Пояснювальна записка

до магістерської наукової роботи

на тему:

«ОПТИМІЗАЦІЯ ІНТЕРФЕЙСІВ ЗАСТОСУНКІВ ДЛЯ МОБІЛЬНИХ ПЛАТФОРМ НА ОСНОВІ UNITY UI»

Спеціальність 124 – Системний аналіз

124 – МКР – 607.21610412

Студент _____ Кутняк В.Є.
«___» _____ 2022 р.

Керівник _____ Кондратенко Г.В.
к.т.н., доцент
«___» _____ 2022 р.

Миколаїв – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 3
ВСТУП 4
1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ. ПОСТАНОВКА ЗАДАЧІ 6
1.1 Опис предметної сфери 6
1.2 Основні поняття та визначення 9
1.3 Огляд існуючих аналогів та опис процесу діяльності 13
Висновки до розділу 1 14
2 МАТЕМАТИЧНІ МОДЕЛІ, МЕТОДИ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ 15
2.1 Методи та інформаційні технології вирішення поставленої задачі 15
2.2 Моделі для вирішення поставленої задачі 21
Висновки до розділу 2 27
3 МОДЕЛЮВАННЯ ОТРИМАННИХ РЕЗУЛЬТАТІВ 28
3.1 Профілювання Unity UI 28
3.2 Початок профілювання Unity UI 29
3.3 Технічна документація 33
Висновки до розділу 3 39
ВИСНОВКИ 40
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 41
ДОДАТОК А 44
ДОДАТОК Б Програмний код оптимізації UI нотіфікацій 53
ДОДАТОК В Програмний код генерації модулів UI 56

ПЕРЕЛІК СКОРОЧЕНЬ

CPU – центральний процесор;

FPS (FramePerSecond) – кількість кадрів у секунду;

MVD (Minimum Vertex Distance) – мінімальна відстань між вертексами;

SF (Shader Forge) – середа розробки шейдерів;

TMP (TextMeshPro) – текстовий компонент UnityUI;

UI – User Interface, інтерфейс користувача;

ВСТУП

Кожного дня ми користуємось смартфонами, а отже, мобільними додатками. У черзі можемо запустити гру на мобільному телефоні та скоротати таким чином час. Взаємодія користувача із додатком відбувається за допомогою користувацького інтерфейсу. Інтерфейс повинен бути зрозумілим, доцільним, завершеним та повністю функціонально відповідати самому додатку/грі. А ще, він повинен бути оптимізованим. Увесь масив даних на екрані з яким користувач взаємодіє постійно оновлюється у реальному часі, тому інтерфейс повинен займати невелику кількість ресурсів девайсу, залишаючи більшу їх частину на складніші розрахунки. І ось після завантаження додатку чи гри та першого запуску Ви помічаєте, що Ваш смартфон починає нагріватись, з'являються просідання кадрів. Але ж смартфон ніби і обладнаний дуже потужним процесором та відеоприскорювачем. І Ви не можете зрозуміти, що не так.

Об’єктом дослідження є інтерфейси застосунків для мобільних платформ.

Предметом дослідження є техніки оптимізація інтерфейсів застосунків для мобільних платформ на основі Unity UI

Мета магістерської роботи полягає в оптимізації інтерфейсу у рушії Unity на основі згенерованої технічної документації щодо максимальної економії ресурсів системи та збереження концептуальної (ідейної) складової інтерфейсу застосунку.

Завдання для досягнення мети:

- аналіз сучасного стану задачі оптимізації інтерфейсів застосунків;
- огляд існуючих технологій оптимізації інтерфейсів;
- експертне оцінювання технологій та методів оптимізації;
- аналіз результатів застосування обраних методів для розв’язання поставленої задачі.

Практична цінність роботи полягає у тому, що кожен спеціаліст, який займається розробкою застосунку чи імплементацією інтерфейсу за допомогою

згенерованої документації у даній роботі із легкістю зможе оптимізувати уже створений інтерфейс чи створити інтерфейс з нуля, використовуючи відповідну документацію. Кожного дня зростає продуктивність мобільних пристроїв та ПК, удосконалюються інструменти для створення ігор. Незважаючи на це, до цих пір існують дуже серйозні технічні обмеження. На ПК і сучасних консолях їх менше, на мобільних платформах - набагато більше.

Інтерфейс користувача (UI) — це широкий термін для будь-якої системи, фізичної чи програмної, яка дозволяє користувачеві взаємодіяти із певною технологією. Багато різних типів інтерфейсів користувача постачаються з різними пристроями та програмами. Багато з них мають певну загальну схожість, хоча кожен є унікальним у ключових аспектах. Один з основних типів інтерфейсу користувача називається графічним інтерфейсом користувача (GUI). Сюди входять інтерфейси сучасних операційних систем, з якими багато з нас знайомі, включаючи Windows, а також інші типи програм, створених переважно за допомогою піктограм або зображень, а не текстових команд. Користувачі можуть порівняти графічний інтерфейс користувача з текстовим інтерфейсом, таким як система MS-DOS, яка використовувалася для керування персональними комп’ютерами протягом останніх кількох десятиліть.

Інші інтерфейси користувача включають інтерфейси сенсорного екрана, звичайні типи інтерфейсів для мобільних пристроїв та інші фізичні типи апаратних інтерфейсів. Наприклад, пульт дистанційного керування DVD-програвачем, аудіосистемою, телевізором або ігровою консолью може вважатися інтерфейсом користувача для цього пристрою. Інші програмно-орієнтовані інтерфейси користувача стають складнішими, часто використовують комбінацію графічних і текстових елементів для керування конкретними користувачами. Процес модифікації інтерфейсу для вдосконалення ефективності і є оптимізацією. Метою оптимізації є зменшення часу, обсягу оперативної та відеопам’яті, необхідних системі для відтворення інтерфейсу на екрані користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ. ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери

Unity UI являє собою набір інструментів для розробки інтерфейсів користувача для ігор і програм. Це система інтерфейсу користувача на основі GameObject, яка використовує компоненти та Game View для організації, розташування та стилю інтерфейсу користувача.

Unity надає три UI системи, які ви можете використовувати для створення інтерфейсів користувача (UI) для редактора Unity та програм, створених у редакторі Unity:

- інструментарій інтерфейсу користувача (UI Toolkit);
- пакет Unity UI (uGUI);
- IMGUI.

UI Toolkit — це найновіша система інтерфейсу користувача в Unity. Він розроблений для оптимізації продуктивності на різних платформах і заснований на стандартних веб-технологіях. Ви можете використовувати UI Toolkit для створення розширень для редактора Unity, а також для створення інтерфейсу користувача під час виконання для ігор і програм (під час встановлення пакета UI Toolkit.

Набір інструментів інтерфейсу користувача включає:

– система інтерфейсу користувача, яка містить основні функції та функціональні можливості, необхідні для створення інтерфейсів користувача;

– типи об’єктів інтерфейсу користувача, створені на основі стандартних веб-форматів, таких як HTML, XML та CSS. Використовується їх для структурування та стилю інтерфейсу;

– інструменти та ресурси для навчання використання UI Toolkit, а також для створення та налагодження інтерфейсів.

Unity має намір, щоб UI Toolkit став рекомендованою системою інтерфейсу користувача для нових проектів розробки інтерфейсу користувача, але в ньому все ще відсутні деякі функції, наявні в Unity UI (uGUI) та IMGUI.

Пакет Unity UI (uGUI). Пакет Unity User Interface (Unity UI) (також званий uGUI) — це старіша система інтерфейсу користувача на основі GameObject, яку можна використовувати для розробки інтерфейсу користувача під час виконання для ігор і програм. В інтерфейсі користувача Unity ви використовуєте компоненти, щоб упорядкувати, розташувати та стилізувати інтерфейс користувача. Він підтримує розширений рендеринг і текстові особливості.

Таблиця 1.1

Порівняння систем інтерфейсу користувача в Unity

Тип UI	UI Toolkit	Unity UI(uGUI)	IMGUI
Runtime (debug)	✓	✓	✓
Runtime (in-game)	✓	✓	X
Рушій Unity	✓	X	✓

У оптимізації користувацького інтерфейсу немає універсальних правил, що працюють у будь-якій ситуації. Все зводиться до пошуку балансу між складовими рушія Unity, а саме: вартістю батчингу та кількістю викликів для відтворення.

По-перше необхідно виділити чотири основні проблеми, що спонукають нас звернутися до проблеми оптимізації інтерфейсу:

– занадто велике завантаження GPU (надмірне навантаження на відтворення зображення);

– занадто багато навантаження на процесор при зміні/оновленні полотна;

– занадто багато змінних елементів, що ведуть до перебудови полотна;

– занадто велике навантаження на процесор при генерації мешів (зазвичай пов'язане з текстом).

