ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методика розробки ігрового застосунку з використанням графічного процесора на базі RTX»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

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

	Даніель РИХТИК
(підпис)	*Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:	здобувач вищої освіти гр. КІД-42	
	Данілеь РИХТИК	
	Ім'я, ПРІЗВИЩЕ	
Керівник:	Андрій ЛЕМЕШКО	
науковий ступінь, вчене звання	Ім'я, ПРІЗВИЩЕ	
Рецензент:		
науковий ступінь, вчене звання	Ім'я, ПРІЗВИЩЕ	

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерної інженерії

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедрою Наталія ЛАЩЕВСЬКА
_____Ім'я, ПРІЗВИЩЕ
«_____» _____ 2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Рихтик Даніель Миколайович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: _____
керівник кваліфікаційної роботи Андрій ЛЕМЕШКО,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «__ _____» _____2024р. № _____
2. Строк подання кваліфікаційної роботи «_____» _____ 2024р.
3. Вихідні дані до кваліфікаційної роботи: _____
_____
_____
_____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні основи з архітектури графічних процесорів та методик розробки
2. ігрових застосувань.
3. Методологія розробки з урахуванням використання платформи Unity3D.
4. Технічний аспект, та експериментальна частина.
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «_____» _____ 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	26.02-22.03.2024	Виконано
2	Обробка матеріалу	23.03-05.04.2024	Виконано
3	Розробка теоретичної частини	06.04-10.04.2024	Виконано
4	Розробка ігрового застосунку	11.04-30.04.2024	Виконано
5	Впровадження графічних технологій у застосунку	01.05-03.05.2024	Виконано
6	Висновки за результатами аналізу	04.05-08.05.2024	Виконано
7	Розробка презентації	09.05-11.05.2024	Виконано
8	Передзахист	14.05-17.05.2024	Виконано
9	Здача в деканат	25.05-1.06.2024	Виконано

Здобувач(ка) вищої освіти _____ Даніель РИХТИК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Андрій ЛЕМЕШКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 61 стор., 39 рис., 2 табл., 10 джерел.

Мета роботи – дослідження можливості розробки методики для ефективного використання графічного процесора на базі RTX у процесі розробки ігрового застосунку на платформі Unity3D.

Об'єкт дослідження – графічний процесор на базі RTX та його використання у розробці ігрового застосунку на платформі Unity3D.

Предмет дослідження – методика розробки ігрового застосунку з використанням графічного процесора на базі RTX на платформі Unity3D.

Короткий зміст роботи: дипломна робота присвячена розробці методики створення ігрового застосунку з використанням графічного процесора на базі RTX на платформі Unity3D. У роботі детально розглядається архітектура графічного процесора RTX та його технічні можливості, а також особливості інтеграції з Unity3D для досягнення найвищої продуктивності та якості графіки. Окрім цього, розробляється оптимальна архітектура гри з урахуванням функціоналу графічного процесора RTX, включаючи використання високорівневих графічних ефектів та обробку реального часу. Проводяться експерименти з метою визначення ефективності та якості гри з використанням даної методики, що дозволяє зробити висновки про переваги та можливі обмеження такого підходу у розробці ігрових застосунків.

КЛЮЧОВІ СЛОВА: ГРАФІЧНИЙ ПРОЦЕСОР, RTX, UNITY3D, POST PROCESSING, C#, АРХІТЕКТУРА, DLSS, GPU.

ЗМІСТ

ВСТУП

Графічні процесори (GPU) є ключовим елементом у сучасних ігрових технологіях, забезпечуючи високу якість графіки та реалістичні візуальні ефекти. Особливу увагу заслуговують графічні процесори на базі технології RTX від NVIDIA, які володіють передовими можливостями, такими як трасування променів (RTX), що значно підвищує реалізм зображення та дозволяє створювати захоплюючі графічні ефекти.

Мета цієї дипломної роботи полягає в розробці методики для ефективного використання графічного процесора на базі RTX у процесі розробки ігрового застосунку на платформі Unity3D. Основні завдання дослідження включають:

- вивчення технічних можливостей графічного процесора RTX;
- проектування оптимальної архітектури гри з використанням функціоналу RTX;
- дослідження технічних аспектів інтеграції та оптимізації продуктивності застосунку з використанням цієї технології.

Дослідження даної теми має важливе значення для подальшого розвитку ігрової індустрії та впровадження передових технологій у процесі розробки ігор. Враховуючи стрімкий розвиток графічних можливостей та потребу у реалістичних ігрових ефектах, використання графічного процесора на базі RTX стає актуальним і перспективним напрямком у сфері розробки ігрових застосунків.

Архітектура графічного процесора RTX від NVIDIA базується на використанні спеціалізованих обчислювальних ядер для трасування променів (RT cores) та штучного інтелекту для покращення графічних ефектів (Tensor cores). RT cores відповідають за обробку променів у реальному часі, що дозволяє досягти реалістичного освітлення, тіней та відблисків у графіці. За допомогою Tensor cores, можливе застосування технік штучного інтелекту, наприклад, для покращення деталізації текстур та виявлення об'єктів у грі.

Крім того, архітектура RTX підтримує передові графічні API, такі як DirectX 12 Ultimate та Vulkan, що дозволяє максимально використовувати потенціал

графічного процесора для створення захоплюючих ігрових вражень.

Приклади ігор, які використовують технології RTX, включають "Cyberpunk 2077", "Shadow of the Tomb Raider" та "Minecraft" з підтримкою RTX, що демонструє великий потенціал цієї технології у створенні реалістичної та захоплюючої графіки.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ З АРХІТЕКТУРИ ГРАФІЧНИХ ПРОЦЕСОРІВ ТА МЕТОДИК РОЗРОБКИ ІГРОВИХ ЗАСТОУВАНЬ

1.1 Графічні процесори та їх роль у розробці ігрових застосунків

Роль графічних процесорів (GPU) у сфері ігор та розробці візуально збагачених застосунків є незаперечною. Особливо важливою стала їхній внесок у контексті технологій RTX від NVIDIA. Технологія RTX принесла значні зміни у сфері візуальних ефектів та графічного реалізму.

По-перше, варто звернутися до переваг GPU у забезпеченні високої продуктивності для графічних обчислень. GPU мають велику кількість ядер, які спрямовані саме на обробку графічних даних. Це дозволяє їм розраховувати освітлення, тіні, текстури та інші візуальні ефекти у реальному часі з вражаючою швидкістю. Такі ігри, як "Control" від Remedy Entertainment, демонструють високий рівень графічного реалізму завдяки роботі GPU із складними графічними ефектами.

Другий аспект полягає у спеціалізованій архітектурі GPU. Вони оптимізовані для ефективної обробки графічних даних, мають швидкий доступ до пам'яті та спеціалізовані обчислювальні блоки. Це дозволяє їм зосередитися на важливих аспектах графічного обчислення, таких як трасування променів, обчислення фізики світла та інші складні операції.

Третій аспект – використання графічних API для взаємодії з GPU. DirectX, OpenGL та інші API надають розробникам доступ до потужності графічних процесорів та дозволяють максимально використовувати їх можливості для створення реалістичних ігрових вражень.

Однією з найважливіших інновацій у цій галузі стала технологія RTX від NVIDIA. Вона впроваджує трасування променів у реальному часі, що робить графіку ще більш реалістичною та деталізованою. Завдяки RTX, ігрові сцени набувають нового рівня глибини та реалізму, змушуючи гравців відчувати себе у

цілком іншому, більш іммерсивному світі.

Крім цього, GPU, які підтримують технологію RTX, мають спеціальні обчислювальні ядра (RT cores) та блоки штучного інтелекту (Tensor cores). RT cores відповідають за трасування променів у реальному часі, що створює реалістичні ефекти світла та тіней. Tensor cores використовують нейронні мережі для підвищення якості графіки та покращення швидкості обчислень.

У підсумку, графічні процесори у контексті технологій RTX відіграють вирішальну роль у створенні реалістичних та захоплюючих ігрових вражень. Їхні можливості забезпечують високу якість графіки, ефективність обчислень та іммерсивний геймплей, що робить ігровий досвід надзвичайно цікавим та захоплюючим для гравців.

Приклади використання прискорення графічних обчислень:

- обчислення освітлення: графічні процесори використовуються для обчислення різних видів освітлення, таких як точкове освітлення, рефлекторне освітлення та відображення тіней. Це дозволяє створювати реалістичні візуальні ефекти у грі;

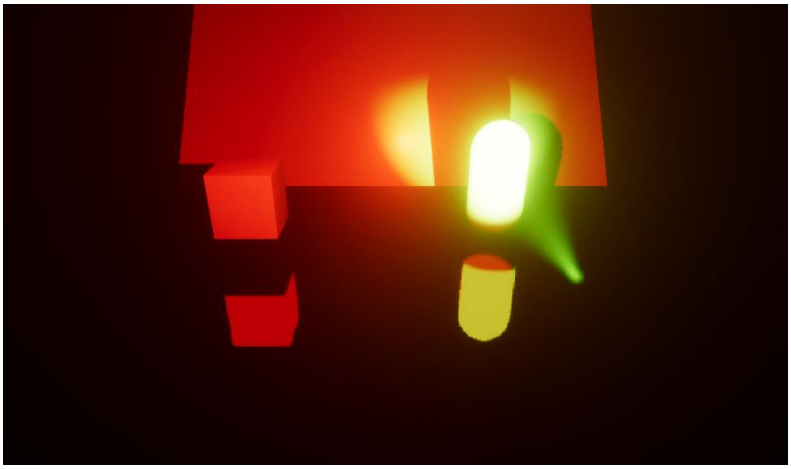

Рисунок 1.2 – Обчислення різних видів освітлення (Point Light, Spot Light)

- текстури та матеріали: графічні процесори обробляють текстури та матеріали, що використовуються для накладання зображень на поверхні об'єктів у грі. Це додає деталізацію та реалістичність графіці;

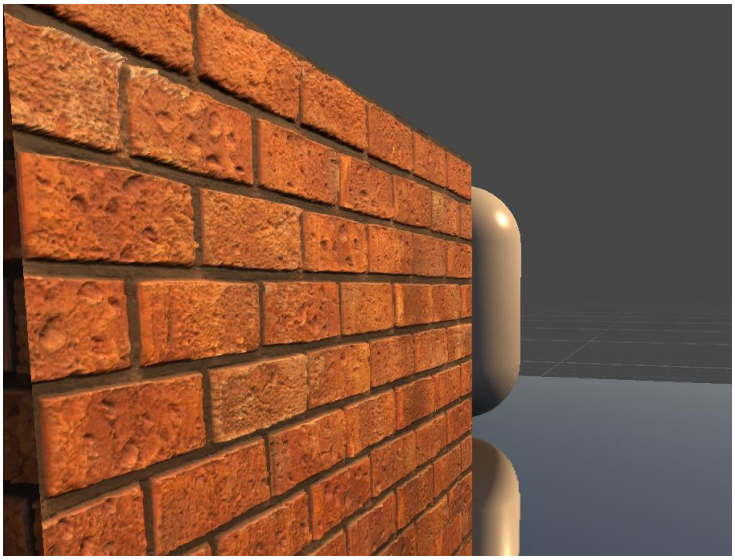

Рисунок 1.1 – Обчислення матеріалу з Height Map і Normal Map

- динамічні ефекти: сучасні графічні процесори підтримують різноманітні динамічні ефекти, такі як водяні ефекти, вибухи, дим та інші, що додають реалістичності та динаміки ігровому світу;
- променеве відстеження: технології променевого відстеження, які широко використовуються у сучасних графічних процесорах, дозволяють моделювати реальне поширення світла та тіней, що підвищує реалізм зображення у грі.

Технологія променевого відстеження (RTX) у графічних обчисленнях – значний крок вперед, оскільки вона точно передає освітлення та тіні, роблячи ігрове середовище ще реалістичнішим. Графічні процесори RTX мають RT ядра, які обробляють промені в реальному часі. У грі "Metro Exodus" від 4A Games, RTX використовується для реалістичного освітлення та відображення води, створюючи дивовижні графічні ефекти.

Технологія RTX дозволяє моделювати взаємодію світла з поверхнями та об'єктами, створюючи реалістичні тіні та відбиття. Це робить ігровий світ більш живим та привабливим для гравців, поглиблюючи їхню іммерсію в гру.

Основна перевага технології RTX полягає у здатності точно відтворювати складні візуальні ефекти, такі як глобальне освітлення, підтримка взаємодії світла з поверхнями та матеріалами, а також створення відбиттів та тіней, які надають грі реалістичний вигляд і відчуття присутності в ігровому світі.

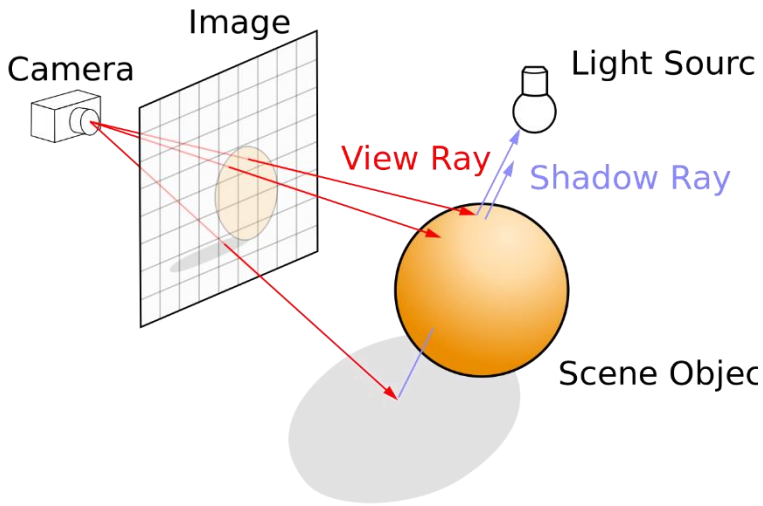

Рисунок 1.3 – Алгоритм трасування променів

Технології штучного інтелекту (AI) в графічних процесорах відкривають нові можливості для покращення якості графіки в ігрових застосунках. Одним із важливих аспектів їх використання є автоматизація деяких процесів, що дозволяє розробникам створювати більш реалістичні та деталізовані ігрові світи. Наприклад, у грі "Red Dead Redemption 2" від Rockstar Games, технологія штучного інтелекту в графічних процесорах RTX використовується для автоматичного підвищення деталізації ландшафту, текстур та динамічних об'єктів у великому відкритому світі гри. Це дозволяє створювати більш живі та реалістичні ігрові сцени, які поглиблюють іммерсію гравців.

Однією з інноваційних технологій у графічних процесорах є DLSS (Deep Learning Super Sampling). Ця технологія використовує штучний інтелект для масштабування графіки з метою підвищення якості зображення та зниження навантаження на графічний процесор. DLSS дозволяє покращити кадрову частоту та різкість зображення в іграх, що використовують цю технологію. Наприклад, у грі "Cyberpunk 2077", яка використовує технологію RTX, DLSS використовується для підвищення кадрової частоти та зменшення артефактів у графіці, що робить ігровий досвід більш плавним та реалістичним.

Технології штучного інтелекту в графічних процесорах відкривають нові можливості для покращення ігрових вражень та забезпечують більшу

реалістичність іммерсії для гравців. Вони дозволяють автоматизувати деякі процеси створення графіки, що забезпечує більшу ефективність у розробці ігор та використанні графічних ресурсів.

1.2 Архітектура графічного процесора на базі RTX

Архітектура графічних процесорів (GPU), зокрема технологія RTX від NVIDIA, має особливості, включаючи ядра RT (Ray Tracing cores). Ці ядра спрямовані на обробку трасування променів (ray tracing), що дозволяє створювати реалістичні візуальні ефекти, наприклад, реалістичне освітлення, відбиття світла та тіні.

Їхні алгоритми включають алгоритм відстеження променів, такі як Ray-Triangle і Ray-Box, для обробки великої кількості променів та взаємодії з об'єктами в грі.Розглянемо принцип роботи алгоритму Ray-Triangle. Таким чином, архітектура RTX дозволяє досягти значного підвищення рівня реалізму та деталізації в графічних додатках порівняно зі звичайними GPU.

