Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Оптимізація рендерингу 3д зображення на основі октодерева»

Виконав: студент групи К23-2М

Спеціальність 122 Комп'ютерні науки

Рябоволенко Е.А.
(прізвище та ініціали)

Керівник к.т.н., доц. Ульяновська Ю. В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місце роботи)

Доцент кафедри кібербезпеки та інфомармаційних технологій
(посада)

к.т.н., доц. Клим В.Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Рябоволенко Е.А. Оптимізація рендерингу 3D-зображень на основі октодерева

Кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2025.

Об'єктом дослідження є процес рендерингу тривимірних сцен у відеоіграх.

Предмет дослідження – методи та алгоритми оптимізації рендерингу 3D-зображень.

Метою роботи є розробка алгоритму оптимізації рендерингу 3D-зображень на основі октодерева для зниження обчислювальних витрат при збереженні високої якості візуалізації.

Дипломна робота присвячена розробці та дослідженню алгоритму оптимізації процесу рендерингу тривимірних зображень, який базується на використанні просторових структур поділу сцени – октодерева. У роботі проведено аналіз методів оптимізації рендерингу, таких як растеризація, трасування променів, Level of Detail та Occlusion Culling. Визначено ключові підходи до зниження обчислювального навантаження під час обробки складних тривимірних сцен. Запропонований алгоритм реалізовано у середовищі Unity із використанням C#. Проведено тестування програмного продукту, яке підтвердило ефективність зниження навантаження на графічний процесор.

Практична цінність роботи полягає у створенні алгоритму, який може бути використаний у сфері розробки відеоігор, систем віртуальної реальності, архітектурної візуалізації та інших інтерактивних 3D-додатків для забезпечення балансу між продуктивністю та якістю зображення.

Ключові слова: рендеринг, 3D-графіка, Occlusion Culling, Octree, Level of Detail, Unity, C#.

ABSTRACT

Ryabovolenko E.A. Optimization of 3D Image Rendering Based on Octree Diploma thesis (project) for obtaining a master's degree in speciality 122 "Computer Science." - University of Customs and Finance, Dnipro, 2025.

The object of the study is the process of rendering three-dimensional scenes in video games.

The subject of the study is methods and algorithms for optimizing 3D image rendering.

The aim of the work is to develop an algorithm for optimizing 3D image rendering based on an octree to reduce computational costs while maintaining high-quality visualization.

This thesis focuses on the development and investigation of a 3D image rendering optimization algorithm based on the use of spatial partitioning structures – the octree. The research includes an analysis of rendering optimization methods such as rasterization, ray tracing, Level of Detail, and Occlusion Culling. Key approaches to reducing computational load during complex 3D scene processing are identified. The proposed algorithm was implemented in the Unity environment using C#. The developed software product was tested, confirming its efficiency in reducing GPU load.

The practical significance of the work lies in the development of an algorithm that can be applied in the fields of video game development, virtual reality systems, architectural visualization, and other interactive 3D applications to ensure a balance between performance and image quality.

Keywords: rendering, 3D graphics, Occlusion Culling, Octree, Level of Detail, Unity, C#.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних умовах розвитку комп’ютерної графіки та тривимірного моделювання зростає потреба у створенні ефективних методів оптимізації рендерингу зображень. Складні 3D-сцени з високою деталізацією, що використовуються в ігровій індустрії, віртуальній реальності, архітектурній візуалізації та кінематографі, часто потребують значних обчислювальних ресурсів. Це створює виклики для досягнення балансу між якістю графіки та продуктивністю.

Одним із ключових підходів до оптимізації рендерингу є використання структур просторового поділу, зокрема октодерева (Octree). Цей метод дозволяє зменшити обсяг обчислень, шляхом поділу сцени на підобласті та обробки лише тих об'єктів, які безпосередньо впливають на кінцеве зображення [1].

Інноваційність дослідження базується на використанні оптимізованого алгоритму рендерингу на основі октодерева, що поєднує просторовий поділ сцени з адаптивним управлінням рівнем деталізації об'єктів (Level of Detail, LOD). Використання цього алгоритму дозволяє зменшити обчислювальне навантаження, зберігаючи при цьому високу якість зображення.

Мета роботи – розробка алгоритму оптимізації рендерингу 3D-зображень на основі октодерева для зниження обчислювальних витрат при збереженні високої якості візуалізації.

Методи дослідження – методи проектування, розробки програмного забезпечення, метод теорії інформації, обробка та аналіз інформації.

У відповідності до поставленої мети у кваліфікаційній роботі вирішувались наступні завдання:

1. Аналіз наукових джерел щодо алгоритмів рендерингу та методів оптимізації;
2. Розробка алгоритму з використанням структур просторового поділу;

3. Програмна реалізація алгоритму в середовищі рушія Unity;

4. Перевірити відповідність програмного продукту функціональним та нефункціональним вимогам.

Об'єкт дослідження – процес рендерингу тривимірних сцен у відеоіграх.

Предмет дослідження – методи та алгоритми рендеринг 3D-зображень та їх програмна реалізація

Практична цінність роботи полягає у розробці алгоритму, що може бути застосований у розробці комп’ютерних ігор, систем віртуальної реальності та інших інтерактивних 3D-додатків, де важливо забезпечити баланс між якістю зображення та продуктивністю.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Обсяг роботи становить 64 сторінок основного тексту, 45 рисунків та 1 таблиці.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАННЯ ДОСЛІДЖЕННЯ

1.1 Аналіз публікацій щодо рендирингу 3Д зображень

Рендеринг тривимірних зображень є фундаментальним процесом у сфері комп'ютерної графіки, який перетворює тривимірні моделі та сцени на двовимірні зображення, придатні для відображення на екрані або друку. Він включає обчислення і відображення геометрії, текстур, освітлення та інших параметрів сцени для створення реалістичного або стилізованого вигляду.

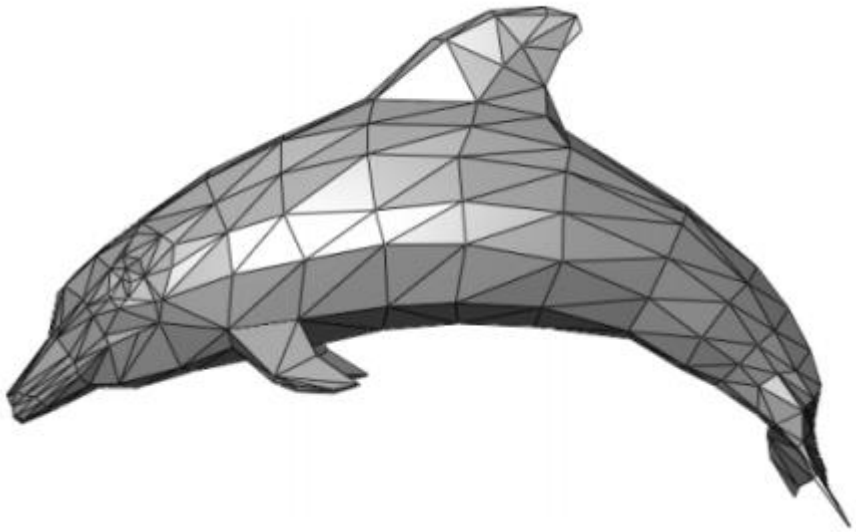


Рисунок 1.1 – Полігональне представлення 3Д об’єкту

Історичний розвиток рендерингу бере свій початок з перших комп'ютерів, коли дослідники прагнули відтворити реалістичні 3D-зображення на екрані. У 1960–1970-х роках були розроблені базові алгоритми рендерингу, такі як растеризація та прості моделі освітлення, які використовувалися для створення простих візуальних ефектів. З розвитком технологій та апаратного забезпечення з'явилися більш складні методи, зокрема трасування променів та глобальне освітлення, що дозволило досягти

високого рівня реалістичності зображень. Сьогодні рендеринг широко застосовується в кінематографі, відеоіграх, архітектурній візуалізації, віртуальній та доповненій реальності.

Тривимірна сцена складається з чотирьох основних компонентів: геометрії сцени, матеріалів і текстур, освітлення та параметрів камери. Кожен із цих елементів взаємодіє між собою, створюючи комплексне та реалістичне візуальне представлення віртуального світу [1].

Геометрія сцени є основою будь-якої тривимірної композиції. Вона визначає форму, розмір і розташування об'єктів у тривимірному просторі. Геометричне моделювання здійснюється за допомогою множини вершин, ребер і граней, які утворюють полігональні сітки. Найпоширенішим примітивом у полігональному моделюванні є трикутник, оскільки він гарантує планарність поверхні та спрощує процес обчислення. Проте для складних форм можуть використовуватися квадрати, багатокутники та криволінійні поверхні, зокрема сплайни та нерви (NURBS). Важливою складовою геометрії є також нормалі – вектори, перпендикулярні до поверхні граней, які використовуються для коректного розрахунку відбиття світла. Геометрія сцени визначає фізичну структуру віртуального простору та слугує базисом для подальших етапів рендерингу.

Матеріали та текстури надають об'єктам сцени їхні візуальні властивості, визначаючи, як поверхні взаємодіють зі світлом. Матеріали описують оптичні характеристики поверхні: колір, відбивну здатність, прозорість, показник заломлення, шорсткість та інші параметри. Такі моделі як Бліна-Фонга або Кука-Торренса дозволяють точно моделювати відбиття світла на основі фізичних принципів. Текстури ж є двовимірними зображеннями, які накладаються на поверхню тривимірних об'єктів з метою додання деталізації та реалістичності. За допомогою текстурного мапінгу можна відтворити дрібні деталі поверхні: нерівності, кольорові візерунки чи структурні особливості матеріалів. Техніки bump mapping та normal mapping дозволяють створювати ілюзію глибини та рельєфності без збільшення

геометричної складності моделі. Поєднання матеріалів і текстур забезпечує правдоподібне відтворення різноманітних поверхонь.

Освітлення визначає візуальну атмосферу сцени та сприйняття глядачем її компонентів. Воно моделює спосіб яким світло взаємодіє з об'єктами, створюючи ефекти світла та тіні, що додають глибини та реалістичності. У тривимірних сценах використовуються різні типи джерел світла:

- point lights – випромінюють світло рівномірно в усіх напрямках з однієї точки, подібно до лампочки. Вони створюють чіткі тіні та використовуються для локального освітлення об'єктів.
- directional lights – імітують паралельні промені, такі як сонячне світло. Вони освітлюють сцену рівномірно та використовуються для створення денних сцен.
- spotlights – випромінюють світло в конічній формі, дозволяючи освітлювати певні області сцени та створювати драматичні ефекти.
- area lights випромінюють світло з певної поверхні, що сприяє створенню м'яких тіней і реалістичному розсіюванню світла.

Камера віртуальної сцени визначає точку зору з якої глядач спостерігає за подіями. Вона моделює властивості реальної фотокамери чи людського ока, включаючи положення, орієнтацію, фокусну відстань та параметри проєкції. Перспективна проєкція використовується для створення ефекту глибини, де об'єкти, які знаходяться далі від камери, здаються меншими. Це відповідає природному сприйняттю простору людиною. Ортографічна проєкція, навпаки, зберігає реальні розміри об'єктів незалежно від їхнього розташування щодо камери та використовується для технічних і архітектурних візуалізацій, де важлива точність розмірів без спотворень перспективи. Параметри камери, такі як кут огляду (Field of View), глибина різкості та діафрагма, впливають на композицію кадру, фокусування на певних об'єктах та загальну естетику зображення [5].

Перед відображенням на екрані графічні дані проходять кілька етапів у так званому конвеєрі графічного рендерингу. Цей конвеєр зазвичай включає

обробку вершин, збірку примітивів, геометричну обробку, обрізання, відбір, растеризацію та обробку фрагментів.

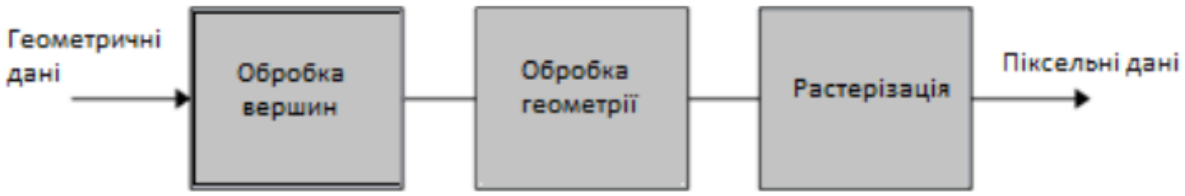


Рисунок 1.2 – Конвеєр рендирингу

1) Обробка вершин на цьому етапі застосовуються вершинні шейдери, які виконують ключові операції, зокрема перетворення координат вершин у просторі та обчислення освітлення сцени. Вихідними даними цього етапу є набір оброблених вершин.

2) Примітивна збірка цей етап передбачає групування оброблених вершин відповідно до інформації про їхні зв’язки, з метою формування графічних примітивів, таких як багатокутники, лінії або точки.

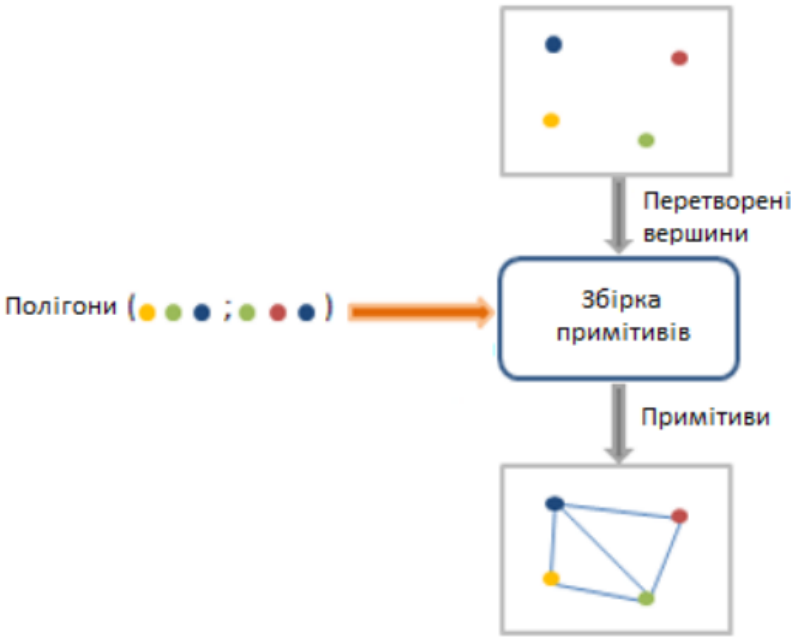


Рисунок 1.3 – Збірка примітивів

3) Геометрична обробка виконується після збірки примітивів та передує відсіканню невидимих об'єктів. Вона приймає примітиви з попереднього етапу для подальших обчислень. На відміну від інших стадій, цей етап здатний генерувати додаткові примітиви на основі вже існуючих, що дозволяє деталізувати сцену.

4) Відсікання та відбір. Даний етап спрямований на усунення примітивів, які є невидимими або виходять за межі області перегляду сцени. Це дозволяє оптимізувати подальші розрахунки, зменшуючи кількість об'єктів для обробки.

5) Растеризація передбачає перетворення примітивів у множину фрагментів, які відповідають окремим пікселям екрану. Ці фрагменти згодом передаються на етап обробки фрагментів для подальших перевірок і обчислень.

6) Обробка фрагментів. На даному етапі кожен фрагмент, отриманий на стадії растеризації, проходить серію тестів, після успішного проходження цих перевірок фрагмент може бути перетворений на піксель, який відображається на екрані.

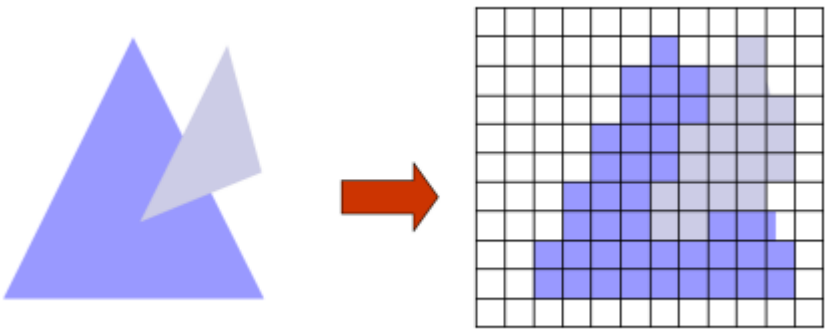


Рисунок 1.4 – Перетворення примітивів на фрагменти

1.2 Аналіз методів рендирингу 3Д зображень

Рендеринг тривимірних зображень є складним процесом, який базується на використанні різноманітних алгоритмів та методів. Серед багатьох доступних підходів розглянемо найпопулярніші.

Програмний та апаратний рендеринг є двома основними підходами до обробки графіки в комп'ютерних системах. Кожен із цих методів має свої переваги та недоліки залежно від складності анімації, продуктивності обладнання та вимог до якості візуалізації.

Апаратний рендеринг використовує можливості графічного процесора (GPU), який спеціалізований на паралельних обчисленнях, що дозволяє значно збільшити швидкість обробки складних графічних сцен [6]. Основні характеристики:

- GPU може обробляти мільйони пікселів одночасно завдяки паралельній архітектурі.
- Апаратний рендеринг використовується у відеоіграх, анімації, симуляціях та інших графічно інтенсивних завданнях.
- Потребує наявності сучасних відеокарт: NVIDIA, AMD або Intel.