Основним елементом інтерфейсу користувача Unity є полотно (Canvas). Він несе відповідальність за генерацію, сортування та відтворення сітки дочірніх елементів інтерфейсу. Всі елементи інтерфейсу повинні бути дочірніми для будь-якого полотна, інакше вони не будуть відображатися на екрані. Відтворення відбувається від далекого до найбільш близького об'єкта від камери (back-to-front) у черзі прозорості (Transparent queue) з альфа-змішуванням. Отже, коректне налаштування полотна є першим етапом оптимізації інтерфейсу.

Наступним етапом є підготовка графічних матеріалів проекту задля подальшого заощадження значної кількості ресурсів девайсу. Саме для таких цілей у Unity існує функціонал Sprite Atlas (атлас спрайтів), що поєднує в собі кілька текстур. Атласи дозволяють зменшити кількість відтворень (draw call) та підвищення продуктивності.

Сам функціонал є простим але чи найнеобхіднішим для кожного проекту, що наповнюється графікою. Слід мати на увазі, що навіть якщо на сцені використовуються один або декілька спрайтів із атласу, то атлас все одно буде повністю завантажено. Тому не має сенсу намагатися ввести всі зображення в один масивний атлас, який на пристроях з невеликою операційною пам'яттю може зайняти велику частину. Краще розділити на деякі менші частини.

Оптимізація завжди про вибір, про пошук балансу між різними, а іноді навіть і суперечливими один одному рішеннями. Немає універсальної інструкції на «всі випадки життя», не можна брати кожен із зазначених методів без попереднього аналізу та впевненості, що ваші дії гарантовано призведуть до позитивного результату.

Не треба негайно сплющувати усю графіку і робити дуже мінімалістичний але оптимізований сірий квадрат на сірому фоні, відмовившись від приємної візуальної складової інтерфейсу. Тим не менш, кожен раз, поглиблюючись, з’являється можливість уникнути збільшення кількості проблем навіть на початку.

Ситуації варіюються від проекту до проекту, від мети до мети, але загальні принципи завжди залишаються однаковими.

1.2 Основні поняття та визначення

Інтерфейс користувача (ІК), (англ. user interface, UI) — засіб зручної взаємодії користувача з інформаційною системою. Сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача; у графічних системах інтерфейс користувача, втілюється багатовіконним режимом, змінами кольору, розміру, видимості (прозорість, напівпрозорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштуваннями як самих вікон, так і окремих їх елементів (файли, теки, ярлики, шрифти тощо), доступністю багатокористувацьких налаштувань.

Інтерактивна система (interactive system) - комп'ютерна система, яка характеризується значною кількістю взаємодії між людиною і комп'ютером. Інтерактивними системами є операційні системи (Macintosh, Windows, Linux, тощо), ігри, редактори, CAD-CAM системи, а також системи введення даних, які включають високий ступінь взаємодії людини з комп’ютером. Веб-браузери і інтегровані середовища розробки (IDE) також є прикладами дуже складних інтерактивних систем.

Програмний інтерфейс - набір методів для взаємодії між програмами.

Фізичний інтерфейс - спосіб взаємодії фізичних пристроїв. Частіше за все мова йде про комп'ютерні портах.

Інтерфейс користувача - це сукупність програмних і апаратних засобів, що забезпечують взаємодію користувача з комп'ютером. Основу такої взаємодії складають діалоги. Під діалогом в даному випадку розуміють регламентований обмін інформацією між людиною і комп'ютером, який здійснюється в реальному масштабі часу і спрямований на спільне вирішення конкретної задачі. Кожен діалог складається з окремих процесів введення / виводу, які фізично забезпечують зв'язок

користувача і комп'ютера. Обмін інформацією здійснюється передачею повідомлення.

Полотно (Canvas) - це основний елемент інтерфейсу, який є контейнером для інших елементів.

Рендеринг (rendering) - процес візуалізації, що виконується за допомогою програмного забезпечення.

Наступні різновиди комп'ютерної візуалізації створені через велику різноманітність сфери її застосувань:

- фотореалістична візуалізація;
- нефотореалістична візуалізація.

Ці різновиди отримуються за допомогою використання одного чи скупності наступних методів:

1) High Dynamic Range Rendering (високодинамічне відображення діапазону);
2) алгоритм «Scanline»;
3) об'ємний рендеринг (англ. volume rendering);
4) Z-буферизація.

Залежно від мети, розрізняють пре-рендеринг, як досить повільний процес візуалізації, що застосовується в основному при створенні відео, і рендеринг у режимі реального часу, застосовуваний у комп'ютерних іграх.

Меш (mesh) – це сукупність вершин, ребер і граней, які визначають форму багатогранного об'єкта в тривимірній комп'ютерній графіці і об'ємному моделюванні. Гранями зазвичай є трикутники, чотирикутники або інші прості опуклі багатокутники (полігони), так як це спрощує рендеринг, але сітки можуть також складатися і з найбільш загальних увігнутих багатокутників, або багатокутників з отворами.

Вчення про полігональні сітки - це великий підрозділ комп'ютерної графіки та геометричного моделювання. Безліч операцій, що проводяться над сітками,

може включати булеву алгебру, згладжування, спрощення та багато інших. Різні уявлення полігональних сіток використовуються для різних цілей і програм. Для передачі полігональних сіток по мережі використовуються мережеві уявлення, такі як «потокові» і «прогресивні» сітки. Об'ємні сітки відрізняються від полігональних тим, що вони явно представляють і поверхню і обсяг структури, тоді як полігональні сітки явно представляють лише поверхню, а не обсяг. Так як полігональні сітки широко використовуються в комп'ютерній графіці, для них розроблені алгоритми трасування променів, виявлення зіткнень і динаміки твердих тіл.

Квад (quad) – це меш, який має форму чотирикутника.

Батчинг (batching) – об’єднання мешів об'єктів у один великий. Для відтворення об'єкта на екрані візуалізатор відправляє команду (draw call) графічного API (наприклад, OpenGL або Direct3D). Графічний API виробляє значну роботу для кожного DC, що сильно впливає на продуктивність CPU.

Unity може об'єднувати об'єкти під час виконання, щоб малювати їх разом в рамках одного DC. Ця операція називається "батчинг" (batching). Більшість об'єктів Unity може батчитись для отримання найкращої продуктивності на стороні CPU.

Вбудована підтримка батчинга в Unity має значну перевагу над простим об'єднанням геометрії в програмі моделювання і перед використанням скриптів є в пакеті Standart Assets скрипту CombineChildren). Батчинг в Unity працює після етапу визначення видимості. Візуалізатор обрізає кожен об'єкт індивідуально і кількість візуалізованої геометрії залишається такою ж як і без батчингу. Об'єднуючи ж геометрію в програмі моделювання або іншим шляхом, знижується ефективність обрізки і, в результаті, візуалізується значно більша кількість геометрії.

Виклик відтворення (draw-call) – команда на відтворення від програми до графічного API (наприклад, OpenGL або Direct3D).

CPU часто обмежений кількістю речей, які повинні бути намальовані, також відомо, як "draw calls". Draw call - Це запит відтворення, чим більше об'єктів на екрані, тим він більше, чим він більший, тим сильніше навантаження.

GPU обробляє занадто багато вершин. Яка кількість вершин є нормальною, визначається GPU і набором вертексних шейдерів.

Transparent queue – черга відтворення прозорих об’єктів.

Альфа-змішування (Alpha blending) – алгоритм змішування пікселів по альфа-каналу для отримання зображення з прозорістю.

Атлас (Atlas) — вид ресурсів, який об’єднує кілька текстур в одну.

Підтекстури відображаються на об'єкт, використовуючи UV-перетворення, при цьому координати в атласі задають, яку частину зображення потрібно використовувати. У візуалізаторах нерідко використовується безліч маленьких текстур, причому перемикання з однієї текстури на іншу є відносно повільним процесом. Тому в подібних ситуаціях буває доцільно застосування одного великого зображення замість безлічі маленьких.

Наприклад, в іграх з tile-графікою можна отримати хороший виграш в швидкості виведення картинки.

Атласи можуть містити як підтекстури однакових розмірів, так і підтекстури різних розмірів (зазвичай, є ступенем двійки). Для складання атласів використовуються як програми-генератори, так і ручне складання. При використанні MIP-текстурування необхідно передбачити, що підтекстури повинні бути розставлені таким чином, щоб не виникало ситуації, коли одна з них «перекриває» іншу.

Парент (Parent) – батьківський елемент.

Інспектор (Inspector) - вікно властивостей редактора.