Промінь можна представити як вектор початку O та напрямку D:

$$R(t) = O + t * D,$$

де t - параметр, який визначає відстань вздовж променя.

Для перевірки перетину променя з трикутником, використовується формула Мёллера-Трумбора. Ця формула визначається через вектори сторін трикутника e1, e2, вектор променя T, вектор першої сторони трикутника P, і параметр перетину u, v. Формула перетину:

$$\mathbf{T} = \mathbf{O} - \mathbf{V}_0$$
$$\mathbf{P} = \mathbf{D} \times \mathbf{e}_2$$
$$\mathbf{Q} = \mathbf{T} \times \mathbf{e}_1$$
$$t = \frac{\mathbf{e}_2 \cdot \mathbf{Q}}{\mathbf{P} \cdot \mathbf{e}_1}$$
$$u = \frac{\mathbf{T} \cdot \mathbf{P}}{\mathbf{P} \cdot \mathbf{e}_1}$$
$$v = \frac{\mathbf{D} \cdot \mathbf{Q}}{\mathbf{P} \cdot \mathbf{e}_1}$$

Під час рендеру сцени, у кожного променя перевіряється його перетин з об'єктами на сцені. У випадку трикутника, визначається, чи перетинає промінь його площину, і якщо так, то визначається точка перетину та параметр t.

Рисунок 1.4 – Архітектура Turing TU102 Full GPU

Ядра Tensor - це спеціалізовані процесори, які прискорюють обчислення штучного інтелекту (AI) у графічних процесорах (GPU). Їх робота ґрунтується на нейронних мережах, які значно покращують якість графіки завдяки:

- деталізації текстур: ядра Tensor роблять текстури більш чіткими та реалістичними;

- антиаліасингу: згладжують нерівності на краях об'єктів, роблячи зображення більш плавним;

- іншим візуальним ефектам: застосовують різні алгоритми для покращення реалістичності та динамічності зображення.

Технологія DLSS (Deep Learning Super Sampling) використовує ядра Tensor

для:

- підвищення роздільної здатності зображення: робить картинку більш чіткою без втрати продуктивності;

- покращення продуктивності GPU: економить час та ресурси при обробці графіки.

Основні алгоритми, які використовують ядра Tensor:

- пряме розповсюдження (Forward Propagation): передає дані через нейронну мережу, обчислюючи вихідні значення нейронів та застосовуючи функції активації;

- зворотне розповсюдження (Backward Propagation): коригує ваги та зсуви нейронів під час тренування мережі, мінімізуючи функцію втрати;

- згорткові нейронні мережі (Convolutional Neural Networks): обробляють зображення, застосовуючи фільтри для виявлення особливостей та зменшуючи розмірність даних;

- перенавчання (Transfer Learning): використовує попередньо навчені нейронні мережі для нових завдань, економлячи час та ресурси розробників. Ядра Tensor в GPU на базі RTX забезпечують швидке виконання цих алгоритмів, роблячи їх ідеальними для задач, пов'язаних з штучним інтелектом та графікою.

Рисунок 1.5 – Архітектура Turing TU102/TU104/TU106

Алгоритми обробки даних та генерації вихідних результатів використовують математичні операції, такі як множення матриць, додавання та застосування функцій активації. Ядра Tensor реалізують ці операції у паралельному режимі для ефективної обробки великих обсягів даних у реальному часі.

Ядра CUDA (Compute Unified Device Architecture cores) є ключовими обчислювальними одиницями в графічних процесорах. Вони відповідають за обробку графічних ефектів, таких як освітлення, текстурні ефекти, анімація та фізика відображення. Ядра CUDA забезпечують втілення різноманітних графічних ефектів, включаючи динамічні тіні, розсіяння світла та фізичні властивості об'єктів у сцені. Давайте розглянемо їх роботу та алгоритми виконання.

Ядра CUDA використовуються для розпаралелювання обчислень, що дозволяє виконувати багато операцій одночасно. Це досягається завдяки великій кількості ядер CUDA у GPU, які працюють паралельно над різними частинами даних. Під час обчислень на ядрах CUDA використовується алгоритм розпаралелювання, який розподіляє завдання між доступними ядрами. Цей підхід

дозволяє ефективно використовувати потужності GPU та забезпечує швидке виконання обчислень.

Ядра CUDA виконують операції над даними, які передаються з центрального процесора на GPU. Ці операції можуть бути арифметичними, логічними, матричними обчисленнями, обробкою зображень та іншими. Кожне ядро CUDA працює над своєю частиною даних, що дозволяє швидко виконувати операції над великими обсягами інформації.

Ядра CUDA оптимізовані для виконання обчислень швидше та ефективніше порівняно з центральним процесором (CPU). Вони використовують спільну пам'ять, кешування даних та інші методи для забезпечення високої продуктивності обчислень.

Для програмування на ядрах CUDA використовується мова програмування CUDA C/C++, яка надає доступ до функціональності ядер CUDA та дозволяє виконувати паралельні обчислення на GPU.

Графічний процесор RTX складається з різних компонентів, таких як підсистема пам'яті (VRAM), контролери пам'яті, шина зв'язку з іншими компонентами системи, що дозволяють ефективно обмінюватися даними та керувати ресурсами графічного процесора.

Ця детальна архітектура графічного процесора на базі RTX демонструє комплексність та високий рівень функціональності цих пристроїв у галузі обробки графіки та відображення візуальних ефектів у графічних застосунках.

1.3 Огляд існуючих методик розробки ігрових застосунків з використанням графічних процесорів

Аналіз підходів до розробки ігрових застосунків з використанням графічних процесорів охоплює різноманітні методики та практики, що застосовуються в галузі розробки ігор для максимально ефективного використання потужності графічних процесорів (GPU). Розглянемо основні напрямки роботи та їхні особливості:

а) прямий доступ до GPU (Direct Access to GPU): Цей підхід передбачає роботу з API графічного процесора, такими як DirectX або OpenGL, що дозволяє розробникам отримати прямий доступ до функціональності GPU для впровадження складних графічних ефектів та обчислень;

б) використання шейдерів (Shader Programming): Шейдери виступають важливим інструментом у розробці графічних ефектів у іграх, дозволяючи програмувати різні аспекти відображення об'єктів, включаючи освітлення, тіні, текстури та спеціальні ефекти;

в) використання бібліотек і фреймворків: У розробці ігор широко використовуються готові бібліотеки та фреймворки, такі як Unity3D та Unreal Engine, які спрощують роботу з графікою та дозволяють ефективно впроваджувати графічні ефекти та оптимізувати використання GPU;

г) оптимізація продуктивності: Важливим аспектом розробки ігор є оптимізація продуктивності для роботи з GPU, яка включає в себе ефективне використання шейдерів, уникання зайвих обчислень та оптимізацію використання ресурсів GPU;

ґ) використання новітніх технологій: Розробники ігор постійно вдосконалюють свої підходи до роботи з GPU за допомогою новітніх технологій, таких як RTX (Ray Tracing), DLSS (Deep Learning Super Sampling) та інші, для створення більш реалістичних та іммерсивних графічних ефектів.

Огляд методик розробки ігрових застосунків з використанням графічних процесорів включає різноманітні підходи та практики, що дозволяють розробникам досягати високої якості графіки та продуктивності у своїх іграх.

РОЗДІЛ 2. МЕТОДОЛОГІЯ РОЗРОБКИ З УРАХУВАННЯМ ВИКОРИСТАННЯ ПЛАТФОРМИ Unity3D

2.1 Вибір платформи для розробки (Unity3D)

Рішення щодо вибору платформи для розробки ігрового застосунку має велике значення і впливає на різні аспекти процесу розробки, продуктивність, можливості та популярність гри серед користувачів. Unity3D є однією з переважних платформ для розробки ігор, особливо для менших і середніх проектів. Давайте розглянемо ключові переваги та особливості використання Unity3D для створення ігрових застосунків.

По-перше, це кросплатформенність. Unity3D надає можливість розробляти ігри для різних платформ, включаючи ПК, мобільні пристрої (iOS, Android), консолі (PlayStation, Xbox), веб-браузери та інші. Це робить платформу привабливим вибором для розробників, які прагнуть залучити широку аудиторію користувачів.

По-друге, Unity3D має широкий функціонал. Вона включає в себе велику кількість інструментів і функцій для розробки графіки, фізики, штучного інтелекту, звуку та іншого контенту для ігор. Вбудовані редактори дозволяють створювати сцени, анімації, шейдери та інтерфейси, що спрощує процес розробки.

Не менш важлива є наявність великої спільноти розробників та підтримки. Unity3D має активну спільноту, яка готова надавати допомогу та поради щодо вирішення проблем. Платформа також має докладну та оновлювану документацію, яка полегшує освоєння інструментів та функціоналу.

Unity3D також славиться швидкістю та зручністю розробки завдяки вбудованому редактору та набору готових компонентів та скриптів. Це дозволяє розробникам швидко прототипувати, тестувати та вдосконалювати свої ігри.

Іншою важливою особливістю Unity3D є підтримка віртуальної та доповненої реальності (VR та AR). Це робить платформу ідеальним вибором для

розробки ігор для різних типів гарнітур VR, AR-додатків та інтерактивних досліджень.

Узагальнюючи, Unity3D приваблива завдяки своїй простоті в освоєнні, широкому функціоналу, кросплатформенності та активній спільноті розробників. Ці фактори роблять її популярним вибором для багатьох розробників ігор, незалежно від їхнього досвіду та обсягу проекту.

Широкий функціонал Unity3D охоплює багато аспектів розробки ігор, включаючи роботу з графікою, анімацією, фізикою, штучним інтелектом та іншими ключовими аспектами. Розглянемо деякі з них.

Unity3D має вбудований редактор для створення та редагування сцен, який включає в себе можливості розміщення об'єктів, додавання світла, налаштування камери та інших параметрів.

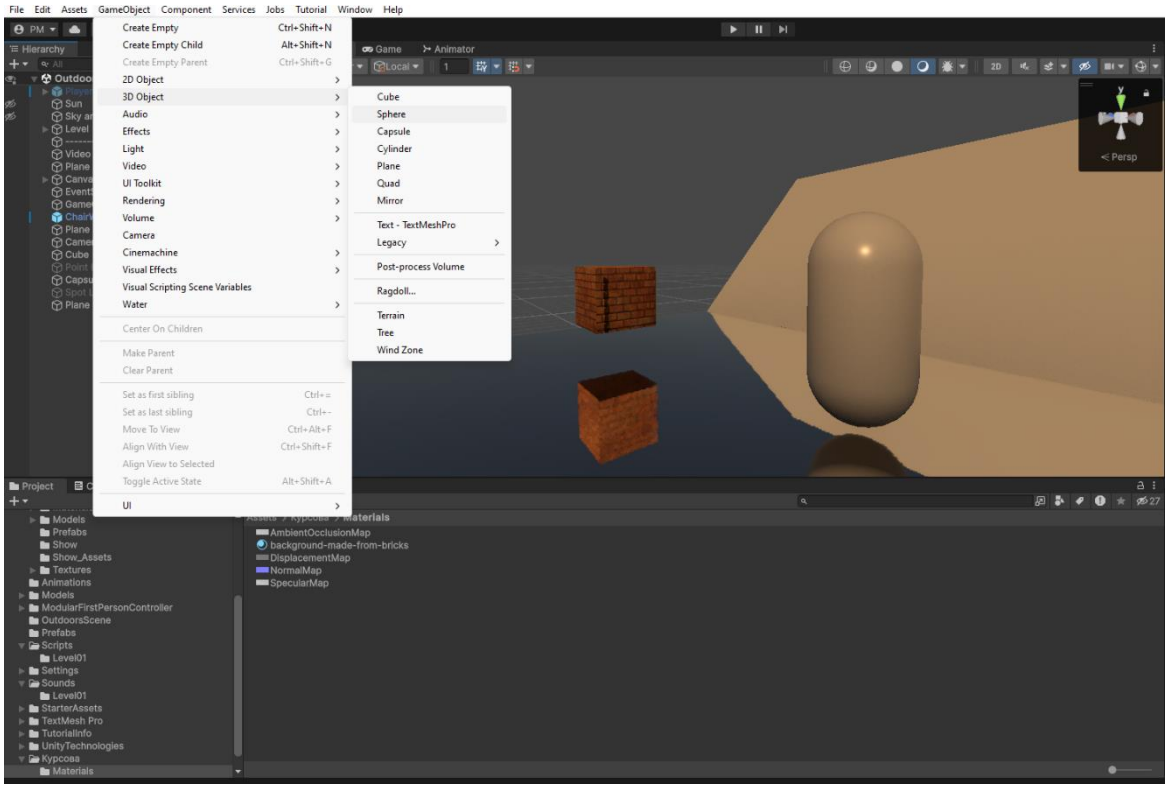

Рисунок 2.1 – Приклад вигляду інтерфейсу та робочого простору

Unity3D дозволяє створювати анімації для об'єктів у грі. Це можуть бути анімовані персонажі, об'єкти руху, ефекти та інше. Для цього використовуються анімаційні кліпи та система анімацій.

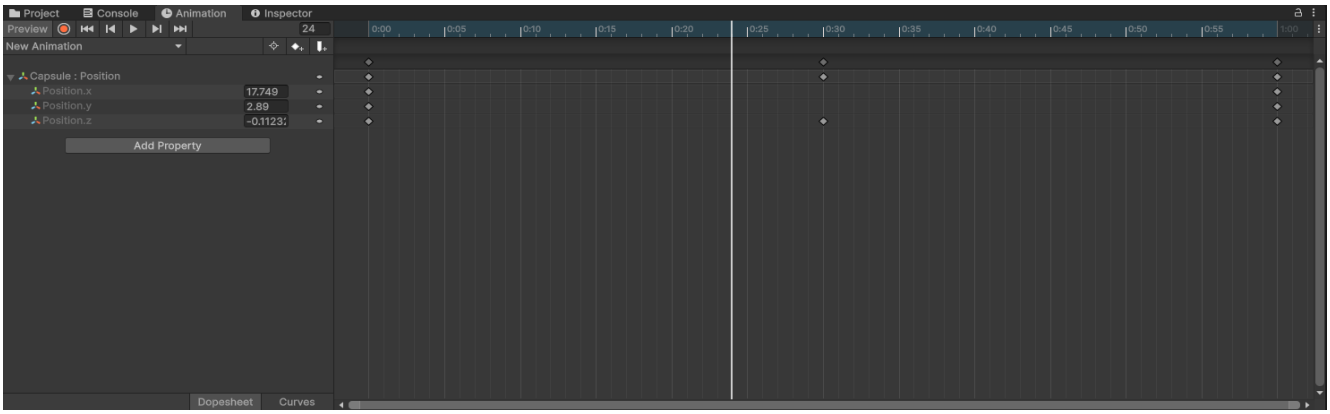

Рисунок 2.2 – Приклад вигляду інтерфейсу роботи з анімаціями

Unity3D підтримує роботу з шейдерами для створення реалістичних графічних ефектів. Це можуть бути шейдери для текстур, освітлення, води, підводного світла та інші.

```
// Upgrade NOTE: upgraded instancing buffer 'Props' to new syntax.

Shader "Custom/Dissolve" {
    Properties {
        _Color ("Color", Color) = (1,1,1,1)
        [HDR]_Emission ("Emission", Color) = (0,0,0,0)
        _MainTex ("Albedo", 2D) = "white" {}
        _Normal ("Normal", 2D) = "bump" {}
        _MetallicSmooth ("Metallic (RGB) Smooth (A)", 2D) = "white" {}
        _AO ("AO", 2D) = "white" {}
        [HDR]_EdgeColor1 ("Edge Color", Color) = (1,1,1,1)
        _Noise ("Noise", 2D) = "white" {}
        [Toggle] _Use_Gradient ("Use Gradient?", Float) = 1
        _Gradient ("Gradient", 2D) = "white" {}
        _Glossiness ("Smoothness", Range(0,1)) = 0.5
        _Metallic ("Metallic", Range(0,1)) = 0.0
        [PerRendererData]_Cutoff ("Cutoff", Range(0,1)) = 0.0
        _EdgeSize ("EdgeSize", Range(0,1)) = 0.2
        _NoiseStrength ("Noise Strength", Range(0,1)) = 0.4
        _DisplaceAmount ("Displace Amount", Float) = 1.5
        _cutoff ("cutoff", Range(0,1)) = 0.0


    }
    SubShader {
        Tags { "Queue"="AlphaTest" "RenderType"="TransparentCutout" "IgnoreProjector"="True" }
        Cull Off

        LOD 200

        CGPROGRAM

        // Physically based Standard lighting model, and enable shadows on all light types
        #pragma surface surf Standard fullforwardshadows vertex:vert addshadow

        // Use shader model 3.0 target, to get nicer looking lighting
        #pragma target 3.0
```

Рисунок 2.3 – Dissolve shader який використовується у матеріалах

Unity3D має вбудовану систему фізики для симуляції реалістичних фізичних взаємодій між об'єктами у грі. Це можуть бути рух об'єктів, колізії, взаємодія з гравітацією та інші ефекти.