Програмний рендеринг покладається на центральний процесор (CPU) для обробки всіх графічних розрахунків. Цей підхід зазвичай використовується у простих або менш вимогливих додатках, де продуктивність графічного процесора (GPU) не є критичною. Основні характеристики:

- Обмежена продуктивність CPU
- Гнучкість у розробці
- Відсутність залежності від спеціалізованого обладнання

До програмних методів рендерингу відносять:

1. Растеризація

Растеризація є одним із найстаріших та найпоширеніших методів рендерингу, широко застосовуваним у реальному часі, зокрема в

комп'ютерних іграх та інтерактивних графічних додатках. Основний принцип растеризації полягає у перетворенні тривимірних геометричних примітивів (зазвичай трикутників) у двовимірні пікселі на екрані через процес проєкції.

У процесі растеризації тривимірні координати об'єктів перетворюються у координати екрану за допомогою матриць моделі, виду та проєкції. Після цього відбувається відсікання примітивів, що знаходяться поза полем зору камери. Кожен трикутник, що залишився, розбивається на пікселі, значення яких визначаються шляхом інтерполяції атрибутів вершин (кольору, текстурних координат, нормалей). Фрагментні шейдери використовуються для обчислення остаточного кольору пікселя з урахуванням освітлення та матеріалів.

Перевагою растеризації є висока швидкість рендерингу, що робить її ідеальною для застосувань, де потрібна висока частота кадрів. Вона ефективно використовує апаратні можливості сучасних графічних процесорів (GPU). Однак растеризація має обмеження у відтворенні складних оптичних ефектів, таких як реалістичні відбиття, заломлення та глобальне освітлення, без використання додаткових технік.

2. Ray Tracing

Ray Tracing є методом рендерингу, що базується на моделюванні шляху світлових променів у зворотному напрямку — від камери до джерел світла. Цей метод дозволяє природно відтворювати складні оптичні ефекти, такі як відбиття, заломлення, тіні та каустики [7].

Процес трасування променів складається з наступних етапів:

1) Генерація первинних променів: Від кожного пікселя на екрані випускається промінь у тривимірний простір.

2) Перетин з геометрією: Визначається перша точка перетину променя з об'єктами сцени.

3) Обчислення локального освітлення: Визначається колір у точці перетину з урахуванням матеріалу поверхні та джерел світла.

4) Генерація вторинних променів: Якщо матеріал має відбивні або заломлюючі властивості, випускаються відповідні промені для моделювання цих ефектів.

5) Тіньові промені: Додаткові промені, що визначають, чи точка освітлюється прямим світлом або знаходиться в тіні.

Основною перевагою трасування променів є його здатність точно моделювати взаємодію світла з матеріалами, забезпечуючи високий рівень реалістичності зображення. Проте цей метод є обчислювально інтенсивним, що довгий час обмежувало його використання в реальному часі. З появою сучасних графічних процесорів із підтримкою апаратного трасування променів (наприклад, технології RTX від NVIDIA), цей метод отримав ширше застосування у інтерактивних додатках та іграх.

3. Path Tracing

Path Tracing є розширенням методу трасування променів, яке враховує глобальне освітлення сцени шляхом статистичного зразкування багатьох можливих шляхів світла. Цей метод використовує алгоритми Монте-Карло для моделювання непрямих взаємодій світла з поверхнями [8].

У патовій трасуванні для кожного пікселя випускається велика кількість променів, кожен з яких може:

- Відбиватися: Випромінюватися в новому напрямку згідно з функцією BRDF матеріалу.
- Заломлюватися: Проходити крізь прозорі матеріали з урахуванням показника заломлення.
- Поглинатися: Відповідно до властивостей матеріалів, частина світла може поглинатися.

Цей процес повторюється рекурсивно, доки внесок променя стає незначним або досягається максимальна глибина трасування. Підсумовуючи внески всіх променів, отримують кінцевий колір пікселя. Патова трасування дозволяє природно моделювати ефекти, такі як м'які тіні, глобальне освітлення, кольорові відблиски та каустики.

Недоліком методу є високий рівень шуму в зображеннях при невеликій кількості зразків, що вимагає значних обчислювальних ресурсів для досягнення чистоти зображення. Тому патова трасування переважно використовується в офлайн-рендерингу для створення фотореалістичних зображень у кінематографі, рекламі та архітектурній візуалізації.

4. Radiosity

Radiosity є методом, спрямованим на обчислення енергетичного балансу між поверхнями в сцені з діффузними відбивними властивостями. Він базується на фізичних принципах обміну тепловою енергією і використовується для моделювання непрямого діффузного освітлення.

Процес радіосіті включає:

- Дискретизацію сцени: Розбиття поверхонь на малі елементарні площини (патчі).
- Обчислення факторів форми: Визначення коефіцієнтів взаємного впливу між патчами залежно від їхнього розташування та орієнтації.
- Складання системи рівнянь: Для кожного патча встановлюється рівняння енергетичного балансу з урахуванням випромінюваної та поглинаємої енергії.
- Розв'язання системи: Обчислюються радіаційні значення для всіх патчів, що дозволяє визначити освітленість кожної ділянки поверхні.

Радіосіті забезпечує високоточне моделювання діффузного непрямого освітлення, створюючи реалістичні м'які тіні та ефекти перенесення кольору між поверхнями. Метод добре підходить для сцен з переважно діффузними матеріалами та статичним освітленням.

До недоліків радіосіті відносяться:

- Висока обчислювальна складність: Особливо для сцен з великою кількістю патчів.
- Обмеження на типи матеріалів: Метод не враховує дзеркальні відбиття та прозорість.

- Непридатність для динамічних сцен: Будь-які зміни в геометрії або освітленні вимагають повного перерахунку [9].

5. Картографування фотонів

Картографування фотонів (англ. Photon Mapping) є двоетапним методом глобального освітлення, який поєднує ідеї трасування променів та зберігання інформації про світлові потоки в спеціальних структурах даних.

Етапи методу:

- Фотонний пас: Від джерел світла випускаються фотони, які випадковим чином взаємодіють з поверхнями сцени, відбиваючись та заломлюючись згідно з властивостями матеріалів. У точках взаємодії фотони зберігаються в фотонній карті з інформацією про їх енергію та напрямок.
- Рендеринговий пас: Під час традиційного трасування променів на кожному перетині променя з поверхнею виконується пошук найближчих фотонів у фотонній карті. Використовуючи ці дані, обчислюється внесок непрямого освітлення в колір точки.

Перевагою картографування фотонів є його здатність ефективно моделювати складні світлові ефекти, такі як каустики, непряме освітлення та субповерхневе розсіювання. Метод є гнучким і може поєднуватися з іншими техніками рендерингу.

Недоліки включають:

- Витрати пам'яті: Зберігання великої кількості фотонів може вимагати значних ресурсів.
- Складність налаштування: Параметри емісії фотонів та розміру пошуку впливають на якість та швидкість рендерингу.
- Шум та артефакти: Можуть виникати при недостатній кількості фотонів або неправильному налаштуванні.

6. Алгоритм Occlusion culling

цей алгоритм оптимізації рендерингу зображень за рахунок усування відображення графічних об’єктів, які лежать поза межами видимості гравця або знаходяться за іншими об’єктами (рис. 1.2).

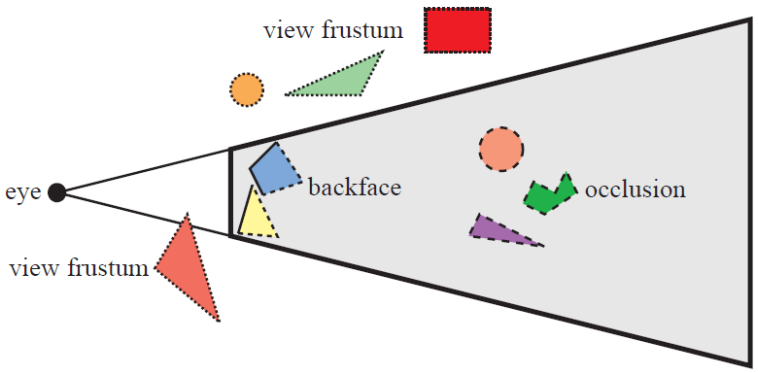


Рисунок 1.5 – Алгоритм Occlusion Culling

Для цілей рендерингу occlusion culling слід порівнювати з еталонним рішенням, яке полягає в простому рендерингу всього що знаходиться в зоні видимості (view frustum). В деяких випадках, наприклад у спортивних іграх, де застосування оклюзії не виправдовує себе оскільки більша частина сцени є видимою.

Натомість у складних 3D-світах з великою кількістю деталей і високою візуальною складністю система оклюзії стає надзвичайно корисною, оскільки дозволяє зменшити кількість відрендерених об'єктів які не видно користувачеві. Це значно покращує продуктивність рендерингу.

Алгоритм базується на принципах роботи Z-буферизації – це технологія, яка допомагає комп’ютеру правильно відображати об’єкти у тривимірному просторі на екрані. Для прикладу потрібно уявити, що персонаж дивиться через вікно, то об’єкти ближче перекривають ті що далі. Так само працює Z-буферизація – для кожної точки на екрані комп’ютер запам’ятовує, наскільки близько знаходиться об’єкт (його "глибину"). Якщо інший об’єкт намагається з'явитися в цій точці, комп’ютер перевіряє чи він ближче. Якщо так – його малює, а якщо ні – ігнорує. Це дозволяє відображати сцени правильно та уникати накладення об’єктів які не повинні бути видимими. Зазвичай така перевірка виконується швидко та допомагає економити ресурси комп’ютера, особливо якщо ближні об’єкти обробляються першими [3].

7. Алгоритм Octree – використовуються для коректного представлення твердотільних об'єктів у 3D-середовищі і є основою для багатьох систем моделювання та рендерингу. Основна мета алгоритму – зменшити кількість порівнянь, необхідних для визначення того, які поверхні в сцені потрібно обробити виявлення зіткнень, визначення видимості об’єктів [2].

8. Алгоритм Level of Detail адаптує рівень деталізації об'єкта залежно від його відстані до камери. Чим далі об'єкт від камери, тим простіша версія його моделі використовується для рендерингу.

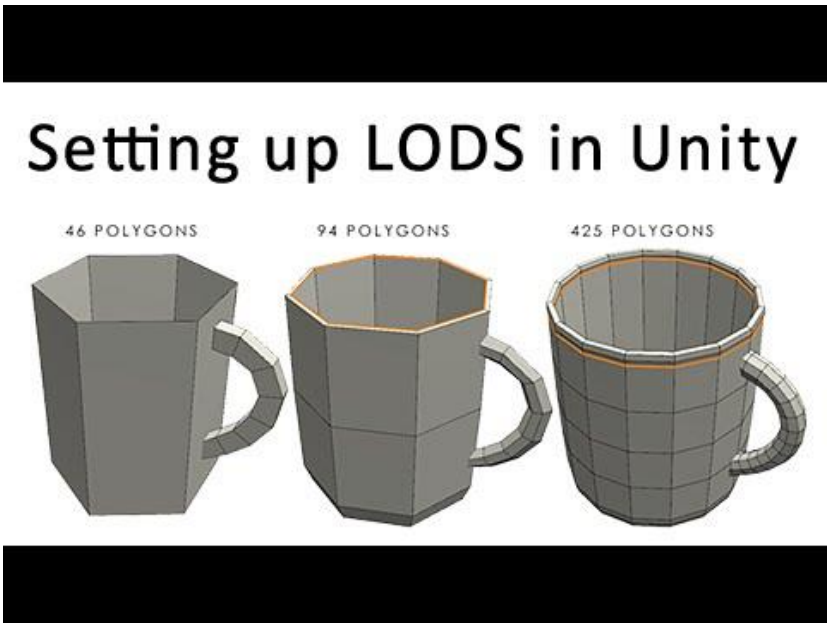


Рисунок 1.6 – зображення роботи алгоритму LOD

Для роботи даного алгоритму потрібно створити моделі різної якість деталізації.

1.3 Висновок до першого розділу

У даному розділі було проведено дослідження методів та алгоритмів рендерингу тривимірних зображень, включаючи растеризацію, трасування променів, path tracing, radiosity та photon mapping. Проаналізовано ключові підходи до оптимізації рендерингу, зокрема алгоритми Occlusion Culling, Octree та Level of Detail (LOD). Було розглянуто принципи просторового поділу сцени, зниження рівня деталізації та управління обчислювальними ресурсами для покращення продуктивності без втрати візуальної якості.

Особливу увагу приділено алгоритму октодерева, який дозволяє зменшити обсяг обчислень шляхом розбиття сцени на підобласті та обробки лише релевантних об'єктів. Розглянуто також динамічне управління рівнем деталізації, що забезпечує адаптивну зміну кількості полігонів у залежності від відстані об'єкта до камери.

Метою дослідження є розробка алгоритму оптимізації рендерингу 3D-зображень на основі октодерева для зниження обчислювальних витрат при збереженні високої якості візуалізації.

Для досягнення поставленої мети було визначено та виконано такі завдання дослідження:

1. Проведено аналіз наукових джерел щодо алгоритмів рендерингу та методів оптимізації.
2. Розроблено алгоритм оптимізації рендерингу з використанням структури октодерева.
3. Виконано програмну реалізацію алгоритму в середовищі рушія Unity.
4. Проведено тестування програмного продукту для перевірки його відповідності функціональним і нефункціональним вимогам.

Вхідними даними для алгоритму є 3D-моделі, які поділяються на підобласті за допомогою структури октодерева, а рівень деталізації об'єктів адаптується залежно від їх віддаленості від камери.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ АЛГОРИТМУ ОПТИМІЗАЦІЇ РЕНДИРУНГУ

2.1 Вибір програмних засобів

У сучасних 3D-сценах використовується велика кількість об'єктів із високою деталізацією, що може значно впливати на продуктивність рендерингу, особливо у реальному часі. Оптимізація 3D-сцен є важливим етапом для досягнення балансу між якістю візуалізації та швидкодією програмного забезпечення. Одним із ключових підходів до цього є зменшення обчислювальних ресурсів шляхом відсіювання невидимих об'єктів, зниження рівня деталізації для віддалених об'єктів та оптимізація структури сцени для прискорення обробки. Такі методи використовуються в ігрових рушіях, системах віртуальної реальності та архітектурних візуалізаціях для забезпечення стабільної продуктивності на різних пристроях.

Розроблено декілька підходів для оптимізації 3D-сцен, кожен із яких має свої переваги та недоліки в залежності від складності сцени та специфіки застосування.

- Level of Detail (LOD) – алгоритм зменшує кількість полігонів об'єкта залежно від його відстані до камери. Чим далі об'єкт, тим менш деталізованою є його модель, що суттєво знижує витрати на рендеринг [9].
- Occlusion Culling (Відсікання об'єктів за перешкодами) – об'єкти, які повністю перекриваються іншими об'єктами, не рендеряться. Цей метод часто використовується у поєднанні з октодерева для більшої ефективності.

Вибір програмного забезпечення відіграє ключову роль, оскільки різні інструменти мають різний рівень підтримки структур просторового поділу сцени, оптимізації та автоматизації цього процесу.

ZBrush – це професійний програмний продукт, розроблений компанією Pixologic, призначений для цифрової скульптури та текстурування 3D-

моделей. Програма орієнтована на створення високодеталізованих тривимірних об'єктів, що робить її одним із провідних інструментів у сфері моделювання персонажів для кіно, відеоігор та анімації.

Основні особливості:

1) Цифрова скульптура (Digital Sculpting)

ZBrush дозволяє створювати складні органічні форми завдяки інструментам цифрового "ліплення". Процес аналогічний роботі зі справжньою глиною, де користувач може додавати або зменшувати об’єм моделі, працювати з поверхнею та створювати складні деталі.

2) Потужна деталізація та DynaMesh

Інструмент DynaMesh дозволяє працювати з моделями високої деталізації, автоматично перерозподіляючи сітку під час модифікації. Це зручно для скульптингу персонажів та об'єктів із великою кількістю дрібних деталей.

3) ZRemesher

ZBrush містить автоматизований інструмент для ретопології моделей – ZRemesher, який дозволяє зменшити кількість полігонів моделі, зберігаючи її основні деталі. Це особливо важливо при підготовці моделей для анімації та інтеграції в ігрові рушії.

4) Polypaint

Інструмент Polypaint дає змогу малювати текстури безпосередньо на 3D-моделі, використовуючи саму геометрію як основу для збереження кольору. Це підходить для створення детальних текстур, оскільки кожен полігон може містити колірну інформацію.

5) Subdivision Levels

Програма підтримує роботу з декількома рівнями підрозділів, що дозволяє збільшувати або зменшувати кількість полігонів у процесі редагування, зберігаючи загальні форми та деталі.

6) Імпорт та експорт даних

ZBrush підтримує формати OBJ, FBX, STL, що дозволяє легко переносити моделі в інші програми для анімації або рендерингу.

7) Інтеграція з іншими інструментами

Програма добре інтегрується з іншими графічними пакетами, такими як Maya, 3ds Max, Blender через GoZ – інструмент для прямого обміну моделями між додатками.

Переваги:

- Висока деталізація моделей.
- Автоматизовані інструменти ретопології.
- Інтуїтивний інтерфейс для цифрової скульптури.
- Можливість створення текстур без UV-розгортки.

Недоліки:

- Відсутність можливостей для анімації та складних симуляцій.
- Не підтримує роботу зі сценами з великою кількістю об'єктів.
- Високі вимоги до продуктивності при роботі з моделями з великою кількістю полігонів [11].

