Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи підвищення продуктивності рендерингу в реальному часі для 3D-графіки

(тема)

Виконав:

студент II курсу, групи КСМм-23-1

Пилипенко О.Р.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Комп’ютерні системи та мережі
(повна назва освітньої програми)

Керівник: ас. Кравченко П.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Пилипенку Олексію Романовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Методи підвищення продуктивності рендерингу в реальному часі для 3D-графіки

затверджена наказом по університету від “ 22 ” листопада 2024 р. № 1237 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20 січня 2025 р.

3. Вхідні дані до роботи
Рівні деталізації
LOD
Оптимізація шейдерів

4. Перелік питань, що потрібно опрацювати у роботі
Теоретичні відомості та основні принципи рендерингу в реальному часі
Методи підвищення продуктивності та оптимізації рендерингу
Реалізація методів та аналіз їх ефективності
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційна презентація. Слайди – 14 штук.

6. Консультанти розділів роботи (заповнюється 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

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання та аналіз літератури	26.11.24 – 29.11.24	
2	Огляд існуючих рішень та методів	30.11.24 – 02.12.24	
3	Вибір програмних засобів	03.12.24 – 24.12.24	
4	Розробка тестової сцени	25.12.24 – 31.12.24	
5	Аналіз отриманих результатів	01.01.25 – 07.01.25	
6	Оформлення записки	08.01.25 – 14.01.25	
7	Представлення роботи в ЕК	15.01.25	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ас. Кравченко П.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 72 с., 22 рис., 1 дод., 16 джерел.

РЕНДЕРИНГ, ОТПИМІЗАЦІЯ, 3D-ГРАФІКА, ВІЗУАЛІЗАЦІЯ, LOD, ШЕЙДЕРИ.

Предметом дослідження є методи підвищення продуктивності рендерингу в реальному часі для 3D-графіки. Об’єктом дослідження є процес оптимізації рендерингу в реальному часі для тривимірних сцен.

Метою кваліфікаційної роботи є аналіз основних сучасних методів підвищення продуктивності рендерингу, які використовуються в ігровій та візуалізаційній індустрії. Серед розглянутих підходів були виділені та детально вивчені методи рівнів деталізації, видалення невидимих об'єктів, оптимізація шейдерів, а також застосування технологій трасування променів у реальному часі. Результатом роботи стало впровадження рекомендацій для оптимізації рендерингу в реальному часі, які можуть бути використані для проєктування інтерактивних 3D-додатків, зокрема ігор, архітектурних візуалізацій та симуляторів. Розроблено тестову сцену, яка продемонструвала ефективність застосованих методів оптимізації, та виконано її порівняльний аналіз до і після впровадження цих підходів.

ABSTRACT

Master’s thesis: 72 pages, 22 figures, 1 appendices, 16 sources.

RENDERING, OPTIMIZATION, 3D-GRAPHICS, VISUALIZATION, LOD, SHADERS.

The subject of the research is methods for improving real-time rendering performance for 3D graphics. The object of the research is the process of optimizing real-time rendering for three-dimensional scenes.

The goal of this qualification work is to analyze the main modern methods for improving rendering performance used in the gaming and visualization industries. Among the approaches considered, the methods of level of detail, occlusion culling, shader optimization, and the application of real-time ray tracing technologies were identified and studied in detail. The result of the work is the implementation of recommendations for real-time rendering optimization, which can be used in the design of interactive 3D applications, including games, architectural visualizations, and simulators. A test scene was developed to demonstrate the effectiveness of the applied optimization methods, and a comparative analysis of its performance before and after the implementation of these approaches was conducted.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ОСНОВНІ ПРИНЦИПИ РЕНДЕРИНГУ В РЕАЛЬНОМУ ЧАСІ 10
1.1 Огляд основних понять рендерингу 10
1.2 Проблеми продуктивності при рендерингу в реальному часі 13
1.3 Огляд сучасних технологій та методів рендерингу 15
1.3.1 Рейкастинг 17
1.3.2 Растеризація 18
1.3.3 Трасування променів 19
1.4 Актуальні проблеми у 3d-графіці 21
1.5 Постановка задач 22
2 МЕТОДИ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ТА ОПТИМІЗАЦІЇ РЕНДЕРИНГУ 24
2.1 Метод оптимізації геометрії та використання LOD 24
2.2 Методи видалення невидимих об'єктів 28
2.2.1 Frustum Culling 30
2.2.2 Occlusion Culling 34
2.2.3 Backface Culling 37
2.3 Метод покращення продуктивності шейдерів 40
2.4 Метод трасування променів у реальному часі 44
3 РЕАЛІЗАЦІЯ МЕТОДІВ ТА АНАЛІЗ ЇХ ЕФЕКТИВНОСТІ 51
3.1 Опис середовища та інструментів для реалізації 51
3.2 Впровадження обраних методів оптимізації 53
3.2.1 LOD 53
3.2.2 Видалення невидимих об’єктів 55

3.2.3 Покращення продуктивності шейдерів 57
3.2.4 Трасування променів у реальному часі 59
ВИСНОВКИ..... 61
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 63
ДОДАТОК А Графічний матеріал кваліфікаційної роботи..... 65

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – інтерфейс прикладного програмування (англ., Application Programming Interface)

AR – доповнена реальність (англ., Augmented Reality)

ASTC – формат стиснення текстур (англ., Adaptive Scalable Texture Compression)

BVH – ієрархія обмежувальних об'ємів (англ., Bounding Volume Hierarchy)

CPU – центральний процесор (англ., Central Processing Unit)

DXT – стиснення текстур у DirectX (англ., DirectX Texture Compression)

FPS – кількість кадрів за секунду (англ., Frames Per Second)

GPU – графічний процесор (англ., Graphics Processing Unit)

LOD – рівень деталізації (англ., Level of Detail)

RTX – технологія трасування променів (англ., Ray Tracing eXtreme)

VR – віртуальна реальність (англ., Virtual Reality)

VRAM – відеопам’ять (англ., Video Random Access Memory)

ВСТУП

У сучасному світі вимоги до якості та швидкості обробки графічної інформації постійно зростають. Зокрема, у сфері тривимірної графіки важливо не лише забезпечити високий рівень деталізації, але й досягти максимальної продуктивності рендерингу в реальному часі. Однак збільшення обсягу графічних даних часто ускладнює ефективне використання ресурсів системи та знижує швидкість обробки. Тому оптимізація процесу рендерингу є важливою задачею, яка дозволяє створювати якісне графічне наповнення з мінімальними витратами обчислювальних ресурсів.

Ефективні методи рендерингу дозволяють прискорити процес відображення складних графічних сцен, що є особливо актуальним для ігор, віртуальної реальності та інших інтерактивних застосувань. Такі методи допомагають виявляти оптимальні способи обробки даних, зменшувати затримки у відтворенні зображень та підтримувати стабільну продуктивність у динамічних середовищах.

Подальший розвиток методів підвищення продуктивності рендерингу є надзвичайно важливим, оскільки це сприятиме підвищенню швидкості обробки даних, зниженню енергоспоживання та забезпеченню високої якості графіки навіть на пристроях з обмеженими ресурсами.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ОСНОВНІ ПРИНЦИПИ РЕНДЕРИНГУ В РЕАЛЬНОМУ ЧАСІ

1.1 Огляд основних понять рендерингу

Рендеринг у реальному часі – це процес створення зображень із тривимірних сцен з максимально можливою швидкістю. Цей процес використовується в різних областях, таких як відеоігри, симуляції, віртуальна та доповнена реальність. Основною метою рендерингу в реальному часі є забезпечення високої продуктивності для досягнення плавного й реалістичного візуального досвіду користувача, часто з обмеженим бюджетом ресурсів, таких як потужність графічного процесора та пам'ять [1].

Основними компонентами рендерингу є геометрія, світло, тіні, текстури та матеріали [2]. Геометрія визначає форму об'єктів у сцені і складається з полігонів. Світло визначає, як об'єкти освітлюються, а тіні – як об'єкти блокують світло, створюючи темні ділянки. Текстури використовуються для деталізації поверхонь, а матеріали задають, як ці текстури взаємодіють зі світлом [2].

Ключовими аспектами рендерингу в реальному часі є швидкість і якість. Частота кадрів – одна з основних метрик продуктивності, яка визначає, скільки зображень може бути згенеровано за секунду. Більшість сучасних додатків, особливо відеоігор, прагнуть до частоти 60 кадрів на секунду, щоб забезпечити плавний ігровий досвід.

Першим кроком у рендерингу є обробка геометрії. На цьому етапі визначаються форми та положення об’єктів у тривимірному просторі. Відбувається трансформація об’єктів із локальної системи координат, де описана їхня форма, у глобальну систему координат сцени, після чого виконується проекція для відображення об’єктів на екрані [3]. Проекція

змінює 3D-координати на 2D-координати, дозволяючи відобразити тривимірну сцену на плоскій поверхні дисплея.

Далі йде обчислення світла та затінення. На цьому етапі визначається, як освітлення впливає на зовнішній вигляд об'єктів сцени. Існує декілька методів обробки світла і тіней, які дозволяють досягти реалістичних зображень. Найбільш поширені методи затінення – це затінення Гуро та затінення Фонга. Затінення Гуро використовує інтерполяцію кольорів між вершинами для більш плавних переходів, тоді як затінення Фонга дає можливість точно враховувати відображення світла, створюючи реалістичніші відблиски на об'єктах.

Етап текстурування та використання матеріалів додають деталізацію та забезпечують реалістичну взаємодію об'єктів із світлом. Текстури – це двовимірні зображення, які наносяться на поверхні об'єктів для відтворення складних деталей без збільшення кількості полігонів [4]. Матеріали визначають оптичні властивості поверхонь: наприклад, металевість, глянець, прозорість. Застосування матеріалів дозволяє контролювати, як текстури будуть взаємодіяти з різними типами освітлення.

На фінальному етапі обробки зображення застосовуються різноманітні візуальні ефекти, які допомагають підвищити якість рендерингу. Сюди входять такі ефекти, як розмиття, додання відблисків, корекція кольорів та підсилення контрасту. Постобробка дозволяє налаштувати візуальний стиль сцени, створити настрій і посилити відчуття реалізму.

Існує ряд технологій, які допомагають досягти високої продуктивності рендерингу в реальному часі, забезпечуючи ефективне використання апаратних ресурсів та покращуючи візуальну якість зображень.

Буфер глибини є однією з основних технологій, що дозволяє визначити порядок відображення об'єктів у сцені. Цей буфер зберігає інформацію про відстань кожного пікселя до камери, що дозволяє правильно накладати об'єкти один на одного. Буфер глибини особливо важливий для складних

сцен з великою кількістю об'єктів, оскільки він дозволяє коректно відображати об'єкти, які частково або повністю перекривають один одного.

Технології бамп-мапінг та нормал-мапінг, показана на рисуноку 1.1, дозволяють створити ілюзію високодеталізованих поверхонь без значного збільшення кількості полігонів. Бамп-мапінг використовує текстуру для симуляції нерівностей на поверхні об'єкта, що створює враження більшої деталізації. Нормал-мапінг працює подібним чином але дає можливість передати ще більше деталей, використовуючи напрямок поверхні на кожному пікселі, щоб відображення світла виглядало більш реалістично.

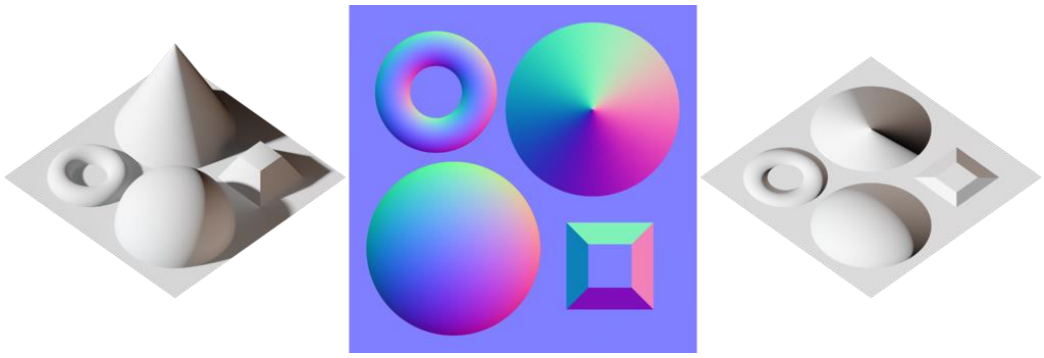

Рисунок 1.1 – Приклад праці мап нормалей

Рейкастинг та рейтрейсинг використовуються для обчислення відображень та світлових ефектів у сцені. Рейкастинг – це спрощений метод, що дозволяє швидко визначати видимі об'єкти та обчислювати прості тіні. Рейтрейсинг є більш точним, оскільки він моделює реальний шлях променів світла у сцені, але потребує більше обчислювальних ресурсів [5]. У реальному часі застосовуються спрощені версії рейтрейсингу ,наприклад, через технології RTX (Ray Rracing eXtreme), щоб досягти високої продуктивності без значного зниження якості.

Критерій оптимальності рендерингу включає досягнення балансу між трьома основними факторами:

- продуктивністю, що забезпечує стабільну частоту кадрів при обмежених апаратних ресурсах;

- якістю зображення, що гарантує відображення максимально деталізованих і реалістичних сцен без значного зниження продуктивності;

- ефективністю використання ресурсів, яка передбачає оптимальне використання пам'яті, обчислювальної потужності та енергоспоживання.

Ці етапи та технології утворюють основу процесу рендерингу в реальному часі, допомагаючи досягати оптимального співвідношення між швидкістю та візуальною якістю зображень.

1.2 Проблеми продуктивності при рендерингу в реальному часі

Однією з найбільших проблем при рендерингу в реальному часі є необхідність досягнення високої продуктивності при обмежених обчислювальних ресурсах. Графічні процесори мають певні обмеження по потужності, пам’яті та пропускній здатності, що вимагає ефективного використання ресурсів для рендерингу складних сцен. Основні проблеми продуктивності можна розділити на кілька категорій: обчислювальні ресурси, пам'ять, оптимізація геометрії та управління потоками даних.

