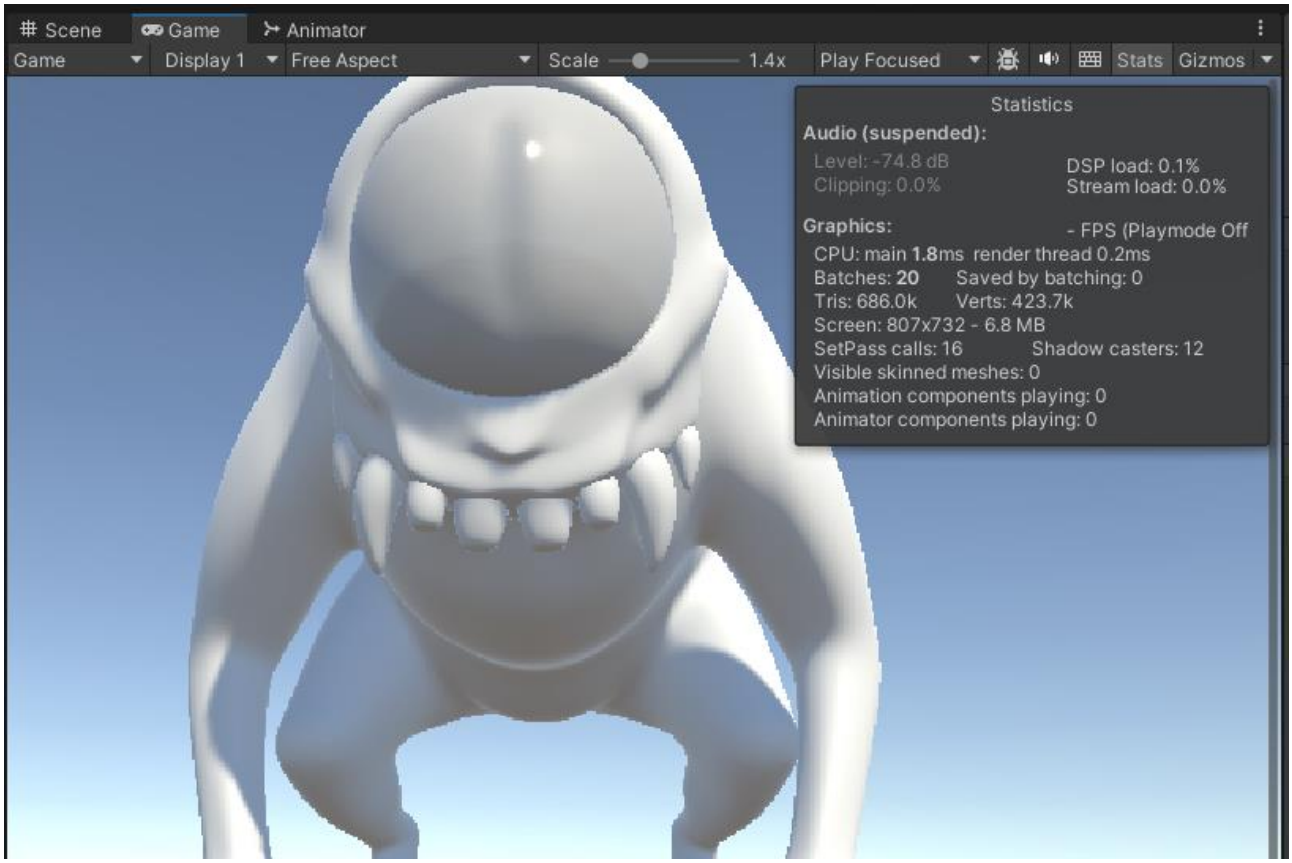



Рисунок 3.4 – Unity Statistics Panel

Таблиця 3.1 – Порівняльна характеристика рушіїв

| **Критерій** | **Unity** | **Unreal Engine** | **Blender** |
|---|---|---|---|
| **1** | **2** | **3** | **4** |
| Тип рушія | Ігровий рушій | Ігровий рушій | Пакет тривимірного моделювання та рендерингу |
| Інструменти профілювання | Profiler, Frame Debugger, Stats | Unreal Profiler, GPU Visualizer | System Console, Render Stats |
| Підтримка LOD | Так (LOD Group, автоматичне налаштування) | Так (LOD System, HLOD) | Обмежена (через модифікатори) |

Продовження таблиці 3.1

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| Робота з текстурними атласами | Вбудована підтримка | Через Material Instances | Потрібні плагіни |
| Потокове завантаження ресурсів | Addressables, Asset Bundles | Streaming Levels, Nanite | Відсутнє |
| Простота реалізації експериментів | Висока | Середня | Висока, але без вимірювання FPS |
| Візуалізація у реальному часі | Так | Так | Обмежено |
| Основна мова скриптів | C# | C++/Blueprints | Python |

Таблиця показує, що Unity оптимально поєднує гнучкість налаштувань, зручність профілювання та швидкість підготовки експериментів. На відміну від Unreal Engine, який має більший поріг входу та складнішу архітектуру управління сценами. Blender, у свою чергу, не пропонує достатніх інструментів для вимірювання реального часу рендерингу, незважаючи на потужність у моделюванні.

Для дослідження оптимізації використовуються наступні основні критерії:

– частота кадрів (FPS, Frames per Second) визначає плавність відображення сцени;

– кількість викликів рендерингу об’єктів (draw calls), адже зменшення кількості draw calls є одним із ключових ефектів оптимізації через LOD, текстурні атласи та батчінг;

– обсяг пам’яті, який використовує сцена (memory usage: RAM, VRAM), адже для Unity важливими параметрами є «Texture Memory», «Mesh Memory»,

«Animation Memory»;

– час завантаження сцени або асетів (scene load time) прямо залежить від кількості полігонів та обсягу текстур;

– загальний розмір збірки проєкту (project build size), який можна зменшити використовуючи оптимізовані текстури та моделі.

Для оцінки впливу оптимізації на продуктивність можна використати коефіцієнт приросту продуктивності.

У Unity для детального аналізу використовуються такі компоненти:

– компонент «Profiler window» є основним інструментом вимірювання FPS, CPU, GPU, ренерингу, пам’яті, аудіо, відео, фізики, тощо;

– компонент «Frame debugger» дозволяє поетапно відстежити рендеринг-процес кожного кадру (draw mesh, set pass, clear), що дає можливість виявляти надлишкові шейдерні проходи або дублювання об’єктів;

– компонент «Stats panel» зручна для попередніх вимірювань під час роботи у scene view;

– компонент «GPU profiler» (RenderDoc інтеграція), що використовується для детального аналізу шейдерів та GPU-навантаження.

Для фіксації результатів дослідження зручно використовувати профільні графіки з Unity Profiler та експорти даних у форматі CSV, що дозволяє обробляти їх статистично (у Excel, MATLAB або Python).

З переваг використання Unity для дослідницьких цілей можна виділити наступні пункти:

– швидкість і наочність експериментів, бо зміни у моделях, текстурах чи LOD відразу відображаються у результатах профайлера;

– вбудовані інструменти аналізу, які не потребують зовнішніх додатків;

– масштабованість, бо результати можна перевірити на мобільних, VR- чи десктопних платформах;

– підтримка автоматизації, так як Unity дозволяє створювати тести продуктивності через C#-скрипти;

– відкрита документація та спільнота, яка має широкий набір кейсів,

туторіалів і бібліотек для оптимізації [45].

Отже, завдяки поєднанню інструментів вимірювання продуктивності, зручній інтеграції моделей різних рівнів деталізації та гнучким можливостям управління ресурсами Unity є ідеальним середовищем для досліджень методів оптимізації тривимірних моделей. Unity надає надійні та репрезентативні результати, придатні як для академічного аналізу, так і для практичного використання при створенні оптимізованого тривимірного контенту.

## 3.2 Методика проведення експерименту

Для аналізу ефективності оптимізаційних підходів були використані три типи моделей, що відрізняються за рівнем полігональності: високополігональна модель, модель середньої деталізації та модель низької деталізації.

Високополігональна модель (high-poly) використовується як базова, необроблена версія. Вона демонструє найвищу якість візуалізації, проте має надмірне навантаження на GPU, має 120000 полігонів та призначена для тестування продуктивності без оптимізації.

Модель середньої деталізації (mid-poly) зазвичай отримується шляхом ретопології та декімації, що дозволяє зменшити полігональність при збереженні загальних форм і силуетів. Ця модель має 40000 полігонів та призначена для тестування після базової оптимізації.