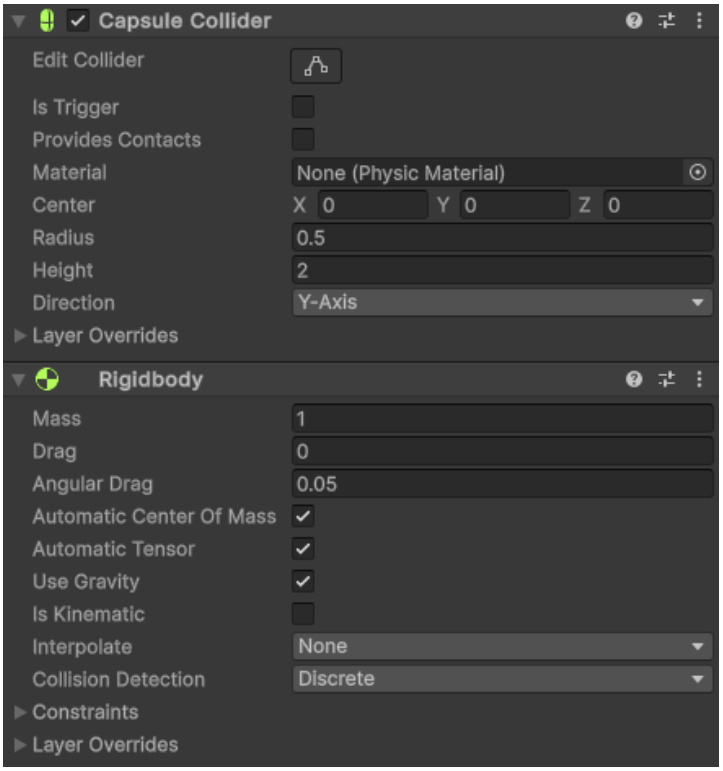

Рисунок 2.4 – Collider і Rigidbody – компоненти, які додають фізику об’єкту

Unity3D дозволяє додавати аудіо ефекти, музику та звуки до гри. Це можуть бути звукові ефекти руху, взаємодії з об'єктами, музичні супроводи та інше.

Рисунок 2.5 – Audio Source і Distorsion Filter – джерело звуку і ефект дістрошину

2.2 Проектування архітектури гри з урахуванням використання графічного процесора на базі RTX

При плануванні архітектури гри з використанням графічного процесора на базі технології RTX, ключовим є увага до реалістичних графічних ефектів, які RTX підтримує, таких як трасування променів (ray tracing), DLSS (Deep Learning Super Sampling) і інші. Проектування архітектури гри в такому контексті охоплює наступні аспекти:

- графічні ефекти - реалізація реалістичного освітлення з використанням RTX, включаючи динамічне освітлення, тіні, відбиття світла тощо. Ці ефекти додають глибину та реалізм до гри;

- моделювання середовища - використання променевого відстеження для створення складних середовищ, таких як вода, скляні поверхні, вогонь тощо. Це дозволяє створювати вражаючі графічні сцени;

- деталізація текстур - використання DLSS для поліпшення деталізації текстур та зображення в реальному часі, що забезпечує високу якість зображення

при високій продуктивності графічного процесора;

- штучний інтелект та фізика - реалізація складних моделей штучного інтелекту та фізики гри, які можуть взаємодіяти з графічним процесором для створення реалістичних сцен та ефектів.

У зв'язку з цим, проектування архітектури гри "Dota 2" у спрощеному вигляді включає такі компоненти:

- Ігрове поле (Game World) - це велике ігрове середовище з локаціями, об'єктами, перешкодами та іншими елементами;

- Персонажі (Characters) - гравці та NPC, які рухаються та взаємодіють у грі з урахуванням їхніх властивостей та навичок;

- Бойова система (Combat System) - система бою, яка включає механіки бою та розрахунок пошкоджень;

- Інтерфейс користувача (User Interface) - елементи інтерфейсу, які відображають інформацію про гру, статус персонажів, інвентар тощо.

Ці компоненти спрощеної версії гри "Dota 2" враховують можливості графічного процесора на базі RTX та дозволяють створювати реалістичні та захоплюючі графічні ефекти.Ці компоненти можуть бути організовані за допомогою паттернів проектування, таких як Model-View-Controller (MVC) або Entity-Component-System (ECS), що дозволить створити структуровану та ефективну архітектуру гри з урахуванням використання графічного процесора на базі RTX.

У нашому прикладі ми реалізуємо MVC паттерн проектування.

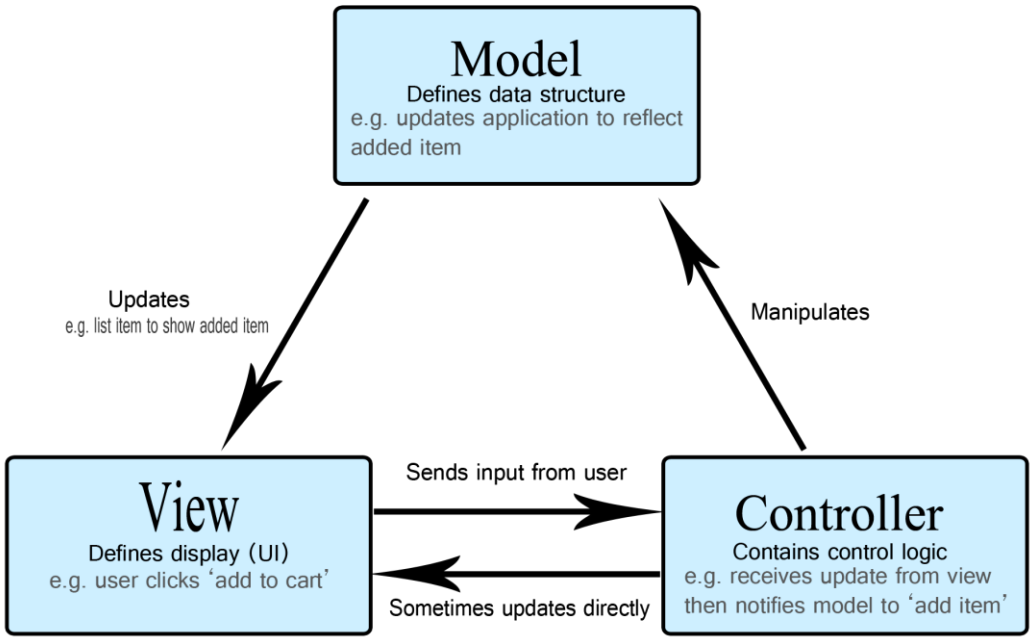

Рисунок 2.6 – Загальний вигляд MVC паттерна

2.3 Опис використаних технологій і інструментів розробки

Відповідно до паттерна, були створені відповідні класи EnemyModel, EnemyView, EnemyController, а також відповідні класи для гравця. Також реалізували повністю ігровий рівень, бойову логіку, рух NPC, розрахунок пошкоджень, простий тестовий інтерфейс.

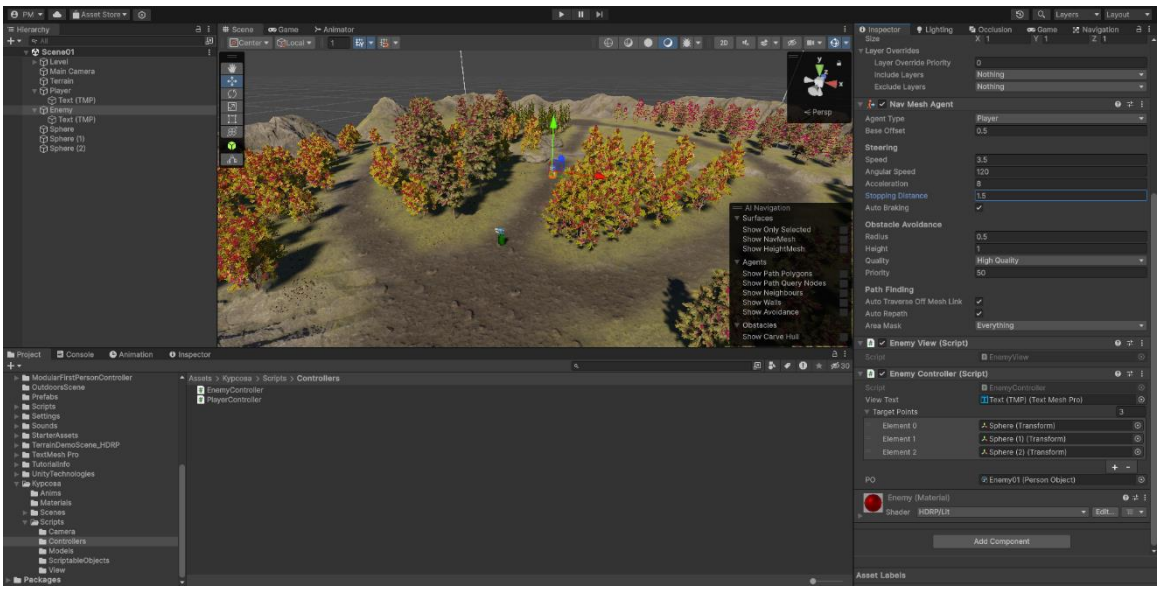

Рисунок 2.7 – Реалізація проєкту в робочому просторі Unity3D

Отже, наша реалізація логіки гри для персонажа та противника з урахуванням паттерну MVC виглядає наступним чином:

- у класі PlayerModel ми зберігаємо дані про персонажа: здоров'я, швидкість, таймаут атаки, броню, урон і т. д. У класі EnemyModel ми зберігаємо дані про противника: здоров'я, скорість, таймаут атаки, броню, урон і т. д. Обидва класи реалізують логіку для зміни своїх параметрів та сповіщення про зміни за допомогою подій або іншого механізму сповіщення. Приклад коду EnemyModel:

```
public class EnemyModel
{
    public int CurrentHealth { get; set; }
    public int Health { get; set; }
    public float Speed { get; set; }
    public float AttackTimeout { get; set; }
    public float Armor { get; set; }
    public int Damage { get; set; }

    public EnemyModel(PersonObject person)
    {
        Health = person.Health;
        Speed = person.Speed;
        AttackTimeout = person.AttackTimeout;
        Armor = person.Armor;
        Damage = person.Damage;

        CurrentHealth = Health;
```

```
    }
};
```

- у класі PlayerController ми відповідаємо за керування персонажем, його рух та взаємодію зі світом гри. У класі EnemyController простий штучний інтелект відповідає за керування противником, його атаки, рух та взаємодію з іншими об'єктами. Уривок коду класа EnemyController:

```
public class EnemyController : MonoBehaviour
{
    public TMP_Text ViewText;

    [SerializeField] private Transform[] targetPoints;
    [SerializeField] private PersonObject PO;

    public EnemyModel Model;
    private NavMeshAgent agent;
    private int currentIndex = 0;
    private GameObject player;
    private float distance;
    private bool canDamage = true;

    private void Awake()
    {
        Model = new EnemyModel(PO);
    }

    private void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        player = GameObject.FindGameObjectWithTag("Player");
        agent.SetDestination(targetPoints[0].position);

        agent.speed = Model.Speed;
    }

    private void Update()
    {
        distance = Vector3.Distance(gameObject.transform.position,
player.transform.position);

        if (distance <= 5)
        {
            if (distance <= 3 && canDamage)
            {

player.GetComponent<PlayerController>().TakeDamage(Model.Damage);
                canDamage = false;
                Invoke("ChangeFlag", Model.AttackTimeout);
            }
```

```
            else
            {
                FollowPlayer();
            }
        }
        else
        {
            Movement();
        }
    };
```

- у класі CharacterView ми відповідаємо за візуальне відображення інформації про персонажа та інші аспекти пов'язані з його представленням. У класі EnemyView ми відповідаємо за візуальне відображення інформації про противника та інші аспекти пов'язані з його представленням. Уривок коду класа EnemyView:

```
 public class EnemyView : MonoBehaviour
{
    private EnemyModel model;
    private EnemyController controller;

    private void Start()
    {
        controller = GetComponent<EnemyController>();
        model = controller.Model;

        controller.ViewText.text = $"Health: {model.CurrentHealth}
\nDamage: {model.Damage}";
    }

    private void Update()
    {
        controller.ViewText.text = $"Health: {model.CurrentHealth}
\nDamage: {model.Damage}";
    }
}.
```

Контролери у грі взаємодіють як між собою, так і з іншими скриптами, обробляючи події, які стосуються зіткнень персонажів, запуску атак, а також реакції на пошкодження та інші події. Вони відповідають за візуальне відображення персонажів і противників на екрані, оновлюючи зображення відповідно до змін у грі.

Цей підхід дозволяє створити структуровану та ефективну архітектуру гри з використанням патерну MVC, де кожен компонент має свою функціональність і легко взаємодіє з іншими компонентами.

Наша реалізація графічної частини гри включає основні графічні ефекти обробки зображень, які використовують трасування променів (ray tracing) та технологію DLSS (Deep Learning Super Sampling).

Трасування променів - це метод візуалізації, який створює зображення, трасуючи шлях світла та імітуючи ефект зіткнення з об’єктом у вигляді пікселів на екрані. Цей метод дозволяє досягти вищого рівня реалістичності порівняно з традиційними методами, проте він потребує значних обчислювальних ресурсів. Трасування променів ідеально підходить для ситуацій, коли час відтворення зображення можна ігнорувати, таких як статичні комп’ютерні зображення або візуальні ефекти в кіно та на телебаченні, але це також добре підходить для ігор, які працюють у реальному часі, де швидкість має велике значення.

Останнім часом апаратне прискорення для трасування променів стало стандартом для комерційних графічних карт, і графічні API докладають зусиль для підтримки цього підходу.

Розробникам рекомендується використовувати цей підхід для ігор та іншого програмного забезпечення, якщо це можливо.

Рисунок 2.8 - Використання трасування променів для зеркального відображення на гладких поверхнх

Ми вдосконалюємо графічні ефекти у грі за допомогою трасування променів для

створення реалістичного освітлення. Це включає різноманітні елементи, такі як відбиття світла, розсіяння тіней, глобальне затемнення тощо, що додає глибину та живописність до графіки. Ми активно використовуємо трасування променів для глобального освітлення, щоб точно відтворювати взаємодію світла з навколишнім середовищем, включаючи відбиття та розсіяння світла на різних поверхнях. Такий підхід дозволяє досягати реалістичних ефектів тіней з урахуванням геометрії об'єктів та джерел світла у грі.

