Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів спрощення та оптимізації часток
для портативних ігрових пристроїв
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-22-4

Шульга М.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник доц. Назаров О.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Шульзі Максимові Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів спрощення та оптимізації часток для портативних ігрових пристроїв»

Затверджена наказом по університету від 29.03. 2024р. № 250 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18.06.2024

3. Вихідні дані до роботи дослідження існуючих систем симуляції часток, методів їх оптимізації, та ключових відмінностей найбільш популярних систем симуляції часток, формування уявлення для подальшого аналізу та роботи над створенням свого методу оптимізації, або вдосконалення вже існуючих

4. Перелік питань, що потрібно опрацювати в роботі
аналіз та порівняння існуючих систем симуляції часток та движків, вибір підходящої системи симуляції для дослідження, проектування інтерактивної системи для проведення дослідження, написання програмних рішень, проведення експериментів та аналіз отриманих результатів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	30.03 – 14.04.24	*виконано*
2	Аналіз та вибір API для дослідження	15.04 – 24.04.24	*виконано*
3	Аналіз та моделювання предметної області	25.04 – 28.04.24	*виконано*
4	Планування експериментів	29.04 – 08.05.24	*виконано*
5	Програмна реалізація кожного з обраних для дослідження API	09.05 – 19.05.24	*виконано*
6	Експериментальні дослідження	20.05 – 25.05.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	25.05 – 28.05.24	*виконано*
8	Написання та оформлення статті та тез доповіді	28.05 – 31.05.24	*виконано*
9	Підготовка пояснювальної записки	01.06 – 09.05.24	*виконано*
10	Підготовка презентації та доповіді	10.06 – 12.06.24	*виконано*
11	Нормоконтроль	13.06 – 14.06.24	*виконано*
12	Рецензування	14.06 – 15.06.24	*виконано*
13	Занесення диплома в електронний архів	16.06.2024	*виконано*
14	Попередній захист	17.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	18.06.2024	*виконано*

Дата видачі завдання 30 березня 2024р.

Студент (ка) _____ Шульга М.В.
(підпис)

Керівник роботи _____ доц. Назаров О.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 99 с., 17 рис., 1 табл., 11 джерел, 5 додатків.

МОБІЛЬНІ ПРИСТРОЇ, ОПТИМІЗАЦІЯ, СИСТЕМИ СИМУЛЯЦІЇ ЧАСТОК, BLUEPRINTS, CASCADE, C++, NIAGARA, UNREAL ENGINE 5.

Об’єктом дослідження є системи симуляції часток, їх движки та їх програмні інтерфейси.

Метою роботи є проведення дослідження продуктивності сучасних систем симуляції часток в реальному часі та прийняття оптимізаційних рішень для її покращення.

Методами розробки та проектування є аналіз проблемної області дослідження, вибір системи симуляції часток, проведення експерименту та оптимізаційних дій.

У результаті кваліфікаційної роботи було розроблено програмну систему на базі движка Unreal Engine 5, яка демонструє результати аналізу та проведення оптимізаційних дій.

MOBILE, OPTIMIZATION, PARTICLE SIMULATION SYSTEMS, CASCADE, NIAGARA, UNREAL ENGINE 5.

The object of the research is particle simulation systems, their engines and their software interfaces.

The aim of the work is to study the performance of modern particle simulation systems in real time and make optimization decisions to improve it.

The methods of development and design are the analysis of the problem area of research, the choice of a particle simulation system, the conduct of an experiment and optimization actions.

As a result of the qualification work, a software system based on the Unreal Engine 5 engine was developed, which demonstrates the results of analysis and optimization actions.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Шульга Максим Віталійович, студент гр. ІПЗм-22-4, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів спрощення та оптимізації часток для портативних ігрових пристроїв», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 9
1 Постановка задачі 11
1.1 Опис предметної галузі 11
1.2 Постановка задачі дослідження 12
1.3 Засоби для проведення досліджень 13
2 Теоретичні основи 15
2.1 Огляд та порівняння сучасних ігрових движків 15
2.2 Аналіз індустрії та місця, яке займає в ній Unreal Engine 4-5 18
3 Аналіз систем симуляції часток 28
3.1 Системи симуляції часток, наявні в UE5 28
3.2 Cascade 29
3.3 Niagara 35
3.4 Оптимізація систем симуляції часток 38
4 Експериментальні дослідження 43
4.1 Сцена для тестування 43
4.2 Інструменти тестування 45
5 Аналіз результатів 48
5.1 Аспекти оптимізації 48
5.2 Оптимізація руху часток 49
5.3 Оптимізація хаотичного руху часток 51
5.4 Оптимізація воронкового руху часток 51
5.5 Оптимізація колізії часток 52
5.6 Оптимізація тривимірного шуму 53
Висновки 55
Перелік джерел посилання 56
Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 58
Додаток А 59
Додаток Б 60

Додаток В 80
Додаток Г 89
Додаток Д 99

ПЕРЕЛІК СКОРОЧЕНЬ

CPU – Central Processing Unit

GPU – Graphics Processing Unit

HLSL – High-Level Shader Language

VFX – Visual Effects

2D – Two-Dimensional

3D – Three-Dimensional

ВСТУП

Системи симуляції часток (VFX) - це технологія, яка дозволяє створювати реалістичні ефекти, такі як вогонь, дим, вода, іскри та інше, за допомогою великої кількості маленьких об'єктів, які називаються частинками. Ці частинки можуть мати різні властивості, такі як колір, розмір, форма, швидкість, напрямок тощо, і взаємодіяти одна з одною та з оточенням за певними правилами. Системи симуляції часток широко використовуються у сучасних ігрових движках для створення захоплюючих візуальних ефектів, які покращують іммерсивність і атмосферу гри.

VFX в Unreal Engine є потужним інструментом для створення візуальних ефектів, які додають глибину та реалістичність до відеоігор, анімацій та інтерактивних середовищ. Unreal Engine надає кілька основних інструментів для роботи з ефектами часток, серед яких найважливіші – це Niagara і Cascade.

На сьогоднішній день Niagara є основною системою симуляції часток, рекомендованою для нових проектів, оскільки вона надає більше можливостей і краще інтегрується з сучасними технологіями Unreal Engine. Cascade все ще корисний для старих проектів або для швидкого створення простих ефектів.

Можна виділити основні приклади використання VFX в Unreal Engine:

- ефекти навколишнього середовища. Дим, туман, дощ, сніг та інші атмосферні явища;
- ефекти персонажів. Спеціальні ефекти для магії, здібностей персонажів, вибухів;
- інтерактивні ефекти. Реакція на дії гравця, такі як сліди від куль, бризки води, вогонь.

Unreal Engine надає розробникам всі необхідні інструменти для створення ефектних та реалістичних візуальних ефектів, що робить його популярним вибором серед студій відеоігор і візуальних ефектів.

Метою дослідження є аналіз та порівняння методів використання систем симуляції часток у сучасних ігрових движках, таких як Unity, Unreal Engine, CryEngine та можливість їх використання в портативних пристроях.

Об'єктом дослідження є процеси створення, оптимізації та інтеграції VFX у ігрових проектах, а також їх вплив на графічну якість, продуктивність та іммерсивність гри. Дослідження має на меті виявити переваги та недоліки різних підходів до реалізації VFX, а також запропонувати рекомендації щодо їх ефективного застосування.

У цій кваліфікаційній роботі ми розглянемо основні принципи та методи роботи систем симуляції часток, а також приклади їх застосування у різних жанрах ігор. Ми також проаналізуємо переваги та недоліки цієї технології, а ще й можливості її подальшого розвитку та вдосконалення. Після проведення експериментів та збору інформації будуть запропоновані рішення, які дозволять оптимізувати системи симуляції часток під мобільні пристрої.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної галузі

Поширення інструментальних засобів та технологій розробки мультимедійних програмних засобів (МПЗ) знайшло своє відображення в сфері комп'ютерної ігрової індустрії [1], інтерактивному кіно, застосунках доповненої та віртуальної реальності.

Разом з розвитком інструментальних засобів розробки на передній план виходить необхідність в динамічній оптимізації моделей. Причиною цього слід вважати той факт, що спрощення процесу розробки достатньо складних моделей призвів до зниженню інтересу розробників до багатофакторної оптимізації вже існуючих моделей – більш простим стає процес побудови нової моделі, ніж оптимізаційна модифікація вже існуючої.

Процес оптимізації може бути застосований майже до усіх елементів сучасних рушіїв. Це пов'язано з тим фактом, що рушії забезпечують максимально широкий функціонал для забезпечення дотримання вимог у тих сферах застосування, на які ці рушії спрямовані. Для таких сфер застосування, як кіноіндустрія або симуляції, де моделі виконуються не в реальному часі, оптимізація не є критичною, бо впливає лише на комфорт роботи з проектом, але не на фінальний результат, який може проходити через процедуру рендерінгу у будь-який час. Основною сферою застосування, де проведення процедури оптимізація є критичним, залишаються програмні додатки, що виконуються в реальному часі. Для цього класу додатків головною вимогою щодо процесу оптимізації залишається мінімізація затримок, пов'язаних з оптимізацією, а також максимальна синхронізація функціонування всіх використаних елементів рушія через те, що вони можуть запускатись на різному обладнанні, а також – у майбутньому – масштабуватись на нові портативні пристрої. Саме цей факт привертає особливу увагу, бо мобільність досить активно розвивається і через це оптимізація вже існуючих додатків під нові типи мобільних пристроїв стає як ніколи актуальною.

Процес оптимізації за своїми параметрами залишається складним, ситуативним, та включає в себе багато критеріїв, що потребують індивідуального підходу. Сучасні рушії складаються із модулів, що працюють спільно, та можуть впливати один на одного. Через цей факт їх частіше за все неможливо відключити: так, наприклад, в іграх, які були адаптовані під віртуальну реальність, досить рідко використовується стандартний модуль інтерфейсу, ініціалізація та порахунок якого вимагає багато ресурсів. Також до складу команди розробників не досить часто включають досвідчених фахівців, які можуть розібратись у структурі коду рушія та вимкнути окремий модуль таким чином, щоб це не вплинуло на інші модулі. В деяких рушіях, наприклад, Unity, код рушія не підлягає редагуванню, через це актуальним стає необхідність проведення оптимізації на більш високому рівні.

1.2 Постановка задачі дослідження

В магістерському дослідженні ми виділили наступні задачі, які будуть розглядатись:

- аналіз та порівняння актуальних інструментів для розробки систем симуляції часток для ігрової індустрії;
- порівняння ігрової індустрії та кіноіндустрії з точки зору створення та використання VFX;
- огляд теоретичних основ часток та їх класифікація за різними критеріями;
- аналіз існуючих методів спрощення та оптимізації часток для портативних ігрових пристроїв, таких як зменшення кількості часток, зміна їх форми та розміру, використання текстур або шейдерів, адаптація до руху камери та інші;
- вибір програмного забезпечення для моделювання та візуалізації часток з використанням обраних методів спрощення та оптимізації;
- висновки про переваги та недоліки розглянутих методів спрощення та оптимізації часток для портативних ігрових пристроїв;

- проведення експериментів та аналіз отриманих результатів;
- надання рекомендацій оптимізаційного характеру на основі проведених експериментів;
- перевірка оптимізаційних рішень;
- висновки щодо успішності проведених оптимізаційних дій.

Цього блоку питань достатньо для огляду індустрії, інструментів розробки VFX [2] та проблем, з якими знадобиться зіштовхнутись. Також отриманні висновки допоможуть скорегувати процес аналізу результатів та методи, якими будуть вирішатись проблеми з оптимізацією систем симуляції часток та провести цю оптимізацію з перевіркою результатів.

1.3 Засоби для проведення досліджень

Для проведення досліджень будуть використовуватись перелічені у вступі ігрові движки. У ході цього дослідження ми проаналізуємо існуючі сучасні движки, переглянемо їх плюси та мінуси, а після цього виберемо той, що найкраще нам підходить.

Так, як ми орієнтуємось під портативні пристрої, то найважливішим параметром цих движків буде можливість збірки проектів під мобільні пристрої з використанням сучасних бібліотек.

Аналіз оптимізації під консолі є доволі складним, бо потребує наявності так званих «девкітів» - консолей, надісланих їх власниками саме для розробки. Для процесу навчання таке зробити неможливо. Також залишаються персональні комп’ютери, але варіативність їх апаратного забезпечення дуже велика, через що неможливо адекватно оцінювати результати оптимізації, бо відеокарти різних поколінь мають фундаментальні відмінності.