ZBrush є ідеальним інструментом для художників, які працюють над створенням високодеталізованих моделей персонажів, скульптур і концепт-артів.

Blender – це потужний багатофункціональний пакет із відкритим вихідним кодом, призначений для створення тривимірної графіки, анімації, рендерингу, текстурування та відеомонтажу. Завдяки своїй універсальності, безкоштовності та кросплатформності Blender став одним із провідних інструментів у сфері комп'ютерної графіки, який активно використовується як для художніх, так і для технічних проєктів.

Програма забезпечує широкий набір інструментів для полігонального, процедурного та скульптурного моделювання. Вона підтримує методи ручного редагування геометрії, включаючи екструзію, різноманітні модифікатори (Subdivision Surface, Decimate), а також цифрову скульптуру, яка дозволяє створювати високополігональні органічні моделі. Функціонал

скульптингу підтримує різні пензлі для роботи з деталізацією поверхонь, а модифікатор Decimate допомагає знижувати кількість полігонів без втрати якості візуалізації.

Blender включає в себе потужний набір інструментів для анімації. Серед основних функцій анімації: система ключових кадрів (Keyframes), обернена кінематика (Inverse Kinematics, IK), нелінійний редактор анімації (NLA Editor) та можливість анімації через Shape Keys (анімовані деформації об'єкта). Це робить Blender зручним для створення як простих анімацій, так і складних кінематографічних сцен із великими персонажами та об'єктами.

Програма пропонує два основні рушії для рендерингу: Cycles та Eevee. Cycles – це фізично коректний рушій із трасуванням променів, який дозволяє отримати фотореалістичний результат із підтримкою глобального освітлення, тіней та реалістичних відображень. Eevee, у свою чергу, є рушієм реального часу, оптимізованим для швидкого рендерингу, ідеально підходить для створення попередніх переглядів або інтерактивних презентацій. Обидва рушії підтримують ефекти, такі як Ambient Occlusion, Bloom, Global Illumination та Volumetrics.

Blender також має вбудовану систему для текстурування та створення матеріалів. Вузловий редактор (Node Editor) дозволяє створювати як процедурні, так і стандартні матеріали з підтримкою фізично коректного рендерингу (PBR). Програма надає повний контроль над UV-розгорткою моделей, що дозволяє ефективно накладати текстури. Підтримуються всі основні формати текстур, включаючи PNG, JPEG, EXR та HDR.

Для візуальних ефектів (VFX) Blender включає модулі Motion Tracking для відстеження руху камери та об'єктів у відео, а також Compositor, який дозволяє редагувати відеокадри, накладати ефекти та створювати композитні сцени без необхідності використання сторонніх програм. Це робить Blender потужним інструментом для інтеграції 3D-об'єктів у відеоматеріали [12].

Blender також пропонує базовий відеоредактор (VSE – Video Sequence Editor) для обробки та монтажу відеоматеріалів. Він підтримує накладення

ефектів, вирізання сцен, додавання аудіодоріжок і анімації. Хоча VSE не конкурує з професійними відеоредакторами, він цілком підходить для базових завдань з монтажу.

Однією з ключових особливостей Blender є його підтримка скриптування мовою Python. Це дозволяє автоматизувати робочі процеси, створювати власні модифікатори, розширення та інструменти для оптимізації сцен. Завдяки відкритому вихідному коду та гнучкому API, Blender є одним із найкращих рішень для розробників, які хочуть створювати власні інструменти для 3D-графіки.

Переваги Blender:

- Відкритий вихідний код, повністю безкоштовний.
- Великий набір інструментів для моделювання, текстурування, анімації та рендерингу.
- Вбудовані рушії рендерингу: Cycles (фотореалістичний) та Eevee (реального часу).
- Підтримка скульптингу та процедурного моделювання.
- Активна спільнота користувачів та наявність великої кількості розширень.

Недоліки Blender:

- Відносно складний для початківців через велику кількість функцій.
- Менш ефективний для великих сцен із великою кількістю об'єктів.
- Обмежені можливості для інтеграції зі складними CAD-системами.

Blender є універсальним і потужним інструментом для створення 3D-моделей, анімації та рендерингу, який чудово підходить для художників, дизайнерів, архітекторів та незалежних розробників ігор. Завдяки широким можливостям і відкритому вихідному коду, Blender можна використовувати як для художніх проєктів, так і для технічних завдань, таких як анімація, архітектурна візуалізація та створення концепт-артів.

Autodesk 3ds Max – це професійне програмне забезпечення для створення тривимірних моделей, анімації та візуалізації, розроблене компанією Autodesk. Воно широко використовується у сферах архітектурної візуалізації, кінематографу, дизайну інтер'єрів та відеоігор завдяки потужним інструментам для моделювання, текстурування, рендерингу та анімації.

3ds Max пропонує гнучкі можливості для полігонального, сплайнового та NURBS-моделювання. Полігональне моделювання базується на редагуванні сітки через Editable Poly та Editable Mesh, що дозволяє створювати складні об'єкти з високим рівнем деталізації. Для роботи з кривими об'єктами використовується Spline-моделювання, де об'єкти створюються на основі кривих ліній. Технологія NURBS (Non-Uniform Rational B-Splines) забезпечує створення складних геометрій із гладкими поверхнями, що особливо актуально для індустріального дизайну.

Програма підтримує різноманітні техніки оптимізації моделей. Вбудований інструмент ProOptimizer дозволяє автоматично знижувати кількість полігонів у моделі, зберігаючи основні деталі. Це особливо корисно при підготовці моделей для анімації або інтеграції у гейм-рушії. Для роботи з моделями великої складності є підтримка багаторівневого підрозділення (Subdivision Surface).

3ds Max надає потужний набір інструментів для анімації. Keyframe Animation дозволяє створювати ключові кадри для об'єктів, тоді як Biped і CAT (Character Animation Toolkit) призначені для створення анімацій персонажів. Інструменти анімації також включають обернену кінематику (IK Solver), систему деформації об'єктів через Morph Targets, а також редактор анімації Motion Mixer для складних рухів [13].

Окремо варто відзначити систему частинок Particle Flow, яка дозволяє створювати симуляції вибухів, диму, вогню та рідин. Для реалістичних фізичних симуляцій використовується модуль MassFX, який підтримує динаміку твердих і м'яких тіл, а також симуляції тканин і рідин.

3ds Max оснащений кількома потужними рендер-рушіями для створення фотореалістичних зображень. Вбудований рендер Arnold Renderer забезпечує високоякісне трасування променів із підтримкою глобального освітлення, реалістичних відображень та складних матеріалів. Додатково підтримуються популярні сторонні рендер-рушії, такі як V-Ray та Corona Renderer, які часто використовуються в архітектурній візуалізації.

Редактор матеріалів Slate Material Editor у 3ds Max дозволяє створювати як прості, так і складні багатошарові матеріали з використанням вузлової системи. Програма підтримує PBR (Physically Based Rendering)-матеріали для фотореалістичних рендерів. Є також можливість працювати з процедурними текстурами, такими як Noise, Gradient, Tiles, що дозволяє створювати детальні матеріали без використання зовнішніх текстурних файлів.

Програма підтримує базові методи UV-розгортки, а також автоматизовані інструменти для створення UV-карт. Це дозволяє ефективно застосовувати текстури навіть на складні поверхні. Додатково 3ds Max має інтеграцію з такими програмами, як Substance Painter, для створення процедурних матеріалів.

3ds Max підтримує широкий набір форматів файлів, включаючи OBJ, FBX, 3DS, STL, DWG та DAE. Це забезпечує зручність імпорту та експорту моделей для роботи з іншими програмами, такими як Blender, ZBrush, AutoCAD та ігровими рушіями, як-от Unreal Engine і Unity.

Переваги Autodesk 3ds Max:

- Потужний набір інструментів для полігонального, сплайнового та NURBS-моделювання.
- Професійні можливості для анімації, включаючи Biped та CAT.
- Вбудовані та сторонні рендер-рушії (Arnold, V-Ray, Corona).
- Глибока інтеграція з CAD-системами та іншими графічними редакторами.
- Високий рівень деталізації моделей та фотореалістичний рендеринг.

Недоліки Autodesk 3ds Max:

- Висока вартість ліцензії, орієнтована на професійні студії.
- Високі системні вимоги для великих сцен із фотореалістичним рендерингом.
- Складний інтерфейс для новачків, потребує тривалого навчання.

Autodesk 3ds Max є одним із найпотужніших інструментів для 3D-моделювання, анімації та візуалізації, орієнтованим на професійне використання у сфері архітектурної візуалізації, кінематографу та дизайну. Його потужні інструменти для роботи з моделями, текстурами, анімацією та фотореалістичним рендерингом роблять його стандартом у галузі.

Unreal Engine – це один із найпотужніших ігрових рушіїв, розроблений компанією Epic Games. Він використовується для створення фотореалістичних 3D-сцен, відеоігор, архітектурних візуалізацій, VR/AR-додатків та кінематографічних проєктів. Завдяки передовим технологіям рендерингу, фізики та анімації, Unreal Engine є стандартом у галузі інтерактивної графіки.

Однією з ключових особливостей Unreal Engine є його можливість створення фотореалістичних сцен у реальному часі. Це досягається завдяки використанню сучасних технологій, таких як Lumen (динамічне глобальне освітлення), Nanite (віртуалізація геометрії для моделей з мільйонами полігонів) та підтримці Ray Tracing (трасування променів). У результаті рушій дозволяє отримувати реалістичні світлові ефекти, відображення, тіні та детальні текстури без необхідності ручної оптимізації моделей.

Unreal Engine не є програмою для створення моделей, але він підтримує імпорт популярних 3D-форматів, таких як FBX, OBJ, USD та Alembic, що робить його сумісним із програмами для моделювання, зокрема Blender, 3ds Max та ZBrush. Для роботи з рівнями та сценами використовується Level Editor, який дозволяє налаштовувати розташування об'єктів, освітлення, анімації та матеріали безпосередньо в середовищі рушія.

Серед вбудованих інструментів для анімації виділяється Sequencer – потужний редактор для створення кінематографічних анімацій та відео. Він

дозволяє працювати з рухами камер, накладенням анімацій, а також редагувати звукові доріжки. Control Rig надає можливості для скелетної анімації, а Live Link дозволяє виконувати захоплення руху (Motion Capture) у реальному часі. Для створення реалістичних цифрових персонажів Unreal Engine пропонує хмарний сервіс MetaHuman Creator, який автоматизує процес створення моделей облич із фотореалістичною деталізацією.

Unreal Engine також надає потужні засоби для фізичних симуляцій через систему Chaos Physics, яка підтримує моделювання твердих тіл, руйнувань, тканин та рідин. Для симуляції ефектів, таких як дим, вогонь, вибухи або дощ, передбачена система частинок Niagara, яка дозволяє створювати складні візуальні ефекти з детальним контролем над фізикою частинок.

Однією з ключових переваг Unreal Engine є система візуального програмування Blueprints, яка дозволяє створювати складні логічні взаємодії без необхідності писати код. Це ідеальний інструмент для дизайнерів та художників, які не мають досвіду програмування. Blueprints працюють за принципом блок-схем, де кожен елемент – це вузол, що виконує певну дію або умову. Для більш досвідчених розробників доступна підтримка мови C++.

Щодо оптимізації, Unreal Engine пропонує систему LOD (Level of Detail), яка дозволяє автоматично знижувати рівень деталізації об'єктів на основі їхньої віддаленості від камери. Крім того, використовується Occlusion Culling – метод, що приховує невидимі об'єкти за кадром, зменшуючи обчислювальне навантаження. Інструмент HLOD (Hierarchical Level of Detail) забезпечує ще вищий рівень оптимізації для великих відкритих сцен.

Unreal Engine є кросплатформним інструментом, що підтримує розробку для таких платформ, як Windows, macOS, Linux, iOS, Android, а також ігрових консолей PlayStation, Xbox та Nintendo Switch. Це робить рушій універсальним рішенням для створення як мобільних ігор, так і високобюджетних AAA-проєктів.

Переваги Unreal Engine:

- Фотореалістичний рендеринг у реальному часі.

- Технології Lumen, Nanite, Ray Tracing для передової графіки.
- Візуальне програмування через Blueprints.
- Підтримка фізичних симуляцій через Chaos Physics та Niagara.
- Вбудовані інструменти для анімації та кінематографії.
- Безкоштовний для некомерційних проєктів (роялті сплачуються лише після досягнення певного прибутку).

Недоліки Unreal Engine:

- Високі вимоги до апаратного забезпечення.
- Складний інтерфейс для новачків.
- Може бути надмірним для простих або мобільних проєктів.

Застосування Unreal Engine:

- Відеоігри: AAA-проєкти, інді-ігри, VR/AR-ігри.
- Архітектурна візуалізація: фотореалістичні віртуальні тури, демонстрації будівельних проєктів.
- Кінематограф: віртуальні знімальні майданчики, анімаційні фільми.
- Навчальні та медичні симуляції: тренажери, інтерактивні навчальні середовища.

Unreal Engine є універсальним рушієм для створення інтерактивного контенту з високим рівнем деталізації та реалістичності. Він підходить для професійних розробників відеоігор, архітекторів, дизайнерів віртуальних середовищ і кінематографістів. Завдяки поєднанню потужних інструментів для анімації, фізики та рендерингу, Unreal Engine є одним із найкращих рішень для створення фотореалістичних інтерактивних проєктів.

Unity – це потужний ігровий рушій, розроблений компанією Unity Technologies, що використовується для створення інтерактивних 3D- та 2D-додатків, відеоігор, анімацій, VR/AR-проєктів та архітектурних візуалізацій. Завдяки простому інтерфейсу, кросплатформенності та гнучким інструментам, Unity став одним із найпопулярніших рушіїв у світі, особливо серед інді-розробників та мобільних ігор.

Однією з ключових переваг Unity є підтримка різноманітних платформ, включаючи Windows, macOS, Linux, iOS, Android, PlayStation, Xbox, Nintendo Switch та VR/AR-гарнітури (Oculus, HTC Vive, HoloLens). Це дозволяє створювати як прості мобільні ігри, так і складні кросплатформені додатки.

Unity надає кілька графічних рушіїв, що відповідають різним вимогам:

- Built-in Render Pipeline – стандартний рендер-рушій для менш вимогливих проєктів.
- Universal Render Pipeline (URP) – оптимізований для мобільних платформ, забезпечує баланс між продуктивністю та якістю зображення.
- High Definition Render Pipeline (HDRP) – призначений для створення фотореалістичних сцен із підтримкою Ray Tracing.

Рушій підтримує технологію Physically Based Rendering (PBR), яка забезпечує фізично коректне освітлення та відображення матеріалів. Постобробка зображень (Post-Processing Stack) дозволяє додавати такі ефекти, як Bloom, Depth of Field, Motion Blur та Ambient Occlusion, що значно покращує візуальне сприйняття сцен.

Unity включає потужний набір інструментів для анімації, серед яких:

- Animator Controller – графічний редактор анімації для створення анімаційних графів.
- Animation Timeline – візуальний редактор ключових кадрів.
- Cinemachine – інструмент для кінематографічного управління камерами.

Програмування в Unity здійснюється за допомогою мови C#. Компонентна архітектура рушія дозволяє створювати складні сцени, де кожен об'єкт складається з компонентів, таких як колізії, анімації, скрипти тощо. Для користувачів без досвіду програмування передбачений інструмент Visual Scripting, що дозволяє створювати ігрову логіку за допомогою візуальних блоків.

Фізичний рушій PhysX забезпечує реалістичні симуляції фізичних процесів, таких як взаємодія твердих тіл, гравітація, зіткнення, пружини та

обмеження. Unity також підтримує систему частинок для створення візуальних ефектів, таких як дим, вибухи, дощ [14].

Оптимізація продуктивності реалізована через функції:

- LOD (Level of Detail) – автоматичне зниження деталізації об'єктів.
- Occlusion Culling – відсікання об'єктів, які не потрапляють у поле зору камери.
- Profiler – інструмент для аналізу продуктивності сцени та споживання ресурсів.

Переваги Unity:

- Простий інтерфейс та зручність для новачків.
- Широка підтримка платформ (від мобільних пристроїв до консолей).
- Вбудовані рендер-рушії для різних завдань (URP, HDRP).
- Потужний фізичний рушій та система частинок.
- Величезна бібліотека готових ресурсів у Asset Store.

Недоліки Unity:

- Менша фотореалістичність порівняно з Unreal Engine.
- Високі вимоги до оптимізації великих сцен.
- Обмежені можливості для надскладних візуалізацій без HDRP.

Застосування Unity:

- Відеоігри: мобільні, інді-проєкти, AR/VR-ігри.
- VR/AR – створення віртуальних турів та навчальних додатків.
- Навчальні симуляції: інтерактивні тренажери для медичних і технічних навчань.
- Архітектурні візуалізації інтерактивні 3D-презентації будівельних проєктів.

Unity – це універсальний рушій для створення інтерактивних 3D-додатків із гнучкими можливостями для розробників усіх рівнів. Завдяки підтримці кросплатформенності, простому інтерфейсу та багатому набору

інструментів, Unity є чудовим вибором для створення мобільних ігор, VR/AR-рішень та симуляцій.

