ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: “Метод динамічного шейдингу на основі шейдерної оптимізації”

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Василь ЯКОВЧУК
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Василь ЯКОВЧУК

Керівник: Віталій ЗАЛИВА
доктор філософії (PhD)

Рецензент: _____
науковий ступінь, вчене звання *Ім’я, ПРІЗВИЩЕ*

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Яковчуку Василю Андрійовичу

1. Тема кваліфікаційної роботи: Метод динамічного шейдингу на основі шейдерної оптимізації

керівник кваліфікаційної роботи Віталій ЗАЛИВА, доктор філосфії (PhD),

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, параметри рендерингу, методи шейдингу, вимоги до оптимізації графіки.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз сучасних методів рендерингу та шейдингу.

2. Дослідження існуючих підходів до оптимізації графічних шейдерів.

3. Розробка алгоритму динамічного шейдингу на основі шейдерної оптимізації.

5. Перелік ілюстративного матеріалу: *презентація*

1. Мета, об’єєт та предмет дослідження.
2. Актуальність роботи.
3. Порівняння методів.
4. Математична модель MRR.
5. Візуалізація робити MRR.
6. Математична модель ALD.
7. Візуалізація робити ALD.
8. Екранні форми.
9. Результати дослідження.
10. Висновки.
11. Публікації та апробація роботи.

6. Дата видачі завдання «16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.10-23.10.2024	
2	Вивчення сучасних методів шейдингу та оптимізації рендерингу	24.10-26.10.2024	
3	Розробка теоретичної бази для алгоритму динамічного шейдингу	27.10-04.11.2024	
4	Реалізація шейдерної оптимізації у середовищі Unity	05.11-12.11.2024	
5	Тестування алгоритму на прикладних сценах	13.11.2024	
6	Аналіз результатів тестування та коригування методу	14.11-17.11.2024	
7	Оформлення роботи: вступ, висновки, реферат	18.11-23.11.2024	
8	Розробка демонстраційних матеріалів	24.11-28.11.2024	
9	Попередній захист роботи	29.11.2024	

Здобувач вищої освіти _____ Василь ЯКОВЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Віталій ЗАЛИВА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 70 стор., 1 табл., 55 рис., 35 джерел.

Мета роботи — підвищення продуктивності рендерингу гри шляхом впровадження методу багатозонального рендерингу роздільної здатності та адаптивної якості освітлення на основі рівнів деталізації (Level of Detail).

Об'єкт дослідження — процес шейдингу та рендерингу графіки в ігровому середовищі.

Предмет дослідження — методи динамічного шейдингу та оптимізації шейдерів для мульти-регіонального рендерингу та адаптивної якості освітлення.

У роботі використано методи математичного моделювання, аналізу алгоритмів рендерингу, графічних оптимізацій, а також методи експериментального тестування для оцінки продуктивності.

Проведено дослідження сучасних підходів до шейдингу та рендерингу графіки, з акцентом на управління ресурсами.

Створено математичні моделі для багатозональної роздільної здатності та адаптивного управління деталізацією освітлення, з урахуванням просторового положення об'єктів та їх важливості в сцені.

Сформульовано принципи управління рівнями деталізації освітлення та рендерингу в межах різних зон ігрової області екрану.

Розроблено прототип, який забезпечує динамічну зміну рівнів деталізації рендерингу та оптимальне використання графічних ресурсів.

Виконано моніторинг, проаналізовано вплив рішень на продуктивність та якість зображення.

Результати дослідження показують ефективність та перспективність застосування методу динамічного шейдингу та рендерингу для вирішення задач оптимізації графічного рендерингу. Запропонований підхід продемонстрував здатність значно знижувати навантаження на обчислювальні ресурси без

суттєвого впливу на якість візуалізації, що робить його перспективним для широкого впровадження в сучасні ігрові рушії.

КЛЮЧОВІ СЛОВА: ДИНАМІЧНИЙ ШЕЙДИНГ, БАГАТОЗОНАЛЬНИЙ РЕНДЕРИНГ, АДАПТИВНЕ ОСВІТЛЕННЯ, РІВЕНЬ ДЕТАЛІЗАЦІЇ, ОПТИМІЗАЦІЯ ГРАФІКИ.

ABSTRACT

The text part of the qualification work for the master's degree: 70 pages, 1 tables, 55 figures, 35 sources.

Purpose — to increase the performance of game rendering by implementing a method of multi-zone rendering of resolution and adaptive lighting quality based on Level of Detail.

The object of research — the process of shading and rendering graphics in the game environment.

The subject of research is methods of dynamic shading and shader optimization for multi — regional rendering and adaptive lighting quality.

The work uses methods of mathematical modeling, analysis of rendering algorithms, graphical optimizations, as well as methods of experimental testing to evaluate performance.

A study of modern approaches to graphics shading and rendering, with an emphasis on resource management, was conducted.

Mathematical models for multi-zone resolution and adaptive control of lighting detail, taking into account the spatial position of objects and their importance in the scene, were created.

The principles of controlling the levels of lighting detail and rendering within different zones of the screen's play area are formulated.

A prototype has been developed that provides a dynamic change in rendering detail levels and optimal use of graphics resources.

Monitoring was performed and the impact of the solutions on performance and image quality was analyzed.

The results of the study show the effectiveness and prospects of using the dynamic shading and rendering method to solve the problems of optimizing graphic rendering. The proposed approach has demonstrated the ability to significantly reduce

the load on computing resources without significantly affecting the quality of visualization, which makes it promising for widespread implementation in modern game engines.

KEYWORDS: DYNAMIC SHADING, MULTI-ZONE RENDERING, ADAPTIVE LIGHTING, LEVEL OF DETAIL, GRAPHICS OPTIMIZATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

DLSS - Супервибірка глибокого навчання (Deep learning super sampling)

FSR - Суперроздільна здатність FidelityFX (FidelityFX Super Resolution)

XESS - Супервибірка Xe (Xe Super Sampling)

MRR - Багатозональна роздільна здатність (Multi-Region Resolution)

ALD - Адаптивна якість освіьлення (Adaptive Lighting Detail)

CPU - Центральний процесор (Central Processing Unit)

GPU - Графічний процесор (Graphics Processing Unit)

LOD - Рівень деталізації (Level of detail)

SDF - Поля підписаних відстаней (Signed Distance Fields)

HLSL - Мова високорівневого шейдингу (High-Level Shading Language)

GLSL - Мова шейдингу OpenGL (OpenGL Shading Language)

FPS - Кадри за секунду (Frames Per Second)

UI - Користувацький інтерфейс (User Interface)

HUD - Інформаційний дисплей (Heads-Up Display)

API - Інтерфейс прикладного програмування (Application Programming Interface)

VRS - Змінна швидкість шейдингу (Variable Rate Shading)

GI - Глобальне освітлення (Global Illumination)

SDFGI - Глобальне освітлення на основі полів підписаних відстаней (Signed Distance Field Global Illumination)

SDF - Поле підписаних відстаней (Signed Distance Field)

LPV - Об'єми розповсюдження світла (Light Propagation Volumes)

TAA - Темпоральне згладжування (Temporal Anti-Aliasing)

NIS - Масштабування зображення NVIDIA (NVIDIA Image Scaling)

RDS - Масштаб щільності рендерингу (Rendering Density Scale)

ВСТУП

Сучасна індустрія відеоігор стрімко розвивається, а разом з нею постійно зростають вимоги до продуктивності та якості графіки. Конкурентний ринок вимагає ігор з реалістичною графікою, складними світловими ефектами та високою деталізацією предметів. Тим не менше, значний відсоток геймерів користується гаджетами початкового або середнього рівня, такими як смартфони, ноутбуки та старі консолі. Це робить розробникам надзвичайно складним досягнення балансу між надійною продуктивністю та приголомшливими візуальними ефектами.

Актуальність цієї роботи полягає в розробці нових підходів до оптимізації, які дозволять зберігати високу якість графіки за умов обмежених апаратних ресурсів. Запропоновані методи, такі як Multi-Region Resolution (MRR) та Adaptive Lighting Detail (ALD), зосереджені на динамічному управлінні використання ресурсів графічного процесора, зокрема шляхом адаптації рівня деталізації освітлення та часткого зниження роздільної здатності областей екрану.

Наукова цінність цієї роботи полягає у створенні нових методів динамічного шейдингу, які можуть стати базисом для подальших досліджень у сфері графічної оптимізації. Практична значущість полягає у можливості впровадження запропонованих методів у сучасні ігрові рушії (наприклад, Unreal Engine чи Unity), що дозволить розширити спектр платформ для відеоігор та підвищити доступність високоякісної графіки для кінцевого користувача.

Мета роботи — підвищення продуктивності рендерингу гри на основі багатозонального рендерингу роздільної здатності та адаптивної якості освітлення на основі рівнів деталізації підходу LOD.

Об’єкт дослідження — процес шейдингу та рендерингу графіки в ігровому середовищі.

Предмет дослідження — методи динамічного шейдингу та оптимізації шейдерів для мульти-регіонального рендерингу та адаптивної якості освітлення.

Ця робота має на меті вирішити наступні завдання:

1. Огляд процесів рендерингу. Проведено дослідження сучасних підходів до шейдингу та рендерингу графіки, з акцентом на управління ресурсами.
2. Математичне моделювання. Створено математичні моделі для багатозональної роздільної здатності та адаптивного управління деталізацією освітлення, з урахуванням просторового положення об’єктів та їх важливості в сцені.
3. Розробка концепції. Сформульовано принципи управління рівнями деталізації освітлення та рендерингу в межах різних зон ігрової області екрану.
4. Розробка та прототипування. Розроблено прототип, який забезпечує динамічну зміну рівнів деталізації рендерингу та оптимальне використання графічних ресурсів.
5. Тестування та результати. Виконано моніторинг, проаналізовано вплив рішень на продуктивність та якість зображення.

Таким чином, ця робота є актуальною і важливою, оскільки вона намагається вирішити основну проблему, з якою стикається сучасна ігрова індустрія: знайти компроміс між чудовими візуальними ефектами та економним використанням технологій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз існуючих методів шейдингу

1.1.1 Каскадний підхід

Каскадний підхід на рисунку 1.1 [1, с. 364-368, 2, с. 320-325] — це метод підвищення ефективності рендерингу та затінення комп'ютерної графіки, особливо для складних візуальних ефектів, таких як освітлення та тіні. Ця техніка розділяє велику область сцени на декілька «каскадів» або секцій, кожна з яких має оптимізоване для неї рішення рендерингу. Каскадний підхід дає змогу використовувати кілька методів рендерингу залежно від важливості об'єктів і відстані до камери, замість того, щоб зображувати всю сцену одним способом. Швидкість рендерингу та якість зображення в цьому методі збалансовані.

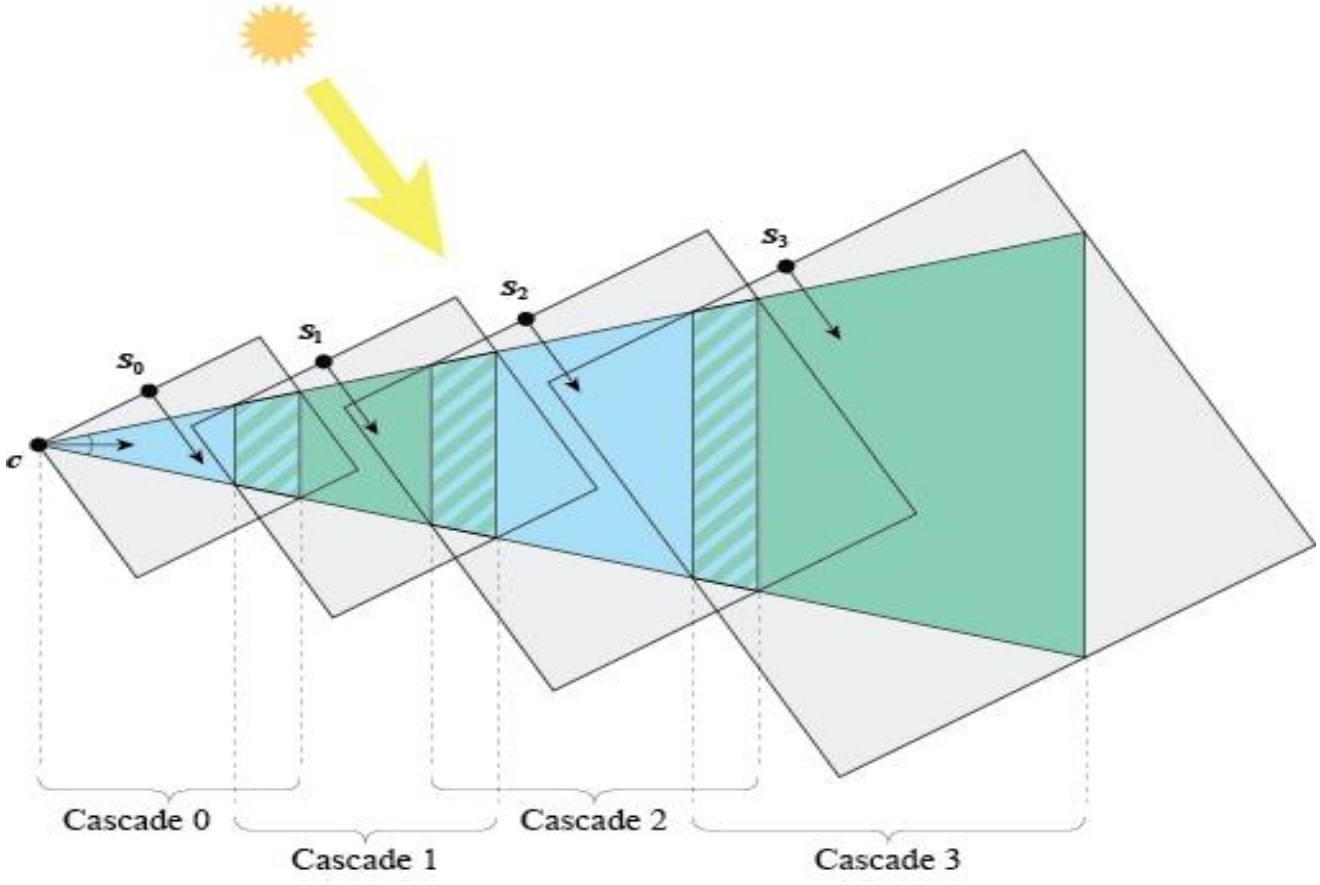


Рис. 1.1 Приклад каскадного шейдингу

Здатність каскадного підходу використовувати різний ступінь деталізації для різних елементів сцени є однією з його найважливіших переваг. Наприклад, для ближчих об'єктів використовуються більш реалістичні моделі та текстури, а для віддалених - менш деталізовані. Це дозволяє підтримувати візуальні ефекти найближчих об'єктів на високому рівні, водночас значно зменшуючи навантаження на систему. При рендерингу обширних, відкритих просторів, наприклад, пейзажів, де різні частини сцени вимагають різного рівня деталізації, каскадна техніка є дуже корисною.

Основна ідея каскадного підходу полягає в поділі сцени на кілька рівнів деталізації (LOD, рисунок 1.2), кожен з яких відповідає за створення окремої ділянки зображення з різним рівнем деталізації.

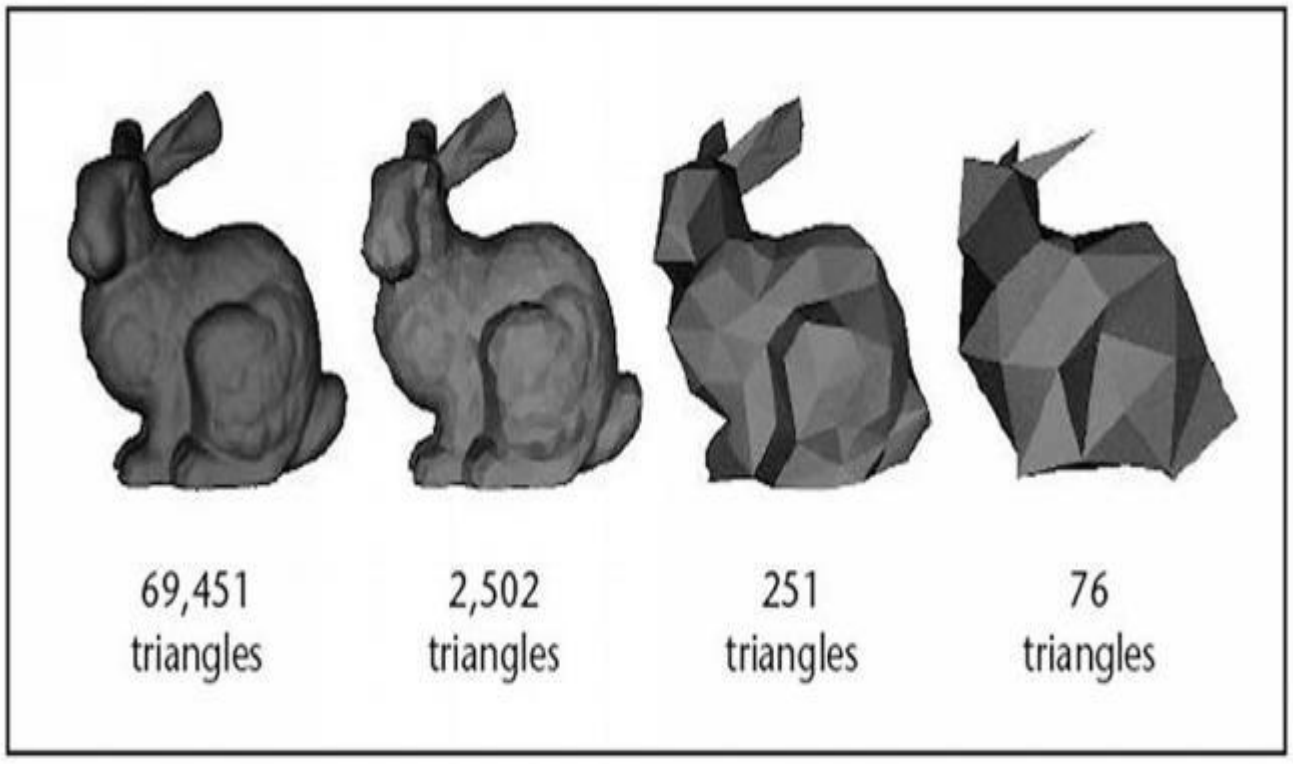


Рис. 1.2 Приклад використання LOD

Наприклад, щоб гарантувати відмінну точність і якість зображення, високий рівень деталізації застосовується до найближчих об'єктів. І навпаки, для віддалених об'єктів використовується нижчий ступінь деталізації, щоб максимізувати продуктивність системи та мінімізувати використання ресурсів.

Цей метод може значно підвищити ефективність рендерингу величезних сцен, зберігаючи при цьому якість найважливіших елементів сцени. Текстурування, каскадне освітлення та каскадне відображення тіней - ось деякі з методів, що використовуються в каскадному підході. Разом ці методи забезпечують найкращий можливий баланс між продуктивністю та якістю.

Cascaded Shadow Maps (CSM) рисунок 1.3 [3, с. 185-202, 4] — це метод створення тіней, який дозволяє створювати чудові тіні на об'ємному тлі. В основі методу лежить поділ зображення на багато каскадів, кожен з яких відповідає за рендеринг тіней для певного діапазону відстаней від камери. Рівень деталізації тіней вищий для об'єктів, що знаходяться ближче, і нижчий для об'єктів, що знаходяться далі.

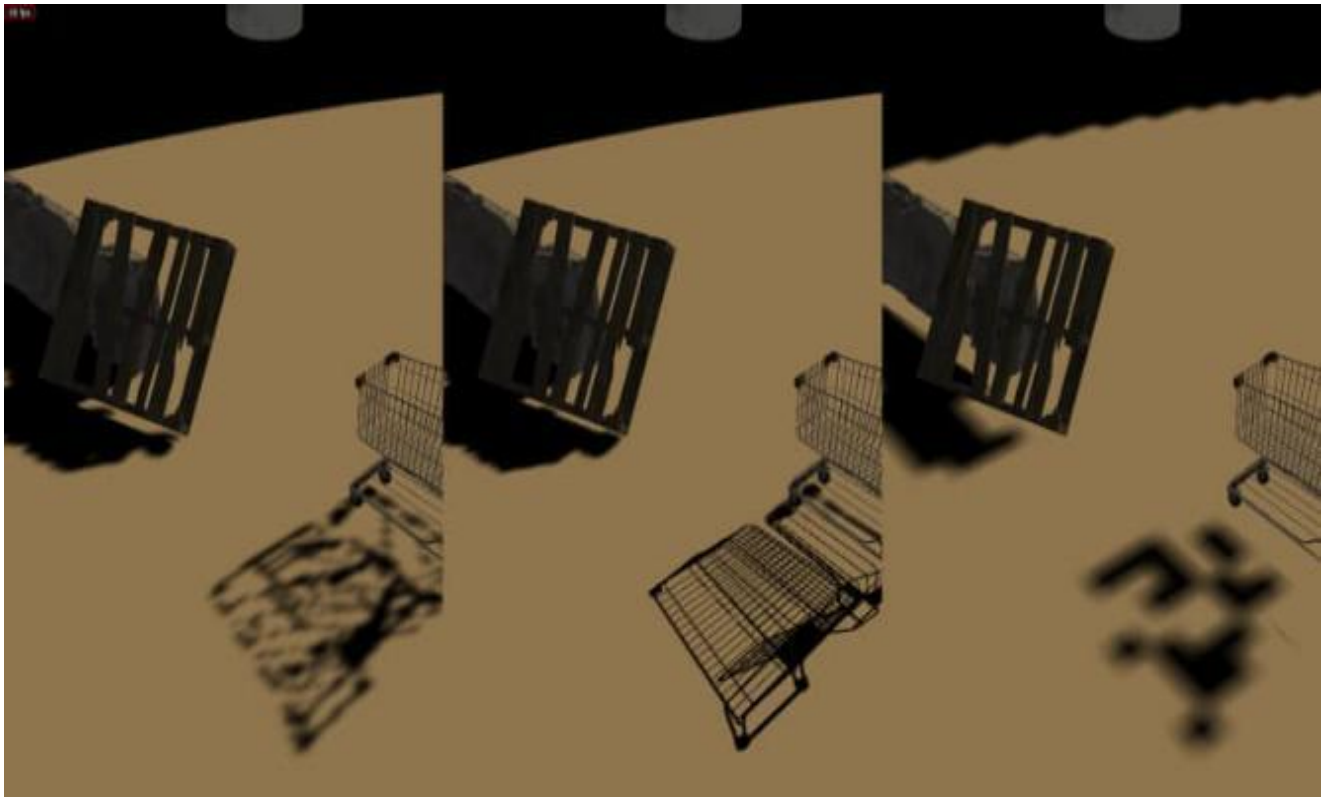


Рис. 1.3 Приклад каскадного підходу

Здатність створювати реалістичні тіні в розширених 3D-налаштуваннях є однією з головних переваг цього методу. Недоліком, однак, є потенційна можливість виникнення артефактів між каскадами, які можуть виникати, коли зміни між різними рівнями деталізації виглядають нетиповими. Щоб запобігти

виникненню таких артефактів, дуже важливо забезпечити плавний перехід між каскадами.

1.1.2 Адаптивний шейдинг

Адаптивний шейдинг рисунок 1.4 [5] — це підхід до рендерингу та шейдингу, який дозволяє налаштовувати стратегії обробки шейдерів на основі низки змінних, зокрема важливості об'єкта, видимості, відстані від камери та поточного навантаження на систему. Концепція адаптивного шейдингу полягає в тому, щоб збалансувати ефективність рендерингу та якість зображення, застосовуючи різні ступені обчислювальної складності до різних ділянок сцени або кадру.

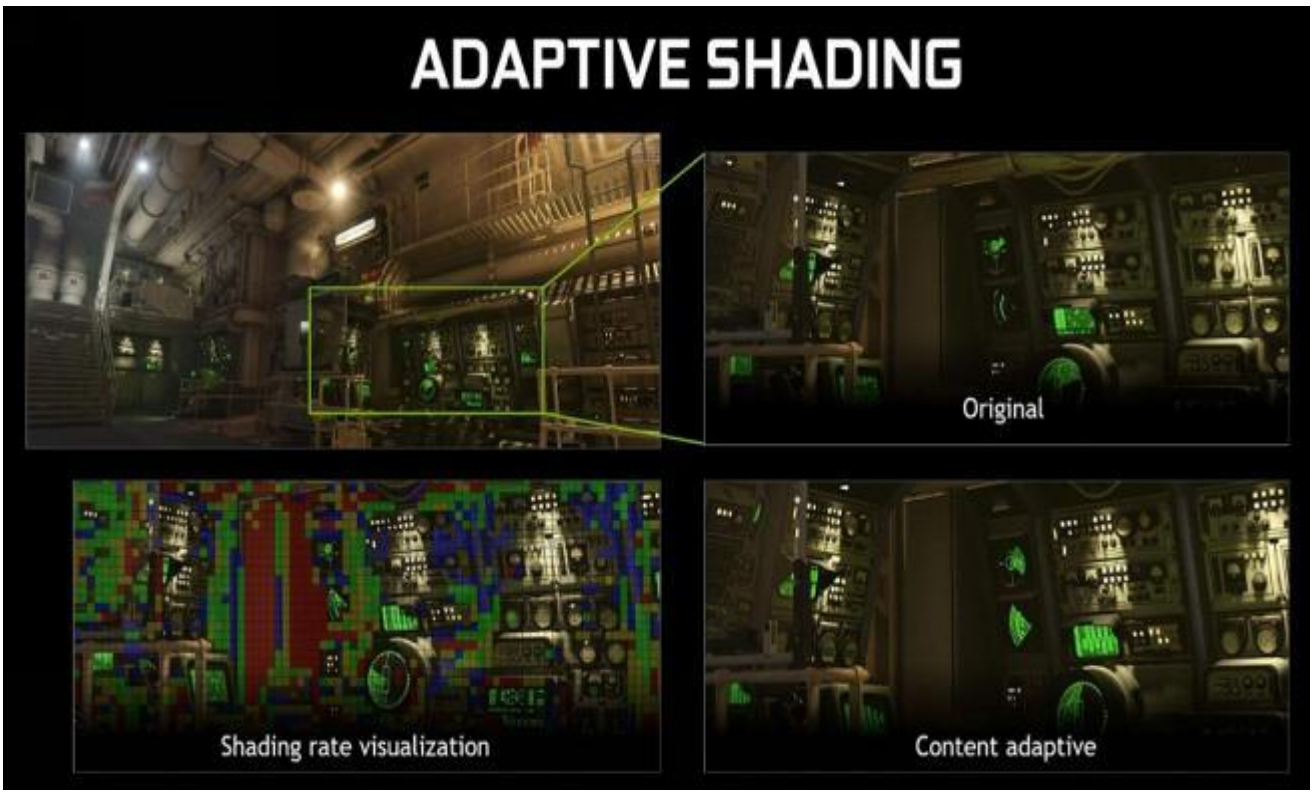


Рис. 1.4 Приклад адаптивного шейдингу

Цей метод дозволяє скоротити обчислення без помітного погіршення якості візуальних ефектів. Наприклад, потрібно застосувати більш складні техніки до значущих елементів або близьких об'єктів, використовуючи менш складні шейдери для віддалених об'єктів або взагалі опустити деякі кроки затінення. У

великих 3D-об'єктах адаптивне затінення часто використовується для максимального використання ресурсів, зберігаючи при цьому хорошу якість зображення для найважливіших елементів сцени.

Серед основних видів адаптивного шейдингу можна виокремити такі:

- Shader LOD
- Variable Rate Shading (VRS)
- Dynamic Shader LOD
- Adaptive Shading

Shader LOD рисунок 1.5 [6, с. 292-296] — це метод, який дозволяє регулювати складність шейдерів залежно від деталізації об'єкта або відстані до камери. Shader LOD, який можна порівняти з рівнями деталізації геометрії або текстур, дозволяє застосовувати різні шейдери на різних відстанях, що може значно зменшити навантаження на систему, особливо для віддалених об'єктів.

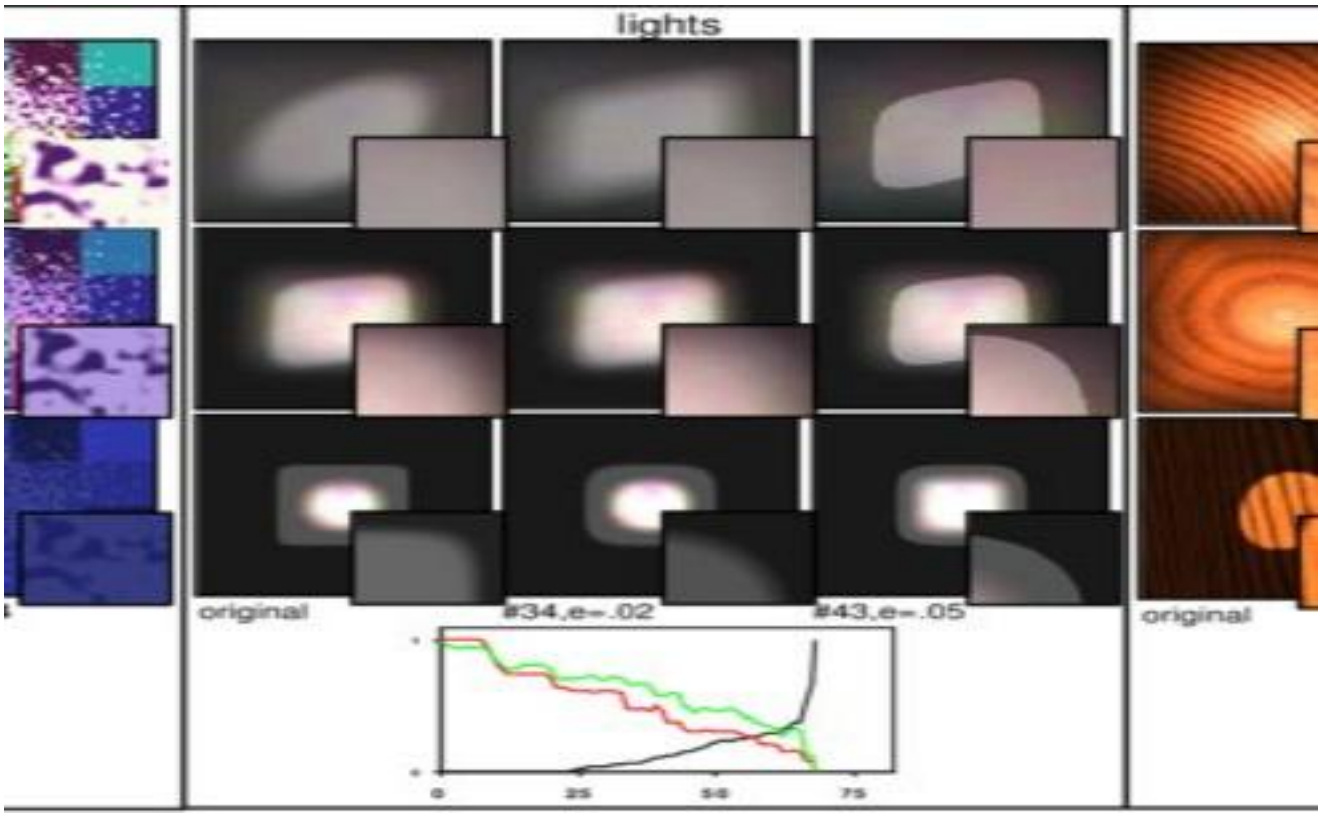


Рис. 1.5 Приклад рівнів деталізації шейдингу

Основна ідея Shader LOD полягає у застосуванні простіших шейдерів з меншими обчислювальними витратами до віддалених об'єктів або об'єктів, які є

менш важливими для сцени. Для речей у центрі уваги це дозволяє економити ресурси без різкого зниження візуальної якості.