Модель низької деталізації (low-poly) – це модель, оптимізована під реальний час, з використанням нормалей і карт висот для компенсації втраченої геометрії. Ця модель має 10000 полігонів та використовується як фінальна версія для оцінки оптимізації LOD та текстурних карт.

Для збереження візуальної відповідності було виконано запікання карт нормалей, карт висот і карти «Ambient Occlusion» із високополігональної на низькополігональну модель (рис. 3.5).

Рисунок 3.5 – Низькополігональна модель із запеченими картами

У рамках кваліфікаційної роботи досліджувалось п’ять основних методів, що розглядались у другому розділі:

– зменшення полігональності (ретопологія, декімація) означає зменшення кількості полігонів без істотної втрати форми;

– рівні деталізації (LOD) означає автоматичне перемикання між моделями різної складності залежно від відстані;

– оптимізація текстур (атласи, PBR, UV-розгортки) означає об’єднання текстур у єдиний атлас для зменшення draw calls;

– використання нормалей, карт висот і процедурних карт означає

відтворення деталей поверхні без збільшення геометрії;

– стиснення ресурсів і потокове завантаження (Addressables) означає зменшення розміру збірки та часу завантаження сцени.

Для забезпечення коректності результатів усі тести виконувалися на одній і тій самій апаратній платформі (табл. 3.2).

Таблиця 3.2 – Характеристики системи тестування

| **Компонент** | **Характеристика** |
|---|---|
| Процесор (CPU) | AMD Ryzen 9 7950X 4.50 GHz |
| Відеокарта (GPU) | NVIDIA GeForce RTX 4070 Ti SUPER 16 GB |
| Оперативна пам’ять (RAM) | 64 GB |
| Накопичувач | Samsung SSD 980 PRO 2 TB |
| Операційна система | Windows 10 x64 |
| Версія Unity | Unity 2022.3.9f1 (URP) |

Для контролю стабільності FPS використовувався вбудований профайлер Unity та програма MSI Afterburner для перевірки навантаження на GPU.

Оцінювання ефективності оптимізації проводилось за такими критеріями:

– базовий показник продуктивності (Frames per Second, FPS);

– кількість звернень до GPU за кадр (draw calls, DC);

– споживання відеопам’яті (VRAM usage, V);

– використання оперативної пам’яті (RAM usage, R);

– час завантаження сцени (scene load time, $T_l$);

– розмір фінальної збірки (build size, S).

Для кожного показника розраховувався відносний коефіцієнт покращення.

$$К = \frac{P_{до} - P_{після}}{P_{до}} \times 100\%, \quad (3.3)$$

де $P_{до}$ – значення показника до оптимізації,

$P_{після}$ – значення показника після оптимізації.

Загальний інтегральний показник ефективності оптимізації вимірюється наступною формулою:

$$K_{\text{заг}} = \frac{1}{n}\sum_{n=1}^{n} K_i, \tag{3.4}$$

де n – кількість вимірюваних критеріїв.

Методика реалізації експерименту включала такі етапи:

Етап 1. Імпорт моделей до Unity у форматах «.fbx» та «.obj».

Етап 2. Налаштування сцен з рівномірним освітленням та стандартними матеріалами.

Етап 3. Проведення початкового тесту без оптимізації.

Етап 4. Послідовне застосування методів оптимізації (ретопологія, LOD, атласи, нормалі, Addressables).

Етап 5. Фіксація результатів профілювання (FPS, Draw Calls, пам’ять, час завантаження).

Етап 6. Аналіз результатів у таблицях і графіках.

Етап 7. Порівняння результатів між методами.

Під час тестування важливим фактором є контроль умов відтворення. Для уникнення спотворень результатів вимикаються всі додаткові камери, постобробка, тіні та режим «VSync» для уникнення ліміту FPS. Також використовується «URP» для уніфікованого рендерингу. Тестування проводиться при однаковій роздільній здатності екрана 3440×2160.

Застосування описаної методики дозволяє:

– забезпечити достовірність результатів через повторюваність тестів;

– кількісно оцінити ефективність кожного методу окремо;

– виявити кореляцію між зменшенням полігональності, розміром ресурсів і FPS;

– сформувати рекомендації для подальшої оптимізації моделей у реальних проєктах.

Методика проведення експерименту передбачає поетапне тестування моделей різного рівня складності у стандартизованих умовах середовища Unity. Вибір єдиної апаратної платформи, контроль параметрів освітлення та використання системного профайлера забезпечують точність і відтворюваність результатів.

## 3.3 Аналіз результатів дослідження з оптимізації тривимірних моделей

Після проведення серії тестів у середовищі Unity було отримано комплексні результати, що відображають вплив різних методів оптимізації на продуктивність візуалізації тривимірних сцен. У ході експериментів порівнювалися показники FPS, використання пам’яті, час завантаження сцен і розмір збірки проєкту для різних рівнів деталізації та ступенів оптимізації.

### 3.3.1 Аналіз результатів оптимізації методом зменшення полігональності

Для аналізу ефективності використання методу зменшення полігональності було сформовано та використано три типи моделей: високополігональна модель на 120000 полігонів, середньополігональна модель на 40000 полігонів та низькополігонально модель на 10000 полігонів. Під час тестування в ігровому двигуні Unity (версія 2022.3 LTS) використовувався однаковий набір текстур та однаковий матеріал. На рисунку 3.6 наведено результати тесту високополігональної моделі, на рисунку 3.7 наведено результати тесту моделі середнього полігонажу, на рисунку 3.8 наведено результати тесту низькополігональної моделі. Результати тестів з оптимізації методом зменшення полігональності зведено у таблиці 3.3.


Рисунок 3.6 – Тест високополігональної моделі


Рисунок 3.7 – Тест середньополігональної моделі


Рисунок 3.8 – Тест низькополігональної моделі

Таблиця 3.3 – Результати тесту з оптимізації тривимірних моделей методом зменшення полігональності

| **Параметр** | **High-poly** | **Mid-poly** | **Low-poly** |
|---|---|---|---|
| Полігони | 120000 | 40000 | 10000 |
| FPS (середній) | 48 | 83 | 122 |
| Пам’ять GPU (MB) | 512 | 326 | 218 |
| Час завантаження (с) | 4,8 | 3,1 | 2,4 |
| Розмір збірки (MB) | 280 | 190 | 160 |

Як видно з рисунку 3.9, зменшення кількості полігонів майже лінійно підвищує FPS до певного порогу.

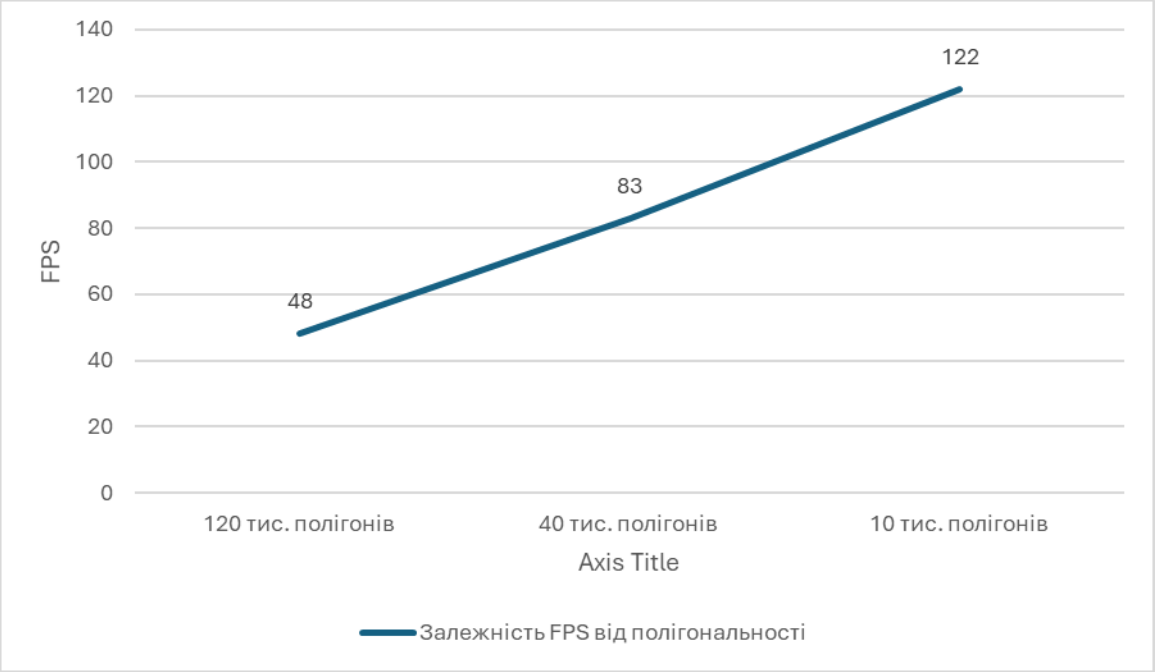



Рисунок 3.9 – Залежність FPS від полігональності

Відношення швидкодії до кількості полігонів має форму логарифмічної залежності.