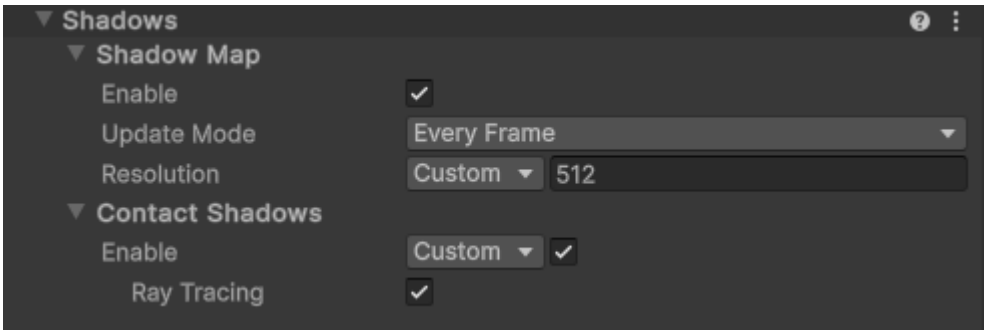

Рисунок 2.9 – Використання Ray Tracing в тінях

Також в даному проекті використовується система NavMesh. NavMesh (навігаційна сітка) - це графічне подання простору в ігровому світі, що використовується для керування рухом персонажів та інших об'єктів у грі.

Якщо розглядати принцип роботи, то NavMesh створюється на основі геометрії ігрового рівня (наприклад, полігонів, стін, перешкод). Це сітка навігації, де кожен вузол (трикутник) визначає доступне для переміщення просторове простір. Агенти (персонажі, NPC і т. д.) використовують NavMesh для розрахунку оптимального маршруту від точки А до точки Б, з урахуванням перешкод та інших факторів.

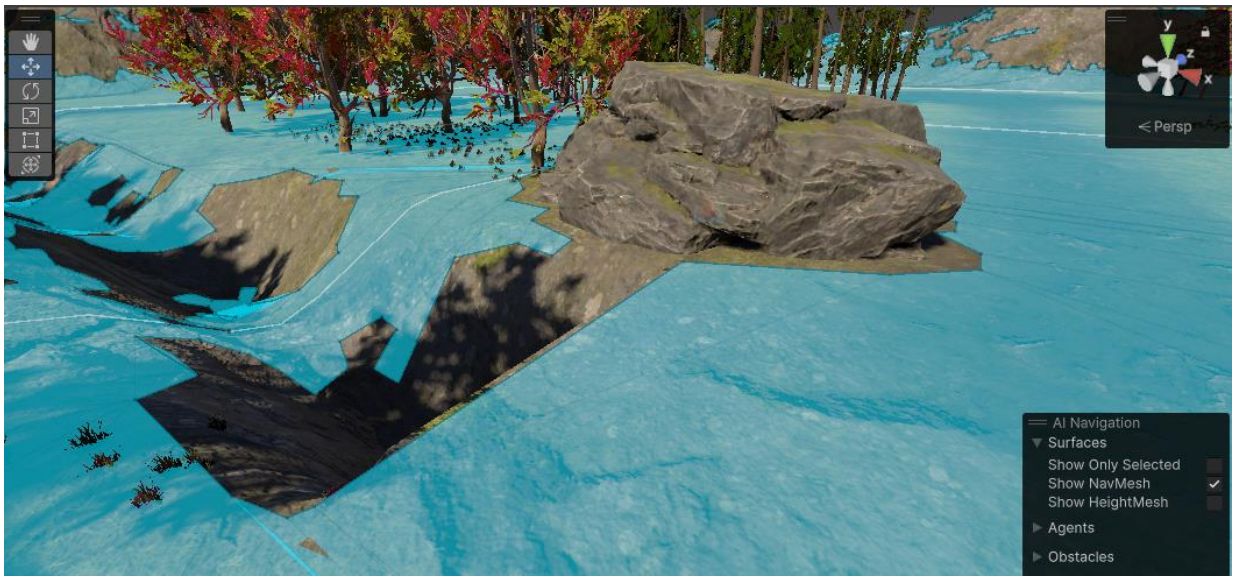

Рисунок 2.10 – Візуальне відображення роботи NavMesh

Серед прикладів використання, можна визначити основні методи роботи з NavMesh. Перше – це пошук шляху (Pathfinding), NavMesh використовується для пошуку оптимального шляху між двома точками в ігровому світі. Unity надає API для виконання пошуку шляху на NavMesh, наприклад, метод NavMesh.CalculatePath. Друге – це навігація агентів (Agent Navigation), NavMesh дозволяє агентам переміщуватися по ігровому світу, обходячи перешкоди. Це включає методи керування рухом, такі як NavMeshAgent.SetDestination, NavMeshAgent.Stop та інші. А також підтримується динамічне оновлення сітки, навігаційну сітку можна динамічно оновлювати під час гри, наприклад, при появі або зникненні перешкод. Unity надає методи для такого оновлення, такі як NavMeshBuilder.BuildNavMesh для перебудови NavMesh.

РОЗДІЛ 3. ТЕХНІЧНИЙ АСПЕКТ, ТА ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

3.1 Оптимізація продуктивності за допомогою графічного процесора на базі RTX

Оптимізація продуктивності за допомогою графічного процесора на базі RTX є важливим аспектом розробки графічних додатків, зокрема ігрових проектів. Низька продуктивність може призвести до погіршення геймплею, зниження кадрової частоти та інших проблем, які впливають на користувальницький досвід. Одним з методів оптимізації є ефективне використання апаратних можливостей графічного процесора. Це включає оптимізацію роботи з пам'яттю, управління потоками обчислень та оптимізацію виконання графічних операцій. Також важливим є оптимізація алгоритмів, що використовуються для обробки графіки. Вдосконалення алгоритмів рендерингу, управління освітленням та інші оптимізації можуть покращити продуктивність графічного процесора. Додатково, розробники можуть використовувати інструменти та бібліотеки для аналізу продуктивності та виявлення проблемних місць у коді або налаштуваннях графічного процесора. Загалом, оптимізація продуктивності за допомогою графічного процесора на базі RTX є складним, але важливим завданням, яке дозволяє забезпечити плавну та ефективну роботу графічних додатків на різних пристроях і забезпечити гарний користувальницький досвід.

В Unity3D є кілька методів оптимізації, які дозволяють покращити продуктивність графічного процесора на базі RTX та забезпечити більш плавну та ефективну роботу ігрового додатку.

Одним з найважливіших методів є оптимізація роботи з ресурсами. Це включає управління текстурами, моделями та іншими ресурсами гри. Важливо використовувати оптимальні розміри та формати текстур, щоб зменшити навантаження на графічний процесор. Також слід уникати зайвого копіювання або завантаження ресурсів у пам'ять.

Іншим методом є оптимізація роботи з об'єктами та анімацією. Це означає

використання оптимальних розмірів та складності моделей, уникання надмірного використання складних анімацій та ефектів, а також використання LOD (Level of Detail) для зменшення деталізації об'єктів на великих відстанях.

Для досягнення оптимальної продуктивності важливо також використовувати правильні налаштування освітлення та ефектів. Це включає використання простіших, але ефективних ефектів освітлення, уникання зайвого використання шейдерів та ефектів частинок, а також оптимізацію роботи з тінями та відображенням об'єктів.

Важливим аспектом є також тестування продуктивності на різних пристроях та умовах. Це дозволяє виявити проблемні місця у грі та вчасно їх виправити для покращення користувальницького досвіду.

Загалом, комбінація цих методів дозволяє досягти оптимальної продуктивності графічного процесора на базі RTX в Unity3D та забезпечити більш зручний та задовільний геймплей для гравців.

Рівень деталізації (LoD) у комп'ютерній графіці відноситься до процесу спрощення різних графічних аспектів тривимірних (3D) об'єктів, які розглядаються на відстані. Головною метою використання LoD у програмній реалізації є покращення швидкості рендерингу при великій кількості об'єктів у віртуальній сцені, особливо тих, які перебувають на великій відстані від глядача, де деталі не є критичними. Цей метод може бути реалізований як дискретний або постійний, в залежності від того, як рівень деталізації інтегрується в програму. Дискретний LoD замінює більш деталізовані 3D-моделі або текстури менш деталізованими, зазвичай вже готовими моделями, тоді як постійний LoD використовує алгоритми для зміни рівня деталізації моделі динамічно в залежності від ситуації.

Багато програм у сфері 3D-графіки, особливо відеоігри, використовують певний рівень деталізації. Якщо реалізація LoD виконана правильно, вона практично не помітна для глядача. Коли відстань між об'єктами у віртуальній сцені та точкою спостереження перевищує певне значення, рівень деталізації може бути знижений. Це зниження зазвичай призводить до зменшення кількості полігонів, які складають 3D-об'єкт, до меншої якості зображення текстури або до їх комбінації.

Сцени з використанням LoD візуалізуються швидше, ніж ті, де об'єкти завжди мають повну деталізацію.

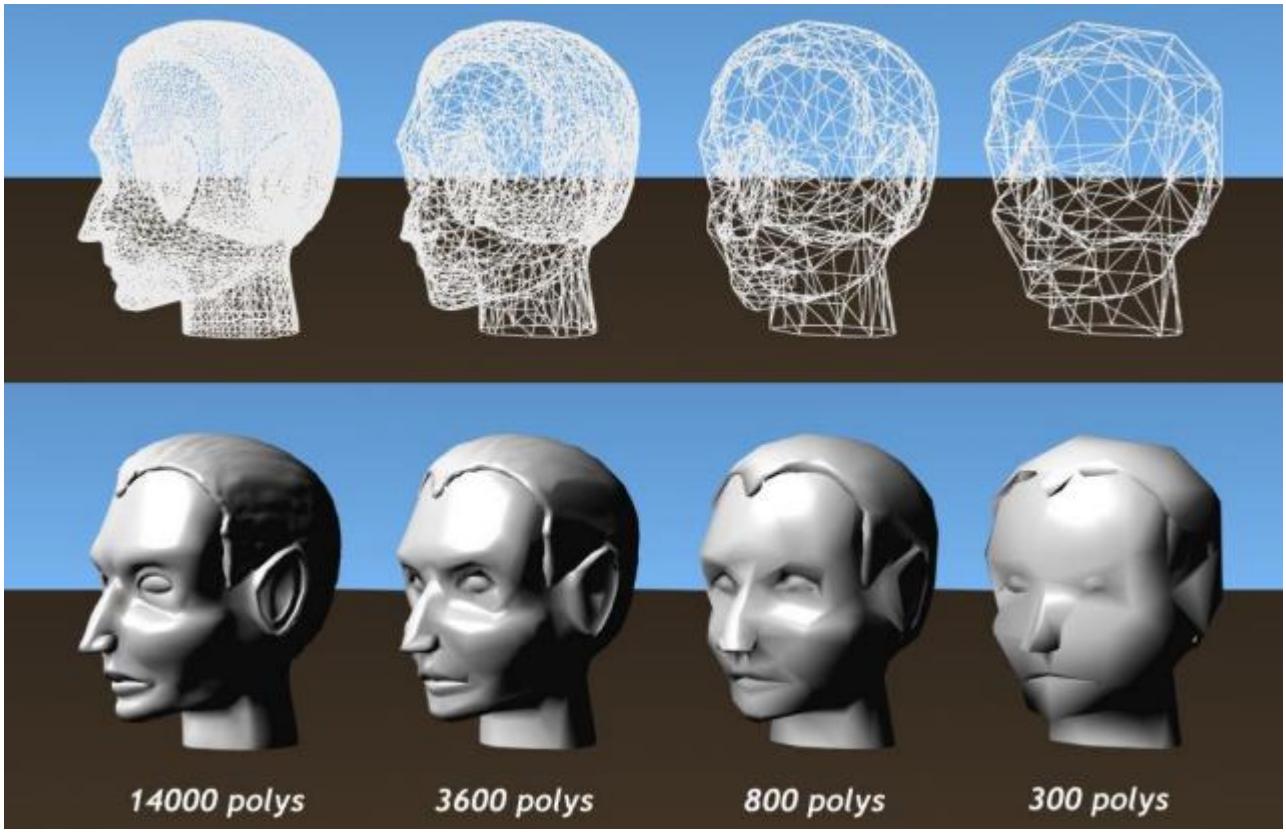

Рисунок 3.1 – Приклад роботи LoD

Дискретний рівень деталізації визначає, яку модель об'єкта використовувати для відображення залежно від заданих відстаней. Різні якості для 3D-моделей можуть бути створені програмами моделювання або обчислені під час роботи програми й збережені для майбутнього використання. Цей підхід ефективний, оскільки він простий і швидкий. Однак деякі програмісти та художники не люблять різкого переходу від однієї LoD моделі до іншої, оскільки це може бути очевидним, а об'єкти здаються зразу більшими й детальнішими.

Безперервний рівень деталізації використовує алгоритм, щоб розділити полігони об'єкта на деталі або об'єднати грані, щоб зменшити деталізацію. Цей варіант LoD забезпечує плавний перехід від одного пікселя до повної деталізації при зближенні. Проте цей алгоритм може значно навантажувати центральний процесор і призводити до непотрібних результатів, таких як відсутність граней у

многокутниках або зміни у 3D-моделі, що можуть спотворити початкову геометрію.

Інші налаштування рівня деталізації (LoD) включають зниження якості зображень за допомогою текстур, використання малих зображень низької якості замість них, а також застосування однотонних кольорів без текстур для апроксимації зовнішнього вигляду на великій відстані. Замість завантаження 3D-моделі низької якості для LoD, ми використовуємо геометричні примітиви, такі як сфери і прямокутники, які рендеруються значно швидше, ніж довільні полігони. Також існують інші алгоритми, які можна застосовувати для специфічних типів розрахунків LoD, наприклад, для модифікації ландшафтних сіток або для апроксимації швидко рухаючихся об'єктів у сцені.

Також існує такий метод оптимізації як Occlusion Culling. Occlusion Culling - це функція, яка автоматично вимикає відображення об'єктів, коли вони знаходяться за межами видимої зони камери через затемнення іншими об'єктами. У 3D-графіці це не відбувається за замовчуванням, оскільки зазвичай об'єкти, які знаходяться далеко від камери, рендеряться першими, а ті, що ближче, накладаються на них (це відомо як "перерисовування"). Occlusion Culling відрізняється від Frustum Culling, оскільки воно вимикає відображення об'єктів, які знаходяться поза полем зору камери, навіть якщо вони знаходяться під іншими об'єктами, які не затемнені. Процес Occlusion Culling полягає в тому, щоб пройти через сцену за допомогою віртуальної камери і створити ієрархію потенційно видимих наборів об'єктів, які використовуються для визначення того, що буде видимо для кожної камери. Це допомагає зменшити кількість об'єктів, які потрібно відобразити, що поліпшує продуктивність рендерингу.

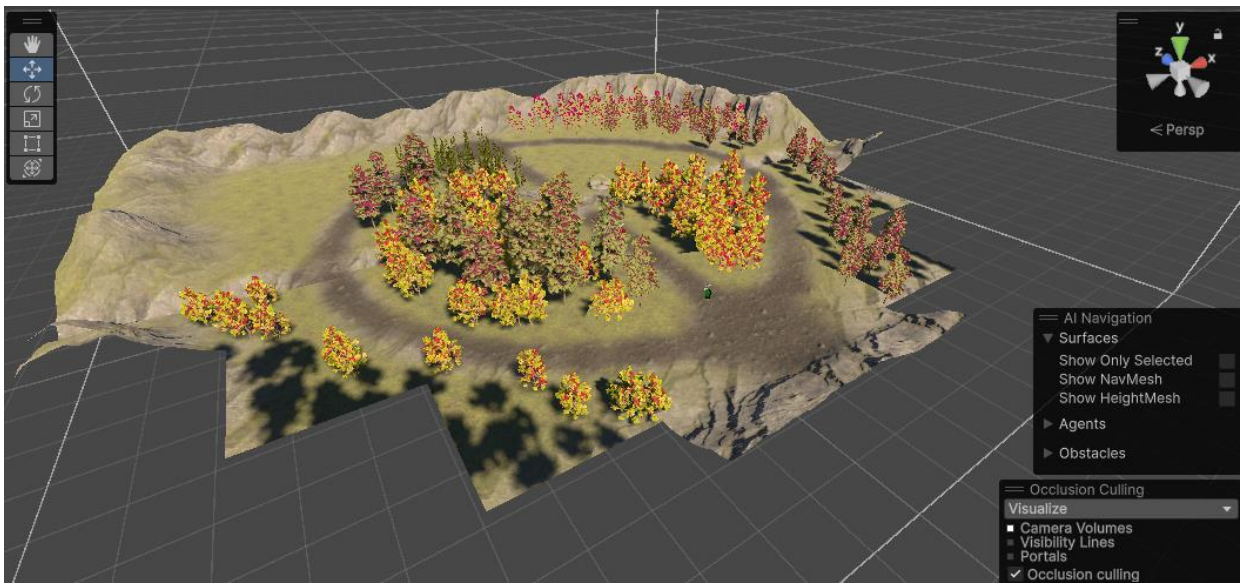

Рисунок 3.2 – Приклад роботи Occlusion Culling в Unity3D

В комп'ютерній графіці відсікання зворотньої сторони визначає, чи будуть малюватися полігони в графічних об'єктах. Цей етап графічного конвеєра перевіряє, як точки багатокутника будуть відображені при проектуванні на екран - за годинниковою стрілкою або проти неї. Якщо ви налаштуєте, що багатокутник, який "дивиться" вперед, має бути повернутий за годинниковою стрілкою, а той, який проектується на екран, - проти годинникової стрілки, то він не буде малюватися, а буде прихований від камери.