Таблиця 2.1 Порівняння програмного забезпечення

Характеристика	ZBrush	Blender	Autodesk 3ds Max	Unreal Engine	Unity
Призначення	Цифрова скульптура та текстурування	3D-моделювання, анімація, рендеринг	Професійне моделювання та архітектурна візуалізація	Ігровий рушій для фотореалістичних сцен	Ігровий рушій для мобільних ігор, VR/AR
Тип ліцензії	Платний (одноразова покупка)	Безкоштовний, з відкритим кодом	Платний (підписка)	Безкоштовний (роялті при комерційному використанні)	Безкоштовний (роялті при комерційному використанні)
Моделювання	Високий	Високий	Високий	Низький	Низький
Анімація	Низький	Високий	Високий	Високий	Високий
Рендеринг	Високий	Високий	Високий	Високий	Середній
Фізика та симуляції	Низький	Середній	Високий	Високий	Середній
Програмування та скрипти	Відсутнє	Середній (Python)	Середній (MAXScript)	Високий (Blueprints + C++)	Високий (C#, Visual Scripting)
Кросплатформеність	Середній	Високий	Середній	Високий	Високий
Оптимізація сцен	Відсутня	Середній	Високий	Високий	Середній

Основне застосування	Скульптура, створення персонажів	Моделювання, анімація, рендеринг	Архітектурна візуалізація, CAD	AAA-ігри, архітектурна візуалізація, VR/AR	Мобільні ігри, VR/AR, інді-проєкти
Висока деталізація моделей	Високий	Високий	Високий	Високий	Середній
Легкість навчання для новачків	Середній	Середній	Низький	Низький	Високий
Глибина функціональності	Високий	Високий	Високий	Високий	Середній
Підходить для інді-розробників	Низький	Високий	Низький	Середній	Високий
Популярність у AAA-проєктах	Низький	Середній	Високий	Високий	Високий

Для вирішення задачі оптимізації рендерингу 3D-зображень на основі октодерева було обано ігровий рушій Unity завдяки своїм вбудованим інструментам управління рівнями деталізації (LOD) та методам відсікання невидимих об'єктів (Occlusion Culling). Використовуючи компонентний підхід та мову C#, Unity дозволяє програмістам створювати кастомні алгоритми просторового поділу сцени, зокрема через реалізацію структур октодерева для динамічного розподілу геометрії та зниження навантаження на GPU. Завдяки гнучким рендер-рушіям URP та HDRP, рушій може обробляти сцени з високою полігональною складністю, а також ефективно керувати рівнями деталізації для покращення продуктивності без втрати візуальної якості.

2.2 Вимоги до програмного забезпечення

Вимоги до алгоритму оптимізації рендерингу зображень визначають набір характеристик, які необхідно реалізувати для забезпечення ефективності, стабільності та якості візуалізації тривимірних сцен. Чітке визначення цих вимог є критично важливим етапом у розробці програмного забезпечення, оскільки воно гарантує відповідність функціональності потребам користувача та технічним обмеженням.

Функціональні вимоги описують основний функціонал, який алгоритм має виконувати. Вони охоплюють обробку геометрії, управління рівнями деталізації та інтеграцію з графічними рушіями.

Нефункціональні вимоги фокусуються на продуктивності, масштабованості, сумісності та якості програмного забезпечення. Вони визначають стандарти для продуктивності, надійності та зручності використання системи.

Функціональні вимоги:

1. Алгоритм має автоматично розділяти сцену на підобласті за допомогою структури октодерева.
2. Динамічна зміна рівня деталізації об'єктів залежно від відстані до гравця.
3. Автоматизоване спрощення геометрії 3D-об'єктів шляхом зменшення кількості полігонів.
4. Збереження візуальної якості об'єктів поблизу гравця.
5. Можливість перегляду зменшення рівня деталізації у реальному часі.
6. Регулювання відстані для зниження рівня деталізації.

Нефункціональні вимоги:

1. Продуктивність – мінімізація навантаження на GPU за рахунок динамічного управління деталізацією об'єктів.

2. Забезпечення стабільної частоти кадрів під час рендерингу великих сцен.

3. Алгоритм має коректно обробляти сцени з великою кількістю об'єктів без аварійного завершення роботи.

4. Відсутність артефактів при зміні рівня деталізації.

5. Можливість адаптації під сцени різної складності та розміру.

6. Мінімізація графічних артефактів під час спрощення геометрії.

7. Збереження високої деталізації для об'єктів, розташованих поблизу гравця.

2.3 Проектування алгоритму оптимізації рендирингу 3D

У рамках даної роботи було розроблено алгоритм оптимізації рендерингу 3D-сцен на основі октодерева. Запропонований алгоритм оптимізації рендерингу на основі октодерева складається з двох основних частин:

- Просторовий поділ сцени (Octree algrithm) – відповідає за розбиття простору сцени на менші області для зручності обробки. Це дозволяє локалізувати об'єкти та зменшити кількість об'єктів, які потрібно перевіряти під час рендерингу чи обробки колізій.
- Динамічне управління рівнем деталізації – використовується для зменшення кількості полігонів і текстурної деталізації об'єктів залежно від відстані до гравця. Чим далі об'єкт – тим менш деталізованим він буде відображатися.

Послідовність роботи алгоритму виглядає наступним чином:

1) Ініціалізація дерева (рис. 2.1)

Встановлюються параметри сцени: мінімальний розмір вузла, порогова кількість об'єктів та максимальна відстань для динамічного управління рівнем

диталізації. Далі створюється кореневий вузол октодерева, який охоплює всю сцену.

Рисунок 2.1 – Діаграма Octree (крок 1)

2) Додавання об'єктів до вузлів октодерева (рис. 2.2).

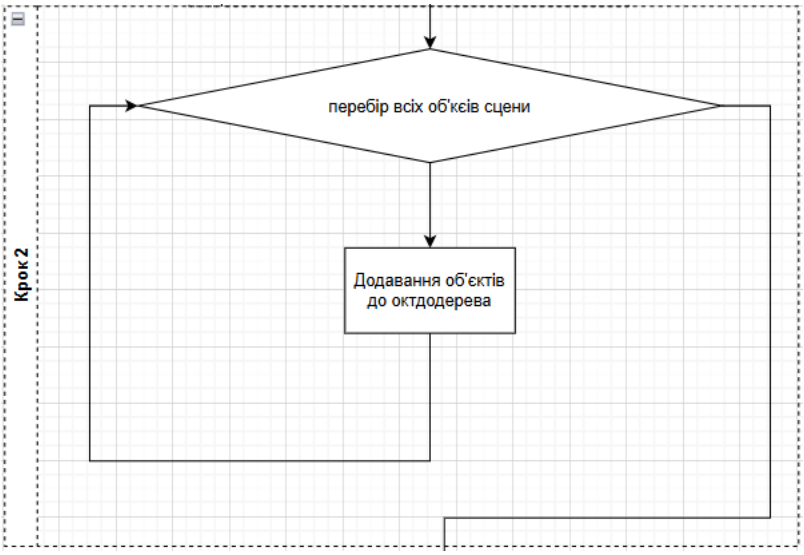


Рисунок 2.2 – Діаграма Octree (крок 2)

3) Поділ простору (рис. 2.3)

Простір вузла поділяється на 8 рівних підобластей. Кожен дочірній вузол створюється з власними межами, що визначаються поділом оригінального

вузла по осях X, Y, Z. Процес триває рекурсивно, доки не буде досягнуто мінімального розміру вузла.

4) Оптимізація рівня деталізації (рис. 2.3)

Об'єкти, які знаходяться далеко від гравця, автоматично знижують свою деталізацію та зменшується кількість вузлів що їх оточують та об’єкднують в загальний простір.

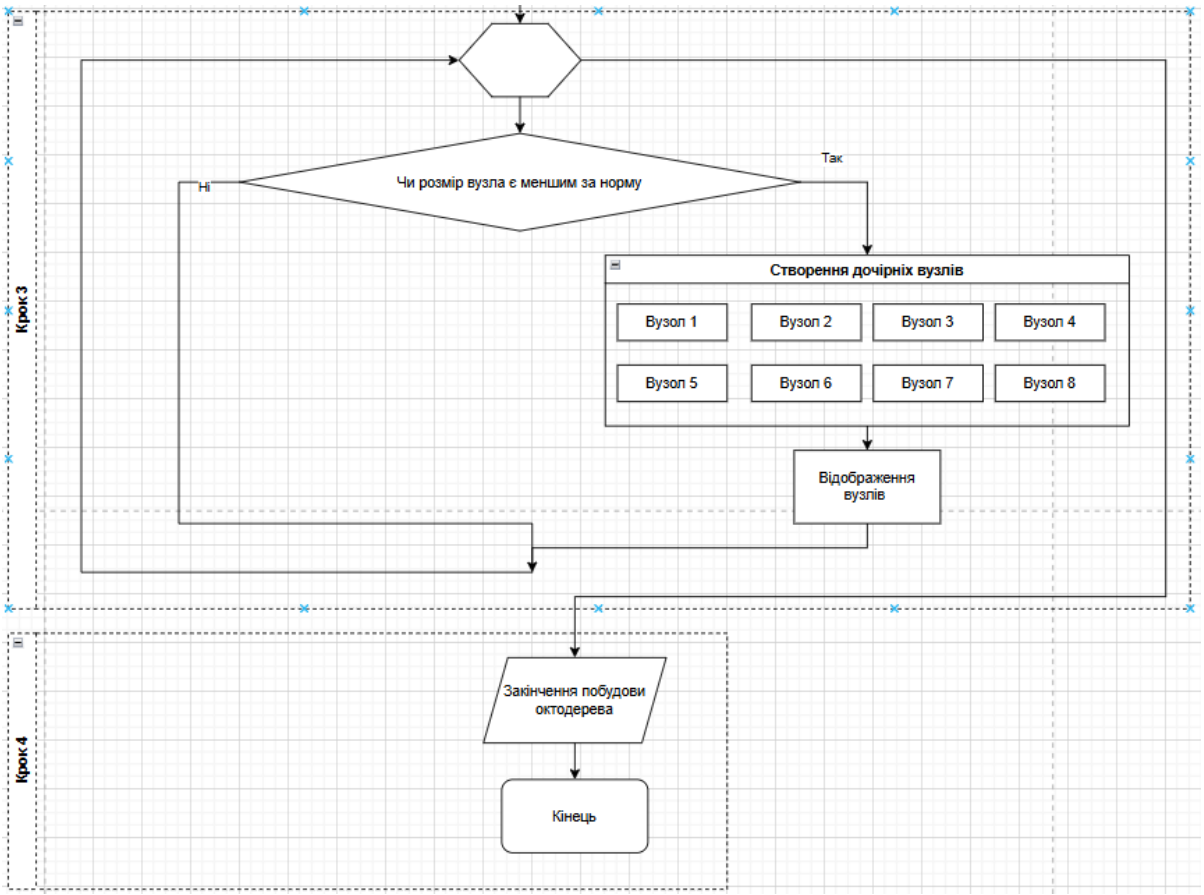


Рисунок 2.3 – Діаграми Octree (крок 3)

Далі для роботи алгоритму динамічне управління рівнем деталізації виконується наступний алгоритм дій:

1) Ініціалізація параметрів (рис. 2.4)

Створюється новий об'єкт для зберігання спрощеної моделі. Ініціалізуються базові дані оригінальної моделі, включаючи вершини та трикутники.

2) Підготовка до спрощення (рис. 2.4)

Створюються структури даних для зберігання нових спрощених вершин та трикутників (списки «List»).

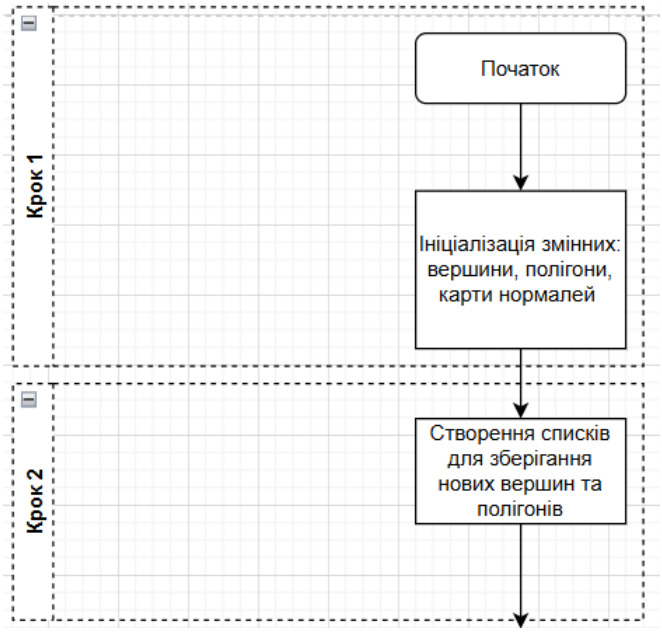


Рисунок 2.4 – Діаграма алгоритму (крок 1,2)

3) Виконання загального циклу перерахунку полігонів (рис. 2.5)

- Виконується ітерація по всіх полігонах моделі.
- Заміна полігонів відбувається якщо відстань між ними становить менше певної величини – коефіцієнту якості.

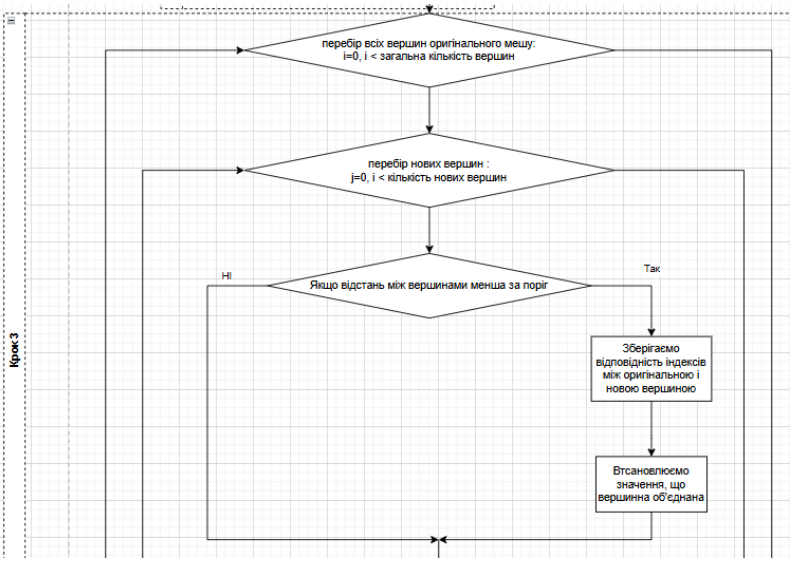


Рисунок 2.5 – Діаграма алгоритму (крок 3)

4) Створення нової структури моделі

- Модифіковані вершини та трикутники передаються новій 3D-моделі.

- Перераховуються нормалі для кожної вершини, що дозволяє зберегти правильне освітлення та відображення на поверхні об'єкта.

Всі етапи алгоритму представлені на наступній діаграмі, рис. 2.6:

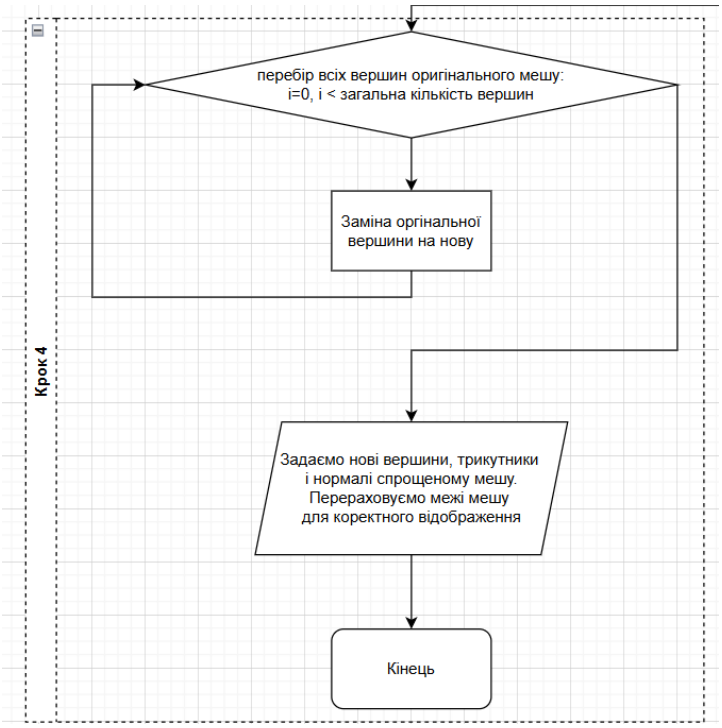


Рисунок 2.6 – Діаграма алгоритму (крок 4)

Головною перевагою запропонованого алгоритму полягає в тому, що розробнику не потрібно самостійно створювати кілька версій текстур різної якості (висока, середня, низька). Алгоритм автоматично генерує та застосовує відповідні текстури для об'єктів, залежно від відстані до гравця, тобто текстури з меншою деталізацією займають менше відеопам'яті.

2.3 Висновок до другого розділу

У другому розділі було проведено комплексний аналіз програмних засобів для реалізації алгоритму оптимізації рендерингу 3D-зображень на основі октодерева. Було розглянуто можливості та функціональність таких інструментів, як Unity, ZBrush, Blender, Autodesk 3ds Max та Unreal Engine. Зокрема, увагу зосереджено на здатності цих середовищ ефективно обробляти складні тривимірні сцени, управлінні рівнями деталізації (LOD) та методах відсікання невидимих об'єктів (Occlusion Culling).