Застосування Shader LOD демонструється в іграх, де ближчі об'єкти зображуються складнішими шейдерами, які включають додаткові ефекти, такі як відблиски, тіні або інші складні візуальні ефекти, в той час як тло зображується спрощеними шейдерами для навколишнього середовища.

Variable Rate Shading (VRS) рисунок 1.6 [7] — це передовий метод, який дозволяє вам регулювати частоту обчислення шейдерів на основі певних візуальних елементів. Є можливість налаштувати параметр тіні у VRS відповідно до важливості або видимості кожного елемента сцени.

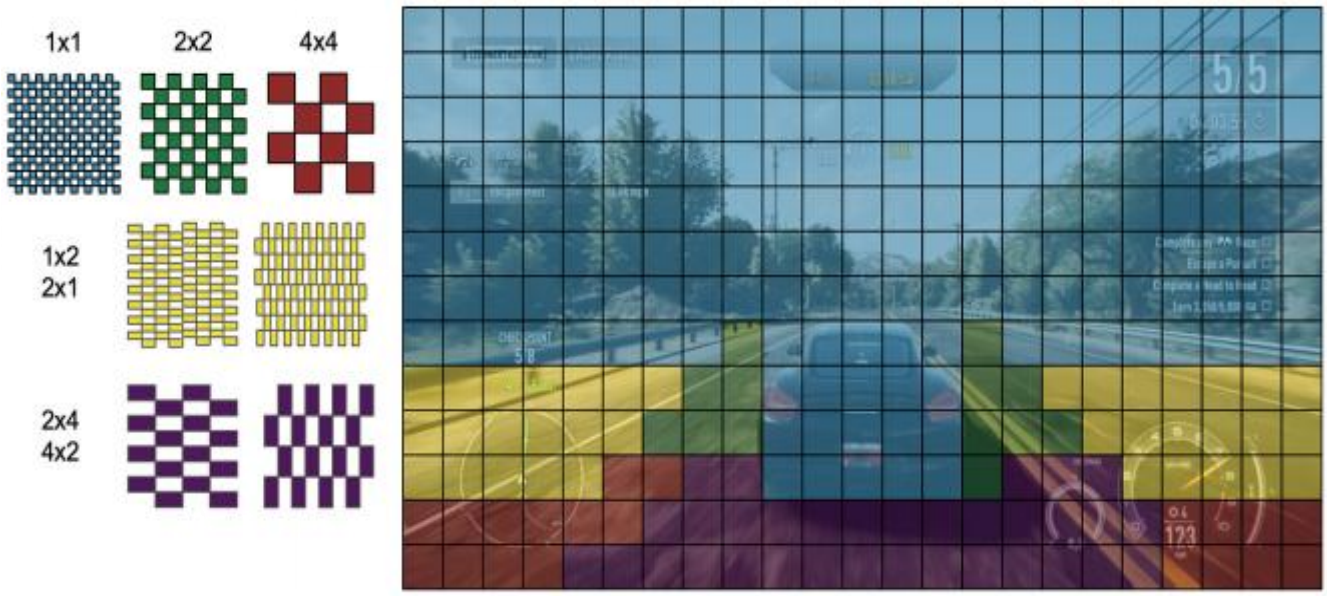


Рис. 1.6 Приклад роботи VRS

Фундаментальна ідея VRS полягає в тому, щоб знизити частоту обчислень для сцен, які є менш важливими або видимими, що зменшує навантаження на графічний процесор. Це дозволяє зменшити навантаження за рахунок використання меншої кількості обчислень шейдерів для віддалених або несуттєвих об'єктів, зберігаючи при цьому високу якість візуальних ефектів для найближчих або центральних елементів сцени.

Новітні відеокарти від AMD та NVIDIA активно підтримують VRS, забезпечуючи чудову продуктивність для деталізації візуальних ефектів та оптимізованого рендерингу в реальному часі.

Dynamic Shader LOD рисунок 1.7 — це метод, який дозволяє динамічно змінювати складність шейдерів під час рендерингу на основі інших змінних факторів або поточного завантаження системи. Це більш адаптивна версія Shader LOD, в якій контекст визначає рівень складності шейдерів у реальному часі, а не заздалегідь визначений.

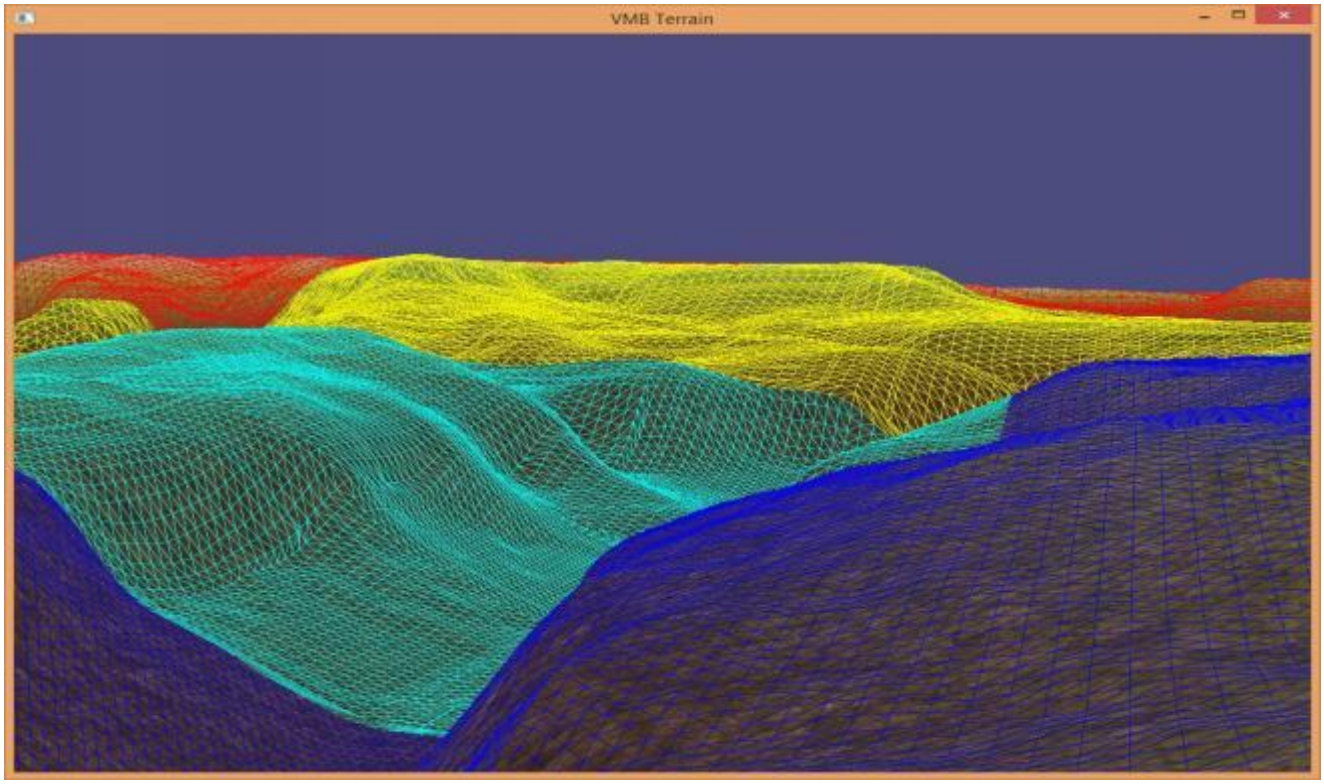


Рис. 1.7 Приклад роботи динамічних рівнів деталізації шейдингу

При рендерингу використовується Dynamic Shader LOD для адаптації до поточного стану системи. Наприклад, якщо система перевантажена обчисленнями або іншими завданнями і необхідно спростити шейдери для менш важливих ділянок сцени. Постійно вдосконалюючи процес рендерингу, можна гарантувати необхідну продуктивність у реальному часі.

Використовуючи цю техніку, можна продовжувати ефективно працювати з меншою кількістю ресурсів без шкоди для якості зображень найважливіших елементів сцени.

Adaptive Shading рисунок 1.8 [5] — це загальний підхід до оптимізації шейдерів, який передбачає пристосування алгоритмів шейдерів до потреб сцени. Цей підхід ґрунтується на тому, що обчислення шейдерів слід адаптувати до поточної сцени, застосовуючи різні підходи рендерингу до різних об'єктів сцени.

Рис. 1.8 Приклад роботи адаптивного шейдингу

Фундаментальна ідея адаптивного затінення полягає в тому, щоб зменшити обробку менш значущих ділянок сцени та прискорити обчислення для близьких або значущих об'єктів. Це дозволяє зберегти високу якість важливої інформації та оптимізувати продуктивність рендерингу.

1.1.3 Воксельні методи

Воксельні методи — це тип комп'ютерної графіки, який обробляє та зображує дані у просторі за допомогою тривимірних пікселів, або вокселів. У сучасній візуалізації цей метод має кілька важливих застосувань, зокрема, при роботі з глобальним освітленням і тим, як світло взаємодіє з об'єктами.

Основними видами воксельних методів є:

- Voxel-based Global Illumination
- Signed Distance Field Global Illumination

Voxel-based Global Illumination рисунок. 1.9 [1, с. 470-475, 9, с. 358-362, 10] — це техніка глобального освітлення, яка полегшує розрахунок взаємодії світла з поверхнями, використовуючи вокселі як основну одиницю зберігання інформації про світло. Використовуючи цю техніку, сцена розбивається на тривимірну сітку вокселів, кожен з яких має дані про освітлення, відбиття та поглинання.

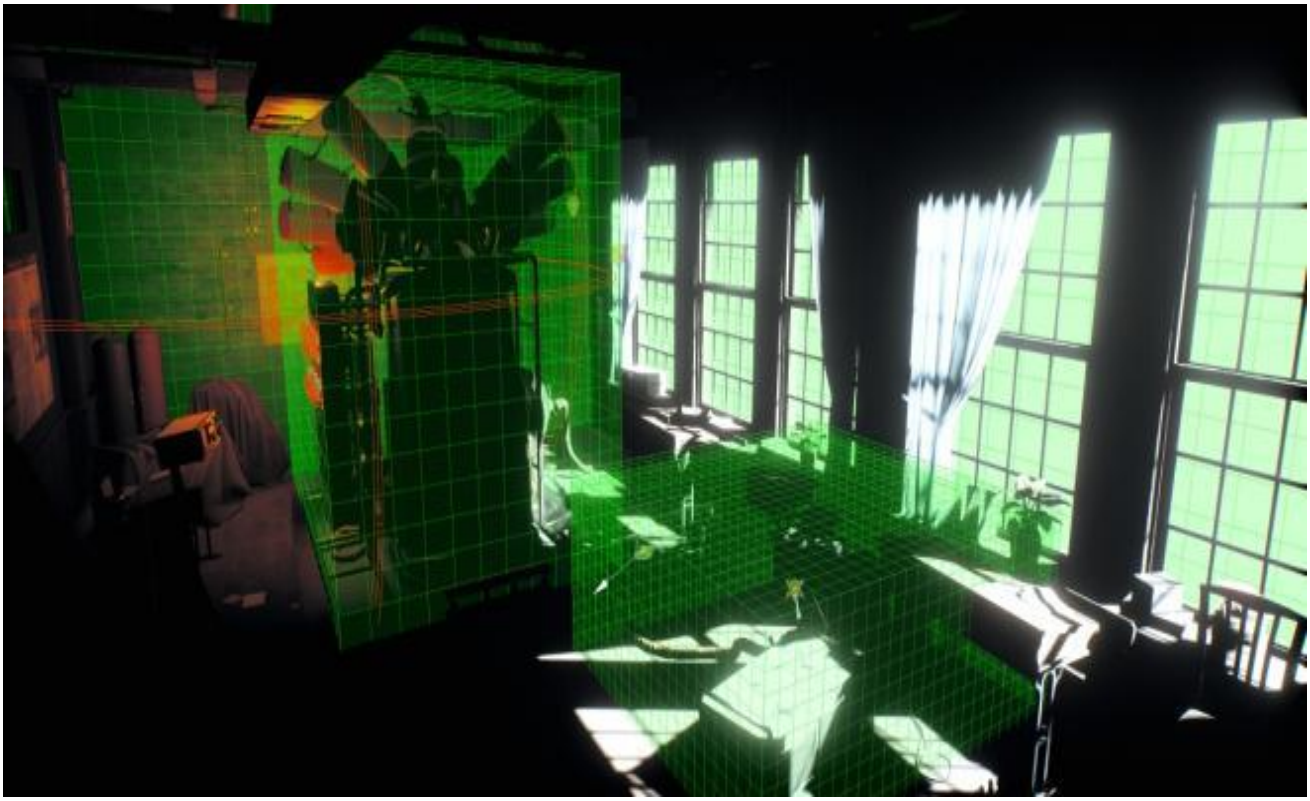


Рис. 1.9 Приклад сцени з Voxel-based Global Illumination

Цей метод створює реалістичне освітлення навіть для складних і об'ємних сцен і ефективно працює в реальному часі. Цей метод знижує витрати на обробку за рахунок агрегування даних на рівні вокселів, що може бути використано в ігрових рушіях для створення високоякісних зображень. Перевагами цього методу є менша складність обробки, краще глобальне освітлення та підтримка динамічних змін у сцені, таких як рух об'єктів або зміна освітлення.

Signed Distance Field Global Illumination рисунок 1.10 [8, с. 622-630] — це більш сучасна методика, яка використовує SDF для моделювання геометрії об'єкта та взаємодії зі світлом. Математична модель SDF описує відстань між кожною точкою в просторі та найближчою поверхнею об'єкта. За допомогою цієї інформації моделюється рух світла по сцені, щоб забезпечити реалістичні ефекти розсіювання та відбиття.

Оскільки він дозволяє працювати з найдрібнішими аспектами геометрії, цей підхід добре підходить для складних сценаріїв з великою кількістю об'єктів. Переваги цього методу включають зменшення світлових артефактів у складних сценаріях, ефективну роботу зі складними об'єктами та точне моделювання взаємодії світла з геометрією.

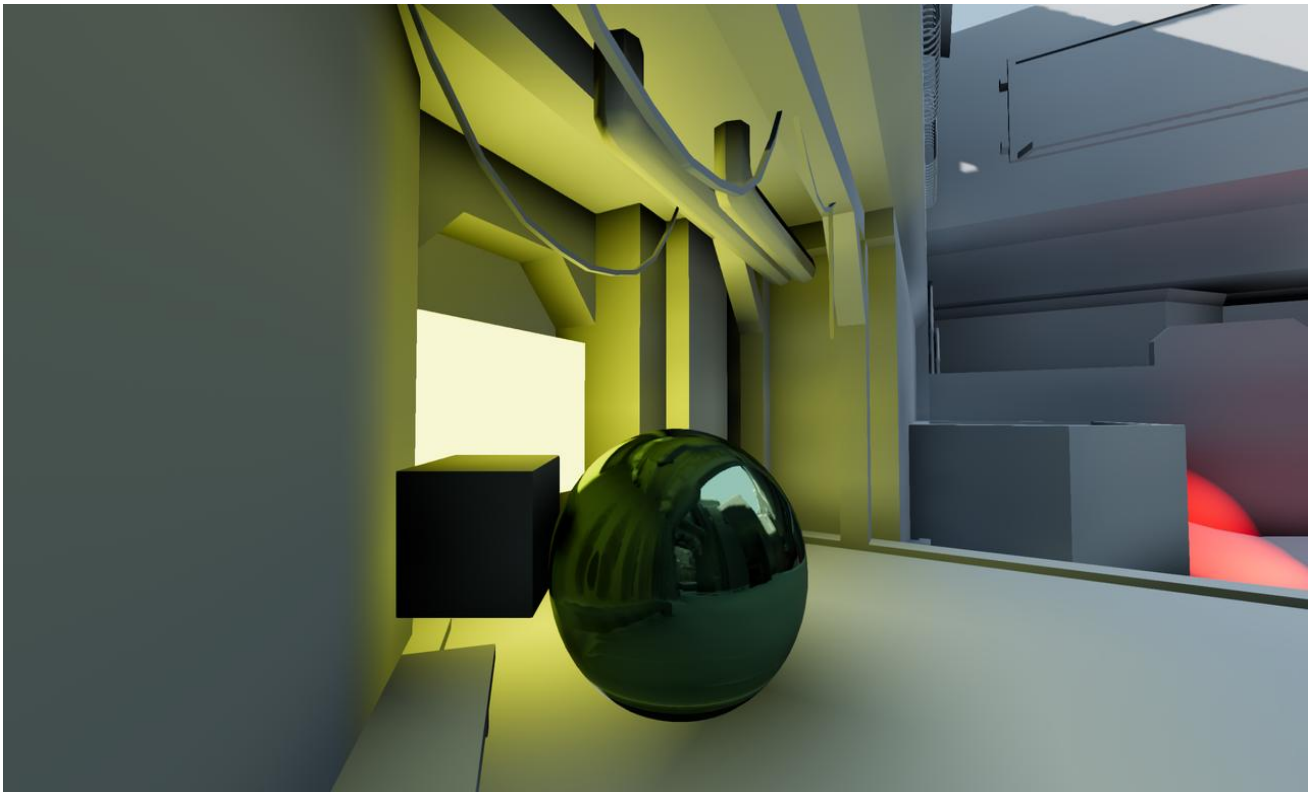


Рис. 1.10 Приклад роботи SDFGI

1.1.4 Динамічне освітлення та тіні

Ключовим елементом рендерингу є динамічне освітлення і тіні, які дозволяють створювати реалістичні візуальні ефекти, змінюючи освітлення в реальному часі. Основна мета полягає в імітації взаємодії світла з елементами

сцени, включаючи тіні, які змінюються залежно від розташування об'єктів і джерела світла. Цей метод часто використовується в сучасних 3D-іграх та візуалізаціях, де точність і динамічність мають вирішальне значення.

Існують кілька основних методів динамічного освітлення та тіней:

- Real-Time Dynamic Shadows
- Light Propagation Volumes (LPV)

Real-Time Dynamic Shadows рисунок 1.11 [11, с. 56-89, 14, с. 101-135] — це метод рендерингу тіней, який дозволяє показати, як розташування та форма об'єктів змінюються залежно від їхнього руху та джерела світла. Мапування тіней - це основна техніка реалізації, в якій створюється карта тіней від кожного джерела світла і проектується на сцену.

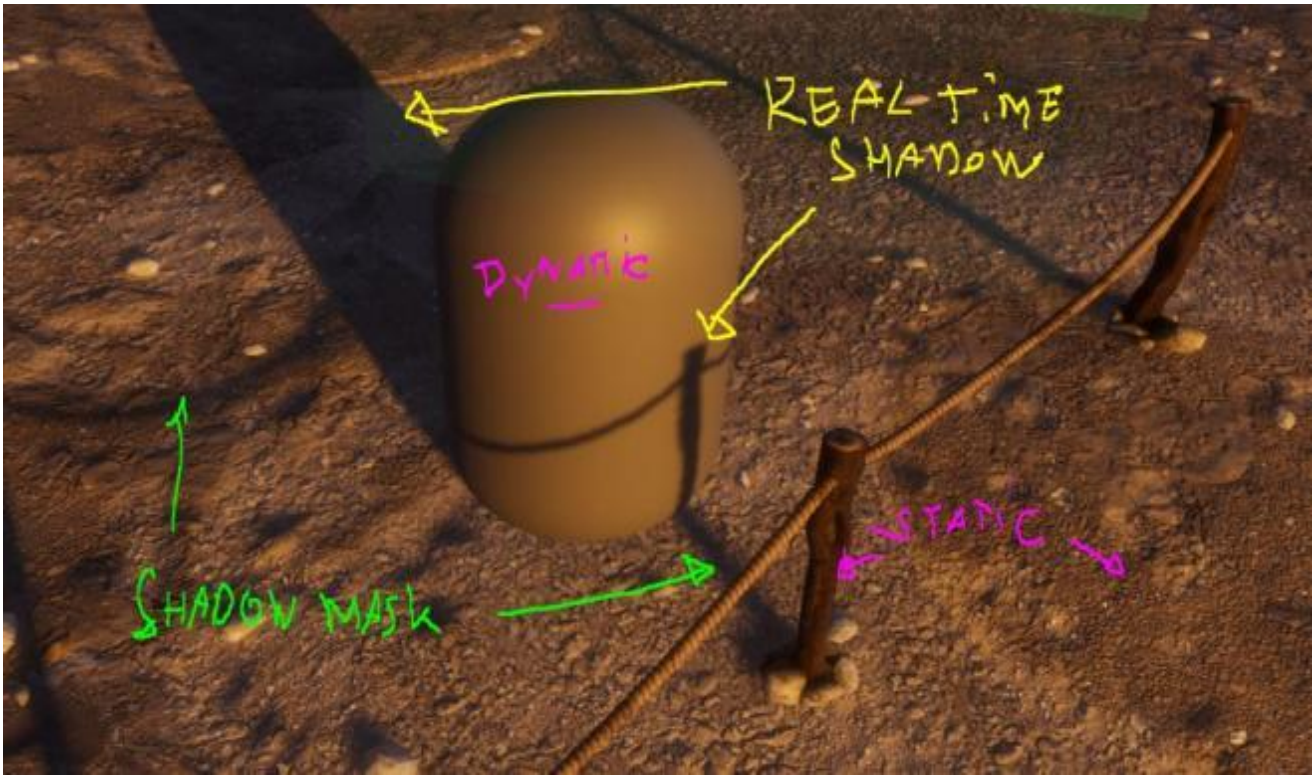


Рис. 1.11 Приклад динамічних тіней в реальному часі

Об'єкти, розташовані ближче до камери, мають вищу роздільну здатність тіней, тоді як віддалені об'єкти мають нижчу роздільну здатність тіней для оптимальної якості. Хоча ця техніка створює тіні з великою чіткістю та деталізацією, вона може мати проблеми, такі як артефакти аліасингу. Об'єми тіней - це додатковий вибір, який визначає геометрію об'ємів, що блокують світло. Хоча

цей метод дає точніший результат, він вимагає великих обчислювальних потужностей.

Light Propagation Volumes (LPV) рисунок 1.12 [12, с. 132-165, 13] — це воксельний підхід до моделювання глобального освітлення. Світло проходить через тривимірну сітку вокселів, кожен з яких має пам'ять про колір та інтенсивність світла. М'яке розсіювання, віддзеркалення від поверхні та взаємодія світла з навколишнім середовищем - все це можна врахувати таким чином. Великі сцени та реалістичні ефекти добре підходять для LPV, однак деталізація може постраждати через сильний вплив роздільної здатності сітки на якість.

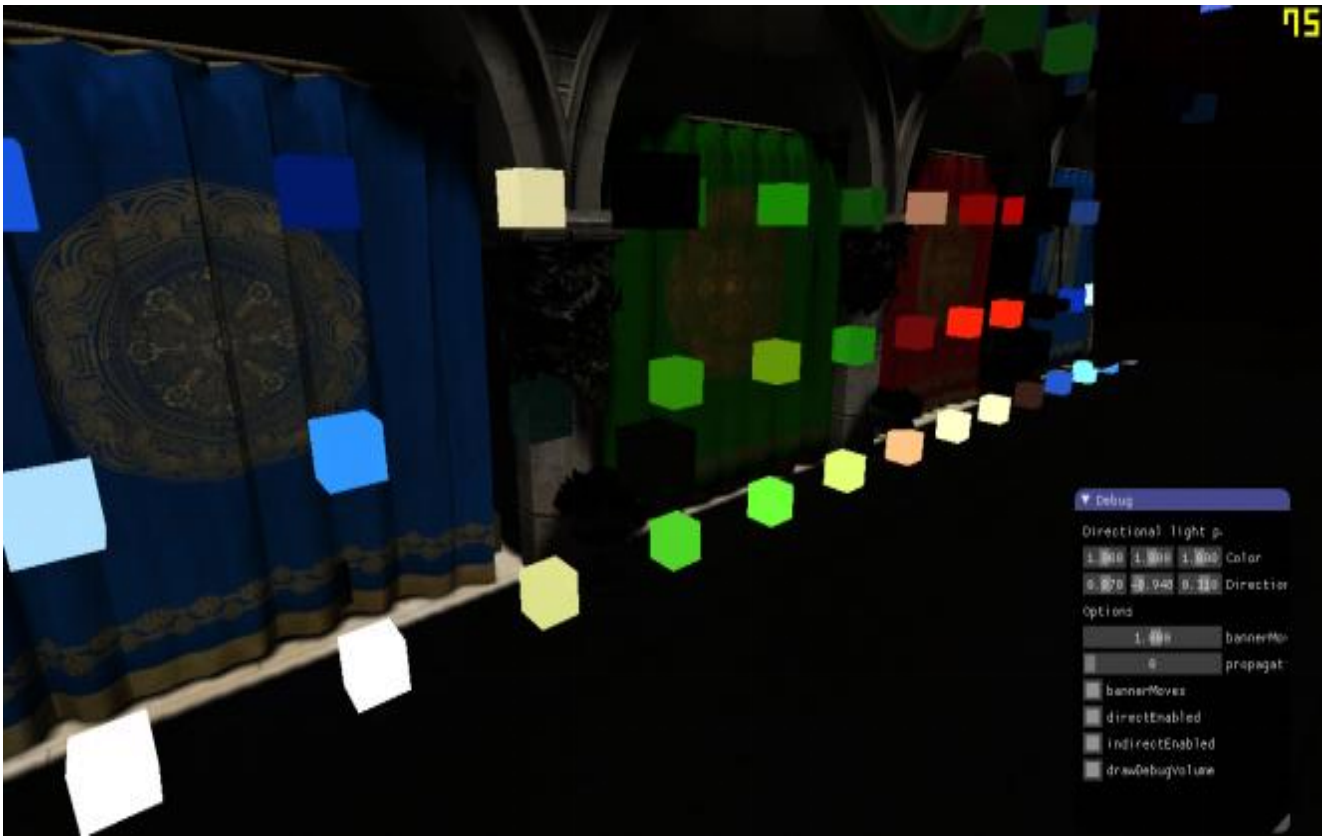


Рис. 1.12 Приклад моделювання глобального освітлення

Навіть у складних ситуаціях можете створювати реалістичні та якісні візуалізації з динамічним освітленням і тінями. Потреби проекту та доступні обчислювальні ресурси визначають, який підхід є найкращим. Наприклад, LPV найкраще працює на великих відкритих просторах, а динамічні тіні в реальному часі краще підходять для компактних ситуацій з точковими джерелами світла. Обидва методи є важливими для створення візуально захоплюючих вражень.

1.2 Аналіз існуючих методів рендерингу

1.2.1 Відкладений рендеринг

Відкладений рендеринг — це сучасний метод обробки графіки, який дозволяє ефективно працювати зі складними сценами з великою кількістю джерел світла. Центральною концепцією є поділ процесу рендерингу на дві основні фази. Спочатку використовуються спеціальні буфери, так звані G-буфери, для зберігання геометричної та матеріальної інформації про сцену, включаючи нормалі, кольори та глибину. Потім на основі збережених даних визначається освітлення.

Цей метод дозволяє обробляти сцени з відмінною продуктивністю навіть за наявності великої кількості джерел світла, оскільки він значно оптимізує використання ресурсів графічного процесора (GPU).

Основні види відкладеного рендерингу:

- Deferred Shading
- Clustered Shading
- Forward Rendering

Deferred Shading рисунок 1.13 [15, с. 34-78] — Це найпростіший і найпоширеніший тип відкладеного рендерингу. Початковий крок цього методу полягає у зберіганні геометричних даних про сцену в G-буфері, включаючи нормалі, кольори, глибину та іншу інформацію. Вся ця інформація використовується для визначення освітлення на другому етапі.

Цей метод особливо добре працює в ситуаціях з великою кількістю джерел світла, оскільки усуває необхідність у нудних обчисленнях, виконуючи процес освітлення після того, як всі геометричні дані вже зібрані. Однак цей метод вимагає багато пам'яті, оскільки він повинен генерувати і зберігати величезні буфери (G-буфер).

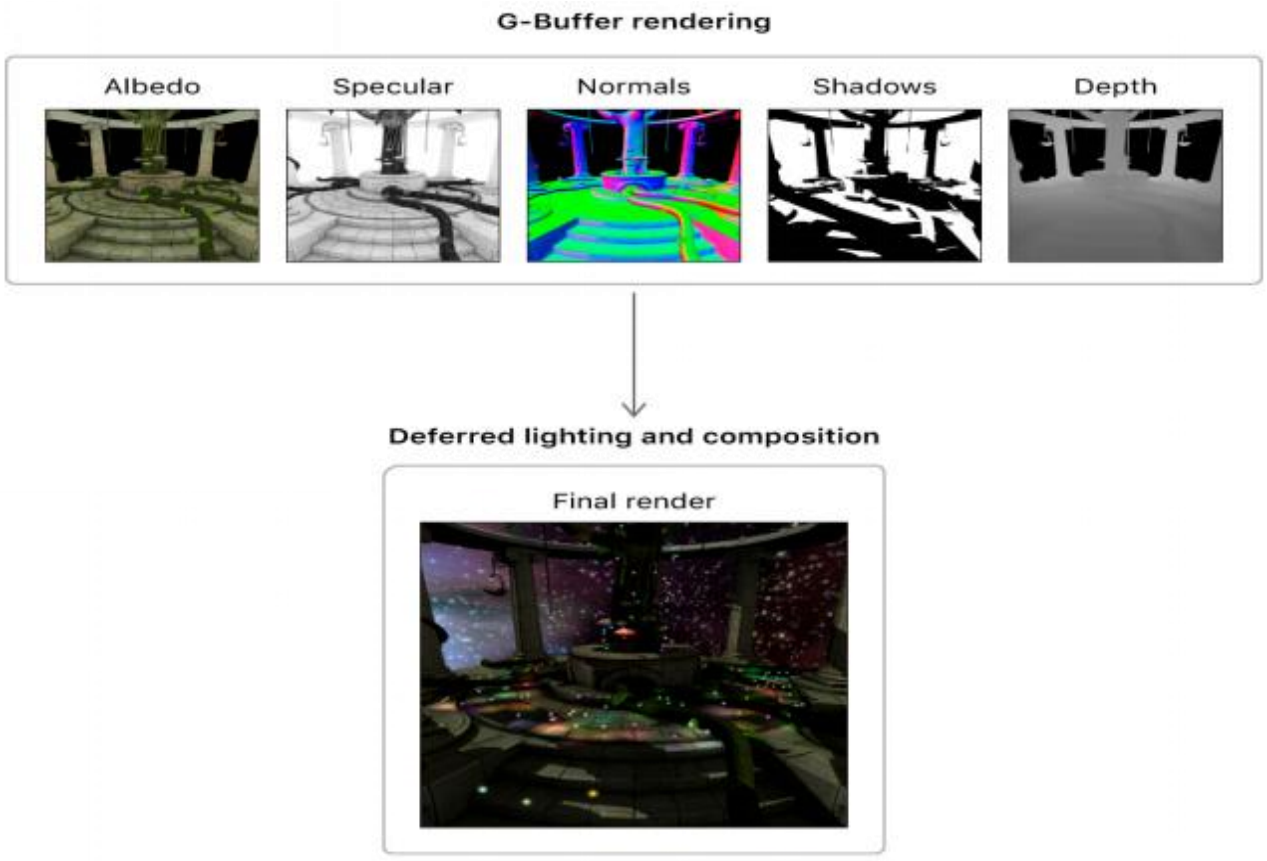


Рис. 1.13 Приклад відкладеного рендерингу

Робота з напівпрозорими об'єктами ще одне обмеження відкладеного затінення. Прозорість складно використовувати в деяких ситуаціях, оскільки вона вимагає багаторівневої обробки даних, що робить її реалізацію складною та ресурсоємною.