Сучасні сцени можуть складатися з мільйонів полігонів, що створює значне навантаження на GPU (Graphics Processing Unit). Рендеринг великої кількості полігонів вимагає великих обчислювальних потужностей та часто призводить до падіння частоти кадрів. Для оптимізації цього процесу застосовують такі техніки, як рівні деталізації, що дозволяють зменшити кількість полігонів для об'єктів на великій відстані від камери. Також використовується відсікання невидимих об'єктів, яке виключає з рендерингу об’єкти, що знаходяться поза полем зору камери або закриті іншими об'єктами. Хоча ці методи допомагають підвищити продуктивність, вони можуть вимагати додаткових обчислень для вибору рівня деталізації та визначення видимих об'єктів, що в деяких випадках також знижує швидкодію.

Інша складність виникає з обробкою освітлення та тіней, які відіграють ключову роль у створенні реалістичних сцен. Освітлення включає як пряме, так і непряме світло, а також складні ефекти, такі як відбиття і рефракція. Для забезпечення фізично коректного освітлення використовується метод трасування променів, проте він є занадто обчислювально затратним для більшості задач реального часу. Щоб компенсувати це, використовуються менш ресурсоємні методи, такі як трасування променів на основі дендричних структур або освітлення на основі зображень, які зменшують обчислювальні витрати, хоча при цьому можуть обмежувати точність освітлення.

Текстурування і управління пам'яттю є ще однією проблемою. Великі текстури високої роздільної здатності споживають значну кількість пам'яті і вимагають високої пропускної здатності для швидкої передачі даних між пам’яттю та процесором. Щоб уникнути перевантаження пам'яті, використовуються стиснені текстури за допомогою таких форматів, як DXT (DirectX Texture) або ASTC (Adaptive Scalable Texture Compression), а також застосовуються MIP-текстури, показано на рисунку 1.2, які дозволяють використовувати текстури різної роздільної здатності в залежності від відстані до камери. Це значно знижує обсяг необхідної пам’яті, хоча стиснення може призводити до артефактів, які погіршують якість зображення.

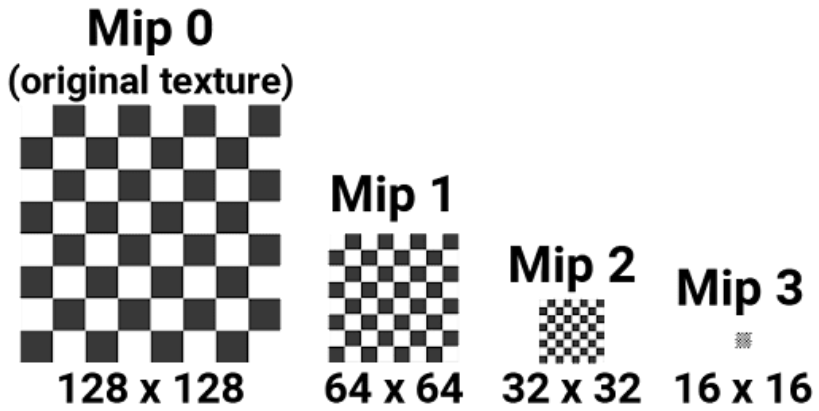

Рисунок 1.2 – MIP-текстури

Крім того, значні затримки можуть виникати при передачі даних між центральним процесором та графічним процесором, особливо коли в процесі рендерингу задіяні великі обсяги геометрії та текстур. У таких випадках саме CPU (Central Processing Unit) може стати «вузьким місцем» системи, обмежуючи швидкість рендерингу. Для оптимізації потоків даних застосовуються методи пакетної обробки, які дозволяють групувати схожі об'єкти для одночасного рендерингу, що значно зменшує кількість викликів рендерингу. Інший метод – асинхронний рендеринг, що дозволяє CPU виконувати інші завдання, поки GPU завершує обробку кадру, що значно покращує загальну продуктивність рендерингу.

1.3 Огляд сучасних технологій та методів рендерингу

Для реалізації рендерингу в реальному часі використовуються різні графічні API (Application Programming Interface), які дозволяють взаємодіяти з апаратним забезпеченням, таким як графічний процесор. Серед основних сучасних технологій, що використовуються для цієї мети, виділяються DirectX, Vulkan і OpenGL. Кожна з цих технологій має свої особливості, переваги та обмеження, і вибір між ними часто залежить від специфіки проекту та вимог до продуктивності.

DirectX – це API, розроблений компанією Microsoft, основним компонентом якого є Direct3D, який відповідає за 3D-графіку [6]. DirectX використовується головним чином на платформах Windows та Xbox, є стандартом для ігор на цих платформах. Direct3D 12 забезпечує розширений контроль над графічним апаратним забезпеченням, що дозволяє досягти вищої продуктивності, знижуючи накладні витрати на комунікацію між CPU і GPU. Основною перевагою DirectX є підтримка функцій низькорівневого програмування, які дозволяють ефективніше використовувати можливості сучасних GPU, а також відмінна інтеграція з екосистемою Windows.

Vulkan – це сучасний низькорівневий графічний API, розроблений Khronos Group. Vulkan є кросплатформним, що дозволяє використовувати його як на Windows, так і на Linux, Android та інших платформах. Основна перевага Vulkan полягає у його низьких накладних витратах і можливості ефективно керувати потоками даних між CPU і GPU, що дозволяє розробникам краще контролювати процес рендерингу. Vulkan надає більше можливостей для багатопоточності, що дозволяє ефективніше використовувати багатоядерні процесори. Це робить Vulkan особливо привабливим для складних додатків, де важлива висока продуктивність і низький рівень затримок.

OpenGL – це один із найстаріших і найпоширеніших графічних API. OpenGL забезпечує кросплатформну підтримку і використовується на різних операційних системах, включаючи Windows, macOS, Linux [6]. Однак, на відміну від Vulkan, OpenGL має вищий рівень абстракції, що робить його простішим у використанні, але менш ефективним з точки зору продуктивності. OpenGL є популярним вибором для навчальних цілей, невеликих проектів та програм, де максимальна продуктивність не є критичною.

Кожен з цих API має свої переваги та недоліки, і вибір між ними залежить від специфіки проекту. DirectX є найкращим варіантом для проектів, орієнтованих на платформу Windows, зокрема ігрових, завдяки високій інтеграції з екосистемою Microsoft. Vulkan підходить для проектів, де важлива кросплатформна підтримка та максимальна продуктивність, особливо для мобільних пристроїв та багатопроцесорних систем. OpenGL залишається простим і зручним інструментом для швидкої розробки і кросплатформної підтримки, але він поступово витісняється більш сучасними API через свої обмеження з точки зору продуктивності.

Сучасні вимоги до рендерингу, особливо для додатків віртуальної та доповненої реальності, часто потребують максимального контролю над апаратними ресурсами. У зв'язку з цим зростає популярність низькорівневих

API, таких як Direct3D 12 і Vulkan, які дозволяють розробникам створювати більш продуктивні і реалістичні графічні сцени.

1.3.1 Рейкастинг

Рейкастинг – це метод у комп’ютерній графіці (рисунок 1.3), який використовується для визначення взаємодії між променем і об’єктами у віртуальному просторі [7]. Він є основою для багатьох функцій у відеоіграх, 3D-моделюванні та фізичних симуляціях.

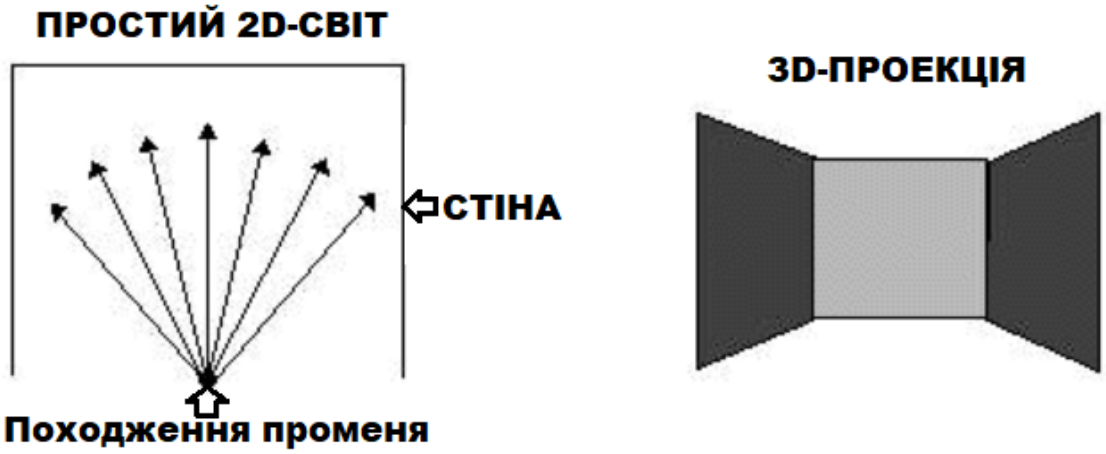

Рисунок 1.3 – Приклад роботи рейкастингу

Суть рейкастингу полягає в тому, що з певної точки у визначеному напрямку випускається уявний промінь. Цей промінь проходить через сцену, і за допомогою математичних розрахунків визначається, чи перетинається він із якимось об’єктом. У разі перетину фіксується точка контакту, відстань до об’єкта та інші параметри, які можуть бути корисними для обчислень.

Рейкастинг часто використовується в задачах визначення зіткнень між персонажами, об’єктами або поверхнями. Наприклад, він дозволяє визначити, чи бачить персонаж певний об’єкт, чи може гравець взаємодіяти з

предметом, на який наводить курсор. У фізичних симуляціях ця техніка допомагає розраховувати траєкторії польоту снарядів, визначати точки падіння або перевіряти можливість пересування.

1.3.2 Растеризація

Растеризація – це процес перетворення тривимірних об’єктів у двовимірне зображення на екрані, зображено на рисунку 1.4, що є основою сучасного рендерингу у реальному часі [7]. Растеризація використовується для побудови сцени, де об’єкти представлені у вигляді полігонів, зазвичай трикутників.

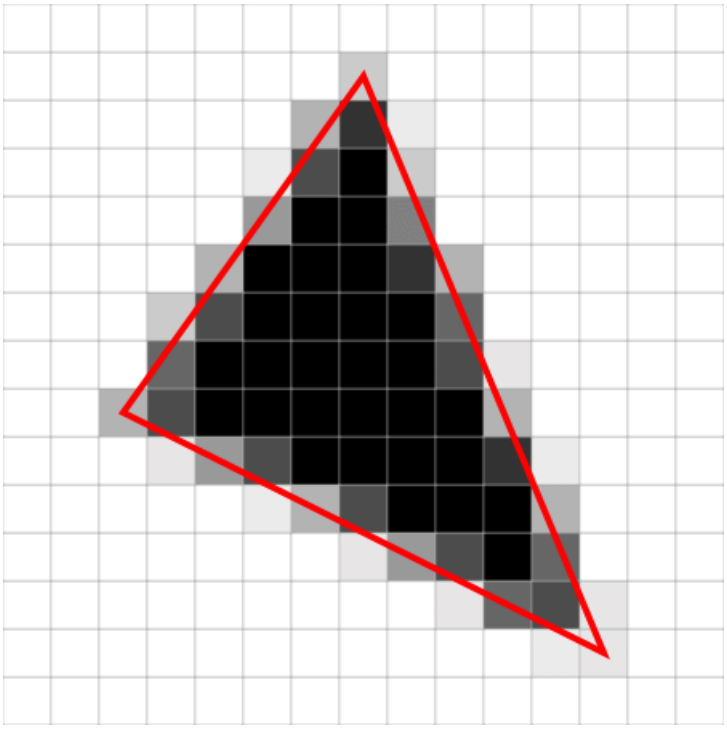

Рисунок 1.4 – Растеризація трикутника

Суть полягає у тому, що після проєкції тривимірних координат у двовимірний простір, кожен полігон розбивається на пікселі, які відповідають його формі та розташуванню. Для кожного пікселя

обчислюється його колір, що залежить від таких факторів, як матеріали об’єкта, освітлення та текстури.

Під час растеризації відбувається кілька етапів. Спершу відсікаються невидимі об’єкти, які знаходяться поза межами поля зору камери. Далі виконується визначення глибини, щоб вирішити, які об’єкти перекривають інші. На фінальному етапі додаються тіні, освітлення та текстури, щоб надати зображенню реалістичності.

Растеризація є основним методом рендерингу у сучасних відеоіграх завдяки її високій швидкості. На відміну від більш точного, але ресурсомісткого трасування променів, растеризація дозволяє досягати прийнятного рівня деталізації та реалістичності без значного впливу на продуктивність.

1.3.3 Трасування променів

Трасування променів – це метод рендерингу, який імітує фізичну поведінку світлових променів, щоб створювати реалістичні зображення [7]. Цей підхід забезпечує високу якість освітлення, рефлексій, рефракцій та тіней, що робить його ключовим інструментом у кіноіндустрії, 3D-анімації та сучасних відеоіграх.

Суть трасування променів полягає у випусканні променів із точки спостереження камери у віртуальну сцену. Кожен промінь перевіряє, чи перетинається він з об'єктами у сцені, і в разі перетину обчислюється, як світло взаємодіє з матеріалами цих об’єктів. Це враховує відбиття, заломлення та розсіювання світла, дозволяючи точно передати складні оптичні явища, такі як глибокі тіні, відблиски та прозорість.

Приклад сцени з використанням трасування променів зображено на рисунку 1.5.

Рисунок 1.5 – Рендеринг сцени за допомогою трасування шляху

Однією з ключових особливостей трасування променів є його здатність генерувати глобальне освітлення. Це означає, що враховуються не лише прямі джерела світла, але й вторинне світло, яке відбивається від поверхонь. Завдяки цьому сцени виглядають значно реалістичніше, ніж при використанні методів, що покладаються лише на растеризацію.

Хоча трасування променів забезпечує чудову якість зображення, воно є дуже ресурсомістким, оскільки вимагає обчислення великої кількості взаємодій між променями та об'єктами сцени. Раніше цей метод використовувався переважно для статичних зображень або попередньо рендерених сцен, але завдяки розвитку апаратного забезпечення, зокрема, графічних процесорів з підтримкою технології RTX, трасування променів стало доступним для рендерингу у реальному часі.

В Unreal Engine 5 трасування променів інтегрується разом із растеризацією, дозволяючи комбінувати переваги обох підходів. Це дає

змогу створювати сцени з фотореалістичним освітленням і високою продуктивністю, що є важливим для сучасних відеоігор та інтерактивних додатків [8].