Останні дослідження та публікації

Some of the best optimization tips for Unity UI/ Unity Technologies. Публікація містить часто використовувані методи оптимізації спеціалістами та розробниками рушія Unity.

Оптимизация Unity UI без кода / Ольга Кінчак, Микита Кандибін

Практично у всіх пунктах роботи можна обійтися без коду, налаштовуючи компоненти безпосередньо в редакторі, а навіть передбачаючи на стадії проектування макетів інтерфейсу.

Graphics performance fundamentals/ Unity

Стаття містить поради та способи оптимізації графіки у проекті для максимальної продуктивності.

Types of Vfx and its Process / Abhijit Sarkar

Опис основних технологій таких як: превізуалізація, ротоскопія, киїнг та інше.

Рекомендации по производительности для Unity / Harrison Ferrone

1.3 Огляд існуючих аналогів та опис процесу діяльності

1. Particle system. Manual / Unity Technologies. У мануалі від розробників Unity є багато вказівок та досліджень у цій сфері. Детально описані деякі функції, але без поглиблення у оптимізацію, чого бракує молодим розробникам та VFX-художникам. Зібрані також певні приклади із технічної документації Unity 5 із застосування нової технології VFX-Graph;

2. Оптимізація VFX для ігрових платформ / Євген Серегін.

У своїй роботі він описує певні особливості оптимізації ефектів для мобільних платформ. Чітко описані правила збереження концепту ефекту та його реалізації.

За середовище дослідження проблеми було прийняте рішення взяти багатоплатформовий інструмент для розробки дво- та тривимірних додатків та ігор - Unity 5. Вибір саме цього інструменту обумовлений доступністю для

випробування усіх досліджень, що передбачені для розкриття рішення завдання дипломної роботи.

Саме завдяки таким потужним інструментам як Unity, створюються найпопулярніші додатки, ігри та інші інструменти, а отже, є можливість проаналізувати середні показники технічного забезпечення користувачів.

Сучасні інструменти створення візуальних ефектів дозволяють рендерити максимально наближені до реалізму фізичні явища та імітації, що мають місце у багатьох сферах діяльності людини. Але ж чи є вони раціональними с точки зору навантаження на систему? Адже більшість девайсів за своєї бюджетності мають невеликі ресурси процессору, оперативної та відеопам’яті.

Висновки до розділу 1

У першому розділі дипломної роботи були проведені аналіз предметної сфери, огляд та аналіз наявних аналогів, були представлені пояснення до термінів, що використовуються у науковій роботі.

Були досліджені та описані основні принципи та технології, що використовуються для оптимізації інтерфейсів, оцінена перспективність цієї галузі та проведено аналіз основних напрямків розвитку. Спираючись на результати аналізу можна зробити висновок, що головними проблемами оптимізації інтерфейсу є використання центрального та графічного процесорів для рендерингу полотна. Можна зробити висновок, що для оптимізації ефектів необхідно редагувати не тільки саме полотно, але й компоненти, що є вкладеними у даний Canvas. Неправильно визначений розмір текстури впливає на продуктивність інтерфейсу на рівні із перебільшенням кількості створенних груп розміток для сортування інтерфейсу. Аналіз аналогів показав, що проблема оптимізації є досить росповсюдженою і методи оптимізації відрізняються від проекту до проекту. Спираючись на дослідження аналогів, можна зробити висновок, що було розглянуто далеко не всі методи оптимізації як для мобільних платформ так і для ПК.

2 МАТЕМАТИЧНІ МОДЕЛІ, МЕТОДИ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Методи та інформаційні технології вирішення поставленої задачі

Для оптимізації візуальних ефектів використовуються наступні методи.

2.1.1 Performance Profiler

Перше місце, куди потрібно дивитися, коли необхідно поліпшити продуктивність - це Profiler (рис. 2.1). Ця функціональність дозволяє аналізувати проблемні місця. Профайлер - безцінний інструмент. З його допомогою можна визначити, де виникають проблеми з частотою кадрів. Для його використання запустіть додаток на мобільному пристрої і профайлер на PC. Після синхронізації профайлер починає завантажувати дані про продуктивність.

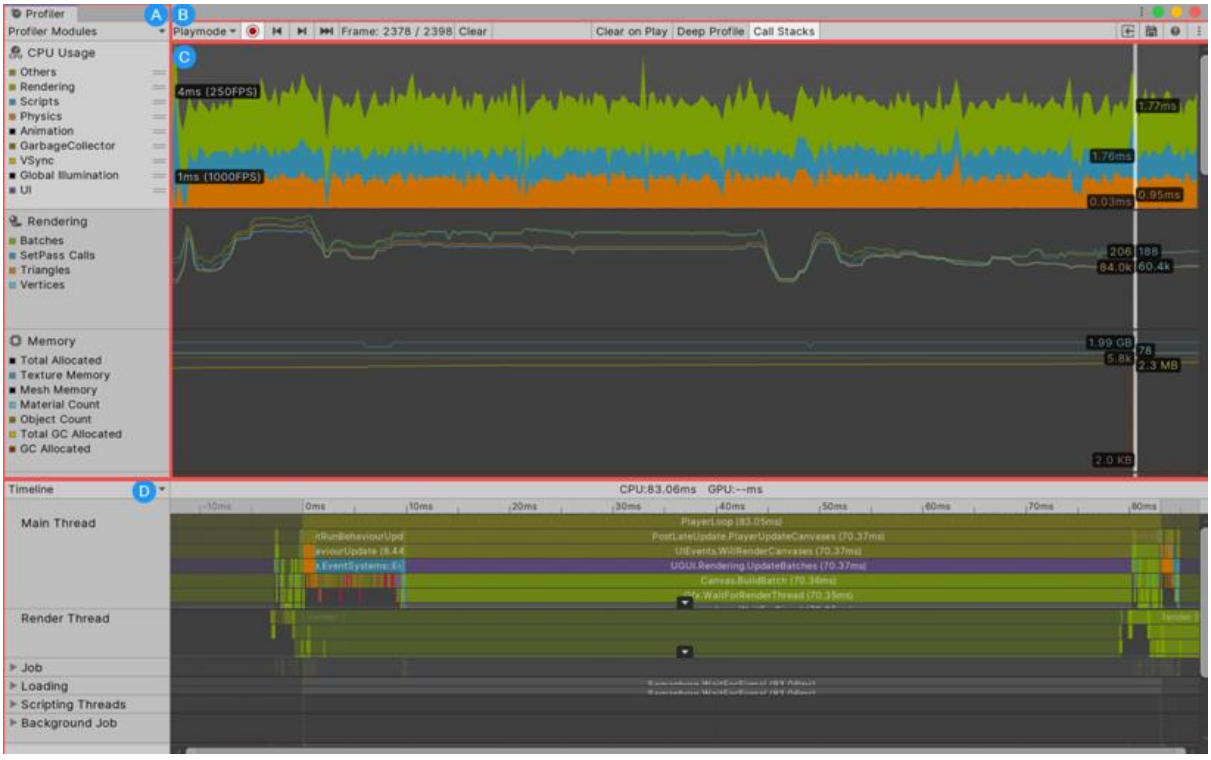

Рис. 2.1. Вікно Профайлера

2.1.2 Static batching

Це метод, який економить багато циклів CPU. Кожен раз, коли об'єкт рендериться, відбувається Draw Call - команда для CPU або GPU про те, що об'єкт повинен бути відрендериним. Unity запускає кілька викликів відтворення і накладає їх один на одного, це і формує сцену (рис.2.2).

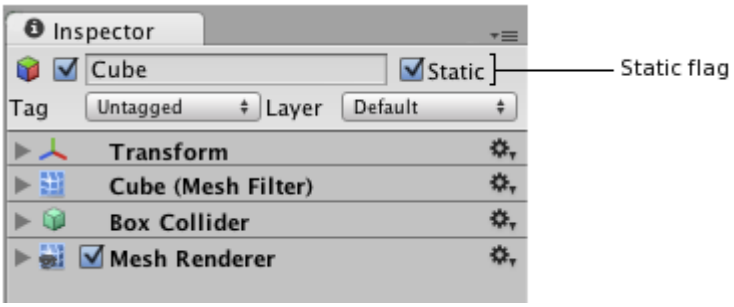

Рис. 2.2. Флаг вибору статичного батчингу

Однак кожен Draw Call вимагає ресурсів CPU (рис.2.3), тому виникає необхідність мінімізувати їх кількість.

Рис. 2.3. Моніторинг ресурсів

Чим більше об'єктів на екрані, тим більше draw calls, чим він більший, им сильніше навантаження.

GPU обробляє занадто багато вершин. Яка кількість вершин є нормальним, визначається GPU і набором вертексних шейдерів (рис.2.4).