### 3.3.2 Аналіз результатів оптимізації методом LOD

Наступний етап передбачає перевірку ефективності оптимізації тривимірних моделей за допомогою LOD системи. Для моделі створювалось три рівні деталізації: LOD0 – оригінальна модель, LOD1 – середня деталізація (40% полігонів), LOD2 – низька деталізація (10% полігонів). Відстань від камери, за якої модель перемикається на LOD1 – 20 м, на LOD2 – 50 м. Результати тестів наведено у таблиці 3.4. LOD-система показала особливо високу ефективність у сценах з великою кількістю об’єктів, де одночасно відображається багато екземплярів (рис. 3.10).

Таблиця 3.4 – Результати тесту з оптимізації тривимірних моделей методом LOD

| Кількість об’єктів у сцені | FPS без LOD | FPS з LOD | Різниця (%) |
|---|---|---|---|
| 50 | 71 | 104 | +46% |
| 100 | 52 | 89 | +71% |
| 250 | 33 | 72 | +118% |
| 500 | 22 | 61 | +177% |

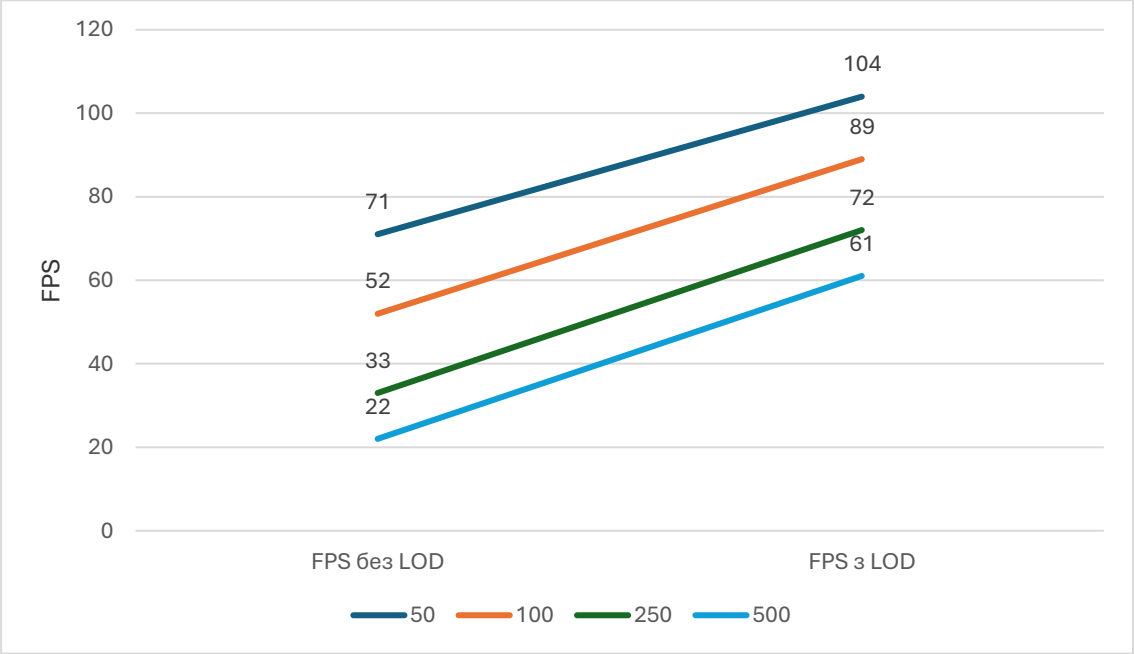



Рисунок 3.10 – Графік приросту FPS при використанні LOD-системи у сценах різної щільності

### 3.3.3 Аналіз результатів оптимізації текстур та матеріалів

Для порівняння впливу тектсурної оптимізації було створено три набори тексутр: 4К тексутри (оригінальні PBR матеріали), 2К текстури та тексутрні атласи 1К із комбінованими картами (diffuse, roughness, metallic). Результати тесту 4К текстури наведено на рисунку 3.11, 2К – на рисунку 3.12, 1К – на рисунку 3.13. Результати тестування зведено у таблиці 3.5.