1.4 Актуальні проблеми у 3d-графіці

Сучасна 3D-графіка знаходиться на перетині кількох швидко зростаючих галузей, таких як віртуальна реальність, доповнена реальність, відеоігри та симуляції. Кожна з цих областей має свої особливі вимоги до продуктивності, які суттєво впливають на методи та технології рендерингу. Незважаючи на досягнутий прогрес, існує ряд проблем, які досі потребують вирішення для покращення досвіду користувачів.

Віртуальна реальність пред'являє дуже високі вимоги до якості та швидкості рендерингу. Основною проблемою є необхідність забезпечити дуже високу частоту кадрів, зазвичай 90 FPS (Frames Per Second) або більше, щоб уникнути дезорієнтації та неприємних відчуттів у користувачів. Крім того, VR-системи (Virtual Reality) використовують два зображення для кожного ока, що подвоює обсяг необхідних обчислень. Затримки між діями користувача та відображенням результатів цих дій також мають бути мінімальними, щоб забезпечити реалістичне занурення в середовище. Це вимагає розробки спеціальних оптимізацій, таких як асинхронний рендеринг та використання технік, які дозволяють зменшити кількість полігонів та обчислень для окремих сцен.

Доповнена реальність, як і VR, має свої специфічні виклики, пов'язані з необхідністю інтеграції віртуальних об'єктів у реальний світ [9]. Трекінг і реєстрація об'єктів у реальному середовищі, що вимагає швидкого й точного аналізу оточення. AR (Augmented Reality) потребує як продуктивного рендерингу, так і розпізнавання та синхронізації з реальними об'єктами, що накладає додаткове навантаження на обчислювальні ресурси пристрою. Оптимізація рендерингу для AR, зокрема на мобільних пристроях, є

важливою, оскільки необхідно досягти високої продуктивності при обмежених апаратних можливостях [10].

Відеоігри також вимагають балансування між реалістичністю зображення та продуктивністю. Одним із викликів є необхідність реалістичного відображення великої кількості персонажів, об'єктів і ефектів у режимі реального часу. Особливо важко досягти оптимальної продуктивності в умовах відкритого світу, де користувач має можливість вільно переміщатися по великих територіях, що вимагає постійного завантаження та вивантаження ресурсів. Для вирішення цих проблем використовуються такі техніки, як тесселяція, оклюзійне відсікання, використання рівнів деталізації і попереднє розрахування деяких аспектів освітлення.

Симуляції в різних галузях, таких як наука, медицина чи військові тренування, також потребують високоякісної 3D-графіки в реальному часі. Головною проблемою є необхідність забезпечення точного відтворення фізичних явищ, таких як динаміка рідин або поведінка об'єктів при зіткненні, що вимагає значних обчислювальних ресурсів. У цих випадках рендеринг має поєднуватися з фізичними розрахунками, що ще більше збільшує навантаження на систему. Методи спрощення симуляцій, такі як використання спрощених моделей або масштабування на кількох рівнях, можуть допомогти підвищити продуктивність.

Таким чином, кожна з цих галузей має свої унікальні проблеми. Забезпечення реалістичності та інтерактивності при обмежених обчислювальних ресурсах вимагає використання передових методів оптимізації, спеціальних технік рендерингу та постійного розвитку апаратного забезпечення.

1.5 Постановка задач

Першим завданням є проведення огляду існуючих технологій, що використовуються для рендерингу 3D-графіки в реальному часі. Сюди

входить дослідження традиційних підходів, таких як растеризація трасування променів. Потрібно оцінити, як ці методи справляються із завданнями візуалізації, їхні сильні сторони, обмеження та сфери застосування. Особливу увагу буде приділено впливу цих підходів на швидкість роботи та якість зображення в умовах обмежених апаратних ресурсів.

Одним з аспектів продуктивності є зменшення обчислювальної складності. У цьому завданні будуть досліджені методи оптимізації, зокрема використання технік рівня деталізації, усунення зайвих елементів сцени за допомогою видалення не видимих предметів та оптимізації шейдерів. Крім того, необхідно розглянути роль багатопотокової обробки на GPU, включаючи технології паралельних обчислень для прискорення обробки графічних даних.

Завершальним етапом є розробка сцени, у якому буде проведено тестування продуктивності до і після застосування оптимізацій. На основі отриманих результатів буде проведено порівняльний аналіз, що дозволить оцінити ефективність запропонованих рішень.

2 МЕТОДИ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ТА ОПТИМІЗАЦІЇ РЕНДЕРИНГУ

2.1 Метод оптимізації геометрії та використання LOD

Високодеталізовані 3D-моделі можуть містити сотні тисяч або навіть мільйони полігонів, що значно підвищує вимоги до обчислювальних ресурсів. Для того, щоб забезпечити високу частоту кадрів і оптимізувати рендеринг, використовуються різні методи зменшення складності геометрії, одним із найпоширеніших з яких є використання рівнів деталізації.

Метод рівнів деталізації (LOD) спрямований на динамічне зменшення обчислювальних витрат під час рендерингу складних 3d-сцен шляхом використання моделей із різним ступенем деталізації. [10]. Залежно від відстані об'єкта до камери, використовується відповідний рівень деталізації: якщо об'єкт знаходиться близько до камери, застосовується високодеталізована версія, а якщо далеко – спрощена. Це дозволяє значно зменшити навантаження на графічний процесор, оскільки об'єкти на віддаленій відстані не потребують високої деталізації, яку користувач все одно не помітить.

Спочатку підбирається підхід до створення рівнів деталізації.

Простий підхід: У цьому випадку кожен рівень деталізації створюється вручну. Моделі для кожного рівня можуть бути згенеровані за допомогою спеціальних інструментів або вручну через 3D-редактор.

Автоматичні алгоритми: використання алгоритмів для автоматичного спрощення моделей на основі певних критеріїв. Такі алгоритми дозволяють зменшити кількість полігонів, видалити малозначущі деталі чи об'єднати полігони без значних втрат у вигляді.

Алгоритм Quadric Error Metrics: Він використовує матриці для обчислення найкращих варіантів злиття вершин з мінімальними втратами у геометрії.

Edge Collapse: Метод злиття сусідніх вершин для спрощення геометрії. Цей алгоритм визначає пари вершин, які можна об'єднати, щоб зберегти форму об'єкта.

Decimation: Підхід, при якому видаляються полігони, що мають низький вплив на вигляд моделі. Алгоритм відбирає полігони на основі їх важливості або за допомогою порогових значень.

Surface Simplification: Загальний метод, який дозволяє спрощувати поверхні шляхом видалення малозначущих частин або сплощення геометрії.

Починається все зі створення максимально деталізованої 3D-моделі, яка служить основою для подальших версій із різними рівнями деталізації. Називатись вона буде LOD 0. Така модель створюється з великою кількістю полігонів, що дозволяє зобразити навіть найдрібніші деталі. Наприклад, для персонажа це можуть бути зморшки на обличчі, текстура шкіри, окремі пасма волосся або складки на одязі. Далі створюються спрощені версії цієї моделі:

LOD 1: Ця версія зберігає більшість важливих деталей, але спрощує дрібні елементи. Наприклад, текстура волосся може бути об'єднана в одну групу, а складки на одязі згладжені.

LOD 2: У цьому рівні деталізації зникають другорядні деталі, такі як аксесуари чи окремі декоративні елементи. Замість складної геометрії використовуються текстури для передачі деяких візуальних особливостей.

LOD 3: Це найспрощеніша версія, яка часто зводиться до базового силуету об'єкта. У ній майже відсутні текстури та додаткові елементи, але зберігається загальна форма об'єкта для розпізнавання з великих відстаней.

При створенні кожної версії змінюється не тільки кількість полігонів, але й текстури: високоякісні текстури замінюються менш деталізованими або взагалі спрощеними, що додатково зменшує навантаження на систему.

Після створення всіх версій моделі вони додаються до сцени через систему LOD. Цей етап передбачає налаштування порогових значень, що визначають, коли саме має використовуватися кожен рівень деталізації.

Наприклад, у Unreal Engine за допомогою компонента LOD Group кожна модель прив'язується до певного діапазону відстаней. Приклад рівнів деталізації зображено на рисунку 2.1. Для об'єкта можна встановити такі межі:

- LOD 0 використовується на відстані до 20 метрів від камери;
- LOD 1 активується в діапазоні від 21 до 50 метрів;
- LOD 2 активується в діапазоні від 51 до 100 метрів;
- LOD 3 застосовується для відстаней більше 101 метрів.

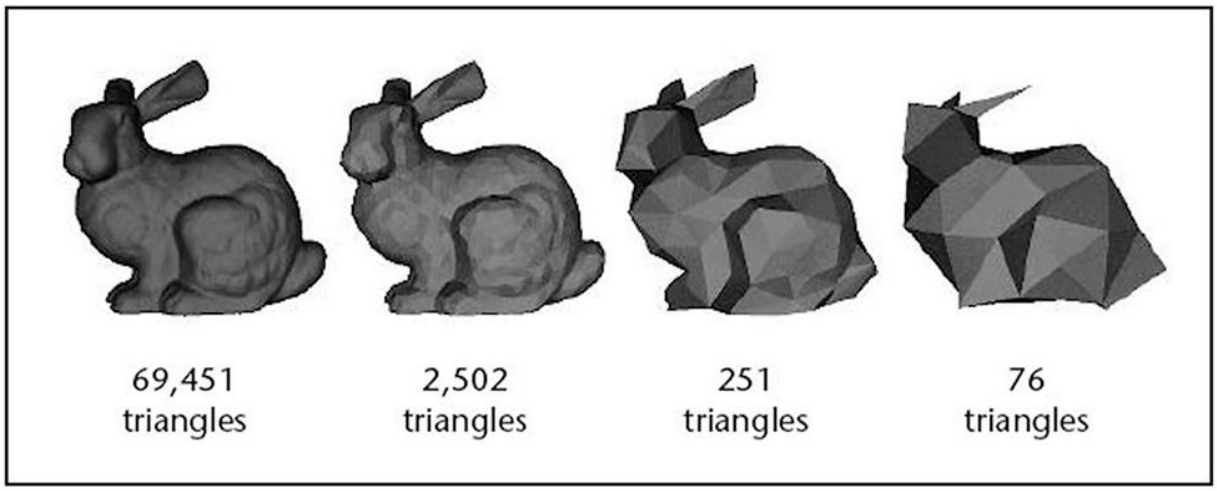

Рисунок 2.1 – Приклад рівнів деталізації

Окрім налаштувань відстаней, важливо також забезпечити узгодженість масштабів, позицій і орієнтації для всіх рівнів моделі. Це потрібно для того, щоб під час переходу між версіями не виникали візуальні розриви чи артефакти.

Також у програмному забезпеченні налаштовуються параметри плавності переходів між рівнями деталізації. Динамічне перемикання між

рівнями деталізації – це процес, який відбувається в реальному часі під час роботи сцени. Коли камера рухається, система LOD постійно аналізує відстані до об’єктів та адаптує рівень деталізації відповідно до встановлених параметрів.

Це перемикання зазвичай здійснюється на кожному кадрі, що дозволяє підтримувати високу продуктивність навіть у динамічних сценах із великою кількістю об’єктів. Для цього використовується механізм кросфейдингу або миттєвого переключення:

Кросфейдинг: Об’єкт плавно змінює вигляд від одного рівня деталізації до іншого, зменшуючи різкі переходи. Наприклад, текстури старого рівня поступово зникають, а нові – проявляються.

Миттєве перемикання: Система замінює одну модель іншою без проміжної анімації. Це більш простий і швидкий метод, але він може викликати візуальні «стрибки», якщо моделі дуже відрізняються.

Для коректної роботи динамічного перемикання важливо налаштувати порогові значення відстаней між рівнями так, щоб зміни були малопомітними для користувача. Наприклад, перехід із LOD 0 на LOD 1 має відбуватися, коли об’єкт досить далекий, щоб дрібні деталі вже не сприймалися оком.

На основі обчислень відстані між камерою та об'єктами сцени система вирішує, який рівень деталізації використовувати для кожного конкретного об'єкта.

Процес обчислення базується на формулі евклідової відстані:

$$distance = \sqrt{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2}, \qquad (2.1)$$

де x1, y1, z1 – координати камери;

x2, y2, z2 – координати об’єкта.

Це значення передається системі, яка на основі встановлених порогів обирає відповідний рівень моделі.

Обчислення відстаней виконуються для кожного об'єкта сцени на кожному кадрі, щоб забезпечити правильну адаптацію рівня деталізації під час руху камери або зміни позицій об'єктів.

Для оптимізації продуктивності у великих сценах обчислення часто обмежують до певних зон або груп об'єктів, які перебувають у полі зору камери. Це дозволяє значно зменшити навантаження на процесор і графічну карту.

Після вибору рівня деталізації об’єкт проходить етап рендерингу, який включає завантаження відповідної геометрії, текстур та матеріалів у пам’ять графічного процесора. На ближніх відстанях рендериться модель із високою кількістю полігонів. На середніх відстанях ці деталі об’єднуються у простіші структури. Обличчя персонажа може стати гладким, а дрібні елементи – частиною текстури. У моделях із високою деталізацією використовуються текстури великої роздільної здатності, 4K чи більше, та складні шейдери. Для далеких об'єктів текстури спрощуються до мінімуму, наприклад, 512x512 пікселів, а шейдери замінюються на базові.

Етапи реалізації методу LOD:

- на початковому етапі проводиться оцінка сцени;
- створення максимально деталізованої моделі;
- створення спрощених моделей;
- інтеграція LOD у сцену;
- забезпечення плавності переходів.

Етапи реалізації методу зображено на рисунку 2.2.

2.2 Методи видалення невидимих об'єктів

У сучасній 3D-графіці видалення невидимих об'єктів є одним із ключових методів оптимізації рендерингу. Ресурси обчислення, які

використовуються для відображення сцени, обмежені, особливо в реальному часі, тому важливо зменшити кількість об'єктів, які потрібно опрацьовувати та рендерити. Методи видалення невидимих об'єктів дозволяють виключити зі сцени ті елементи, які не впливають на кінцевий результат, тобто не видно з точки зору камери.

Ці методи допомагають зменшити навантаження на центральний процесо та графічний процесор, підвищуючи продуктивність додатка. У деяких випадках вони дозволяють досягти зменшення часу рендерингу до кількох разів, що є критично важливим для інтерактивних додатків, таких як ігри, віртуальна реальність та інженерні симуляції.