Цей процес допомагає зменшити кількість полігонів, які програма малює, що робить рендеринг об'єктів швидшим і ефективнішим. Наприклад, у сцені з міськими вулицями часто не потрібно малювати полігони з боку будівлі, яка знаходиться перед камерою, бо вони повністю приховані зверненою до камери стороною.

Загалом, припускають, що видалення задньої поверхні не спричинить видимих артефактів у відрендереній сцені, якщо вона складається з закритої і непрозорої геометрії. Але в сценах з прозорими полігонами, можливо, будуть видно полігони, які звернені назад через альфа-компонування. У випадку каркасного рендерингу використання відсікання зворотньої сторони може допомогти вирішити проблеми прихованих ліній, але це має значення тільки для

замкнутих опуклих форм.

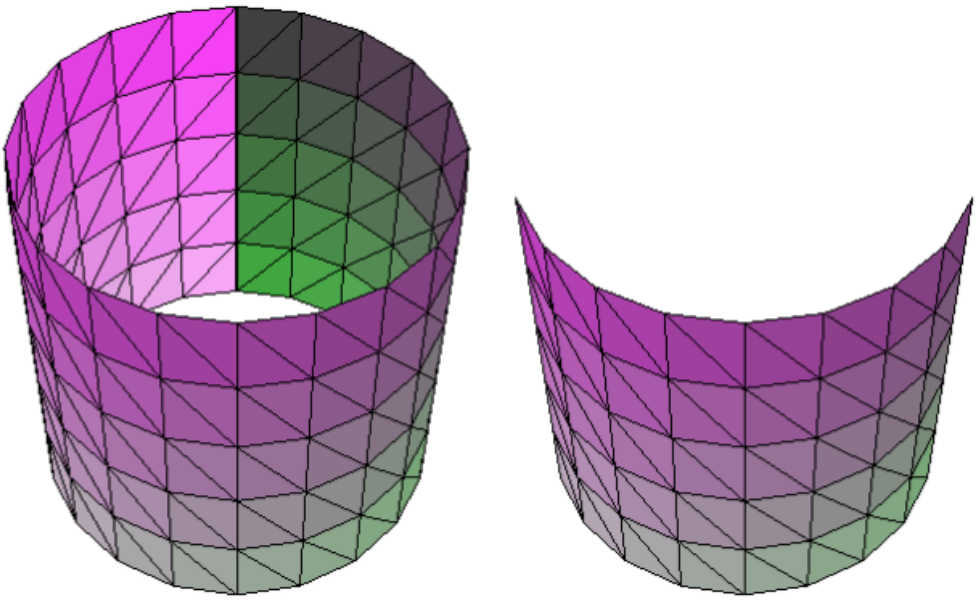

Рисунок 3.3 - Відсікання непотрібних полігонів для користувача

Додатково, ми використовуємо технологію Deep Learning Super Sampling (DLSS) для покращення якості графіки та збільшення продуктивності графічного процесора. Цей метод використовує нейронні мережі для інтелектуального підвищення роздільної здатності зображень, що дозволяє отримати більш реалістичні та чіткі зображення без зайвого навантаження обчислювальних ресурсів. У термінах Nvidia, DLSS означає супервибірку глибокого навчання (Deep Learning Super Sampling). Supersampling відноситься до методу згладжування, який спрямований на згладжування нерівностей на границях об'єктів у відтворюваній графіці.

У порівнянні з іншими методами згладжування, як от SSAA (Super Sample Anti-Aliasing), DLSS працює з використанням високороздільних зображень для відтворення графіки з більшою якістю та заповненням деталей у вихідній роздільній здатності. Аспект "глибокого навчання" є важливим в технології Nvidia, де за допомогою машинного навчання створюються моделі штучного інтелекту, які аналізують високороздільні дані і заповнюють прогалини у зображеннях. Цей

підхід дозволяє використовувати переваги суперсемплінгу без великих обчислювальних витрат, а також забезпечує автономну роботу моделей AI віддалено від основного комп'ютера, що дозволяє отримати значний приріст якості зображення.

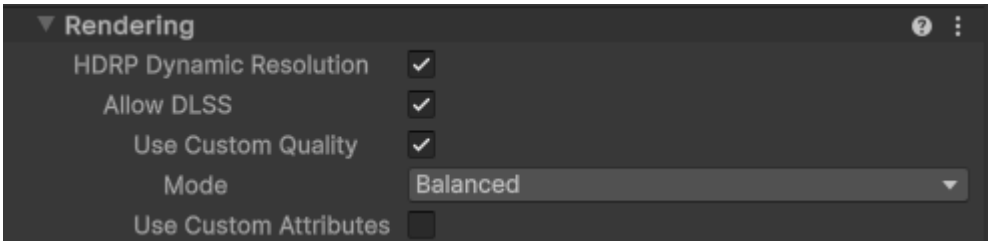

Рисунок 3.4 – Використання технології DLSS

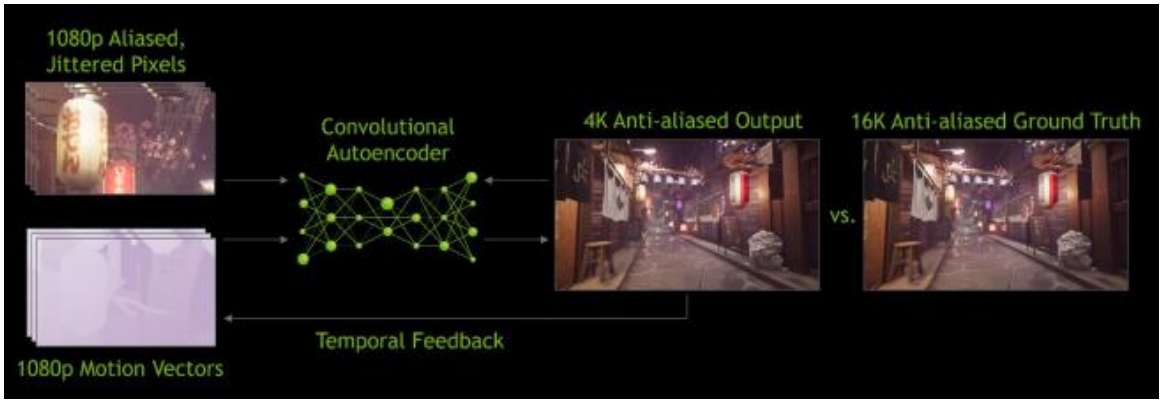

Рисунок 3.5 – Пайплайн DLSS

Все це можливо завдяки ядрам Tensor, які доступні тільки в процесорах RTX. Наприклад, RTX 20 містять ядра першого покоління, тоді як 30 серія вже оснащена ядрами другого покоління, що забезпечує кращу продуктивність на кожному з них. Технологія DLSS є результатом довгого навчання штучного інтелекту від Nvidia для створення ігор.

У суті, DLSS є покращеною версією Nvidia Ansel для оптимізації скріншотів. Вона спочатку відтворює зображення нижчої якості, а потім застосовує різні ефекти для покращення його продуктивності. Цей процес схожий на збільшення роздільної здатності, але може призвести до різних кадрових частот. Проте загалом, це приводить до кращих результатів без значної втрати якості зображення.

За словами Nvidia, DLSS і трасування променів можуть збільшити продуктивність до 75%, що є вражаючим показником. Ці технології та ефекти обробки графіки додають великий рівень реалізму і деталізації до графіки у нашій грі, роблячи її більш захоплюючою та ефектною для гравців.

DLSS використовує свою базу даних, навчену на зображеннях високої роздільності, щоб покращити зображення гри, яке було відтворене з меншою роздільною здатністю. Ідея полягає в тому, щоб гри, які були відтворені в низькій роздільності, виглядали так, ніби вони працюють на вищому рівні деталізації, щоб досягти кращого зовнішнього вигляду. Наприклад, DLSS 2.0 може збільшити роздільну здатність гри в чотири рази, перетворюючи її з відтворенням у 1080p до виведення в 4K.

DLSS вирішує проблему можливих артефактів і помилок, які зазвичай виникають при використанні традиційних методів високої роздільної здатності. Він спроектований таким чином, щоб працювати з цими помилками та створювати ще кращі зображення. DLSS дозволяє значно підвищити продуктивність, не жертвуючи зовнішнім виглядом гри; навпаки, це може зробити гру ще більш якісною. Під час використання DLSS гра спочатку відтворюється у нижчій роздільній здатності (зазвичай 1440p), а потім за допомогою навченого алгоритму штучного інтелекту визначається, як вона виглядала б у вищій роздільності (зазвичай 4K). Це досягається за допомогою ефектів згладжування та автоматичного підвищення різкості. Також DLSS усуває візуальні артефакти, які можуть виникнути при збільшенні роздільної здатності, та використовує їх для покращення деталізації зображення.

Алгоритм штучного інтелекту навчений аналізувати конкретні ігри з надзвичайно високою роздільною здатністю, наприклад, 64-кратною надвибіркою, та зменшувати їх розмір до кількох мегабайт, перш ніж включати цей алгоритм у останні версії драйверів Nvidia для доступу геймерів по всьому світу. Спочатку Nvidia мусила проводити цей процес окремо для кожної гри. Однак у DLSS 2.0 Nvidia пропонує універсальне рішення, тому модель штучного інтелекту більше не потребує індивідуального навчання для кожної гри.

Також існує технологія AMD FidelityFX Super Resolution (FSR). Технологія є відповіддю від AMD на технологію супервибірки глибокого навчання Nvidia (DLSS). Подібно до DLSS, FSR дозволяє грі виглядати так, ніби вона має вищу роздільну здатність, ніж фактично. Наприклад, рушій може відтворювати гру з роздільною здатністю 1080p, а потім FSR заповнить пропущені пікселі, щоб зробити вихід схожим на 1440p.

Наразі існують дві версії FSR, причому FSR 2.0 є значно вдосконаленою по відношенню до оригіналу. Більшість ігор зараз підтримують лише FSR 1.0, але в майбутньому друга версія буде стандартом. Між ними є кілька відмінностей, які ми розглянемо пізніше.

Одна з ключових відмінностей між FSR і DLSS - це сумісність з обладнанням. Для DLSS необхідний графічний процесор Nvidia RTX серії 20 або 30, тоді як FSR працює як з відеокартами AMD, так і Nvidia. Офіційна підтримка охоплює GTX 10-серії та Radeon RX 400-серії, хоча FSR також може працювати на старішому обладнанні.

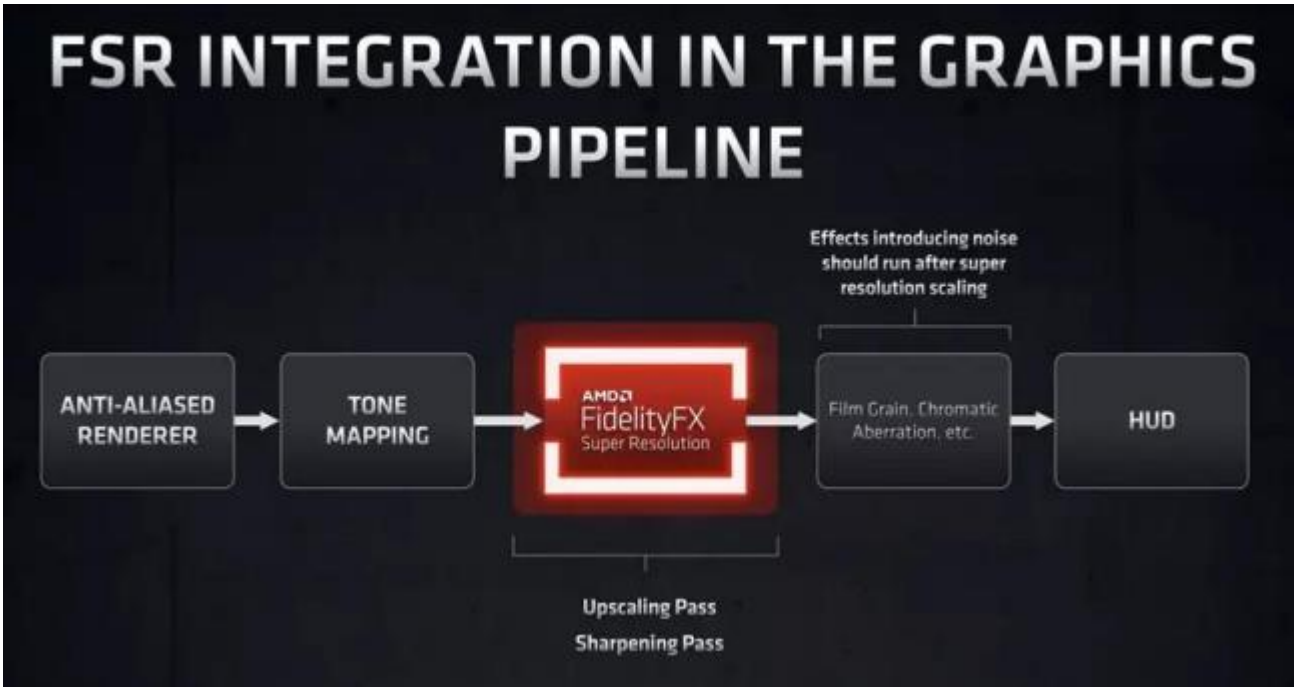

Рисунок 3.6 – Пайплайн FSR

Крім цього, вихідний код FSR доступний безкоштовно для розробників на

платформі AMD GPUOpen і може бути використаний через рушії Unreal Engine 4 і Unity. Це означає, що будь-який розробник, незалежно від свого бюджету або зв'язків, може використовувати FSR у своїх іграх. Проте FSR доступний лише в застосунках, де розробники вирішили включити його.

AMD також пропонує функцію Radeon Super Resolution (RSR) спеціально для графічних процесорів Radeon. По суті, це FSR 1.0, який доступний через драйвер AMD і дозволяє застосовувати масштабування до будь-якої гри, якщо у вас є підтримуваний GPU.

FSR 1.0 і 2.0 працюють по-різному, але вони базуються на одному ядрі. Обидві функції надвибірки використовують алгоритм Ланцоша для масштабування, який починається з передачі зображення з низькою роздільною здатністю для підвищення якості та додаткових деталей на основі алгоритму. Потім FSR виконує проходження різкості, щоб відновити більше деталей.

Однак FSR відрізняється від DLSS тим, що він не використовує штучний інтелект для прискорення. Для DLSS необхідні виділені ядра Tensor на відеокартах RTX для запуску моделі штучного інтелекту, яка допомагає в масштабуванні. FSR, натомість, є частиною конвеєра рендерингу гри і може працювати з відеокартами різних виробників.