2.1.3 Робота з Canvas

Основним елементом інтерфейсу користувача Unity є полотно. Воно несе відповідальність за покоління, сортування та надання сітки дочірніх елементів інтерфейсу. Всі елементи інтерфейсу повинні бути дочірньою компанією для будь-якого полотна, інакше вони не будуть відображатися в грі.

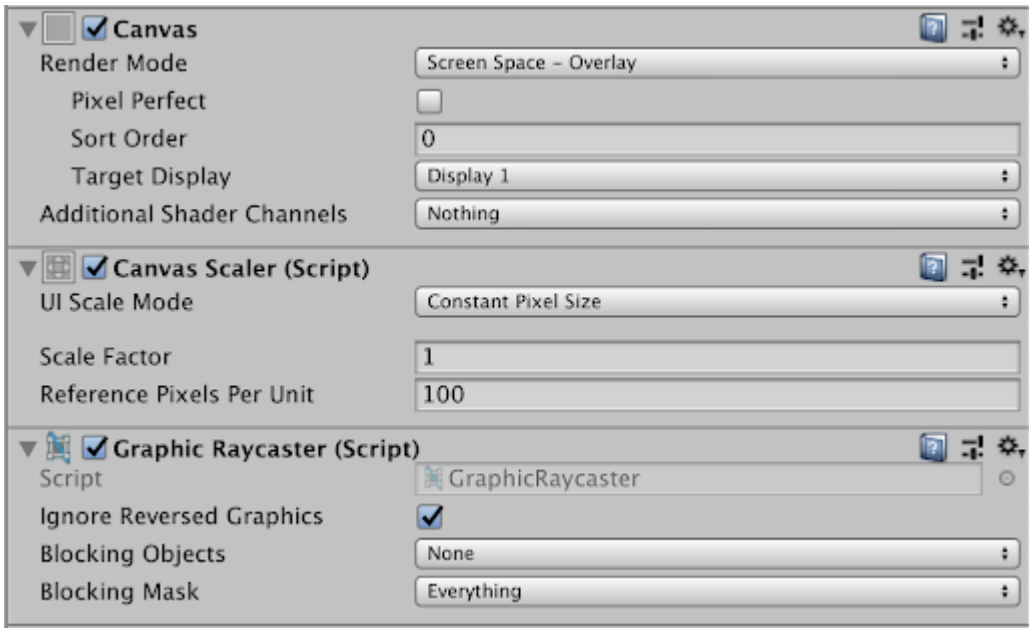

Рис. 2.4. Налаштування полотна у Інспекторі

Відтворення інтерфейсу походить від далекого до найбільш близького об'єкта з камери (назад до передньої) Transparent queue з альфа-змішуванням.

Окремо слід зазначити, що прозорість елементів інтерфейсу не впливає на продуктивність. Навіть якщо елемент повністю складається з "непрозорих" пікселів, він все одно буде намальований за допомогою альфа-змішування.

Всі пікселі всіх активних елементів обробляються. Це не залежить від того, чи їх видно, заблоковані іншими об'єктами або, як правило, повністю прозорі.

2.1.4 Перебудова інтерфейсу користувача

Перебудова користувальницького інтерфейсу - це багатоступеневий процес, який включає в себе конструкцію мешів кожного елемента інтерфейсу та спроби батчингу цих мешів, щоб мінімізувати кількість відтворень (draw calls).

Перебудова відбувається в чотири етапи:

1) аналіз структур елементів;

2) перебудова усіх активних мешів, включаючи нульові елементи прозорості;

3) матеріали для батчингу мешів елементів перебудовуються;

4) всі елементи намальовані відповідно до їх місця у черзі.

Перебудоване полотно кешується і заново використовується, поки один з елементів у полотні не позначений як змінений.

Позначені модифіковані (dirty) об'єкти, які були активізовані або деактивовані; елементи із зміненим матеріалом, положенням, масштабом, поворотом; змінене значення тексту в текстовому компоненті; Виконано перепризначення батьківського елементу (parent) тощо.

У цьому випадку відбувається відновлення полотна, що містить щонайменше один змінений елемент. Правда, він застосовується лише до полотна, в якому розташований елемент. Тобто зміни в елементах дочірніх полотнах не впливають на парент-полотно.

Чим більше елементів на полотні, тим більше витрат на аналіз та сортування об'єктів.

2.1.5 Dynamic batching

Batching буває двох видів:

– статичний;

– динамічний.

Статичний дає кращу продуктивність, тому для вирішення поставленої задачі будемо використовувати його. Щоб ефективно застосовувати Static Batching, необхідно використовувати якомога менше різних матеріалів. Для цього потрібно скомбіновувати всі матеріали в одну велику текстуру.

Для не статичних об'єктів використовується Dynamic Batching. Об'єкти з Dynamic Batching вимагають певні ресурси на кожну вершину, тому він застосовується тільки до мешів, що містять менше 900 вершин.

Динамічний батчинг пов'язаний з додатковим навантаженням для кожної вершини, так що він застосовується лише до мешів, що в сумі містять менше 900 вершин.

Є деякі особливості, які потрібно враховувати під час такого методу:

1) якщо шейдер використовує Vertex Position, Normal і єдиний UV, то можна батчити до 300 вершин; тоді як якщо шейдер використовує Vertex Position, Normal, UV0, UV1 і Tangent, то тільки 180 вершин;

2) рівномірно масштабовані об'єкти не будуть батчитись з нерівномірно масштабованими об’єктами;

3) використання різних матеріалів призведе до збою батчинга;

4) об'єкти з *Lightmap* мають додатковий (прихований) параметр матеріалу: зміщення / масштаб в Lightmap, тому об'єкти з Lightmap НЕ будуть батчитись.

Lightmap (в перекладі з англ. - карта освітлення) - метод освітлення простору в 3D-додатках, що полягає в тому, що створюється текстура, що містить інформацію про освітленості тривимірних моделей (рис. 2.5).

Цей метод значно економить ресурси системи, так як тривимірному візуалізатору не доводиться розраховувати падіння світла в режимі реального часу, але при цьому *Lightmap* програє динамічному освітленню в реалістичності.

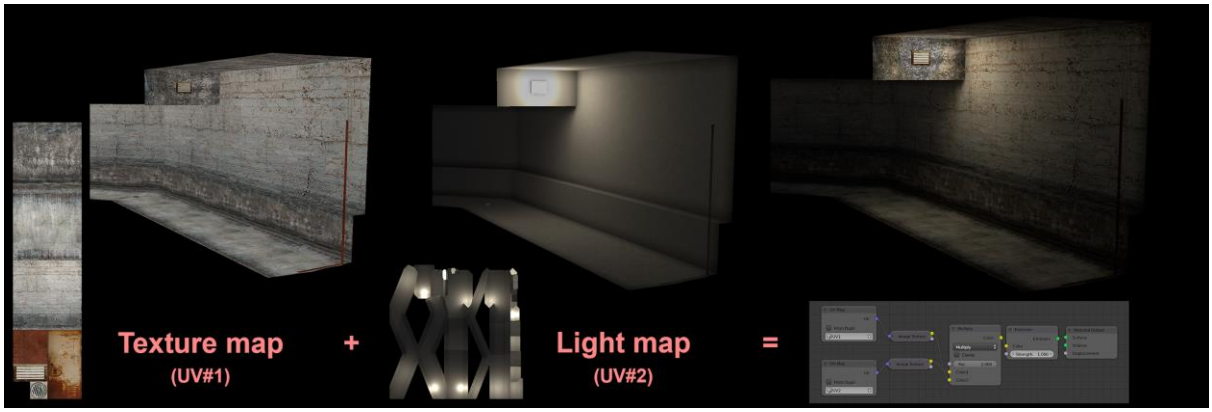

Рис. 2.5. Приклад використання лайтмапу

2.1.6 Виключення невидимих об’єктів

Якщо елемент заблоковано непрозорим елементом, то потрібно вимкнути GameObject і GameObject батьківського елемента. У той же час, елементи

інтерфейсу, в яких альфа встановлюється в 0, все ще буде намальовано. Такі об'єкти повинні включати в себе Cull Transparent Mesh у компоненті Canvas Renderer або просто вимкнути невидимі об'єкти.