Рисунок 2.2 – Етапи реалізації методу LOD

Основні методи для виключення невидимих об'єктів: Frustum Culling, Backface Culling, Occlusion Culling. Кожен з них має свою специфіку, переваги та недоліки, залежно від складності сцени та ресурсів обчислення.

2.2.1 Frustum Culling

Frustum Culling – це метод оптимізації рендерингу в комп'ютерній графіці, що дозволяє зменшити кількість об’єктів, які потрібно обробляти графічному процесору. Основна мета полягає в тому, щоб виключити з процесу рендерингу об’єкти, які не перебувають у полі зору камери, приклад роботи зображено на рисунку 2.3. Це підвищує продуктивність графічної системи та зменшує навантаження на апаратні ресурси.

Усічений конус, що визначається камерою, є основою цього методу. Ця область видимості обмежується кількома площинами: передньою, дальньою, а також чотирма боковими площинами, які визначають ширину та висоту фрустума. Передня площина знаходиться найближче до камери, і все, що знаходиться ближче, не буде видимим. Рівняння цієї площини формується на основі координат камери та кута огляду. Дальня площина визначає найдальшу точку видимості. Всі об’єкти, що знаходяться поза цією площиною, також не рендеряться. Її рівняння розраховується подібно до передньої площини, але з урахуванням відстані до найбільш віддаленого об’єкта. Бокові площини обмежують видимість за горизонталлю та вертикаллю. Їхні рівняння визначаються на основі кутів огляду камери та її орієнтації в просторі. Усі об'єкти, що перебувають всередині цієї області або частково її перетинають, потребують рендерингу. Об’єкти, що повністю знаходяться поза межами, відсікаються й не обробляються графічним процесором.

Процес Frustum Culling починається з побудови фрустума камери. Frustum – це піраміда з усіченою вершиною, що визначається параметрами камери:

- положення в просторі;
- напрямок погляду;
- кут огляду;
- відношення сторін;

- ближня та дальня площини відсікання.

Frustum має 6 площин, що обмежують видиму область:

- ліва;
- права;
- верхня;
- нижня;
- ближня;
- дальня.

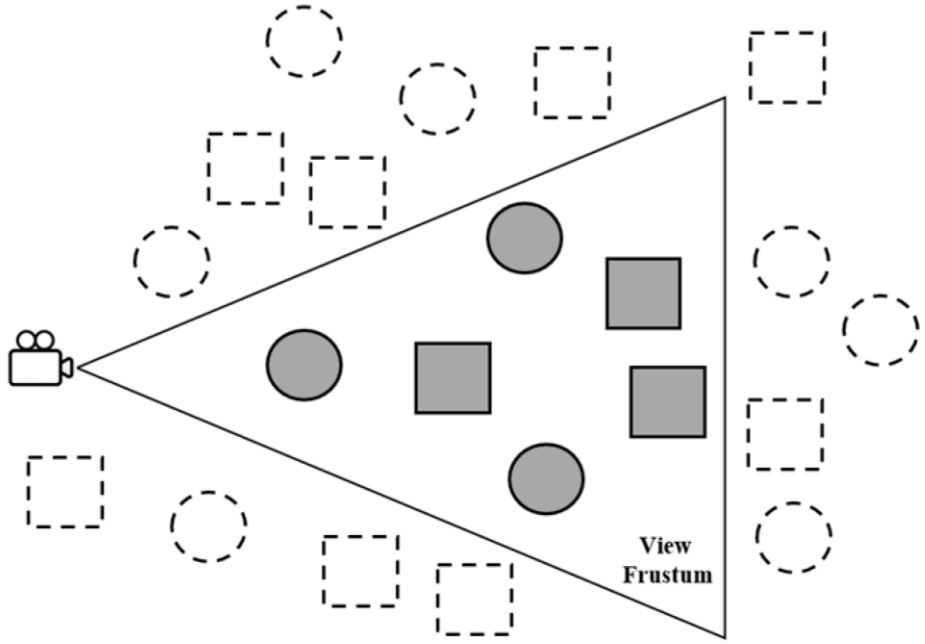

Рисунок 2.3 – Поле огляду камери

Для кожної з 6 площин визначається рівняння у вигляді:

$$A_x + B_y + C_z + D = 0, \tag{2.2}$$

де A, B, C — це нормальний вектор площини;

D — відстань до початку координат.

Координати (A, B, C, D) можна отримати з матриці проекції та видової матриці камери.

На наступному етапі для кожного об'єкта або його обмежуючого об'єму, наприклад, сфера або паралелепіпед, виконується перевірка на знаходження з кожної сторони площини Frustum. Для цього підставляються координати характерних точок об'єкта, наприклад, центр сфери або вершини паралелепіпеда, в рівняння площини. При використанні сфери для перевірки, використовується така формула:

$$Ax_c + By_c + Cz_c + D \geq -r, \tag{2.3}$$

де x_c, y_c, z_c — координати центра сфери;

r — радіус.

Якщо обмежувальний об'єм об'єкта знаходиться поза хоча б однією площиною фрустума, цей об'єкт виключається з рендерингу. У результаті, рендеряться тільки ті об'єкти, які залишилися у межах видимої області.

Для перевірки об’єктів можуть використовуватися різні підходи, такі як послідовна перевірка площин або використання спеціальних структур даних, наприклад, Octree чи Bounding Volume Hierarchy. Octree – дерево, де кожен вузол представляє куб, розділений на вісім менших кубів. При перевірці з фрустумом спочатку тестуються великі області, вузли дерева. Якщо вся область знаходиться поза фрустумом, усі об’єкти в цій області автоматично виключаються без додаткових перевірок. У випадку BVH (Bounding Volume Hierarchy), об’єкти організовуються за принципом вкладених обмежувальних об’ємів. Перевірка починається з верхнього рівня і поступово заглиблюється тільки в ті області, які можуть бути видимими.

Ці структури дозволяють ефективно обробляти великі сцени, швидко виключаючи великі групи об’єктів, які не потрапляють у фрустум. Для додаткової оптимізації застосовуються SIMD-інструкції, що дозволяють

виконувати перевірки паралельно. При використанні SIMD можна одночасно перевірити положення чотирьох вершин об'єкта відносно площини фрустума. Це значно зменшує кількість циклів перевірки й прискорює обробку, особливо у випадках, коли на сцені присутня велика кількість об’єктів.

Етапи реалізації методу зображено на рисунку 2.4.

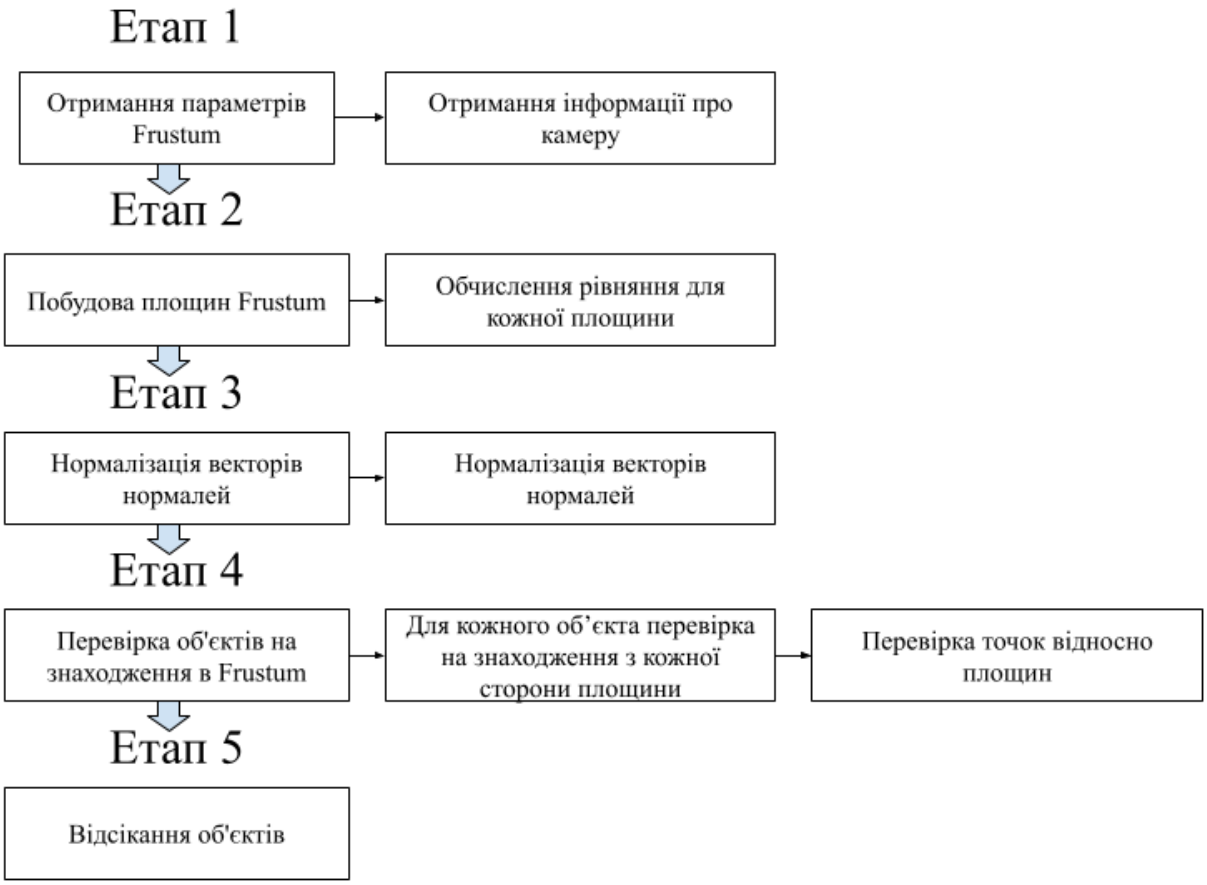

Рисунок 2.4 – Етапи реалізації методу Frustum Culling

Frustum Culling широко використовується в 3D-іграх для оптимізації рендерингу великих сцен із численними об’єктами. Також цей метод є важливим у програмах для візуалізації великих наборів даних і в архітектурних симуляціях. Використання цього методу дозволяє досягти значного зниження навантаження на систему та підвищення продуктивності.

Метод має свої переваги та обмеження. До його переваг належать значне зменшення кількості об'єктів для обробки, підвищення швидкості рендерингу та простота реалізації для статичних сцен. Однак Frustum Culling не враховує випадки, коли об’єкти перекриваються, і не виключає з обробки ті з них, що приховані іншими об’єктами.

2.2.2 Occlusion Culling

Occlusion Culling – це метод оптимізації графічного рендерингу, який дозволяє підвищити продуктивність, виключаючи з процесу відображення об’єкти, що повністю приховані іншими об’єктами, як зображено на рисунку 2.5.

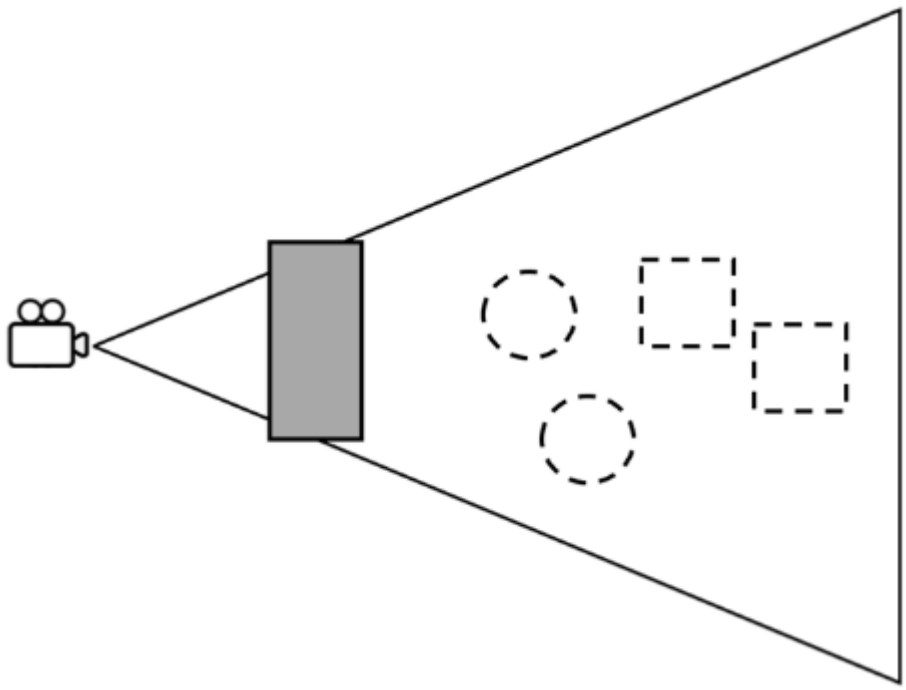

Рисунок 2.5 – Принцип роботи оклюзії

Головна ідея полягає в тому, щоб перевірити видимість кожного об'єкта сцени. Якщо об'єкт повністю перекритий іншими елементами сцени, він виключається з процесу рендерингу. Для цього метод використовує

глибину сцени (Z-буфер) і обмежувальні об'єми (Bounding Volume), які спрощують геометричну форму об'єктів для перевірки.

Процес визначення видимості починається із використання Z-буфера. Z-буфер зберігає глибину для кожного пікселя сцени, яка відповідає відстані до найближчого об’єкта на цьому пікселі. Під час рендерингу нового об’єкта його пікселі перевіряються на відповідність умові:

$$Z_{object} < Z_{buffer}, \tag{2.4}$$

де Z_{object} – значення глибини пікселя об’єкта;

Z_{buffer} – поточне значення глибини в Z-буфері.

Якщо глибина пікселя об’єкта є меншою за поточну глибину у Z-буфері, цей піксель вважається видимим, і значення глибини в Z-буфері оновлюється. У протилежному випадку піксель залишається прихованим, а об’єкт може бути частково чи повністю виключений з рендерингу.

Для оптимізації перевірки використовуються обмежувальні об’єми, які є спрощеними геометричними формами, що апроксимують складну геометрію об'єктів. Обмежувальний об'єм у вигляді сфери задається координатами центру (*x0, y0, z0*) і радіусом *r*. Сфера вважається видимою, якщо хоча б частина її поверхні розташована ближче до камери, ніж поточне значення глибини, записане в Z-буфері. Сфера вважається видимою, якщо хоча б одна її частина проходить умову:

$$Z_{sphere} - r < Z_{buffer}, \tag{2.5}$$

де Z_{sphere} – відстань від камери до цента сфери;

Z_{sphere} – поточне значення глибини в Z-буфері.