За результатами аналізу, для реалізації поставлених завдань було обрано Unity завдяки його розширеній підтримці оптимізаційних методів, просторових структур даних, а також гнучкості у програмуванні за допомогою мови C#. Unity забезпечує інтеграцію алгоритмів динамічного управління деталізацією об'єктів та спрощення полігональних сіток, що дозволяє ефективно зменшити навантаження на графічний процесор (GPU) та покращити продуктивність при рендерингу великих сцен.

Також було визначено вимоги до алгоритму оптимізації рендерингу. Серед основних функціональних вимог – автоматичний поділ сцени на підобласті за допомогою структури октодерева, динамічне управління рівнями деталізації та збереження візуальної якості об'єктів поблизу камери. Серед нефункціональних вимог – висока продуктивність алгоритму, мінімізація артефактів при зниженні рівня деталізації та стабільність рендерингу.

В результаті аналізу засобів та формулювання вимог до алгоритму дозволили обґрунтовано обрати Unity як основний інструмент для реалізації алгоритму оптимізації рендерингу 3D-зображень на основі октодерева.

РОЗДІЛ 3. РОЗРОБКА АЛГОРИМТУ ОПТИМІЗАЦІЇ 3D ЗОБРАЖЕННЯ НА ОСНОВІ ОКТОДЕРЕВА

3.1 Актуальність розробки нового алгоритму

Зростання складності 3D-сцен, підвищення очікувань користувачів щодо якості графіки, а також прагнення до забезпечення стабільної продуктивності на різних пристроях (від потужних ігрових комп'ютерів до мобільних телефонів) обумовлюють необхідність розробки ефективних підходів до оптимізації рендерингу зображення. Оптимізація графічних процесів стає критично важливою у сучасних умовах, адже дозволяє досягати високого рівня деталізації та реалістичності без втрат у продуктивності, що є особливо важливим для інтерактивних медіа, таких як відеоігри та віртуальна реальність.

Деякі алгоритми, прив'язані до апаратного забезпечення, ставлять вимогу для звичайного споживача медіаконтенту, наприклад ігор, придбати дороге обладнання. Наприклад, технологія DLSS, розроблена компанією NVIDIA, є чудовим прикладом інноваційного підходу до масштабування зображення. DLSS (рис. 3.1) – це технологія масштабування з використанням штучного інтелекту, яка дозволяє підвищити продуктивність рендерингу без значного зниження якості зображення. Вона базується на використанні тензорних ядер, спеціалізованих апаратних блоків, інтегрованих у відеокарти серій NVIDIA RTX. Тому важливо запропонувати універсальне рішення, що дасть змогу забезпечити оптимізацію, без використання складного обладнання.

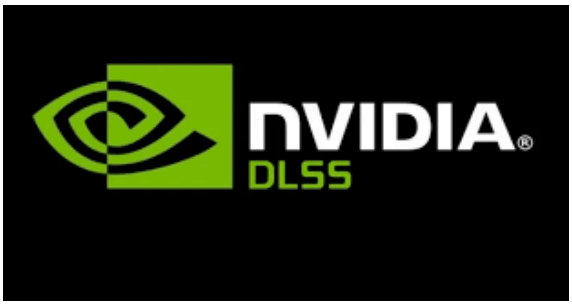


Рисунок 3.1 – Технологія NVIDIA DLSS

Unity є однією з найпопулярніших платформ для розробки ігор та 3D-додатків, завдяки її зручності, кросплатформеності та широким функціональним можливостям. Однак, стандартні інструменти Unity не завжди забезпечують оптимальну продуктивність у масштабних проектах. Відсутність автоматизованих інструментів для оптимального управління рівнями деталізації та виключення невидимих об'єктів у великих сценах збільшує час рендерингу. Додаткові труднощі виникають із динамічним завантаженням ресурсів, що може спричиняти затримки під час гри. Через це розробники змушені використовувати кастомні алгоритми або сторонні плагіни, що ускладнює процес розробки та підтримки великих проектів.

Розробка більш складних ігрових додатків з безліччю об’єктів для цього рушія є досить складним завданням, для яких стартовий інструментарій не пристосований для створення відкритих світів.

Для підвищення ефективності рендерингу зображення необхідно знизити навантаження на відео-пам’ять, спрощуючи геометрію об'єктів, які більше не перебувають у зоні активної взаємодії гравця. Найпростішим способом є створення спрощених варіантів текстур з допомогою різноманітних редакторів, що вимагає знань в побудові 3D об’єктів.

Крім цього, також виникає необхідність вирішення багатьох проблем ефективного управління та обробки об'єктів у тривимірному просторі, з якими взаємодіє гравець. Перевірка та налагодження роботи в симуляціях або системах зіткнень потребує аналізу кожного об'єкта на можливе пересічення з іншими. У великих сценах це призводить до експоненційного зростання часу

обробки. Проблеми відображення малих областей, що може спричинити колізії між об'єктами, тобто не вірне відображення текстур, які перебувають у межах однієї області.

Метою розробки алгоритму є створення ефективного рішення, яке автоматизує оптимізацію об'єктів у великій сцені залежно від їх відстані до гравця. Алгоритм також забезпечуватиме поділ простору сцени на області для спрощення аналізу взаємодії між об'єктами, що спрощує можливості моніторингу колізій та запобігання графічним артефактам. Дане рішення спрямоване на підвищення продуктивності рендерингу, забезпечення стабільної роботи додатку та полегшення управління складними сценами.

3.2 Концептуальна структура алгоритму

За функціоналом алгоритм можна поділити на 2 частини (рис. 3.2):

- Перша частина відповідає за спрощення геометрії об’єктів.
- Друга частина відповідає безпосередньо за розподіл простору сцени на окремі області.

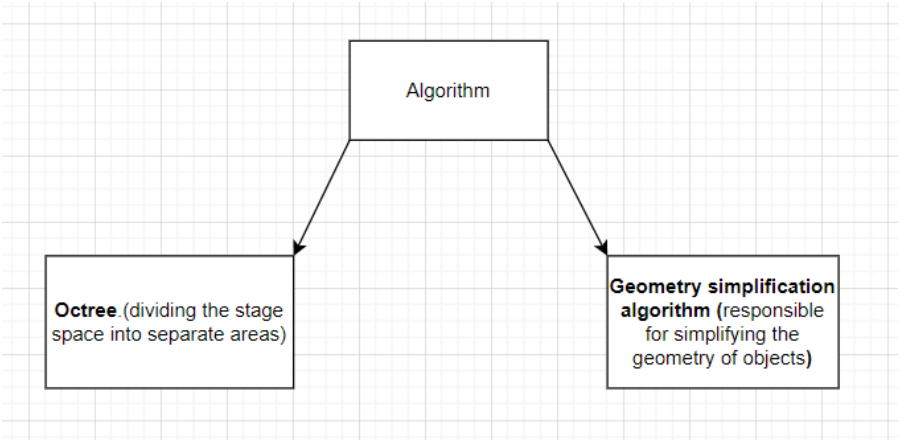


Рисунок 3.2 – Структура алгоритму оптимізації рендерингу

Octree (октодерево) — це перша частина алгоритму яка використовується для розбиття тривимірного простору. Вона являє собою

дерево, кожен вузол якого може бути поділений на вісім рівновеликих підвузлів. Ця структура ідеально підходить для ефективного управління та пошуку об'єктів у великих 3D-сценах.

Geometry Simplification Algorithm - друга частина алгоритму відповідає за зменшення кількості полігонів у 3D об'єктів залежно від їхньої відстані до гравця. Головна мета алгоритму — оптимізувати продуктивність сцени, зберігаючи візуальну якість у важливих ділянках простору

Для демонстрації роботи алгоритму прототип має адаптивно змінювати рівень деталізації кожного об'єкта залежно від положення гравця. Об'єкти, які знаходяться поблизу гравця, повинні залишатися максимально деталізованими, щоб забезпечити якісне візуальне відображення.

3.3 Розробка алгоритму та її структура

Основним інструментом розробки є Visual Studio, забезпечує потужне середовище для створення інтерактивних 3D-додатків (рис. 3.3). Unity активно взаємодіє з бібліотеками та інструментами Visual Studio, що забезпечує інтегровану розробку, дозволяючи розробнику ефективно відлагоджувати код завдяки зручному дебагеру, автозаповненню коду та підтримці рефакторингу. Крім того, інтеграція між Unity та Visual Studio дає змогу швидко створювати і додавати нові структури класів, які організовують код для кожного об’єкта сцени. Це дозволяє розподіляти функціональні навантаження між різними класами, зберігаючи зручність у підтримці та масштабуванні проєкту.

Рисунок 3.3 – Інтеграція Visual Studio в Unity

Завдяки використанню принципів об'єктно-орієнтованого програмування (ООП), мова C# дозволяє створювати класи для кожної області Octree. Ці класи містять власні параметри (розміри, координати, стан перетину з об’єктами) та функції (поділу вузлів, додавання об'єктів). Такий підхід дозволяє ефективно управляти складною структурою Octree, розділяючи відповідальність між різними частинами коду. Використання C# у Unity сприяє зручному моделюванню складних сценаріїв поведінки об'єктів у сцені. Гнучкість ООП дозволяє створювати модульну та масштабовану архітектуру проєкту, що значно спрощує як початкову розробку, так і подальші зміни чи розширення функціоналу [14, 15].

Загалом проект складається з 6 класів. За створення октодерева відповідають наступні 3 класи: CraeteOctree, Octree, OctreeNode (рис. 3.4).

MeshCompr в собі зберігає функціонал для оптимізації рендерингу об’єктів, що входять в октодерево.

Допоміжні класи MainScript MovementPlayer, зберігають дані про глобальні змінні для простоти доступу та управління гравцем.

Рисунок 3.4 – Структура класів проекту

Для реалізації функціоналу спрощення геометрії об’єктів у класі MeshCompr використовуються функції, які взаємодіють із мешем об’єкта. Меш об'єкта (Mesh) є основною тривимірною геометричною структурою в Unity, яка визначає форму об'єкта у просторі. Ця структура складається з декількох основних елементів, кожен із яких відіграє важливу роль у формуванні та візуалізації об'єкта:

- Вершини (vertices): Це набір точок у тривимірному просторі, які визначають основні геометричні контури мешу. Вершини слугують базовими елементами для побудови полігонів і формування складних об’єктів.
- Полігони (triangles): Полігони є поверхнями, що з'єднують вершини для створення зовнішньої форми об'єкта. Зазвичай полігони представлені у вигляді трикутників, які є найпростішою формою для рендерингу у тривимірній графіці.

- Карта нормалей (normals): Нормалі визначають орієнтацію кожної вершини або поверхні полігона у просторі. Вони використовуються для обчислення того, як світло падає на об'єкт, впливаючи на його зовнішній вигляд, особливо на освітлення та тіні.
- UV-координати (originalUVs): Це координати текстурування, які вказують, як текстура повинна бути нанесена на поверхню мешу.

Доступ до таких об’єктів відбувається за рахунок компонентів MeshFilter та MeshRenderer, а також використання відповідних функцій для зміни або оптимізації геометрії. Інформації, щодо вище зазначених параметрів можна спостерігати в редакторі сцени (рис. 3.5):

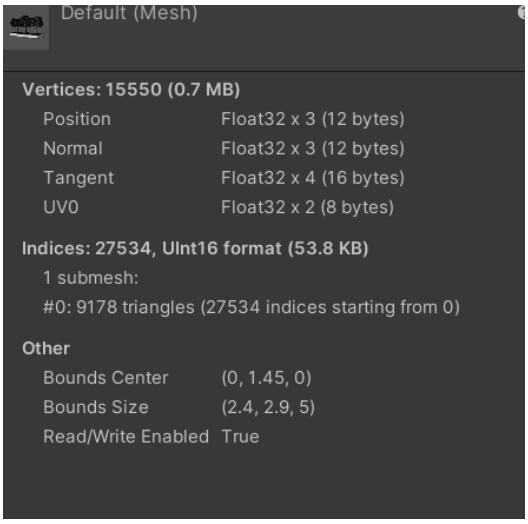


Рисунок 3.5 – Інформація параметрів мешу об’єкту

Для зміни геометрії об’єктів використовується основна функція класу SimplifyMesh(Mesh newMesh). Дана функція складається з наступних частин: ініціалізація вхідних даних мешу об’єкту, створення контейнерів для запису нових даних мешу, перерозподіл вершин для з’єднання полігонів, оновлення даних мешу об’єтку.

Для ініціалізації алгоритму спрощення геометрії об’єктів використовується об’єкт типу Mesh (рис. 3.6), який дозволяє взаємодіяти з даними про меш. Цей об'єкт містить інформацію про вершини (їхні

координати в просторі), трикутники (полігони, що складають геометрію об'єкта), а також інші параметри, як-от нормалі (орієнтація поверхонь) і UV-координати (текстурування). Для доступу до цих даних використовуються стандартні властивості мешу, наприклад: OriginalMesh.vertices, OriginalMesh.triangles, OriginalMesh.normals, OriginalMesh.uv. Ці дані зберігаються в масивах відповідних типів (Vector3[], int[], Vector2[]), що є базою для подальшої обробки алгоритмом.

```
public void SimplifyMesh(Mesh newMesh)
{
    // Отримуємо вершини, трикутники та нормалі оригінального мешу
    Vector3[] originalVertices = OriginalMesh.vertices;
    int[] originalTriangles = OriginalMesh.triangles;
    Vector3[] originalNormals = OriginalMesh.normals;
    Vector2[] originalUVs = OriginalMesh.uv;
    // Списки для нових вершин, трикутників і нормалей
    List<Vector3> newVertices = new List<Vector3>();
    List<int> newTriangles = new List<int>();
    List<Vector3> newNormals = new List<Vector3>();
    List<Vector2> newUVs = new List<Vector2>();
    //Dictionary<int, int> vertexMap = new Dictionary<int, int>();
```

Рисунок 3.6 – Ініціалізація вхідних даних функції

У процесі роботи алгоритму результати спрощення (оновлені вершини, трикутники та інші параметри) зберігаються в динамічних списках типу List<T>. Використання списків замість статичних масивів забезпечує зручність і гнучкість під час додавання, видалення чи оновлення даних. Наприклад, під час об'єднання вершин список автоматично змінює свій розмір, що зменшує потребу в додаткових обчисленнях для управління пам’яттю.

Головною частиною функції є наступні 2 цикли, які перезаписують послідовність вершин для об’єднання в нові полігони (рис. 3.7).

У першому циклі відбувається перевірка кожної вершини оригінальної моделі: вона порівнюється з уже обробленими вершинами, і якщо відстань між ними менша за заданий поріг, вершини об'єднуються (зберігається індекс відповідності між старими та новими вершинами). Якщо ж вершина не була об'єднана, вона додається до списку нових вершин разом із відповідними

нормалями та UV-координатами. Другий цикл проходить по трикутниках оригінальної сітки, замінюючи старі індекси вершин на нові, відповідно до створеної раніше карти індексів. Це дозволяє значно знизити кількість унікальних вершин, зберігаючи при цьому основну форму об'єкта.

Процес об’єднання відбувається, якщо вершини знаходяться на меншій відстані ніж величина threshold – це основний параметр який регулює міру спрощення геометрії текстури. Чим більше дана величина, тим більше спрощення.

```
for (int i = 0; i < originalVertices.Length; i++)
{
    Vector3 currentVertex = originalVertices[i];
    bool merged = false; // Флаг для перевірки, чи була вершина об'єднана
    // Порівнюємо поточну вершину з усіма новими вершинами
    for (int j = 0; j < newVertices.Count; j++)
    {
        // Якщо відстань між вершинами менша за поріг, об'єднуємо їх
        if (Vector3.Distance(newVertices[j], currentVertex) < threshold)
        {
            vertexMap[i] = j; // Зберігаємо відповідність індексів між ориг
            merged = true;
            break; // Виходимо з циклу, оскільки вершина вже об'єднана
        }
    }
    // Якщо вершина не була об'єднана, додаємо її як нову вершину
    if (!merged)
    {
        vertexMap[i] = newVertices.Count; // Зберігаємо відповідність індек
        newVertices.Add(currentVertex); // Додаємо нову вершину
        newNormals.Add(originalNormals[i]); // Додаємо відповідну нормаль
        newUVs.Add(originalUVs[i]);
    }
}
// Створюємо нові трикутники на основі об'єднаних вершин
for (int i = 0; i < originalTriangles.Length; i++)
{
```

Рисунок 3.7 – Цикли перерозподілу вершин полігонів

Остання частина функції це присвоєння результатів алгоритму новому мешу newMesh. Для коректного відображення нової геометрії алгоритм закінчується викликом функції newMesh.RecalculateBounds. Метод RecalculateBounds (рис. 3.8) в Unity є важливим інструментом для коректної роботи з сітками (Meshes) після того, як відбулися зміни у їхніх геометричних даних.

```
// Задаємо нові вершини, трикутники і нормалі спрощеному мешу
newMesh.vertices = newVertices.ToArray();
newMesh.triangles = newTriangles.ToArray();
newMesh.normals = newNormals.ToArray();
newMesh.uv = newUVs.ToArray();
newMesh.RecalculateBounds(); // Перераховуємо межі мешу для ко
```

Рисунок 3.8 – Присвоєння результатів обчислення

Коли відбуваються зміни у вершинах, додаються чи змінюються компоненти сітки, межі (bounds) можуть стати некоректними, що може призвести до проблем у відображенні, рендерингу або фізичних взаємодіях конкретного об’єкту. Викликаючи цей метод, Unity заново обчислює межі для поточної сітки, забезпечуючи точність її взаємодії з іншими об'єктами в сцені. В пам’яті зберігається 2 типи мешів для 1 обєкту – спрощений меш та оригінальний.