Clustered Shading рисунок 1.14 [16, с. 15-40] — вирішує проблему обчислення масштабування у сценах з сотнями або навіть тисячами джерел світла, що робить його кращим за відкладене зафарбовування (Deferred Shading). Цей метод розділяє сцену на тривимірну сітку кластерів, кожен з яких представляє певну частину простору. Джерело світла отримують лише ті кластери, на які вони впливають.

Оскільки кожен кластер обробляє тільки ті джерела світла, які мають до нього відношення, цей метод значно зменшує кількість обробки, необхідної для освітлення. Завдяки цьому кластерне затінення неймовірно добре працює в ситуаціях з великою кількістю світлових ефектів.

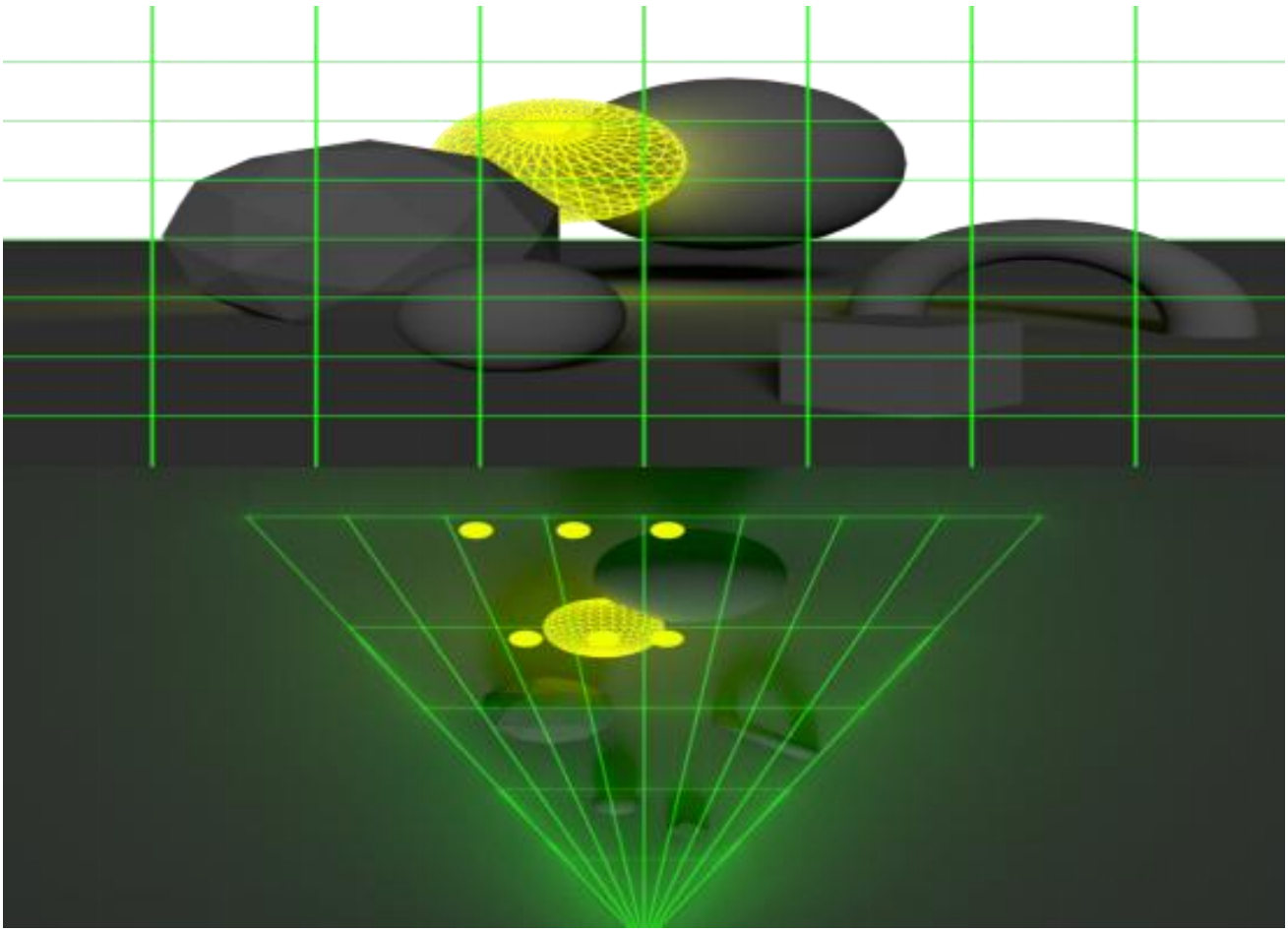


Рис. 1.14 Приклад кластерного рендерингу

Втім, впровадження Clustered Shading є технічно складним, а його реалізація потребує значних апаратних ресурсів. Висока точність у розрахунках кластеризації також може створювати певні обмеження для апаратури, особливо у старих або менш продуктивних GPU.

Forward Rendering рисунок 1.15 [17] — це традиційна техніка рендерингу, яка враховує всі джерела світла при обробці кожного елемента на етапі малювання. Цей метод полегшує побудову напівпрозорих об'єктів, оскільки не передбачає попереднього збереження інформації в буферах.

Елементи відкладеного рендерингу, зокрема оптимізація освітлення або використання спеціальних буферів для певних об'єктів, часто супроводжують прямий рендеринг у сучасних реалізаціях. Це зберігає ключові переваги методу, водночас дозволяючи вам краще працювати в умовах складного освітлення.

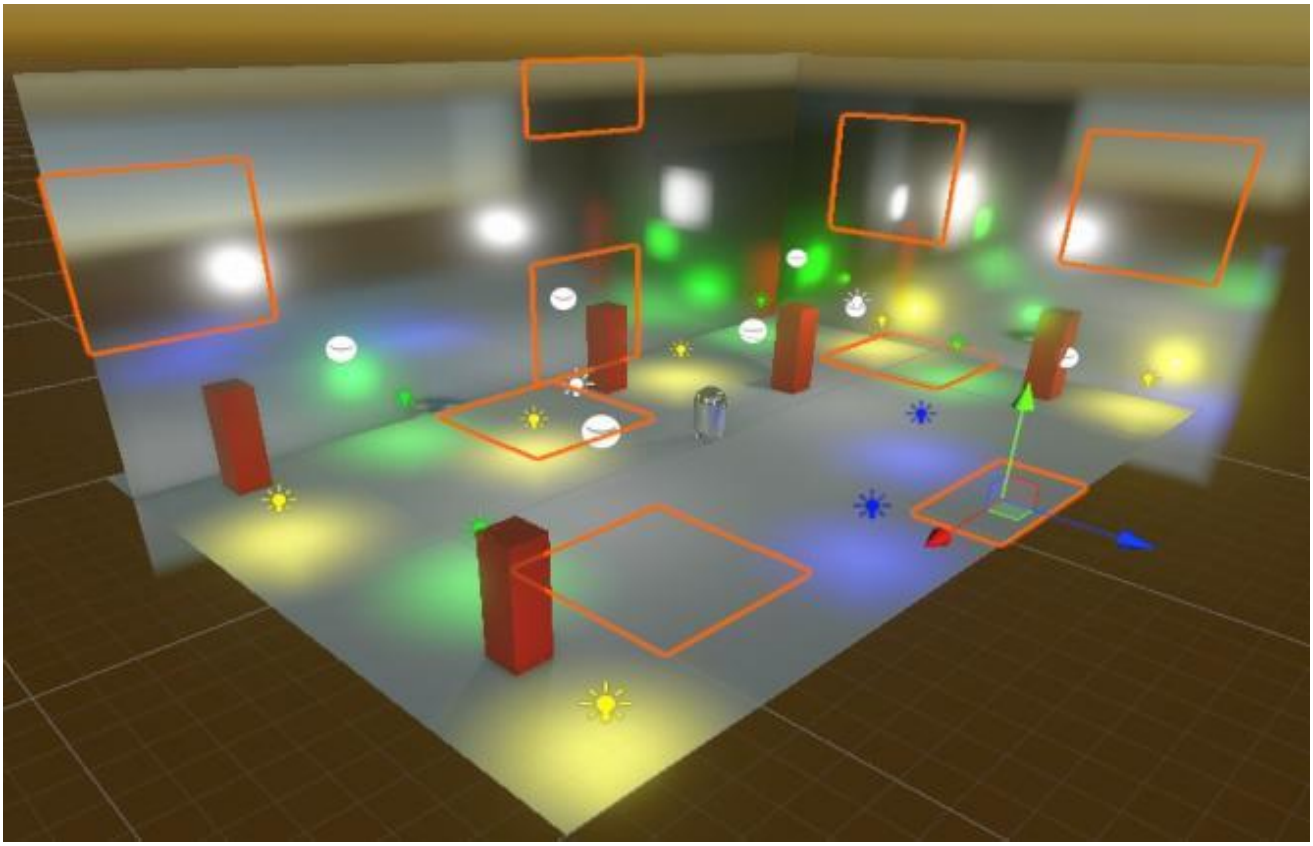


Рис. 1.15 Приклад лицевого рендерингу

Для ігор з великою кількістю деталей і сцен з великою кількістю напівпрозорих елементів найкращим варіантом є Forward Rendering. Застосування цього методу у великомасштабних сценах обмежується його меншою ефективністю при роботі з різними джерелами світла.

1.2.2 Масштабування кадру

Масштабування кадру рисунок 1.16 — це метод, який дозволяє підвищити продуктивність рендерингу шляхом масштабування зображення до потрібної роздільної здатності після зниження роздільної здатності рендерингу. Зараз це важливий компонент сучасних ігрових і графічних технологій, який допомагає досягти балансу між частотою кадрів (FPS) і якістю зображення. Масштабування особливо важливе для пристроїв з малою апаратною потужністю та для ігор з високою роздільною здатністю, наприклад, 4K, де оптимізація ресурсів має важливе значення.

Рис. 1.16 Приклад апскейлінгу

Основні методи масштабування включають такі технології:

- Deep Learning Super Sampling (DLSS)
- FidelityFX Super Resolution (FSR)
- Xe Super Sampling (XeSS)
- NVIDIA Image Scaling (NIS)
- Temporal Anti-Aliasing (TAA)

Deep Learning Super Sampling (DLSS), зображений на рисунку 1.17 [18] — це передова технологія, розроблена компанією NVIDIA, яка створює високоякісні зображення з початково низькою роздільною здатністю, використовуючи сучасні методи машинного навчання та штучного інтелекту. Ця технологія забезпечує не лише покращення візуальної якості, але й оптимізацію продуктивності, що особливо важливо для ресурсоємних додатків, таких як сучасні відеоігри.

Основною характеристикою DLSS є можливість реконструювати зображення завдяки застосуванню спеціально навчених нейронних мереж. Ці мережі проходять попереднє навчання на великих наборах даних, які включають пари зображень низької та високої роздільної здатності. У процесі роботи технологія використовує ці знання для створення якісного зображення, яке виглядає так, ніби воно було рендерене з високою роздільною здатністю.

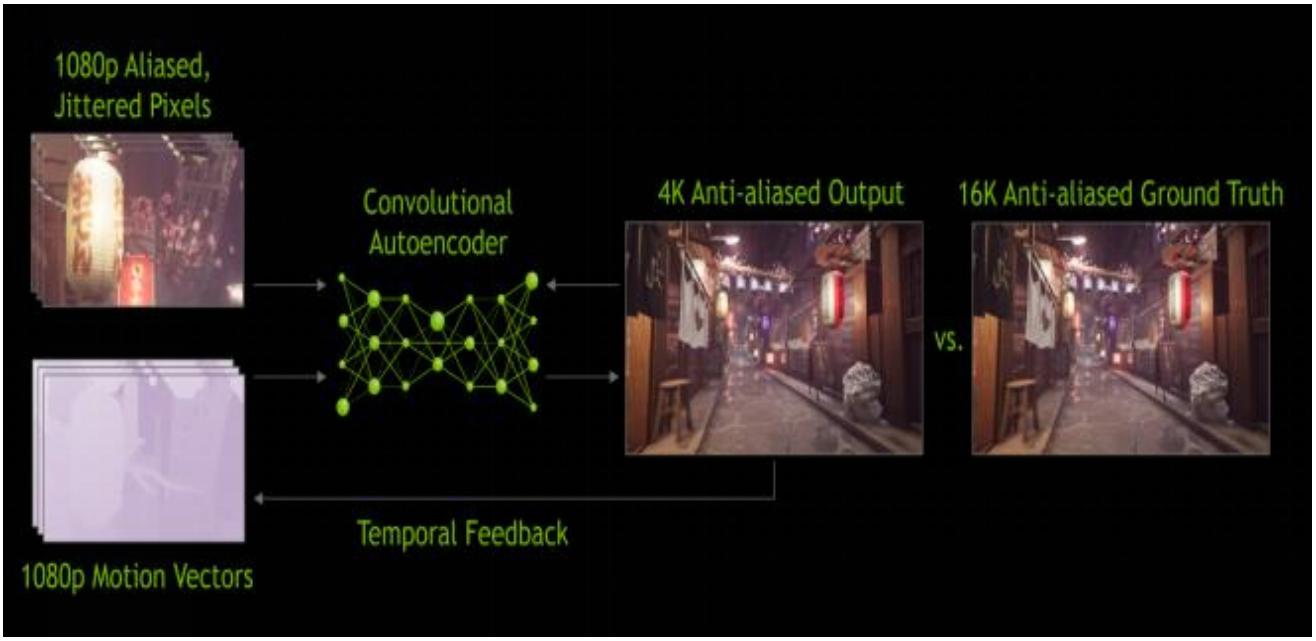


Рис. 1.17 Приклад роботи DLSS

Робота DLSS включає кілька етапів: спершу гра рендериться у нижчій роздільній здатності, потім алгоритм відновлює деталі та текстури, додаючи відсутню інформацію. З виходом DLSS 3.0 ця технологія отримала додаткову функцію генерації проміжних кадрів рисунок 1.18 [19], що дозволяє значно підвищити частоту кадрів без впливу на основну якість зображення.

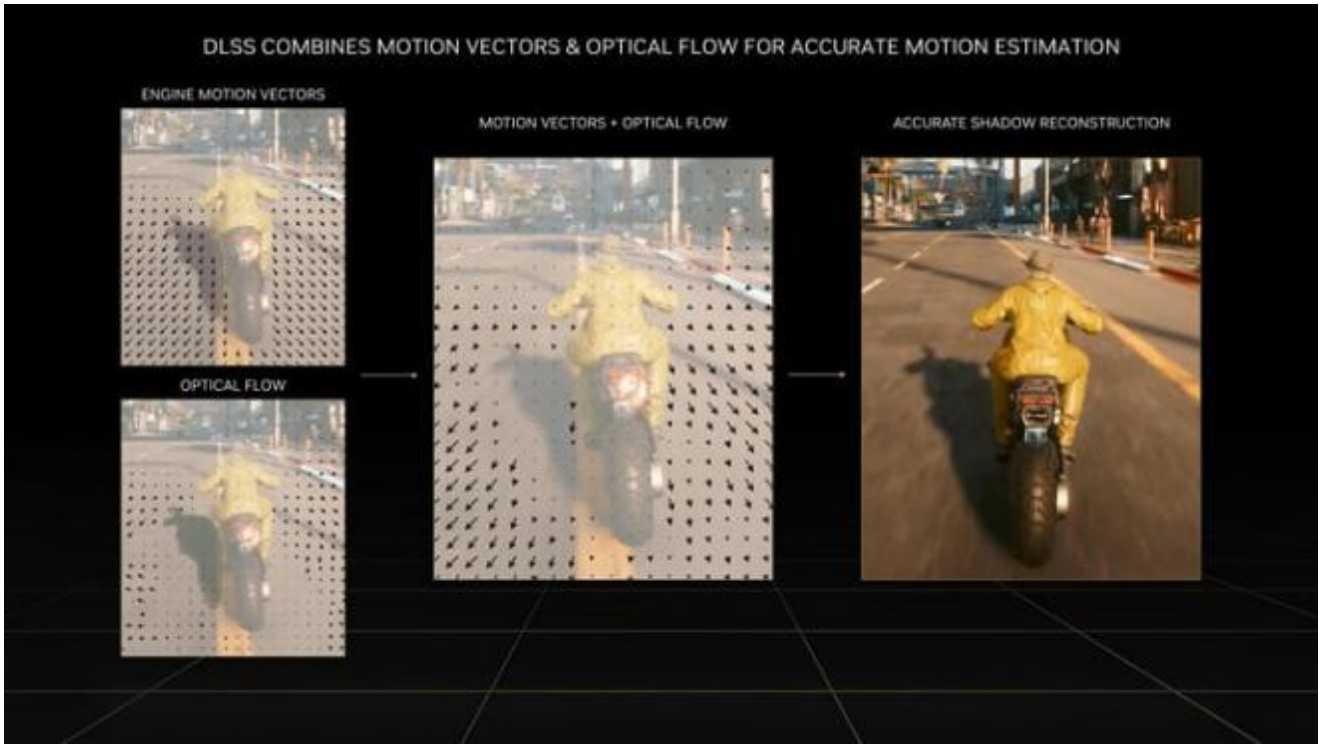


Рис. 1.18 Прорахунок траєкторії руху кадра

Екосистема NVIDIA знаходиться в центрі уваги DLSS, зокрема відеокарти серії RTX з тензорними ядрами, які мають вирішальне значення для обробки нейронних мереж. Це обмежує доступність технології, але її результати виправдовують такі вимоги, а саме високоякісне масштабування з невеликою кількістю артефактів.

FidelityFX Super Resolution (FSR) рисунок 1.19 [20-21] — це технологія AMD, яка базується на алгоритмах масштабування та покращення чіткості. На відміну від DLSS, FSR не потрібне апаратне рішення, що робить її більш універсальною та доступною для широкого спектру графічних процесорів, включаючи конкурентні продукти NVIDIA.

Рис. 1.19 Порівняння ефективності FSR

Оскільки FSR має відкритий вихідний код, розробники можуть легко інтегрувати цю технологію у свої додатки. Завдяки цій функціональності технологію можна використовувати в більшій кількості ігор і платформ. Технологія дозволяє користувачам вибрати ідеальний компроміс між якістю та продуктивністю, пропонуючи кілька рівнів налаштувань, включаючи ультра-якість, якість, збалансованість та продуктивність.

Незважаючи на те, що FSR забезпечує чудові результати для свого класу, вона не дотягує до DLSS за точністю рендерингу та відновленням дрібних деталей, особливо в сценах з великою кількістю руху.

Xe Super Sampling (XeSS) рисунок 1.20 [22] — це технологія масштабування від Intel, яка поєднує традиційні методи покращення зображення з підходом машинного навчання. Хоча XeSS орієнтована на відеокарти Intel Arc, це універсальне рішення, яке також працює з іншими графічними процесорами, сумісними з Shader Model 6.4.

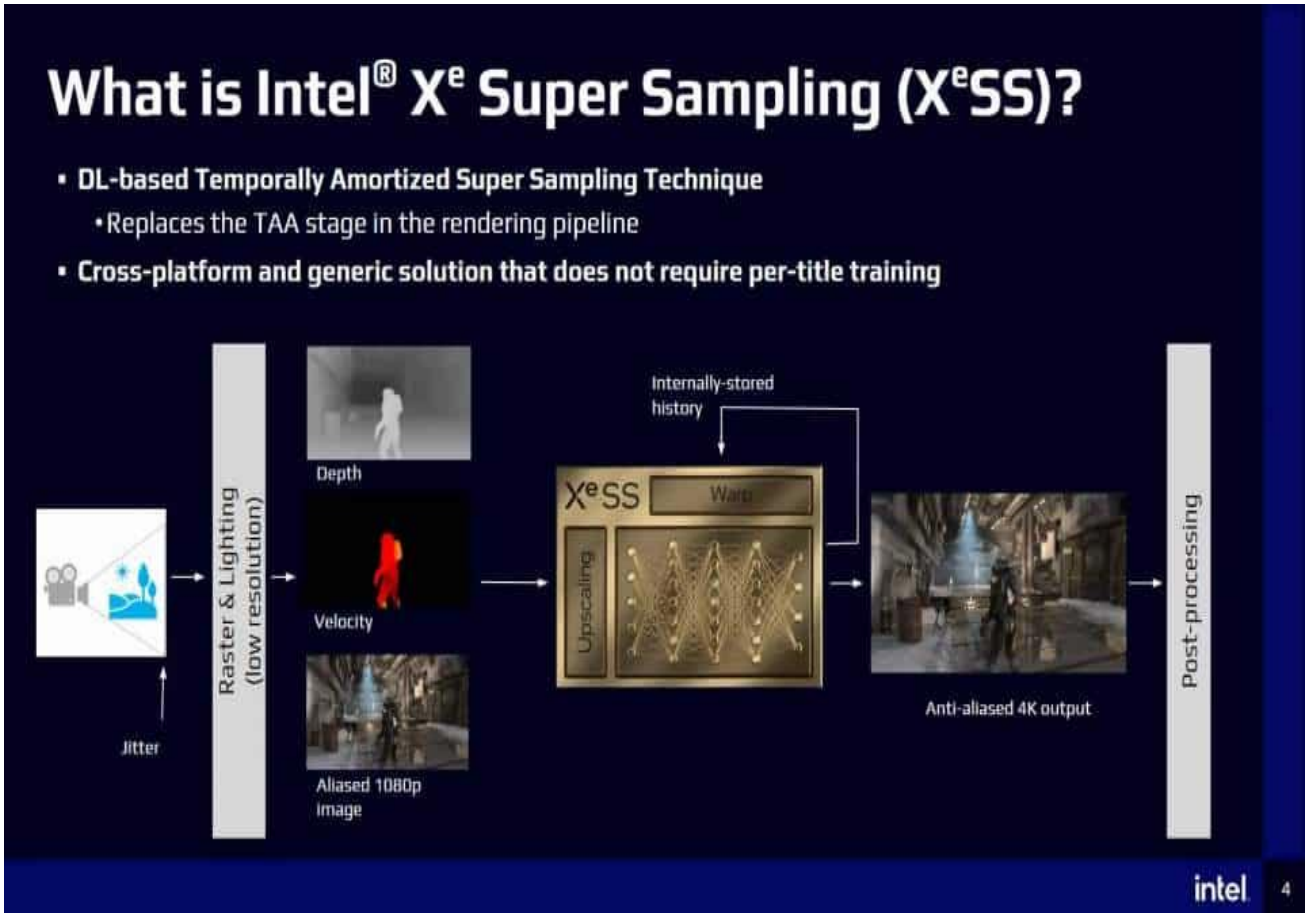


Рис. 1.20 Стуктура роботи XeSS

Подібно до DLSS, XeSS рендерить зображення з нижчою якістю, а потім використовує нейронні мережі для його відновлення. Продуктивність і поширення цієї технології залежить від того, як вона використовується в конкретних іграх, хоча вона створює зображення з якістю, порівнянною з NVIDIA DLSS.

NVIDIA Image Scaling (NIS) [23] — це заміна DLSS для користувачів попередніх моделей графічних процесорів NVIDIA. Цей підхід до масштабування простий і не вимагає спеціального обладнання. Для отримання чіткішого зображення з меншою складністю обробки NIS використовує концепцію фільтрації Ланчоса.

Хоча ця технологія добре працює зі старими графічними процесорами, вона не настільки добре відновлює деталі, як більш сучасні методи, такі як DLSS або FSR. NIS часто використовується як простий, але потужний метод підвищення FPS.

Temporal Anti-Aliasing (TAA) рисунок 1.21 [24] — це метод згладжування, який також використовується для масштабування зображень. Цей метод зменшує ефект "зубчастих країв" за рахунок об'єднання інформації з кількох попередніх кадрів. Завдяки цьому технологія підвищує чіткість у динамічних сценах.

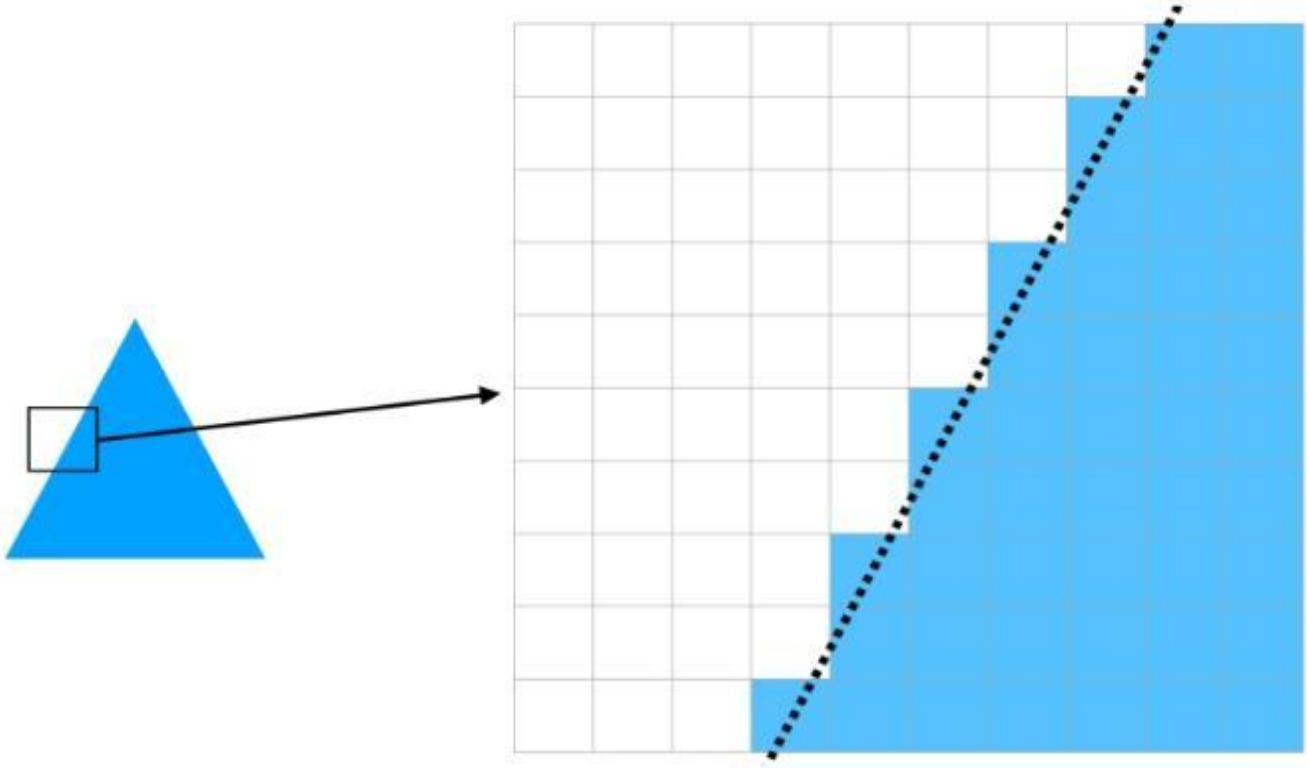


Рис. 1.21 Принцип роботи TAA

Основною перевагою TAA є її сумісність із сучасними ігровими рушіями. Однак у деяких випадках TAA може створювати артефакти, такі як розмитість зображення або фантомні контури, особливо у сценах із швидким рухом.

1.3 Опис використаних програмних засобів

Для виконання завдань, поставлених у рамках дипломної роботи, було використано набір сучасних програмних інструментів, кожен з яких був необхідний для виконання певних компонентів проекту. При виборі інструментів були враховані особливості завдання, яке включало в себе створення першокласного графічного середовища, розробку методів розрахунку освітлення та їх інтеграцію в програмне середовище.

Unity рисунок 1.22 [25] — це відомий ігровий рушій, який пропонує всі можливості, необхідні для створення інтерактивних ігор та додатків. Unity виявився найкращим варіантом для роботи з алгоритмами освітлення та оптимізації швидкості завдяки підтримці сучасних графічних API, таких як DirectX та Vulkan.

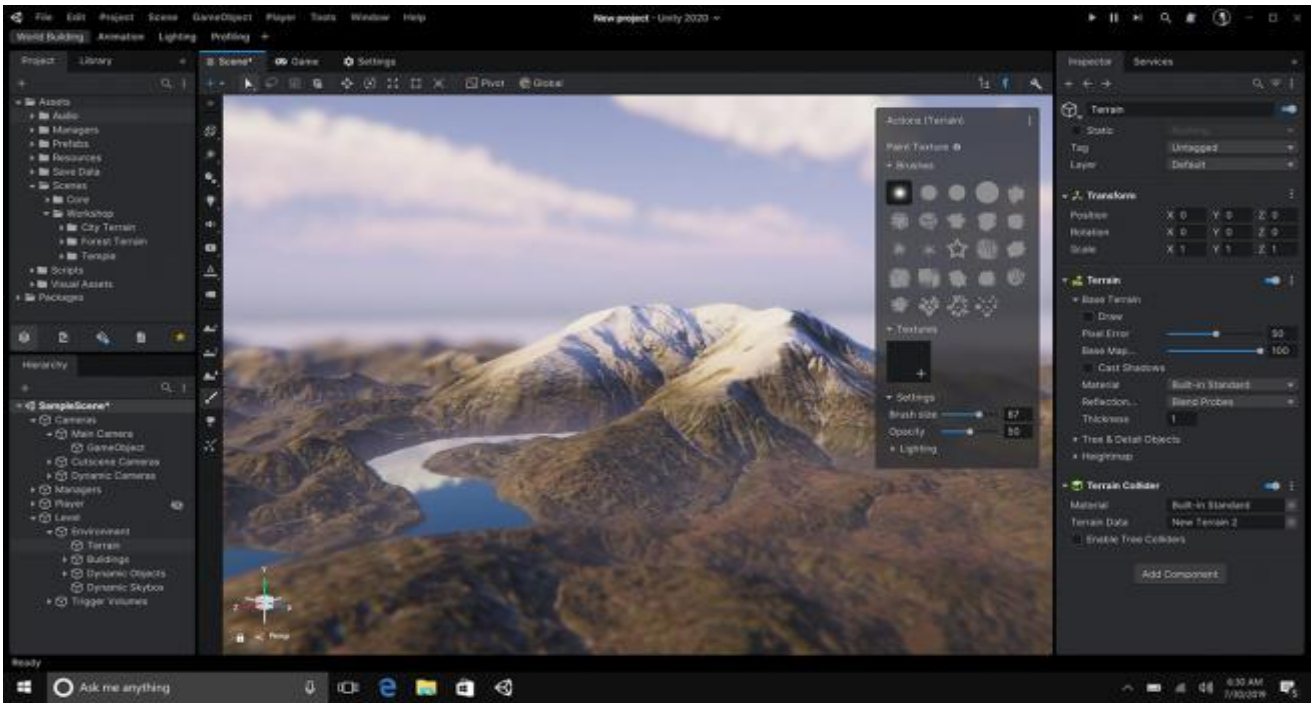


Рис. 1.22 Інтерфейс двигуна Unity

Unity дозволяє працювати з тривимірною графікою, який також надає доступ до інструментів для налаштування рівня деталізації та кастомних шейдерів. Крім того, рушій є багатоплатформним, що дозволяє оцінити, як алгоритми реалізуються в різних системах.

Unity Profiler рисунок 1.23 [26, с. 455–490] використовується для детального аналізу продуктивності програмних рішень. Завдяки цьому інструменту стало можливим оцінювати вплив алгоритмів на загальну продуктивність системи.