Це означає, що різниця між відстанню від камери до центра сфери та її радіусом є меншою за значення глибини в Z-буфері. Іншими словами, якщо

зовнішня поверхня сфери потрапляє у видиму область, її слід обробляти під час рендерингу.

Проекція об'єктів на площину екрану виконується за допомогою матриці проекції камери. Всі вершини об’єктів трансформуються з координат сцени у координати екрана. Після цього виконується порівняння отриманої проекції об'єкта із значеннями глибини у Z-буфері. Це дозволяє швидко визначити, чи перекритий об'єкт іншими елементами сцени, і прийняти рішення щодо його рендерингу.

Реалізація методу Occlusion Culling складається з кількох етапів, зображено на рисунку 2.6, кожен з яких важливий для досягнення оптимальної продуктивності рендерингу. Першим етапом є підготовка сцени. На цьому етапі визначаються статичні об'єкти, тобто ті, що не рухаються, наприклад, будівлі, стіни чи ландшафт. Саме ці об'єкти використовуватимуться для розрахунку затінення. У багатьох графічних двигунах, таких як Unity або Unreal Engine, необхідно явно позначити статичні об'єкти як Occluder (ті, що затінюють) та Occludee (ті, що можуть бути затінені). Важливо пам'ятати, що об'єкт може бути одночасно і тим, і іншим. Також на цьому етапі налаштовуються додаткові параметри, що впливають на якість та продуктивність Occlusion Culling, наприклад, розмір або рівень деталізації.

Наступний етап – генерація даних затінення. Тут створюється ієрархічна структура, що описує взаємне розташування об'єктів та їх здатність затінювати один одного. Зазвичай для цього використовується бінарне дерево або подібні структури. Для кожної частини сцени, або ячейки, обчислюється, які об'єкти видимі, а які затінені. Цей процес може бути досить ресурсомістким, тому його виконують заздалегідь, на етапі розробки.

Третій етап – використання даних затінення під час виконання гри. Спочатку дані затінення завантажуються в пам'ять. Потім кожного кадру для кожного об'єкта перевіряється, чи він видимий з поточної позиції камери, використовуючи заздалегідь обчислені дані. Якщо об'єкт повністю закритий

іншими об'єктами, він виключається з процесу рендерингу. Для перевірки видимості обмежувального об'єму об’єкт проектується на площину екрану. Використовується матриця проекції камери *P*:

$$v_{screen} = P * v_{world}, \tag{2.6}$$

де v_{screen} – її координати на екрані;

v_{world} – координати вершини в просторі сцени;

P – матриця проекції камери.

Після проекції перевіряється накладання об’єкта на вже відображені частини сцени в Z-буфері.

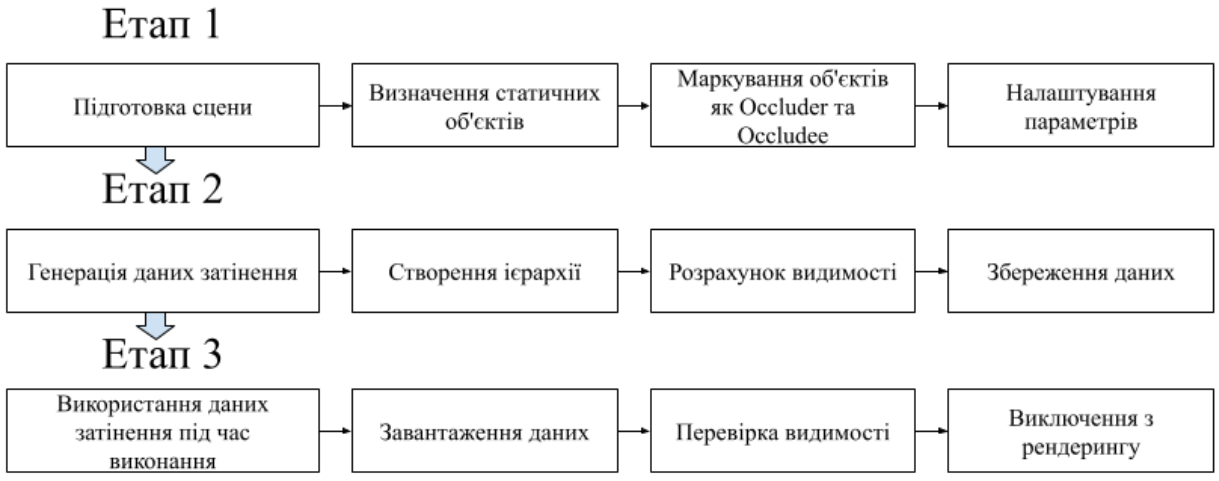

Рисунок 2.6 – Етапи реалізації методу Occlusion Culling

2.2.3 Backface Culling

Backface Culling – це метод, який виключає з процесу рендерингу грані об’єктів, невидимі з точки зору камери. Основна ідея полягає у відмові від відображення граней полігонів, які відвернуті від камери. Оскільки внутрішні поверхні об'єктів зазвичай невидимі для спостерігача, їх рендеринг є зайвим і

призводить до непотрібних витрат обчислювальних ресурсів. Застосування Backface Culling дозволяє значно зменшити кількість полігонів, що обробляються графічним процесором, що безпосередньо впливає на підвищення продуктивності рендерингу та збільшення частоти кадрів. Це особливо важливо для складних сцен з великою кількістю полігонів.

Однак, Backface Culling має певні обмеження. По-перше, він не працює для не замкнутих об'єктів, таких як площини або листки. Для цих об'єктів обидві сторони можуть бути видимі залежно від положення камери, тому відсікання однієї з них призведе до некоректного відображення. По-друге, для коректної роботи Backface Culling необхідно, щоб усі грані об'єкта мали узгоджений порядок обходу вершин, тобто всі грані повинні бути визначені або за годинниковою стрілкою, або проти. Неузгодженість порядку обходу вершин призведе до того, що деякі видимі грані будуть помилково відсічені, або навпаки, невидимі грані будуть відображені.

Backface Culling працює на рівні окремих трикутників або полігонів і дозволяє зменшити обчислювальне навантаження на графічний процесор. Ця техніка базується на тому, що в більшості випадків внутрішні поверхні об'єктів або ті, що повернуті «спиною» до камери, не потрібно відображати.

Для кожного трикутника, визначеного вершинами v_1, v_2, v_3, обчислюється нормаль n. Нормаль – це вектор, перпендикулярний до площини трикутника, який визначає його орієнтацію у просторі. Вона обчислюється через векторний добуток двох векторів, які утворюють ребра трикутника:

$$n = (v_2 - v_1) * (v_3 - v_1), \tag{2.7}$$

де $v_2 - v_1$ та $v_3 - v_1$ є векторами, що утворюють дві сторони трикутника. Нормаль визначає, з якого боку трикутник вважається «лицевим».

Щоб визначити, чи повернутий трикутник до камери, розраховується вектор d, який вказує від будь-якої точки на площині трикутника до позиції камери. Зазвичай для цього беруть першу вершину трикутника v_1. Формула обчислення вектора напрямку:

$$\mathrm{d} = C - v_1, \tag{2.8}$$

де C — це позиція камери у просторі;

d — вказує напрямок від трикутника до камери.

Для визначення, чи нормаль трикутника спрямована до камери, використовується скалярний добуток між нормаллю n і вектором напрямку d. Скалярний добуток обчислюється як:

$$n * d = n_x d_x + n_y d_y + n_z d_z, \tag{2.9}$$

де n_x, n_y, n_z — компоненти нормалі;

d_x, d_y, d_z — компоненти вектора d.

Скалярний добуток визначає, чи кут між n та d є гострим, тупим або прямим:

Якщо $n * d > 0$, трикутник спрямований до камери.

Якщо $n * d \leq 0$, трикутник спрямований від камери.

Цей критерій дозволяє відсікти значну кількість граней об'єктів, які не видимі для камери, тим самим оптимізуючи процес рендерингу.

Етапи реалізації методу зображено на рисунку 2.7.

Рисунок 2.7 – Етапи реалізації методу Backface Culling

2.3 Метод покращення продуктивності шейдерів

Шейдери – це спеціальні програми, які виконуються на графічному процесорі та використовуються для обробки зображень і графічних даних. Їх основна функція полягає у визначенні вигляду пікселів і вершин геометрії у кінцевому рендерингу. Шейдери забезпечують обробку кольорів, текстур, освітлення та різноманітних візуальних ефектів, які додають деталізації до графічних сцен.

Одним із ключових типів шейдерів є вершинний шейдер, який обробляє вершини геометрії. Він виконує трансформацію положення вершин у просторі, наприклад, переводить їх із локального простору у проєкційний. Такий шейдер також може відповідати за налаштування освітлення для кожної вершини. Іншим важливим видом є піксельний шейдер, який працює з кольорами й текстурами кожного пікселя. Він додає деталізацію через тіні, освітлення, прозорість і відбиття, забезпечуючи фотореалістичний вигляд сцени.

Геометричний шейдер обробляє примітиви, наприклад трикутники або лінії, і дозволяє змінювати їх форму або створювати нові примітиви. Це розширює можливості графічного рендерингу. Комп’ютерний шейдер виконує завдання, які не завжди пов’язані з графікою, наприклад симуляції фізичних процесів чи обробку великих масивів даних. Завдяки цьому його можна використовувати у складних обчислювальних задачах. Процес визначення, які шейдери підлягають оптимізації, складається з декількох етапів.

Перший етап – це профілювання шейдера**,** яке передбачає використання спеціальних інструментів, таких як RenderDoc або NVIDIA Nsight, для аналізу продуктивності. У процесі профілювання збираються ключові метрики, що включають кількість виконаних інструкцій, обсяг використаної пам’яті, а також кількість викликів шейдера в кадрі.

Другий етап – виявлення вузьких місць. На основі зібраних даних визначаються основні проблеми, які впливають на продуктивність шейдера. Це можуть бути надмірно складні інструкції, неефективне використання пам’яті або велика кількість викликів. Виявлені проблеми дозволяють зосередитися на конкретних аспектах, які потребують оптимізації.

Третій етап – вибір методів оптимізації. Залежно від результатів профілювання, обирається найбільш релевантний метод або комбінація методів

На четвертому етапі здійснюється впровадження оптимізації. Це передбачає внесення змін у код шейдера. Наприклад, можуть бути замінені складні функції на більш ефективні, оптимізовані структури даних або змінені параметри текстур і ресурсів, що використовуються у процесі рендерингу. Цей етап часто потребує тестування, щоб перевірити, чи зберігається якість зображення на прийнятному рівні.

Заключний етап – це оцінка результатів. Після внесення змін виконується повторне профілювання, щоб оцінити новий час виконання і

якість. Якщо результати виявляються задовільними, процес оптимізації завершується

Продуктивність шейдерів оцінюється за кількома ключовими параметрами. Серед них час виконання інструкцій, що позначається як T, кількість кадрів за секунду, що характеризує плавність оновлення зображення, а також рівень завантаження GPU і використання пам’яті, які мають бути збалансованими для забезпечення стабільної роботи.

Оптимізація шейдерів полягає у скороченні часу виконання програм при збереженні або мінімальному зниженні якості візуалізації. Основна ціль оптимізації формулюється таким чином:

$$\text{minT, за умови } Q \geq Q_{min}, \tag{2.10}$$

де T — це час виконання шейдера;

$minT$ — мінімізація часу виконання шейдера;

Q — якість зображення;

Q_{min} — мінімально допустимий рівень якості.

Якість зображення Q може бути оцінена за допомогою декількох методів. Групі користувачів демонструються зображення або сцени, і вони оцінюють їх за шкалою, наприклад, від 1 до 10. Цей підхід дає результати, орієнтовані на кінцевого споживача, але він залежить від суб'єктивності і складний у масштабуванні. Аналізуються артефакти, що виникають після компресії чи спрощення сцени, такі як розмиття, пікселізація чи шум. Чим менше артефактів, тим вище значення Q.

Q_{min} встановлюється залежно від вимог до конкретного застосування. Мінімально допустима якість може бути пов'язана з продуктивністю. Наприклад, якщо FPS падає нижче 30, це вважається неприйнятним для багатьох застосувань. У процесі оптимізації важливо враховувати, як різні

методи впливають на продуктивність шейдерів та якість зображення. Оптимальні рішення досягаються шляхом аналізу взаємозв’язку між витратами обчислювальних ресурсів і отриманим результатом.

Формулу залежності між змінами у продуктивності та внесеними оптимізаційними рішеннями можна записати наступним чином:

$$T_{new} = T - (\Delta T_C + \Delta T_M + \Delta T_R + \Delta T_L), \quad (2.11)$$

де T — час виконання шейдера;

ΔT_C — скорочення часу завдяки оптимізації коду шейдера;

ΔT_M — скорочення часу через зменшення використання пам’яті;

ΔT_R — скорочення часу, досягнуте шляхом оптимізації ресурсів;

ΔT_L — скорочення часу за рахунок ефективнішого освітлення чи обчислень.

Кожне з цих зменшень досягається за допомогою різних підходів, таких як переписування шейдерного коду, зменшення складності алгоритмів або використання більш ефективних текстур.

Для впровадження методу, потрібно виконати декілька етапів.

Етап 1: Цей етап є фундаментом для всієї оптимізації. Він полягає у зборі даних про роботу шейдера за допомогою спеціалізованих інструментів, таких як RenderDoc або NVIDIA Nsight. Ці інструменти дозволяють отримати детальну інформацію про продуктивність шейдера, включаючи кількість виконаних інструкцій, обсяг використаної пам'яті та кількість викликів шейдера в кадрі.

Етап 2: На основі зібраних даних профілювання відбувається аналіз та ідентифікація проблем, що найбільше впливають на продуктивність. Типовими проблемами є надмірно складні інструкції, неефективне використання пам'яті та велика кількість викликів шейдера.

Етап 3: На цьому етапі безпосередньо вносяться зміни в код шейдера. Важливим є тестування після кожної зміни для перевірки збереження візуальної якості зображення. Оптимізація не повинна призводити до неприйнятних візуальних артефактів.

Етап 4: Після внесення змін необхідно провести повторне профілювання для оцінки ефективності оптимізації. Ключовими параметрами для оцінки є час виконання, який потрібно мінімізувати, кількість кадрів за секунду, яку потрібно максимізувати, завантаження GPU та використання пам'яті, які потрібно збалансувати, та якість зображення, яку потрібно зберегти на прийнятному рівні.