Саме цьому для тестування проміжних результатів для пришвидшення процесу аналізу буде використовуватись персональний комп’ютер для отримання відносних результатів та формування методів оптимізації. Подальше тестування буде вже проводитись на сучасному смартфоні, бо ця платформа є в наявності та

відповідає апаратним обмеженням, які можуть відобразити результативність знайдених методів оптимізації.

2 ТЕОРЕТИЧНІ ОСНОВИ

2.1 Огляд та порівняння сучасних ігрових движків

Вибір правильного ігрового рушія, безумовно, є найважливішим рішенням, яке повинен прийняти розробник ігор, щоб отримати найкращі результати від свого продукту. Перша загадка, яку розробник гри повинен розгадати під час створення гри, полягає в тому, який ігровий движок він повинен використовувати, щоб отримати найкращий користувацький досвід. Ігровий движок допомагає створювати не тільки класичні аркадні ігри [3], такі як Ping, Tetris, Snake, але й інноваційні та передові ігри рівнів, такі як GTA та Assassin's Creed.

Будь-який розробник ігор чув про деякі неймовірні ігрові движки, такі як Unreal Engine, CryEngine і Unity3d. Це одні з найбільш поширених і провідних ігрових рушіїв для створення ігор просунутого рівня на сьогоднішній день. Кожен із цих 3 ігрових рушіїв має свої якості та потенціал. Ми повинні чітко усвідомлювати характер нашого проекту, наприклад, бюджет ліцензії, ігрову платформу, розміри (2D або 3D) і так далі. Отже, ваша мета повинна бути чіткою, перш ніж вибрати найкращий ігровий движок 2023 року для нашого проекту.

Unity має велику кількість учасників у своїй спільноті, які можуть допомогти нам із проєктом одразу. Ще однією особливістю, яка виділяє його як один із найкращих 3D-ігрових движків 2023 року, є його здатність підтримувати низку форматів файлів, що використовуються в провідних 3D-програмах, включаючи 3D Max, Blender, CINEMA, Maya, Softimage та багато інших. Крім того, розробники ігор можуть мати доступ до використання понад 15 000 безкоштовних та платних 3D-моделей, аудіо, анімації, розширень редактора, матеріалів, сценаріїв і шейдерів під час розробки ігор.

Unity 3D використовує C# або JavaScript, що є більш бажаним, ніж C++, оскільки це не дає проблем для переходу з Java на C# порівняно з C++. Тим не менш, Unity 3D має простий і швидкий інтерфейс і досить легкий, щоб працювати навіть на Windows XP з пакетом оновлень 2 (SP2).

Другим у списку є Unreal Engine 4, який є останнім рушієм, випущеним однією з найбільших американських компаній з розробки відеоігор та

програмного забезпечення Epic Games. Unreal Engine 4 успадкований Unreal Development Kit, широко відомим як UDK у світі ігор.

Unreal Engine 4 пропонує неймовірну графіку, яка надає ігровому досвіду реалістичного відтінку завдяки таким функціям, як просунуте динамічне освітлення. Що робить цей ігровий рушій ще більш дивовижним, так це його нова система частинок, яка має здатність обробляти до мільйона частинок в одній сцені.

Крім того, UE4 є абсолютно безкоштовним у використанні, однак вам потрібно заплатити роялті в розмірі 5% від грошей, які ви заробляєте на своїх іграх, розроблених Unreal Engine 4. У двох словах, Epic Games отримає 5% від усього, що ви заробляєте, будь то покупки в додатку, внутрішньоігрова реклама або гроші, які ви стягуєте з користувачів за покупку вашої гри. Однак творці Unreal Engine 4 дозволяють розробникам безкоштовно використовувати його повну версію, якщо дохід, який ви отримуєте від гри, становить до 3,000 доларів США на квартал.

Крім того, в Unreal Engine 4 використовується технологія Blueprint Visual Scripting, яка дозволяє створювати ігри за допомогою Blueprint. Однак такі ігри мають деякі обмеження. Крім усього, погане видовище цього движка полягає в тому, що він не здатний розробляти ігри для консолей минулого покоління.

Останній, але, безумовно, не менш важливий у нашому списку найкращих ігрових движків 2023 року – це CryEngine. Вперше представлений великою корпорацією розробників Crytek у першій грі Far Cry, CryEngine, безсумнівно, є одним із найпотужніших і домінуючих ігрових рушіїв, які ми маємо сьогодні. Що робить CryEngine гідним списку, так це його графічні можливості, які затьмарюють графічні можливості Unity і еквівалентні тому, що має Unreal. Незважаючи на те, що CryEngine є важким і потужним ігровим рушієм, користувачу потрібно трохи часу, щоб мати можливість ефективно використовувати цю платформу, і трохи важче зрозуміти новачкам, які раніше не використовували інші ігрові рушії.

CryEngine підтримує віртуальну реальність і має дивовижні візуальні ефекти, включаючи об'ємну систему візуалізації туману та хмар, яка надає хмарам повну 3D-просторову візуалізацію та реалістичну візуалізацію для ефектів туману та погоди. Крім того, найкраща частина вибору платформи CryEngine для розробки ігор полягає в тому, що вона не вимагає від своїх користувачів сплати роялті під час розробки гри. Однак, щоб отримати доступ до CryEngine, вам потрібно платити фіксовану суму, а саме $9.90 /- на місяць. Крім того, у CryEngine є спеціальний форум запитань і відповідей під назвою CryEngine Answers, який розвіює всі ваші сумніви та запити та допомагає вам отримати найкращий досвід.

Порівнюючи Unreal, Unity та CryEngine, ми натрапили на найкращі функції, які надають нам ці ігрові рушії. Порівнюючи продуктивність Unity та Unreal, ми зрозуміли, що Unity є кращою платформою для розробки мобільних та 2D/3D-ігор, тоді як Unreal найкраще підходить для розробки високографічних та фотореалістичних ігор. У цьому виявляється велика різниця між Unreal та Unity.

CryEngine, з іншого боку, також має можливості для створення ігор з високою графікою. Крім того, порівнюючи CryEngine з Unreal 2023 за шкалою надання функцій платформи наступного покоління з більш привабливою ціновою моделлю, CryEngine, безсумнівно, перевершує Unreal своєю економічно ефективною структурою. Однак для новачка, коли справа доходить до CryEngine 5 проти Unreal Engine 4 на основі легкості та мінімалізму, незважаючи на приголомшливі функції, які має CryEngine, Unreal Engine 4 та Unity, безсумнівно, варто спробувати.

Наприкінці цього аналізу ми прийшли до висновку, що найбільш універсальна та якісна система симуляції часток реалізована в Unreal Engine. До того ж, обидві системи цього движка мають розвинуті інструменти оптимізації, а не тільки створення цих часток. Саме цьому подальший аналіз треба провести саме по цьому движку.

2.2 Аналіз індустрії та місця, яке займає в ній Unreal Engine 4-5

Наприкінці 90-х років відбувся монументальний розвиток персональних комп'ютерів та ігор, у які можна було грати на них. Графіка швидко вдосконалювалася, і хоча за сучасними мірками це смішно, розвиток 3D-графіки від першої особи зробив відеоігри набагато більш захоплюючими. Інтернет швидко поширювався на споживачів, що полегшувало участь в онлайн-іграх, приєднуватися до дискусій на ігрових форумах і навіть вчитися програмувати ігри самостійно. А ігри, які виходили в той період, – це назви, про які ми досі говоримо з благоговінням: Doom, Sim City, Duke Nukem, Half-Life, StarCraft, Myst та багато інших (див. рис. 2.1).

Тим не менш, Unreal Tournament був у власній лізі: швидкий темп, великі карти та комп'ютерні супротивники зі штучним інтелектом, які вперше насправді здавалися розумними. Навколо гри почала зростати спільнота гравців, які модифікували гру, робили її власною та ділилися своїми твореннями зі спільнотою. Все це сталося завдяки компанії з розробки ігор з надзвичайно світлим майбутнім і створеному ними програмному забезпеченню під назвою «Unreal Engine».

Рисунок 2.1 – Doom

Epic Games була заснована в 1991 році і є однією з небагатьох історій успіху компанії-розробника програмного забезпечення, які заслужили їй гідне місце в ігровій історії. Їхні перші кілька ігор мали комерційний успіх, але те, що дійсно вивело їх на карту, з'явилося наприкінці тисячоліття, коли вони випустили Unreal [4] у 1998 році, а наступного року – продовження, Unreal Tournament.

Геймплей, графіка та можливості гри відрізняють її від безлічі інших шутерів від першої особи на ринку. Але крім самої гри, Epic Games прийняла рішення, яке привело гру до статусу легендарної. Вони вирішили постачати гру з тим самим інструментом – Unreal Editor – який використовується для створення рівнів та ігрового процесу, що дозволяє гравцям налаштовувати гру та робити її своєю. Це один з багатьох ігрових рушіїв, розроблених протягом багатьох років, але він далеко не найуспішніший і широко використовуваний.

«Якщо ви думаєте про технологію, яка витримала випробування часом, Unreal Engine є одним із найбільших», – каже Джейкоб Фельдман, інженер програмного забезпечення та рішень для рендерингу в CoreWeave, який підробляє авторизованим інструктором Unreal Engine. «Він має понад 20 років коду, тому це значний, складний продукт».

Зрозумівши, що у них є переможець з Unreal Engine, команда Epic почала ліцензувати програмне забезпечення для інших розробників, щоб створювати свої ігри на його основі. Перенесемося в 2017 рік, коли студія випустила Fortnite, культурну сенсацію, яка ще більше зміцнить спадщину Epic і Unreal Engine.

Зовсім недавно Unreal Engine відіграє важливу роль в іншому культурному моменті, який просунув цей інструмент у абсолютно нову індустрію – кіновиробництво [5] (див. рис. 2.2).

У 2019 році Disney випустила відзначений нагородами серіал «Мандалорець» як частину своєї лінійки для свого нового потокового сервісу Disney+. Шоу миттєво стало хітом і продовжило довгу історію постановок «Зоряних війн», які розширювали технологічні межі кіноіндустрії. «Мандалорець» продемонстрував величезний стрибок вперед у дизайні декорацій

завдяки використанню рендерингу декорацій у реальному часі за допомогою Unreal Engine, що фактично зробило зелений екран застарілим.

Рисунок 2.2 – Використання UE4 в кіно

«Unreal зробила стрибок у ширші індустрії, такі як кіно, після виходу Fortnite», – каже Фельдман. «Стало ясно, що ігровий рушій, такий як Unreal, який призначений для гнучкості, продуктивності та візуальної якості, має більше застосувань, ніж просто ігри. Це стало особливо помітно, коли більш потужне обладнання почало полегшувати такі речі, як трасування променів у реальному часі та інші візуальні ефекти, які були просто неможливі 5-10 років тому. Індустрія візуальних ефектів висуває одні з найсуворіших вимог серед усіх візуальних засобів, тому Еріс взяла на себе серйозне зобов'язання підтримувати ті види інструментів, які потрібні VFX-студіям».

Кінематографісти завжди знаходили способи обдурити очі глядачів, змусивши їх думати, що вони бачать щось більше, ніж те, що є насправді – те, що в індустрії називається «розширенням декорацій». На зорі Голлівуду це включало буквальне малювання сцен, які потім плавно інтегрувалися в декорації, щоб створити ефект нескінченного простору.

Пізніше технологія зеленого екрану стала кращим методом для створення великих світів. Однак цей метод викликав деякі суттєві проблеми. Акторам і знімальній групі на знімальному майданчику може бути важко показати переконливу гру на порожньому яскраво-зеленому тлі. Актори не бачать

навколишній світ (див. рис. 2.3), а отже, не реагують на нього так, щоб переконати глядача.

Рисунок 2.3 – Зйомка кіно

Але що, якби існував спосіб створювати великі світи за командою, які актори могли б бачити прямо перед собою? Це було б майже те саме, що закинути цих акторів у відеогру і змусити їх виступати в цьому світі, з усіма видами, доступними їхнім власним очам.

Творець «Мандалорця» Джон Фавро вирушив до команди Epic Games, щоб перевірити, чи можуть вони допомогти йому зробити це, по суті, зануривши своїх акторів у відеогру. Використовуючи масивні світлодіодні екрани високої чіткості, що оточують свій знімальний майданчик, команда «Мандалорця» взяла стару голлівудську концепцію розфарбованих декорацій і перенесла її в двадцять перше століття. Тепер ці намальовані фони були живим, дихаючим середовищем з динамічним освітленням і рухом. Актори могли їх бачити і реагувати на них. Фізичні елементи на передньому плані органічно вписуються у віртуальний фон.