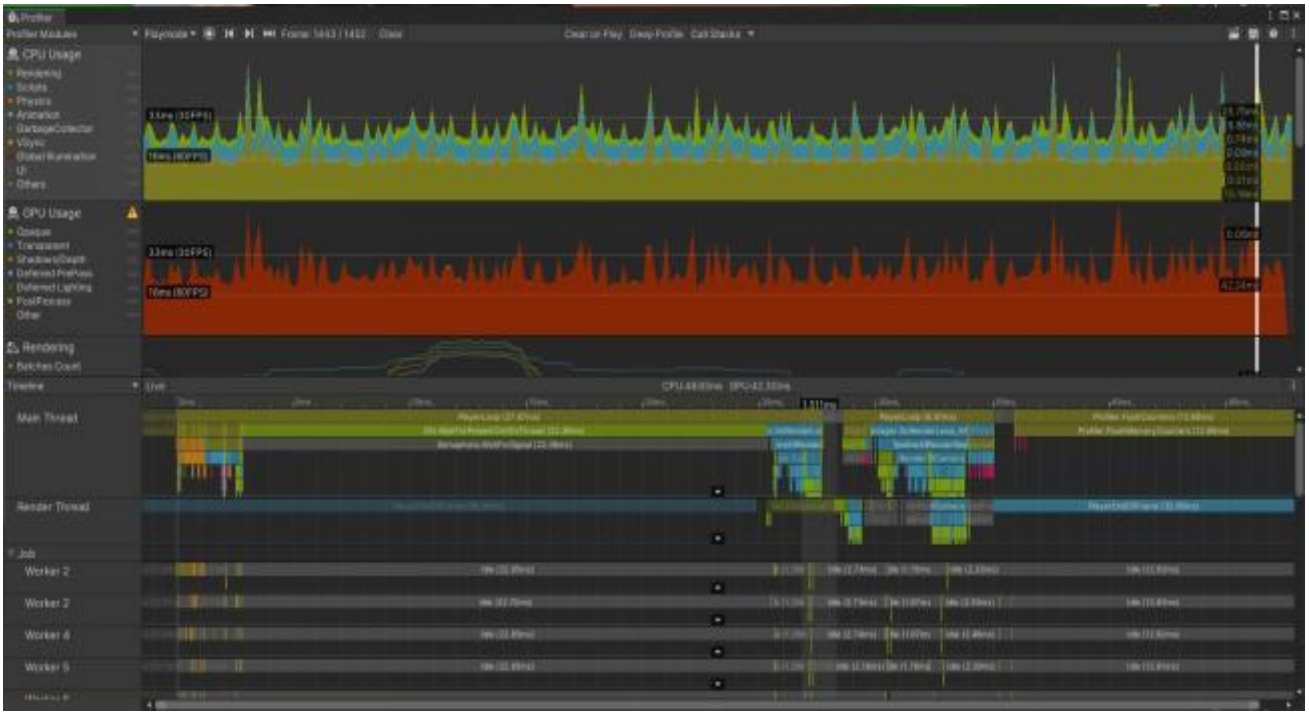


Рис. 1.23 Сережовище для моніторингу етапів рендерингу

Завдяки Unity Profiler можна визначити, які рендеринг об'єктів або обробка шейдерів, використовують найбільше ресурсів. В результаті підвищилася продуктивність і стала можливою оптимізація коду. Він також став важливим інструментом для відстеження використання пам'яті, CPU та GPU.

C# [27, с. 120-180] — це основна мова програмування Unity. Вона стала ідеальним варіантом для реалізації графічних обчислень та алгоритмів ігрової логіки завдяки простоті використання та чудовій продуктивності. C# дає можливість розробити специфічні рішення, такі як скрипти для роботи з графічними об'єктами, керування камерами та взаємодії з шейдерами, оскільки інтеграція C# з Unity дає нам доступ до всіх компонентів рушія.

Visual Studio рисунок 1.24 [28, с. 55-85] — це середовище розробки для написання, налагодження та тестування коду. Ця програма пропонує автоматичне виправлення помилок, підтримку інтеграції з Unity та інтуїтивно зрозумілий

інтерфейс для роботи з великими обсягами коду. Це сприяло швидкій розробці та модифікації функціональності проекту, а також оптимізації та перевірці логіки.

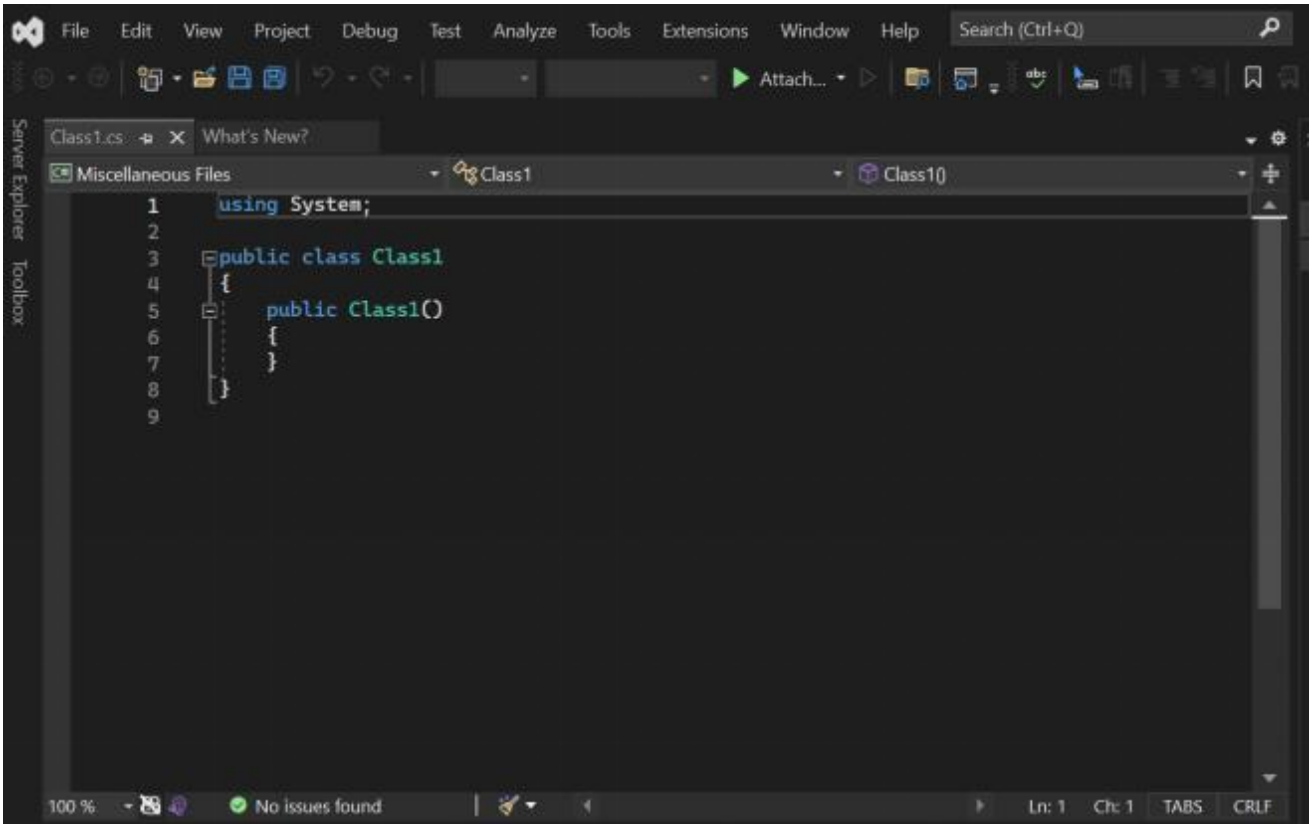


Рис. 1.24 Інтерфейс середовища розробки

Shader Graph, **HLSL** та **GLSL** рисунок 1.25 [2, с. 325-350] — це ключові інструменти для роботи зі шейдерами та створення кастомних графічних рішень у Unity. Вони дають можливість створювати матеріали з чудовим освітленням і деталізацією сцени за допомогою інтуїтивно зрозумілого інтерфейсу налаштування візуальних шейдерів Shader Graph. Зручний спосіб роботи з цим інструментом дозволив нам швидко тестувати та налаштовувати параметри, модифікуючи матеріали відповідно до певних потреб.

Для подальшого поглибленого налаштування та оптимізації використаємо мови шейдерів HLSL та GLSL. Вони дозволяють тонко налаштовувати шейдери на рівні коду, даючи точний контроль над тим, як працюють текстури, освітлення та тіні. GLSL було використано для покращення сумісності з OpenGL-орієнтованими платформами, тоді як HLSL було інтегровано для максимізації продуктивності для платформ на базі DirectX.

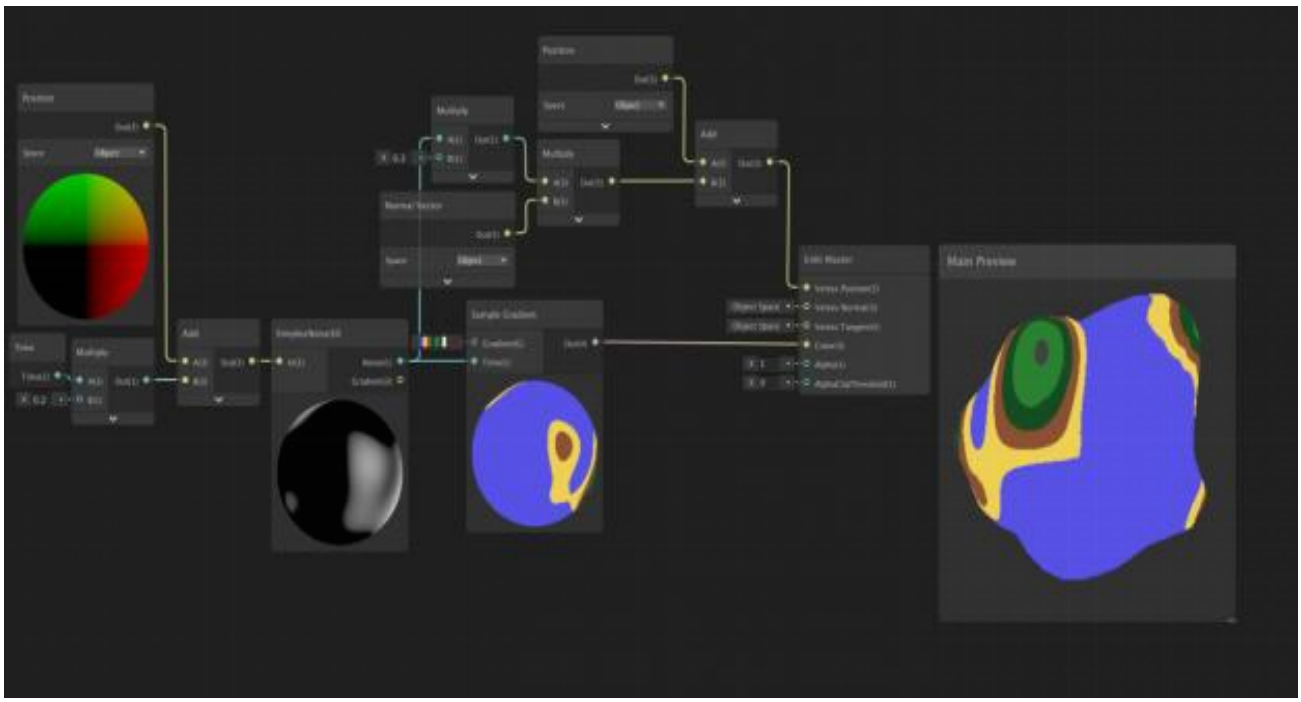


Рис. 1.25 Приклад візуального конструктора шейдерів

Завдяки візуальній інтеграції та можливостям ручного кодування через HLSL і GLSL, Shader Graph став потужним інструментом для тестування та експериментів з різними графічними рішеннями, гарантуючи максимальну гнучкість і високу якість результатів.

Math.NET [29, с. 210-260] — це математична бібліотека, створена для виконання складних числових операцій в середовищі .NET, яка пропонує широкий спектр інструментів для числових обчислень, аналізу даних та математичного моделювання. Робота з матрицями, векторами, спеціальними функціями та статистичними обчисленнями спрощується за допомогою цієї бібліотеки. Наприклад, Math.NET дозволяє обчислювати додавання, множення та обернені матриці, що має вирішальне значення при розробці алгоритмів для освітлення та обробки зображень. Вона також містить можливість розв'язувати диференціальні рівняння та інтегрувати їх чисельно, що важливо для розрахунку фізичних моделей, особливо для імітації руху світла або тіні в тривимірному просторі.

DirectX [30, с. 217-245, с. 451-478] — це графічний API, який став стандартом Windows і був створений компанією Microsoft для роботи з

мультимедійними програмами, зокрема відеоіграми. Його основною перевагою є прямий доступ до апаратних ресурсів комп'ютера, включаючи графічний процесор, для оптимізації продуктивності.

За допомогою DirectX можна створювати чудові графічні ефекти, такі як освітлення, відображення, текстурування та шейдери. Крім того, цей API містить вбудовані функції для керування складними візуальними ефектами та фізичними симуляціями, які виконуються на графічних процесорах. Щоб гарантувати сумісність з Windows і створити ефективні графічні рішення, які максимально використовують графічний процесор, в рамках проекту було використано DirectX. Це дозволило створювати високоякісні візуальні ефекти без помітного зниження продуктивності.

Vulkan [31, с. 250-290] — це передовий низькорівневий графічний API, створений компанією Khronos Group, який надає користувачам ще більшу гнучкість у роботі з апаратним забезпеченням, ніж DirectX. Його основною перевагою є кросплатформенна сумісність, яка дозволяє розробникам оптимізувати свої програми для Windows, Linux, Android та інших платформ.

Vulkan пропонує складні можливості для паралельного виконання завдань, синхронізації процесів та управління пам'яттю графічного процесора. Це дозволяє ефективно використовувати багатоядерні процесори, що має вирішальне значення для досягнення високої продуктивності при роботі з високими обчислювальними навантаженнями. Продуктивність побудованих графічних алгоритмів була протестована в проекті за допомогою Vulkan. Це дозволило ретельно контролювати ресурси графічного процесора та гарантувати високу продуктивність навіть у складних ситуаціях, що дало змогу оцінити ефективність створених рішень.

2 ТЕОРЕТИЧНІ ОСНОВИ ТА РОЗРОБКА MRR

2.1 Концепція багатозонального рендерингу

Метод багатозонального рендерингу (MRR) рисунок 2.1 — це метод обробки графічних зображень, який регулює рівень деталізації залежно від важливості різних областей екрану, оптимізуючи таким чином використання ресурсів рендерингу.

Основна концепція MRR полягає у поділі екрану на кілька зон, кожна з яких має свої конфігурації рендерингу, визначені її пріоритетністю в контексті загальної сцени. Наприклад, центральна область може оброблятися з високим рівнем деталізації, тоді як периферійні частини з меншим значенням для користувача можуть рендеритися з меншою деталізацією.

Цей підхід дозволяє суттєво зменшити обчислювальну складність завдяки використанню різних рівнів деталізації (LOD) у залежності від важливості кожної зони. Таким чином, метод MRR досягає балансу між забезпеченням високої якості зображення та підтриманням оптимальної продуктивності системи, що є важливим для сучасної графічної обробки, особливо у відеоіграх і мультимедіа.

Екранний простір ділиться на 3 зони:

- Центральна зона, на яку спрямована основна увага користувача, рендериться з максимальною роздільною здатністю, забезпечуючи високу деталізацію.
- Середня зона, яка виконує роль перехідної області, обробляється із зниженою роздільною здатністю для зменшення навантаження на ресурси.
- Найменш важлива крайня зона рендериться з мінімальною роздільною здатністю, оскільки зазвичай перебуває поза фокусом уваги користувача і використовується для відображення фонових елементів.

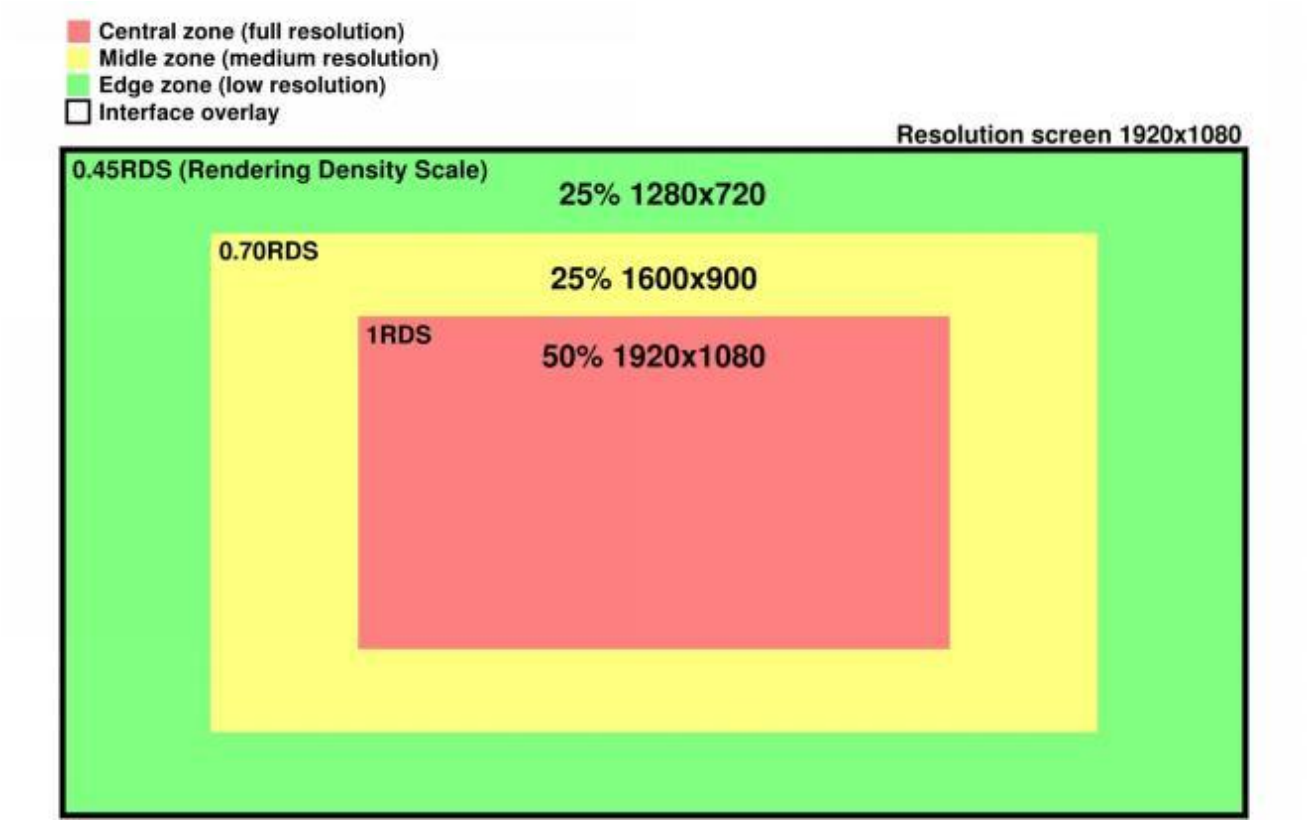


Рис. 2.1 Схема розподілу зон рендерингу

Спочатку кадр розділяється на окремі зони за допомогою правил, отриманих на основі адаптивних алгоритмів або геометричного поділу. Потім кожна зона відображається з певним рівнем деталізації: середня та крайні зони відображаються з нижчою якістю, виходячи з їхньої важливості, тоді як центральна зона відображається з повною роздільною здатністю. Фінальний кадр створюється шляхом об'єднання результатів рендерингу, які масштабуються до загальної роздільної здатності екрану. Останнім кроком є накладання елементів HUD, які окремо візуалізуються у високій роздільній здатності, щоб гарантувати чіткість.

MRR зменшує навантаження на графічний процесор, що важливо для пристроїв з обмеженими ресурсами, таких як консолі та мобільні пристрої. Крім того, покращується користувацький досвід, оскільки відмінна якість зображення зберігається у важливих для користувача областях. Оскільки він може підлаштовуватися під різні розміри екранів, роздільну здатність і навіть

положення погляду користувача (при використанні відстеження погляду), цей підхід також є неймовірно універсальним.

У відеоіграх, де дуже важливо максимізувати продуктивність, не жертвуючи при цьому якістю сцени у критичних місцях, MRR є дуже корисним.Технологія допомагає зменшити затримки рендерингу у VR/AR додатках [32, с. 88-102], що є важливим для комфорту користувача.

MRR дозволяє ефективно використовувати ресурси, зберігаючи необхідний ступінь деталізації в системах симуляції та візуалізації, які обробляють великі обсяги даних.

2.2 Опис структури MRR

2.2.1 Методи розподілу кадру на зони

Екран представляється у вигляді двовимірної таблиці пікселів, і метою є визначення координат, які є межами кожної зони. Розглянемо методи визначення меж зон у кадрі на основі розподілу зображення.

Геометричний метод (фіксовані області) Цей підхід, показаний на рисунку 2.1 ділить екран на прямокутні зони на основі заздалегідь визначених координат у двовимірній таблиці пікселів.

$$W \times H,$$

де:

W — ширина екрану в пікселях;

H — висота екрану в пікселях.

Центральна зона рисунок 2.2 визначається положенням камери (x_c, y_c) та розмірами зони, що встановлюються як пропорція до розмірів екрану (α, β).

Формула для координат меж центральної зони:

$$x_{min} = x_c - \frac{\alpha \times W}{2},\ x_{max} = x_c + \frac{\alpha \times W}{2},$$

$$y_{min} = y_c - \frac{\beta \times H}{2},\ y_{max} = y_c + \frac{\beta \times H}{2},$$

де:

x_{min}, x_{max}, y_{min}, y_{max} — координати меж зони;

x_c, y_c — координати центру камери;

α, β — пропорції ширини та висоти центральної зони (наприклад, 0.5 для 50% розміру).

Примітка: Якщо координати виходять за межі екрану ($x_{min} < 0$ або $x_{max} > W$), їх потрібно обмежити.

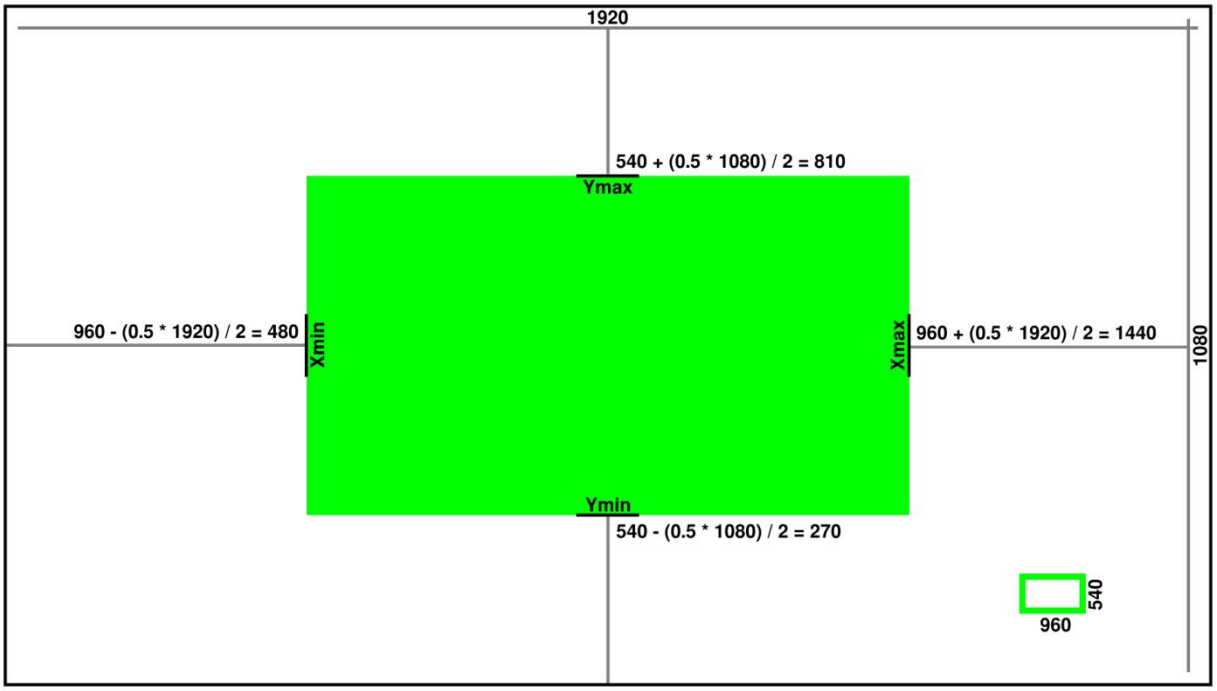


Рис. 2.2 Визначення центральної зони

Середня зона рисунок 2.3 обчислюються як залишкова частина екрану після визначення центральної зони.

$$x_{min}^{m} = max(0, x_{min} - \frac{\delta_x}{2}),\ x_{max}^{m} = min(W, x_{max} + \frac{\delta_x}{2}),$$

$$y_{min}^{m} = max(0, y_{min} - \frac{\delta_y}{2}),\ y_{max}^{m} = min(H, y_{max} + \frac{\delta_y}{2}),$$

де:

x_{min}^{m}, x_{max}^{m}, y_{min}^{m}, y_{max}^{m} — межі середньої зони;

γ — коефіцієнт (наприклад, 0.25);

δ_x, δ_y — відступи для середньої зони, зазвичай розраховуються як:

$$\gamma \times W,\ \gamma \times H.$$

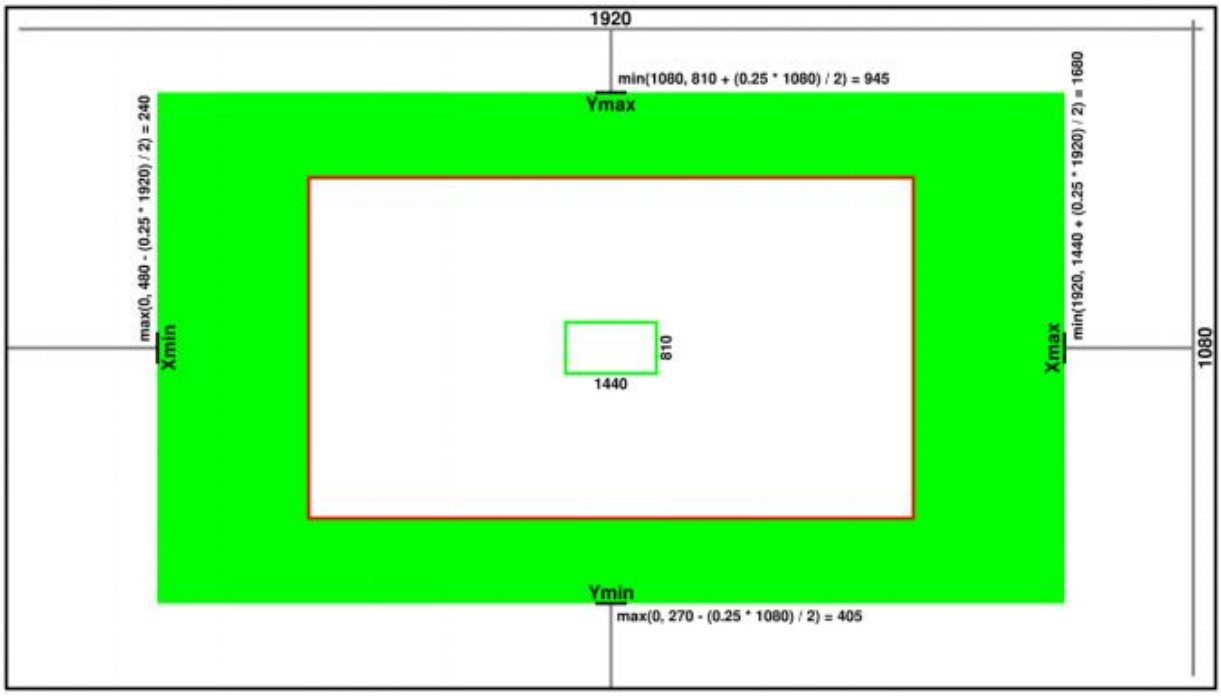


Рис. 2.3 Визначення середньої зони

Крайня зона рисунок 2.4 покриває всі області, що не входять до центральної та середньої зон, тобто крайня зона визначається залишковими пікселями:

$$Z_e = \frac{Екран}{Z_c \cup Z_m}$$

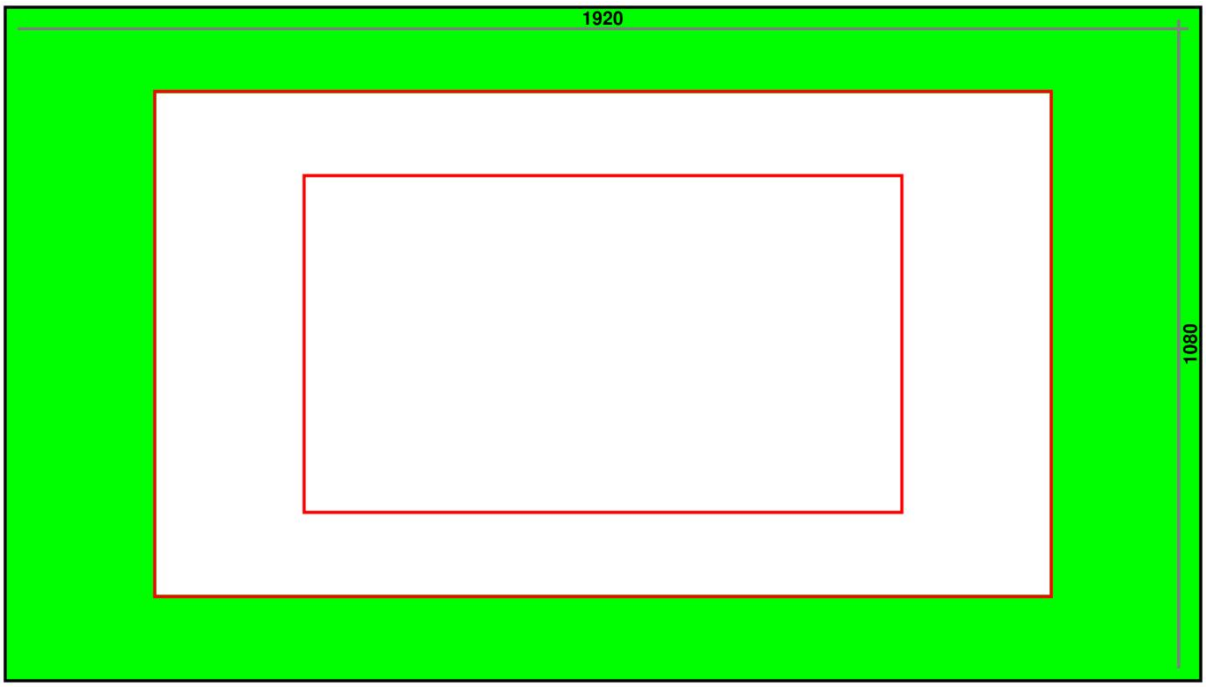


Рис. 2.4 Визначення крайньої зони

2.2.2 Рендеринг із різною роздільною здатністю

Розглянемо процес часткового рендерингу кадру із застосуванням різних роздільних здатностей для різних зон. Описуються два підходи до рендерингу — Quality та Performance, які дозволяють балансувати між якістю зображення та продуктивністю. Також детально розглянуто роль шейдерів у цьому процесі та алгоритми апскейлу, які використовуються для об'єднання зон у підсумковий кадр.