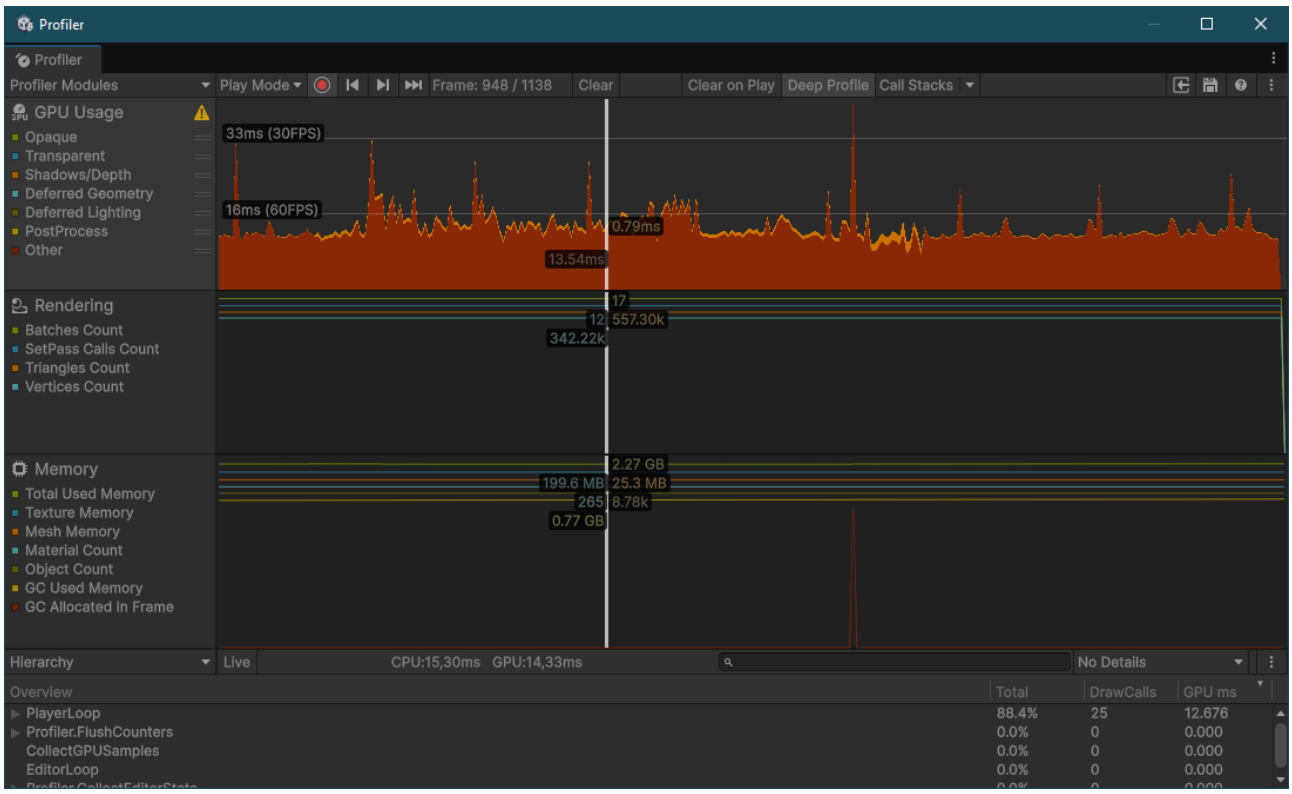



Рисунок 3.11 – Завантаження GPU під час тесту моделі з 4K текстурою

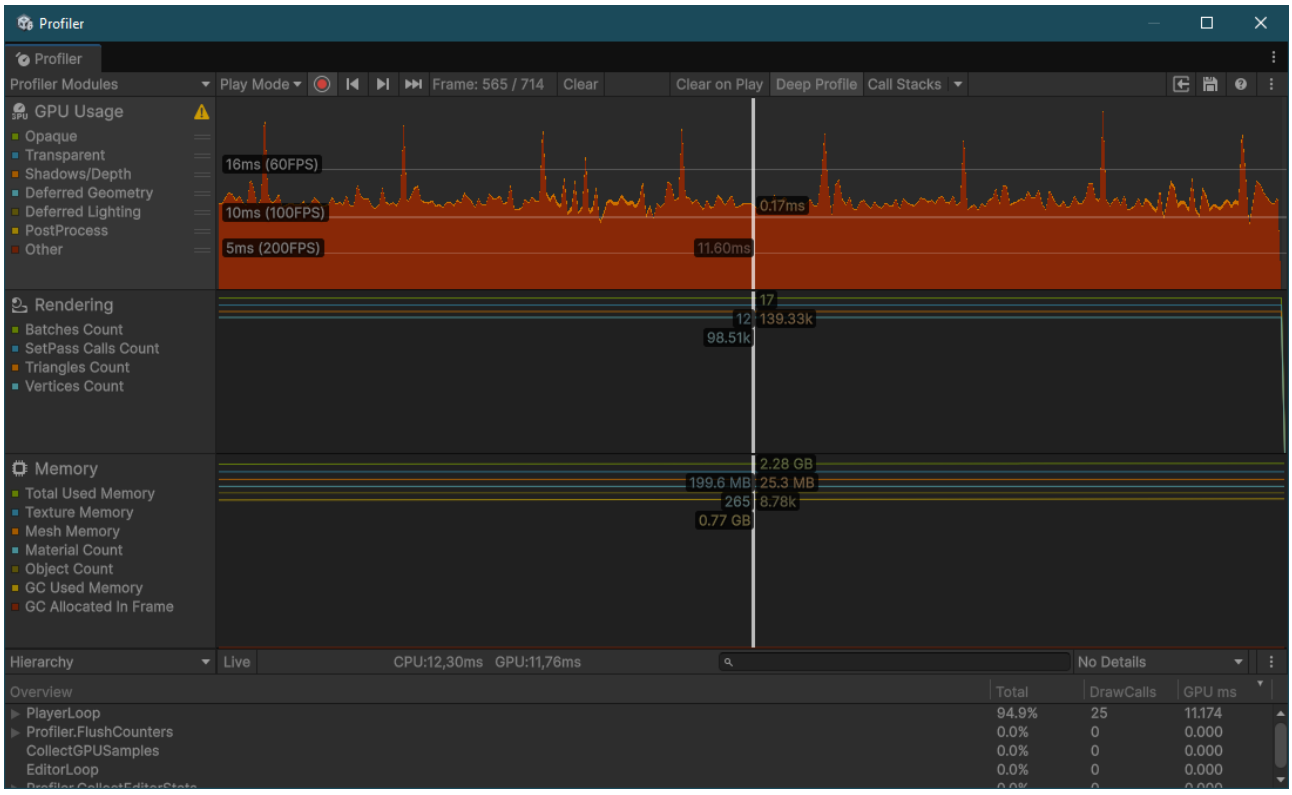



Рисунок 3.12 – Завантаження GPU під час тесту моделі з 2K текстурою

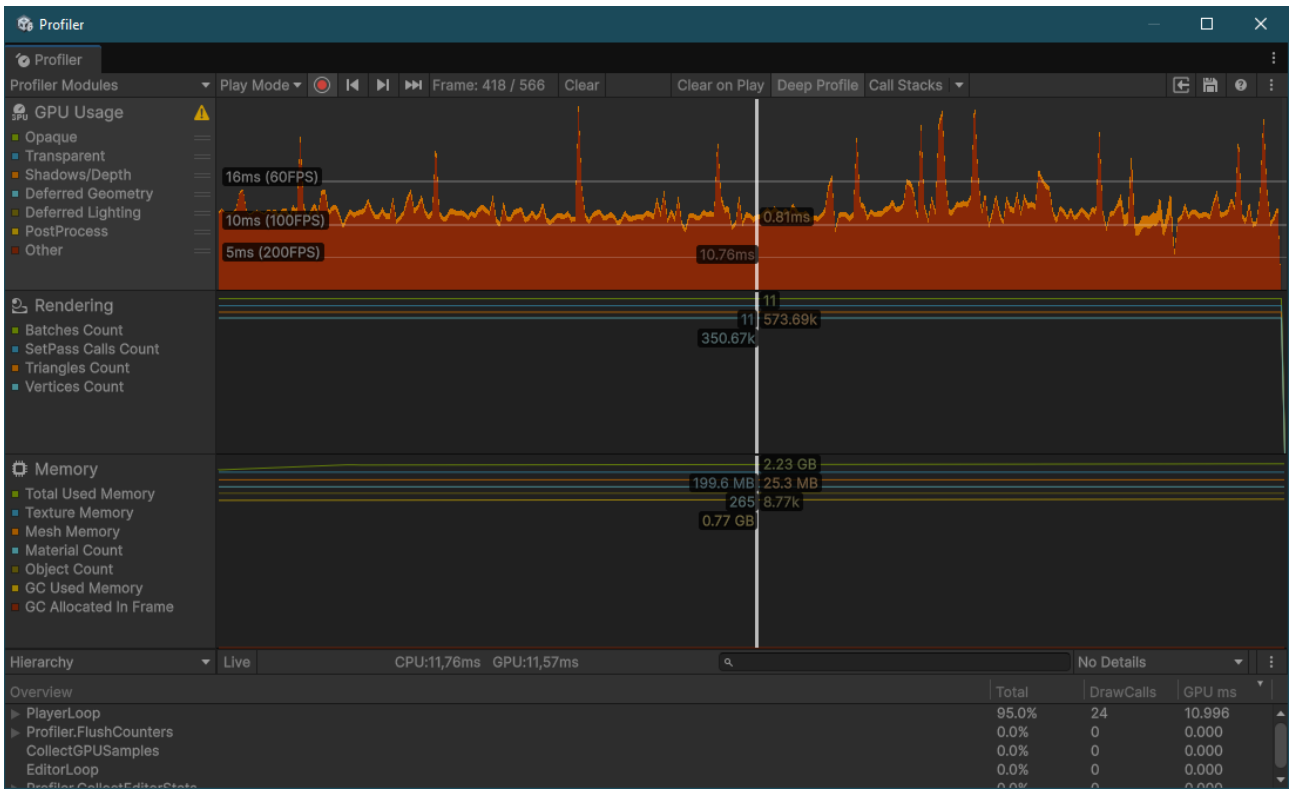



Рисунок 3.13 – Завантаження GPU під час тесту моделі з 1K текстурою

Таблиця 3.5 – Результати тесту з оптимізації текстур та матеріалів

| **Параметр** | **4K** | **2K** | **1K Атлас** |
|---|---|---|---|
| FPS | 57 | 79 | 88 |
| GPU пам’ять (MB) | 804 | 512 | 368 |
| Час завантаження (с) | 5,1 | 3,8 | 3,2 |
| Візуальні якість (%) | 100 | 95 | 90 |

Якість оцінювалася експертним методом. Візуальна якість моделей до та після оптимізації текстур наведена на рисунку 3.14.

Отримані результати показують, що зменшення роздільності текстур удвічі дає приблизно 30% приросту FPS, а використання атласів ще додаткові 10–12% (рис. 3.15).