Але стає краще – використовуючи технологію ігрового рушія, вони можуть прив'язати рух камери до фону (див. рис. 2.4), дозволяючи йому змінюватися

разом із камерою, щоб ідеально відповідати рухам і забезпечувати плавний рух паралакса, який неможливо відрізнити від реального знімка.

Рисунок 2.4 – Процес зйомки «Мандалорця»

По суті, вони створили графіку в реальному часі, що лежить в основі кіновиробництва. На етапі постпродакшену не потрібно прибирати, ці ефекти виконуються в камері в режимі реального часу.

«Це величезна допомога для акторів і знімальної групи, щоб вони могли побачити своє оточення в даний момент», – каже Фельдман. «Актор сьогодні повинен уявляти на зеленому екрані, що відбувається навколо. Цей прогрес дозволяє їм це побачити».

Окрім полегшення життя акторів, середовище в реальному часі також робить роботу команди постпродакшну нескінченно менш стресовою. На зеленому екрані рефлексії – це постійна боротьба. Будь-який блискучий матеріал буде відображати зелене середовище і зіпсувати ефект VFX. Ці відблиски потрібно видалити, а потім додати віртуальний світ до крихітних відображень в окулярах, шоломах або шматочках металу в навколишньому середовищі. Навколишнє середовище в реальному часі виключає все це з рівняння, оскільки відображення, зняті в камеру, відображають світ таким, яким він має бути в історії.

Epic Games зняла обкладинки свого останнього ігрового движка Unreal Engine 5 і значною мірою зламала частину інтернету, яка цікавиться іграми.

Працюючи на версії обладнання для розробників PlayStation 5, результати, завантажені на Vimeo, спочатку виглядали досить приголомшливо, але коли ви зрозуміли, що це кадри ігрового процесу в реальному часі, вони справді вразили.

Візуальна якість практично не має собі рівних ні в чому, крім поточних високоякісних попередньо оброблених візуальних ефектів і демонструє неймовірний рівень деталізації поряд із справді фотореалістичним освітленням [6]. І, звичайно ж, що ще краще, так це те, що новини про UE5 також висвітлили ті частини Інтернету, які мають справу з графікою трансляції, кіноефектами, віртуальними студіями та майже скрізь, де йдеться про високоякісні візуальні ефекти.

У той час як інші графічні движки, такі як Unity, звичайно, доступні, Epic провела акуратний трюк, поєднавши провідний набір функцій з простотою використання та бізнес-моделлю, яка заохочує використання UE в іншому програмному забезпеченні. Наприклад, будь-який розробник ігор, який використовує Unreal у комерційних цілях, не сплачує жодних ліцензійних зборів, доки валовий дохід від програмного забезпечення не досягне бар'єру в 1 мільйон доларів США (нещодавно піднятий з 50 тисяч доларів), і це допомогло широко інтегрувати його в цілу низку технологій. Для розробників, які не є розробниками ігор, це 100% роялті-фрі.

Його використання в графічному стеку аналогічно тому, як підключення до IT-індустрії в цілому прискорило розвиток у секторі мовлення; Це дозволяє порівняно невеликій галузі використовувати зусилля з розвитку набагато більшої, і швидкість змін, яку ми спостерігаємо в результаті, вражає.

«Коли вийде новий Unreal Engine 5, ви не зможете сказати, що справжнє, а що ні» (див. рис. 2.5), – коментує Філ Вентре, віце-президент зі спорту та трансляції Ncam Technologies. «Інтеграція ігрових рушіїв демократизує те, як компанії використовують AR і VR, і це буде не просто технологією для мовників першого рівня в майбутньому».

В той час як нова демо-версія була частково створена, щоб продемонструвати деякі з дуже розумних нових технологій майбутньої PS5, таких

як твердотільний накопичувач M.2, який буквально змінює правила гри, UE5 демонструє деякі нові технології, які кардинально змінюють ворота для роботи CG в реальному часі. Особливо варто згадати два: Nanite і Lumen.

Рисунок 2.5 – Демо-сцена на UE5

Nanite – це нова віртуалізована геометрія [7] мікрополігонів, яка, по суті, дозволяє художникам створювати стільки геометричних деталей, скільки вони хочуть. Він транслюється та масштабується в режимі реального часу, що важливо враховувати, коли ви плануєте двигун, який працюватиме на всьому, аж до смартфона.

Крім того, є Lumen. Це повністю динамічна система глобального освітлення, яка миттєво реагує на зміну сцени та освітлення, і є частиною секрету того, як зробити ігрову графіку такою гарною під час роботи в консолі. Він здатний відтворювати дифузне взаємовідбиття з нескінченними відскоками та непрямими дзеркальними відображеннями в тому, що Еріс називає «величезними, деталізованими середовищами в масштабах від кілометрів до міліметрів». Він також адаптується: проб'єте дірку в стіні, і сцена зміниться, щоб пристосуватися до світла, що надходить через отвір.

«Можливість освітлювати та рендерити сотні мільйонів полігонів у режимі реального часу – це квантовий зсув, який змінить рівень взаємодії

кінематографістів із зображеннями, які вони створюють», – каже Майлз Перкінс, менеджер із розвитку бізнесу Epic Games. «Ці нові технології дозволять творчим людям побачити всю повноту свого бачення без необхідності відокремлювати різні частини своїх знімків, переглядаючи анімацію окремо від освітлення, окремо від оточення та ефектів. Все буде прямо перед ними, повністю кероване. Кінематографісти зможуть компонувати та освітлювати кадри в режимі реального часу, незалежно від того, чи є вони фізичними, віртуальними чи поєднують обидва».

Можливо, одним із ключових моментів тут є те, що Unreal Engine, який зараз працює на версії 4.25, вже дуже хороший.

«В даний час ми активно використовуємо Unreal Engine 4 як у попередньому, так і у віртуальному виробництві», – каже Х'ю Макдональд, директор з технологічних інновацій Nviz. «Для попереднього, природа UE4 в реальному часі означає, що ми можемо отримати неймовірно якісні зображення з мінімальним часом рендерингу. Ми також використовуємо його для віртуального виробництва, забезпечуючи кращу інтеграцію, ніж ми могли б це зробити історично»

Nviz використовує Unreal у двох основних інструментах, які є гарною ілюстрацією того, як він використовується для попередньої роботи в галузі та куди він може рухатися. Система віртуальних камер дозволяє віртуально розвідувати попереднє середовище та дозволяє режисерам і кінематографістам отримати практичний досвід роботи з камерою, тоді як набір інструментів симулькама тісно інтегрований у Unreal і забезпечує попередній перегляд знімальної групи на знімальному майданчику того, як виглядатиме кадр після додавання візуальних ефектів у пост.

«Unreal дозволяє нам гарантувати, що це і гнучко, і якісно, – каже Макдональд. Виходячи з того, що ми знаємо на даний момент про UE5, основні стрибки будуть пов'язані з деталями геометрії, оскільки ресурси з набагато вищою роздільною здатністю можна буде використовувати та транслювати. Це набагато краще поєднується з робочим процесом плівкового VFX, оскільки є надія, що

ресурси не потребуватимуть такої великої обробки, щоб зробити їх готовими до роботи з двигуном. Повністю динамічне освітлення, яке є люменом, означатиме, що для отримання того ж результату буде менше потреби в запіканні освітлення. Це дозволить нам зберегти сцени повністю динамічними, дозволяючи нам регулювати освітлення в реальному часі під час виробництва, якщо це необхідно, що часто просять на знімальному майданчику, оскільки фізичне освітлення змінюється залежно від кадру».

Нова творчість Поряд зі збільшенням якості, яке Ventre з NCam порівнює зі стрибком від UE3 до UE4, UE5 відкриває спокусливу перспективу впровадження як нових способів роботи, так і нових способів творчого дослідження віртуальних просторів.

«Unreal Engine стане великою частиною майбутнього кінематографа», – каже Сем Мір із CVP. «Це повертає можливість отримувати практичні ефекти в камері, будь то інтерактивне освітлення актора або динамічна зміна фону в реальному часі, навіть якщо вони були створені у віртуальному просторі. Можливість отримати миттєвий зворотний зв'язок про те, як щось виглядатиме, є безцінною».

Це буде поповнювати набагато більше живих просторів, ніж до початку. Макдональд згадує сектор театру та подій, де відеоекрани стали частиною взаємодії з освітленням для створення абсолютно нових живих видовищ, тоді як на віртуальних декораціях відбудеться подальший стрибок якості, щоб зробити їх настільки невідмінними від реальних, що лише жива аудиторія схилить рішення про використання фізичної декорації. Навіть ці глядачі можуть побачити зовсім інші шоу, зі змішаними елементами доповненої реальності, які плавно використовуються у фінальному ТХ. У постковідні часи ви абсолютно зможете зібрати трьох гостей на дивані для чат-шоу, не виходячи з дому, і ніхто не буде ні краплі мудрішим.

І, звичайно, є спосіб, яким це може прискорити розвиток живих зйомок на світлодіодних екранах, як це було започатковано такими шоу, як «Мандалорець».

«Незважаючи на те, що Unreal використовувався на світлодіодних екранах під час зйомок кілька разів, нові оновлення, сподіваюся, дозволять просунутися набагато далі і отримати більшу частку готових знімків безпосередньо з камери», – захоплюється Макдональд.

3 АНАЛІЗ СИСТЕМ СИМУЛЯЦІЇ ЧАСТОК

3.1 Системи симуляції часток, наявні в UE5

Cascade та Niagara [8] – це дві системи частинок, які використовуються в Unreal Engine для створення ефектів, таких як вогонь, дим, дощ, сніг та іскри. Cascade – це старіша система, яка була введена ще в Unreal Engine 3, тоді як Niagara – це новіша система, яка була додана в Unreal Engine 4.25. Обидві системи мають свої переваги та недоліки, які будуть розглянуті нижче.

Cascade базується на концепції емітерів та модулів. Емітер - це об'єкт, який генерує частинки за певними правилами, а модуль - це компонент, який змінює властивості частинок, такі як колір, розмір, швидкість, обертання тощо. Cascade дозволяє створювати складні ефекти з декількох емітерів та модулів, а також налаштовувати їх за допомогою графічного інтерфейсу. Cascade також підтримує функції, такі як GPU-прискорення, колізії частинок, взаємодію з освітленням та тіньованням, анімацію спрайтів та інше.

Niagara базується на концепції систем та емітерів. Система - це об'єкт, який управляє одним або декількома емітерами та їхньою логікою. Емітер - це об'єкт, який генерує частинки за певними правилами, але в Niagara ці правила можуть бути задані за допомогою графових редакторів або скриптованих мов. Niagara дозволяє створювати більш гнучкі та динамічні ефекти з різноманітними типами частинок, такими як меш-частинки, лайн-частинки, рибон-частинки тощо.

Порівняння Cascade та Niagara можна побачити у наступному вигляді:

- Cascade простий у використанні та має багато налаштувань за замовчуванням для швидкого створення ефектів.
- Cascade має обмеження щодо кількості частинок та їхнього типу.
- Cascade не дозволяє модифікувати логіку генерації частинок або створювати власні модулі.
- Niagara складний у використанні та вимагає більше знань про програмування та математику для створення ефектів.
- Niagara дозволяє створювати безліч частинок різних типів та форм.

– Niagara дозволяє модифікувати логіку генерації частинок або створювати власні функції та модулі.
– Niagara покращує продуктивність та якість ефектів за рахунок оптимізації GPU та симуляції рідини.

Niagara також покращує продуктивність та якість ефектів за рахунок оптимізації GPU, симуляції рідини, взаємодії з функціями двигуна, такими як Niagara Volumetric Fog та Niagara Skeletal Mesh Sampling.

3.2 Cascade

Основною і всеосяжною концепцією каскаду є концепція модульно спроектованих систем частинок. У деяких пакетах 3D-ефектів, таких як Maya, створюється система частинок з більшістю необхідних властивостей для поведінки. Потім користувач змінює ці властивості, щоб досягти бажаного результату.

У Каскаді, з іншого боку, система частинок починається лише з кількох властивостей голих кісток і кількох поведінкових модулів.

Кожен модуль представляє певний аспект поведінки частинок і містить лише властивості, які керують цією поведінкою, такі як колір, положення при народженні, рух, масштабування та багато інших. Потім користувач може додавати або видаляти модулі на свій розсуд, тим самим додатково визначаючи поведінку частинок. Так як додаються тільки модулі для необхідної поведінки, то немає стороннього обчислення непотрібних властивостей.

Найкраще те, що модулі можна легко додавати, видаляти, копіювати і навіть приміряти з випромінювачів у системі частинок, що робить складні налаштування дуже легкими для досягнення після ознайомлення користувача з доступними модулями.