Обидва типи мешу активно використовується в функції ReturnOriginalMesh (рис. 3.9). Дана функція використовується для повернення оригінального мешу підчас наближення об’єкту до гравця. Виконується дана операція за рахунок простої заміни: GetComponent<MeshFilter>().mesh = OriginalMesh.

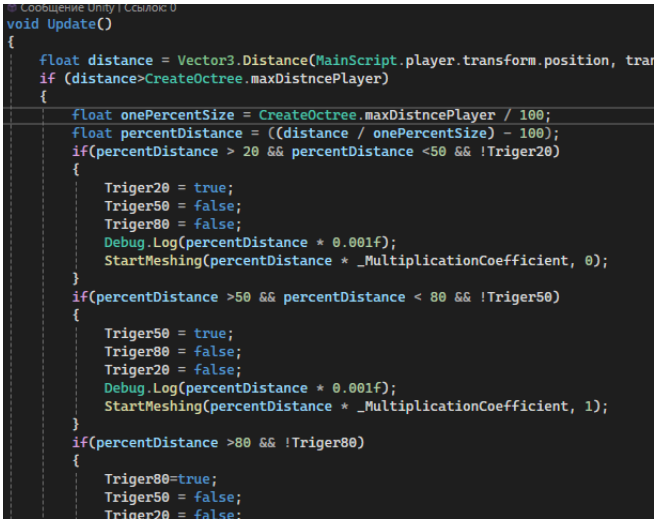
```
public void ReturnOriginalMesh()
{
    if(isMeshFilter)
    {
        GetComponent<MeshFilter>().mesh = OriginalMesh;
    }
    else
    {
        GetComponent<SkinnedMeshRenderer>().sharedMesh = OriginalMesh;
    }
}
```

Рисунок 3.9 – Функція ReturnOriginalMesh

Виконання алгоритму спрощення визначає функція Update. Системна функція виконується 60 кадрів за секунду. В кожному оновленні (Update) обчислюється відстань між об'єктом гравця та поточним об'єктом. Якщо ця

відстань перевищує максимальну дозволену, відбувається перехід між трьома рівнями деталізації (Triger20, Triger50, Triger80), кожен з яких активується при досягненні певного порогу відстані. Залежно від відстані, викликається функція StartMeshing, яка змінює рівень деталізації об'єкта, використовуючи коефіцієнт множення. Якщо ж відстань стає меншою за встановлену межу, об'єкт повертається до оригінальної моделі за допомогою функції ReturnOriginalMesh.

В Update також є тригери типу bool Triger20, Triger50, Triger80. Вони контролюють, щоб алгоритм не спрацьовував повторно на однаковій відстані (рис. 3.10).



```
void Update()
{
    float distance = Vector3.Distance(MainScript.player.transform.position, tra
    if (distance>CreateOctree.maxDistncePlayer)
    {
        float onePercentSize = CreateOctree.maxDistncePlayer / 100;
        float percentDistance = ((distance / onePercentSize) - 100);
        if(percentDistance > 20 && percentDistance <50 && !Triger20)
        {
            Triger20 = true;
            Triger50 = false;
            Triger80 = false;
            Debug.Log(percentDistance * 0.001f);
            StartMeshing(percentDistance * _MultiplicationCoefficient, 0);
        }
        if(percentDistance >50 && percentDistance < 80 && !Triger50)
        {
            Triger50 = true;
            Triger80 = false;
            Triger20 = false;
            Debug.Log(percentDistance * 0.001f);
            StartMeshing(percentDistance * _MultiplicationCoefficient, 1);
        }
        if(percentDistance >80 && !Triger80)
        {
            Triger80=true;
            Triger50 = false;
```

Рисунок 3.10 – Функція Update

Остання Функція StartMeshing (рис. 3.11) призначена для ініціалізації процесу спрощення сітки (meshing) на основі певного порогу точності та індексу, що вказує на відповідний елемент у масиві сіток. Параметр distanceTriangles задає поріг для спрощення трикутників, визначаючи відстань, на яку трикутники можна спрощувати, залежно від їхньої між сусідніми точками. Спочатку функція перевіряє, чи існує меш для заданого індексу в масиві _simplifiedMesh. Якщо на цьому індексі меш ще не був ініціалізована

(тобто null), то створюється новий об'єкт Mesh і викликається метод SimplifyMesh(), який виконує спрощення сітки.

```
public void StartMeshing(float distanceTriangles, int index)
{
    threshold = distanceTriangles;
    if (_simplifiedMesh[index] == null)
    {
        _simplifiedMesh[index] = new Mesh();
        SimplifyMesh(_simplifiedMesh[index]);
    }
    if (GetComponent<SkinnedMeshRenderer>() != null)
    {
        GetComponent<SkinnedMeshRenderer>().sharedMesh = _simplifiedMes
    }
    else
    {
        GetComponent<MeshFilter>().mesh = _simplifiedMesh[index];
    }
}
```

Рисунок 3.11 – Функція StartMeshing

В кінці перевіряється, чи є на об'єкті компонент SkinnedMeshRenderer. Якщо так, спрощена сітка присвоюється для цього компонента. Якщо ж компоненту SkinnedMeshRenderer немає, то сітка призначається компоненту MeshFilter, що також виконує рендеринг статичних об'єктів. Таким чином, функція забезпечує налаштування та застосування спрощеної сітки до об'єкта залежно від його компонентів.

Наступні класи, що потрібно розглянути CraeteOctree, Octree, OctreeNode.

```
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 4
public class CreateOctree : MonoBehaviour
{
    public GameObject[] allGameObjects;
    public float nodeSizeMin = 5;
    public static float maxDistncePlayer = 70f;
    private GameObject PlyerObject;
    Octree octree;
    Сообщение Unity | Ссылок: 0
    void Start()
    {
        octree = new Octree(allGameObjects, nodeSiz
        for (int i = 0; i < allGameObjects.Length;
        {
```

Рисунок 3.12 – Клас CraeteOctree

CreateOctree (рис. 3.12) це стартовий клас для створення октодерева. В класі для роботи використовується наступні величини:

- GameObject[] allGameObjects – масив об’єктів навколо яких будується октодерево
- float nodeSizeMin – мінімальний розмір вузла, до якого може будуватися октодерево
- public static float maxDistncePlayer – відстань від гравця, на якій розпочинає роботу алгоритм спрощення геометрії 3D об’єктів

Функція Start (рис. 3.13) цього класу виконує кілька важливих операцій під час ініціалізації об'єкту. Спочатку створюється новий екземпляр класу Octree, який використовує масив об'єктів allGameObjects, мінімальний розмір вузла nodeSizeMin та максимальну відстань до гравця maxDistncePlayer. Це дозволяє побудувати просторову структуру, яка організовує об'єкти у 3D (откодерево).

```
Сообщение Unity | Ссылок: 0
void Start()
{
    octree = new Octree(allGameObjects, nodeSizeMin, maxDistncePlayer);
    for (int i = 0; i < allGameObjects.Length; i++)
    {
        if(allGameObjects[i].tag == "Player")
        {
            PlyerObject = allGameObjects[i];
        }
        else
        {
            allGameObjects[i].AddComponent<MeshCompr>();
        }
    }
}
```

Рисунок 3.13 – Функція Start

Далі, у циклі, перевіряється кожен елемент масиву allGameObjects. Якщо елемент має тег "Player", то цей об'єкт зберігається в змінній PlyerObject. Якщо ж об'єкт не є гравцем (не має тегу "Player"), то до нього додається компонент

MeshCompr через метод AddComponent<MeshCompr>(). Цей компонент, відповідає за стиснення або оптимізацію мешу об'єкта,

В Update будується октодерево в реальному часі, шляхом ініціалізації об’єкту: octree = new Octree(allGameObjects, nodeSizeMin, maxDistncePlayer).

Для відображення Octree використовується функція OnDrawGizmos (рис. 3.14). Gizmos в Unity — це інструмент для візуалізації даних у сцені, який розробники використовують для створення допоміжних графічних елементів. Вони не відображаються під час виконання гри (в режимі Play), але дуже корисні в редакторі для налагодження та візуалізації різних об'єктів і механік.

```
Сообщение Unity | Ссылок: 0
private void OnDrawGizmos()
{
    if(Application.isPlaying)
    {
        octree.mainNode.DrawBoxGizmo();
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(PlyerObject.transform.position, maxDistncePlayer);
    }
}
```

Рисунок 3.14 – Функція OnDrawGizmos

Далі побудова октодерева в даному випадку здійснюється через створення об'єкта класу Octree, який ініціалізує головний вузол дерева (mainNode), що охоплює всі передані ігрові об'єкти (рис. 3.15).

```
Ссылок: 2
public Octree(GameObject[] allGameObjects, float minNodeSize, float maxDistance)
{
    Bounds newBounds = new Bounds();

    //Розшир.ємо головні межі коробки
    foreach (GameObject gameObject in allGameObjects)
    {
        newBounds.Encapsulate(gameObject.GetComponent<Collider>().bounds);
    }
    //
    float MaxSize = Mathf.Max(newBounds.size.x, newBounds.size.y, newBounds.size.z);
    Vector3 sizeBox = new Vector3(MaxSize, MaxSize, MaxSize) * 0.5f;// задаємо у вект
    newBounds.SetMinMax(newBounds.center - sizeBox, newBounds.center + sizeBox);
    mainNode = new OctreeNode(newBounds, minNodeSize, maxDistance);// Операція створе
    AddObjects(allGameObjects);
}
```

Рисунок 3.15 – Ініціалізація октодерева

Для цього викликається конструктор класу OctreeNode, який отримує межі (Bounds) для вузла, мінімальний розмір вузла (minNodeSize) та максимальну відстань до гравця (maxDistance). Конструктор класу OctreeNode починає з ініціалізації основних параметрів вузла, зокрема встановлення межі та мінімального розміру вузла. Далі розраховуються межі для дочірніх вузлів (childBounds), які діляться на 8 частин для подальшого поділу простору. Кожен з них має свій розмір, який обчислюється як половина висоти батьківського вузла, і кожен з них отримує нові координати для свого розташування відносно центра батьківського вузла (рис. 3.16).

Дочірні вузли розташовуються навколо центра батьківського вузла таким чином, щоб вони покривали всі можливі напрямки (по осі X, Y та Z), створюючи 8 нових меж для кожного з підпросторів. Останнім кроком є пошук об'єкту гравця за тегом "Player", що дозволяє зберігати його для подальших операцій в межах дерева. Це дозволяє ефективно організувати ігрові об'єкти у тривимірному просторі, оптимізуючи їх обробку та взаємодію з гравцем.

```
public OctreeNode(Bounds nodeBounds, float minSize, float maxDistance) // Створення коробки
{
    _nodeBounds = nodeBounds;
    _minSize = minSize;
    _maxDistancePlayerToAnother = maxDistance;
    //Для дочірніх вузлів
    float centrSize = _nodeBounds.size.y / 4;
    float childBoxLength = _nodeBounds.size.y / 2;
    Vector3 childSize = new Vector3(childBoxLength, childBoxLength, childBoxLength);
    childBounds = new Bounds[8];
    //Позаду центру
    childBounds[0] = new Bounds(_nodeBounds.center + new Vector3(-centrSize, centrSize, -centr
    childBounds[1] = new Bounds(_nodeBounds.center + new Vector3(-centrSize, -centrSize, -cent
    childBounds[2] = new Bounds(_nodeBounds.center + new Vector3(centrSize, -centrSize, -centr
    childBounds[3] = new Bounds(_nodeBounds.center + new Vector3(centrSize, centrSize, -centrS
    //попереду центру
    childBounds[4] = new Bounds(_nodeBounds.center + new Vector3(-centrSize, centrSize, centrS
    childBounds[5] = new Bounds(_nodeBounds.center + new Vector3(-centrSize, -centrSize, centr
    childBounds[6] = new Bounds(_nodeBounds.center + new Vector3(centrSize, -centrSize, centrS
    childBounds[7] = new Bounds(_nodeBounds.center + new Vector3(centrSize, centrSize, centrSi
    player = GameObject.FindGameObjectWithTag("Player");
}
```

Рисунок 3.16 – Конструктор класу

Даний клас зберігає масив дочірніх елементів, інформацію про об’єкт гравця, мінімальний розмір вузла та об’єкт типу Bounds.

Bounds — це структура в Unity, яка використовується для представлення тривимірної рамки. Вона часто використовується для визначення обсягу

об'єктів у просторі, їхньої видимості та взаємодії. Bounds складається з двох основних властивостей: center – точка у просторі, що визначає центр рамки, size це тривимірний вектор, який вказує повну ширину, висоту та глибину рамки (рис. 3.17).

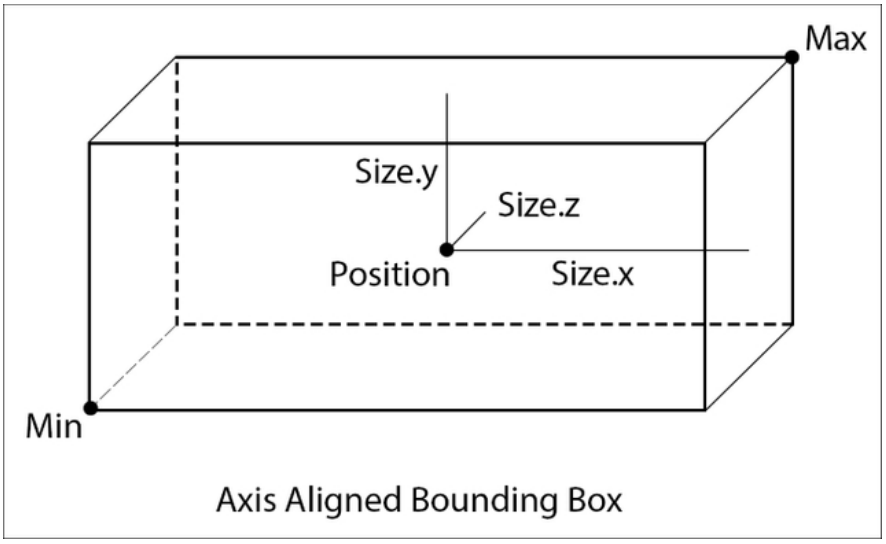


Рисунок 3.17 – Структура Bounds

Функція DivideAndAdd (рис. 3.18) призначена для ділення простору октодерева на ще менші сегменти, а також для додавання об'єктів до відповідних вузлів дерева, залежно від того, чи перетинаються межі вузлів з об'єктами гри. Спочатку перевіряється, чи не досягла висота поточного вузла мінімального розміру (_minSize). Якщо вузол вже дуже малий, подальше ділення не відбувається, і функція просто повертається. Далі, якщо дочірні вузли ще не створені (тобто масив children дорівнює null), ініціалізується масив з 8 дочірніх вузлів.

```
public void DivideAndAdd(GameObject go)
{
    //Обмеження розмірів на ділення
    if (_nodeBounds.size.y<= _minSize){ return;}
    if(children==null){children = new OctreeNode[8];}
    bool dividing  = false;// ділити чи не ділити на ще менші сегменти
    for(int i = 0; i < children.Length; i++)
    {
        if (children[i] == null)
        {
            children[i] = new OctreeNode(childBounds[i], _minSize, _maxDistancePlayerToAnother);
        }
        if (childBounds[i].Intersects(go.GetComponent<Collider>().bounds))//Якщо вузол пересікає об'
        {
            if (go.tag == "Player")
            {
                children[i].IsInPlayerNode = true;
            }
            else
            {
                children[i].IsInPlayerNode = false;
                float distance = Vector3.Distance(go.transform.position, player.transform.position);
                if (distance>_maxDistancePlayerToAnother)// Якщо між об'єктами більша за максимальну
                {
                    children[i] = ChnageSizeOctree(children[i], childBounds[i], distance);
                    children[i].IsInPlayerNode = false;
                }
            }
            dividing = true;
```

Рисунок 3.18 – Функція поділ октодерева (перша частина)

Цикл проходить по кожному з 8 можливих дочірніх вузлів і перевіряє, чи перетинається поточний вузол з колайдером переданого об'єкта (go). Якщо перетин є, відбувається перевірка тегу об'єкта: якщо це гравець, то встановлюється прапор IsInPlayerNode для відповідного дочірнього вузла. Якщо об'єкт не є гравцем, то обчислюється відстань між його позицією та позицією гравця. Якщо ця відстань перевищує максимальну допустиму відстань (_maxDistancePlayerToAnother), розмір відповідного дочірнього вузла змінюється за допомогою методу ChnageSizeOctree (рис. 3.19). Це дозволяє оптимізувати структуру дерева залежно від відстані до гравця.

```
public OctreeNode ChnageSizeOctree(OctreeNode node, Bounds bounds,float distance)
{
    float onePercentSize = _maxDistancePlayerToAnother / 100;
    float percentDistance = ((distance / onePercentSize) - 100) * onePercentSize;

    return new OctreeNode(bounds, _minSize + percentDistance, _maxDistancePlayerToAnother);
}
```

Рисунок 3.19 – Функція ChnageSizeOctree

Після цього, незалежно від того, чи був змінений розмір дочірнього вузла, викликається рекурсивно метод DivideAndAdd (рис. 3.20) для поточного дочірнього вузла, щоб додати об'єкт до ще більш дрібніших сегментів дерева, якщо це необхідно. Якщо жоден з дочірніх вузлів не потребує подальшого ділення (не було перетину з об'єктами), масив дочірніх вузлів скидається на

null, що свідчить про завершення процесу ділення. Цей підхід дозволяє ефективно організувати обробку об'єктів в октодереві, динамічно коригуючи структуру дерева в залежності від взаємодії об'єктів з гравцем.