Intel XeSS, або Xe Super Sampling, це функція масштабування, доступна на відеокартах Intel Arc Alchemist. Ця технологія працює шляхом відтворення гри з меншою роздільною здатністю та подальшого масштабування за допомогою машинного навчання та спеціального апаратного забезпечення AI, вбудованого в GPU. Такий підхід допомагає покращити продуктивність, зберігаючи при цьому високу якість зображення. Intel XeSS дуже схожий на DLSS від Nvidia, але з різницею у підтримці відеокарт від різних виробників.

Основна відмінність між Intel XeSS і DLSS полягає в тому, що XeSS підтримує відеокарти від різних виробників, тоді як DLSS обмежено відеокартами Nvidia. Це досягнуто завдяки двом різним версіям XeSS від Intel — одна для графічних процесорів Arc, таких як Arc A770 і A750, інша — для відеокарт інших виробників. Версія для Intel Arc, ймовірно, має кращу ефективність завдяки

вдосконаленій моделі масштабування та підтримці ядер штучного інтелекту Xe Matrix Extension (XMX). У той час як XeSS, який можуть випробувати власники відеокарт AMD і Nvidia, має менш надійну модель масштабування та використовує інструкції DP4a, що робить його менш ефективним порівняно з ядрами XMX від Intel.

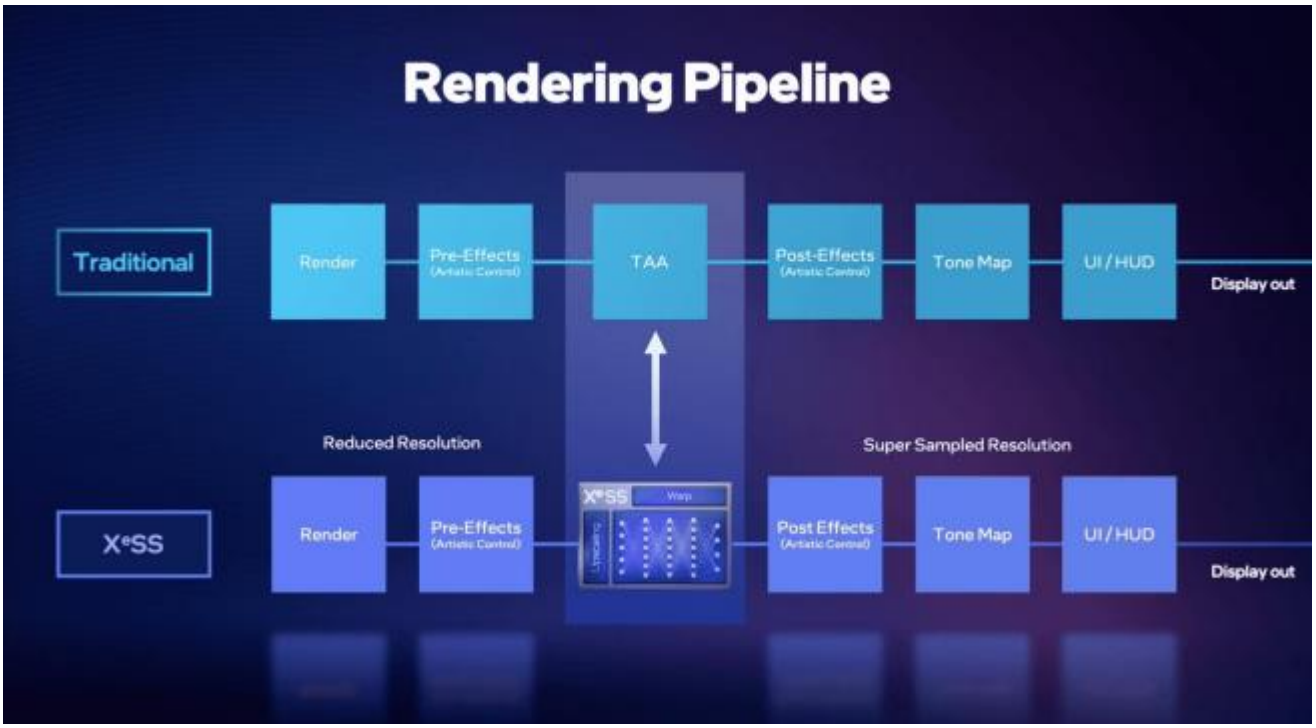

Рисунок 3.7 – Пайплайн Intel XeSS

XeSS — це досить простий у розумінні процес. Натомість, щоб запустити гру зі створеною нами роздільною здатністю (скажімо, 4K), вона використовує різні дані з гри з меншою роздільністю (наприклад, 1080p). Ці дані потрапляють до механізму масштабування XeSS, який має в собі модель штучного інтелекту для покращення масштабування. У результаті має бути зображення, яке теоретично схоже на оригінальне, але з вищою продуктивністю.

Щоб уточнити, XeSS використовує глибину, рух, колір і освітлення з гри. Після масштабування результати зберігаються, а потім передаються в наступний кадр для продовження процесу. У той час, XeSS використовує тимчасове згладжування (TAA) для чистоти зображення. Це замінює традиційні методи, оскільки XeSS може масштабувати та згладжувати в одному кроці. Графічні

карти Intel Arc Alchemist мають ядра XMX, які використовуються для масштабування за допомогою моделі штучного інтелекту. Ці ядра подібні до Tensor від Nvidia в RTX 30 і, за словами Intel, надають найкращу якість з XeSS. Але XeSS також може працювати і без цих ядер, якщо карта підтримує інструкції DP4a, які корисні для роботи з штучним інтелектом.

Таблиця 3.1 – Порівняння характеристик технологій покращення зображення

Характеристика	Nvidia DLSS	AMD FSR	Intel Xess
Залежність від апаратних додатків	+	-	+
Можливість роботи на прискорювачах інших виробників	-	+	+
Відкритий доступ	-	+	-

3.2 Проведення тестів продуктивності та якості гри з використанням графічного процесора на базі RTX, аналіз результатів тестування

Процес проведення тестів продуктивності та якості гри з використанням графічного процесора на базі RTX є критичним для оцінки потужності та ефективності цього обладнання у різних ігрових умовах. В ході тестування зазвичай оцінюються такі аспекти, як кількість кадрів на секунду (FPS), час завантаження та реакції гри, якість зображення, вплив на ресурси системи тощо.

Початковим етапом тестування є підготовка тестового стенду, що включає в себе налаштування графічного процесора, обране графічне ядро, драйвери та ігрове середовище. Після цього проводяться тести на різних налаштуваннях графіки та роздільній здатності, щоб оцінити ефективність процесора в різних умовах гри.

Один з ключових аспектів тестування - це аналіз динаміки FPS під час геймплею. Це допомагає з'ясувати, наскільки графічний процесор може ефективно обробляти складні візуальні ефекти та сцени без зниження

продуктивності. Крім того, оцінюється якість зображення, рівень деталізації, наявність артефактів та загальний реалізм гри.

Після завершення тестів проводиться аналіз отриманих даних, включаючи графіки зміни FPS в залежності від умов гри, порівняльні таблиці продуктивності на різних налаштуваннях, а також оцінка впливу графічного процесора на ресурси системи. Цей аналіз допомагає розуміти можливості та обмеження графічного процесора на базі RTX в контексті ігрових завдань та вимог користувачів.

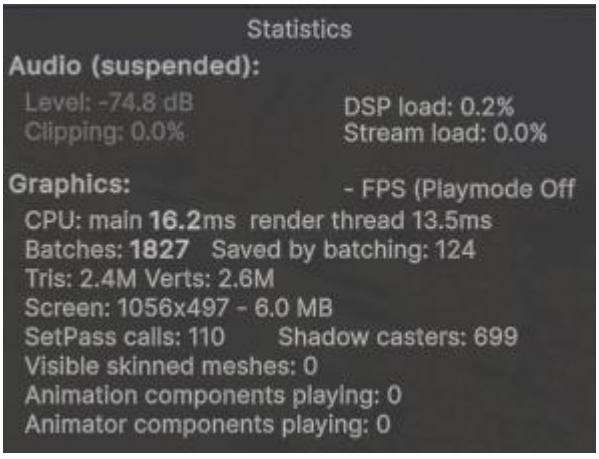

Рисунок 3.8 – Вікно із статистикою продуктивності під час тестування гри

Для ефективного відстеження продуктивності та тестування гри в Unity3D існує кілька ключових інструментів, які допомагають розробникам виявляти й виправляти проблеми для покращення якості ігор.

Unity Profiler - цей інструмент надає детальну інформацію про використання ресурсів під час виконання гри. Ви можете відслідковувати час виконання кожного елемента гри, використання CPU, пам'яті та інших ресурсів. Це дозволяє ідентифікувати потенційні проблеми з продуктивністю та оптимізувати їх.

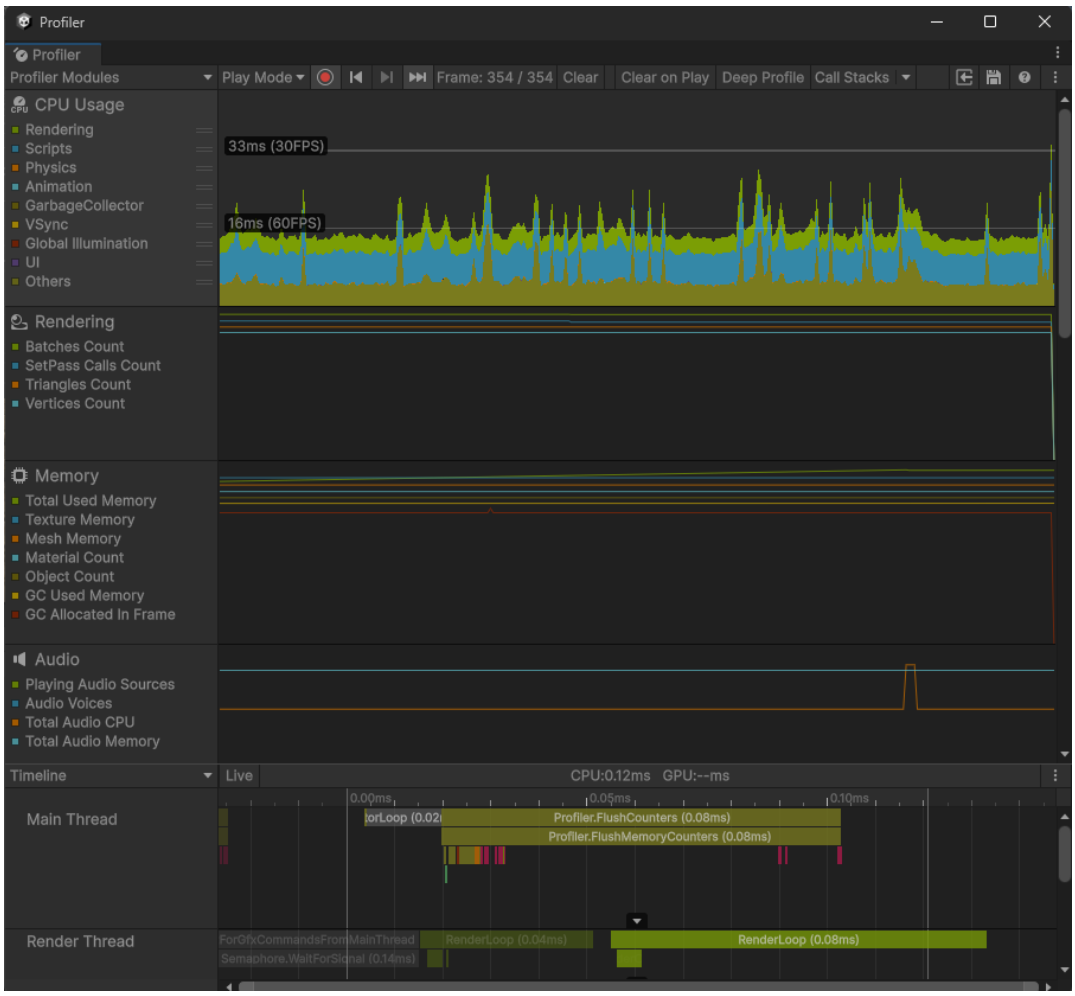

Рисунок 3.9 – Вікно Profiler, відображена в ньому інформація про використання ресурсів

Unity Test Framework - фреймворк дозволяє автоматизувати тестування різних аспектів гри, таких як поведінка ігрових об'єктів, реакція на взаємодію гравця та інші. Ви можете створювати тести, які перевіряють функціональність гри, а це допомагає виявляти й усувати помилки ще на ранніх стадіях розробки.

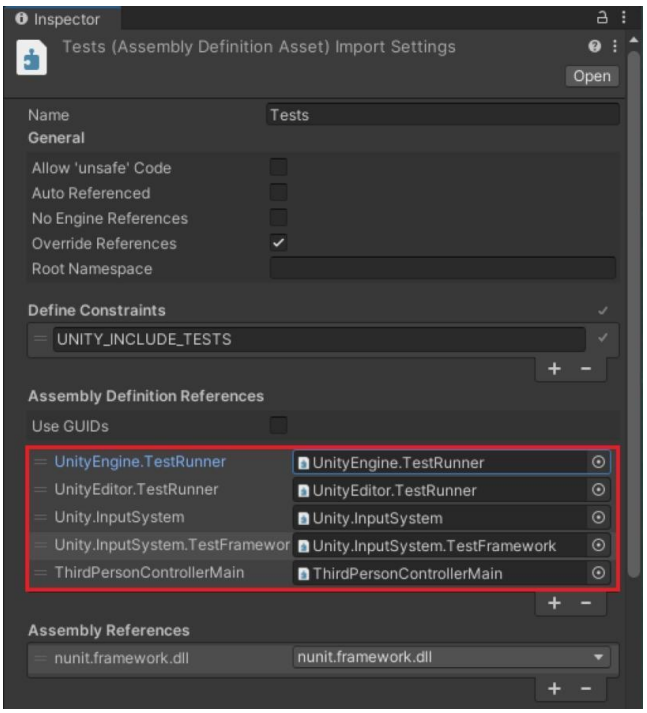

Рисунок 3.10 – Вікно Inspector, додавання посилань на тести

Unity Test Runner - інструмент дозволяє запускати тести, які ви створили в Unity Test Framework, і аналізувати їх результати. Ви можете виконувати автоматичні тести під час розробки гри та під час тестування перед релізом, що сприяє виявленню й усуненню проблем швидше.

Unity Remote – інструмент, який дозволяє підключити ваше мобільне пристрій до Unity Editor, щоб тестувати гру безпосередньо на ньому. Ви можете перевіряти реакцію гри на реальних пристроях та виявляти проблеми, які можуть виникати на конкретних платформах.

Unity Analytics - сервіс надає дані про користувацький досвід гравців у вашій грі. Ви можете отримувати інформацію про час гри, поведінку гравців та їхні дії, що допомагає зрозуміти, як гравці взаємодіють з грою та як можна її покращити.

Ці інструменти разом надають розробникам необхідні засоби для відстеження продуктивності гри, тестування функціональності та виявлення проблем, що допомагає створити якісну та оптимізовану гру для гравців.

Ми провели значну роботу з оптимізації нашої гри в Unity3D. Наша ціль була забезпечити оптимальні налаштування, які б зберігали високу якість графіки і

одночасно забезпечували стабільну продуктивність. Для досягнення цієї мети ми використовували різні інструменти відслідковування продутивності.

Один з основних інструментів, які ми використовували, - Unity Profiler. Цей інструмент дозволяє нам аналізувати різні аспекти продуктивності гри, включаючи використання CPU, пам'яті, GPU, і так далі. З його допомогою ми виявили та виправили деякі недоліки, що впливали на швидкість гри.

Крім того, ми налаштували різні параметри, такі як рівень деталізації графіки, розмір текстур і якість освітлення, щоб забезпечити оптимальний баланс між візуальною привабливістю та продуктивністю гри. Ми провели численні тести, щоб визначити оптимальні налаштування для різних конфігурацій обладнання.

Найголовніше - вирішити проблему освітлення статичних об'єктів. Розрахунок освітлення в реальному часі завжди важкий з точки зору ресурсів. У цьому проекті статичні об'єкти розміщені у початковій сцені, і ми розглянемо переваги запікання світла на їхньому прикладі.