Деякі модулі за замовчуванням існують на випромінювачі частинок. Кожного разу, коли новий спрайт-випромінювач [9] – ключовий компонент будь-якої системи частинок – додається до системи частинок, він завжди створюється з наступними модулями за замовчуванням:

- Required – містить різноманітні властивості, абсолютно необхідні для системи частинок, такі як матеріал, нанесений на частинки, як довго випромінювач повинен випромінювати частинки та багато інших;
- Spawn – цей модуль контролює, як швидко частинки будуть з'являтися з випромінювача, чи будуть вони з'являтися сплесками, і будь-які властивості, пов'язані з часом народження частинок;
- Lifetime – цей показник визначає, як довго кожна частинка буде жити після свого народження. Без цього модуля частинки будуть жити нескінченно довго;
- Initial Size – керує масштабом частинки в момент її народження;
- Initial Velocity – керує рухом частинки в момент її народження;
- Color Over Life – цей модуль контролює, як колір кожної частинки буде змінюватися протягом її життя.

Модулі Required і Spawn є постійними і не можуть бути видалені з випромінювача. Однак всі інші модулі можна зняти за бажанням.

Системи частинок також дуже тісно пов'язані з різними матеріалами і текстурами, нанесеними на кожну частинку. Основне завдання самої системи частинок полягає в управлінні поведінкою частинок, в той час як специфічний вигляд і відчуття системи частинок в цілому часто контролюється за допомогою матеріалів.

Існує багато модулів, які можна додати до випромінювачів частинок. Щоб уникнути плутанини, ці модулі розбиті на різні категорії. До таких категорій належать ті, що наведені в таблиці 1.

Таблиця 3.1 – Категорії модулів систем випромінювання часток

Категорія	Опис
Acceleration	Модулі, що регулюють те, як на прискорення частинок може впливати, наприклад, сили опору.

Продовження таблиці 3.1

Attraction	Модулі, які керують рухом частинок, притягуючи частинки до різних точок простору.
Camera	Модулі, що керують переміщенням частинок у просторі камери, дозволяючи користувачеві змушувати їх здаватися ближчими або віддаленими від камери.
Категорія	Опис
Collision	Модулі, що керують тим, як обробляються зіткнення між частинками та геометрією.
Color	Модулі, які впливають на колір частинок.
Event	Модулі, які керують активацією подій частинок, які, у свою чергу, можуть викликати різноманітні внутрішньоігрові реакції.
Kill	Модулі, що регулюють видалення частинок.
Lifetime	Модулі, які контролюють, як довго повинні жити частинки.
Light	Модулі, що регулюють світло частинок.
Location	Модулі, що контролюють місце народження частинок щодо розташування випромінювача Actor.
Material	Модулі, які керують матеріалом, нанесеним на самі частинки.
Orbit	Модулі, які забезпечують орбітальну поведінку екранного простору, щоб додати додатковий рух ефектам.
Orientation	Модулі, що дозволяють фіксувати вісь обертання частинок.
Parameter	Модулі, які можуть бути параметризовані або керовані за допомогою зовнішніх джерел, таких як креслення та ранки.

Кінець таблиці 3.1

Rotation	Модулі, які керують обертанням частинок.
RotationRate	Модулі, що регулюють зміну швидкості обертання, наприклад, віджимання.
Категорія	Опис
Size	Модулі, що контролюють масштаб частинок.
Spawn	Модулі для додавання спеціалізованих швидкостей появи частинок, таких як породжування частинок на основі відстані, що переміщується.
SubUV	Модулі, які дозволяють відображати анімовані спрайтові аркуші на частинці.
Velocity	Модулі, які контролюють швидкість кожної частинки.

Дві важливі концепції, про які слід пам'ятати при роботі з модулями частинок, - це початкові та життєві властивості або властивості на життя. Початкові модулі використовуються для управління деякими аспектами частинок в той момент, коли вони з'являються на світ. Модулі Over Life або Per Life існують для того, щоб дозволити деякому аспекту частинки змінюватися протягом її життя.

Наприклад, модуль «Початковий колір» дозволить вам встановити колір модуля в момент народження частинки, в той час як властивість «Колір протягом життя» дозволяє кольору частинки поступово змінюватися між моментом народження і моментом смерті.

Якщо ви змінюєте властивість на тип розподілу, який змінюється з часом, деякі модулі використовують «відносний час», а деякі – «абсолютний час»

Абсолютний час [10] - це, по суті, час, що містить випромінювач. Якщо у вас є випромінювач, налаштований на 3 цикли по 2 секунди, Абсолютний час для модулів у цьому випромінювачі тричі пройде від 0 до 2 секунд.

Відносний час становить від 0 до 1, що вказує на час життя кожної частинки.

Коли ви працюєте з Каскадом над створенням власних ефектів частинок, важливо пам'ятати про те, як кожен з об'єктів пов'язаний один з одним. У цьому документі ми вже обговорювали поняття модулів, але вони складають лише один компонент повного ефекту частинок. Загалом, компонентами системи частинок є модулі, випромінювачі, системи частинок і актори-випромінювачі.

Подібно до того, як існує велика різноманітність типів ефектів, які ви захочете створити за допомогою своїх частинок, існує також безліч типів випромінювачів, які допоможуть вам створити саме те, що вам потрібно. Нижче наведено список доступних типів випромінювачів:

Слід зазначити, що всі випромінювачі, незалежно від типу, є спрайтовими випромінювачами на старті. Потім до випромінювача додаються різні модулі Emitter Type Data, щоб змінити його тип на щось інше.

Не всі аспекти системи частинок можуть бути визначені заздалегідь. Іноді певні частини поведінки системи частинок потрібно контролювати або змінювати під час виконання. Наприклад, ви можете створювати магічний ефект, який змінює колір залежно від кількості магічної енергії, спожитої під час застосування заклинання. У таких випадках до системи частинок потрібно буде додати параметри.

Параметр – це тип властивості, який може надсилати або отримувати дані до/з інших систем, таких як Blueprints, Matinee, матеріал або багато інших джерел. У Каскаді параметру може бути присвоєно майже будь-яку задану властивість, що означає, що властивістю можна керувати ззовні системи частинок.

Наприклад, налаштування параметра для контролю швидкості створення для ефекту вогню може дозволити гравцеві збільшувати або зменшувати кількість полум'я, що випромінюється під час роботи.

І навпаки, існують модулі параметрів, які можуть бути додані до системи частинок, які, в свою чергу, можуть керувати іншими речами на рівні, такими як колір, що використовується в даному матеріалі.

Розрахунки систем частинок можуть легко стати дуже дорогими. Навіть при використанні частинок графічного процесора, які перевантажують значну частину обчислення частинок на графічний процесор, важливо враховувати значення обчислення частинок, від яких гравець знаходиться занадто далеко, щоб побачити або адекватно оцінити.

Наприклад, розглянемо систему частинок вогнища. Якщо подивитися зблизька, можна побачити, як вугілля та іскри здіймаються в дим. Однак, якщо дивитися на них з відстані кількох сотень метрів, ці вуглинки будуть настільки маленькими, що монітор або екран навіть не зможуть їх відтворити. Так навіщо ж їх обчислювати?

Саме тут у гру вступають рівні деталізації (LOD). Система LOD [7] дозволяє встановити власні діапазони відстаней, на яких ваша система частинок буде автоматично спрощуватися. Кожен діапазон становить інший LOD. Спрощення відбувається у вигляді знижених значень властивостей, деактивованих модулів або, можливо, навіть деактивованих випромінювачів. Наприклад, у наведеному вище прикладі з багаттям було б ідеально повністю деактивувати випромінювач, який додавав іскри до загального ефекту, коли гравець був занадто далеко, щоб їх побачити.

Ваша система частинок може мати стільки LOD, скільки вам потрібно, і ви можете вручну керувати діапазонами для кожного з них.

Дистрибутиви – це набір типів даних для обробки даних спеціалізованими способами, такими як використання діапазону для значення або інтерполяція значення вздовж кривої. Кожного разу, коли ваша система частинок вимагає рандомізації або здатності до зміни якогось аспекту частинки з часом, ви будете використовувати розподіл для контролю цієї властивості.

Багато властивостей, знайдених у модулях Cascade, можуть мати різні розподіли, застосовані до них. Потім фактичне значення цієї властивості встановлюється в розподілі.

3.3 Niagara

Навіщо заново винаходити візуальні ефекти для Unreal Engine? Unreal Engine розширює свою базу користувачів і зараз використовується в багатьох галузях за межами простору розробки ігор.

Користувачі Unreal Engine різноманітніші, ніж будь-коли раніше: від студентів-дизайнерів до невеликих інді-розробників, великих професійних студійних команд, приватних осіб і компаній за межами ігрової індустрії. У міру того, як вони рухаються вперед, розробники Epic не знатимуть усього про кожну індустрію, яка використовує Unreal Engine. Вони хотіли створити систему візуальних ефектів (VFX), яка б працювала для всіх користувачів у різних галузях.

Вони хотіли створити нову систему, яка б надала всім користувачам гнучкість для створення потрібних їм ефектів. Їх цілями для нової VFX-системи були:

- передача повного контролю в руки художників;
- можливість програмувати та налаштовувати на кожній осі;
- кращі інструменти для налагодження, візуалізації та продуктивності;
- підтримка даних з інших частин Unreal Engine або із зовнішніх джерел.

Тотальний контроль користувача починається з доступу до даних. Epic Games хочуть, щоб користувач міг використовувати будь-які дані з будь-якої частини Unreal Engine, а також використовувати дані з інших додатків. Тому вони вирішили надати все користувачеві.

Для того, щоб надати всі ці дані користувачеві, вони повинні встановити, як хтось може використовувати ці дані. Простори імен надають контейнери для ієрархічних даних. Наприклад, Emitter.Age [12] містить дані для випромінювача; Particle.Position містить дані для частинки. Карта параметрів – це корисне

навантаження частинки, яке несе в собі всі атрибути частинки. В результаті цього все стає необов'язковим.

Будь-який тип даних може бути доданий як параметр частинки. Ми можемо додавати складні структури, трансформувати матриці або булеві прапорці. Ми можемо додати ці або будь-які інші типи даних і використовувати їх у симуляції ефектів.

Існують переваги як для парадигми стека (наприклад, та, що використовується в Cascade), так і для парадигми графа (така, яка використовується в кресленнях). Стеки забезпечують користувачам модульну поведінку та читабельність. Графіки дають користувачам більше контролю над поведінкою. Ця нова система ефектів поєднує в собі переваги обох цих парадигм.

Модулі працюють у графічній парадигмі – ми можемо створювати модулі з HLSL у редакторі сценаріїв за допомогою візуального вузлового графа. Модулі взаємодіють із загальними даними, інкапсулюють поведінку та складаються разом.

Випромінювачі працюють у парадигмі стека – вони служать контейнерами для модулів і можуть складатися разом для створення різних ефектів. Випромінювач є одноцільовим, але він також багаторазовий. З модулів параметри передаються на рівень емітера, але можна змінювати модулі і параметри в емітері.

Як і випромінювачі, системи працюють у стековій парадигмі, а також працюють з часовою шкалою секвенсера, яку ми можемо використовувати для керування поведінкою випромінювачів у системі. Система - це ємність для випромінювачів. Система об'єднує ці випромінювачі в один ефект. При редагуванні системи в Niagara Editor ми можемо змінювати і перевизначати будь-який параметр, модуль або випромінювач, який знаходиться в системі.

Симуляція частинок в Ніагарі концептуально працює як стек – симуляція перетікає з верхньої частини стека в нижню і виконує модулі по порядку. Важливо, що кожен модуль призначається групі, яка описує, коли модуль виконується. Наприклад, модулі, які ініціалізують частинки або діють, коли частинка з'являється, належать до групи «Відродження частинок».

Усередині кожної групи може бути кілька стадій, які називаються в певні моменти життєвого циклу системи. Випромінювачі, системи та частинки за замовчуванням мають стадії Spawn та Update. Стадії спавна викликаються у першому кадрі, де існує ця група. Наприклад, системи викликають свою стадію відродження, коли система вперше створюється на рівні та активується. Частинки викликають свою стадію Spawn щоразу, коли випромінювач випромінює частинку, і інструкції Spawn будуть виконуватися для кожної нової створеної частинки. Етапи оновлення викликаються в кожному кадрі, де активна система, випромінювач або частинка.

Існують також просунуті етапи, такі як події та етапи симуляції, які можна додати до потоку спавна та оновлення. Події викликаються щоразу, коли частинка генерує нову подію, і випромінювач налаштований на обробку цієї події. Там, де це можливо, етапи обробника подій відбуваються в тому самому кадрі, але після вихідної події. Етапи симуляції – це розширена функція графічного процесора. Ця функція дозволяє послідовно виконувати кілька етапів спавна та оновлення, а також корисна для побудови складних структур, таких як симуляції рідини.