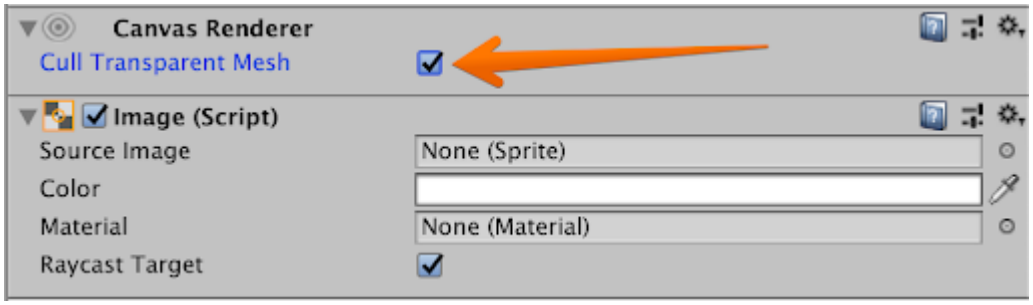

Рис. 2.6 Активація Cull Transparent Mesh

2.1.7 Відключення об'єктів світу, прихованих непрозорим інтерфейсом

Якщо інтерфейс перекриває не весь світ, можна зберегти його в Render Texture, і вимкнути глобальну камеру.

2.1.8 Мінімізація кількості пікселів для малювання

Необхідно поєднувати декілька зображень в одному. Наприклад, має сенс робити кнопки одним спрайтом, а не окремими шарами з підкладкою, тінню, обведенням, тощо. Це зменшить гнучкість роботи з такими елементами і може призвести до засмічення ресурсів, тому необхідно шукати компроміс.

2.1.9 Уникайте використання Camera.main

Проблема: Полотна World Space повинні знати, з якої камери мають відбуватися події взаємодії.

Під час налаштування Canvas для відтворення або у World Space, або в просторі екрана камери, можна вказати камеру, яка буде використовуватися для створення подій взаємодії для Graphic Raycaster інтерфейсу користувача. Цей параметр необхідний для полотен «Screen Space - Camera» і називається «Render Camera».

2.2 Моделі для вирішення поставленої задачі

2.2.1 GPU: стиснення текстур і міпмапи

Використання стислих текстур зменшує розмір атласів (в результаті вони швидше завантажуються і займають менше пам'яті), а також це може значно підвищити продуктивність. Стислі текстури використовують малу частину пропускної здатності пам'яті, в порівнянні з 32-бітними RGBA-текстурами.

Використання міпмап для текстур

Як правило, параметр імпорту Generate Mip Maps включений для текстур, які використовуються в 3D-сцені. В цьому випадку стиснення текстур допоможе обмежити кількість текстурних даних, що транспортуються в GPU при візуалізації. Міпапи дозволяють GPU використовувати для маленьких трикутників текстури зниженого дозволу.

Є виключення з цього правила: коли один тексель (піксель текстури) відповідає одному пікселю екрану, що зустрічається в елементах користувальницького інтерфейсу і в 2D-іграх.

Бувають ситуації, коли на Android і iOS одні і ті ж шейдери можуть працювати по різному чи не працювати взагалі.

2.2.2 Оптимізація текстур

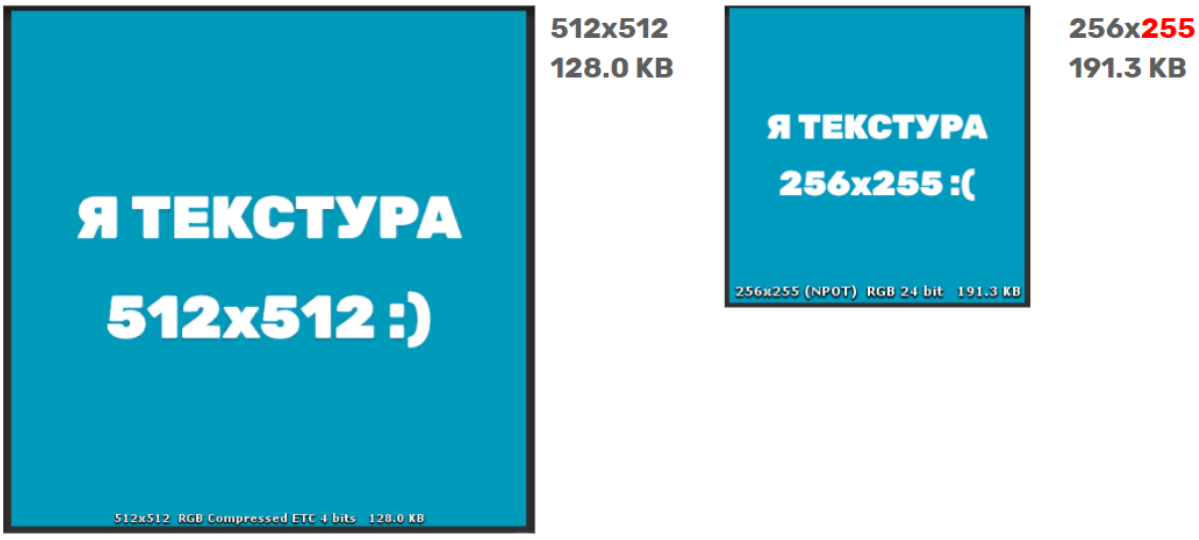

Рис. 2.7. Різниця у розмірі файлів

По-перше, це дозволяє включати у властивостях текстури компресії і зменшувати її вагу. А по-друге, такі текстури легше ділити на однакового розміру кадри, якщо створювати анімовані текстури. По суті, це послідовність кадрів, яка йде зліва направо і зверху вниз. Так само, як ми читаємо будь-який текст.

2.2.3 Атласи

Атлас - це різновид ресурсів, що поєднує в собі кілька текстур. Вони дозволяють зменшити кількість креслень (draw calls) та підвищення продуктивності. Текстури у атласі займають менший об’єм пам’яті та швидше завантажуються на полотно.

Створення атласу.

Створення Atlas: Asset> Create> Sprite Atlas.

Вибераємо атлас і завантажуємо необхідний спрайт у Objects for Packing для упаковки (рис. 2.8).

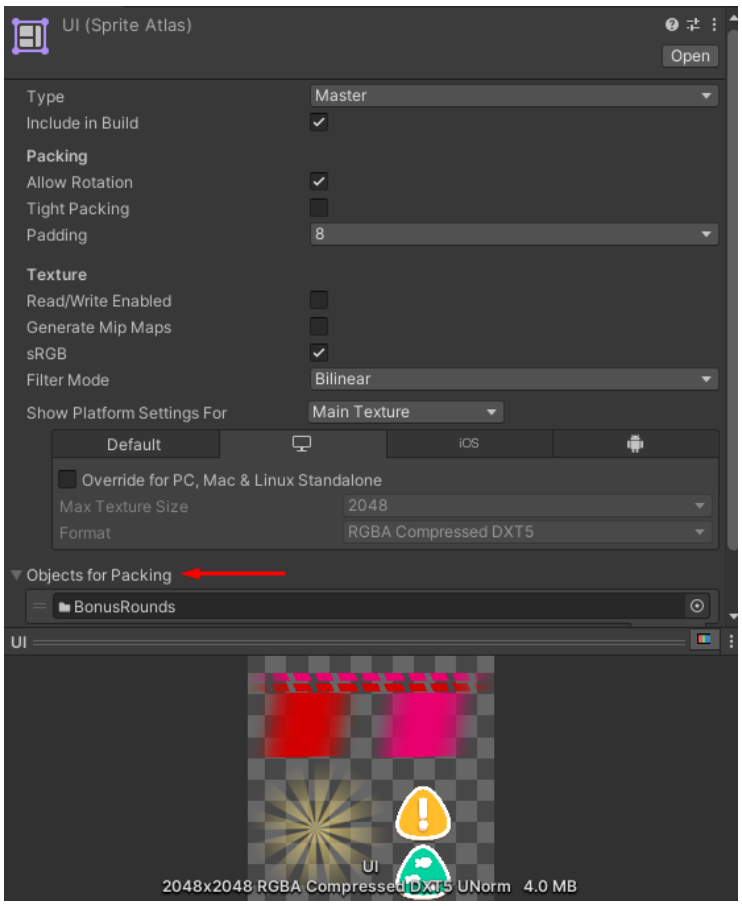

Рис. 2.8. Пакування атласу

Слід мати на увазі, що навіть якщо на сцені використовуються один або декілька спрайтів з атласу, то атлас все одно буде повністю завантажено. Тому не має сенсу намагатися ввести всі зображення в один гігантський атлас, який на пристроях з невеликою операційною пам'яттю може займати відчутну частину. Краще розділити на деякі менші атласи, наприклад, на користувальницькому інтерфейсі Atlas в меню гри та Atlas режиму гри.

Треба уникати великих порожніх місць у атласі, щоб не зайняти додаткове місце в пам'яті. Для цього можна змінити розмір атласу або додати додаткові зображення, щоб максимально зменшити порожній простір.

Для зображень, які не потрапили в атлас, потрібно вибрати правильні налаштування.