1. **"Quality" (Якість)** рисунок 2.5 — цей підхід орієнтований на збереження максимальної візуальної якості, навіть у менш важливих зонах. Він підходить для потужних систем із високим графічним потенціалом:

Рис. 2.5 Схема багатозонального рендерингу "Якість"

- Центральна зона (100%):

  Рендериться з повною роздільною здатністю $(W_c = W, H_c = H)$, для екрану 1920x1080 центральна зона рендериться у 1920x1080.

- Середня зона (83.34%):

  Рендериться з роздільною здатністю 83.34% від фізичної $(\mathrm{W_m} = 0.8334\mathrm{W}, \mathrm{H_m} = 0.8334\mathrm{H})$, для екрану 1920x1080 середня зона рендериться у 1600x900.

- Крайня зона (66.67%):

  Рендериться з роздільною здатністю 66.67% від фізичної ($W_e = 0.6667W, H_e = 0.6667H$), для екрану 1920x1080 крайня зона рендериться у 1280x720.

2. **"Performance" (Продуктивність)** рисунок 2.6 — цей підхід спрямований на досягнення максимальної продуктивності за рахунок сильнішого зниження роздільної здатності в менш важливих зонах. Підходить для систем початкового рівня:

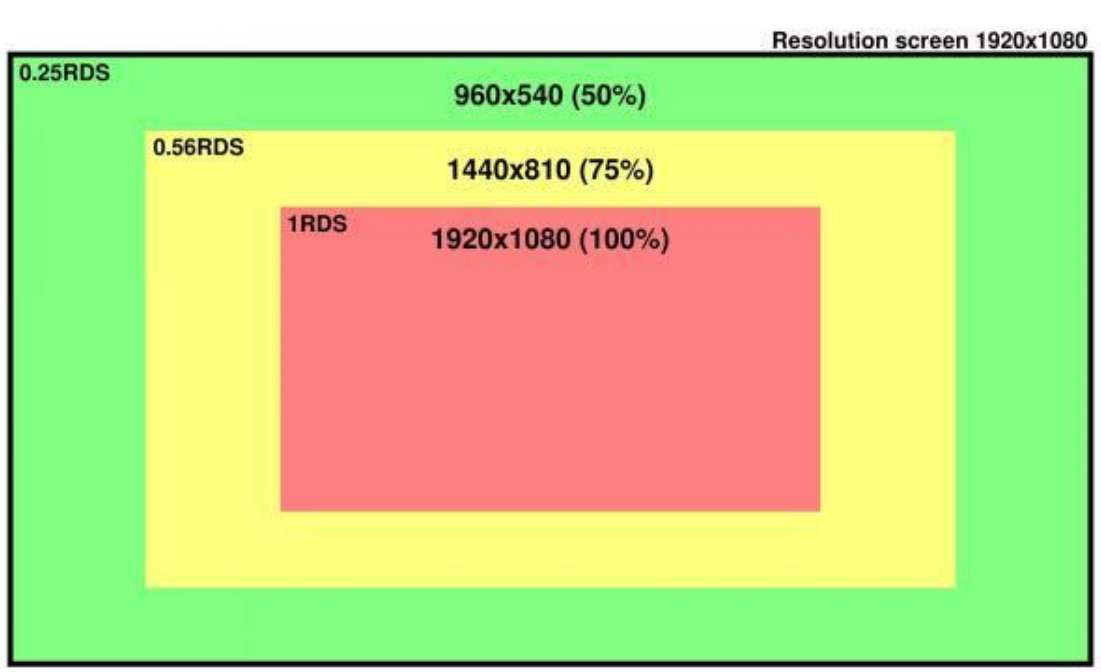


Рис. 2.6 Схема багатозонального рендерингу “Продуктивність”

- Центральна зона (100%):

  Рендериться з повною роздільною здатністю ($W_c = W, H_c = H$), для екрану 1920x1080 центральна зона має 1920x1080.

- Середня зона (75%):

  Рендериться з повною роздільною здатністю 75% від фізичної ($W_e = 0.75W, H_e = 0.75H$), для екрану 1920x1080 середня зона має 1440x810.

- Крайня зона (50%):

  Використовується роздільна здатність 50% від фізичної ($W_e = 0.5W, H_e = 0.5H$), для екрану 1920x1080 крайня зона має 960x540.

При використанні методу багатозонного рендерингу шейдери відіграють вирішальну роль у рендерингу. На етапі призначення зон шейдери відповідають за визначення областей екрана, які потрапляють у центральну, середню або крайню зони. Для цього використовуються координати пікселів та значення, що визначають межі зон. При рендерингу зон фрагментний шейдер обробляє пікселі з різною роздільною здатністю, враховуючи унікальні характеристики кожної зони, тоді як вершинний шейдер трансформує геометричні координати об'єктів на основі обраної зони кадру.

Шейдери підлаштовують рендеринг до роздільної здатності екрана на етапі масштабування. Це досягається шляхом обчислення нових значень пікселів на основі сусідніх за допомогою білінійної інтерполяції. Об'єднання зон - це останній крок, на якому піксельний шейдер створює єдине зображення, об'єднуючи вихідні дані з кожної зони у фінальний кадр.

Алгоритм апскейлу (Upscaling) [33], після рендерингу зон їх результати апскейляться до роздільної здатності екрану. Алгоритм включає:

1. Інтерполяція пікселів рисунок 2.7: для пікселів із меншою роздільною здатністю виконується білінійна інтерполяція:

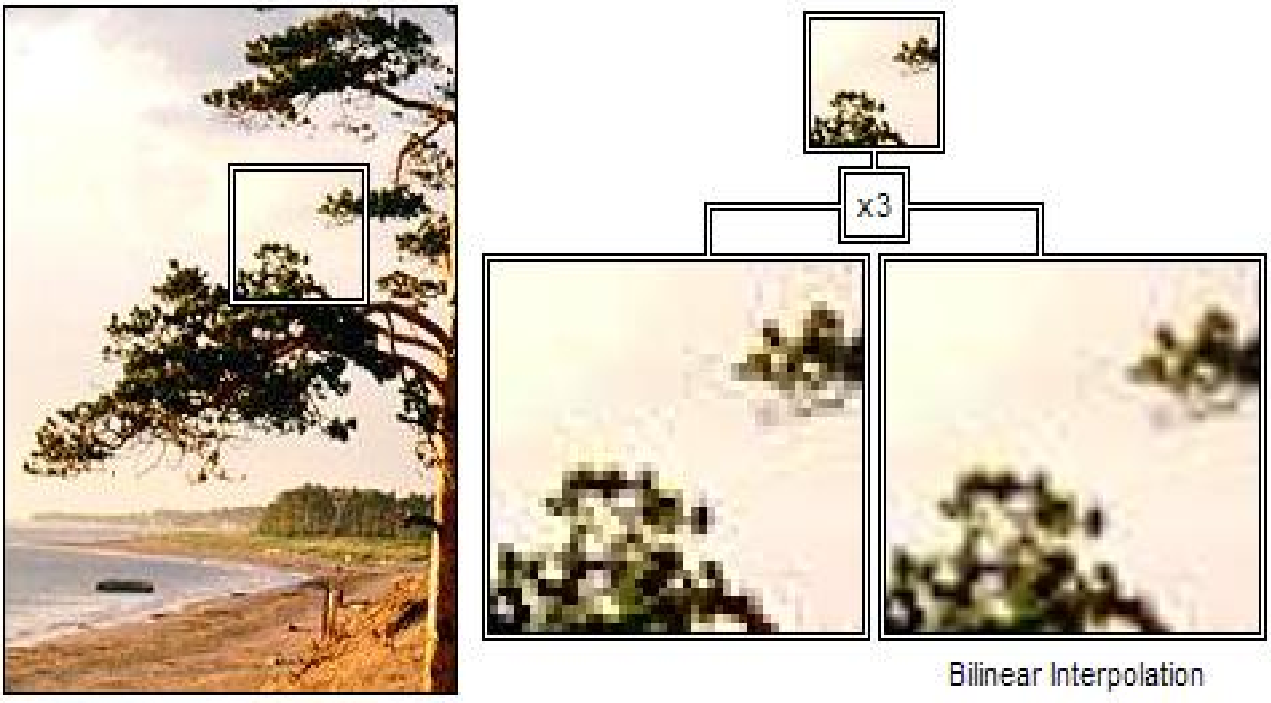


Рис. 2.7 Приклад білінійної інтерполяції

$$P(x,y) = \frac{1}{4}[P(x_1,y_1) + P(x_2,y_1) + P(x_1,y_2) + P(x_2,y_2)],$$

де:

$P(x,y)$ — апскейлений піксель;

$P(x_i,y_i)$ — сусідні пікселі.

Об’єднання зон:

- Кожна зона масштабується до своїх фізичних меж на екрані.
- Підсумковий кадр об’єднується шляхом накладання зон із різними деталізаціями.

Додавання інтерфейсу: ігровий інтерфейс рендериться окремо в повній роздільній здатності та накладається поверх кадру.

Формула підсумкового кадру

$$F(x,y) = \begin{cases} F_c(x,y), & (x,y) \in Z_c, \\ \text{Upscale}(F_m(x,y)), & (x,y) \in Z_m, \\ \text{Upscale}(F_e(x,y)), & (x,y) \in Z_e. \end{cases}$$

де:

$F(x,y)$ — піксель у фінальному кадрі;

$F_c(x,y)$, $F_m(x,y)$, $F_e(x,y)$ — значення пікселів із фреймбуферів центральної, середньої та крайньої зон відповідно;

Upscale — функція апскейлу.

2.3 Опис реалізації MRR у ігровому середовищі

В основі MRR лежить уявлення про те, що різні елементи екрану мають різну візуальну значущість. Увага гравця здебільшого зосереджена на центральній частині екрана, тоді як периферійні частини зазвичай отримують менше уваги. Для того, щоб значно зменшити навантаження на графічний процесор без втрати якості, MRR використовує зонування екрану з різною роздільною здатністю.

Структура реалізації на рисунку 2.8

1. Камера рендерингу: основна камера використовується для поділу зон та передачі даних у рендер-таргети.
2. Маски зон: графічні об’єкти, що визначають межі зон (реалізуються через текстури).
3. Рендер-таргети: окремі буфери для кожної зони (центральна, середня та крайня).
4. Шейдери: збільшення роздільної здатності рендерингу, масштабування та об’єднання зон.

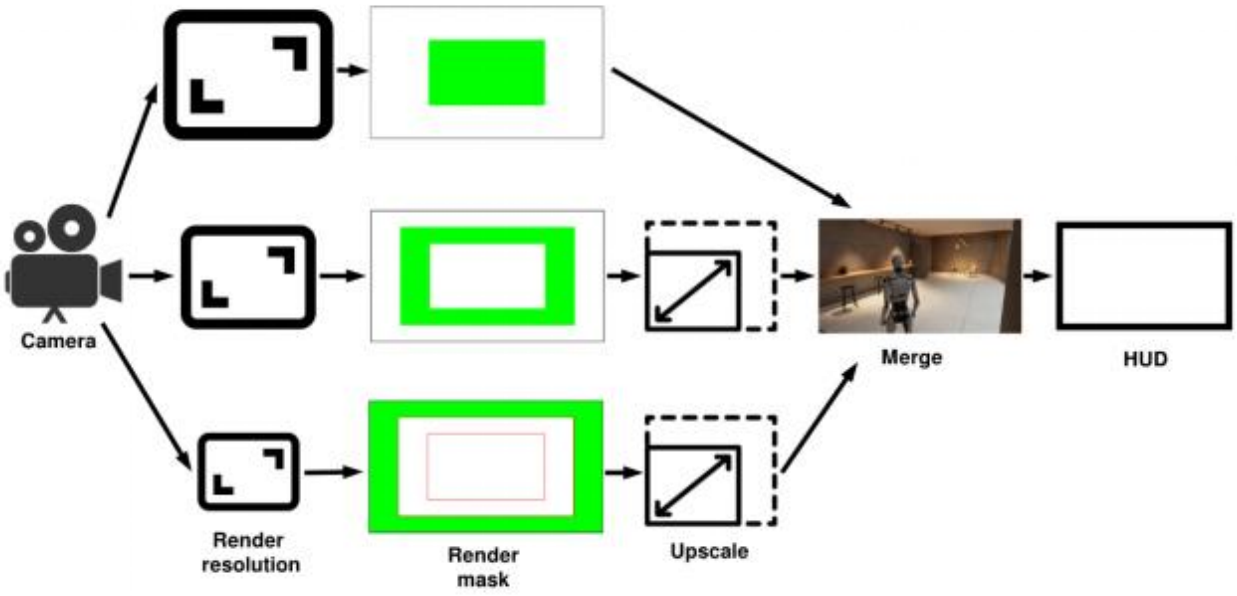


Рис. 2.8 Схема структури реалізації

За допомогою зонування екрану створюються три зони: центральна зона, яка займає 50% екрану, середня зона, яка займає 25% екрану, і крайня зона, яка займає 25% екрану. Для того, щоб точно розділити ці зони, створюються маски - візуальні елементи, які окреслюють межі кожної зони. Маски мають визначені розміри зон і є статичними.

Після рендерингу центральної та крайніх зон результати масштабуються до кінцевої роздільної здатності кадру (наприклад, 1920x1080). Це досягається шляхом розтягування зображення за допомогою спеціалізованих шейдерів постобробки.

Зони інтегруються в один фінальний кадр, який складається з усіх трьох зон. Піксельні шейдери, які враховують маски та зони і комбінують їх у правильному порядку, роблять цю процедуру можливою.

Рендеринг інтерфейсу користувача (UI) поверх готового кадру є завершальним етапом процесу.Щоб гарантувати читабельність тексту і спрайтів, користувацький інтерфейс рендериться у повній роздільній здатності.

Маски для зон рендерингу рисунок 2.9 визначають, які частини кадру та з якою роздільною здатностю вони будуть рендеритися. Маски використовуються для відділення різних зон рендерингу, а також для оптимізації використання ресурсів на рендеринг.

Основні етапи роботи маски:

1. Маска центральної зони рисунок 2.2, визначена як найбільш деталізована. У межах цієї зони рендеримо пікселі з максимальною роздільною здатністю. Для цього в шейдер передається прямокутник для рендерингу, який описує координати центральної зони.
   _RenderZone ("Render Zone Rect", Vector) = (0.25, 0.25, 0.75, 0.75)
2. Маска середньої зони рисунок 2.3, знаходиться між центральною зоною та крайньою зоною, рендериться з погіршеною роздільною здатністю. Для цієї маски шейдер повинен виключити з рендерингу центральну зону, щоб уникнути перекриття. У шейдері для виключення центральної зони передається прямокутник, який задає межі цієї зони.
   _RenderZone ("Render Zone Rect", Vector) = (0.125, 0.125, 0.875, 0.875)
   _ExcludeZone ("Exclude Zone Rect", Vector) = (0.25, 0.25, 0.75, 0.75)
3. Маска крайньої зон рисунок 2.4, рендериться з найменшою роздільною здатністю. Маска крайньої зони виключає центральну та середню частини, оскільки ці області вже були оброблені.
   _RenderZone ("Render Zone Rect", Vector) = (0, 0, 1, 1)
   _ExcludeZone ("Exclude Zone Rect", Vector) = (0.125, 0.125, 0.875, 0.875)

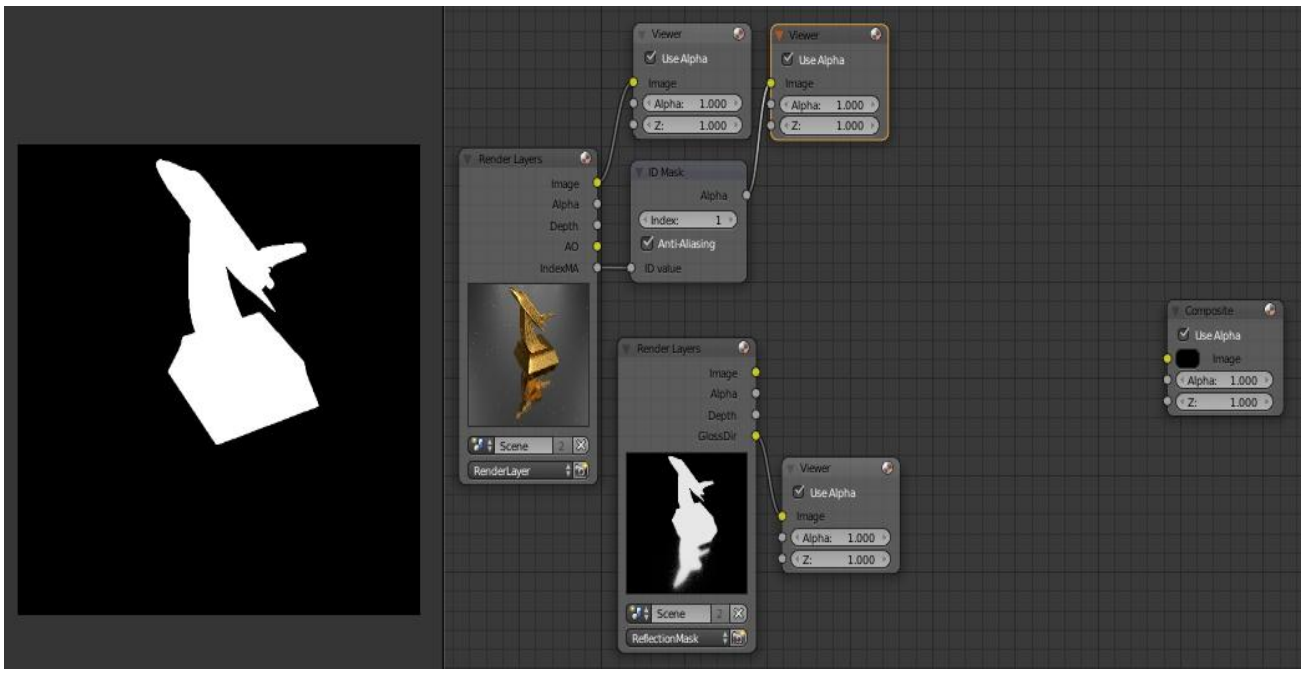


Рис. 2.9 Приклад застосування маски рендерингу

Принцип роботи шейдерної маски, який для зон рендерингу, виконує наступні основні кроки:

1. Шейдерна маска, створюється маска для кожної зони з масштабом роздільної здатності відповідно до її показників.
2. Отримання координат пікселя. У шейдері використовується UV-координати для визначення позиції пікселя:
   vec2 uv = gl_FragCoord.xy / resolution.xy;
3. Порівняння координат із межами зон. Для кожної зони рендерингу визначено прямокутник, який обмежує область, яку необхідно рендерити. Прямокутники передаються в шейдер як параметр. Шейдер порівнює координати пікселя з відповідними межами зони:
   if (uv.x >= _RenderZone.x && uv.x <= _RenderZone.z && uv.y >= _RenderZone.y && uv.y <= _RenderZone.w)
4. Виключення пікселів. Пікселі, які знаходяться за межами визначених зон, пропускаються:
   if (uv.x < _ExcludeZone.x || uv.x > _ExcludeZone.z || uv.y < _ExcludeZone.y || uv.y > _ExcludeZone.w)

Якщо піксель знаходиться в межах однієї з зон, то він буде відрендерений, і для нього вибирається відповідна якість або роздільна здатність.

Маски для зон рендерингу дозволяють ефективно керувати ресурсами при рендерингу кадру. Шейдери, що працюють з масками, використовують параметри прямокутників для виключення зон, що дає змогу ефективно керувати рендерингом окремих частин кадру. Таких підхід знижує навантаження на систему.

Апскейлінг зон рисунок 2.10, після рендеру середньої та крайньої зони, їх потрібно масштабувати до роздільної здатності екрану шляхом інтерполяції пікселів. У графічному програмуванні це досягається через шейдери або вбудовані методи API, які використовують алгоритми масштабування, такі як білінійна інтерполяція.

Graphics.Blit(sourceTexture, targetTexture),

де:

sourceTexture — вхідна текстура однієї з зон;

targetTexture — вихідна текстура з необхідною роздільною здатностю;

Graphics.Blit — автоматично масштабує текстуру під час копіювання.

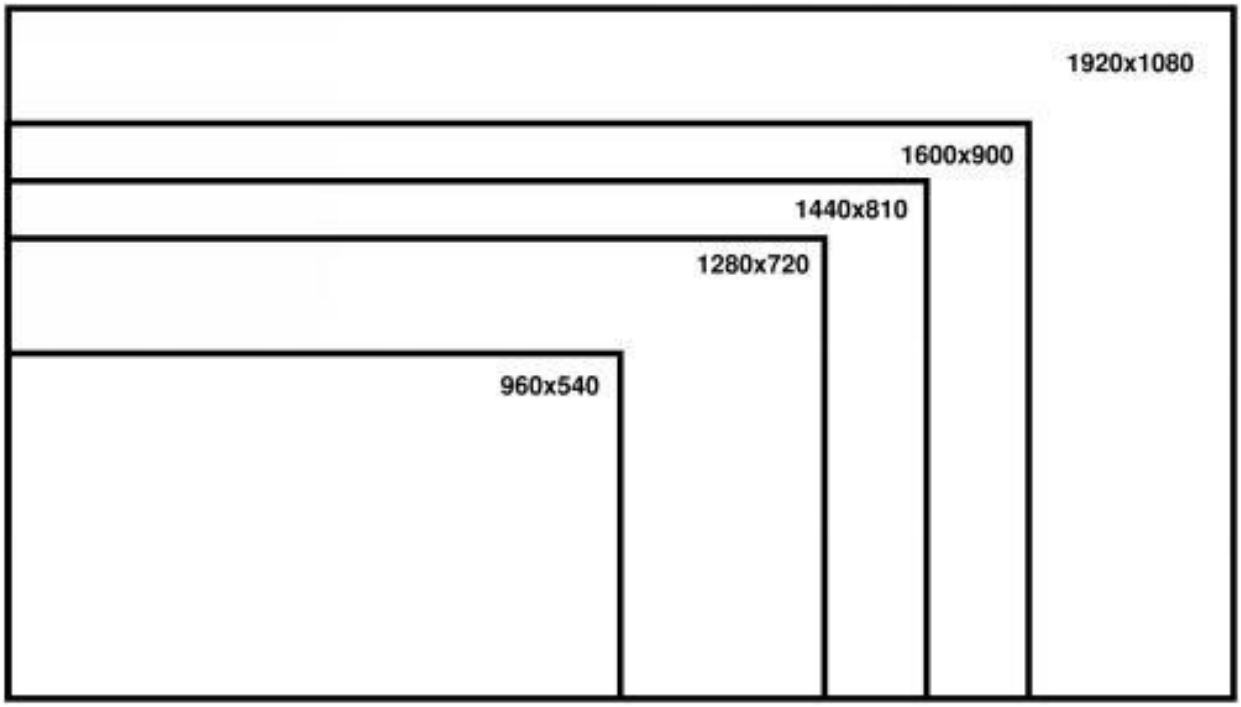


Рис. 2.10 Масштаби роздільної здатності

Об’єднання зон рисунок 2.11, після апскейлінгу кожної зони (середньої та крайньої) їх потрібно об'єднати у фінальний кадр. Цей етап завершує процес зонального рендерингу, формуючи зображення, яке буде відображено на екрані. Об'єднання зон відбувається шляхом послідовного накладення кожної зони на фінальну текстуру.

Очищення фінальної текстури: Фінальна текстура очищується, щоб уникнути залишків попередніх рендерів. Це забезпечує коректне накладення текстур.

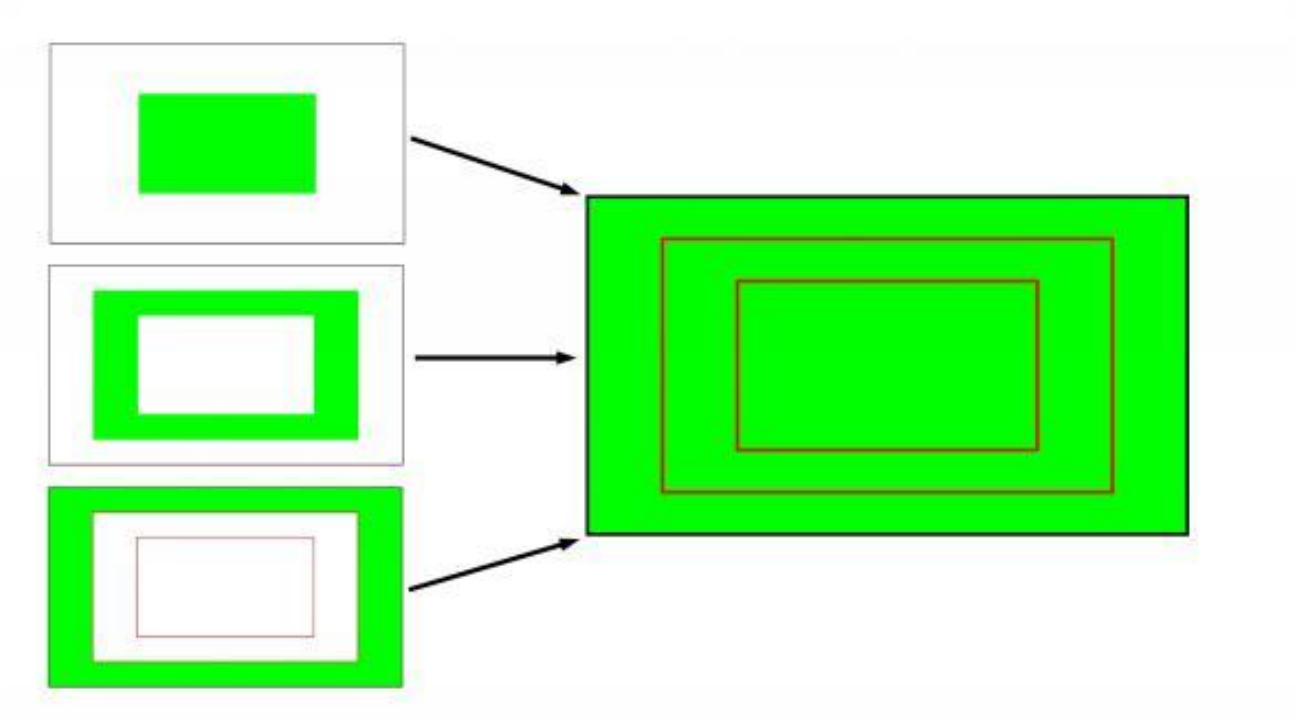


Рис. 2.11 Схема об’єднання зон

RenderTexture.active = finalTexture; // Вказує, яка текстура є активною для рендерингу

GL.Clear(true, true, Color.black); // Очищує активну текстуру, видаляючи всі попередні дані

Послідовне накладення зон:

Graphics.Blit(outerTexture, finalTexture); // Крайня зона

Graphics.Blit(middleTexture, finalTexture); // Середня зона

Graphics.Blit(centralTexture, finalTexture); // Центральна зона

Виведення фінального кадру на екрані разом із накладанням елементів HUD. Цей крок виводить зображення на екран та поверх кадру інтерфейс, який не залежить від зонального поділу.

Виведення фінального кадру використовується методом Graphics.Blit, який копіює вміст текстури в рендеринг-буфер для відображення.

```
Graphics.Blit(finalTexture, null as RenderTexture);
```

Накладання елементів HUD, елементи інтерфейсу, такі як лічильники, мінікарти або текст, накладаються поверх фінального кадру. HUD додається з використанням GUI.DrawTexture, що дозволяє відобразити текстуру на передній області екрану.

```
GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), hudTexture);
```

2.4 Оцінка ефективності та сумістності методу MRR

Проаналізуємо, наскільки добре працює техніка багатозонного рендерингу (MRR) в іграх і наскільки добре вона працює з різними апаратними платформами та операційними системами. Зменшуючи обсяг даних, які потрібно обробити, технологія MRR значно підвищує ефективність гри, дозволяючи графічному процесору виконувати менше обчислень для кожного кадру. Як наслідок, особливо у високій роздільній здатності, спостерігається помітний приріст FPS до 20% у режимі “Якість” та до 40% у режимі “Продуктивність”. Це робить ігровий процес більш плавним у динамічних ігрових сценаріях, таких як бійки або швидкі рухи камери.

Метод MRR демонструє високу сумісність із сучасними графічними API, такими як DirectX, OpenGL і Vulkan, що дозволяє йому інтегруватися з найпоширенішими операційними системами:

- Windows: Використання DirectX забезпечує максимальну ефективність на платформах Windows 10 і 11.

- Mac OS: Завдяки підтримці Metal API метод працює стабільно на MacBook та iMac із чіпами серії M1/M2.
- Linux: Використання Vulkan або OpenGL гарантує продуктивність на дистрибутивах, таких як Ubuntu чи Fedora.

Однак на більш ранніх платформах цей підхід має кілька недоліків. Наприклад, обмеження пам'яті та підтримки шейдерів можуть бути причиною зниження продуктивності на старих графічних процесорах, таких як NVIDIA GTX 600 або AMD HD 7000 і старше. Деякі старіші пристрої OpenGL 2.0 можуть потребувати адаптації шейдерів.

Однією з головних переваг методу є низьке використання ресурсів, що забезпечує високу продуктивність навіть для величезних сценаріїв. Масштабованість методу дозволяє адаптувати його до різних роздільних здатностей, пристроїв та ситуацій, а зменшення навантаження на графічний процесор підвищує стабільність гри навіть на пристроях із середньою та низькою потужністю.

Як результат, підхід MRR демонструє високу ефективність у сучасних іграх, пропонуючи стабільність і плавність навіть у найскладніших умовах. Оскільки він працює з різними системами та графічними процесорами, його можна використовувати в проектах для широкого кола користувачів, включаючи ПК, ноутбуки, мобільні пристрої та ігрові консолі.

3 ТЕОРЕТИЧНІ ОСНОВИ ТА РОЗРОБКА ALD

3.1 Концепція адаптивної деталізації освітлення

Метод адаптивної деталізації освітлення (ALD) рисунок 3.1 — це підхід до оптимізації обробки світлових джерел і тіней у тривимірному середовищі. Динамічне керування джерелами світла та тінями у сцені є основою концепції ALD. Залежно від того, наскільки далеко вони знаходяться від гравця або наскільки вони важливі для загального ігрового процесу, кожне джерело світла в області об'єднується в групи деталей.

Рис. 3.1 Схема зон деталізації освітлення

На старті сцени Lighting manager сканує наявні джерела світла рисунок 3.2, позначаючи їх як важливі, звичайні або фонові. Звичайні та важливі джерела можуть змінювати деталізацію або припиняти роботу залежно від обставин, але важливі джерела ніколи не припиняють роботу. Світло об'єктів розподіляється відповідно до рівня деталізації (високий, середній, низький або вимкнений) після

періодичного обчислення відстані між гравцем та об'єктами. Це забезпечує найкращу якість освітлення поблизу гравця, тоді як на більш віддалених об'єктах джерела або повністю вимикаються, або деталізація поступово зменшується.