Додаючи кожен модуль до етапу (Update, Spawn, Event або Simulation) у групі (System, Emitter або Particle), ми можемо контролювати, коли модуль виконується, а також які дані він обробляє. Групи стеків пов'язані з просторами імен, які визначають, які дані модулі в цій групі можуть читати або записувати.

Наприклад, модулі, які виконуються в системній групі, можуть зчитувати та записувати параметри в системному просторі імен, але можуть читати лише з параметрів, які належать до просторів імен рушія або користувача. У міру того, як виконання спускається вниз по стеку від System Group до Emitter Group, модулі, що виконуються в Emitter Group, можуть зчитувати і записувати параметри в Emitter Namespace, але можуть читати тільки з параметрів в просторах імен System, Engine і User Namespaces. Модулі в групі частинок можуть зчитувати лише з параметрів у просторах імен System та Emitter Names.

Оскільки модулі в групах Emitter можуть зчитувати параметри в System Namespace, моделювання, яке є актуальним для всіх випромінювачів, може бути

виконано один раз модулями в групі System, а результати цього (що зберігаються в System Namespace) можуть бути прочитані модулями групи Emitter в кожному окремому випромінювачі. Це продовжується модулями груп частинок, які можуть зчитувати параметри в просторах імен системи та випромінювача.

В інших своїх аспектах Niagara доволі схожа на Cascade. Ця система більш досконала, гнучка, має нескінченні можливості по розширенню функціоналу, хоча і через це більш складний процес створення часток. У цьому підрозділі були наведені основні відмінності від Cascade та Niagara та архітектурні особливості, так як процес автоматизованої оптимізації в цих системах концептуально відрізняється.

3.4 Оптимізація систем симуляції часток

Під час створення гри ми можемо мати велику варіативність робочого навантаження VFX залежно від композиції сцени. Іноді може знадобитися вжити заходів, які допоможуть керувати ефективністю, наприклад вибракувати екземпляри за межами певного діапазону або екземпляри, які перевищують указаний бюджет.

Ресурси «Тип ефекту» дають змогу один раз налаштувати різноманітні параметри, а потім застосувати їх до колекції систем Niagara.

В об'єкті «Тип ефекту» можна налаштувати кілька різних методів відбору систем, які перевищують ваш бюджет. Усі ці параметри доступні під заголовком Масштабування бюджету.

Максимальне глобальне використання бюджету (див. рис. 3.1): цей параметр дозволяє встановити бюджет, вищий за який вибраковується будь-яка система. Зазвичай для цього параметра встановлюється значення від 0 до 1, що відповідає відсоткам від 0 до 100%. Ви можете встановити його на 1.5, якщо хочете, щоб система була більш дозвільною. Це означає, що як тільки будь-яка система досягає цього відсотка від вашого бюджету, вона вибраковується. Це найкращий варіант, якщо ви віддаєте перевагу продуктивності, а не візуальним ефектам.

Шкала максимальної відстані за глобальним бюджетним використанням: цей параметр дозволяє налаштувати криву, щоб визначити, як відстань, на якій ви відбираєте системи, зменшується зі збільшенням використання бюджету. Наприклад, якщо ваш бюджет дуже високий, то Niagara рендерить лише ті, які знаходяться поблизу, а не ті, що далеко.

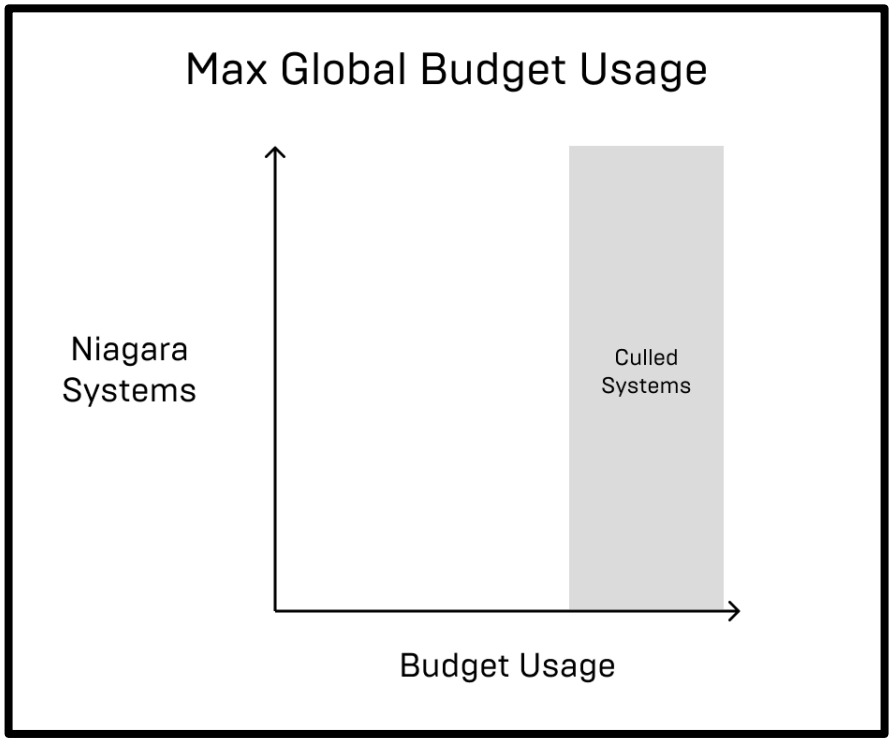

Рисунок 3.1 – Максимальне глобальне використання бюджету

Шкала максимальної кількості екземплярів за глобальним бюджетним використанням: цей параметр дозволяє налаштувати криву, яка визначає, як зі збільшенням використання бюджету зменшуватиметься кількість екземплярів на вашому рівні. Це зменшує масштаб усіх екземплярів усіх систем, які відповідають цьому типу ефекту.

Максимальна шкала кількості екземплярів системи за глобальним використанням бюджету: цей параметр дає змогу налаштувати криву (див. рис. 3.2), яка визначає, як зі збільшенням використання бюджету кількість екземплярів на вашому рівні зменшується. Втім, у цьому варіанті, замість того, щоб вибраковувати всі екземпляри у всіх системах, ви вибраковуєте певну кількість екземплярів для кожної системи.

Для кожного з цих 3 параметрів, які приймають значення Start X, Start Y, End X та End Y, ці значення визначають лінійно інтерпольовану криву. Все, що вище цієї кривої, буде вибракувано. Для прикладу дивіться наступну діаграму того, як буде виглядати крива.

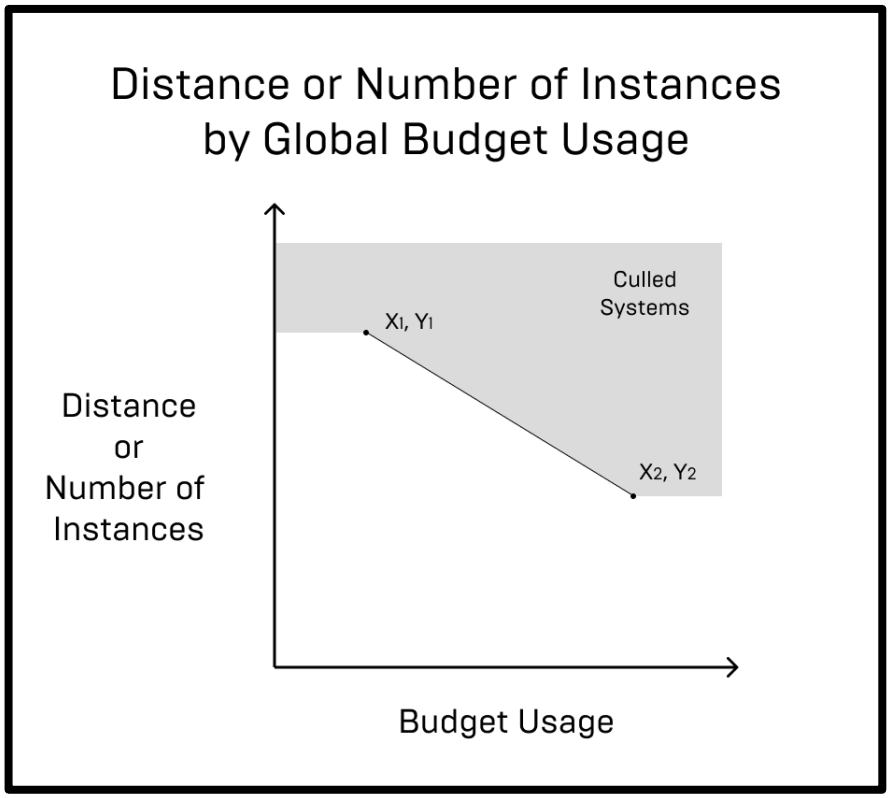

Рисунок 3.2 – Демонстрація роботи системи куллінгу

Кількість частинок дійсно відіграє незначну роль у загальній схемі речей, коли справа доходить до продуктивності. Незалежно від того, розділений екран чи ні, складність матеріалу та покриття екрана (overdraw) є вашими явними ворогами, коли справа доходить до загальних витрат на певну систему. Проста емісійна іскра з нічим іншим, як текстурою, помноженою на кольори вершин і підключеною до випромінювального входу в неосвітленому матеріалі, відповідає лише жменьці інструкцій. Ви можете створювати їх протягом усього дня натовпами, і ваш загальний вплив на продуктивність, ймовірно, буде дуже невеликим. Спрайти невеликі, що означає, що покриття екрану низьке, а складність матеріалу робить їх дешевими та швидкими для рендерингу. Кількість вершин насправді не є чимось, про що вам потрібно турбуватися в довгостроковій

перспективі, якщо тільки ви не досягаєте дійсно екстремальних чисел (сотні, тисячі або більше).

Набагато більший вплив на загальну продуктивність матиме кількість інструкцій щодо ваших матеріалів. Для таких матеріалів, як вогонь і дим, у вас, по суті, є два шляхи. Перший шлях - створити більш складний матеріал для вашого ефекту. На прикладі вогню ви створите менше спрайтів і дозволите випадковості та складності покращеного матеріалу зробити роботу за вас, щоб оживити випромінювач. Інший варіант – використовувати дешевший матеріал і породжувати більше спрайтів, зберігаючи загальну вартість незмінною, але дозволяючи більшій кількості частинок виконувати вашу роботу в досягненні випадковості, на відміну від складнішого матеріалу. Майте на увазі, що матеріальні витрати зменшуються експоненціально за рахунок зменшення відстані (чотирикутник, намальований на екрані вдвічі далі від камери, коштує в 4 рази дешевше через те, що загальна площа пікселя експоненціально зменшується з відстанню, зменшуючи кількість пікселів, які перетягуються).

Таким чином, Niagara є сучасною системою симуляції часток в Unreal Engine, яка надає гнучкість і потужність для створення складних візуальних ефектів. Основні особливості Niagara включають:

- візуальне програмування. Niagara використовує візуальний скриптинг, що дозволяє художникам і розробникам налаштовувати ефекти без написання коду. Інтерфейс Niagara складається з модулів, які можна комбінувати для створення складних ефектів;
- гнучкість і масштабованість. Система дозволяє створювати як прості, так і дуже складні ефекти. Ви можете контролювати окремі частки або групи часток, застосовувати фізичні сили, змінювати поведінку часток в реальному часі;
- інтеграція з іншими системами. Niagara може інтегруватися з іншими системами Unreal Engine, такими як матеріали, шейдери, анімація і навіть ігрова логіка;

- продуктивність. Niagara оптимізована для використання в реальному часі, що дозволяє створювати ефекти, які працюють плавно навіть у складних сценах.

В той самий час Cascade – це попередник Niagara, який все ще використовується в багатьох проектах. Основні особливості Cascade:

- простота використання. Cascade має інтуїтивний інтерфейс, що дозволяє швидко створювати базові ефекти часток;
- бібліотеки ефектів. Unreal Engine включає багато готових ефектів часток, створених у Cascade, які можна використовувати і модифікувати;
- обмежені можливості. Порівняно з Niagara, Cascade має менш гнучкі можливості і менш підходить для створення складних ефектів.

У нашому випадку нам слід проаналізувати, наскільки дорогі наші матеріали, скільки спрайтів ми створюємо і наскільки близько до екрану ми будемо підходити до цих ефектів. Ці три властивості є головними особами, які приймають рішення з точки зору складності випромінювача, і всі вони повинні бути збалансовані.