Спочатку оглянемо сцену з реальним часом освітлення. Такий підхід не завжди дає високу якість - тіні можуть бути неправильними, а деякі деталі можуть виглядати неякісно. Для уникнення таких проблем потрібно підвищити якість освітлення, наприклад, збільшивши розмір тіней у реальному часі та використовуючи "м'які тіні". Однак це збільшить навантаження на обчислення освітлення.

Тепер проведемо аналіз рендерінгу сцени з реальним часом освітлення. Хоча цей підхід зручний з точки зору динаміки, він може вимагати більше обчислювальних ресурсів для досягнення високої якості.

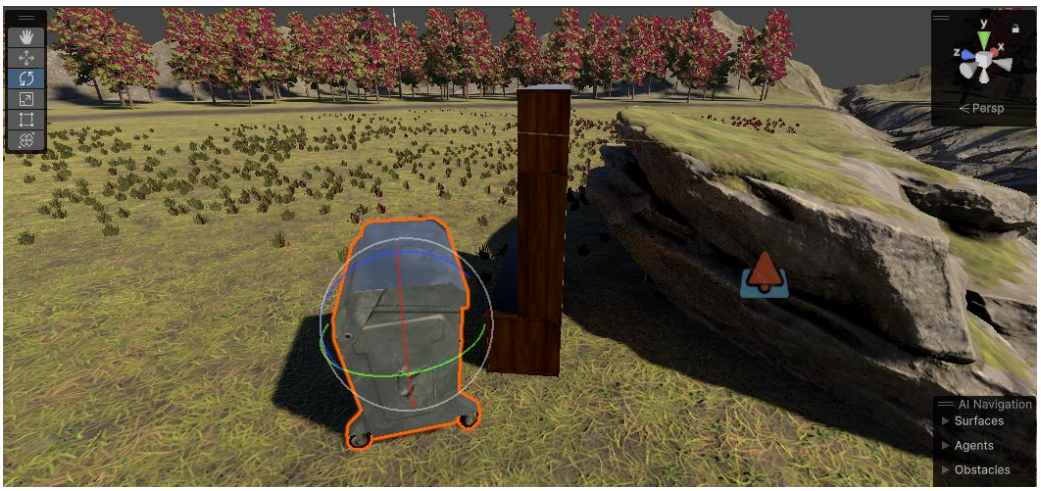

Рисунок 3.11 – Сцена з динамічним освітленням

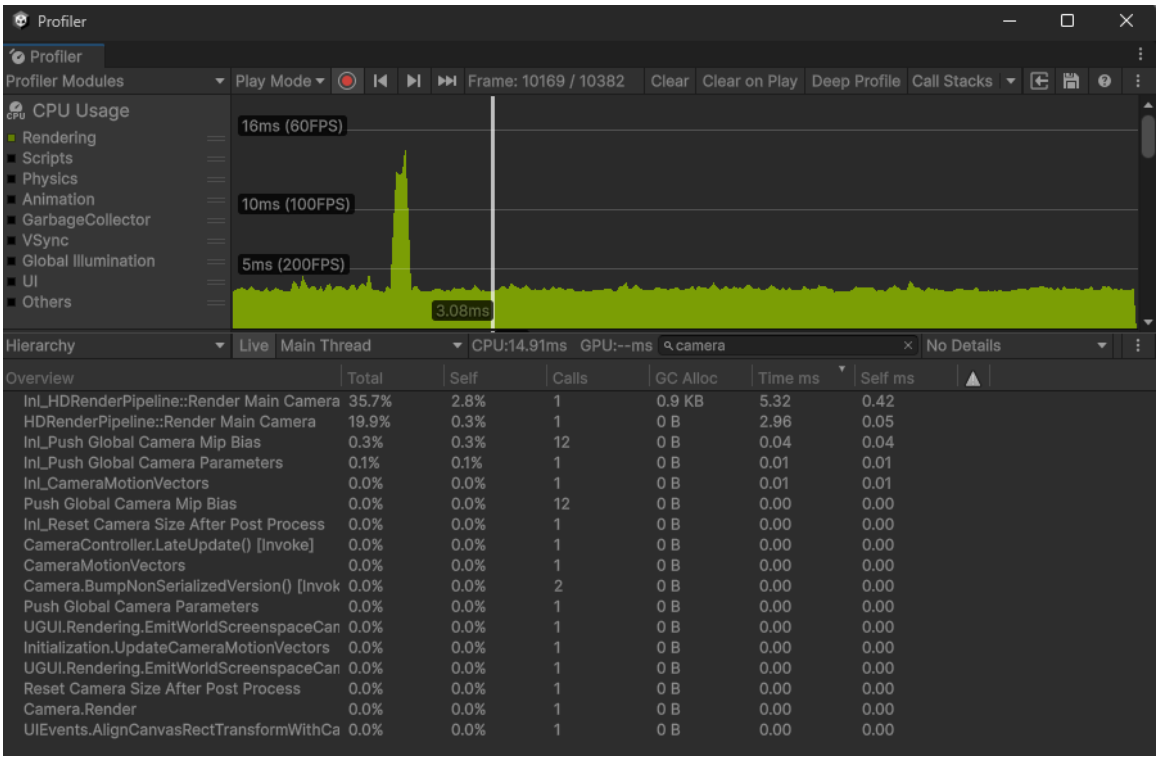

Рисунок 3.12 – Аналіз використання CPU і GPU

Для створення одного кадру потрібно приблизно 5 мілісекунд процесорного часу, а середня частота кадрів становить близько 48. Також на відтворення одного кадру витрачається 35% загального часу роботи процесора. Додаткові показники можна знайти на рисунку 3.11.

Тепер перейдемо до процесу запікання освітлення. Для цього необхідно налаштувати параметри освітлення та параметри LightMapper.

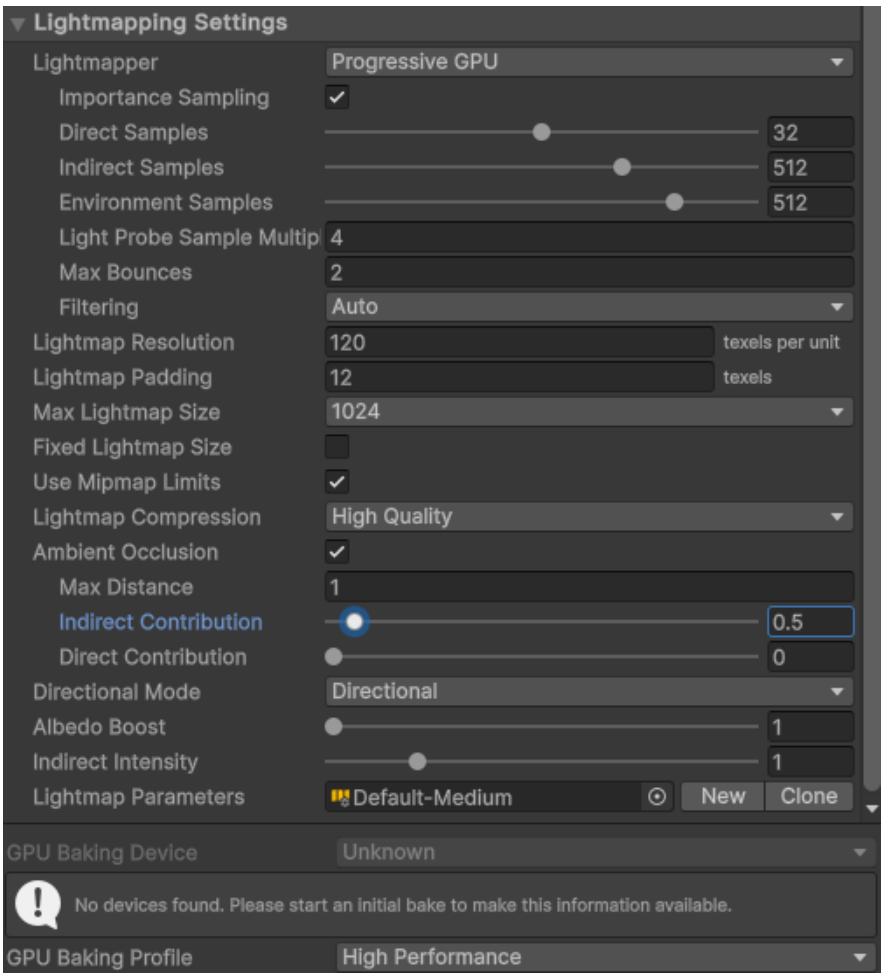

Рисунок 3.13 – Налаштування запікання світла

На рисунку 3.3. представлені параметри LightMapper. Для запікання освітлення було встановлено роздільність світлової карти 1024 x 1024 пікселів. Direct Samples - 32. Це кількість вибірок (шляхів), які були взяті з кожного текселю. Цей параметр регулює кількість вибірок, які Progressive Lightmapper використовує для розрахунків прямого освітлення. Збільшення цього значення може покращити якість світлових карт, але збільшує час запікання. Також, для покращення якості запікання світла, слід вимкнути стиснення світлових карт, оскільки стиснення виконується окремо після процесу запікання. Bounces - 2. Це кількість відбиттів світла під час трасування шляхів. Для більшості сцен достатньо 2 відбиттів. Важливо пам'ятати, що запікання світла працює тільки зі статичними об'єктами на сцені, тому для кожного об'єкта, який не буде переміщуватися або змінювати свою форму, слід встановити «Static». Також

джерело освітлення повинно бути установлено в режим запікання. Після проведення запікання світла ми отримуємо результат, показаний на рисунку 3.13.

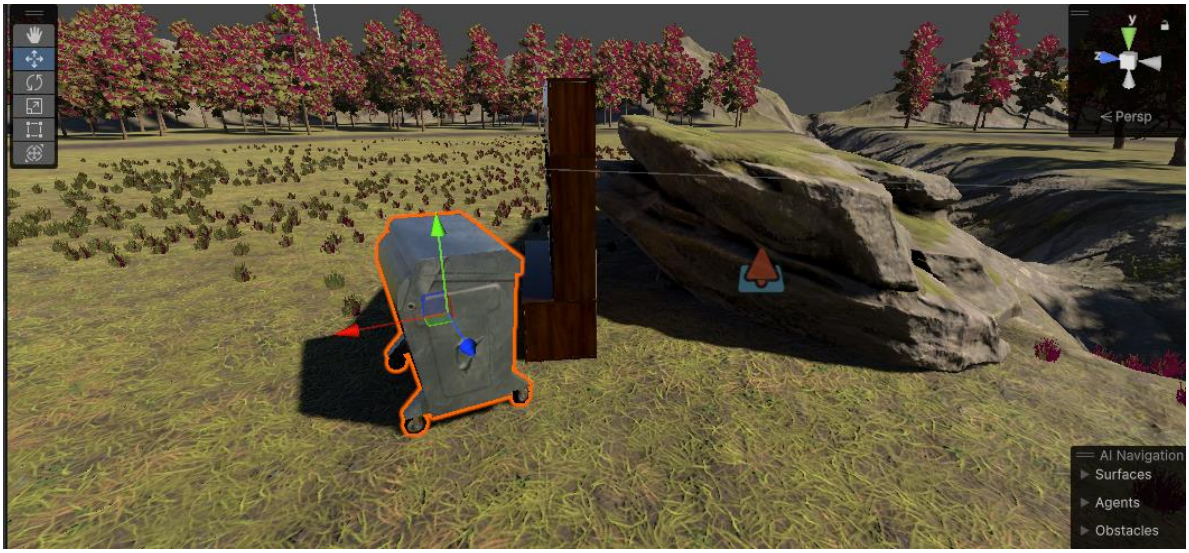

Рисунок 3.14 – Налаштування запікання світла

Після запікання, ми можемо побачити інформацію про кількість згенерованих світлових карт і обсяг пам'яті, зайнятий цими картами

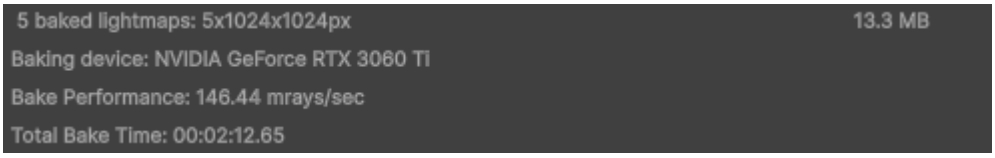

Рисунок 3.15 – Інформація про запечені світлові карти

Приклад самої світлової карти зображено на рисунку 3.14.

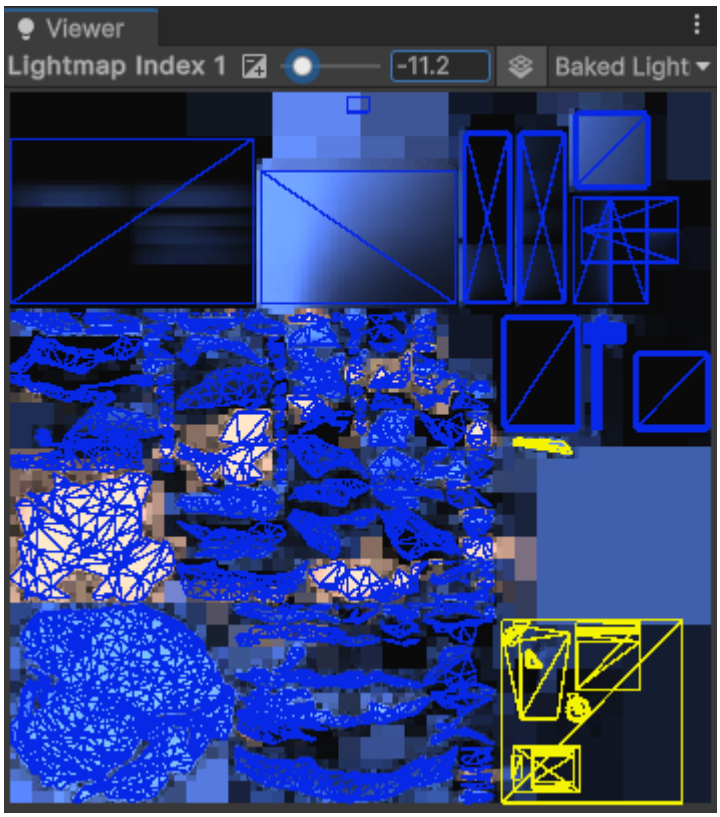

Рисунок 3.16 – Обчислена світлова карта

Після запікання світла проведемо оцінку продуктивності. Середній час відображення одного кадру скоротився з 5.32 мс до 3.4 мс, а використання часу процесора для відображення кадру зменшилося з 35% до 22%. Середня частота кадрів зросла до 72. Показники використання CPU показані на рисунку 3.15.

Отримані результати свідчать про вплив освітлення на загальну продуктивність гри. У великих сценах з багатьма джерелами освітлення оптимізація є критичною для забезпечення плавності та задоволення гравця.