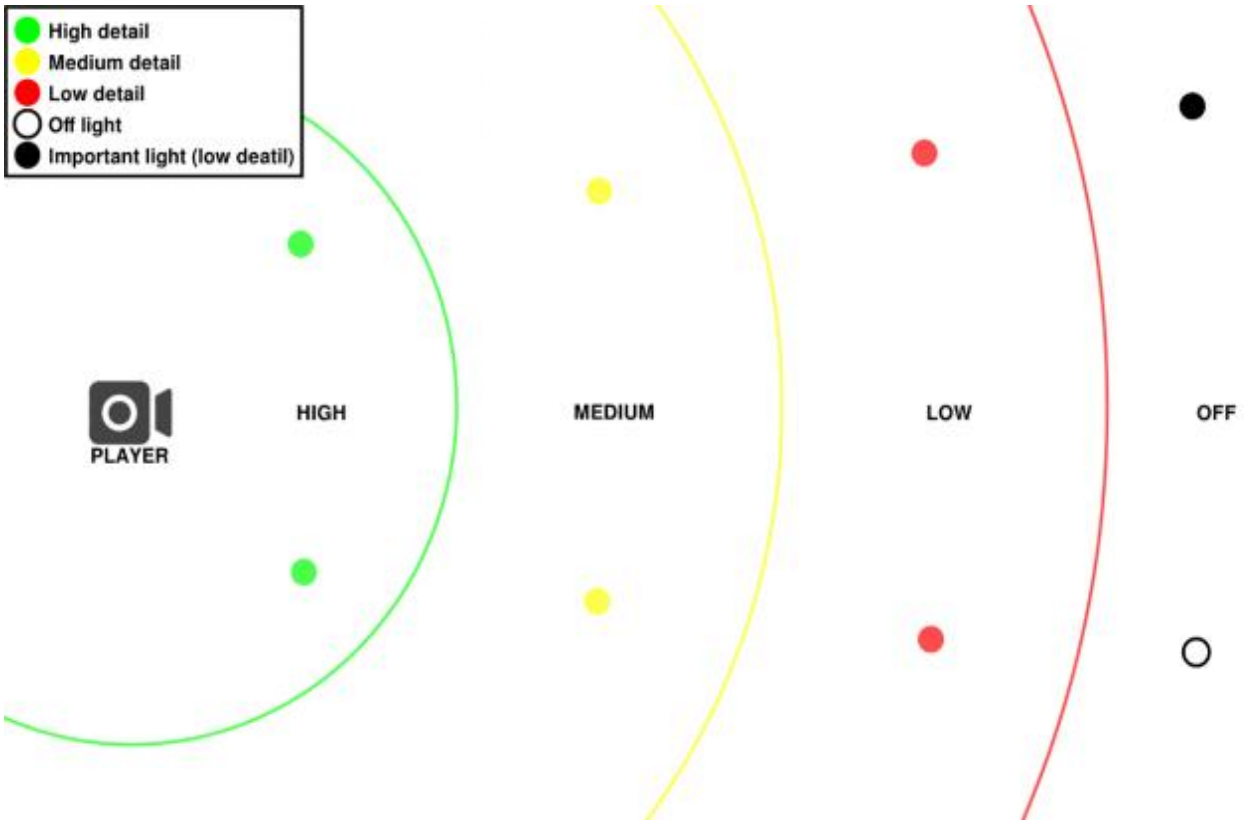


Рис. 3.2 Схема розподілу Lighting manager джерел освітлення на зони

Тіні рисунок 3.3, створені джерелами світла, змінюються залежно від відстані до гравця. Тіні мають велику складність і роздільну здатність зблизька, тому потрібно зображати найдрібніші деталі оточення. Рівень деталізації тіней неухильно падає зі збільшенням відстані, що майже непомітно для гравця, але значно підвищує продуктивність.

Статичні текстури можна використовувати замість тіней на великих відстанях, щоб зберегти зовнішній вигляд сцени без суттєвих змін у продуктивності. Щоб заощадити ресурси комп'ютера, відображення тіней припиняється, коли вони стають невидимими.

Lighting Manager використовується для асинхронного оновлення тіней і працює в іншому потоці. Це гарантує стабільність кадрів у складних сценаріях і зменшує навантаження на основний рендер. Частота оновлення тіней також

контролюється менеджером освітлення, який змінює її відповідно до вимог сцени. Продуктивність і якість графіки в цьому методі збалансовані.

Рис. 3.3 Види деталізації тіней

Такий підхід має низку суттєвих переваг. Вимикаючи сторонні джерела світла і знижуючи якість тіней у віддалених регіонах, він значно знижує навантаження на графічний процесор і гарантує відмінну продуктивність навіть у ситуаціях з великими відкритими ландшафтами. ALD дозволяє зберігати якість освітлення в ключових регіонах, адаптуючись до різних ситуацій, таких як інтенсивні бойові дії або додатки VR/AR. Завдяки розподілу джерел світла за категоріями, важливі компоненти залишаються постійно активними, зберігаючи високий ступінь реалістичності.

Ця стратегія часто використовується в проектах, що вимагають ефективного управління ресурсами, таких як ігри у відкритому просторі, VR/AR-симуляції та додатки зі складними налаштуваннями освітлення. Вона дозволяє створювати сценарії з відмінною візуальною якістю, використовуючи при цьому наявні у вашому розпорядженні ресурси.

3.2 Опис структури ALD

Метод адаптивної деталізації освітлення (ALD) рисунок 3.4 дозволяє керувати динамічним освітленням у реальному часі, зберігаючи відмінну візуальну якість і максимально використовуючи наявні ресурси. Основна ідея полягає в регулюванні рівня деталізації джерел світла залежно від їхньої відстані від гравця та значущості в поточному ігровому середовищі. Це зберігає реалістичність сцени, водночас значно знижуючи навантаження на обчислювальні ресурси.

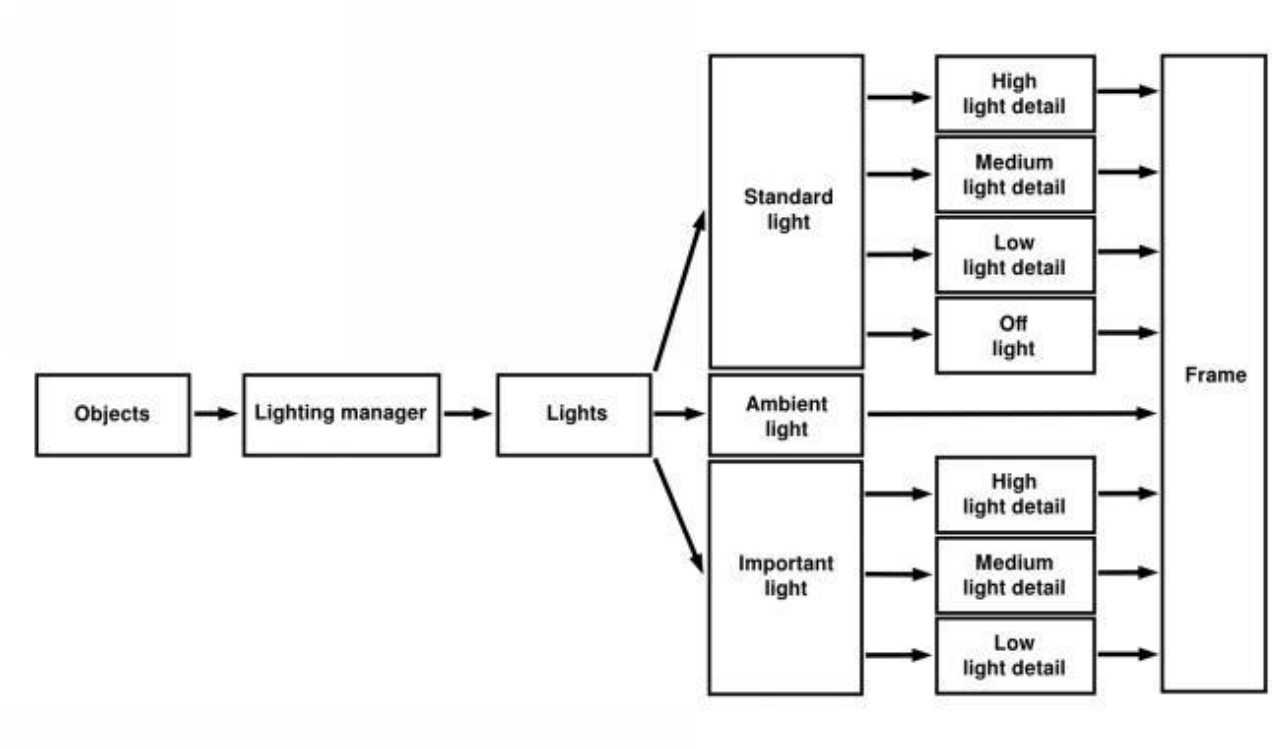


Рис. 3.4 Схема покрокової роботи ALD

Робота системи починається з аналізу сцени. ALD використовує вбудований механізм ідентифікації для сканування всіх доступних джерел світла та класифікації освітлення відповідно до його важливості. Найбільше значення надається джерелам світла, які мають важливе значення для створення настрою або впливу на ігровий процес. Залежно від обставин гри, менш важливі джерела можуть бути тимчасово вимкнені або оптимізовані.

Система визначає джерела, а потім обчислює, наскільки вони віддалені від гравця. Система використовує ці дані, щоб визначити, які джерела можна зобразити за допомогою зменшених методів рендерингу, а які вимагають високого рівня деталізації. Наприклад, джерела поблизу гравця зберігають деталізацію і мають чудові динамічні тіні, а джерела, розташовані далі, можуть бути вимкнені або перетворені на статичні.

Одним з найважливіших аспектів ALD є асинхронність, показана на рисунку 3.5 [34]. Оскільки кожне обчислення виконується в окремому потоці, це менше впливає на рендеринг основної сцени. Це зберігає продуктивність, дозволяючи системі пристосовуватися до змін в ігровому просторі. Плавний перехід між рівнями деталізації забезпечується частим оновленням параметрів освітлення.

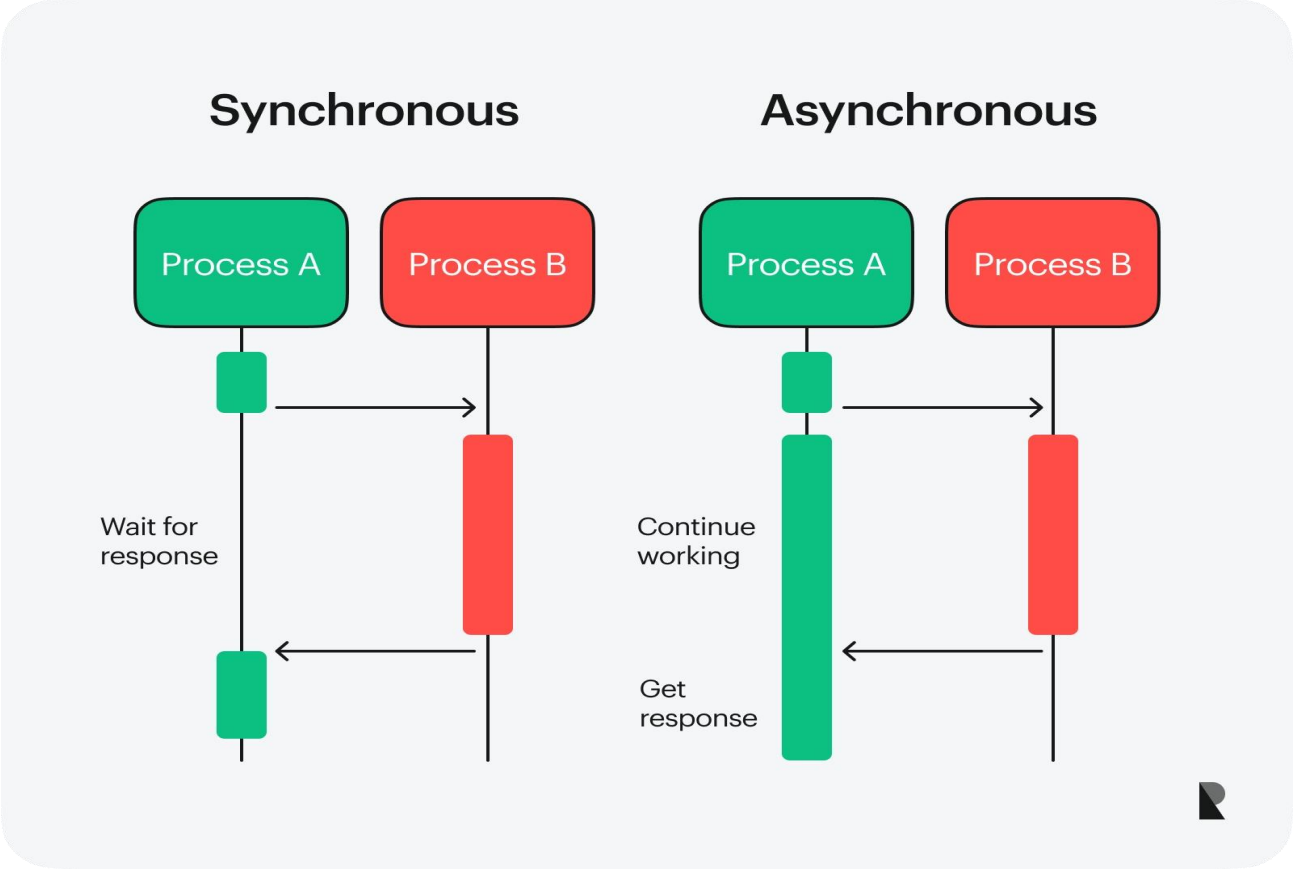


Рис. 3.5 Приклад асинхроної роботи

ALD є універсальним інструментом, який може бути адаптований для різних ігрових проєктів. Він ефективно працює як у компактних внутрішніх

сценах, так і у великих відкритих просторах, забезпечуючи баланс між продуктивністю та візуальною якістю.

3.2.1 Методологія адаптивного управління якістю освітлення

Кожен з основних етапів адаптивної технології управління якістю освітлення ALD призначений для динамічної зміни характеристик освітлення. Фундаментальна концепція полягає в застосуванні різних ступенів деталізації і визначенні в реальному часі, наскільки важливим є кожне джерело світла для сцени. Наприклад, джерела світла поблизу гравця отримують найбільшу деталізацію, тоді як віддалені джерела світла можуть бути вимкнені або працювати з меншою роздільною здатністю, якщо вони мало впливають на сцену.

Аналіз джерел світла відбувається за допомогою формули для розрахунку рівня деталізації:

$$L_i = \frac{1}{1+e^{\frac{D_i - T}{s}}},$$

де:

L_i — рівень деталізації для джерела світла *i* (значення від 0 до 1);

D_i — відстань від гравця до джерела світла *i*;

T — порогова відстань, на якій рівень деталізації змінюється;

s — параметр плавності переходу між рівнями (чим менше s, тим різкіший перехід).

В основі цієї формули лежить сигмоїдальна функція на рисунку 3.6, яка забезпечує плавний і органічний перехід між рівнями деталізації. Чим більше деталей, тим ближче джерело світла до гравця; чим далі, тим менше деталей, аж до повного вимкнення джерела світла, якщо дозволяє сцена.

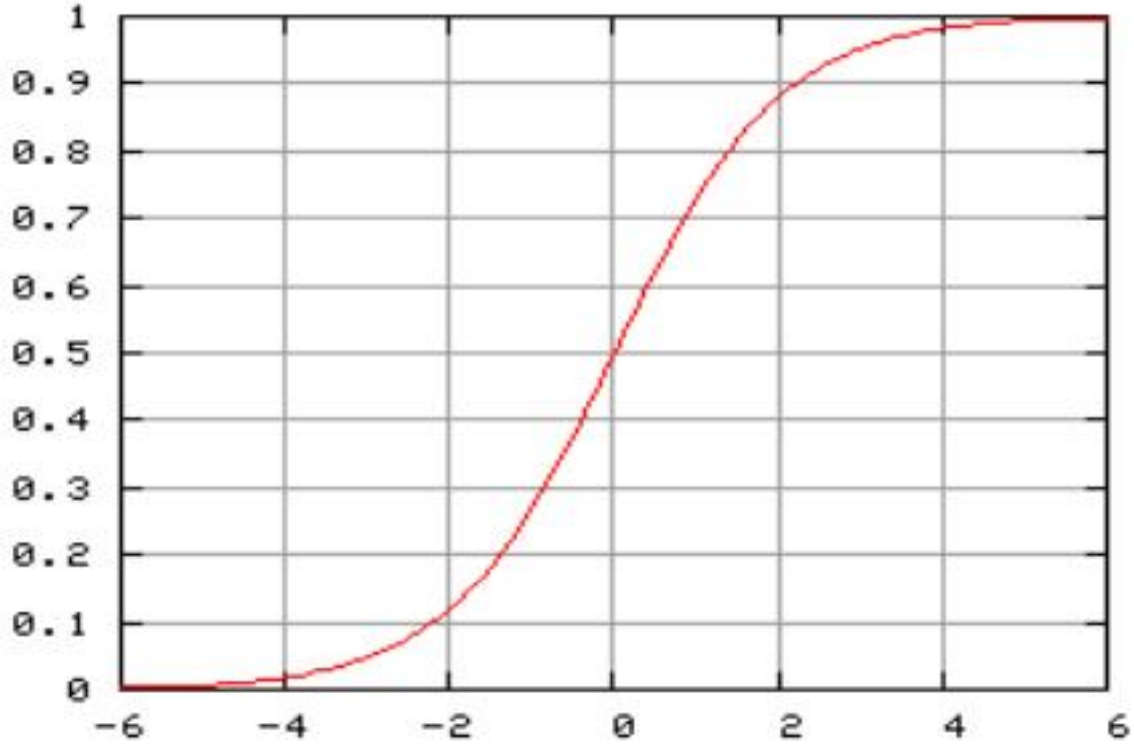


Рис. 3.6 Вигляд сигмоїної функції

Окрім рівня деталізації, іншим важливим параметром є інтенсивність джерел світла, яку також можна регулювати залежно від відстані, є важливим параметром на додаток до кількості деталей. Наприклад, щоб зменшити вплив віддалених джерел на сцену, можна зменшити інтенсивність їхнього світла. Нижче наведено формулу для визначення інтенсивності:

$$I_i = I_{max} \times (1 - L_i),$$

де:

I_i — фактична інтенсивність джерела світла iii;

I_{max} — максимальна інтенсивність джерела світла.

3.2.2 Визначення пріоритетних джерел освітлення

Не кожне джерело світла в сцені має однакову важливість. Оскільки кожне джерело класифікується в структурі ALD на основі того, наскільки воно важливе для гри, сконцентруємо свої ресурси обробки на найбільш важливих регіонах. Для цього кожному джерелу світла надається пріоритет на основі наступної формули:

$$P_i = W_i \times \frac{1}{D_i + \epsilon},$$

де:

P_i — пріоритет джерела світла i;

W_i — ваговий коефіцієнт, що залежить від типу джерела;

ϵ — невелике число для запобігання діленню на нуль.

Класифікуючи джерела світла відповідно до їхньої важливості, цей метод допомагає оптимізувати ресурси. Також можемо створювати унікальні класи в ALD для джерел світла, які ніколи не слід вимикати або знижувати їхню якість. Наприклад, джерела, які висвітлюють найважливіші компоненти гри, такі як супротивники або об'єкти завдань, позначаються як «критичні» і підтримують найвищий рівень інформації на всіх відстанях.

Управління тінями рисунок 3.7 є невід'ємною частиною освітлення, що додає реалістичності сцені. Є можливість регулювати роздільну здатність залежно від відстані до джерела світла завдяки технології адаптивного керування якістю тіней ALD.

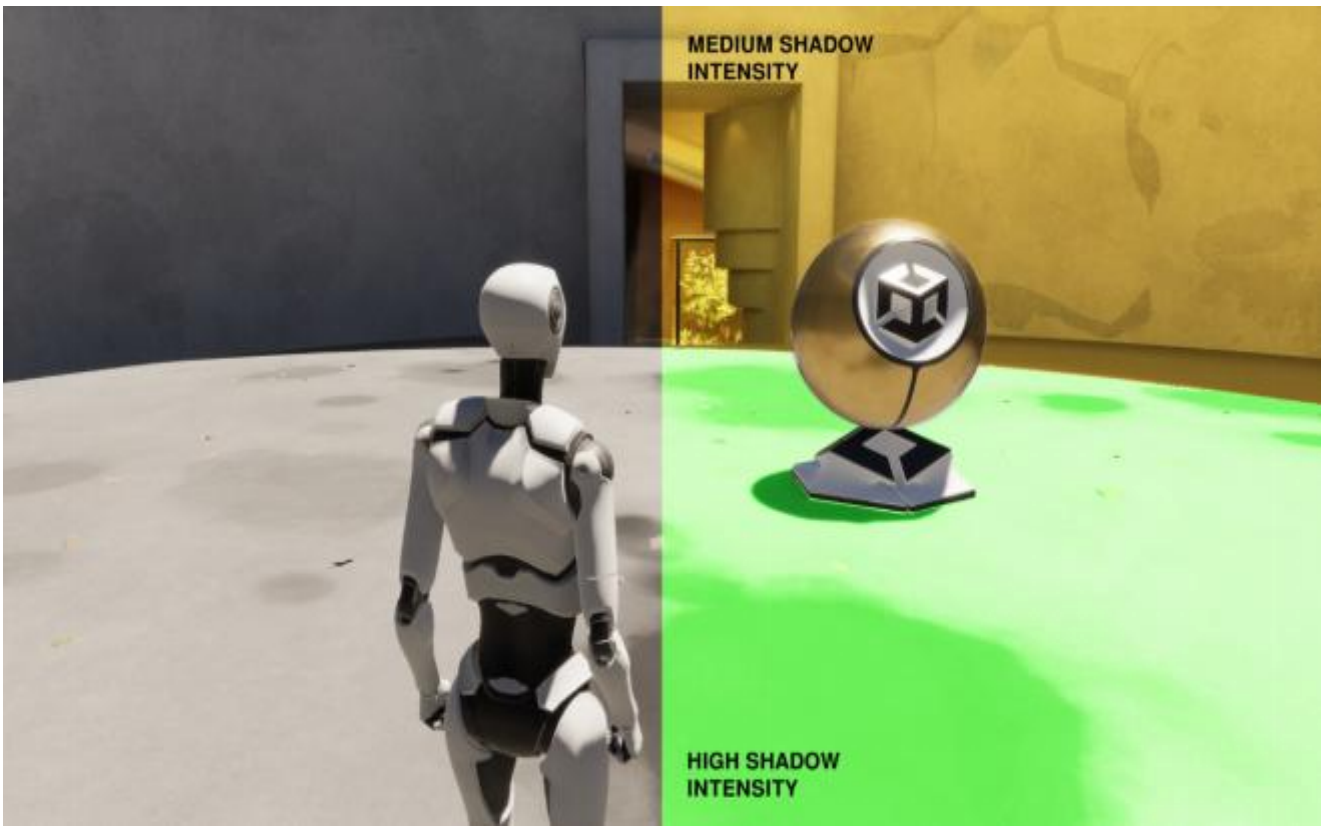


Рис. 3.7 Приклад інтенсивності тіней

$$R_i = R_{max} \times L_i,$$

де:

R_i — роздільна здатність тіней для джерела iii;

R_{max} — максимальна роздільна здатність тіней.

Тут R_{max} — максимальна роздільна здатність тіней, яка використовується для близьких джерел світла. У міру збільшення відстані рівень деталізації тіней знижується, що дозволяє зменшити кількість використовуваних ресурсів.

3.3 Опис реалізації ALD у ігровому середовищі

3.3.1 Інтеграція з ігровим рушієм

Необхідно правильно інтегрувати систему адаптивного рівня освітлення Unity з рушієм. Реєстрація джерел світла, аналіз сцени, імплементація шейдерів для адаптації освітлення та динамічне керування параметрами освітлення - це етапи цього процесу. Нижче розглянемо ці етапи більш детально.

Реєстрація джерел світла у сцені рисунок 3.8, щоб працювати з джерелами світла, їх спочатку потрібно знайти в сцені. Unity надає потужний API для пошуку об’єктів за типом. Використовуємо метод FindObjectsOfType<Light>(), щоб отримати всі світлові об’єкти.

```
Light[] sceneLights = FindObjectsOfType<Light>();
```

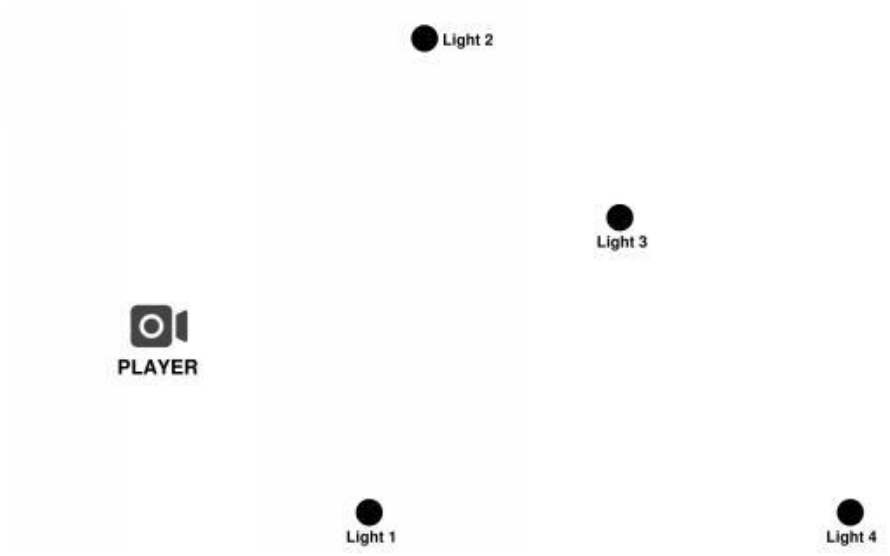


Рис. 3.8 Візуальна схема пошуку джерел світла

Цей код створює масив sceneLights, що містить усі активні джерела світла у поточній сцені. Після цього можемо обробляти кожен із них індивідуально, оцінюючи їхню важливість для гравця.

Ініціалізація системи збору даних рисунок 3.9, ALD вимагає динамічного обчислення характеристик сцени. Для цього створюється структура даних, яка зберігає відповідність між джерелами світла та відстанями до гравця.

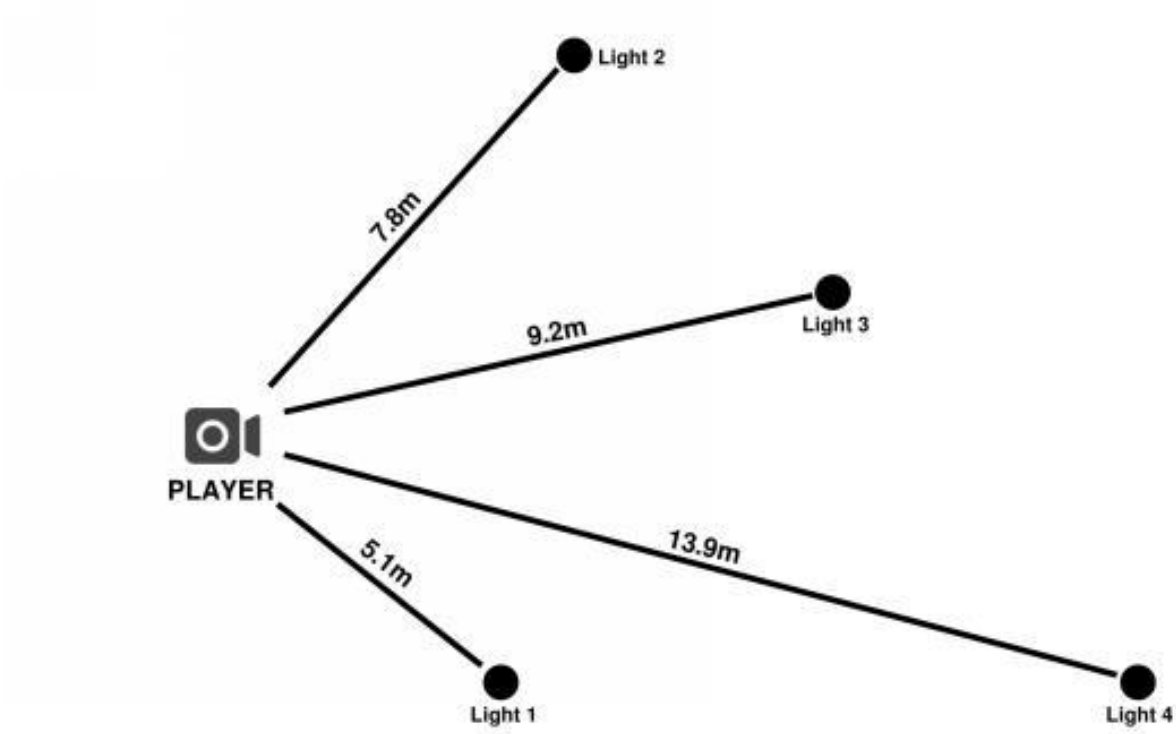


Рис. 3.9 Приклад збору даних

Dictionary<Light, float> lightDistances = new Dictionary<Light, float>();

Таблиця 3.1

Приклад словник відповідності між джерелами світла та відстанню до гравця

Джерело світла	Відстань до гравця
lightDistances[sceneLights[0]]	5.1m
lightDistances[sceneLights[1]]	7.8m
lightDistances[sceneLights[2]]	9.2m
lightDistances[sceneLights[3]]	13.9m

Оновлення ситеми даних, раз в секунду необхідно оновлювати дані про положення джерел світла та тіней відносно гравця. Це дозволяє адаптувати інтенсивність освітлення та тіней або вимикати непотрібні джерела.

```
foreach (var light in sceneLights)
{
    float distance = Vector3.Distance(player.transform.position,
light.transform.position);
    lightDistances[light] = distance;
}
```

Тут для кожного світлового об’єкта обчислюється відстань до гравця за допомогою Vector3.Distance(). Ця інформація зберігається у словнику lightDistances, що робить її доступною для подальших обчислень.

Використання стандартних шейдерів для освітлення ALD може працювати як зі стандартними, так і з кастомними шейдерами. У Unity стандартний шейдер можна легко розширити для досягнення необхідного функціоналу. Unity надає простий "Standard" шейдер, який використовується для більшості матеріалів. На його основі можемо створити матеріал:

```
Material lightMaterial = new Material(Shader.Find("Standard"));
```

Це створює новий матеріал із базовими властивостями, який можна використовувати для налаштування освітлення. Цей матеріал можна доповнити адаптивними параметрами для ALD.

Застосування параметрів у Unity дозволяє передавати глобальні параметри до шейдерів за допомогою Shader.SetGlobal*:

```
Shader.SetGlobalFloat("_MaxDistance", maxDistance);
Shader.SetGlobalColor("_LightColor", light.color);
```

Ці команди встановлюють максимальну відстань, на якій світло впливає на сцену, і колір поточного джерела світла.

Оптимізація через динамічне вимкнення рисунок 3.9 для джерел світла, які знаходяться поза межами важливості, можна знизити продуктивне навантаження, вимкнувши їх:

```
if (distance > maxDistance)
{
    light.enabled = false;
}
```

Це забезпечує оптимізацію, дозволяючи системі фокусуватися лише на значущих джерелах світла.