Загалом, зосередимось на зменшенні складності наших матеріалів як на способі підвищення продуктивності, і завжди треба пам'ятати про потенційне перетягування, коли ми працюємо з випромінювачами в цілому.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

4.1 Сцена для тестування

В ході підготовки до проведення експериментів було створено демонстраційну сцену, яка дозволяє розміщувати оптимізуємі ефекти та порівнювати їх з оригінальними версіями. Побачити її можна на рисунку 4.1.

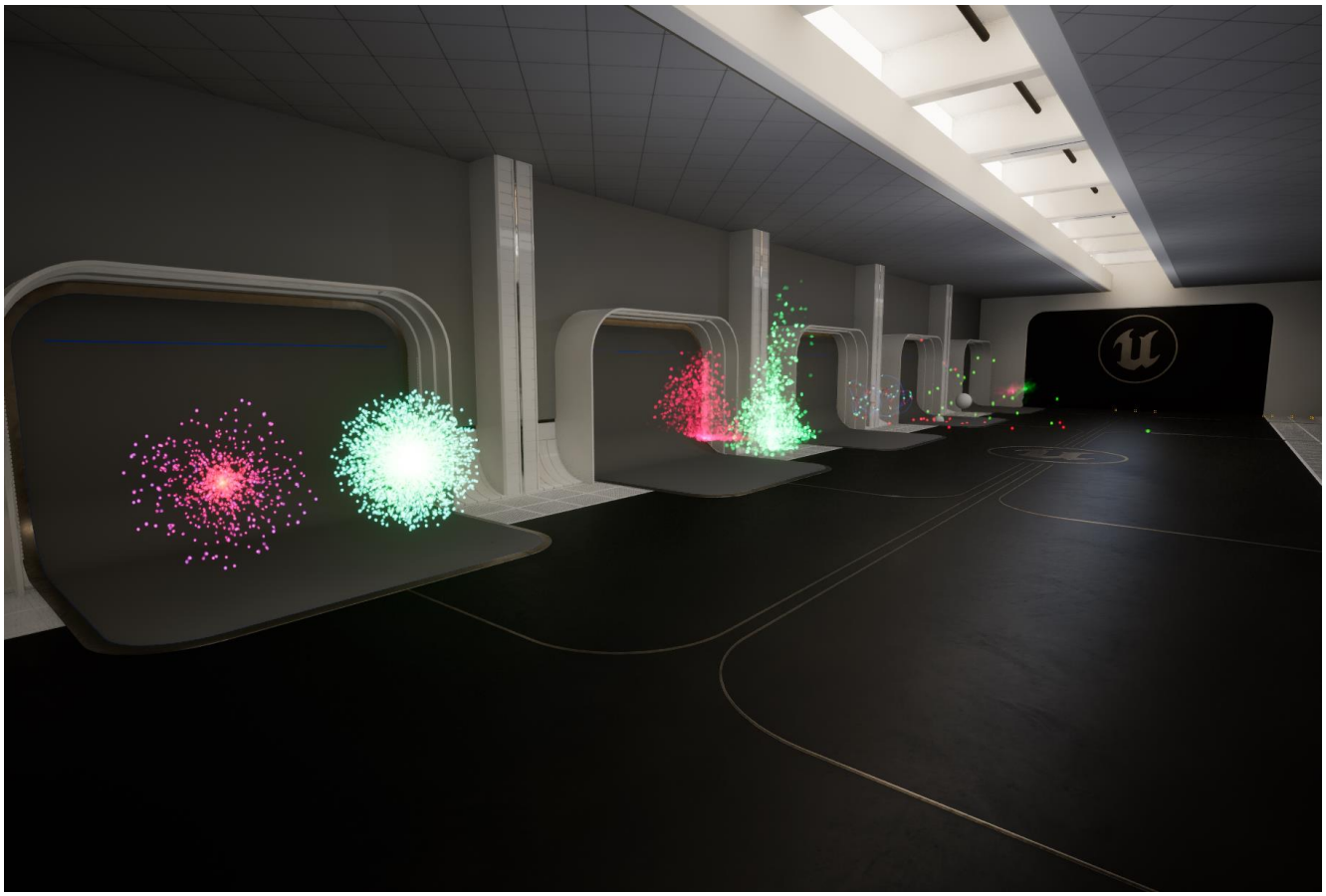

Рисунок 4.1 – Демонстраційна сцена

В движку активно використовуються буфери візуалізації. Буфери візуалізації в Unreal Engine є важливою частиною процесу рендерингу, що дозволяє реалізовувати складні графічні ефекти та оптимізувати продуктивність. Розуміння різних типів буферів та їх функцій допомагає розробникам створювати візуально привабливі сцени з ефективним використанням ресурсів.

Про буфери візуалізації можна довго розписувати, тому ми надаємо тільки базову інформацію для розуміння контексту:

а) G-Buffer:

1) G-Buffer містить інформацію про геометрію сцени, таку як нормалі, кольори, координати текстур, глибину і матеріали;
2) G-Buffer використовується у техніці деферентного рендерингу, що дозволяє обробляти світло та тіні в пост-процесингових проходах. Це підвищує продуктивність, оскільки обробка світла застосовується лише до видимих пікселів;

б) Depth Buffer (Z-Buffer):
1) містить інформацію про відстань від камери до кожного пікселя сцени;
2) використовується для вирішення проблем видимості, визначення, які об'єкти перекривають інші, і застосування ефектів, залежних від глибини, таких як туман і глибина різкості (depth of field);

в) Stencil Buffer:
1) зберігає цілочисельні значення для кожного пікселя, які можуть використовуватися як маски;
2) використовується для різних ефектів, таких як маскування частин зображення, відображення дзеркал та складних ефектів пост-обробки;

г) Color Buffer:
1) зберігає кольорову інформацію для кожного пікселя;
2) основний буфер для відображення кінцевого зображення на екран. Результат рендерингу сцен, включаючи всі застосовані світлові та пост-обробні ефекти;

д) Normal Buffer:
1) зберігає інформацію про нормалі поверхні для кожного пікселя;
2) використовується для обчислення світлових ефектів, зокрема освітлення і тіней, а також для застосування нормал-маппінгу;

е) Specular Buffer:
1) містить інформацію про відбивальні властивості поверхонь (спекулярність);
2) використовується для моделювання відбивань і блисків на поверхнях;

Як виглядає сцена з рисунку 4.1 у буферному уявленні можна побачити на рисунку 4.2.

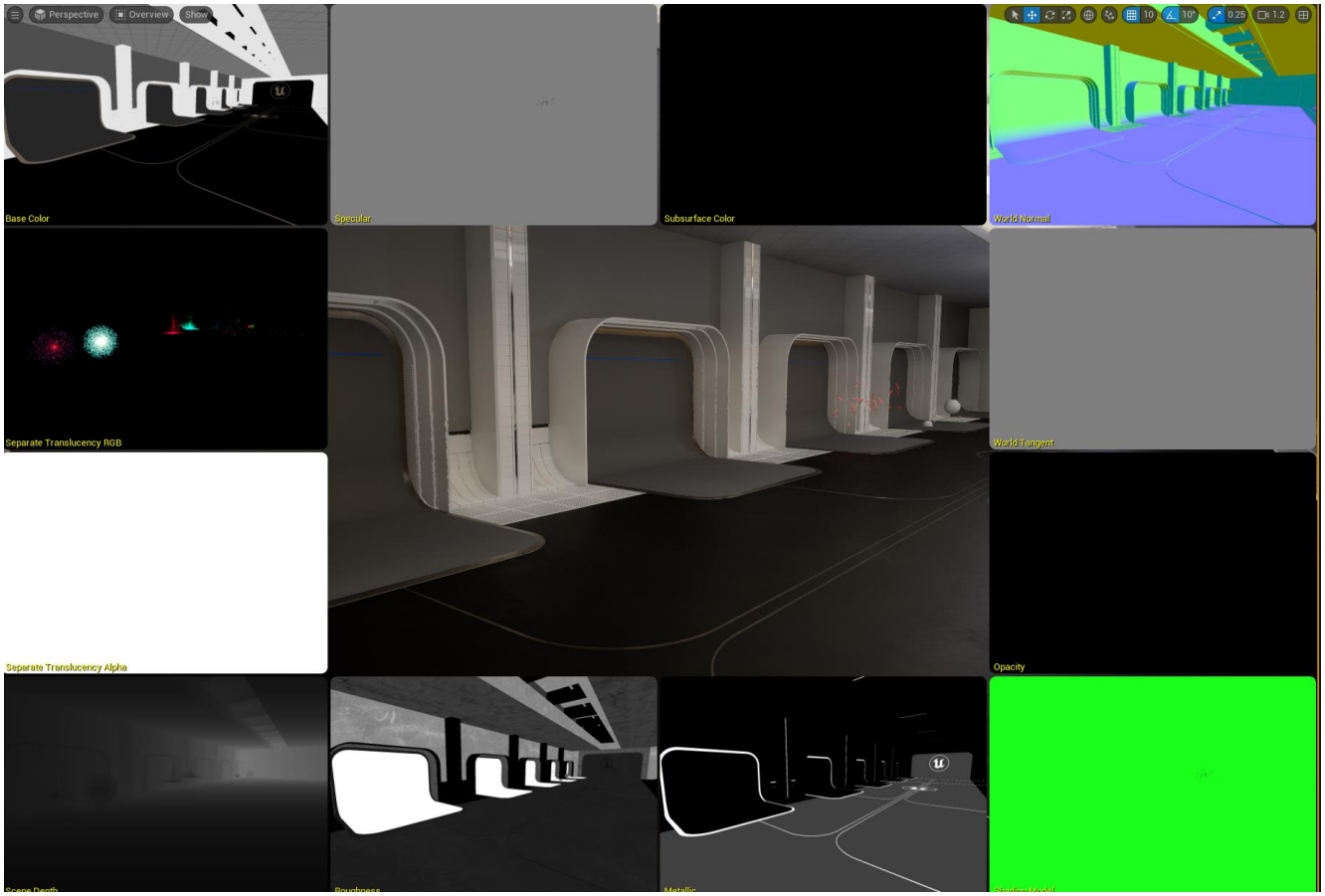

Рисунок 4.2 – Буферне уявлення демонстраційної сцени

Буферне уявлення потрібно саме для більш глибокого аналізу всього, що відбувається на сцені. Без нього складно виявити деякі проблеми, бо рендерінг кадру – дуже складний процес, який неможливо аналізувати в початковому вигляді. Для цього і потрібні уявлення.

4.2 Інструменти тестування

Для аналізу Niagara-систем є вбудовані інструменти, які дають можливість оцінювати різноманітні параметри цих систем. Як мінімум, якість оптимізації складно перевірити, коли робота йде на потужному ПК, тому, що доволі складно його навантажити настільки, щоб побачити вагому різницю. Якщо ж кожного разу робити білд під мобільний пристрій, можна втратити дуже багато часу саме на цей процес. Так, вбудовані інструменти мають деяку похибку і не враховують

специфіку цільових платформ, але все ж вони надають цифрові дані, з якими легко працювати і оцінювати відносну оптимізацію, яку вже після цього можна перевірити на цільовій платформі. Приклад одного з таких інструментів можна побачити на рисунку 4.3.

Рисунок 4.3 – Інструмент оцінки кількості часток в реальному часі

Другий з основних інструментів, який ми використовували для аналізу результатів експериментів – це показники абсолютного та відносного часу виконання для окремих функцій та підфункцій системи симуляції часток. Побачити його можна на рисунку 4.4.

Виконання випромінювача системи симуляції часток будується з декількох основних блоків. Перший блок відповідає за сам випромінювач, другий – за його оновлення. Третій відповідає за створення частки, четвертий – за її оновлення і так далі. Треба відмітити, що ми також можемо створювати блоки подій та окремих шагів обробки часток. Для них також будуть відображатись дані про час, який було затрачено на їх виконання.

Також треба відмітити, що нами були вказані основні інструменти аналізу ефективності оптимізаційних дій.