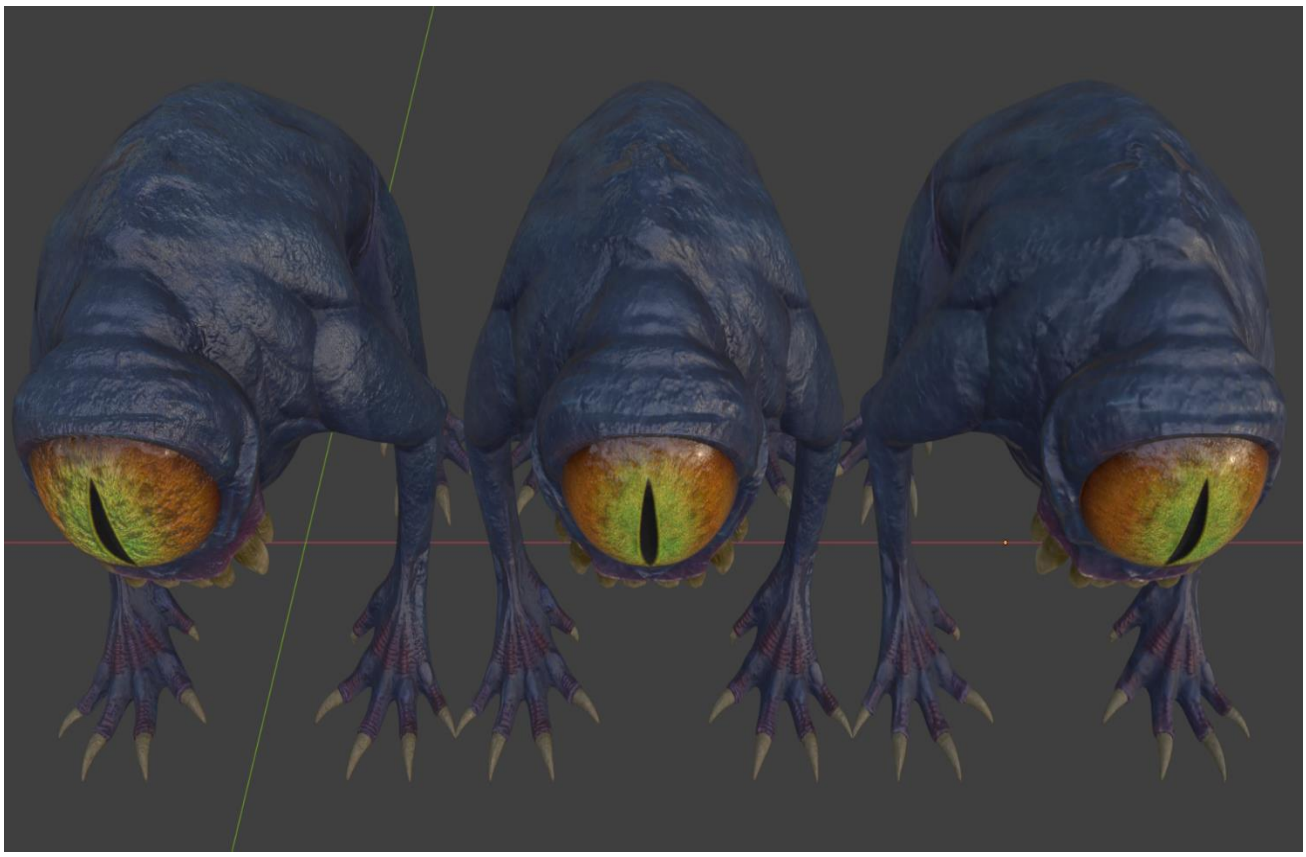


Рисунок 3.14 – Порівняння візуальної якості моделей при оптимізації текстур

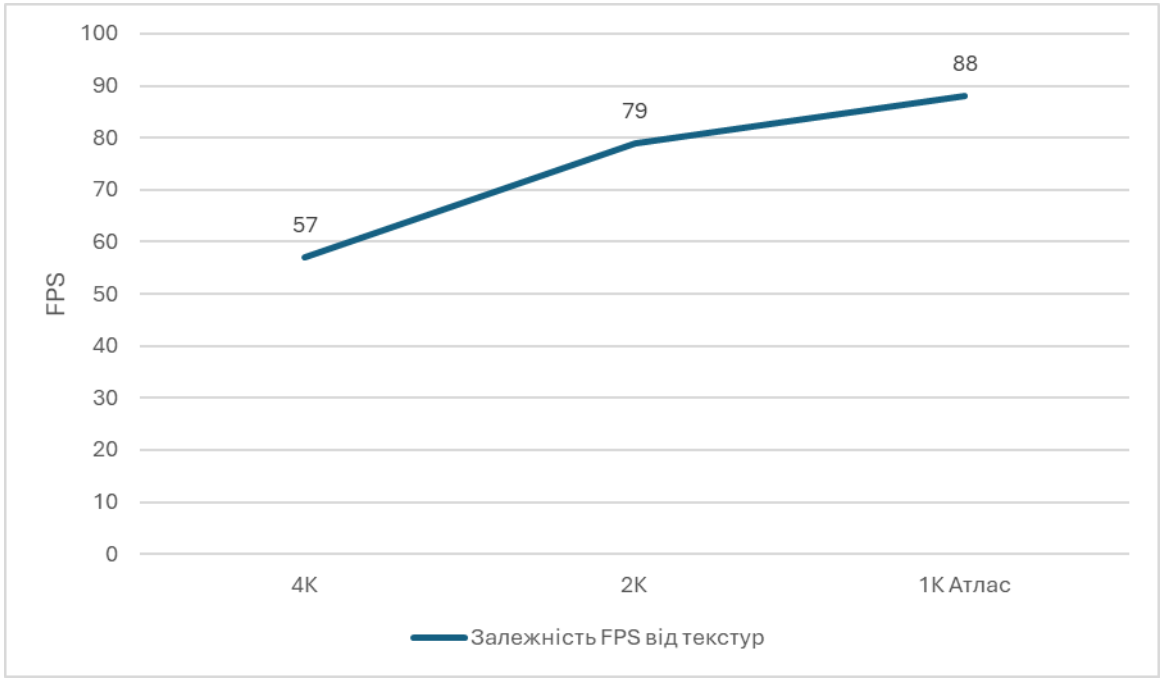



Рисунок 3.15 – Графік приросту FPS при оптимізації текстур

### 3.3.4 Аналіз результатів оптимізації моделей за допомогою карт нормалей та висот

Наступний експеримент перевіряв ефективність карти нормалей у порівнянні з геометричними деталями. Тестова модель мала дві версії:

– високополігональна геометрія, де деталі збережені на сітці (модель A);

– низькополігональна геометрія та карта нормалей, запечена з моделі A (модель B).

Результати наведено у таблиці 3.6.

Таблиця 3.6 – Результати тесту з оптимізації моделей за допомогою карт нормалей та висот

| **Модель** | **Полігони** | **FPS** | **GPU load (%)** | **Якість деталей (%)** |
|---|---|---|---|---|
| A | 120000 | 48 | 97 | 100 |
| B | 10000 | 116 | 62 | 93 |

Таким чином, використання нормалей зменшило навантаження на GPU на 36% при збереженні приблизно 93% візуальної якості.

### 3.3.5 Аналіз результатів оптимізації моделей за допомогою групування

Наступний експеримент перевіряв вплив кількості об’єктів на продуктивність. Було створено декілька сцен з low-poly об’єктів: на 100, 500, 1000 та 2500 екзамплярів. У таблиці 3.7 наведено результати використання методів static batching та GPU instancing.

На рисунку 3.16 наведено графік продуктивності при різних методах обробки об’єктів.

Таблиця 3.7 – Результати тесту з оптимізації моделей за допомогою групування

| **Кількість об’єктів** | **FPS без batching** | **FPS з static batching** | **FPS з GPU instancing** |
|---|---|---|---|
| 100 | 117 | 120 | 122 |
| 500 | 72 | 95 | 104 |
| 1000 | 49 | 74 | 91 |
| 2500 | 23 | 47 | 68 |

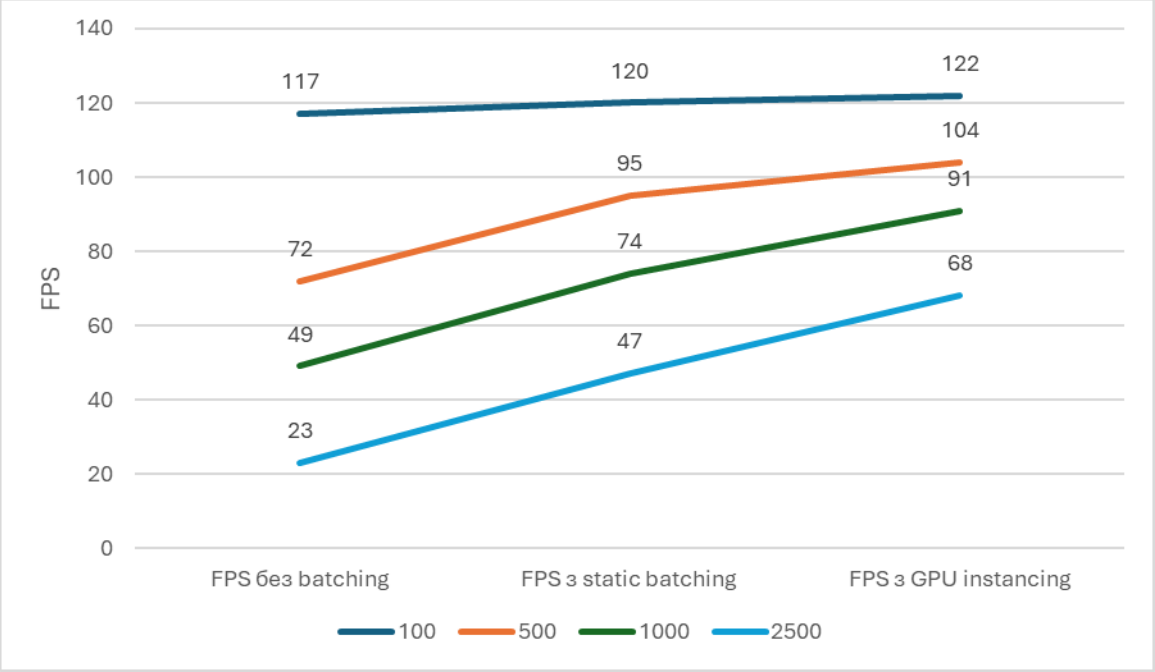



Рисунок 3.16 – Графік приросту FPS при різних методах обробки об’єктів

Використання GPU Instancing продемонструвало найкращі результати, адже середній приріст FPS становив 60% у порівнянні з рендерингом без оптимізації.