Кожен формат стиснення має вимоги, за умови, що він буде ефективно працювати. Найчастіші вимоги до зображень:

• Ширина та висота повинні бути кратними 2;

• Ширина і висота повинні бути кратними 4;

• обидва пункти.

В іншому випадку текстура не буде стиснутою, з’являться додаткові витрати на пам'ять.

При виборі стиснення, Unity дає підказки, якщо будь-які вимоги до вибраного формату не виконуються.

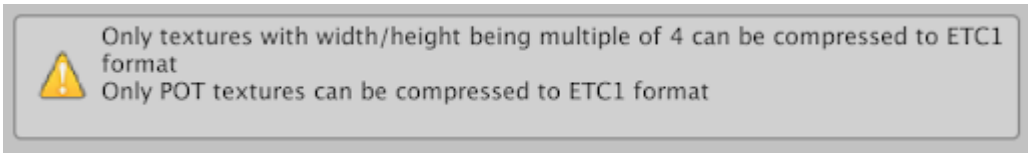

Рис. 2.9. Підказка редагування обраного формату

Також необхідно пам’ятати, що формати відрізняються від цільового пристрою. Додаткова інформація та рекомендації щодо форматів можна перевірити в офіційній документації.

2.2.4 Оптимізація матеріалів і шейдерів

У Unity 2018 було анонсовано вбудований аналог відомого та популярного ассету Shader Forge. Шейдери можна збирати, як конструктор, набираючи потрібні функції і налаштовуючи їх без особливих знань в програмуванні.

Що ж стосується вбудованих дефолтних (стандартних) шейдерів, то краще за все використовувати шейдери mobile> particles і інші з вкладки mobile при роботі з ефектами для мобільних пристроїв. Для демонстрації прикладів використовую не мобільний шейдер Particles> Multiply (Double).

Немає необхідності використовувати 3 або більше матеріалів. При дійсної необхідності – можна додавати новий матеріал. Але намагайтеся групувати текстури в атласи, як я згадував вище і призначати все це на один матеріал.

Наприклад, цілком можна зробити атлас, де будуть кілька варіантів гало з різною прозорістю та інтенсивністю «світіння» і, використавши один матеріал, вже в самій ПС викликати потрібний кадр.

Так само нерідкі випадки конфліктів ресурсів проекту та подібних «конструкторів». Адже автори роблять їх, щоб продавати, а не щоб вони працювали у всіх ідеально. Часто вдається обійтися добре зробленої ілюзією на дефолтних шейдерах. Часто цікавий результат виходить випадково. І вже на його основі можна зробити цікавий ефект.

2.2.5 Зменшуємо кількість Layout Group

Layout групи використовуються для організації дочірніх елементів інтерфейсу всередині контейнера. Є 3 види груп: вертикальна, горизонтальна та сітка.

Наприклад, вертикальний макет/група (Vertical Layout Group) поміщає елементи макета дочірніх елементів один на одного. Їх висоти визначаються їхніми мінімальними, кращими та гнучкими висотами за такою моделлю:

– мінімальна висота всіх елементів макета дочірніх ел-тів додаються разом, а інтервал між ними додають також. Результатом є мінімальна висота вертикальної групи макетів;

– додаються висоти, що задані користувачем як бажані. Результат є бажана висота вертикальної групи макетів;

– якщо вертикальна група приймає розмір мінімальної або менше, всі дочірні елементи також матимуть мінімальну висоту;

– чим ближче вертикальна група до бажаної висоти, тим ближче кожен дочірній елемент до його заданої бажаної висоти;

– якщо вертикальна група вище, ніж його бажана висота, вона збільшується вертикально пропорційно до висот дочірніх елементів.

Отже, суть проблеми полягає у наступному: кожен елемент інтерфейсу, який вміщує лейаут групу, буде виконувати принаймні один виклик GetComponents.

Коли один або більше дочірніх елементів змінюється, група макету стає «забрудненою». Змінені дочірні елемент(-и) стають недійсними щодо групи макету, яка ним володіє.

Кожен елемент інтерфейсу, який є частиною макету, але позначається рушієм як брудний, буде викликати, як мінімум, один GetComponents. Цей виклик шукає дійсну групу макета на паренті елементів макета. Якщо він знаходить один, він продовжує пошук по відношенню до ієрархії перетворення, доки вона не зупинить пошук груп макетів або досягне кореня ієрархії - залежно від того, що станеться першим. Тому кожна група макетів додає один GetComponents виклик до процесу відтворення лейаут групи на полотні, що робить вкладені групи макетів надзвичайно «важкими» для продуктивності.

Рішення: уникнення груп макетів, де це можливо.

Раціональним рішенням є якорі для пропорційних макетів. На массивному UI з динамічним числом елементів, необхідне написання власного коду, який буде розраховувати макети перед їх беспосереднім відтворенням на Canvas. На власній

практиці був досвід використання подібного скрипту для розрахування висоти ігрового вікна через його насичену структуру і велику кількість груп макетів.

2.2.3 Відключення полотен для зменшення навантаження на GPU

Іноді виникає необхідність приховати деякі елементи інтерфейсу та полотна. Найбільш ефективним рішенням буде вимкнення самого компоненту Canvas.

Відключення компонента Canvas зупинить ріст викликів (draw calls) на GPU, тому саме полотно більше не буде видно. Однак полотно не змінить свій вертексний буфер; Воно збереже всі його сітки та вершини, і коли полотно буде знову ввімкненим, не буде запущено процесс реструктуризації, замість цього лише буде запущено відтворення (відмалювання) полотна.

Крім того, відключення компонента Canvas не викликає перенавантаження зворотних викликів на ієрархії полотна. Однак, користуючись цим пунктом для оптимізації, необхідно слідкувати за дочірніми елементами і не виключати їх окремо від полотна водночас.

2.2.4 Скриптування модулів

1) Доступ до модулів

Щоб максимально оптимізувати роботу потрібно дописувати вручну чи змінювати певні властивості до вже існуючих модулів. Наприклад, як виглядає зміна властивості “швидкість” з модуля Emission (рис. 2.11).

```
using UnityEngine;

public class AccessingAParticleSystemModule : MonoBehaviour
{
    // Використовуємо це для ініціалізації
    void Start ()
    {
        // Отримуємо доступ до модуля Emission
        var forceModule = GetComponent<ParticleSystem>().forceOverLifetime;

        Ставимо значення
        forceModule.enabled = true;
        forceModule.x = new ParticleSystem.MinMaxCurve(15.0f);
    }
}
```

Рис. 2.10. Код зміни властивості

Тобто ми захоплюємо структуру та встановлюємо її значення, але ніколи не присвоюємо її назад ПС. ПС дізнається про цю зміну завдяки інтерфейсу.

Висновки до розділу 2

У даному розділі були описані методи та інформаційні технології для вирішення поставленої задачі.

Були описані такі засоби як Perfomance Profiler, Dynamic та Static Batching. Моделі, що були застосовані для оптимізації, і є ключовими: оптимізація текстур, матеріалів та шейдерів.

Також для оптимізації ПС, для більш тонкої настройки модулів є ефективним їх скриптування.

3 МОДЕЛЮВАННЯ ОТРИМАННИХ РЕЗУЛЬТАТІВ

3.1 Профілювання Unity UI

3.1.1 Аналіз впливу навантаження UI на GPU та CPU

Завдяки методам описаним раніше інтерфейс стає настільки оптимізований, що він ледве впливає на таймінги GPU. Реалізовані методи затемнення непрозорого UI компенсували більшу частину перевідтвореннь, викликаних накладенням UI.

На даному етапі необхідно підключення Unity Profiler аби виявити вплив інтерфейсу на графічний та центральний процессори.

Під час тестування було виявлено, що перевантажений ЦП витрачає у кожному кадрі більше 1 мс на рендеринг UI. Це велика кількість часу для платформи, яка дає бюджет у 13 мс, щоб виконати всю гру: фізика, логіка, 3D-рендеринг, вхід та мережевий код.

Рис. 3.1. Зайві батчі, що навантажують процесор

Висновок: UI можна оптимізувати під GPU, але це не означає, що інтерфейс оптимізований до навантажень ЦП. Це відбувається тому що при рендерингу Unity UI задачі ЦП та GPU дуже відрізняються.

Подальше профілювання Unity UI знайшло очевидну проблему: UI постійно відтворювався у кожному кадрі, тобто у кожному кадрі відбувався процес перебудови Canvas (Canvas Rebuild).

Постійне навантаження на ЦП у 1 мс. На перший погляд здається, що Unity не кешує Canvas інтерфейсу. Але це не так. Насправді Unity ефективно кешує його аби вони збирались тільки один раз. Проблема з’являється тільки тоді, коли відбуваються зміни властивостей будь-якого з елементів UI в Canvas – колір,

позицію, і тд. Отже, усі анімації або ж ефекти при натисканні кнопок «вбивають» продуктивність.