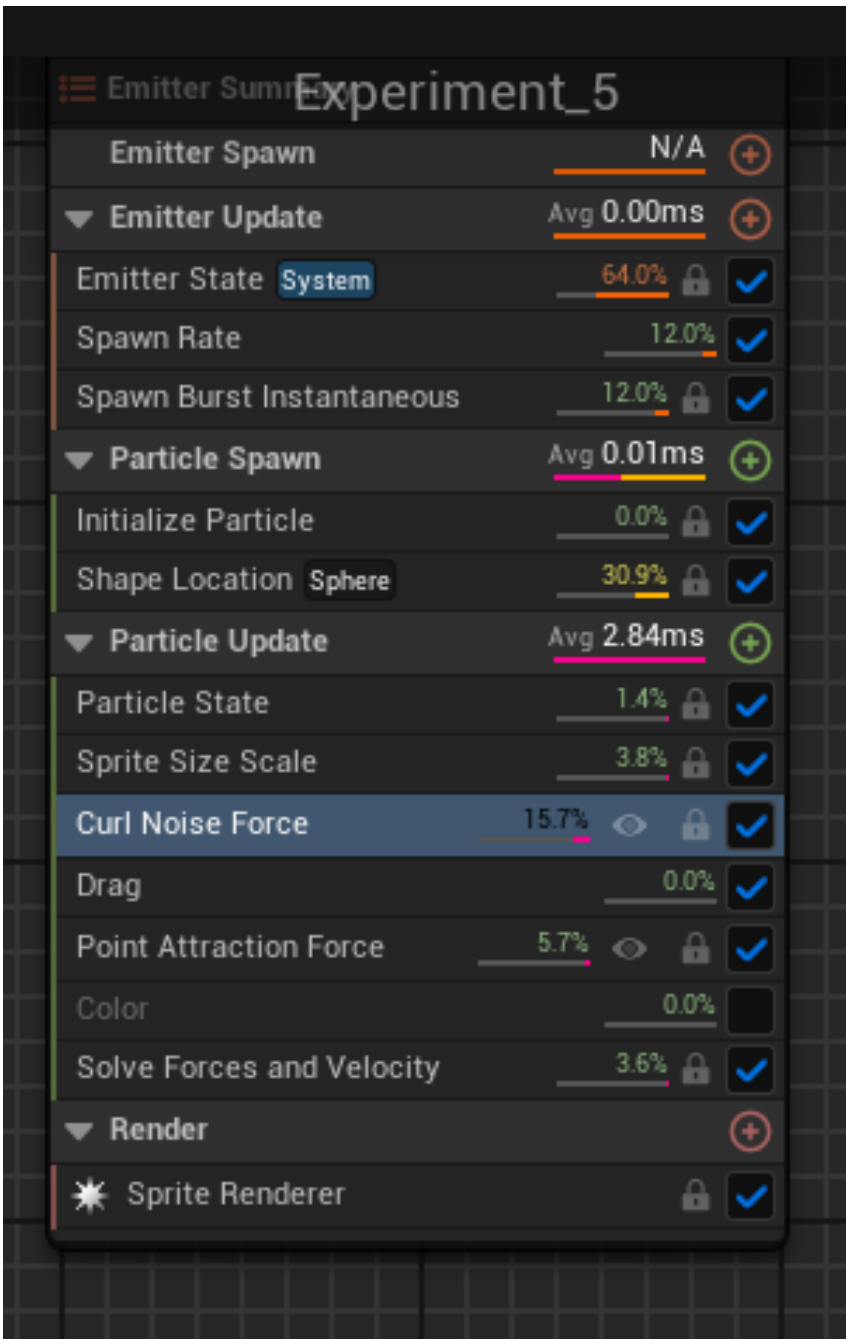

Рисунок 4.4 – Інструмент оцінки абсолютного та відносного часу виконання

В движку є багато додаткових інструментів, які також використовувались в залежності від необхідності для отримання прозорих даних, які дадуть змогу зробити коректні висновки.

5 АНАЛІЗ РЕЗУЛЬТАТІВ

5.1 Аспекти оптимізації

В Unreal Engine 5 всі реалізовані методи мають універсальну природу. Саме через це деякі недостатньо компетентні розробники стверджують, що він погано оптимізований. Він надає велику палітру можливостей, що потребує універсальності. А як відомо – універсальні рішення завжди менш-оптимізовані. Це відбувається через обробку додаткових умов. І навіть за умов відключаємого функціоналу ми маємо як мінімум перевірки на включення або відключення блоку.

Зазначені в аналізі систем оптимізації часток методи є базовими, тому маємо на увазі, що всі вони також використовуються для оптимізаційних дій. Основну ж і найбільш-вагому роботу нами було проведено при написанні особистих шейдерів. Універсальні функції-шейдери добре оптимізовані, але вони не знають яку саме частину функціоналу ми використовуємо в контексті нашої системи симуляції часток. Зрозуміло, що якість ігор, зроблених під мобільні платформи, набагато менша. Саме цьому ми можемо відрізати частину непотрібного функціоналу, розбивши функції на групи функцій з різною наповненістю і оптимізувати те, що в них виконується, під наші потреби. Ми виділили деякі основні блоки оптимізаційних дій, про які і розповімо далі.

Для оптимізації пам’яті використовуємо мінімальну кількість тимчасових буферів. Очищуємо і перевикористовуємо існуючі буфери замість створення нових. Слідкуємо за життєвим циклом тимчасових даних і видаляємо їх, коли вони більше не потрібні. Використовуємо текстури меншого розміру для тимчасових обчислень і ефектів, які не потребують високої роздільної здатності. Використовуємо прості і ефективні структури даних для зберігання тимчасової інформації. Мінімізуємо використання складних структур, які можуть займати більше пам'яті і потребувати більше часу на обробку.

Уникаємо зайвих обчислень і операцій, які можна виконати заздалегідь або під час передобробки. Використовуємо кешування результатів обчислень, коли це можливо, щоб зменшити кількість повторних операцій. Використовуємо GPU для

паралельних обчислень, коли це можливо, щоб зменшити навантаження на CPU. Переконуємося, що обчислення рівномірно розподілені між CPU і GPU, щоб уникнути вузьких місць. Використовуємо простіші шейдери для тимчасових обчислень і ефектів, які не вимагають високої точності або деталізації. Уникаємо використання складних функцій і інструкцій у шейдерах, коли це можливо.

5.2 Оптимізація руху часток

Першим кроком ми провели оптимізацію руху часток, результати якої можна побачити на рисунку 5.1. Тут і далі неоптимізований результат буде зображено ліворуч, а оптимізований – праворуч.

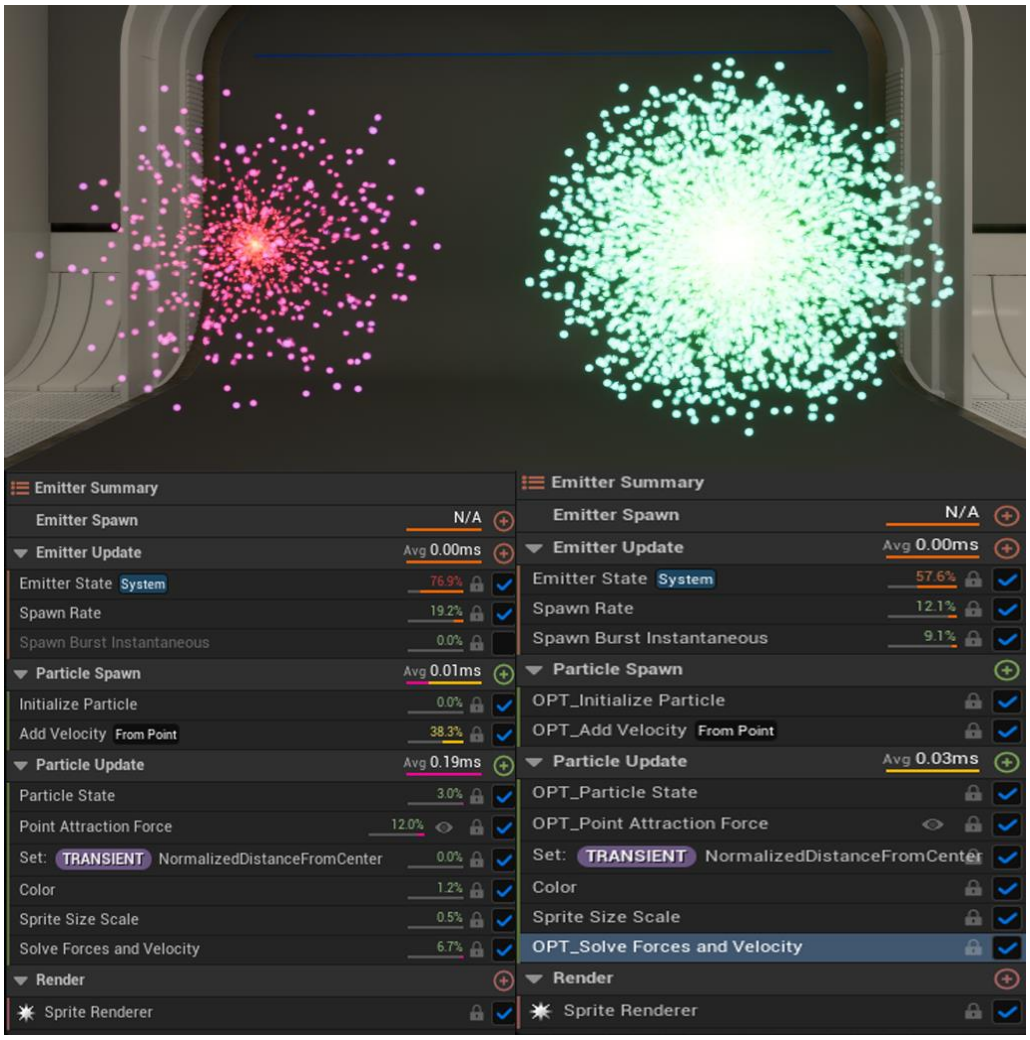

Рисунок 5.1 – Оптимізація руху часток

Нами було перероблено функції, які мають у назві приписку «OPT_». Як можна побачити, це вплинуло на оновлення часток, зменшивши час виконання з

0.19 мс. До 0.03 мс. Реальний результат куди більший, але точність знаків після крапки не передає значення, менші за одну соту, тому для демонстрації кількість часток було збільшено в 10 разів, що можна побачити на рисунку 4.3, подивившись на системи часток «Experiment_1» та «Experiment_1_opt». Код функцій в Niagara написаний мовою програмування HLSL. Мої навички програмування цією мовою доволі слабкі, тому я зробив їх оптимізовану версію з використанням Blueprints (див. рис. 5.2).

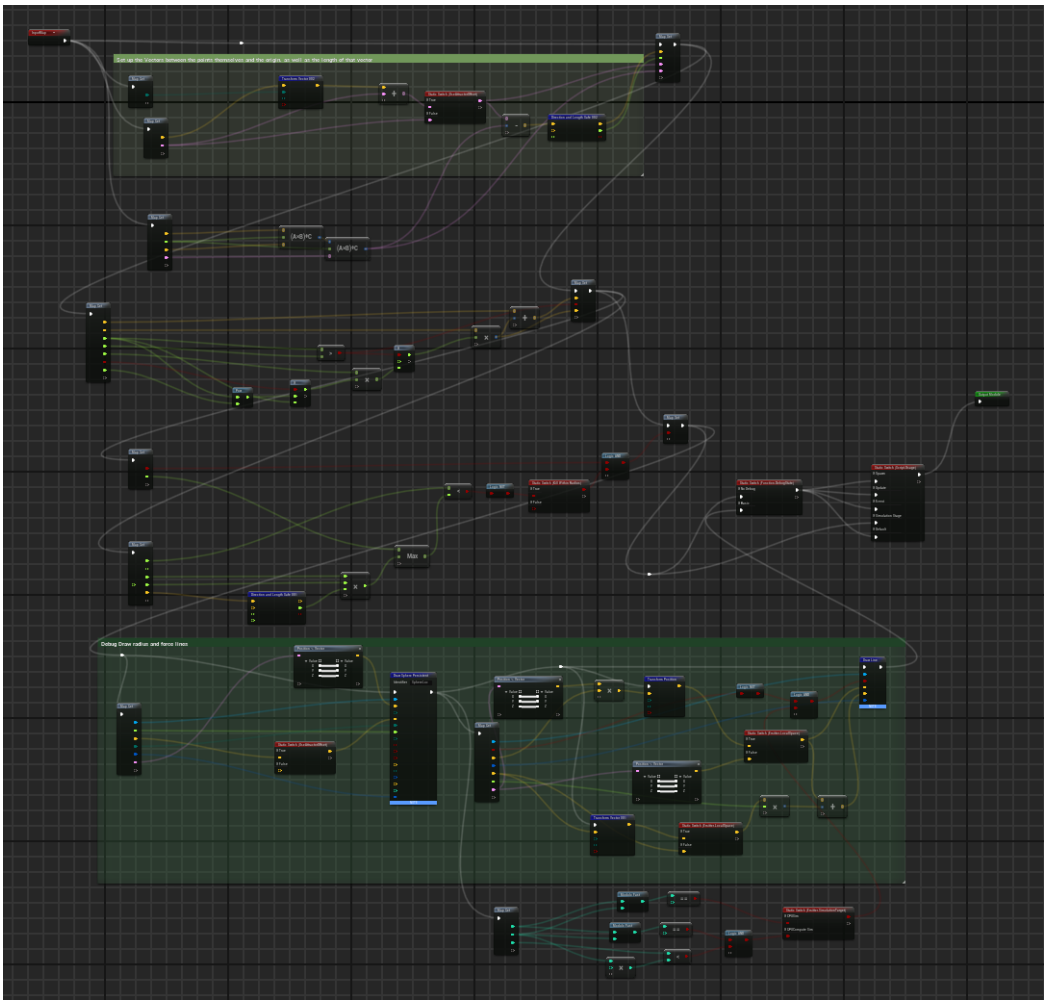

Рисунок 5.2 – Функція «OPT_Point Attraction Force»

Як видно з рисунку 5.2, код, написаний з допомогою мови програмування Blueprints при всій своїй зручності та компактності все ще доволі громіздкий для демонстрації в вигляді картинки. Щоб якісно його продемонструвати, треба робити розбивку на декілька картинок, що не є добре з точки зору формування

тексту, тому було прийнято рішення продемонструвати тільки найкомпактнішу функцію, як приклад того, як виглядала робота по оптимізації коду.