### 3.3.6 Сумарна оцінка ефективності методів

Метою даного етапу дослідження є порівняльний аналіз ефективності різних методів оптимізації тривимірних моделей у середовищі Unity. На основі зібраних експериментальних даних здійснено аналітичну оцінку впливу кожного методу на продуктивність системи, візуальну якість моделі та ресурсоспоживання. Такий підхід дозволяє визначити найоптимальніше поєднання технік оптимізації для конкретних типів проєктів, таких як ігри, VR-середовища, інтерактивні візуалізації та мобільні додатки.

Для зручності аналізу результати експериментів були зведені в інтегровану оцінювальну таблицю, у якій кожен метод порівнювався за чотирма головними критеріями ефективності: приріст FPS (продуктивність), зменшення використання пам’яті GPU, скорочення часу завантаження сцени та відносна візуальна якість порівняно з high-poly моделлю (табл. 3.8).

Таблиця 3.8 – Порівняльна характеристика впливу методів оптимізації

| **Метод оптимізації** | **Приріст FPS (%), F** | **Зниження GPU Memory (%), M** | **Скорочення часу завантаження (%), L** | **Візуальна якість (%), Q** | **Загальна ефективність** |
|---|---|---|---|---|---|
| Зменшення полігональності | +154 | -57 | -50 | 92 | 0,83 |
| LOD-система | +118 | -42 | -31 | 96 | 0,8 |
| Оптимізація текстур (атласи) | +54 | -65 | -37 | 90 | 0,74 |
| Карти нормалей і висот | +142 | -36 | -20 | 93 | 0,81 |
| GPU Instancing | +60 | -10 | -12 | 100 | 0,68 |

Загальну ефективність було розраховано за формулою:

$$E = 0{,}4F + 0{,}3M + 0{,}2L + 0{,}1Q. \quad (3.6)$$

Результати свідчать, що найвищий показник ефективності демонструють ретопологія (зменшення полігональності) та LOD-система. Ці методи забезпечують найбільший приріст FPS при мінімальній втраті якості. Карти нормалей показали подібний результат, що робить їх універсальним інструментом для оптимізації геометрії без помітного погіршення візуальної складової.

Щоб виявити закономірності між підвищенням FPS і втратою візуальної якості, було побудовано кореляційну залежність між цими параметрами. Для кожного методу визначено коефіцієнт кореляції Пірсона (табл. 3.9).

$$r = \frac{cov(x,y)}{\sigma_x \sigma_y}, \quad (3.7)$$

де $cov(x, y)$ – коваріація змінних X та Y,

$\sigma_x$ та $\sigma_y$ – стандартні відхилення цих змінних.

Таблиця 3.9 – Залежність між FPS і візуальною якістю після оптимізації

| **Метод** | **r (FPS ↔ Якість)** | **Характер зв'язку** |
|---|---|---|
| Зменшення полігональності | -0,76 | Сильний обернений |
| LOD-система | -0,45 | Помірний обернений |
| Оптимізація текстур | -0,51 | Помірний обернений |
| Карти нормалей | -0,29 | Слабкий обернений |
| GPU Instancing | -0,12 | Майже відсутній |

Отримані дані свідчать, що найменший негативний вплив на якість має GPU instancing, оскільки цей метод не змінює геометрію або текстури моделей, а

лише оптимізує процес їх відображення. Карти нормалей також показали низьку кореляцію, що означає, що візуальні втрати практично не сприймаються користувачем навіть при значному підвищенні FPS.

У межах тестування також оцінювалося середнє споживання енергії графічного процесора при різних методах оптимізації. Для цього використовувався вбудований інструмент NVIDIA FrameView у режимі моніторингу (табл. 3.10).

Таблиця 3.10 – Енергоспоживання та використання ресурсів GPU

| **Метод** | **Навантаження GPU (%)** | **Енергоспоживання (Вт)** | **Температура GPU (°C)** |
|---|---|---|---|
| Без оптимізації | 98 | 162 | 78 |
| Ретопологія | 65 | 109 | 67 |
| LOD-система | 71 | 116 | 69 |
| Оптимізація текстур | 73 | 118 | 70 |
| Карти нормалей | 62 | 104 | 66 |
| GPU Instancing | 68 | 110 | 68 |

Ретопологія та карти нормалей забезпечують найбільше зниження енергоспоживання (до 35 – 40%). Це має важливе значення не лише для настільних систем, але й для VR-та мобільних пристроїв, де автономність напряму залежить від ефективності рендерингу.

Тестування показало, що на час завантаження найбільше впливає розмір текстур і кількість draw call, тоді як геометрична складність має менший ефект (табл. 3.11).

Отже, текстурні атласи та GPU instancing демонструють найвищу ефективність у скороченні часу ініціалізації сцени.

На практиці оптимізаційні методи рідко застосовуються ізольовано, адже найкращих результатів досягають комбіновані стратегії. Було створено чотири конфігурації сцени, порівняння наведено у таблиці 3.12.

Таблиця 3.11 – Порівняння часу завантаження та кількості draw call

| **Метод** | **Час завантаження сцени (с)** | **Кількість draw call** | **Обсяг даних у пам'яті (MB)** |
|---|---|---|---|
| Без оптимізації | 5,1 | 412 | 812 |
| Оптимізація текстур (атласи) | 3,2 | 156 | 368 |
| LOD-система | 3,5 | 189 | 402 |
| GPU Instancing | 3,3 | 141 | 360 |
| Зменшення полігональності | 3,8 | 310 | 520 |

Таблиця 3.12 – Порівняння комплексних конфігурацій оптимізації

| **Конфігурація** | **Застосовані методи** | **FPS** | **GPU Memory (MB)** | **Якість (%)** | **Енергоспоживання (Вт)** |
|---|---|---|---|---|---|
| A | Без оптимізації | 48 | 804 | 100 | 162 |
| B | Ретопологія + LOD | 118 | 420 | 94 | 112 |
| C | Ретопологія + Атласи + Нормалі | 127 | 360 | 91 | 108 |
| D | Усі методи | 138 | 312 | 89 | 104 |

Комбінація всіх методів дозволила підвищити FPS утричі порівняно з базовою сценою, зменшити GPU-пам’ять на 61 % і скоротити споживання енергії майже на 36 %. Втрата візуальної якості при цьому не перевищує 11 %, що можна вважати прийнятним для більшості ігрових проєктів, особливо у VR- та мобільному середовищі.

## 3.4 Практичні рекомендації щодо оптимізації та узагальнення результатів

Результати проведених експериментів та порівняльного аналізу дозволили визначити низку практичних рекомендацій, які можуть бути використані у виробничих пайплайнах створення тривимірного контенту для різних платформ – від мобільних ігор до VR-додатків. Оптимізація тривимірних моделей є не лише технічним етапом, а й стратегічною складовою розробки, що визначає баланс між якістю візуалізації та продуктивністю системи.

Перш ніж обирати конкретний метод, важливо усвідомити загальні принципи оптимізації: збереження візуальної правдоподібності, комплексний підхід та орієнтація на кінцеву платформу.

Збереження візуальної правдоподібності, адже будь-яке зменшення полігональності чи роздільної здатності текстур повинно проводитись із урахуванням сприйняття користувачем. Людське око менш чутливе до дрібних деталей на далеких об’єктах, тому оптимізація повинна бути контекстно-залежною.

Комплексний підхід, бо найвищий результат досягається не одним методом, а поєднанням, наприклад, ретопологія + LOD + атласування + нормалі.

Орієнтація на кінцеву платформу, адже оптимізаційні параметри для AAA-ігор відрізняються від мобільних, VR- або AR-додатків. Для мобільних пристроїв пріоритетом є мінімальний розмір моделей та економія текстурної пам’яті, тоді як для VR – стабільний FPS при високій чіткості зображення.