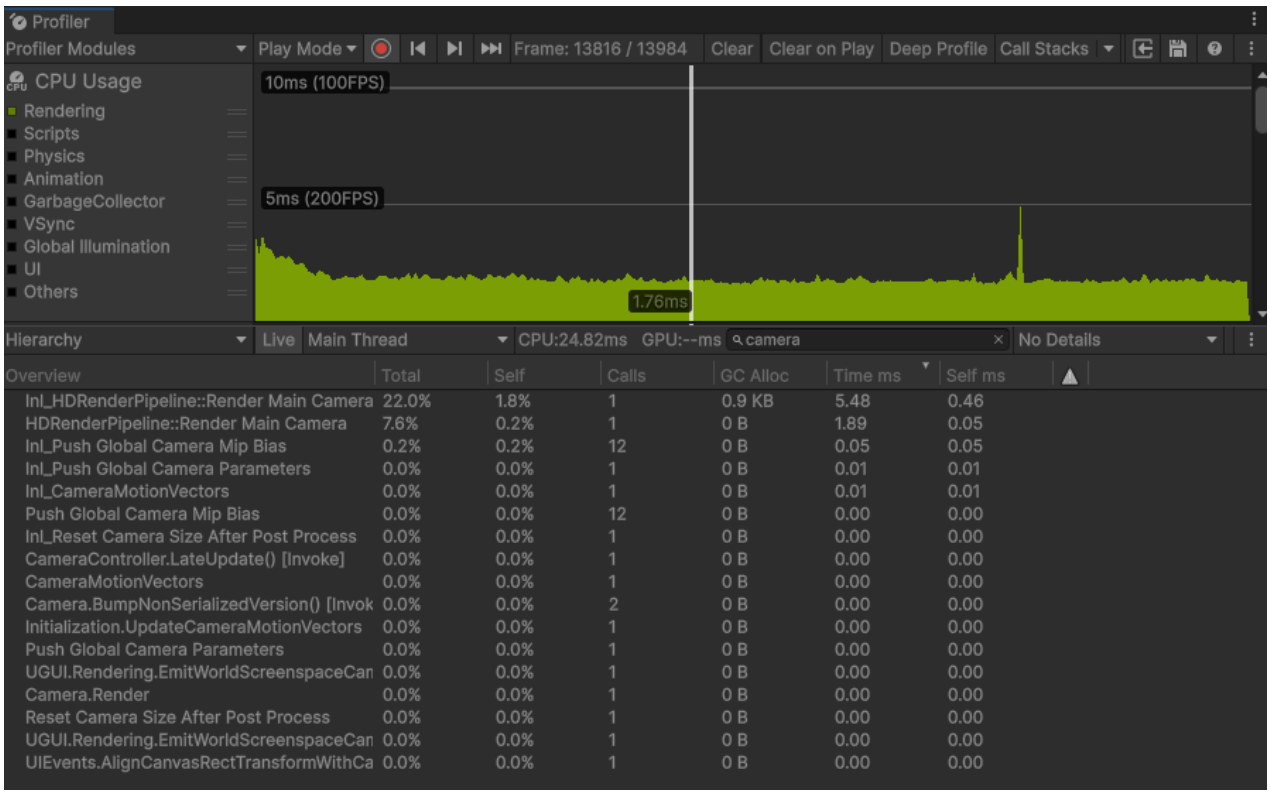

Рисунок 3.17 – Аналіз використання CPU і GPU

Наступним етапом у вдосконаленні гри є використання системи LOD (рівні деталізації). Під час гри може виникати ситуація, коли на сцені відображається велика кількість моделей з великою кількістю вершин і полігонів. Це може призвести до значного навантаження на процесор для обробки цих моделей. LOD дозволяє зменшити кількість вершин, враховуючи віддаленість об'єктів від гравця. Чим далі об'єкт від гравця, тим менше деталізована версія цього об'єкта відображається під час рендерингу.

Для ілюстрації цього принципу ми взяли модель каменю, яка є частиною гри. Генерація цього каменю відбувається на певній відстані від гравця, яка зменшується з часом. Тому при генерації нам не потрібно високодеталізованої версії, а лише при наближенні гравця до об'єкта на відстань, визначену параметрами LOD.

Найбільш деталізований рівень LOD містить 13798 вершин. Кожен наступний рівень містить у три рази менше вершин, ніж попередній.

Unity має зручний інструмент: якщо модель, експортована з 3D редактора у

форматі FBX, має відповідну назву LOD (LOD0, LOD1), то движок автоматично додає до цієї моделі групу LOD. Залишається лише налаштувати ці групи в Unity.

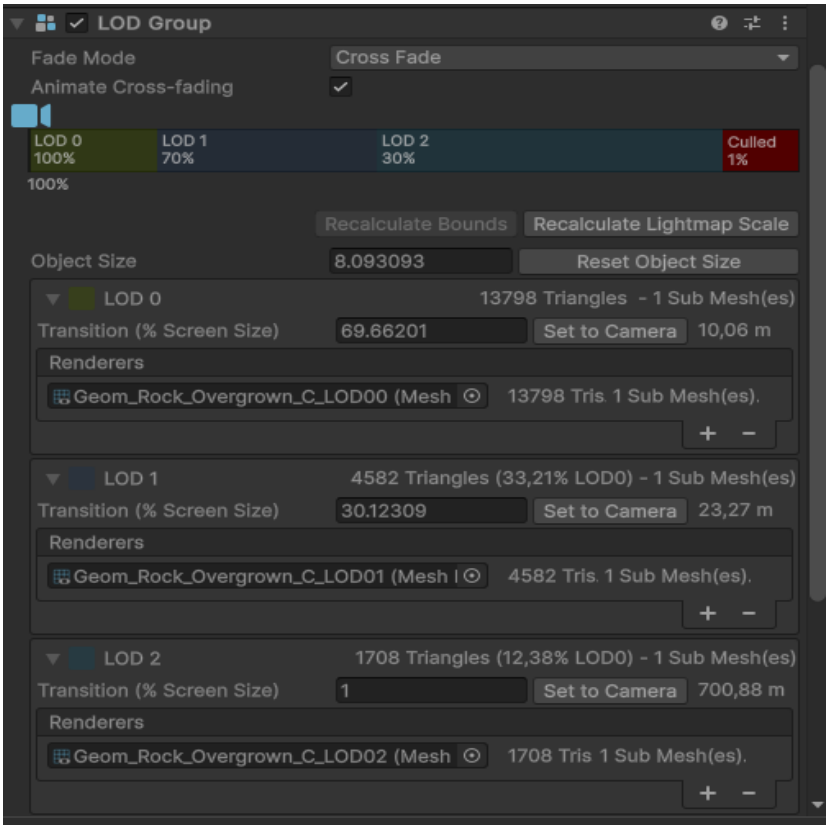

Рисунок 3.18 – Налаштування LOD групи

Застосувавши LOD групи до всіх моделей, ми успішно зменшили кількість вершин майже вдвічі, зберігаючи при цьому візуальну привабливість сцени. Це призвело до покращення частоти кадрів і загальної продуктивності гри. Результати цих змін відображено в таблиці 3.2.

Таблиця 3.2 – Порівняння характеристик технологій покращення зображення

Характеристика	Без LOD	З LOD
Вершин	124210	84200
Полігонів	70400	50500
Частота кадрів в секунду	70	76

Тепер розглянемо використання технології DLSS. Ця технологія повинна зменшити навантаження на графічний процесор. Відповідно спочатку треба її увімкнути в налаштуваннях проекта.

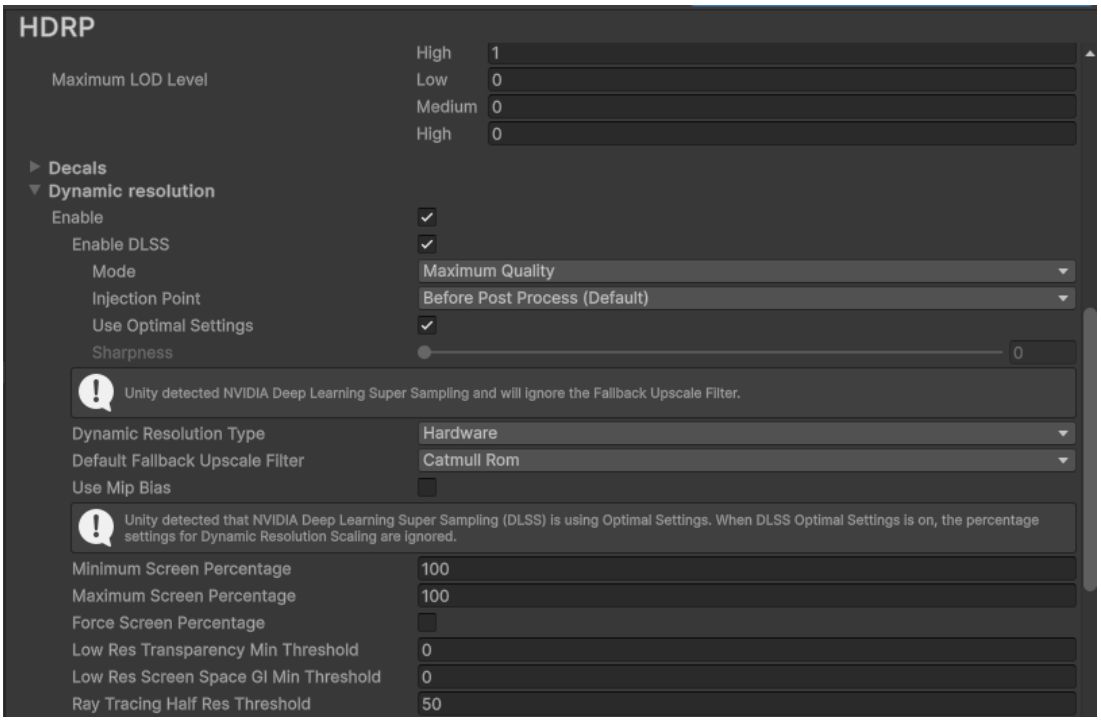

Рисунок 3.19 – Налаштування DLSS

Настройки DLSS (Deep Learning Super Sampling) в Unity3D можуть бути різними, в залежності від потреб проекту та обраної стратегії використання технології.

Можемо вибрати режим продуктивності - максимальна ефективність використання ресурсів для забезпечення високої частоти кадрів. Може бути корисним для швидкісних ігор або VR-проектів.

Можемо вибрати режим збалансованої якості - компроміс між якістю зображення та продуктивністю. Підходить для більшості ігрових сценаріїв.

Можемо вибрати режим високої якості - забезпечує максимальну деталізацію зображення за рахунок більшої нагрузки на ресурси комп'ютера.

Нативна роздільна здатність - використання роздільності, заданої для проекту без збільшення або зменшення масштабу.

Масштабування до вищого розширення - використання DLSS для збільшення

роздільності зображення для покращення якості.

Рівень деталізації (низький, середній, високий) - налаштування рівня деталізації обробки текстур та ефектів для відповідності вимогам проекту.

Антиаліасин - TAA (Temporal Anti-Aliasing) - використання технології TAA для зменшення ефектів аліасингу та поліпшення зображення.

Автоматичний вибір режиму DLSS - Unity3D може автоматично вибирати оптимальні налаштування DLSS в залежності від характеристик системи та поточних умов.

Ці параметри можна налаштовувати в редакторі Unity3D або програмно через API DLSS для досягнення оптимальної якості зображення та продуктивності в проекті.

Після основних налаштувань DLSS, його потрібно увімкнути конкретно на ігровій камері у сцені.

Рисунок 3.20 – Налаштування DLSS на камері

Отже, проведені тести до та після використання DLSS, показали наступні результати. До ввімкнення цієї технології, в середньому на сцені ми мали 75 кадрів на секунду, а вже після того, як ми почали використовувати дану технологію, наші кадри виросли приблизно до 89.

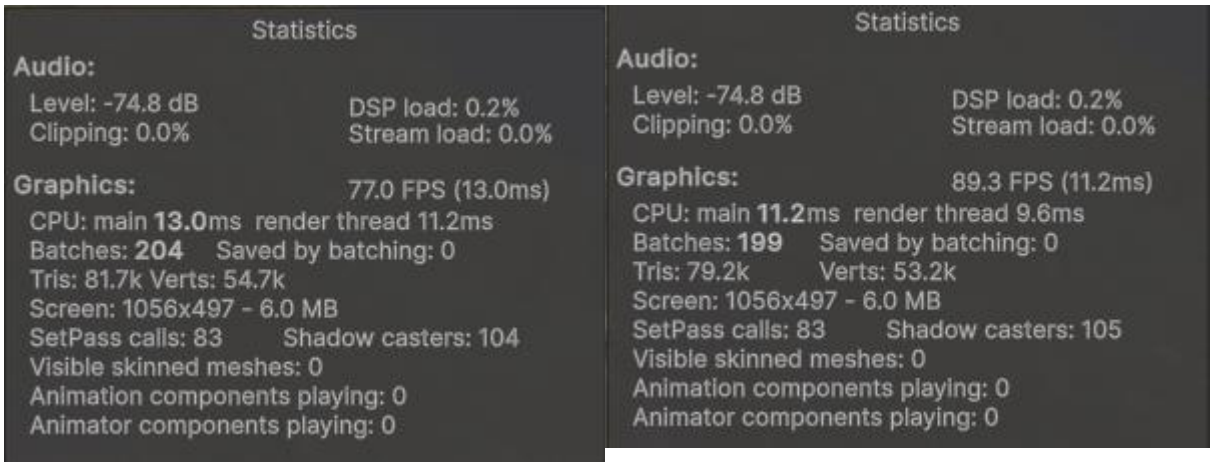

Рисунок 3.21 – Результат використання технології, до та після

ВИСНОВКИ

У результаті цієї роботи була розроблена методика розробки гри, яка використовує графічне ядро типу RTX. В процесі розробки були враховані основні архітектурні особливості цього графічного процесора, що дозволило досягти оптимальної продуктивності та якості зображення.

Розібрали архітектуру графічних процесорів типу RTX, які їх переваги, з яких компонентів вони складаються, як виглядають. Після чого підібрали ігровий рушій, який підтримує всі основні технології, і ним виявився рушій під назвою Unity3D.

Основні ігрові механіки були успішно реалізовані на цьому рушії, і включають в себе елементи екшну, битви з NPC, реалізацію ШІ, реалізацію UI, що робить гру цікавою. Також були впроваджені основні графічні ефекти, які базуються на трасуванні променів, такі як реалістичне освітлення, тіні, відзеркалення. Розглянули як все це додається до проекту та налаштовується.

Завдяки використанню графічного ядра RTX вдалося досягти вражаючої деталізації та реалістичності зображення, що створює відчуття присутності в ігровому світі. Наявність оптимально налаштованих настройок DLSS дозволила підтримувати високу частоту кадрів при збереженні високої якості зображення.

Також відбулася робота над оптимізацією. Ми розглянули основні інструменти, та провели роботу по оптимізації використання ресурсів нашого графічного процесора, а також центрального процесора. Досягли в цьому успіхів – піднявши середній показник частоти кадрів на одній сцені з 55 кадрів на секунду, до 89.

У цілому, розробка гри на відповідному графічному ядрі RTX виявилася успішною і вдалою, що дозволило створити захоплюючий та візуально привабливий ігровий досвід для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ray-triangle intersection. / Brian Curless – University of Washington Computer Science & Engineering, 2006. – 1-5 с.
2. Unity 5 Game Optimization. Master performance optimization for Unity3D / Кріс Дікінсон - ДМК Пресс, 2016. – 306 с.
3. Game Architecture and Design / Andrew Rollings, Dave Morris - New Riders Pub, 2015. – 1040 с.
4. NVIDIA TURING GPU ARCHITECTURE / Nvidia, 2018. – 58 с.
5. Game Engine Architecture 2nd Edition / Jason Gregory - CRC Press 2014 – 1052 с.
6. Rendering (computer graphics) / URL: https://en.wikipedia.org/wiki/Rendering_(computer_graphics)
7. Nvidia Turing GPU Architecture / URL: https://www.hardwarezone.com.sg/feature-what-you-need-know-about-ray-tracing-and-nvidias-turing-architecture/rt-cores-and-tensor-cores
8. Мова програмування C# 7.0 та платформи .NET та .NET Core / Філіп Джепікс, Ендрю Троєлсен – apress, 2017. – 184 с.
9. Unity и C#. Геймдев от идеи до реализации. 2-е изд. / Джереми Гибсон Бонд - Print2print, 2019. – 76 с.
10. Комп'ютерна графіка / Михайло Пічугін, Іван Канкін, Володимир Воротніков - Центр навчальної літератури, 2019. – 182 с.

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

Державний Університет Телекомунікацій
Факультет Інформаційних технологій
Кафедра Комп'ютерної інженерії

БАКАЛАВРСЬКА РОБОТА

на тему: «Дослідження нейрокомп'ютерних інтерфейсів як аналогів механічної периферії»

Виконав студент групи КІД-41
Тужилін Денис Ігорьович

Керівник дипломної роботи:
Бученко Ігор Анатолійович,
викладач кафедри КІ

Київ – 2023