Етапи реалізації методу зображено на рисунку 2.8.

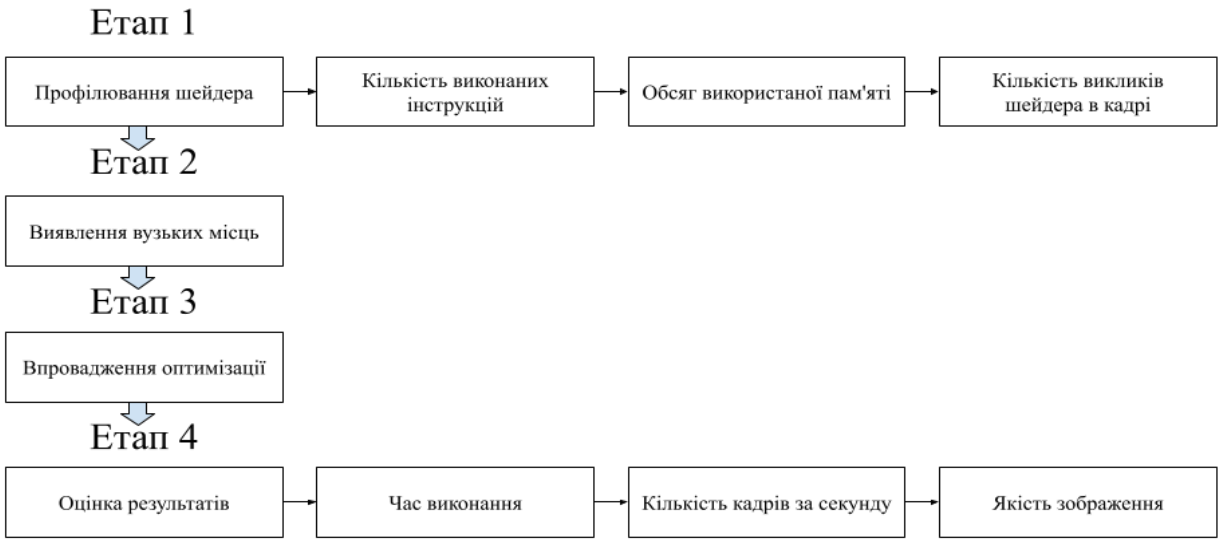

Рисунок 2.8 – Етапи реалізації методу покращення продуктивності шейдерів

2.4 Метод трасування променів у реальному часі

У природі джерело світла випромінює промінь, який рухається, поки його не перешкодить який-небудь об'єкт. Цей «промінь» можна уявити як потік фотонів, що летять в одному напрямку. Якщо не враховувати релятивістські ефекти, у вакуумі цей потік рухатиметься по прямій лінії.

Однак у реальному середовищі променю можуть впливати різні явища, такі як поглинання, відбиття, заломлення, флуоресценція та інші. Поверхня може відбивати світло в одному або кількох напрямках, або ж розсіювати його. Частина світла може бути відбита, частина розсіяна, а частина поглинена матеріалом. Якщо ж поверхня прозора, світло проходитиме через неї, частково поглинаючись і змінюючи колір променя, а його напрямок буде змінюватися через заломлення. Іноді поглинене світло може бути знову випромінене у вигляді флуоресценції, змінюючи колір і напрямок променя. Сума всіх ефектів – відбитого, заломленого, поглиненого і випроміненого світла – має дорівнювати отриманому світлу. Подальші промені відбитого, заломленого чи випроміненого світла потрапляють на інші об'єкти, де зазнають подібних ефектів. Частина з цих променів досягає наших очей, і завдяки інформації про їх колір і інтенсивність ми можемо сприймати навколишній світ [13].

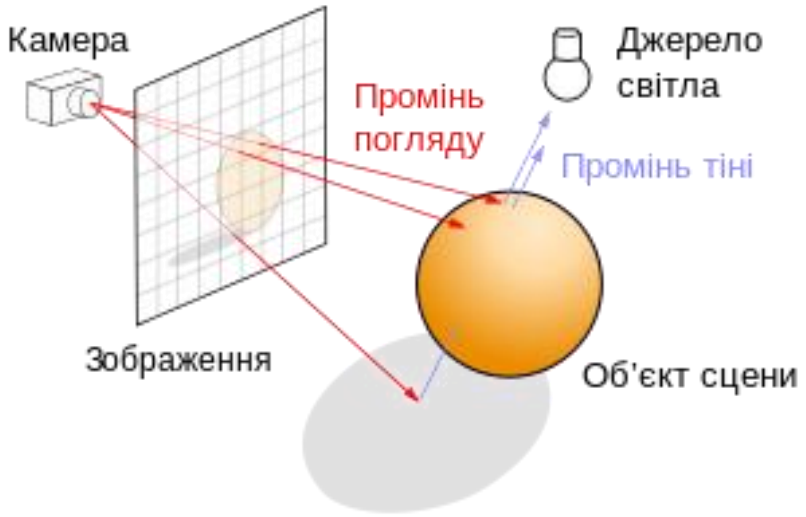

Рисунок 2.9 – Принцип роботи трасування променів

Принцип роботи методу на основі трасування променів базується на фізичних законах поширення світла, зображено на рисунку 2.9. Для цього кожен об’єкт на сцені має матеріал з наступними властивостями:

- випромінювання енергії – кількість та довжина хвилі світла, яку випромінює об’єкт;

- шорсткість – наскільки сильно промені розсіюються при зіткненні з об’єктом;

- відбивна здатність – кількість та довжина хвилі світла, яку відбиває об’єкт;

- прозорість – відношення пропущеного крізь об’єкт світла до відбитого;

- заломлення – показник заломлення світла.

На початковому етапі для кожного пікселя на екрані випускається промінь. Якщо промінь взаємодіє з об’єктом у сцені, його подальша поведінка залежить від властивостей матеріалу: він може відбиватися, заломлюватися або розсіюватися.

При дзеркальному відбиванні кут променя, який відбився, буде дорівнювати куту променя, який падає.

При дифузному відбиванні відбиті промені розходяться в різних напрямках, що пояснюється розсіюванням світла на нерівних поверхнях. Оскільки в програмному коді неможливо ідеально відтворити текстуру поверхонь об'єктів, напрямок відбиття променя при дифузному відбиванні визначається випадковим чином [13].

Обчислення напрямку заломлених променів відбувається за допомогою закона Снеліуса. Формула виглядає так:

$$\frac{sin\theta_1}{sin\theta_2} = \frac{n_2}{n_1}, \tag{2.12}$$

де θ_1 – кут падіння;

θ_2 – кут відбиття;

n_1 та n_2 – коефіцієнти заломлення двох середовищ.

Основним недоліком методу трасування променів є те, що хоча точно враховуються різноманітні оптичні ефекти, традиційне трасування променів не завжди дає фотореалістичні зображення. Для досягнення справжнього фотореалізму необхідно використовувати більш точні апроксимації рівняння рендерингу, яке враховує всі можливі ефекти потоку світла. Однак це вимагає величезних обчислювальних ресурсів. Тому будь-який метод рендерингу є лише наближенням до цього рівняння, і іноді метод трасування променів не дає найбільш реалістичних результатів. Інші методи, такі як відображення або розподіл фотонів, частково використовують трасування променів, але дозволяють отримати значно реалістичніші зображення, враховуючи складні ефекти, такі як каустика. Приклад каустики зображено на рисунку 2.10.

Рисунок 2.10 – Каустика світла

Для забезпечення трасування променів у реальному часі важливими є кілька ключових технологій та оптимізацій. Однією з них є гібридні методи рендерингу. Трасування променів часто застосовується разом із традиційним растеризаційним рендерингом [14]. Такий підхід, відомий як гібридне рендерування, дозволяє використовувати трасування променів лише для специфічних візуальних ефектів, таких як точні відблиски, тіні або глобальне

освітлення, тоді як основна частина сцени обробляється за допомогою растеризації. Це дозволяє уникнути обробки всіх пікселів сцени через трасування променів, що значно зменшує обчислювальні витрати.

Адаптивне трасування променів передбачає зменшення кількості променів, що обробляються в менш важливих частинах сцени. Наприклад, області, що знаходяться поза фокусом, або ті, які мають менший вплив на загальне освітлення сцени, можуть бути оброблені з меншою кількістю променів. Цей підхід дозволяє ефективніше використовувати ресурси і досягати прийнятної якості зображення з меншою кількістю обчислень.

Через обмежену кількість променів, які можна обробити в реальному часі, зображення можуть містити велику кількість шуму. Для вирішення цієї проблеми використовується денойзінг – процес, який дозволяє згладжувати зображення, зменшуючи шум. Сучасні методи денойзінгу можуть базуватися на класичних алгоритмах фільтрації або на методах машинного навчання, які здатні ефективно відновлювати зображення, зменшуючи вплив шуму.

Для застосування цього методу слід послідовно виконати ряд дій. Етапи реалізації методу зображено на рисунку 2.11.

Етап 1: Перш за все, необхідно визначити, які задачі вирішуватимуться за допомогою трасування променів у реальному часі: створення реалістичного освітлення, тіней, відображень чи інших графічних ефектів. Важливо також оцінити апаратні можливості цільових пристроїв та встановити пріоритети між якістю графіки і продуктивністю.

Етап 2: Реалізується базовий механізм трасування променів. Генеруються первинні промені від камери, які перевіряють перетини з геометрією сцени. Впроваджуються базові моделі освітлення, такі як ламбертове відображення або фонгове затінення, для початкового рендерингу.

Етап 3: Щоб забезпечити ефективність роботи, реалізуються структури для оптимізації пошуку перетинів, , наприклад, BVH або KD-дерева. Ці

структури дозволяють швидко визначати об’єкти, з якими взаємодіють промені, зменшуючи загальний обсяг обчислень.

Етап 4: Щоб забезпечити високу швидкість роботи, зменшується кількість вторинних променів, які використовуються для обчислення відбиттів, заломлень і глобального освітлення. Впроваджуються методи вибіркового семплювання, які дозволяють обробляти лише найважливіші напрямки, зберігаючи баланс між продуктивністю та якістю.

Рисунок 2.11 – Етапи реалізації методу трасування променів

Етап 5: Через обмежену кількість вибірок може виникати шум у результатах рендерингу. Для його усунення додається алгоритм денойзингу, який згладжує зображення без значного впливу на продуктивність.

Етап 6: Після оптимізації додаються складніші ефекти, такі як багатократні відбиття, заломлення та глобальне освітлення. Впроваджується фізично коректна модель рендерингу, яка враховує властивості матеріалів та джерел світла.

3 РЕАЛІЗАЦІЯ МЕТОДІВ ТА АНАЛІЗ ЇХ ЕФЕКТИВНОСТІ

3.1 Опис середовища та інструментів для реалізації

Для реалізації методів оптимізації рендерингу в реальному часі було обрано Unreal Engine 5 – одну з найсучасніших платформ для роботи з тривимірною графікою. Unreal Engine 5 забезпечує широкий спектр можливостей для розробки високоякісного контенту, включаючи ігри, візуалізації, VR і AR-додатки [16]. Ключовими перевагами є підтримка інноваційних технологій оптимізації графіки, таких як Nanite і Lumen, які дозволяють розв'язувати задачі ефективного використання ресурсів і одночасно забезпечувати високий рівень деталізації зображень.

Nanite – це революційна система обробки геометрії, яка дає змогу використовувати мільярди полігонів у сценах без значного навантаження на графічний процесор [16]. Завдяки автоматичному створенню рівнів деталізації і адаптації рівня геометричної складності залежно від відстані до камери, також значно спрощує процес оптимізації сцен. Nanite усуває необхідність ручного налаштування LOD, що було однією з найтрудомістких задач у традиційному пайплайні.

Lumen – це глобальна система освітлення, яка забезпечує реалістичну взаємодію світла в динамічному середовищі. Використання Lumen дозволяє уникнути дорогих обчислень попереднього освітлення і працювати з реалістичним освітленням у реальному часі [16]. Ця технологія особливо важлива для VR і AR, де динамічність сцени є ключовою вимогою, а висока якість освітлення суттєво впливає на загальне візуальне враження.

Unreal Engine 5 підтримує широкий спектр графічних API, включаючи DirectX 12, Vulkan і Metal. Для цього дослідження був обраний DirectX 12 як основний API через його високу продуктивність і низькорівневий доступ до апаратного забезпечення. DirectX 12 забезпечує підтримку багатопоточних

обчислень, що дозволяє ефективно розподіляти завдання між ядрами CPU і максимально завантажувати GPU.

Однією з ключових переваг Unreal Engine 5 є його вбудовані інструменти для аналізу продуктивності. GPU Visualizer дозволяє детально вивчати графічний пайплайн, визначати вузькі місця в обробці кадрів і аналізувати час виконання окремих шейдерів. Крім цього, Unreal Engine підтримує використання сторонніх інструментів, таких як RenderDoc і NVIDIA Nsight Graphics, які забезпечують додаткові можливості для профілювання та дебагу графіки. Ці інструменти дають змогу детально аналізувати завантаження GPU, відстежувати використання пам'яті та ідентифікувати проблеми, які можуть впливати на частоту кадрів.

Сцени для тестування створювалися з використанням широкого спектра вбудованих можливостей Unreal Engine 5. Вони включали статичні та динамічні об'єкти, високополігональні моделі, складні шейдери та системи частинок. Для оцінки ефективності методів оптимізації було згенеровано кілька тестових середовищ, включаючи сцени для симуляцій віртуальної реальності та традиційні ігрові локації. Це дозволило оцінити продуктивність оптимізованих сцен у різних сценаріях і умовах.

Моніторинг продуктивності проводився за допомогою інструментів GPU-Z і Fraps, які дозволяють відстежувати частоту кадрів, завантаження GPU та використання пам’яті в реальному часі. Використання цих інструментів у поєднанні з вбудованими засобами Unreal Engine 5 дозволило отримати точні дані для аналізу ефективності впроваджених методів оптимізації.

Таким чином, Unreal Engine 5 у поєднанні з DirectX 12 та сучасними інструментами профілювання забезпечує повний набір засобів для реалізації та тестування методів підвищення продуктивності рендерингу. Ця платформа є не лише потужним інструментом для створення графічного контенту, а й дозволяє детально вивчати ефективність різних технологій оптимізації у реальних умовах.