Ефективність візуальної оптимізації залежить від контексту сцени: моделі, розташовані в полі зору користувача, повинні мати більший рівень деталізації, тоді як периферійні об’єкти можуть бути суттєво спрощені без втрати загальної якості візуального сприйняття.

Найбільш ефективним способом зменшення полігональності є ретопологія, коли топологія моделі перебудовується для кращої структурної організації полігонів. Вона дозволяє зменшити кількість полігонів у 4 – 8 разів без значних втрат якості, якщо додатково використовуються карти нормалей для відтворення мікродеталей поверхні.

Для великих сцен (наприклад, open-world ігор) доцільно проводити автоматичну пакетну декіміацію перед імпортом у рушій, що може зменшити загальну вагу проєкту на 30 – 50 %.

Система LOD дозволяє зменшувати складність моделей у реальному часі залежно від відстані до камери. У Unity реалізація LOD Group забезпечує автоматичне перемикання між 2 – 4 рівнями моделей, що істотно знижує навантаження на GPU при великій кількості об’єктів у кадрі. Застосування LOD-систем у середовищах типу Unity чи Unreal дозволяє збільшити середній FPS у середньостатистичній сцені на 25 – 40 % без помітних артефактів.

Текстури займають найбільший обсяг пам’яті у більшості тривимірних проєктів. Ефективне управління текстурними ресурсами може зменшити використання VRAM до 40 %. Поєднання кількох текстур в один атлас зменшує кількість draw calls, що суттєво впливає на продуктивність у реальному часі. Unity автоматично оптимізує такі матеріали під GPU-рендерінг. Використання фізично коректних матеріалів (PBR) дозволяє зменшити кількість карт за рахунок узагальнення властивостей поверхні. Наприклад, «metallic-roughness workflow» об’єднує дві карти у єдину структуру RGBA-каналів.

Використання normal map дозволяє замінити геометричні деталі на текстурні, що еквівалентно зменшенню кількості полігонів без втрати візуальної складності. При правильному освітленні така модель візуально не відрізняється від високополігонального оригіналу. Карти висот (height maps) використовуються

для паралакс-ефектів (Parallax Occlusion Mapping), що дозволяє створювати враження глибини поверхні без додавання геометрії. Використання процедурних карт замість унікальних текстур у сценах з великою кількістю об’єктів дозволяє скоротити розмір проєкту на 35 – 45 % без помітних втрат якості.

Для великих проєктів з великою кількістю моделей і текстур доцільно застосовувати потокове завантаження (streaming). Unity підтримує технологію Addressables і Asset Bundles, які дозволяють динамічно підвантажувати ресурси під час виконання. Також ефективним є стиснення геометрії (наприклад, Google Draco Compression), що зменшує вагу моделей у форматах GLTF/FBX на 50 – 80 %. Цей підхід особливо корисний для WebGL-ігор та VR-додатків, де кожен мегабайт ресурсу впливає на швидкість завантаження сцени.

Практичні рекомендації:

– для персонажів у реальному часі цільова щільність сітки повинна становити від 10 тис. до 40 тис. полігонів;

– для статичних об’єктів (декору, архітектури) цільова щільність сітки зазвичай повинна становити 5–15 тис. полігонів, але це залежить від відстані до камери, ігровий об’єкт, чи об’єкт для фону, наскільки багато уваги ігроки будуть звертати на цей об’єкт, тощо;

– використання автоматизованих інструментів, наприклад так як «Zremesher» у Zbrush або «QuadRemesher» у Maya/Blender, дозволяє швидко досягати оптимального результату при збереженні загальної форми, але все залежить від потреб конкретного проєкту;

– мінімум три рівні LOD: LOD0 (оригінал), LOD1 (50 % полігонів), LOD2 (20–30 %);

– для мобільних ігор рекомендується додатковий LOD3 із силуетною формою (5–10 % полігонів);

– використання «cross-fade transitions» у Unity дозволяє уникати видимого «стрибка» між рівнями деталізації.

У таблиці 3.13 наведено практичні рекомендації щодо роздільності текстур. У таблиці 3.14 наведено практичні рекомендаціх для різних типів проєктів.

Таблиця 3.13 – Рекомендовані роздільності та формати текстур

| **Тип об’єкта** | **Рекомендована роздільність** | **Формат** |
|---|---|---|
| Головний персонаж | 2048×2048 | PNG / TGA |
| Другорядний персонаж | 1024×1024 | JPG / DDS |
| Декор | 512×512 | JPG |
| Масивні сцени | 4096×4096 (атласи) | TIF / EXR |

Таблиця 3.14 – Рекомендовані методи оптимізації для різних типів проєктів

| **Тип проєкту** | **Рекомендовані методи** | **Основні цілі** |
|---|---|---|
| Мобільні ігри | LOD, ретопологія, атласи, стиснення | мінімальний розмір і стабільний FPS |
| VR-додатки | LOD, нормалі, потокове завантаження | стабільний FPS при високій чіткості |
| WebGL | декіміація, Draco, procedural maps | швидке завантаження |
| AAA-ігри | комплекс усіх методів + профайлінг | максимальна якість при балансі продуктивності |

Згідно з проведеним дослідженням, усі методи оптимізації можна поділити на три рівні:

– базові (геометричні), тобто ретопологія, декіміація;

– текстурно-візуальні, наприклад атласування, нормалі, PBR-матеріали;

– системно-ресурсні такі як LOD, потокове завантаження, стиснення.

Оптимальна стратегія полягає у поетапному застосуванні цих рівнів. Спершу спрощення геометрії, потім оптимізація матеріалів і, нарешті, адаптація під рушій. Такий підхід забезпечує найвищий приріст продуктивності без відчутних втрат якості візуалізації.

# ВИСНОВКИ

Таким чином, у кваліфікаційній роботі досліджено та проаналізовано методи оптимізації тривимірних моделей, що застосовуються у процесі створення ігор, і вирішено такі завдання:

– проведено аналіз літературних джерел і визначено основні підходи до оптимізації тривимірних моделей, що дозволило виявити сучасний стан проблеми, систематизувати методи зниження геометричної складності моделей, мінімізації використання ресурсів пам’яті та підвищення ефективності рендерингу у комп’ютерних іграх;

– досліджено сучасні програмні засоби для реалізації оптимізації тривимірних моделей, зокрема програмні пакети Blender, Maya, ZBrush, Substance 3D Painter та ігрові рушії Unity і Unreal Engine, що дало можливість визначити їхні інструменти для ретопології, керування рівнями деталізації (LOD), створення текстурних атласів і профілювання продуктивності;

– проведено порівняння ефективності різних методів оптимізації, серед яких зменшення полігональності моделі (ретопологія, декімація), використання рівнів деталізації (LOD), оптимізація текстур і матеріалів (UV-мапи, атласи, PBR), застосування карт нормалей і карт висот, стиснення та потокове завантаження ресурсів у рушії реального часу, це підтвердило, що комплексне використання кількох методів забезпечує найкраще співвідношення якості зображення та продуктивності;

– проведено тестування та оцінено результати за допомогою показників FPS, GPU-навантаження та якості відображення, у ході експериментів, виконаних у середовищі Unity, визначено, що застосування системи LOD дозволяє підвищити частоту кадрів у середньому на 40–60 %, тоді як оптимізація текстур за допомогою атласів скорочує кількість draw calls на 25–35 %, а використання нормальних карт дозволило зменшити полігональність без помітної втрати візуальної деталізації.

У результаті дослідження встановлено, що найефективнішими є

комбіновані підходи, у яких одночасно застосовуються геометрична, текстурна та ресурсна оптимізація. Це дозволяє досягати стабільної продуктивності навіть у складних ігрових сценах із великою кількістю динамічних об’єктів.

Наукова новизна роботи полягає у комплексному підході до оцінювання ефективності методів оптимізації тривимірних моделей у середовищі реального часу з урахуванням різних типів ігрових проєктів. Отримані результати дозволяють визначити оптимальні поєднання технологій для конкретних сценаріїв розроблення ігор.

Практична значущість дослідження полягає у формуванні рекомендацій для розробників ігор та художників з тривимірної графіки щодо вибору оптимізаційних стратегій під конкретні типи ігрових проєктів. Результати можуть бути використані під час створення ігор, VR-додатків, інтерактивних симуляцій та навчальних візуалізацій.

Перспективи подальших досліджень полягають у розробці адаптивних систем автоматичної оптимізації моделей, які здатні динамічно змінювати рівень деталізації залежно від характеристик пристрою користувача, а також у застосуванні методів машинного навчання для передбачення найефективніших параметрів оптимізації у реальному часі.