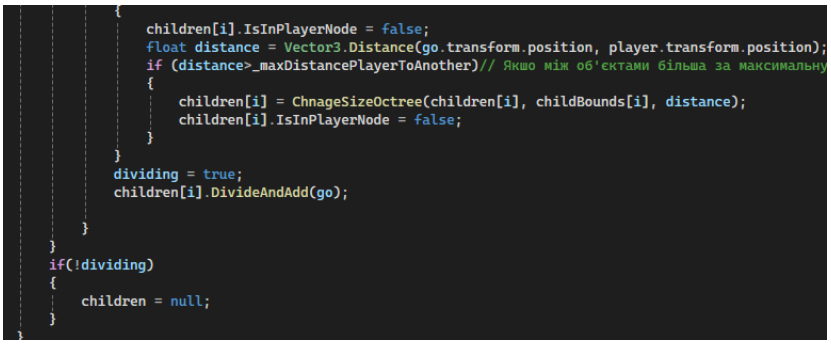


Рисунок 3.20 – Функція поділ октодерева (друга частина)

В іншому випадку переходить на новий ігровий об’єкт. Для відображення використовується функція DrawBoxGizmo (рис. 3.21). Вона також працює рекурсивно, проходячи по всіх дочірніх елементах відображаючи у відповідний колір: if (IsInPlayerNode), Gizmos.color = Color.blue.

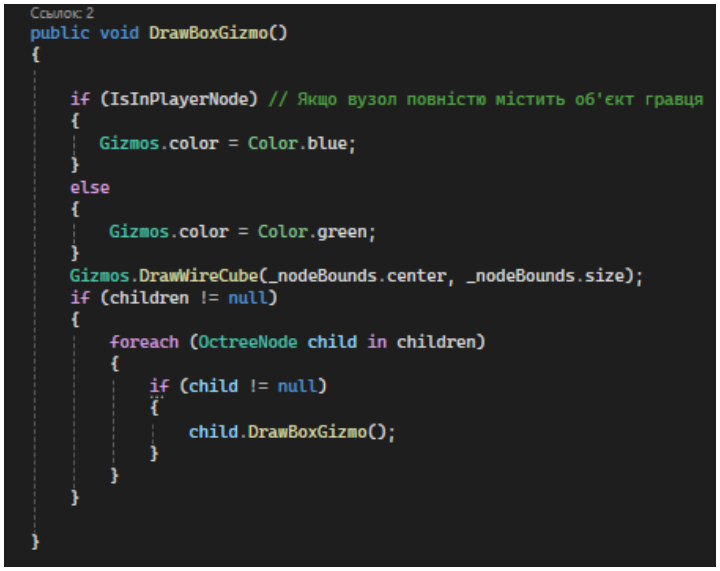


Рисунок 3.21 – Метод DrawBoxGizmo

Отже в результаті роботи даного класу, алгоритм може створювати ієрархічну модель октодерева та відображати її в реальному часі.

Серед допоміжних класів потрібно розглянути клас MainScript. Використовується для зображення продуктивності в проекті та встановлення глобальної змінної GameObject player тобто об’єкту гравця (рис. 3.22).

```
public class MainScript : MonoBehaviour
{
    public Text fpsText;
    public float deltaTime;
    public static GameObject player;
    // Start is called before the first frame update
    Сообщение Unity | Ссылок: 0
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
    }

    // Update is called once per frame
    Сообщение Unity | Ссылок: 0
    void Update()
    {
        deltaTime += (Time.deltaTime - deltaTime) * 0.1f;
        float fps = 1.0f / deltaTime;
        fpsText.text = Mathf.Ceil(fps).ToString();
    }
}
```

Рисунок 3.22 – Клас MainScript

3.4 Налаштування сцени для демонстрації алгоритму

Для роботи алгоритму потрібно створити сцену Unity, разом з 3D об’єктами з якими саме взаємодіятиме алгоритм. Для коректної роботи об’єкти повинні мати наступні ключові компоненти (рис. 3.23):

- Mesh Collider - дозволяє об'єктам використовувати тривимірну сітку (mesh) як форму для визначення фізичних зіткнень (колізій). Цей компонент є частиною системи фізики Unity і дозволяє створювати складніші форми колізій порівняно з простими примітивними формами (як-от Box Collider, Sphere Collider, Capsule Collider). Використовується для побудови октодерева навколо об’єкту.
- SkinnedMeshRenderer або MeshFilter - компонент у Unity, який використовується для зберігання Mesh (сітки) об'єкта. Він працює у парі з

MeshRenderer для візуалізації статичних або анімованих моделей. Використовується в MeshCopr.cs для спрощення геометрії об’єкту.

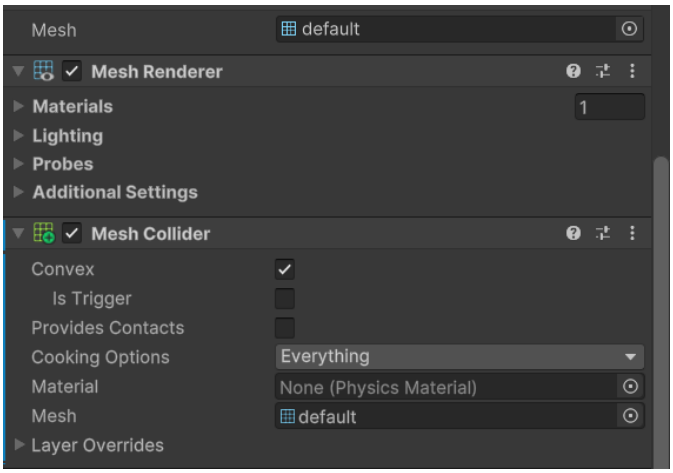


Рисунок 3.23 – Ключові компоненти для роботи алгоритму

Для створення тривимірних об’єктів у проєкті використовувалося програмне забезпечення MagicVoxel (рис. 3.24) — популярний інструмент для моделювання та рендерингу воксельної графіки. MagicVoxel дозволяє створювати детальні тривимірні моделі на основі вокселів (3D-пікселів), що ідеально підходить для проєктів з низькополігональною графікою або стилізованими візуальними ефектами.

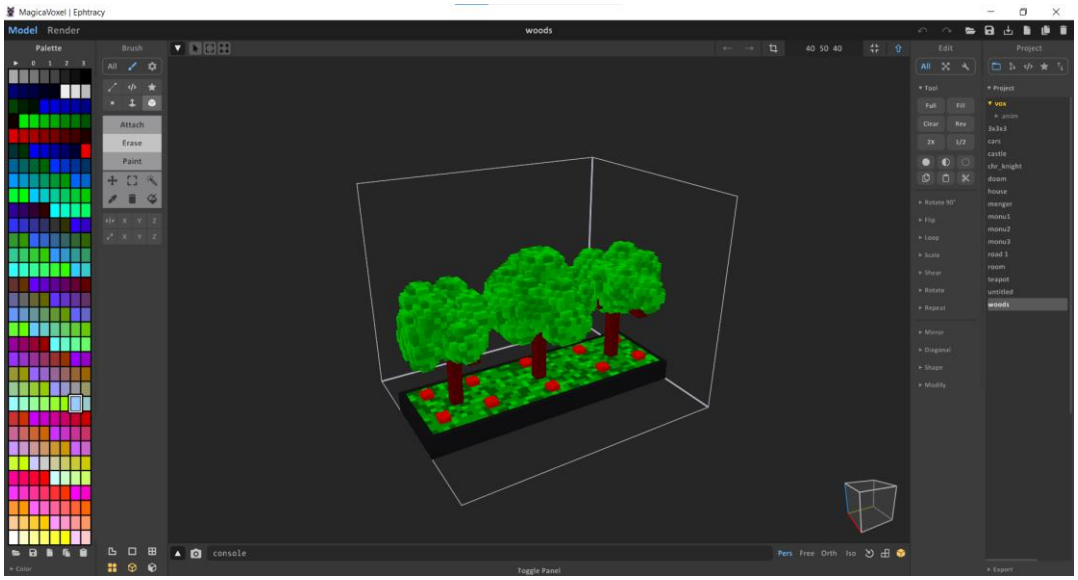


Рисунок 3.24 – MagicVoxel інструмент для моделювання

Сцена складається з декількох 3D об’єктів, які разом створюють реалістичну імітацію міського середовища. Це будівлі, дороги та дерева (рис. 3.25). Послідовність розташування об’єктів забезпечує плавний перехід між зонами, що дозволяє уникнути різких змін у деталізації та порушення занурення гравця.

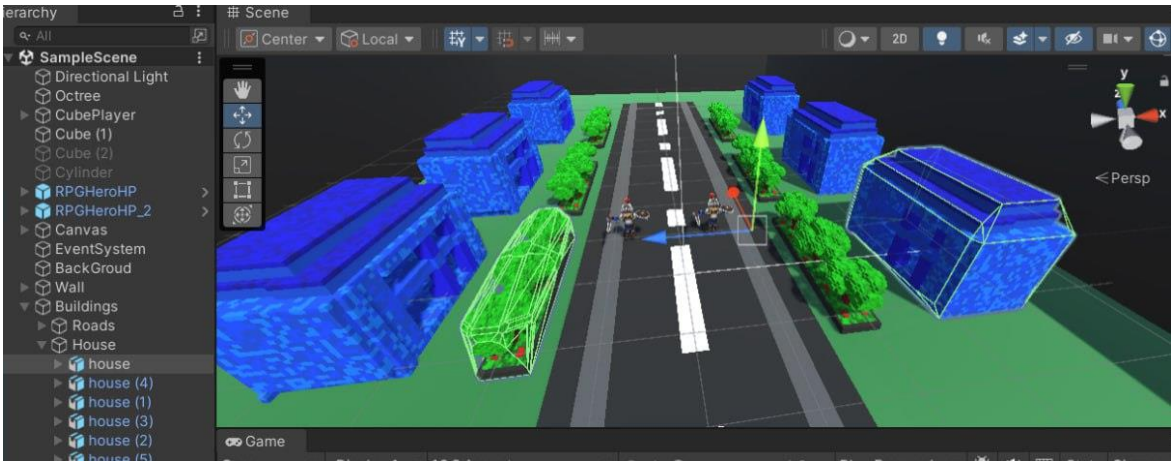


Рисунок 3.25 – Розміщення об’єктів на сцену

Кількість і якість цих об’єктів підібрана таким чином, щоб під час роботи алгоритму оптимізації рівня деталізації, гравець майже не помічав змін у графіці. Це досягається шляхом поступового зниження рівня деталізації для об’єктів, що знаходяться на значній відстані від гравця.

Для роботи алгоритму слугує об’єкт Octree, який має компонент CreateOctree.cs Та має наступні налаштування (рис. 3.26): алгоритм працює з 18 об’єктами сцени. Та розмір мінімального вузла – 5.

Рисунок 3.26 – Об’єкт Octree

3.5 Тестування алгоритму

Для коректної роботи алгоритму оптимізації рівня деталізації розробнику необхідно обрати 3D-об'єкти, які мають відповідні компоненти для обробки геометрії: Mesh Collider, SkinnedMeshRenderer або MeshFilter. Ці компоненти відповідають за обробку сітки об'єкта та дозволяють алгоритму виконувати спрощення геометрії без порушення візуальної цілісності моделі. Обрані об'єкти слід додати до спеціального масиву All Game Objects (рис. 3.27), який використовується алгоритмом для ітеративної обробки та оптимізації. Масив забезпечує централізоване зберігання всіх об'єктів, які підлягають обробці, що дозволяє алгоритму виконувати аналіз геометрії сцени та змінювати рівень деталізації для кожного об'єкта окремо.

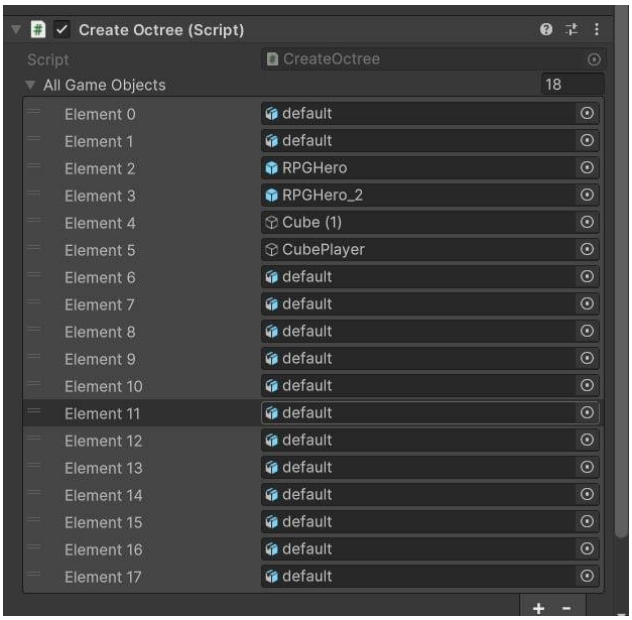


Рисунок 3.27 – Налаштування компоненту CreateOctree.cs

Далі потрібно запустити проект та спостерігати за відбудовою октодерева навколо доданих об’єктів (рис. 3.28):

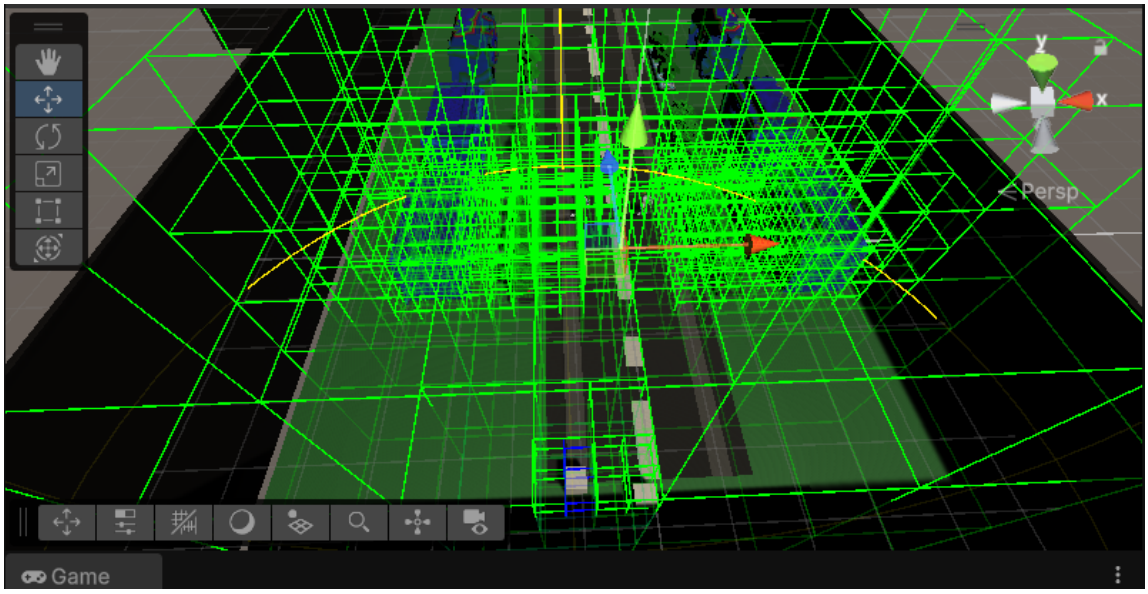


Рисунок 3.28 – Побудова Октодерева

В результаті, можна спостерігати, як на екрані редактора відображається вузли в якій в більшій кількості оточують ті об’єкти які ближче до гравця.

Вузли які оточують гравця мають синій колір. Чим далі гравець від об’єктів тим менше вузлів оточує дані об’єкти (рис. 3.29):

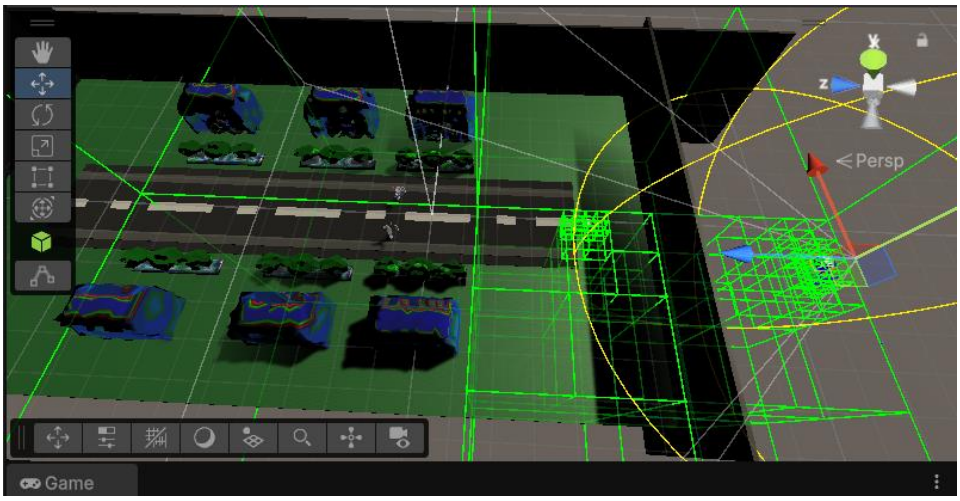


Рисунок 3.29 – Робота алгоритму під час віддалення об’єктів від гравця

Відповідно, під час переміщення об'єкта гравця все далі від інших об'єктів сцени, їхня деталізація поступово зменшується для оптимізації продуктивності. Це реалізується шляхом динамічного зниження рівня деталізації на основі відстані до камери гравця. Об'єкти, що знаходяться у безпосередній близькості до гравця, зберігають високу деталізацію, включаючи складні геометричні форми та текстури високої роздільної здатності. Це важливо, оскільки гравець може чітко спостерігати деталі оточення поблизу.