3.2 Впровадження обраних методів оптимізації

3.2.1 LOD

У створеній сцені розміщено 640 столів, кожен із яких має 200 тисяч полігонів. Загальна кількість полігонів у сцені становить 128 мільйонів. Такий обсяг геометрії створює значне навантаження на графічний процесор, особливо для проєктів, які працюють у реальному часі. Це спричинило низьку продуктивність сцени, що фіксувалася на рівні приблизно 11 FPS (рисунок 3.1). Через це сцена ставала непридатною для плавного відтворення.

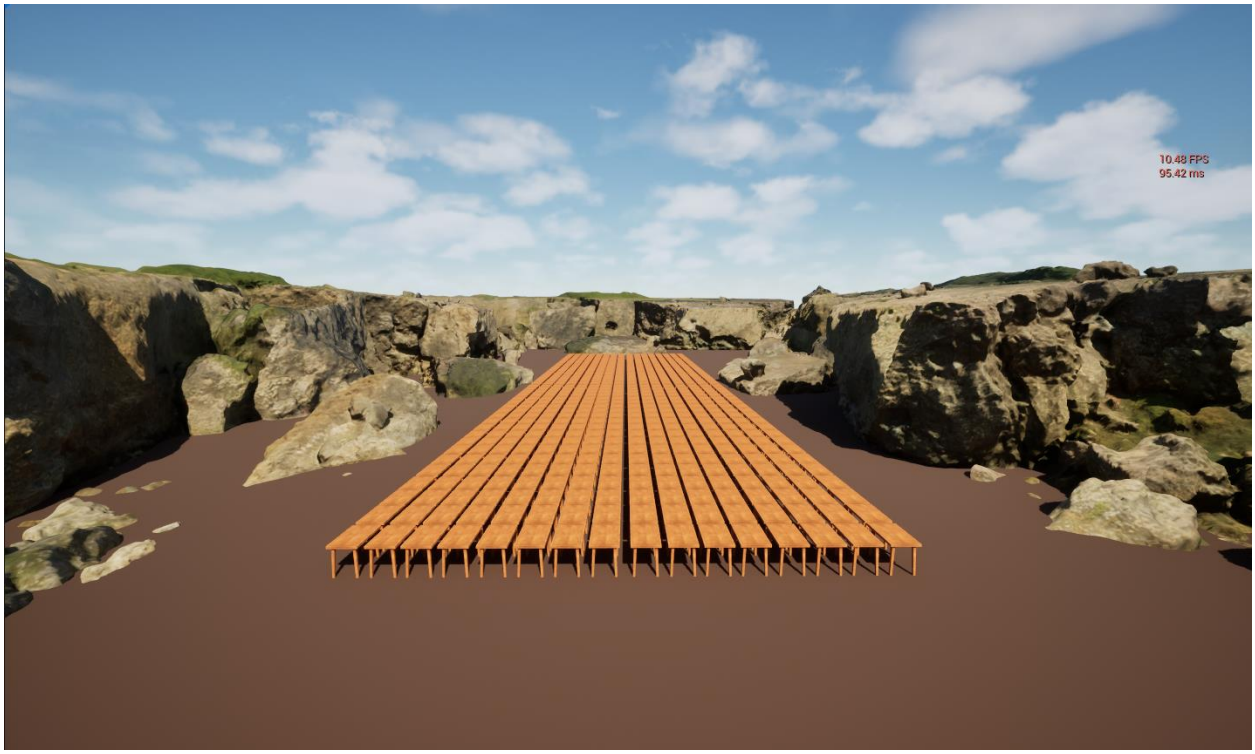

Рисунок 3.1 – Сцена до впровадження LOD

Головною проблемою продуктивності те, що рендеринг усіх об'єктів сцени виконувався з максимальною деталізацією, навіть для столів, розташованих далеко від камери. Це перевантажувало обчислювальні ресурси GPU та використовувало значну кількість відеопам'яті. Для

вирішення цих проблем було впроваджено систему LOD, яка дозволяє адаптувати рівень деталізації моделі залежно від відстані до камери.

У процесі оптимізації для кожного столу було створено чотири рівні деталізації. Максимальний рівень LOD 0, зберігав 200 тисяч полігонів і використовувався для об'єктів, які перебувають поблизу камери. На середній відстані застосовувалися моделі з LOD 1, які містили 100 тисяч полігонів. Для більш далеких об'єктів використовувався LOD 2 із 50 тисяч полігонів. Найвіддаленіші столи рендерилися за допомогою LOD 3, що містив лише 25 тисяч полігонів.

У Static Mesh Editor було налаштовано критерії переходу між рівнями деталізації. Наприклад, для столів, розташованих на відстані до 5 метрів від камери, застосовувався LOD 0. При збільшенні відстані до 50 метрів модель переключалася на LOD 1. Для об’єктів на відстані до 80 метрів використовувався LOD 2, а LOD 3 призначався для об’єктів, що знаходяться ще далі. Такий підхід дозволив мінімізувати кількість полігонів, які GPU повинен обробляти у кожному кадрі, без помітних втрат у якості.

Після впровадження LOD продуктивність сцени суттєво покращилася. До оптимізації середній FPS становив приблизно 11, оскільки GPU доводилося обробляти всі 128 мільйонів полігонів. Після оптимізації загальна кількість полігонів у середньому кадрі зменшилася до 10-15 мільйонів. Це підвищило FPS до 37 (рисунок 3.2), що відповідає приросту продуктивності на 236%. Таке покращення зробило сцену придатною для використання в режимі реального часу, водночас зберігши високу якість графіки для об’єктів, які перебувають у полі зору камери.

LOD дозволив ефективно зменшити навантаження на графічний процесор, оптимізувати використання відеопам’яті і забезпечити стабільну продуктивність сцени. Завдяки цьому вдалося досягти балансу між високою якістю графіки та плавністю рендерингу.

3.2.2 Видалення невидимих об’єктів

Методи Frustum Culling**,** Occlusion Culling та Backface Culling дали змогу оптимізувати рендеринг, залишаючи у видимому полі лише ті об'єкти, які роблять вагомий внесок у кінцеве зображення.

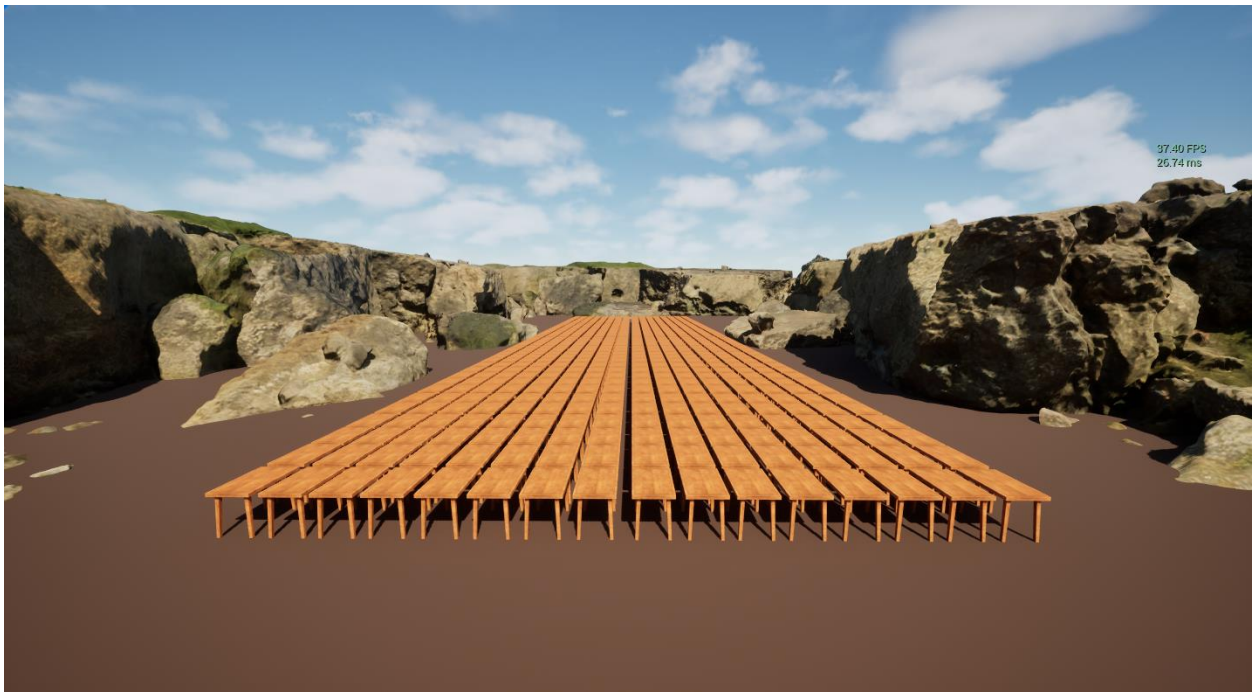

Рисунок 3.2 – Сцена після впровадження LOD

Frustum Culling реалізується в Unreal Engine 5 автоматично, але для досягнення максимального ефекту було виконано додаткову перевірку коректності його роботи. У сцені з великою кількістю столів система визначає, які об'єкти потрапляють у поле зору камери, і виключає з обробки ті, що залишаються поза межами фруструму. Для цього використовуються граничні обсяги об'єктів Bounding Volumes, що задаються як частина властивостей кожної статичної сітки. За допомогою інструмента Visualize Culling було перевірено, що об'єкти за межами фруструму не рендеряться, навіть якщо їхні полігони знаходяться на краю видимого поля. Така оптимізація дозволила значно зменшити навантаження на GPU.

Occlusion Culling дозволяє виключати з обробки об'єкти, які повністю закриті іншими. Наприклад, у нашій сцені, якщо стіл знаходиться за іншими столами та повністю перекритий ними, програма не витрачає ресурси на його обробку. Налаштування Occlusion Query було виконано через вкладку Rendering Settings, де параметри обчислення кількості видимих пікселів були встановлені так, щоб балансувати між продуктивністю й точністю.

Додатково був застосований Backface Culling, який дозволяє не рендерити полігони, зворотні до камери. Цей підхід особливо ефективний для складних моделей, таких як столи у сцені. У матеріалах кожного стола параметр Two-Sided Rendering було вимкнено, щоб графічний процесор ігнорував зворотні сторони полігонів. Завдяки цьому значно зменшилася кількість обчислень у сцені, оскільки більшість полігонів були орієнтовані в напрямку від камери.

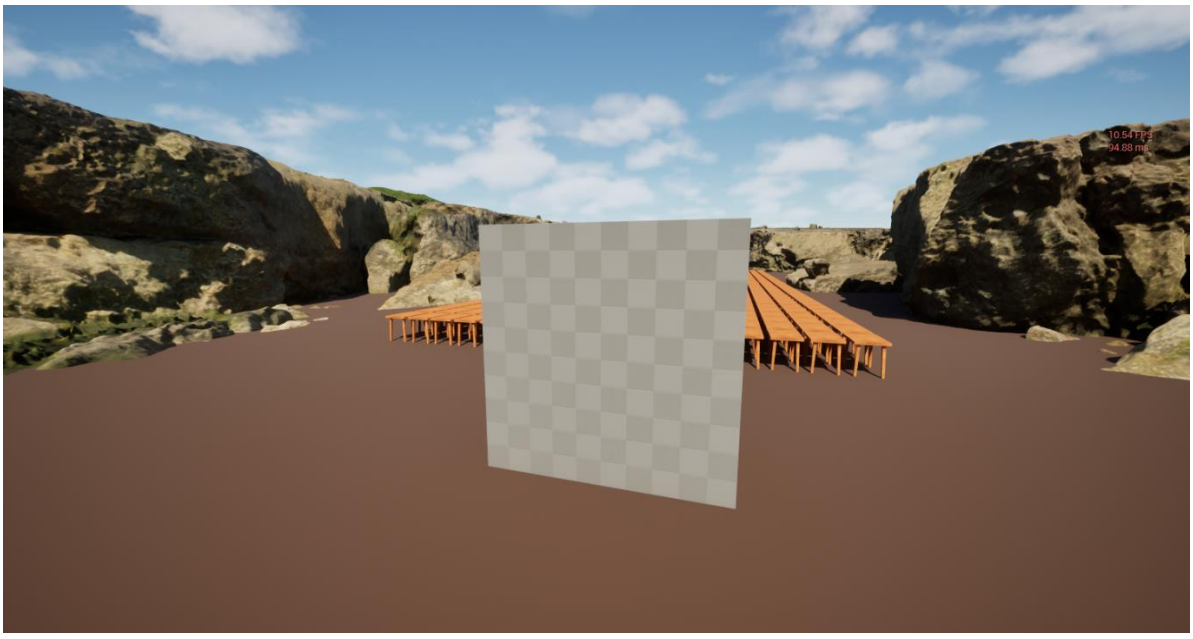

Рисунок 3.3 – Сцена до видалення невидимих об’єктів

Після впровадження цих методів продуктивність сцени суттєво покращилася. Загальна кількість полігонів, які оброблялися в середньому, зменшилася приблизно на 35–40%. Це забезпечило приріст FPS у межах 17-

20 кадрів залежно від ракурсу камери та густоти об'єктів у сцені. Наприклад, початкова продуктивність у сцені до впровадження Culling-методів становила близько 11 FPS, що фіксувалося у важких кадрах із великою кількістю геометрії. Після оптимізації середній FPS досяг 28 (рисунок 3.4), що продемонстровано на відповідних тестових скріншотах.

