УДК 004.921

# ОПТИМІЗАЦІЯ ГРАФІЧНОГО РЕСУРСУ В ІГРОВОМУ РУШІЇ UNREAL ENGINE 5

Гречка А. О.
Науковий керівник – ст. викл. Новіков Ю. С.
Харківський національний університет радіоелектроніки, каф. ПІ,
м. Харків, Україна
e-mail: anna.hrechka@nure.ua

This work is devoted to the graphical resource optimization in Unreal Engine 5. It describes some methods that help to achieve better performance and visual fidelity in games. It covers techniques, ranging from measuring shader performance to implementing shader optimization methodologies. In addition, the work includes information about Nanite virtualized geometry system, providing an examination of its functionality and benefits. The work also shows how you can enable Nanite, what you should avoid when using it, and gives a simple explanation of how it works.

Метою роботи є розгляд та аналіз методів оптимізації графіки в рушії Unreal Engine 5 задля покращення продуктивності та візуальної якості в ігрових проєктах. Розглянути вимірювання продуктивності шейдерів, впровадження методів оптимізації шейдерів та проаналізувати систему Nanite для віртуалізованої геометрії.

Розглянемо кілька методів виміру продуктивності шейдеру [1]. Один з методів – аналіз його складності в режимі перегляду, що надає загальне уявлення про вартість та ґрунтується на кількості інструкцій, що не гарантує високої точності. Інший метод – аналіз кількості інструкцій шейдеру, що дозволяє отримати більш точне уявлення про його продуктивність, бо враховує конкретні дії, які він виконує. Зменшення їхньої кількості сприяє швидшому виконанню, але серед недоліків є те, що не всі інструкції виконуються за однаковий проміжок часу й кожна платформа компілює різну кількість інструкцій.

При оптимізації шейдерів [3] позбуваємося того, чим не користуємось. На рисунку 1 наведено частину математичних дій, у вигляді вузлів, у яких немає необхідності. Також проводимо трансформацію математичних формул, тобто замінюємо на дешевші. Наприклад, у нас є вузли, які необхідні для створення маски. У вузлі Add зберігається результат Z компоненти нормалі та ще одного числового значення. Цей результат використовується у вузлі Power(x, 100), який виконує піднесення x до степеня 100, і він необхідний для зміни контрасту. Тож замінюємо Power(x, 100) на Subtract(, 0.5), отриманий результат є одним із множників вузлу Multiply(, 20) і потім отримане значення додаємо Add(, 0.5).

Крім того комбінуємо компоненти, щоб зменшити кількість математичних дій, пакуємо текстурні канали, що наведено на рисунку 2.