Коли відбувається зміна властивості UI, Unity виконує Canvas Rebuild, що погіршує продуктивність гри.

Canvas Rebuild спонукає рушій Unity ітераційно обходити усі елементи UI цього полотна, щоб створити оптимізований список викликів відтворення (набори вершин, кольорів, матеріалів тощо). І на Canvas Rebuild витрачається велика кількість часу.

3.2 Початок профілювання Unity UI

Для перевірки створюємо простий UI. Нехай є набір із елементів, що контролюється сіткою (Grid Layout Group).

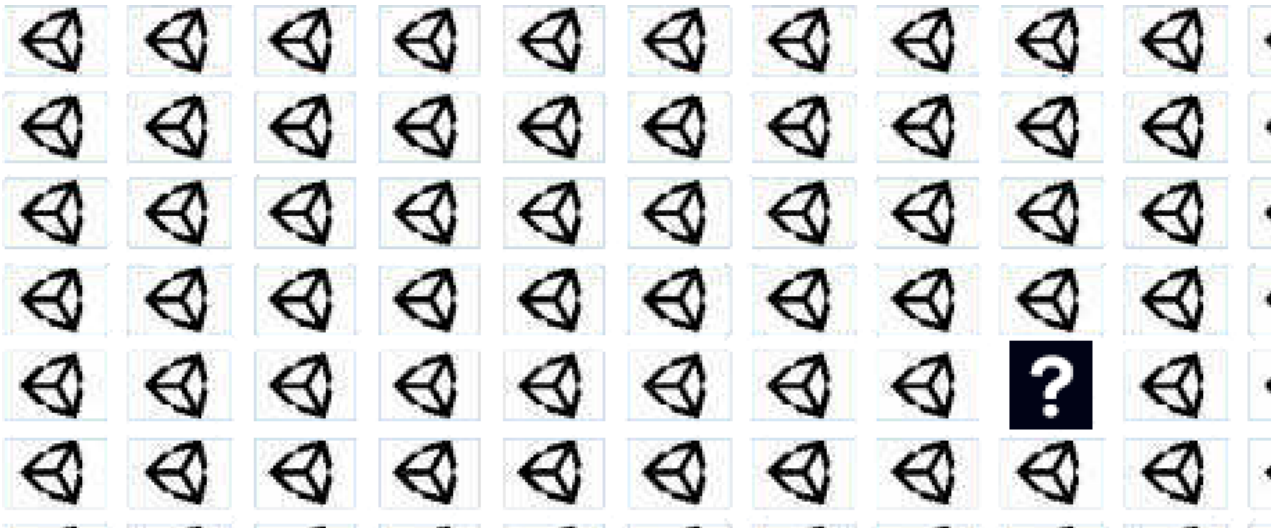

Рис. 3.2. Приклад профілювання Unity UI

Інтерфейс відтворюватиметься нормально, навіть незважаючи на 350 зображеннь у контейнері. У стандартному випадку інтерфейс буде відтворюватися лише за 2 виклики (draw call), оскільки існує лише два унікальних зображення, які не є в Sprites Atlas.

Фактично, в профайлері можна побачити, що з боку ЦП практично немає навантаження. Більшу частину часу на інтерфейс витрачається менше 0,01 мс.

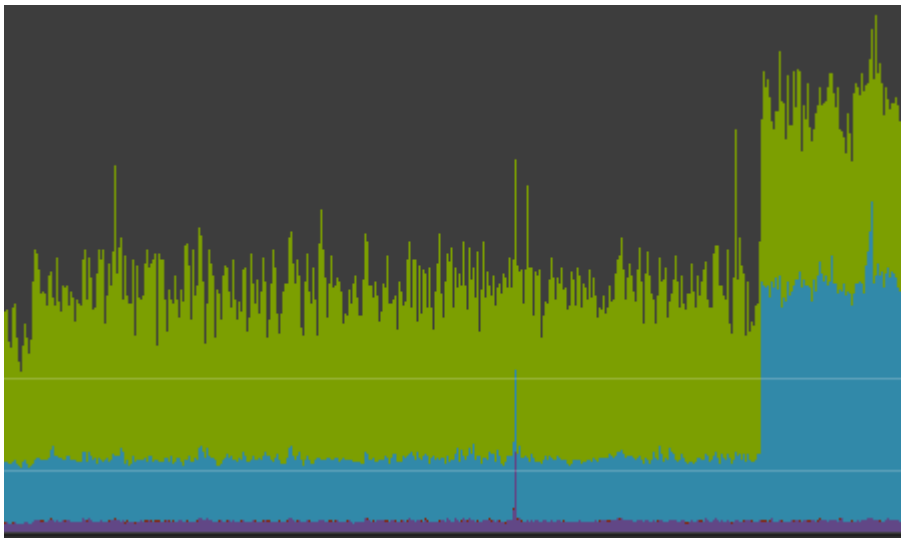

Рис. 3.3. Загадковий сплеск у кінці графіка

Але, як видно із графіку, з’явився невідомий сплеск ресурсів ЦП

3.2.1. Робота із Canvas Rebuild

Отже, за секунду, витрати процесора зросли майже у двічі.

Для наглядності розглянемо два приклади: оптимізований Canvas та Canvas Rebuild.

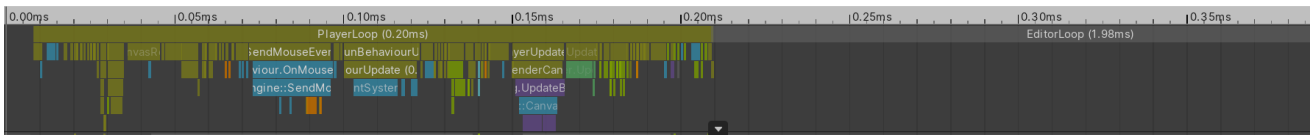

Рис. 3.4. Оптимізований Canvas

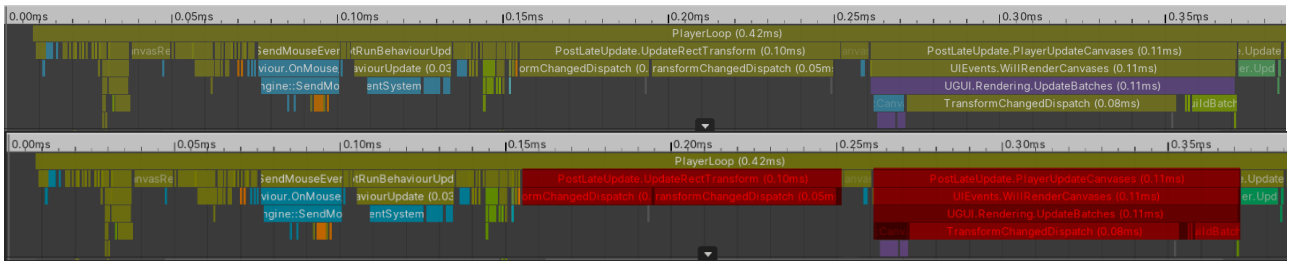

Рис. 3.5. Витрати ресурсів системи на Canvas Rebuild

PostLateUpdate.UpdateRectTransform та UGUI.Rendering.UpdateBatches – саме ці процеси викликають перебудову.

Що роблять ці області?

Перша, UpdaterRectRansform, дає зрозуміти, що змінився певний transform об’єкта, і тому рушій повинен виконувати «важку» логіку, щоб відобразити візуальні зміни. Ми не знаємо, чи це була змінена позиція, обертання, масштаб або інші властивості RectRansform.

Другий фрагмент витрат UpdateBatches пов'язаний з тим, що необхідно відновити всю геометрію полотна. Саме цей процес називається Canvas Rebuild. Реструктуризація полотна означає, що Unity оновлює всю ієрархію полотна, для генерації списку викликів для відтворення. Розраховуються вершини, індекси, кольори, uv всіх елементів, а потім відбувається прохід батчингу, щоб поєднати найбільшу кількість викликів відтворення, щоб зменшити надмірне навантаження на центральний процесор, що передає його до графічного драйвера.

Отже тепер, коли відомо що відбувається, необхідно запобігти заходів щодо уникнення перебудови полотна.

Висновки щодо процесу реструктуризації:

– зміна атрибута в елементі UI позначає сам елемент як "брудний";

– елемент UI може бути повністю брудним, але може бути частково: брудні вершини, брудне розташування, брудні матеріали. Від частково брудних станів, відновитися легше;

– після того, як будь-які елементи полотна позначені як брудні, Unity оновлюється повністю;

– реструктуризація полотна є затратною для ЦП, тому важливо уникнути цього.