5.3 Оптимізація хаотичного руху часток

На рисунку 5.3 можна побачити приклади хаотичного руху часток та результат їх оптимізації.

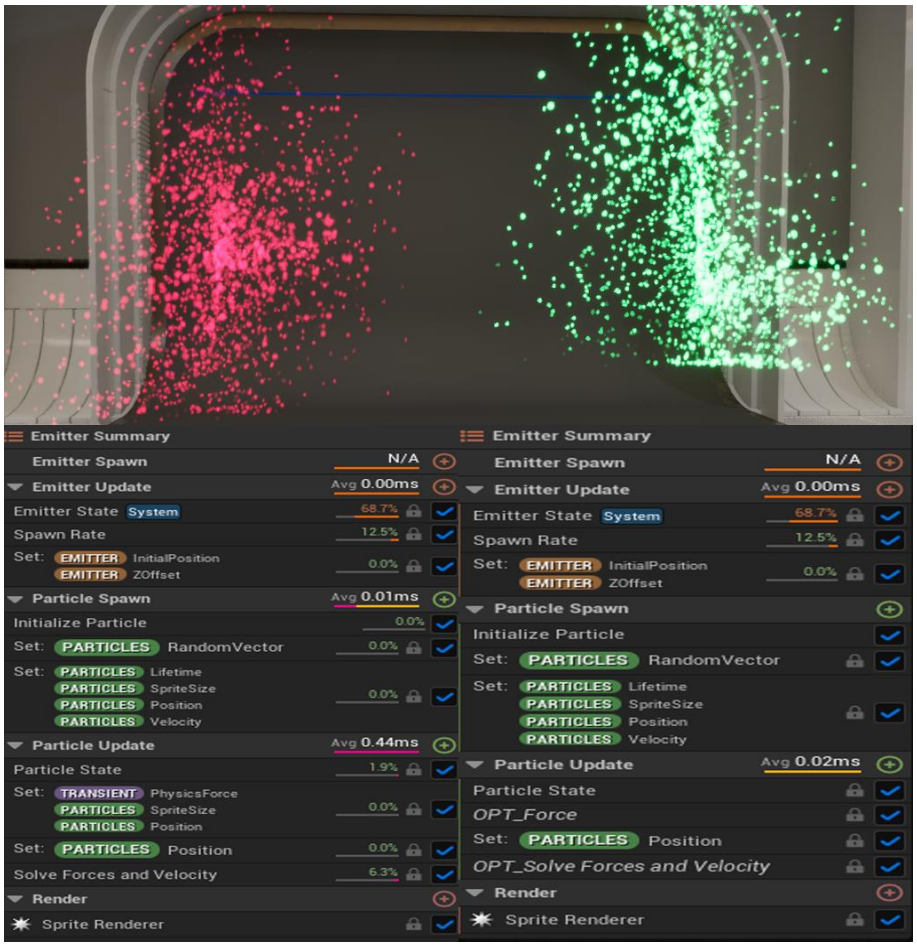

Рисунок 5.3 – Оптимізація хаотичного руху часток

Таким саме чином було оптимізовано хаотичний рух. Для даного прикладу кількість часток для обох систем однакова.

5.4 Оптимізація воронкового руху часток

На рисунку 5.4 можна побачити приклади воронкового руху часток та результат їх оптимізації.

Рисунок 5.4 – Оптимізація воронкового руху часток

Оптимізація воронкового руху не дала такого яскравого результату тому, що алгоритм вже було добре оптимізовано. Він не мав великої кількості додаткових функцій, як попередньо, тому основна робота складалась саме з переробки алгоритму на більш простий. Таким чином ми досягли результату зменшення часу виконання в 4 рази.

5.5 Оптимізація колізії часток

На рисунку 5.5 можна побачити приклади колізії часток та результат її оптимізації.

Рисунок 5.5 – Оптимізація колізії часток

Оптимізація колізії була найскладнішою задачею. Все через те, що ця функція дуже масштабна та має багато зв’язків з іншими функціями. Через це основною задачею було розібратись в коді та всіх зв’язках. В мене не вийшло її розбити на декілька, тому прийшлось тільки оптимізувати сам алгоритм. Через це оптимізація вийшла не тека значна, але на великій кількості часток це буде вагомо.

5.6 Оптимізація тривимірного шуму

На рисунку 5.6 можна побачити приклади тривимірного шуму та результат його оптимізації.

Рисунок 5.6 – Оптимізація тривимірного шуму

Оптимізація тривимірного шуму схожа з хаотичним рухом на рисунку 5.3, але вони працюють по-різному і використовуються для різних задач. Хаотичний рух повністю хаотичний, в той час, як тривимірний шум використовує специфічну тривимірну матрицю векторів, як шаблон. Цьому ми бачимо, що в двох системах частки рухаються майже однаково. Оптимізація трохи змінює траєкторії руху часток, але це непомітно, коли ми кажемо про візуальні ефекти, а не симуляцію, де не важна математична точність. Результат оптимізації коштує цієї ціни.

ВИСНОВКИ

В ході аналізу предметної галузі ми зробили висновки, що краще за все під наші задачі підходить Unreal Engine. Той самий Unity має більшу популярність серед розробників під портативні пристрої, але, в ту ж саму чергу, менш розвинуту систему створення VFX. Система, реалізована в Unity, глобально не оновлювала свою концепцію з моменту створення движка, а реалізовані в ній можливості недостатньо гнучкі для проведення чистого експерименту та аналізу можливих стратегій оптимізації. В той самий час Unreal Engine має найсучаснішу систему для створення VFX – Niagara, яка надає найкращу можливість вивчення і оптимізації систем симуляції часток, так як дозволяє не тільки використовувати вже готові рішення, але і писати код самостійно.

Порівняння Niagara та Cascade продемонстрували фундаментальну відмінність у принципах оптимізації систем симуляції часток. Так саме актуальний зараз, як і раніше тільки один принцип – зменшення кількості часток, але, як показує практика, цього частіше за все буває недостатньо. Зміна принципів оптимізації цих систем демонструє, що пошук все ще ведеться і прийняті раніше розробниками движка рішення вже неактуальні. Це підказує, що і нові рішення теж можуть бути недосконалими, бо намагаються вирішувати проблему оптимізації універсально, а не найкращим саме для мобільних пристроїв шляхом.

Як видно з результатів, найкращим напрямком для оптимізації систем симуляції часток є переробка вже існуючих універсальних функцій під конкретні задачі. Дуже складно розробити альтернативні бібліотеки і підтримувати їх, але можна прикладати описані принципи оптимізації для конкретних задач, коли ми маємо проблеми зі швидкодією часток на мобільних платформах.

Подальший аналіз вже існуючих методів оптимізації та їх покращення, або створення своїх є задачею актуальною і перспективною. Подальшим напрямком розвитку цієї теми ми бачимо аналіз найбільш вагомих функцій, виявлення моментів, які підпадають під оптимізацію, їх класифікація, сортування та описання алгоритмів оптимізації кожного кейсу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Матвєєв Д.І., Лановий О.Ф. Методи спрощення опрацювання систем симуляції незалежних часток у середовищі Unreal Engine 4 // Èlektronnoe Modelirovanie, 2023, №45(2) с. 95-107. URL: https://doi.org/10.15407/emodel.45.02.095 (дата доступу 15 травня 2024 р.)

2. Cascade Particle Systems // Unreal Engine Documentation. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/ (дата доступу 15 травня 2024 р.)

3. Niagara Visual Effects // Unreal Engine Documentation. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Niagara/ (дата доступу 15 травня 2024 р.)

4. GPUSprites Type Data // Unreal Engine Documentation. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/TypeData/GPUSprites/ (дата доступу 15 травня 2024 р.)

5. Collision Modules // Unreal Engine Documentation. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/Modules/Collision/ (дата доступу 15 травня 2024 р.)

6. Event Modules // Unreal Engine Documentation. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/Modules/Event/ (дата доступу 15 травня 2024 р.)

7. VFX Optimization Guide // Unreal Engine Documentation. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Optimization/ (дата доступу 15 травня 2024 р.)

8. Матвєєв Д.І., Лановий О.Ф. Проблеми оптимізації графіки під пристрої віртуальної реальності // ΛΌГОΣ.ONLINE, 2020, №14. URL: http://eoi.citefactor.org/ 10.11232/2663-4139.14.04 (дата доступу 15 травня 2024 р.)

9. Особливості підготовки 3D моделей для використання у VR проектах // Science, Research, Development. URL: http://www.xn-e1aajfpcds8ay4h.com.ua/files/118_01_xi_2021.pdf#page=35 (дата доступу 15 травня 2024 р.)

10. Порівняння методів текстурування моделей для мобільних платформ // Science, Research, Development. URL: http://www.xn-e1aajfpcds8ay4h.com.ua/files/118_01_xi_2021.pdf#page=37 (дата доступу 15 травня 2024 р.)

11. Дослідження інструментів та засобів оптимізації 3D-графіки в комп'ютерних іграх та їх застосування до ігор у жанрі "First-person Shooter" // Електронний архів Харківського національного університету радіоелектроніки. URL: https://openarchive.nure.ua/server/api/core/bitstreams/e8582e45-10b9-44bf-aabc-cb0b120389ee/content (дата доступу 15 травня 2024 р.)

12. Gamification approach to the creation of virtual laboratory works and educational courses / Revenchuk I., Sus B., Tmienova N., Bauzha O., Stirenko S. // CEUR Workshop Proceedings, 2020, 2711, с. 68-78. URL: https://ceur-ws.org/Vol-2711/ paper6.pdf (дата доступу 15 травня 2024 р.)

13. A Study of Optimization Models for Creation of Artificial Intelligence for The Computer Game in The Tower Defense Genre / Oksana Mazurova, Oleksandr Topchii, Oleksandr Samantsov, Mariya Shirokopetleva // Problem of Infocommunications. Science and Technology (PIC S&T′2020), Kharkiv, Ukraine.- 6-9 October 2020. Pp. 491-497. URL: https://ieeexplore.ieee.org/document/9468057 (дата доступу 15 травня 2024 р.)

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

1. Матвєєв Д.І., Лановий О.Ф. Методи спрощення опрацювання систем симуляції незалежних часток у середовищі Unreal Engine 4 // Èlektronnoe Modelirovanie, 2023, №45(2) с. 95-107. URL: https://doi.org/10.15407/ emodel.45.02.095 (дата доступу 15 травня 2024 р.)

8. Матвєєв Д.І., Лановий О.Ф. Проблеми оптимізації графіки під пристрої віртуальної реальності // ΛΌГОΣ.ONLINE, 2020, №14. URL: http://eoi.citefactor.org/ 10.11232/2663-4139.14.04 (дата доступу 15 травня 2024 р.)

12. Gamification approach to the creation of virtual laboratory works and educational courses / Revenchuk I., Sus B., Tmienova N., Bauzha O., Stirenko S. // CEUR Workshop Proceedings, 2020, 2711, с. 68-78. URL: https://ceur-ws.org/Vol-2711/ paper6.pdf (дата доступу 15 травня 2024 р.)

13. A Study of Optimization Models for Creation of Artificial Intelligence for The Computer Game in The Tower Defense Genre / Oksana Mazurova, Oleksandr Topchii, Oleksandr Samantsov, Mariya Shirokopetleva // Problem of Infocommunications. Science and Technology (PIC S&T′2020), Kharkiv, Ukraine.- 6-9 October 2020. Pp. 491-497. URL: https://ieeexplore.ieee.org/document/9468057 (дата доступу 15 травня 2024 р.)