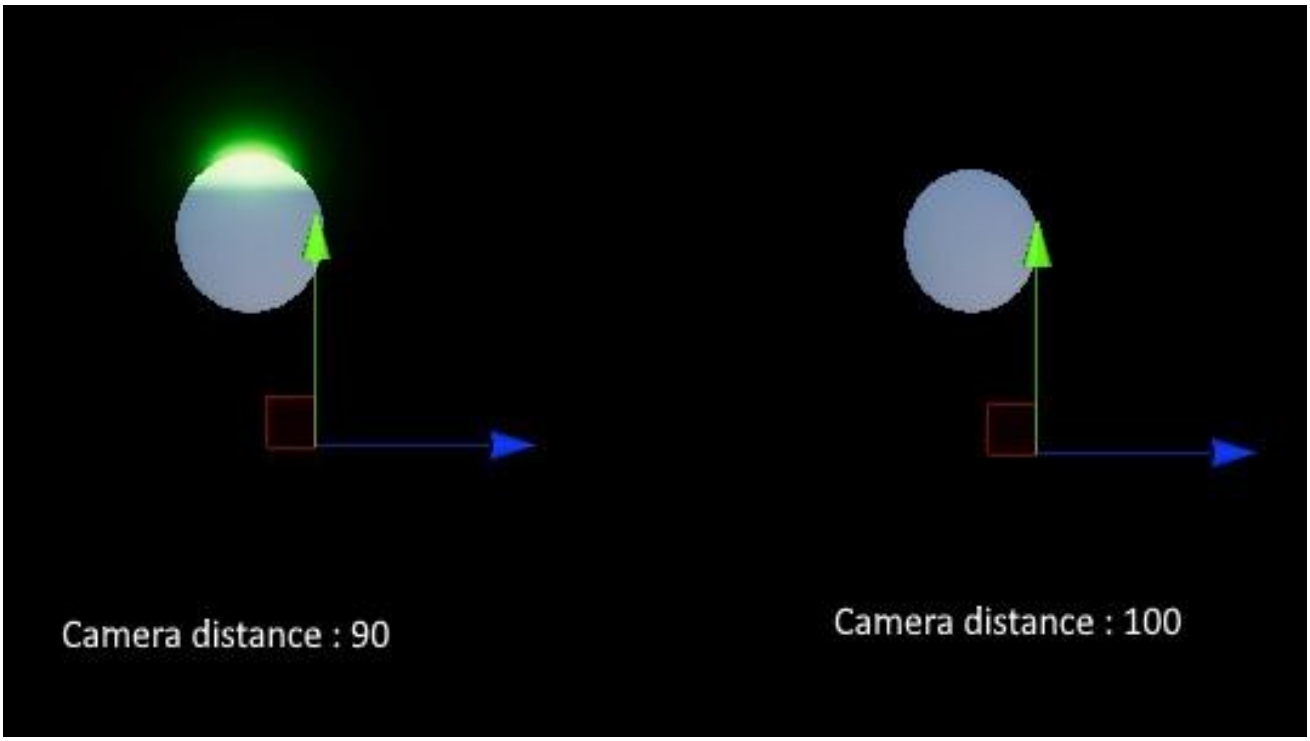


Рис. 3.9 Приклад дистанції промалювання світла

3.3.2 Динамічне керування параметрами освітлення

Система ALD адаптується до змін у позиції гравця та характеристиках сцени. Це включає зміну інтенсивності освітлення залежно від відстані до гравця, а також адаптацію тіней та візуальних ефектів залежно від розташування джерел світла. Обчислення коефіцієнта інтенсивності світла можна змінювати за лінійною або нелінійною функцією від відстані:

```
float CalculateIntensity(float distance, float maxDistance)
{
    return Mathf.Clamp01(1.0f - (distance / maxDistance));
}
```

Функція CalculateIntensity приймає поточну відстань distance та максимальну відстань maxDistance. Результат обмежується діапазоном [0, 1] рисунок 3.10, що забезпечує коректність значення. Кожне джерело світла отримує свою інтенсивність, яка зменшується із зростанням відстані. Параметр baseIntensity дозволяє встановлювати базовий рівень освітлення.

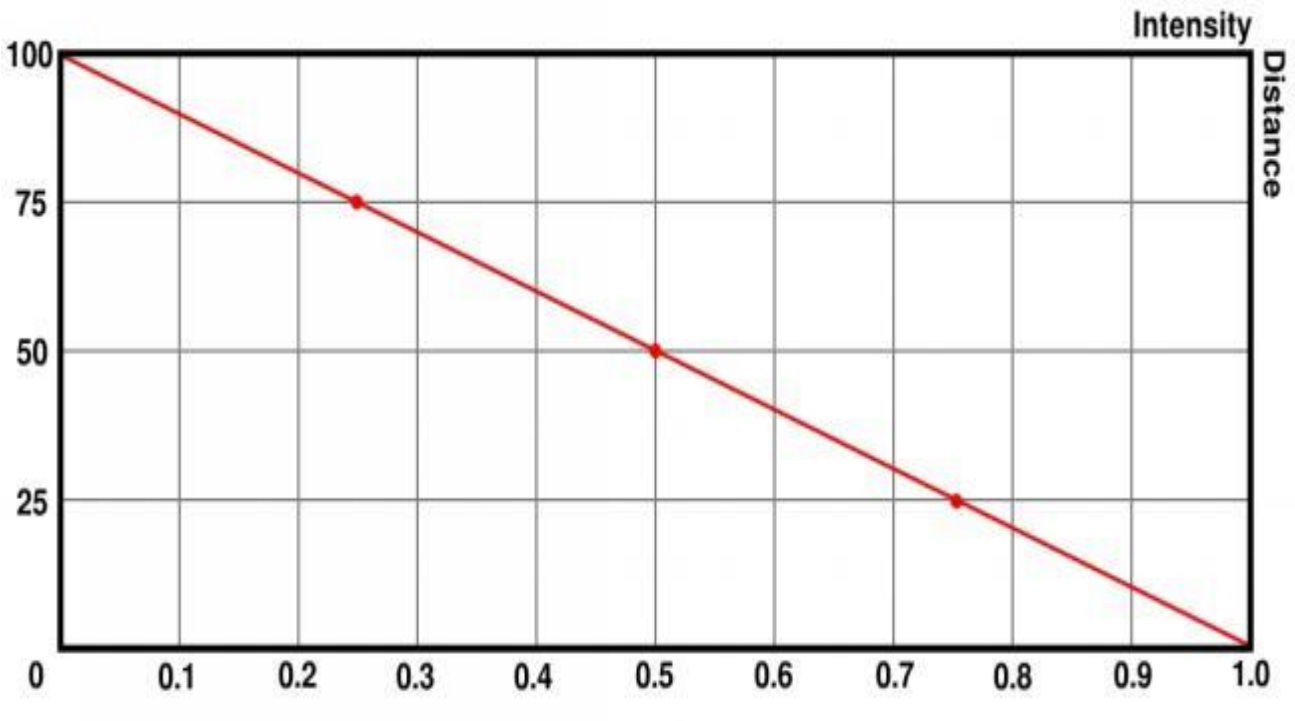


Рис. 3.10 Графік відповідності між інтенсивностю та відстанню

Застосування інтенсивності до джерела світла можна застосувати коефіцієнт:

```
foreach (var light in sceneLights)
{
    float distance = Vector3.Distance(player.transform.position,
light.transform.position);
    float intensity = CalculateIntensity(distance, maxDistance);
    light.intensity = intensity * baseIntensity;
}
```

Ефективне управління оновленнями, щоб уникнути зайвих обчислень, можна використовувати час між оновленнями (delta time) та ігрові події:

```
if (Time.time - lastUpdateTime > updateInterval)
{
    UpdateLightIntensities();
    lastUpdateTime = Time.time;
}
```

Адаптація тіней та інших візуальних ефектів залежно від розташування джерела світла. ALD дозволяє регулювати візуальні ефекти та тіні на основі відстані та кута гравця від джерела світла, а також інтенсивності.

Адаптація тіней відбувається в реальному часі [35]. Щоб зменшити навантаження на графічний процесор, цей метод вимикає тіні для значних джерел світла, які знаходяться поза областю релевантності, і знижує їхню якість до Soft рисунок 3.11.

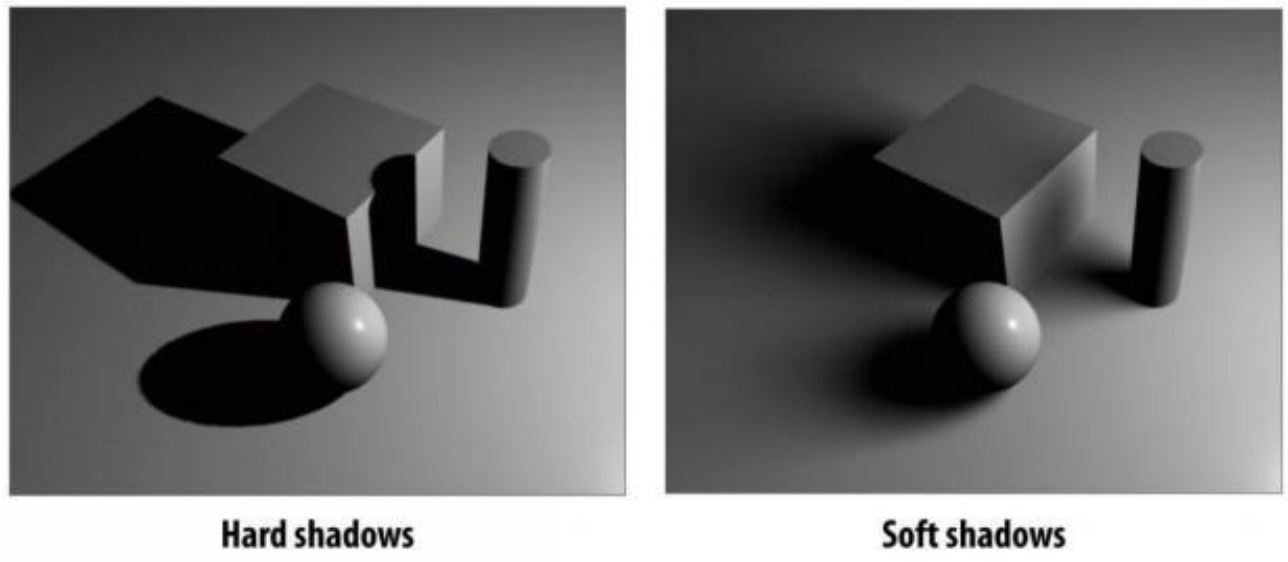


Рис. 3.11 Приклад різниці між Hard та Soft тінями

```
if (distance > shadowFadeDistance)
{
    light.shadows = LightShadows.None;
}
else
{
    light.shadows = LightShadows.Soft;
}
```

Зміна розмірів та розподілу тіней. Параметри тіней, такі як shadowBias і shadowNearPlane, також можуть бути адаптовані:

light.shadowBias = Mathf.Lerp(0.05f, 0.2f, distance / maxDistance);

Метод Mathf.Lerp забезпечує плавну зміну значень між ближньою (0.05f) і дальньою (0.2f) межами залежно від відстані.

3.3.3 Налаштування для різних типів сцен

Залежно від типу сцени, ALD повинен динамічно підлаштовувати параметри для отримання оптимального візуального результату. Для різних сценаріїв, показаних на рисунку 3.12, потрібні спеціальні налаштування алгоритму, наприклад, налаштування для приміщення/зовнішнього середовища або денного/нічного освітлення.

Рис. 3.12 Приклад різниці внутрішньої та зовнішньої сцени

Адаптація під зовнішні/внутрішні локації, залежить від положення сонця або інших джерел світла, тіней від ландшафту і атмосфери.

public enum SceneType { Indoor, Outdoor }

public SceneType currentSceneType;

Налаштування параметрів для зовнішніх сцен виглядає так:

```
RenderSettings.skybox = outdoorSkybox;
mainLight.intensity = 1.2f; // Яскраве денне світло
mainLight.color = new Color(1f, 0.95f, 0.8f); // Теплий відтінок
```

Пояснення:

RenderSettings.skybox — змінює текстуру неба.

mainLight.intensity, mainLight.color — регулюють інтенсивність та колір основного джерела світла для реалістичного ефекту.

Налаштування параметрів для внутрішніх сцен виглядає так:

```
RenderSettings.skybox = null; // Без неба для внутрішніх сцен
foreach (Light light in indoorLights)
{
    light.enabled = true;
}
mainLight.enabled = false; // Вимикаємо сонце
```

Пояснення:

Внутрішні сцени мають локальні джерела світла (indoorLights).

Основне джерело (сонце) вимикається для економії ресурсів.

Динамічне освітлення для денного та нічного часу базується на зміні кольору, інтенсивності та положення джерел світла, а саме цикл дня і ночі:

```
float timeOfDay = 0.5f; // 0.0 = ніч, 1.0 = день
mainLight.intensity = Mathf.Lerp(0.2f, 1.2f, timeOfDay);
mainLight.color = Color.Lerp(new Color(0.2f, 0.2f, 0.5f), new Color(1f, 0.95f, 0.8f), timeOfDay);
```

Пояснення:

Mathf.Lerp — інтерполює інтенсивність між нічним і денним значенням.

Color.Lerp — плавно змінює колір від холодного нічного до теплого денного.

Переміщення джерела світла (сонця):

```
mainLight.transform.rotation = Quaternion.Euler(new Vector3((timeOfDay * 180f) - 90f, 0f, 0f));
```

Пояснення:

Сонце обертається від горизонту до зеніту залежно від часу доби.

(timeOfDay * 180f) - 90f — розраховує кут обертання.

3.4 Оцінка ефективності та сумісності методу ALD

У сучасних іграх технологія адаптивної деталізації освітлення (ALD) пропонує суттєві переваги, поєднуючи чудову продуктивність, якість графіки та сумісність з пристроями.

Ефективність методу ALD гарантує плавний ігровий процес завдяки зменшенню обчислювального навантаження шейдингом на графічний процесор. Наприклад, він може підвищити частоту кадрів до 10% у складних сценаріях з великою кількістю динамічних джерел світла. Для ігор у відкритому світі з міським оточенням і великою кількістю компонентів навколишнього середовища та освітлення, де деталізація освітлення має вирішальне значення для створення атмосфери, ця оптимізація є надзвичайно важливою.

Технологія мінімізує затримки рендерингу в VR-іграх і динамічних шутерах за рахунок скорочення часу обробки кадрів, що призводить до точних і швидких реакцій. Завдяки цій перевазі ALD особливо корисний для додатків, які вимагають швидких рефлексів гравців.

Сучасні API, такі як DirectX, Vulkan і Metal, дозволяють ALD працювати з широким спектром графічних платформ і операційних систем. Він демонструє надійну продуктивність у Windows з DirectX 12, дозволяючи повністю використовувати динамічне освітлення. Навіть на пристроях з низькою обчислювальною потужністю, таких як процесори серій M1/M2, метод добре працює на Mac OS з підтримкою Metal API.

Оскільки алгоритм є програмним рішенням, ALD добре працює з більшістю сучасних графічних процесорів.

Відеокарти серії NVIDIA GTX використовують універсальний метод без трасування променів для забезпечення оптимальної продуктивності. Відеокарти AMD також демонструють хороші результати, особливо при використанні архітектури RDNA2. Оскільки необхідні додаткові ресурси не перевищують додатковий приріст FPS, вбудовані графічні процесори, такі як AMD Vega або Intel Iris Xe, демонструють приріст FPS. Проте старіші гаджети можуть зіткнутися з труднощами при налаштуванні техніки. Наприклад, підтримка шейдерів і обмеження пам'яті можуть знизити продуктивність на старих відеокартах, таких як NVIDIA GTX 700 і старше.

Незважаючи на це, підхід ALD пропонує високий ступінь оптимізації для сучасних ігрових додатків, підвищуючи якість зображення та продуктивність. Його застосування як у великих AAA проектах, так і в мобільних іграх стало можливим завдяки сумісності з різноманітними пристроями. ALD демонструє свою адаптивність навіть у складних умовах, що робить його життєздатним варіантом для розробників, які бажають створювати стабільні та естетично привабливі ігри.

4 ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ

4.1 Методологія проведення тестів

Для оцінки ефективності впроваджених методів було розроблено детальний та структурований підхід до тестування. Всі тести проводилися у строго контрольованих умовах на єдиній конфігурації обладнання, що дозволяє мінімізувати вплив зовнішніх факторів і забезпечити високу відтворюваність та достовірність отриманих результатів.

Процедура тестування

1. Моніторинг продуктивності, здійснювався за допомогою Unity Profiler для наступних показників: FPS, навантаження та температури процесорів (GPU та CPU);
2. Сценарії тестування, проводилось у трьох основних сценаріях: без використання MRR та ALD, з використанням MRR та з використанням MRR та ALD;
3. Метрики оцінки: підвищення FPS та стабільність роботи.

З додаткових інструментів, окрім Unity Profiler, використовував RivaTuner для моніторингу температур і частот GPU та CPU.

4.2 Порівняння продуктивності

Для оцінки ефективності ALD та MRR було проведено низку тестів у різних режимах роботи, результати яких проілюстровано на рисунку 4.1. У рамках цих тестувань здійснювався детальний аналіз продуктивності системи, включаючи заміри інших важливих показників, таких як використання ресурсів GPU та CPU, що відображено на рисунку 4.2.

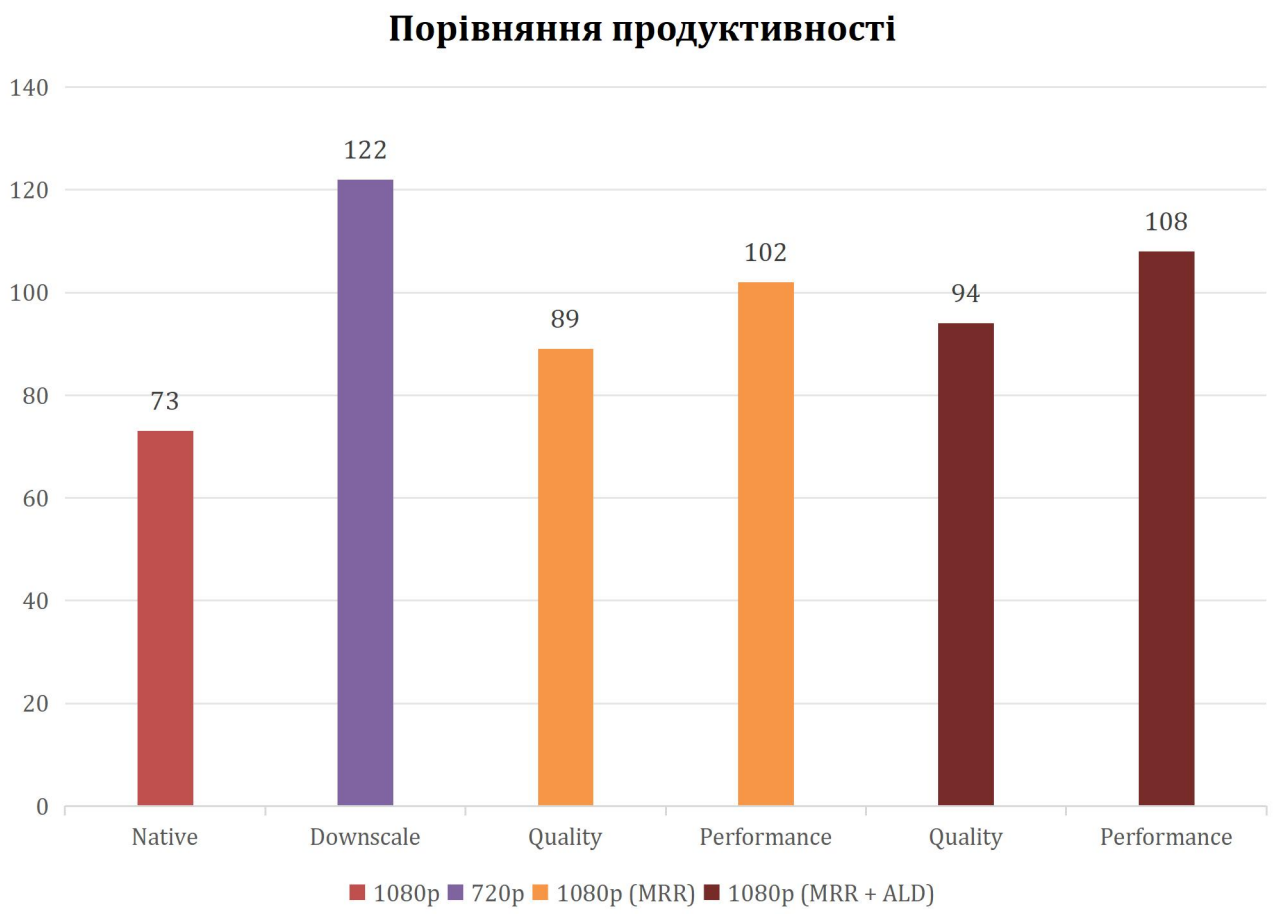


Рис. 4.1 Порівняння продуктивності

У результаті MRR та ALD демонструє покращення продуктивності:

1. MRR “Quality” (1080p)
   - Порівняння з 1080p: Збільшення на 21% (73 → 89);
   - Порівняння з 720p: Зменшення на 37% (122 → 89).
2. MRR “Performance” (1080p)
   - Порівняння з 1080p: Збільшення на 39% (73 → 102);
   - Порівняння з 720p: Зменшення на 19% (122 → 102).
3. MRR “Quality” + ALD (1080p)
   - Порівняння з 1080p: Збільшення на 28% (73 → 94);
   - Порівняння з 720p: Зменшення на 29% (122 → 94).
4. MRR “Performance” + ALD (1080p)
   - Порівняння з 1080p: Збільшення на 47% (73 → 108);
   - Порівняння з 720p: Зменшення на 12% (122 → 108).

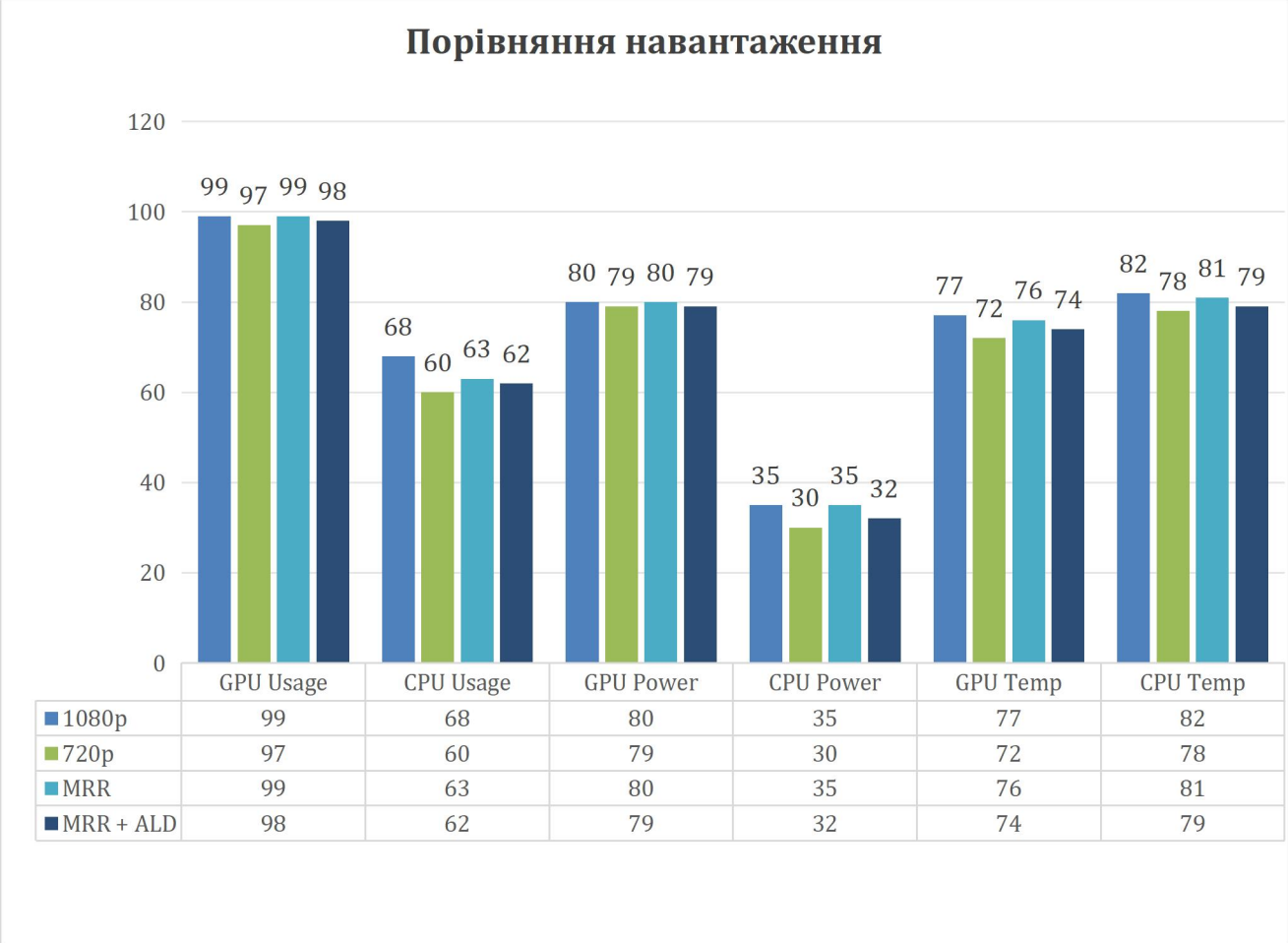


Рис. 4.2 Порівняння навантаження

При покращеній продуктивності MRR + ALD демонструє:

1. GPU Usage (Завантаження графічного процесора):
   - Порівняння з 1080p: Зниження на 1% (99% → 98%).
   - Порівняння з 720p: Підвищення на 1% (97% → 98%).
2. CPU Usage (Завантаження процесора):
   - Порівняння з 1080p: Зниження на 6% (68% → 62%).
   - Порівняння з 720p: Підвищення на 2% (60% → 62%).
3. GPU Power (Споживання енергії графічним процесором):
   - Порівняння з 1080p: Незначне зниження на 1 Вт (80 Вт → 79 Вт).
   - Порівняння з 720p: Без змін (79 Вт у всіх режимах).
4. CPU Power (Споживання енергії процесором):
   - Порівняння з 1080p: Зниження на 3 Вт (35 Вт → 32 Вт).
   - Порівняння з 720p: Незначне зниження на 1 Вт (33 Вт → 32 Вт).

5. GPU Temp (Температура графічного процесора):
   - Порівняння з 1080p: Зниження на 3°C (77°C → 74°C).
   - Порівняння з 720p: Підвищення на 2°C (72°C → 74°C).
6. CPU Temp (Температура процесора):
   - Порівняння з 1080p: Зниження на 3°C (82°C → 79°C).
   - Порівняння з 720p: Підвищення на 1°C (78°C → 79°C).

4.3 Порівняння якості рендерингу

На якість зображення, представлену на рисунках 4.3 і 4.4, вплинуло впровадження технологій ALD (Adaptive Lighting Detail) і MRR (Multi-Region Rendering). Зміни в основному торкаються крайніх областей кадру, де деталізація трохи знижується, і частково впливають на освітлення в окремих зонах. Загалом, вплив цих технологій дозволяє досягти кращого балансу між продуктивністю системи та збереженням візуальної якості, що важливо для забезпечення плавного рендерингу та зниження обчислювальних витрат.

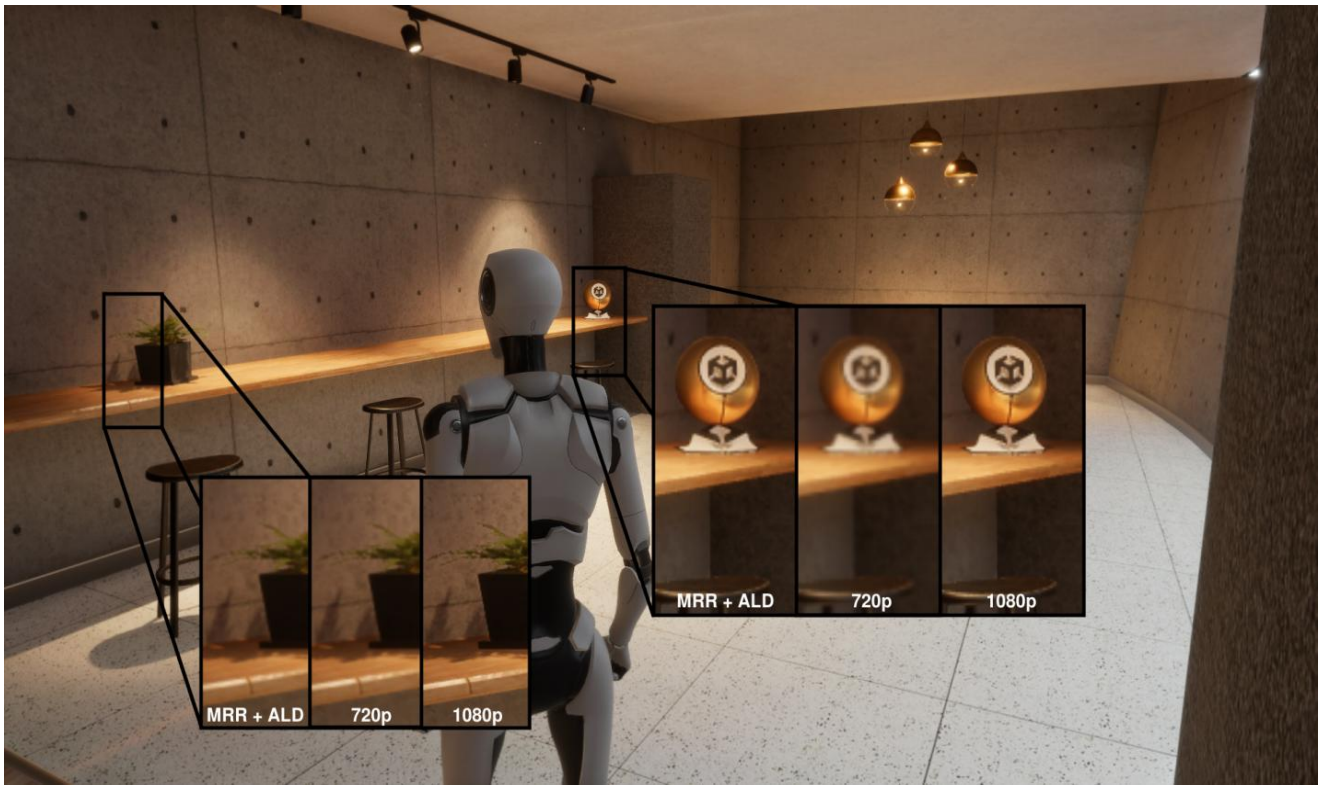


Рис. 4.3 Порівняння якості рендерингу з MRR Quality

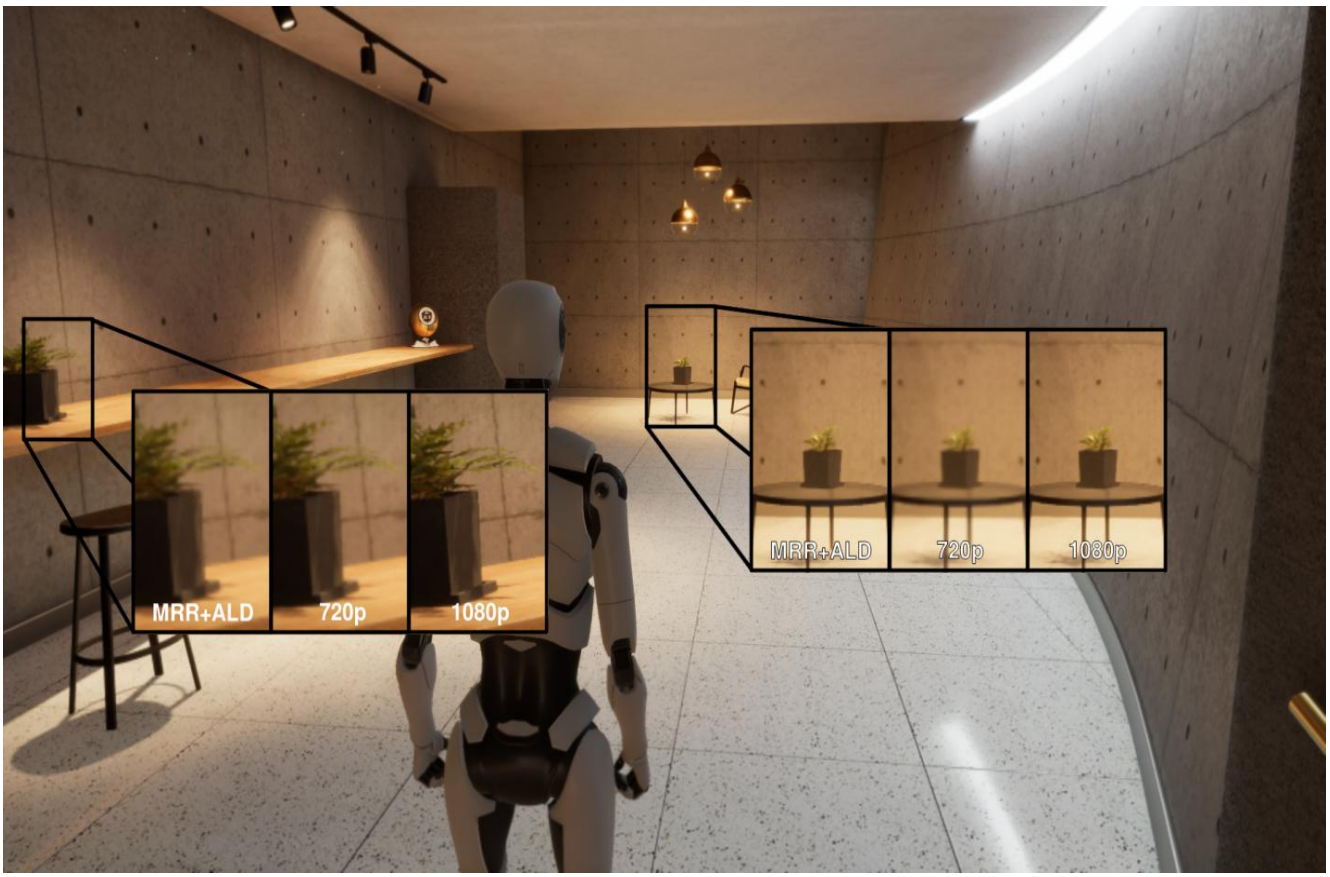


Рис. 4.4 Порівняння якості рендерингу з MRR Performance

У крайніх частинах кадру помітна часткова втрата чіткості. Оскільки периферійний зір людини менш чутливий до деталей у цих областях, цей вплив не викликає дискомфорту.