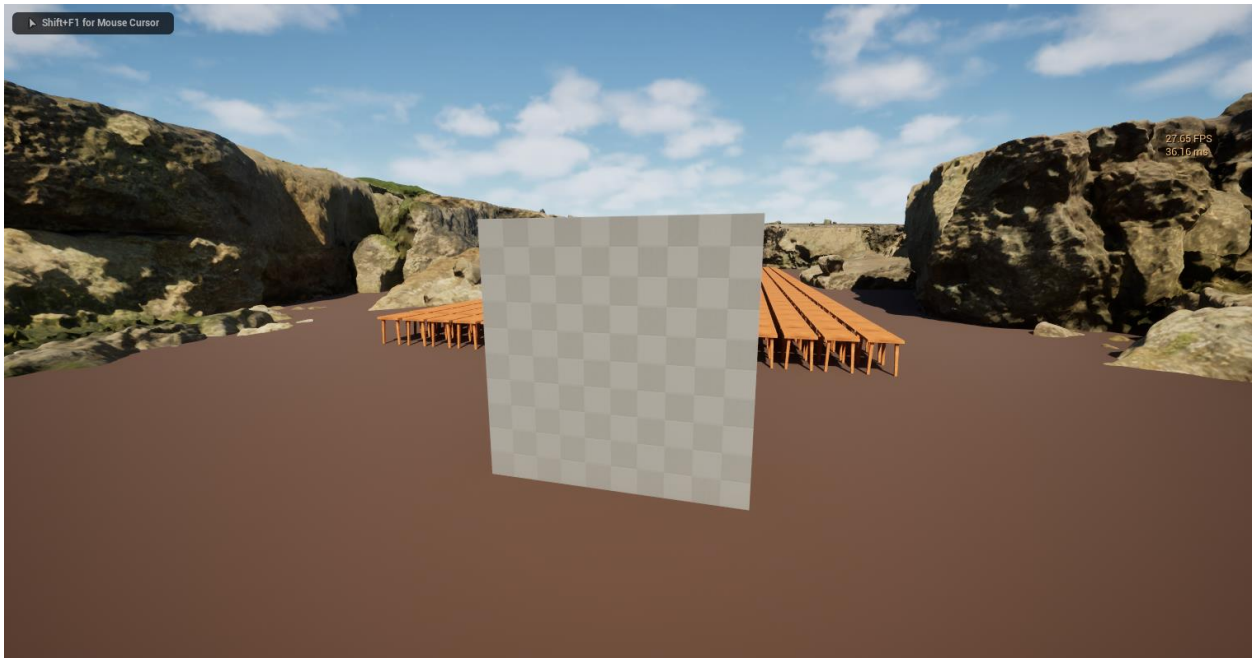

Рисунок 3.4 – Сцена після видалення невидимих об’єктів

3.2.3 Покращення продуктивності шейдерів

Було реалізовано кілька підходів до покращення продуктивності шейдерів, зокрема спрощення матеріалів, зменшення обсягу текстур і використання інстансів матеріалів.

Першим етапом стала перевірка складності матеріалів для об'єктів сцени. У випадку столів, які містили велику кількість полігонів, були виявлені складні матеріали з великою кількістю функцій і текстур високої роздільної здатності. Для оптимізації було видалено непотрібні вузли в графі шейдера, такі як складні налаштування блиску та зайві мапи нормалей. Крім того, для далеких об'єктів сцени, які не потребують високої деталізації,

використовувалися спрощені матеріали. Наприклад, для віддалених столів застосовувалися однотонні кольори з мінімальною кількістю текстур, що значно зменшило навантаження на GPU.

Другим кроком стало зменшення розміру текстур. Для об'єктів, які перебувають на середній та великій відстані від камери, текстури з роздільною здатністю 4096x4096 були замінені на текстури 2048x2048 та 1024x1024 (рисунок 3.5). Це було зроблено за допомогою налаштувань Texture Streaming, що дозволяє динамічно змінювати рівень деталізації текстур залежно від відстані до камери. Такий підхід дозволив значно зменшити обсяг відеопам'яті, необхідної для обробки текстур, без видимих втрат у якості зображення.

Рисунок 3.5 – До та після спрощення текстур

Крім того, було активно застосовано функцію Instanced Materials. У випадку, коли на сцені розташовано багато однакових об'єктів, використання

інстансів матеріалів дозволило зменшити кількість унікальних викликів до GPU. Це було досягнуто шляхом створення базового матеріалу та його інстансів із мінімальними змінами, такими як варіації кольорів. Таким чином, графічний процесор міг обробляти багато об'єктів одночасно.

Результати тестів показали, що після впровадження методу середній FPS сцени зріс із 11 до 42. Це дало змогу підвищити продуктивність на 282%, та зберегти високу якість візуалізації.

3.2.4 Трасування променів у реальному часі

Трасування променів у реальному часі є сучасним підходом до рендерингу, який забезпечує високу якість освітлення, відображень та тіней. Проте цей метод вимагає значних ресурсів графічного процесора, особливо у великих сценах.

Першим етапом стало налаштування параметрів трасування променів. Для цього в налаштуваннях проєкту було активовано опцію Ray Tracing, а також обрано лише ті ефекти, які мають найбільший вплив на якість зображення. Наприклад, у сцені було використано трасування променів для обчислення глобального освітлення Global Illumination та реалістичних відображень на поверхнях столів. При цьому такі ресурсоємні ефекти, як тіні від кожного об'єкта, були замінені на попередньо розраховані мапи тіней Shadow Maps.

Наступним кроком стало впровадження Hybrid Rendering, який поєднує традиційну растеризацію з трасуванням променів. У сцені всі об’єкти, які перебували на великій відстані від камери або мали низький рівень деталізації LOD 2 і LOD 3, оброблялися методом растеризації. Трасування променів застосовувалося лише для об'єктів у полі зору камери та в межах близького і середнього діапазону. Такий підхід дозволив зберегти якість відображень та освітлення для ключових елементів сцени, одночасно значно зменшивши кількість променів, які GPU обчислював у кожному кадрі.

Особлива увага приділялася матеріалам об'єктів із трасуванням променів. Наприклад, поверхні столів були налаштовані таким чином, щоб використовувати оптимізовані властивості відображень. Замість повного обчислення кожного відбитого променя було використано параметр Max Bounces, який обмежував кількість відбиттів до двох. Це дозволило уникнути надмірного збільшення часу обчислень, зберігаючи водночас реалістичність поверхонь.

Результати тестування всіх методів представлені на рисунок 3.6.

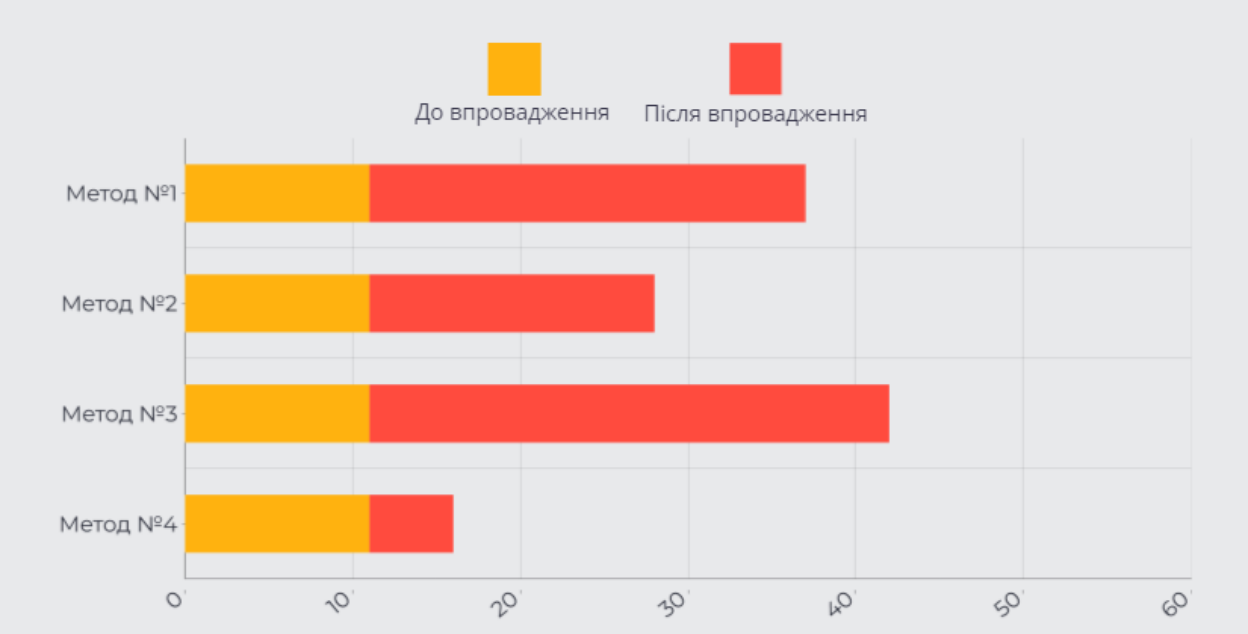

Рисунок 3.6 – Результати тестування всіх методів

Результати тестів показали, що впровадження трасування променів у реальному часі в оптимізованому вигляді покращило якість сцени без суттєвого зниження продуктивності. Після застосування Hybrid Rendering та інших методів оптимізації середній FPS зріс із 11 до 16.

ВИСНОВКИ

Проведено аналіз існуючих методів підвищення продуктивності рендерингу в реальному часі, розглянуто основні підходи, такі як використання рівнів деталізації, видалення невидимих об'єктів, оптимізація шейдерів і впровадження трасування променів. Виділено ключові завдання, що постають під час реалізації таких методів, та основні проблеми, пов’язані з їхнім впровадженням у великих сценах із високою геометричною складністю.

У результаті проведеної оптимізації сцени вдалося досягти значного покращення її продуктивності, що зробило можливим плавне відтворення в режимі реального часу без втрати якості графіки. Основним елементом, який сприяв цьому, стало впровадження системи LOD. Завдяки використанню різних рівнів деталізації для кожного об’єкта залежно від відстані до камери, загальну кількість полігонів у кадрі вдалося скоротити з 128 мільйонів до 10–15 мільйонів. Це дозволило підвищити середній FPS із 11 до 37, забезпечивши приріст продуктивності на 236%. Важливо, що зменшення кількості полігонів для об’єктів, які перебувають далеко від камери, не вплинуло на загальну якість візуалізації сцени.

Додатковим кроком до оптимізації стало видалення невидимих об’єктів за допомогою Frustum Culling, Occlusion Culling та Backface Culling. Автоматизована система відсікала об’єкти поза полем зору камери, а також ті, що повністю перекриті іншими об’єктами, знижуючи навантаження на GPU. Видалення зворотних полігонів також дозволило значно скоротити обчислення, оскільки значна частина полігонів була орієнтована у протилежний від камери бік. Завдяки цим методам обсяг геометрії, яка оброблялася в середньому кадрі, зменшився на 35–40%, що підвищило середній FPS до 28. Таке покращення зробило сцену більш ефективною навіть у складних ракурсах із великою кількістю об’єктів.

Складні матеріали об’єктів були оптимізовані шляхом видалення зайвих функцій та зменшення роздільної здатності текстур для віддалених об’єктів. Заміна текстур для об’єктів, які перебувають на середній і великій відстані, зменшила обсяг відеопам’яті, необхідної для їх обробки. У результаті середній FPS зріс до 42, забезпечивши приріст продуктивності на 282%, при цьому якість графіки залишилася на високому рівні.

Окрему увагу було приділено трасуванню променів у реальному часі. Цей метод забезпечує реалістичність освітлення, відображень та тіней, але є ресурсомістким. Для підвищення продуктивності було впроваджено гібридний рендеринг, де об’єкти на далекій відстані рендерилися традиційною растеризацією, а трасування променів застосовувалося лише до об’єктів у полі зору камери. Зменшення кількості відбиттів променів до двох також сприяло зниженню обчислювального навантаження. У підсумку середній FPS підвищився з 11 до 16, що дозволило досягти високої якості візуалізації без суттєвого зниження продуктивності.

Загалом, проведені заходи оптимізації значно підвищили ефективність сцени, забезпечивши плавне відтворення навіть у складних умовах. Поєднання системи LOD, методів Culling, спрощення шейдерів і текстур, а також оптимізованого трасування променів дозволило досягти балансу між високою якістю графіки та стабільною продуктивністю. Такий підхід демонструє, що комплексна оптимізація є ключем до створення реалістичних сцен, придатних для роботи в режимі реального часу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Пилипенко О.Р., Кравченко П.О. Методи підвищення продуктивності рендерингу в реальному часі для 3D-графіки. Проблеми інформатизації. Тези доповідей дванадцятої міжнародної НТК. – Баку: ІСУ АР; Харків: НТУ «ХПІ»; Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Бельсько-Бяла: УТіГН, 2024. – 21-22 листопада 2024. – Том 2. – С. 55.

2. Akenine-Möller T., Haines E., Hoffman N. Real-Time Rendering. Fourth Edition. Boca Raton: CRC Press, 2018.

3. Hughes J. F., Van Dam A., McGuire M. Computer Graphics: Principles and Practice. Third Edition. San Francisco: Addison-Wesley, 2021. 264 p.

4. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. Fourth Edition. San Francisco: Morgan Kaufmann, 2022. 120-131 p.

5. Glassner A. Ray Tracing in One Weekend Series. Expanded Edition. San Francisco: Morgan Kaufmann, 2020. 650 p.

6. Woo M., Neider J., Davis T. OpenGL Programming Guide: The Official Guide to Learning OpenGL. Ninth Edition. Boston: Addison-Wesley, 2022. 952 p.

7. Havran V., Bittner J. Efficient Ray Tracing Algorithms for Modern GPUs. Prague: Springer, 2023. 11-17 p.

8. Powell M. Mastering Unreal Engine 5: A Guide for Realistic Graphics and Gameplay. Birmingham: Packt Publishing, 2024. 152 p.

9. Parisi T. Learning Virtual Reality: Developing Immersive Experiences and Applications for Desktop, Web, and Mobile. Sebastopol: O'Reilly Media, 2019. 110-127 p.

10. Billinghurst M., Clark A., Lee G. Augmented Reality: A Practical Guide. Boca Raton: CRC Press, 2021. 274 p.

11. Guthe M., Klein R. Advanced Techniques for Level of Detail Rendering. Prague: Springer, 2020. 278 p.

12. Barkovska, O., Shulinus, O., Rosinskiy, D., Lebodkin, Y., & Serdechnyi, V. (2023, September). Research on Model Rendering Performance in Blender 3D Using Massively Parallel Systems. In 2023 IEEE East-West Design & Test Symposium (EWDTS) (pp. 1-5).

13. Shirley P. Ray Tracing in One Weekend. Amazon Digital Services LLC, 2020. 55 p.

14. Akenine-Möller T., Wenzel J., Haines E. Real-Time Ray Tracing. Boca Raton: CRC Press, 2019. 405 p.

15. Barkovska, O., Filippenko, I., Semenenko, I., Korniienko, V., & Sedlaček, P. (2023). Adaptation of FPGA architecture for accelerated image preprocessing. Radioelectronic and Computer Systems, 94-106 p.

16. Gregory J. Game Engine Architecture. Third Edition. Boca Raton: CRC Press, 2018. 240 p.