Результати роботи апробовано у вигляді 2 тез доповідей під час IV Міжнародої науково-практичної конференції «Technologies, theories and developments: modern scientific teaching» [46], II Міжнародної науково-практичної студентської конференції «ІТ-простір сьогодення: тенденції, інновації та перспективи розвитку» [47].

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kim, D., & Park, S. (2022). Texture streaming and memory optimization in modern graphics pipelines. *Journal of Computer Graphics & Visualization*, 18(3), 45–58.

2. Kobylin, O.; Putiatina, O. (2025). Some aspects of real-time image denoising influenced by shot noise and compound Poisson noise. CEUR Workshop Proceedings, volume = 3943 , 109-117

3. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень.

4. Wang, H., Zhao, Y., & Liu, Z. (2021). Optimization strategies for real-time 3D rendering in game engines. IEEE Access, 9, 125678–125690.

5. Kuzminska, O., Mazorchuk, M., Morze, N., & Kobylin, O. (2019, June). Digital learning environment of ukrainian universities: The main components to influence the competence of students and teachers. In International Conference on Information and Communication Technologies in Education, Research, and Industrial Applications (pp. 210-230). Springer, Cham.

6. Гончарова О.В. (2024) Дослідження методів семантичної кластеризації для аналізу новин // Радіоелектроніка та молодь у XXI столітті. Т.7: Конференція «Комп'ютерний зір, системний аналіз та математичне Моделювання»: матеріали 28-го Міжнар. молодіж. форуму, 16–18 квіт. 2024 р. / М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків: ХНУРЕ, 2024. – 320 с. – DOI: 10.30837/IYF.CVSAMM.2024. – с. 31-32.

7. Кобилін, О., Вечірська, І., Афанасьєв, А. (2024). Аналіз існуючих моделей глибинного навчання в задачах обробки природної мови. Information Technology: Computer Science, Software Engineering and Cyber Security, 3, 63–76, https://doi.org/10.32782/IT/2024-3-7

8. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., & Al-Dhaifallah, M. (2023). Statistical data analysis models for determining the relevance of structural image descriptions. IEEE Access, 11, 126938-126949.

9. Gorokhovatskyi, V. A., Rusakova, N., & Tvoroshenko, I. S. (2020). The application of image analysis methods and predicate logic in applied problems of magnetic monitoring. Telecommunications and Radio Engineering, 79(20).

10. Yakovleva, O., & Nikolaieva, K. (2020). Research of descriptor based image normalization and comparative analysis of SURF, SIFT, BRISK, ORB, KAZE, AKAZE descriptors. Advanced Information Systems, 4(4), 89-101.

11. Daradkeh, Y. I., & Tvoroshenko, I. (2020). Technologies for making reliable decisions on a variety of effective factors using fuzzy logic. International Journal of Advanced Computer Science and Applications, 11(5).

12. Daradkeh, Y. I., & Tvoroshenko, I. (2020). Technologies for making reliable decisions on a variety of effective factors using fuzzy logic. International Journal of Advanced Computer Science and Applications, 11(5).

13. Wang, H., Zhao, Y., & Liu, Z. (2021). Optimization strategies for real-time 3D rendering in game engines. IEEE Access, 9, 125678–125690.

14. Kobylin, O., Vyskrebentseva, S., & Petrova, R. (2019). Обробка даних, що містять пропуски в задачах кластеризації. Системи управління, навігації та зв'язку. Збірник наукових праць, 5(57).

15. Gorokhovatskyi V., Tvoroshenko I., Kobylin O., and Vlasenko N. (2023) Search for visual objects by request in the form of a cluster representation for the structural image description, Advances in Electrical and Electronic Engineering, 21(1), pp. 19-27.

16. Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Vlasenko, N. V. (2020). Using fuzzy clustering in structural methods of image classification. Telecommunications and Radio Engineering, 79(9).

17. Oleg, K., Sergii, M., & Mykhailo, S. (2017, October). Video Clustering via Multidimensional Time-Series Analysis. In Proceedings of the 9th International Conference on Information Management and Engineering (pp. 60-63). ACM.

18. Bodyanskiy, Y., Vynokurova, O., Kobylin, I., & Kobylin, O. (2016). Adaptive fuzzy clustering of short time series with unevenly distributed observations

in Data Stream Mining tasks. Information Technology and Management Science, 19(1), 23-28.

19. Nanite Virtualized Geometry in Unreal Engine | Unreal Engine 5.6 Documentation | Epic Developer Community

20. Kim, D., & Park, S. (2022). Texture streaming and memory optimization in modern graphics pipelines. Journal of Computer Graphics & Visualization, 18(3), 45–58.

21. Gorokhovatskyi , V., Chmutov , Y., Tvoroshenko , I., & Kobylin , O. (2025). Reducing computational costs by compressing the structural description in image classification methods. Advanced Information Systems, 9(1), 5–12. https://doi.org/10.20998/2522-9052.2025.1.01

22. Minecraft Wiki URL: https://uk.minecraft.wiki/ (дата звернення 19.10.2025).

23. Kobylin, O., & Lyashenko, V. (2016). Contrast Modification as a Tool to Study the Structure of Blood Components.

24. Lyashenko V., Kobylin O., Selevko O. (2020) Wavelet Analysis and Contrast Modification in the Study of Cell Structures Images. International Journal of Advanced Trends in Computer Science and Engineering. 9(4). – 4701-4706.

25. Gorokhovatskyi, V., & Tvoroshenko, I. (2023). Identification of visual objects by the search request. International scientific symposium «INTELLIGENT SOLUTIONS-S» (pp. 25-27).

26. Pharr, M., Humphreys, G. Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann, 2023.

27. Garland, M., Heckbert, P. Surface simplification using quadric error metrics. SIGGRAPH, 1997.

28. Simplygon AB. Simplygon Documentation URL: https://www.simplygon.com/docs (дата звернення 19.10.2025).

29. Unreal Engine Documentation. Level of Detail (LOD) Tools. Epic Games, 2024.

30. Möller, T., Haines, E. Real-Time Rendering. CRC Press, 2021.

31. Pharr, M., Humphreys, G. Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann, 2023.

32. Engel, W., et al. GPU Pro 8: Advanced Rendering Techniques. CRC Press, 2022.

33. Empirical performance tests in Unreal Engine 5.3 and Blender 3.6, 2025.

34. Guerrilla Games. (2022). Rendering Optimization in Horizon Forbidden West. GDC Presentation.

35. Lefebvre, S., & Hoppe, H. (2006). Perfect Spatial Hashing for Graphics Hardware. ACM Transactions on Graphics.

36. Wu, P., Chen, Y., & Wang, L. (2023). Performance Optimization Techniques for 3D Assets in Game Engines. Computers & Graphics, 113, 45–59.

37. Mikkelsen, M. (2008). Tangent Space Normal Mapping – Theory and Implementation.

38. Кобилін, О. А., & Путятіна, О. Є. (2024). Знешумлення зображень, зіпсованих дробовим шумом, у реальному часі. Системи обробки інформації, (1 (176), 46-51.

39. Kronrod, A. (2018). Quantized Encoding for 3D Geometry Compression. SIGGRAPH Asia.

40. Rossignac, J. (1999). EdgeBreaker: Connectivity Compression for Triangle Meshes. IEEE Transactions on Visualization and Computer Graphics.

41. Epic Games. (2023). Nanite Virtualized Geometry and Virtual Texturing in UE5. Unreal Engine Documentation.

42. Unity Technologies. (2022). Addressables and Asset Bundles: Efficient Memory Streaming. Unity Docs.

43. Google. (2020). Draco Compression and glTF 2.0 Integration. Google Developers Blog.

44. Unity Technologies. Unity Documentation: Profiler overview. URL: https://docs.unity3d.com/Manual/Profiler.html (дата звернення 19.10.2025).

45. Unity Manual. Level of Detail (LOD) system. URL: https://docs.unity3d.com/Manual/LevelOfDetail.html (дата звернення 19.10.2025).

46. Колісниченко М. В. (2025) Роль процедурних генеративних методів у створенні оптимізованих 3d-моделей для ігор. *IV International Scientific and Practical Conference «Technologies, theories and developments: modern scientific teaching», September 23-26, 2025, Valencia, Spain,* pp. 44-47.

47. Колісниченко М. В. (2025) Використання рівнів деталізації (lod) у мережевих іграх для зниження навантаження. *Міжнародна науково-практична студентська конференція «IT-простір сьогодення: тенденції, інновації та перспективи розвитку», 15 жовтня 2025, Харків, Україна,* pp. 385-387.