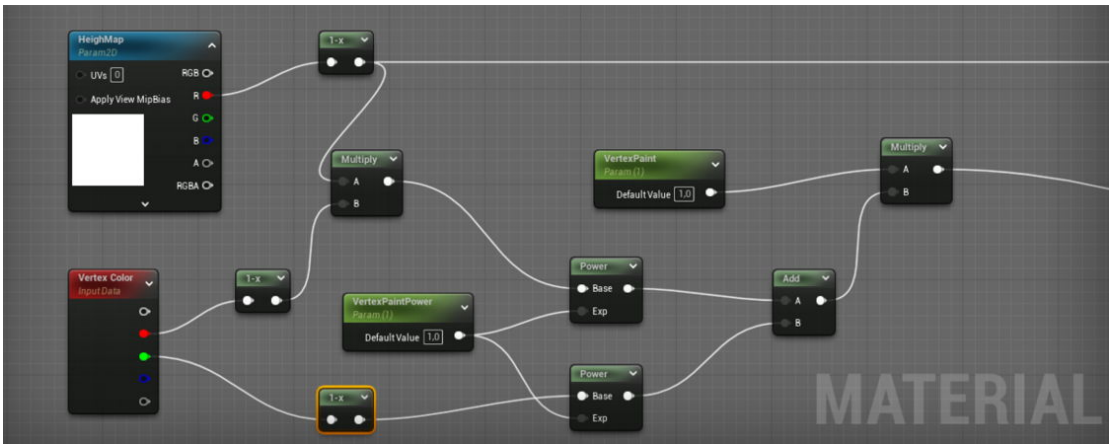


Рисунок 1 – Вузли одного з матеріалів

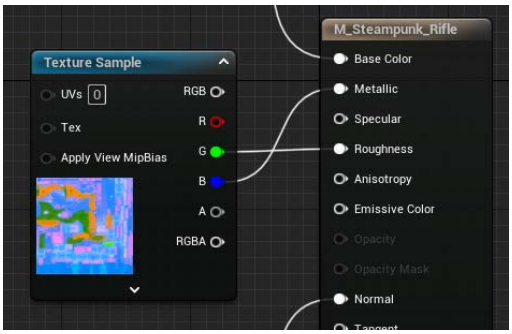


Рисунок 2 – Текстурний зразок, який поєднує в собі шорсткість та металічність

Розглянемо систему віртуалізованої геометрії Nanite [2], при якій рівень деталізації (LOD) обробляється автоматично й більше не вимагає ручного налаштування для окремих LOD мешів, а втрата якості малоймовірна. Будь-який статичний меш, на якому ця функція активована, зазвичай відтворюватиметься швидше, займатиме менше пам'яті та дискового простору. Активація Nanite можлива при імпорті, як зображено на рисунку 3, або в налаштуваннях Nanite та Import Settings для відповідного мешу.

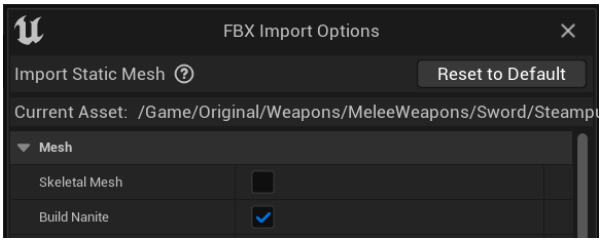


Рисунок 3 – Імпорт мешу

Варто вказувати необхідний масштаб мешу при імпорті, бо при створенні Nanite даних він береться до уваги й при подальшій його зміні з’являться дефекти. Для того, щоб прибрати дефекти, необхідно імпортувати меш знову або виконати необхідні дії в його налаштуваннях.

Якщо є необхідність переглянути, які з мешів на сцені мають увімкненою Nanite функцію, треба перейти до Nanite Visualization та обрати режим Triangles (Рис. 4).

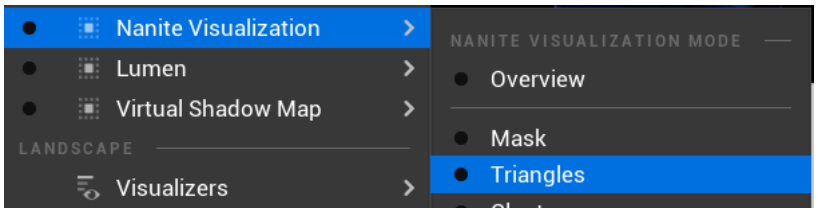


Рисунок 4 – Налаштування режиму

На рисунку 5 наведено меш з увімкненим Nanite.

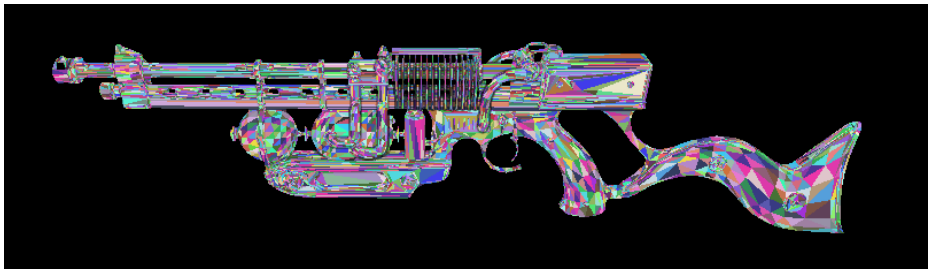

Рисунок 5 – Меш з увімкненим Nanite

Трикутники є основою Nanite. При імпорті меш аналізується й розбивається на ієрархічні кластери груп трикутників. Під час відтворення ці кластери змінюються на різних рівнях деталізації залежно від положення камери, і з'єднуються без швів з сусідніми кластерами в межах того ж об'єкта.

У результаті роботи було розглянуто методи виміру продуктивності шейдеру, їхні переваги та недоліки. Також наведено способи оптимізації шейдерів. Було продемонстровано підключення системи віртуалізованої геометрії Nanite, проаналізовано її особливості.

Список використаних джерел:

1. Guidelines for Optimizing Rendering for Real-Time. Unreal Engine 5.0 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/5.0/en-US/guidelines-for-optimizing-rendering-for-real-time-in-unreal-engine/ (дата звернення: 03.03.2024).

2. Nanite Virtualized Geometry. Unreal Engine 5.0 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/ (дата звернення: 04.03.2024).

3. Матвєєв Д., Лановий О. Проблеми оптимізації графіки під пристрої віртуальної реальності. 2020. URL: https://ojs.ukrlogos.in.ua/index.php/2663-4139/article/view/5033.