У той же час, об'єкти, що розташовані далі від гравця або лише частково потрапляють у поле зору камери, автоматично спрощуються: зменшується кількість полігонів у сітці, а також використовується текстура з меншою роздільною здатністю. Це дозволяє знизити навантаження на графічний процесор, забезпечуючи стабільну частоту кадрів без значної втрати якості зображення.

У результаті, під час руху гравця сценою, можна спостерігати, що дальні об'єкти, такі як будівлі, поступово деформуються та втрачають деталізацію. (рис. 3.30):

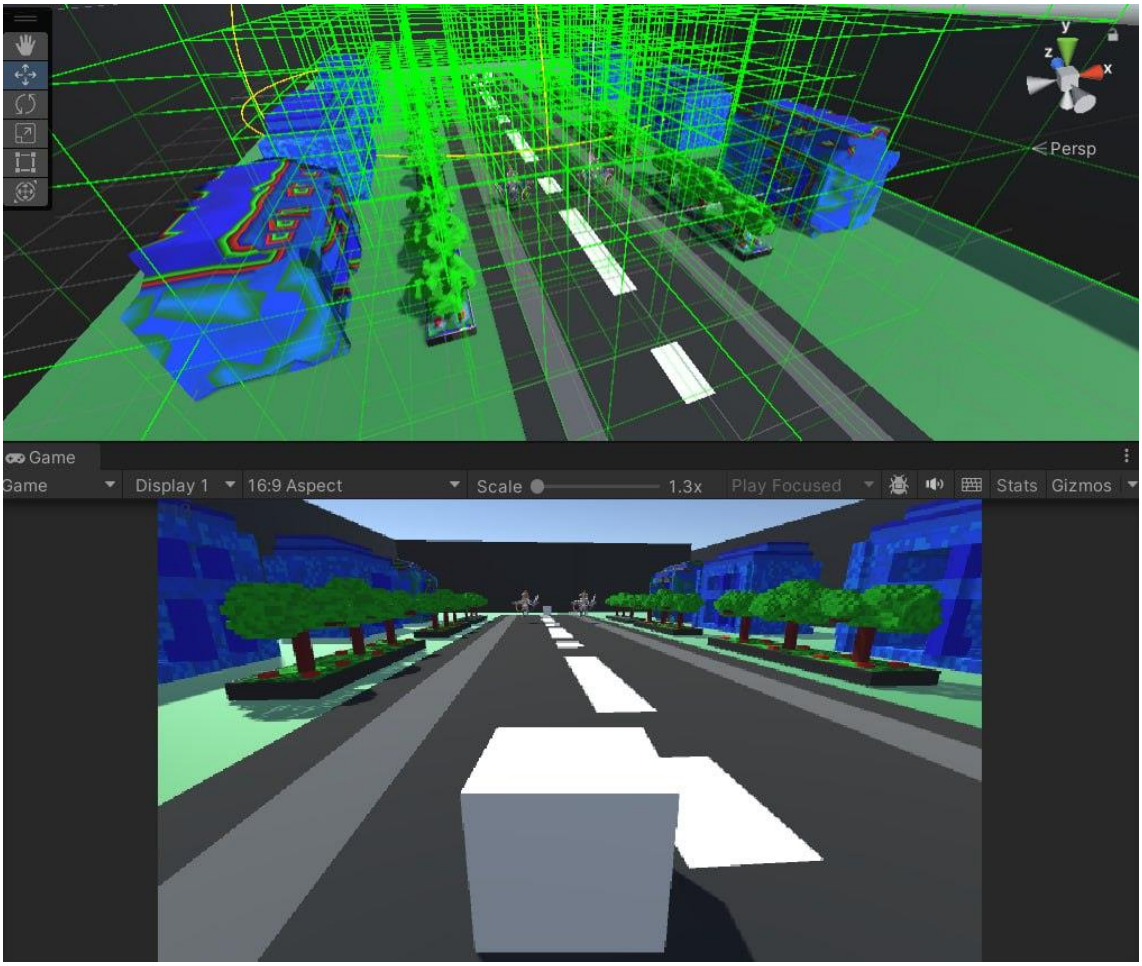


Рисунок 3.30 – Деформація віддалених об’єктів

Більш показовим є приклад роботи алгоритму при порівнянні 2 однакових текстур (рис. 3.31):

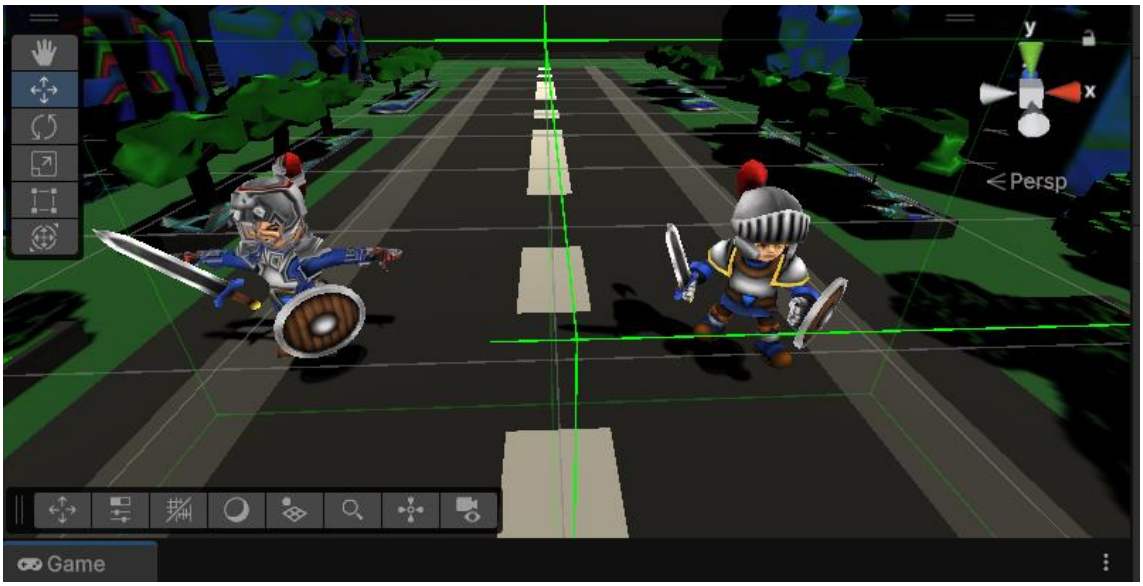


Рисунок 3.31 – Деформація текстури

Загалом якщо відкрити «Диспетчер задач», та перейти в меню «Продуктивність», то можна спостерігати числові показники роботи графічного процесора (3D). При звичайних умовах навантаження на графічний процесор становить 93% (рис. 3.32):

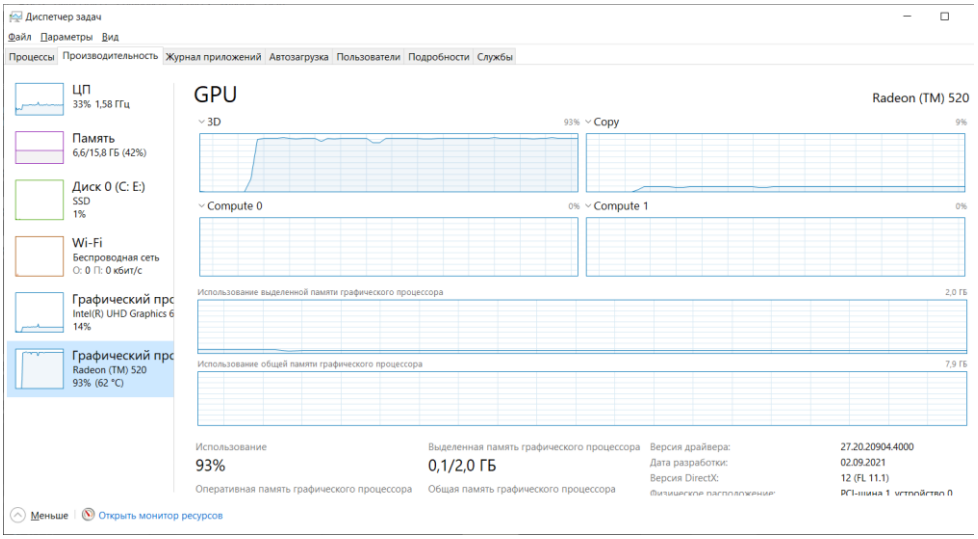


Рисунок 3.32 – Навантаження на графічний процесор

Під час роботи алгоритму оптимізації навантаження на графічний процесор становить в середньому 70% (рис. 3.33):

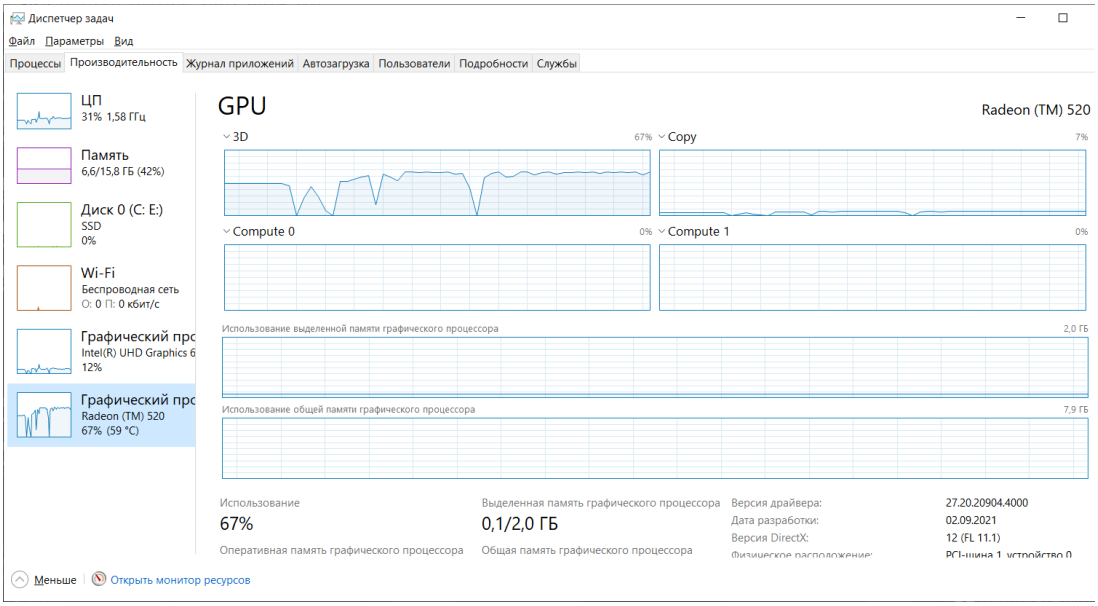


Рисунок 3.33 – Наватаження під час роботи алгоритму

3.6 Висновок до третього розділу

У результаті виконаної роботи було розроблено систему оптимізації рівня деталізації з використанням октодерева. Розроблений алгоритм дозволяє динамічно змінювати деталізацію 3D-об’єктів залежно від їхньої відстані до гравця. Для цього використовувалася структура октодерева, яка забезпечила поділ сцени на сектори, що дозволило зменшити кількість об’єктів, які потребують високої деталізації одночасно. Було впроваджено механізм відслідковування положення гравця в реальному часі, що дозволяє адаптивно змінювати рівень деталізації. Усі об’єкти були оптимізовані таким чином, щоб гравець не помічав різких змін у якості графіки, зберігаючи відчуття занурення у сцену.

ВИСНОВОК

У процесі виконання кваліфікаційної магістерської роботи на тему «Оптимізація рендерингу 3D-зображення на основі октодерева» було проведено комплексне дослідження методів оптимізації рендерингу тривимірних сцен та розроблено програмний алгоритм для підвищення продуктивності рендерингу. Робота включала теоретичне обґрунтування вибору методів, практичну реалізацію алгоритму та тестування програмного продукту.

Основними завданнями, які були вирішені під час виконання роботи, стали:

1. Проведення аналізу існуючих алгоритмів рендерингу 3D-зображень та методів оптимізації, включаючи растеризацію, трасування променів, path tracing, radiosity та photon mapping.
2. Обґрунтування вибору технології октодерева для реалізації просторового поділу сцени та динамічного управління рівнем деталізації (LOD).
3. Розробка алгоритму, що поєднує структуру октодерева з автоматичним управлінням рівнем деталізації об'єктів, для зниження обчислювального навантаження.
4. Програмна реалізація алгоритму оптимізації в середовищі Unity із використанням мови програмування C#, що забезпечило ефективне управління рендерингом у реальному часі.
5. Тестування програмного продукту на відповідність функціональним та нефункціональним вимогам, включаючи якість візуалізації, стабільність роботи та продуктивність.

У роботі було здійснено ґрунтовний аналіз програмних засобів для розробки 3D-додатків, включаючи Unity, Blender, ZBrush, Autodesk 3ds Max та Unreal Engine. Вибір рушія Unity було обґрунтовано його підтримкою

механізмів оптимізації, таких як Occlusion Culling та LOD, а також гнучкістю у програмуванні через C#.

Особливу увагу було приділено принципам оптимізації сцен за допомогою октодерева, яке дозволяє зменшити обчислювальні витрати шляхом розбиття сцени на підобласті. Це дозволило досягти значного покращення продуктивності рендерингу без втрати візуальної якості.

У процесі роботи було сформовано наступні вимоги до програмного забезпечення:

- Функціональні вимоги: автоматизоване управління рівнем деталізації об'єктів, поділ сцени на підобласті, динамічна зміна рівня полігональності.
- Нефункціональні вимоги: забезпечення стабільної частоти кадрів, мінімізація навантаження на GPU, сумісність із великими сценами.

Розроблений алгоритм успішно пройшов тестування у середовищі Unity, підтвердивши свою ефективність у зниженні обчислювальних витрат при рендерингу складних тривимірних сцен. Він може бути використаний у розробці відеоігор, додатків для віртуальної реальності та архітектурних візуалізацій.

Таким чином, поставлена мета щодо розробки алгоритму оптимізації рендерингу 3D-зображень на основі октодерева була досягнута, а отримані результати можуть бути використані для подальших досліджень у сфері комп'ютерної графіки та обробки зображень.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Jing Sun, Hong Tao Wang. Research on Optimization and Application of LOD Algorithm in Virtual Reality Visualization of Terrain, Applied Mechanics and Materials, 2014, P.1258-1261: https://doi.org/10.4028/www.scientific.net/AMM.687-691.1258

2. Brigham Bagley, Shankar P. Sastry, Ross T. Whitaker, A Marching-tetrahedra Algorithm for Feature-preserving Meshing of Piecewise-smooth Implicit Surfaces. Procedia Engineering, Vol. 163, 2016, P. 162-174

3. Introduction to Occlusion Culling, – [Електронний ресурс] – Режим доступу: https://medium.com/@Umbra3D/introduction-to-occlusion-culling-3d6cfb195c79

4. Levenber G J. Fast view-dependent level-of-detail rendering using cached geometry. In: Proceedings of IEEE Visualization, Boston, USA: IEEE Press, 2002: 259-266. DOI: 10.1109/visual.2002.1183783

5. Real-Time Rendering Resources – [Електронний ресурс] – Режим доступу: https://www.realtimerendering.com/#pipeopt

6. *Physically Based Rendering: From Theory To Implementation* – [Електронний ресурс] – Режим доступу: https://www.pbr-book.org/4ed/contents

7. Fuetterling, V., C. Lojewski, F.-J. Pfreundt, B. Hamann, and A. Ebert. Accelerated single ray tracing for wide vector units. *Proceedings of High Performance Graphics (HPG '17)*, 2017. 6:1–9. DOI:10.1145/3105762.3105785

8. Hofmann, N., J. Hasselgren, P. Clarberg, and J. Munkberg. Interactive path tracing and reconstruction of sparse volumes. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 4* (1), 2021. 5:1–19. DOI:10.1145/3451256

9. Rendering an Image of a 3D Scene – [Електронний ресурс] – Режим доступу: https://www.scratchapixel.com/lessons/3d-basic-rendering/rendering-3d-scene-overview/computer-discrete-raster.html

10. Learn OpenGL. Textures – [Електронний ресурс] – Режим доступу: https://learnopengl.com/Getting-started/Textures

11. Особливості програми Zbrush [Електронний ресурс] – Режим доступу: https://3ddevice.com.ua/blog/3d-printer-obzor/obzor-programmy-zbrush/

12. Blender для новачків [Електронний ресурс] – Режим доступу: https://younglinux.info/blender.php

13. 3Ds основи Max [Електронний ресурс] – Режим доступу: https://3ddevice.com.ua/blog/3d-printerobzor/obzor-3ds-max/

14. Документація Unity3D [Електронний ресурс]. – Режим доступу: https://docs.unity.com

15. Документація C# [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/ru-ru/dotnet/csharp/