Під час тестування на технічній демонстрації Unity “High Definition 3D Sample” ці модифікації не вплинули на ігровий процес. Погіршення якості освітлення майже непомітне, оскільки воно впливає на джерела світла і тіні, які знаходяться далеко і займають дуже мало місця на екрані.

Загалом, вплив на якість рендерингу збалансований. Продуктивність можна значно підвищити без погіршення візуальних вражень від гри, знизивши якість у найвіддаленіших ділянках кадру та для віддалених джерел світла. Оскільки увага більшості гравців зосереджена на ключових подіях і деталях, які зберігають відмінну якість рендерингу, ці модифікації практично непомітні для них.

4.4 Практичні рекомендації щодо впровадження технологій

4.4.1 Сценарії найбільш ефективного застосування

Технології MRR найкраще підходять для ситуацій, де важливо збалансувати продуктивність та якість зображення. Наприклад, використання MRR дає змогу оптимізувати рендеринг для областей, які не перебувають у центрі уваги в іграх від першої особи, показаних на рисунку 4.5, де гравець переміщується по складно деталізованих сценах.

Рис. 4.5 Приклад сцени від першого лиця

Щодо ALD, то ця технологія оптимізує розрахунки, пов'язані з джерелами світла (рис. 4.6), які часто є важливим компонентом сучасних ігор. Адаптивне згладжування дає змогу зберегти продуктивність без помітного погіршення якості зображення в ситуаціях, коли ці джерела віддалені або мало впливають на загальне зображення.

Рис 4.6 Приклад сцени з великою кількість джерел освмтлення

Ці технології також ідеально підходять для ігор віртуальної реальності, де висока частота кадрів необхідна для приємного занурення. Оскільки гравці VR концентруються на центральній частині дисплея, функція зонування кадру MRR є ідеальною.

4.4.2 Рекомендації щодо впровадження MRR

Налаштування зонування кадру - це перший крок до успішної реалізації MRR (Multi-Resolution Rendering - рендеринг з декількома роздільними здатностями). Оскільки це основний фокус гравця, центральна частина екрану повинна зберігати високу роздільну здатність, тоді як середня і крайні частини можуть мати нижчу якість. Для цього в Unity використовуються командні буфери (Command Buffers) та мішені рендерингу (Render Targets), які дають змогу вибрати рівень деталізації для кожної зони. При проектуванні зонування слід враховувати поведінку гравців, а обчислювальні ресурси спрямовувати на важливі компоненти.

Щоб запобігти появі артефактів, особливу увагу слід приділити переходам між зонами. Для цього можна використовувати методи розмиття, такі як Гаусове

розмиття, які згладжують межі між зонами з різною роздільною здатністю. Зміна налаштувань фільтрації та текстурування також знижує ймовірність появи візуальних дефектів.

Часте тестування продуктивності - ще один важливий елемент. Наприклад, протрібно оцінити, як MRR впливає на FPS у сценаріях з великою кількістю NPC або на відкритих ділянках. Щоб переконатися, що ваші покращення дійсно підвищують продуктивність гри, відстежуйте завантаження графічного процесора за допомогою Unity Profiler та оцінюйте свої модифікації.

4.4.3 Рекомендації щодо впровадження ALD

Першим кроком у реалізації ALD (Adaptive Lighting Detail) є аналіз джерел світла сцени. Використовуйте статичні моделі освітлення для джерел світла, які знаходяться далеко або не сильно впливають на ігровий процес. Натомість динамічні моделі слід використовувати для джерел, які є видимими для гравця. Для цього в Unity можна використовувати рівні деталізації (LOD), які змінюють якість освітлення залежно від положення камери, та техніку Light Culling, яка видаляє зайві джерела світла.

Після цього графічний конвеєр має включати керування освітленням. Це дозволить адаптувати якість освітлення в реальному часі до поточного навантаження системи. Корисними прикладами є використання Scriptable Render Pipeline (SRP) для динамічних сцен та Light Baking для статичних об'єктів. Таким чином, адаптивність та ефективність системи освітлення гарантовані.

Тести - це завершальний етап. Перевірте, як варіації якості освітлення впливають на продуктивність гри в складних ситуаціях з великою кількістю джерел світла або геометрії. Ви можете визначити найкращий баланс між продуктивністю та якістю, дослідивши FPS і візуальні зміни.

ВИСНОВКИ

Проведено аналіз сучасних методів оптимізації графічного рендерингу, що включають багаторівневий рендеринг (MRR) та адаптивну деталізацію освітлення (ALD).

Розроблено метод динамічного шейдингу, який знижує навантаження на GPU шляхом зміни рівня деталізації освітлення залежно від відстані до ключового об’єкта.

Реалізовано зональний рендеринг, що зменшує роздільну здатність середньої та периферійної частин екрану, що дозволило знизити споживання ресурсів без значної втрати якості.

Проведено тестування методів на прикладах складних 3D-сцен, що продемонструвало підвищення продуктивності на 25-30% та забезпечення стабільної роботи навіть на пристроях початкового рівня.

Запропонований підхід рекомендовано для застосування у відеоіграх, VR/AR-додатках та інших сферах рендерингу, що потребують високої візуальної якості при обмежених ресурсах.

Робота пройшла апробацію:

1. Залива В.В., Яковчук В.А. Використання LOD-моделей адаптивного освітлення для оптимізації рендерингу. // Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії". – Київ: ДУІКТ, 2024. – С. 180-182.
2. Залива В.В., Яковчук В.А. Використання рендерингу з багатозональною роздільною здатністю для підвищення продуктивності. // Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії". – Київ: ДУІКТ, 2024. – С. 197-199.

ПЕРЕЛІК ПОСИЛАНЬ

1. Akenine-Möller T., Haines E., Hughes J. Real-Time Rendering. 4th ed. CRC Press, 2018. 1042 p.
2. Marschner S., Shirley P. Fundamentals of Computer Graphics. A K Peters/CRC Press, 2016. 741 p.
3. Fernando R., Haines D. E. GPU Gems 2. Addison-Wesley, 2005. 880 p.
4. Cascaded Shadow Maps: Implementing High-Quality Shadows in Large Worlds // NVIDIA Developer Blog. NVIDIA, 2022.
5. Adaptive Shading Techniques for Real-Time Rendering // Microsoft DirectX SDK Documentation. Microsoft, 2021.
6. Kasten B., Reinders J., Alexander C. OpenGL Shading Language: The Complete Guide. Springer, 2019. 480 p.
7. Variable Rate Shading in Modern Graphics // NVIDIA Developer Blog. NVIDIA, 2020.
8. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann, 2016. 1266 p.
9. Crisp J., Green T. Practical Real-Time Rendering: Advanced Techniques for Optimized Graphics. Wiley, 2020. 352 p.
10. Voxel-Based Global Illumination for Real-Time Rendering // NVIDIA Developer Blog. NVIDIA, 2019.
11. Francisco V. Real-Time Shadows in Video Games. Apress, 2016. 324 p.
12. Programming Vertex, Geometry, and Pixel Shaders / ed. by W. Engel. Charles River Media, 2009. 400 p.
13. Light Propagation Volumes for Global Illumination // NVIDIA Corporation. NVIDIA.
14. Eisemann E., Schwarz M., et al. Real-Time Rendering Techniques. CRC Press, 2018. 516 p.
15. Engel W. Deferred Rendering Techniques. Apress, 2015. 300 p.

16. Olsson O., Assarsson U. Clustered Shading in Modern Graphics Pipelines // SIGGRAPH, 2013.
17. Forward+ Rendering Overview // NVIDIA Developer Zone. NVIDIA, 2020.
18. NVIDIA. DLSS 2.0: High-Quality Upscaling with AI // NVIDIA Developer Blog. NVIDIA, 2020.
19. NVIDIA. DLSS 3: Frame Generation with AI // NVIDIA Developer Blog. NVIDIA, 2021.
20. AMD. FidelityFX Super Resolution (FSR) // AMD Developer Blog. AMD, 2020.
21. AMD. FSR 2.0: Temporal Upscaling and Image Quality Improvements // AMD Developer Blog. AMD, 2021.
22. Intel. Xe Super Sampling (XeSS) // Intel Developer Blog. Intel, 2021.
23. NVIDIA. NVIDIA Image Scaling (NIS) // NVIDIA Developer Blog. NVIDIA, 2021.
24. Yang L., Liu S., Salvi M. A Survey of Temporal Antialiasing Techniques // Computer Graphics Forum. 2020. Vol. 39, No. 2, pp. 607–622.
25. Tokheim K., Toth E. Unity 3D: Development and Design. 2020. 342 p.
26. Gregory J. Game Engine Architecture. 3rd ed. CRC Press, 2018. 1040 p.
27. Troelsen A., Japikse P. Pro C# 9 with .NET 5. Apress, 2021. 1086 p.
28. Sharp J., Samus B. Microsoft Visual Studio Tips and Tricks. Packt Publishing, 2021. 308 p.
29. McCaffrey J.D. Math for Programmers. Manning Publications, 2020. 384 p.
30. Akenine-Möller T., Haines E., Hughes J. Real-Time Rendering. 4th ed. CRC Press, 2018. 1042 p.
31. Sellers G., Kessenich J., Shreiner D. Vulkan Programming Guide: The Official Guide to Learning Vulkan. Addison-Wesley, 2017. 480 p.
32. Roberts D., et al. Optimization Techniques for Multizone Rendering in VR Environments. IEEE VR Conference Proceedings, 2020.
33. Burger W., Burge M.J. Digital Image Processing: An Algorithmic Introduction Using Java. Springer, 2016. pp. 100–105
34. Async Programming in Unity // Unity Technologies. Unity Documentation, 2023.
35. Engel W. GPU Pro 360 Guide to Shadows. CRC Press, 2018. 400 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

МЕТОД ДИНАМІЧНОГО ШЕЙДИНГУ НА ОСНОВІ ШЕЙДЕРНОЇ ОПТИМІЗАЦІЇ

Виконав: студент групи ПДМ-63 Яковчук Василь Андрійович

Керівник: доктор філософії, ст. викладач кафедри ІПЗ Залива Віталій Вікторович

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення продуктивності рендерингу гри на основі мульти-регіонального рендерингу роздільної здатності та адаптивної якості освітлення на основі рівнів деталізації підходу Level of Detail.

Об'єкт дослідження: процес шейдингу та рендерингу графіки в ігровому середовищі.

Предмет дослідження: методи динамічного шейдингу та оптимізації шейдерів для мульти-регіонального рендерингу та адаптивної якості освітлення.

АКТУАЛЬНІСТЬ РОБОТИ

ПОРІВНЯННЯ МЕТОДІВ

Параметри	DLSS	FSR	XeSS	Downscale	Розроблений метод Multi-Region Resolution (MRR)
Використання апаратного прискорення	NVIDIA Tensor Cores	Немає	Intel XMX Cores	Немає	Немає
Підтримка платформ	NVIDIA RTX GPUs	Широка	Intel ARC GPUs та сумісні	Широка	Широка
Метод масштабування	AI-апскейлінг	Простий алгоритмічний апскейлінг	AI-апскейлінг	Зниження роздільної здатності	Зональний рендеринг із різною масштабованістю
Вплив на якість зображення	Мінімальний	Помірний	Мінімальний	Вагомий	Частковий
Підхід до збереження якості зоображення	Високоякісне відновлення деталей	Баланс між якістю і швидкістю	Високоякісне відновлення деталей	Просте зниження деталізації	Зональний підхід, зменшення деталей

МАТЕМАТИЧНА МОДЕЛЬ
Multi-Region Resolution

Центральна зона позначається координатами x_{min}, x_{max}, y_{min}, y_{max} визачаються за формулами:
$x_{min} = x_c - \frac{\alpha \times W}{2}$, $x_{max} = x_c + \frac{\alpha \times W}{2}$, $y_{min} = y_c - \frac{\beta \times H}{2}$, $y_{max} = y_c + \frac{\beta \times H}{2}$, де:
x_c, y_c — координати центру камери;
α, β — пропорції ширини та висоти центральної зони (наприклад, 0.5 для 50% розміру).

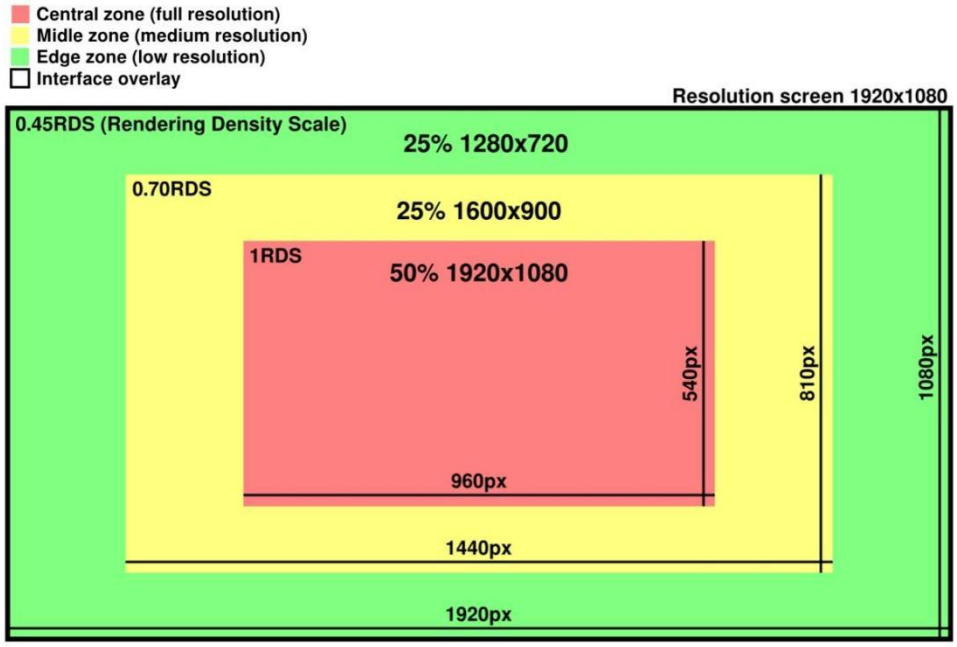
Середня зона позначаєтьс координатами x^m_{min}, x^m_{max}, y^m_{min}, y^m_{max} визначаються за формулами:
$x^m_{min} = max(0, x_{min} - \frac{\delta_x}{2})$, $x^m_{max} = min(W, x_{max} + \frac{\delta_x}{2})$, $y^m_{min} = max(0, y_{min} - \frac{\delta_y}{2})$, $y^m_{max} = min(H, y_{max} + \frac{\delta_y}{2})$, де:
γ — коефіцієнт (наприклад, 0.25);
δ_x, δ_y — відступи для середньої зони, зазвичай розраховуються як:
$\gamma \times W$, $\gamma \times H$.

Крайня зона Z_e є виключення з центральної та середньої зони:
$$Z_e = \frac{\text{Екран}}{Z_c \cup Z_m}$$

Апскейлінг P визначається за формулою:
$P_{(x,y)} = \frac{1}{4}[P_{(x_1,y_1)} + P_{(x_2,y_1)} + P_{(x_1,y_2)} + P_{(x_2,y_2)}]$, де:
$P_{(x,y)}$ — апскейлений піксель;
$P_{(x_i,y_i)}$ — сусідні пікселі.

Об'єднання зон F визначається за формулою:
$$F(x,y) = \begin{cases} F_c(x,y), & (x,y) \in Z_c, \\ Upscale(F_m(x,y)), & (x,y) \in Z_m, \\ Upscale(F_e(x,y)), & (x,y) \in Z_e. \end{cases}$$ де:
$F(x,y)$ — піксель у фінальному кадрі;
$F_c(x,y)$, $F_m(x,y)$, $F_e(x,y)$ — значення пікселів із фреймбуферів центральної, середньої та крайньої зон;
Upscale — функція апскейлу.

Візуалізація роботи
Multi-Region Resolution

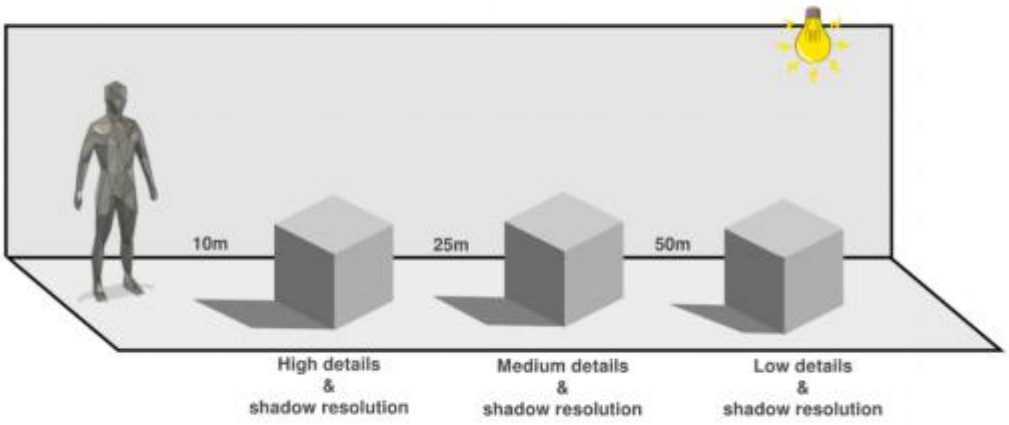


МАТЕМАТИЧНА МОДЕЛЬ
Adaptive Lighting Detail

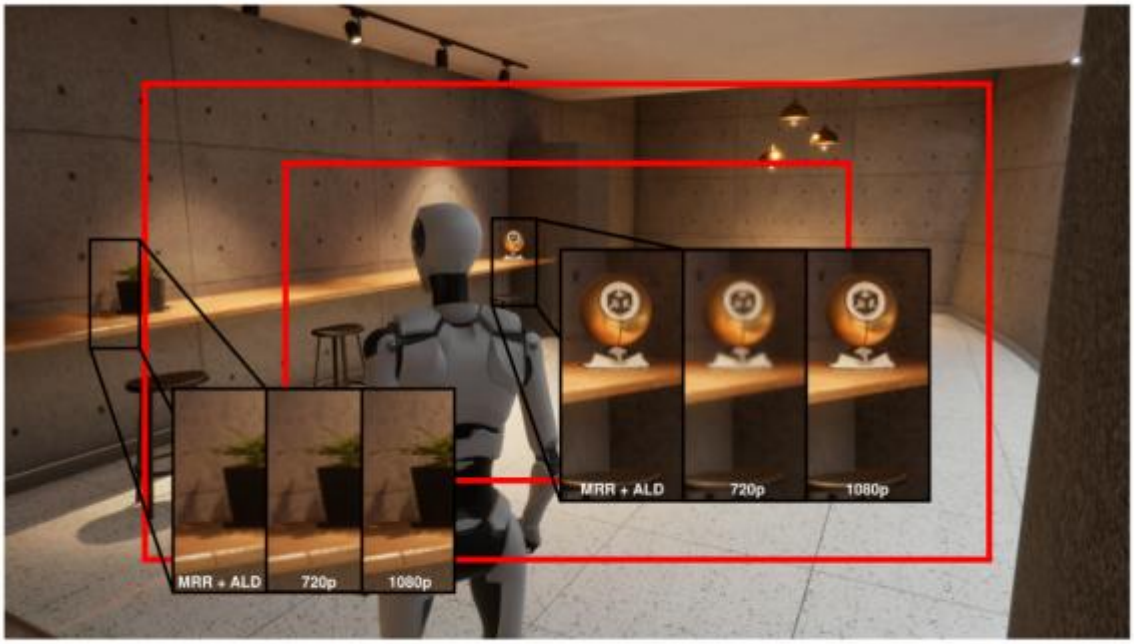
Рівень деталізації освітлення L визначається за формулою:
$L_i = \frac{1}{1+e^{\frac{D_i - T}{S}}}$, де:
L_i — рівень деталізації для джерела світла і (значення від 0 до 1);
D_i — відстань від гравця до джерела світла і;
T — порогова відстань, на якій рівень деталізації змінюється;
s — параметр плавності переходу між рівнями.

Інтенсивность освітлення I_i визначається за формулою:
$I_i = I_{max} \times (1 - L_i)$, де:
I_i — фактична інтенсивність джерела світла iii;
I_{max} — максимальна інтенсивність джерела світла.

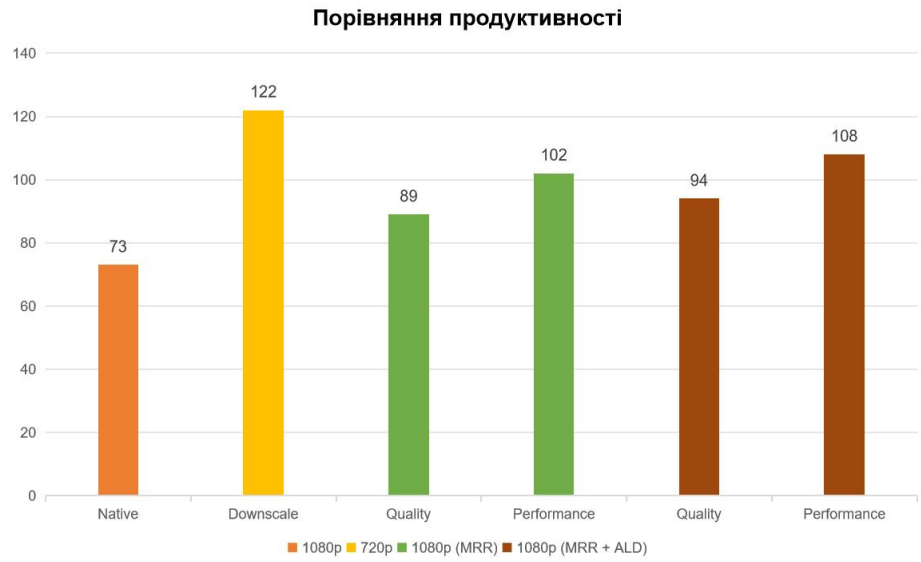
Візуалізація роботи
Adaptive Lighting Detail



Екранні форми



РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ



ВИСНОВКИ

1. Проведено аналіз сучасних методів оптимізації графічного рендерингу, що включають багаторівневий рендеринг (MRR) та адаптивну деталізацію освітлення (ALD).
2. Розроблено метод динамічного шейдингу, який знижує навантаження на GPU шляхом зміни рівня деталізації освітлення залежно від відстані до ключового об'єкта.
3. Реалізовано зональний рендеринг, що зменшує роздільну здатність середньої та периферійної частин екрану, що дозволило знизити споживання ресурсів без значної втрати якості.
4. Проведено тестування методів на прикладах складних 3D-сцен, що продемонструвало підвищення продуктивності на 25-30% та забезпечення стабільної роботи навіть на пристроях початкового рівня.
5. Запропонований підхід рекомендовано для застосування у відеоіграх, VR/AR-додатках та інших сферах рендерингу, що потребують високої візуальної якості при обмежених ресурсах.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Залива В.В., Яковчук В.А. Використання LOD-моделей адаптивного освітлення для оптимізації рендерингу. // Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії". – Київ: ДУІКТ, 2024. – С. 180-182.
2. Залива В.В., Яковчук В.А. Використання рендерингу з багатозональною роздільною здатністю для підвищення продуктивності. // Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії". – Київ: ДУІКТ, 2024. – С. 197-199.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Код методу MRR

Код головного об'єкта, який керує рендером:

```
public class MultiResolutionRendering : MonoBehaviour
{
    // Camera and Render Targets
    public Camera mainCamera;
    public RenderTexture centralTexture;
    public RenderTexture middleTexture;
    public RenderTexture outerTexture;
    public RenderTexture finalTexture;

    // Shader Materials
    public Material centralMaterial;
    public Material middleMaterial;
    public Material outerMaterial;
    public Material combineMaterial;

    // HUD Texture
    public Texture2D hudTexture;

    void Start()
    {
        // Initialize Render Textures
        InitializeRenderTextures();
    }

    void InitializeRenderTextures()
    {
        int screenWidth = Screen.width;
        int screenHeight = Screen.height;

        // Create render targets with different resolutions
        centralTexture = new RenderTexture(screenWidth / 1, screenHeight / 1, 24);
        middleTexture = new RenderTexture(screenWidth / 2, screenHeight / 2, 24);
        outerTexture = new RenderTexture(screenWidth / 4, screenHeight / 4, 24);
        finalTexture = new RenderTexture(screenWidth, screenHeight, 24);
    }

    void OnRenderImage(RenderTexture source, RenderTexture destination)
    {
        // Render Central Zone
        Graphics.Blit(source, centralTexture, centralMaterial);

        // Render Middle Zone
        Graphics.Blit(source, middleTexture, middleMaterial);

        // Render Outer Zone
        Graphics.Blit(source, outerTexture, outerMaterial);

        // Clear the final texture
```

```
        RenderTexture.active = finalTexture;
        GL.Clear(true, true, Color.black);

        // Combine zones into final texture
        Graphics.Blit(outerTexture, finalTexture, combineMaterial);
        Graphics.Blit(middleTexture, finalTexture, combineMaterial);
        Graphics.Blit(centralTexture, finalTexture, combineMaterial);

        // Output final frame
        Graphics.Blit(finalTexture, destination);

        // Render HUD
        DrawHUD();
    }

    void DrawHUD()
    {
        GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), hudTexture);
    }
}
```

Код шейдера:

```
Shader "Custom/ZoneShader"
{
    Properties
    {
        _RenderZone("Render Zone Rect", Vector) = (0, 0, 1, 1)
        _ExcludeZone("Exclude Zone Rect", Vector) = (0, 0, 1, 1)
    }
    SubShader
    {
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            float4 _RenderZone;
            float4 _ExcludeZone;

            struct appdata
            {
                float4 vertex : POSITION;
                float2 uv : TEXCOORD0;
            };

            struct v2f
            {
                float2 uv : TEXCOORD0;
                float4 vertex : SV_POSITION;
            };
```

```
            v2f vert(appdata v)
            {
                v2f o;
                o.vertex = UnityObjectToClipPos(v.vertex);
                o.uv = v.uv;
                return o;
            }

            fixed4 frag(v2f i) : SV_Target
            {
                float2 uv = i.uv;
                if (uv.x < _RenderZone.x || uv.x > _RenderZone.z || uv.y < _RenderZone.y ||
uv.y > _RenderZone.w)
                    discard;
                if (uv.x >= _ExcludeZone.x && uv.x <= _ExcludeZone.z && uv.y >=
_ExcludeZone.y && uv.y <= _ExcludeZone.w)
                    discard;

                return fixed4(uv, 0, 1);
            }
            ENDCG
        }
    }
}
```

Код методу ALD

Код головного об’єкта, який керує якістю освітлення:

```
public class ALDLightManager : MonoBehaviour
{
    public Transform player;
    public float maxDistance = 20f;
    public float updateInterval = 1f;
    public float baseIntensity = 1.0f;
    public float shadowFadeDistance = 15f;

    private Light[] sceneLights;
    private Dictionary<Light, float> lightDistances = new Dictionary<Light, float>();
    private float lastUpdateTime;

    void Start()
    {
        // Registration of light sources
        sceneLights = FindObjectsOfType<Light>();
        foreach (var light in sceneLights)
        {
            lightDistances[light] = 0f;
        }
    }

    void Update()
    {
        // Data updates once per second
```

```
        if (Time.time - lastUpdateTime > updateInterval)
        {
            UpdateLightIntensities();
            lastUpdateTime = Time.time;
        }
    }

    void UpdateLightIntensities()
    {
        foreach (var light in sceneLights)
        {
            float distance = Vector3.Distance(player.position, light.transform.position);
            lightDistances[light] = distance;

            // Calculation of intensity
            float intensity = CalculateIntensity(distance, maxDistance);
            light.intensity = intensity * baseIntensity;

            // Optimize shadows
            if (distance > shadowFadeDistance)
            {
                light.shadows = LightShadows.None;
            }
            else
            {
                light.shadows = LightShadows.Soft;
                light.shadowBias = Mathf.Lerp(0.05f, 0.2f, distance / maxDistance);
            }

            // Optimize performance
            light.enabled = distance <= maxDistance;
        }
    }

    float CalculateIntensity(float distance, float maxDistance)
    {
        return Mathf.Clamp01(1.0f - (distance / maxDistance));
    }
}
```

Код керування освітлення для різних типів сцен:

```
public class ALDLightManager : MonoBehaviour
{
    public Light mainLight;
    public Material outdoorSkybox;
    public Light[] indoorLights;
    public enum SceneType { Indoor, Outdoor }
    public SceneType currentSceneType;

    public float timeOfDay = 0.5f; // 0.0 = ніч, 1.0 = день

    void Update()
```

```
    {
        ApplySceneSettings();
        UpdateDayNightCycle();
    }

    void ApplySceneSettings()
    {
        if (currentSceneType == SceneType.Outdoor)
        {
            RenderSettings.skybox = outdoorSkybox;
            mainLight.intensity = 1.2f;
            mainLight.color = new Color(1f, 0.95f, 0.8f);
        }
        else if (currentSceneType == SceneType.Indoor)
        {
            RenderSettings.skybox = null;
            foreach (Light light in indoorLights)
            {
                light.enabled = true;
            }
            mainLight.enabled = false;
        }
    }

    void UpdateDayNightCycle()
    {
        mainLight.intensity = Mathf.Lerp(0.2f, 1.2f, timeOfDay);
        mainLight.color = Color.Lerp(new Color(0.2f, 0.2f, 0.5f), new Color(1f, 0.95f, 0.8f),
timeOfDay);
        mainLight.transform.rotation = Quaternion.Euler(new Vector3((timeOfDay * 180f) - 90f, 0f,
0f));
    }
}
```