3.2.2. Пошук та вирішення проблеми

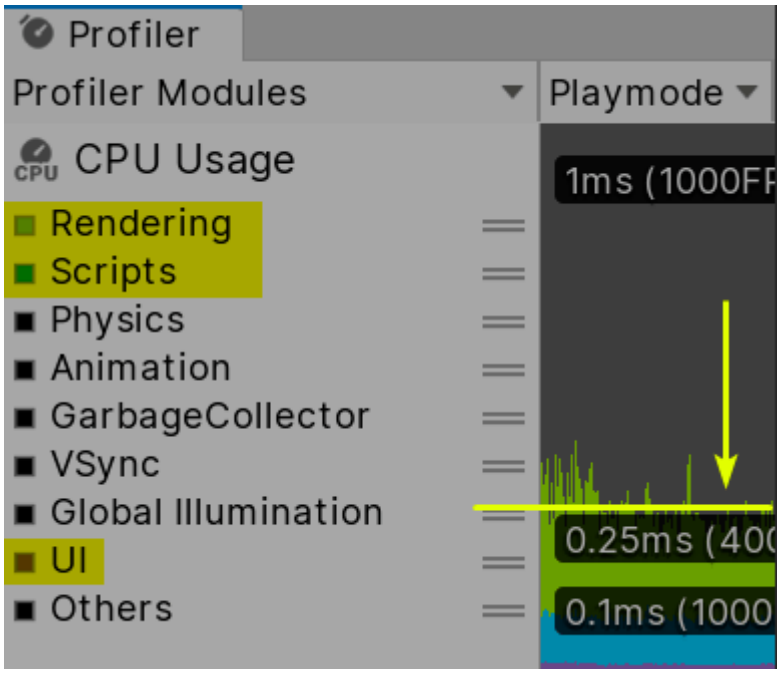

Рис. 3.6. Аналізування метрик профайлера

Відфільтруємо метрики, щоб можна було зосередитись на найважливішому: рендерингу, скриптах та UI.

Відстежуйте вихідну позначку, щоб розуміти витрати на поточну схему, до якої повинні входити і Canvas Rebuild.

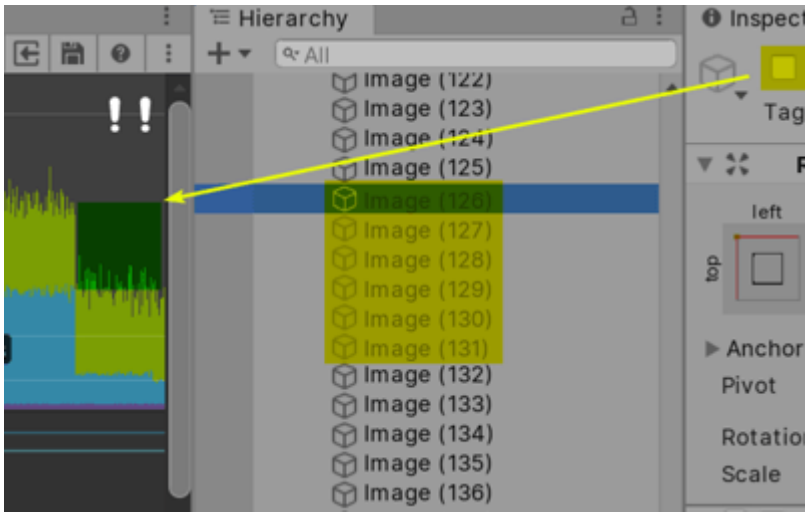

Рис. 3.7. Приклад зменшення навантаження

Наступним кроком є деактивація GameObject із UI та порівняння результатів:

1) виберіть групу game objects та деактивуйте їх;

2) порівняйте показники продуктивності.

Якщо показання сильно не покращилися, продовжуйте деактивувати game objects, поки не побачите значне поліпшення.

Далі, необхідно знайти, що саме змінює його властивості.

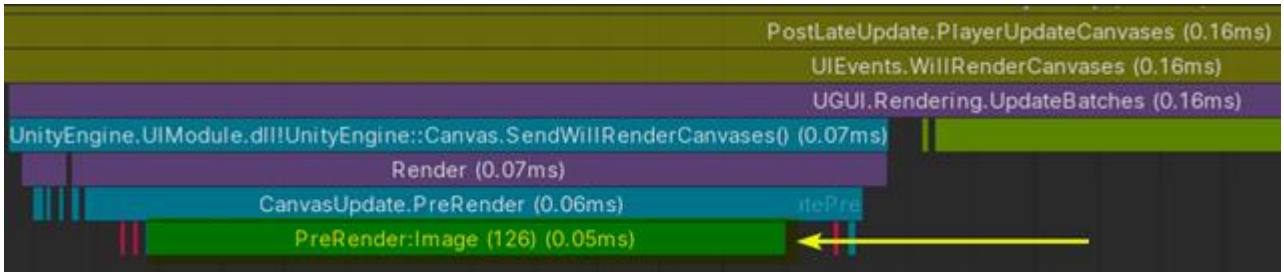

Рис. 3.8. Знайдений елемент, що змінює властивості

3.3 Технічна документація

Для оптимізації арту використовуємо наступні параметри:

1) тип текстури Sprite (2D and UI) (рис. 3.9);

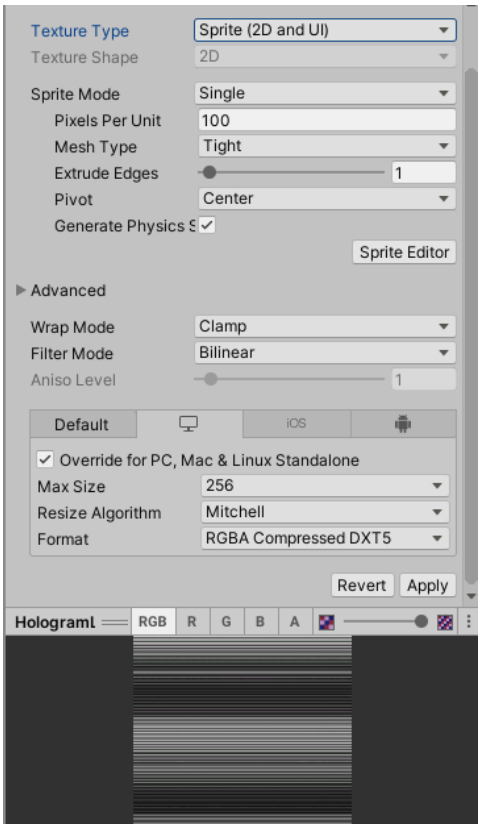

Рис. 3.9. Вікно «Інспектора» налаштувань текстури

Текстуру такого типу було використано для демонстрації ефекту голограми. Цей тип дозволяє робити особливі налаштування для текстур, а саме:

– Sprite Mode – налаштування що дозволяє вибрати як витягти спрайт із зображення. У моєму випадку це режим Single. Цей режим дозволяє використовувати спрайт на сцені таким, яким і є саме зображення. Текстуру можна відрізати та відредагувати у режимі редактора, щоб додатково вдосконалити її. Також є можливість вибрати такі режими як: Multiple та Polygon. Режим Multiple (множина) доцільно використовувати, якщо вхідне зображення містить декілька елементів в одному (наприклад атлас). Режим Polygon вибирають для відсікання текстури спрайту відповідно до мешу.

– Pixels Per Unity – налаштування, що дозволяє встановити кількість пікселів ширини / висоти в зображенні спрайту, які відповідають одній одиниці відстані у світовому просторі. За замовчуванням залишаю 100.

– Mesh Type – у цьому налаштуванні необхідно визначити тип мешу, який потрібно створити графічному візуалізатору. За замовчуванням це режим Tight. У цьому режимі меш створюється на основі піксельного значення Alpha, тобто меш повністю відповідає формі спрайту.
Також у цьому налаштуванні є режим Full Rect. Це значення необхідно для створення чотиристороннього багатокутника для відображення спрайту на ньому.

– Extrude Edges – налаштування, що дозволяє за допомогою повзунка визначити область, яку потрібно залишити навколо спрайту.

– Pivot – параметр, що задає точку опори.

Та багато інших налаштувань.

2) вимкнення невидимих об'єктів. Якщо елемент перекритий непрозорим елементом, то потрібно відключити його GameObject або батьківський GameObject елемента, що перекривається. При цьому елементи інтерфейсу, у яких альфа виставлена у 0, все одно будуть відмальовані. У таких об'єктах потрібно включити Cull Transparent Mesh у Canvas Renderer або просто вимкнути невидимі об'єкти;

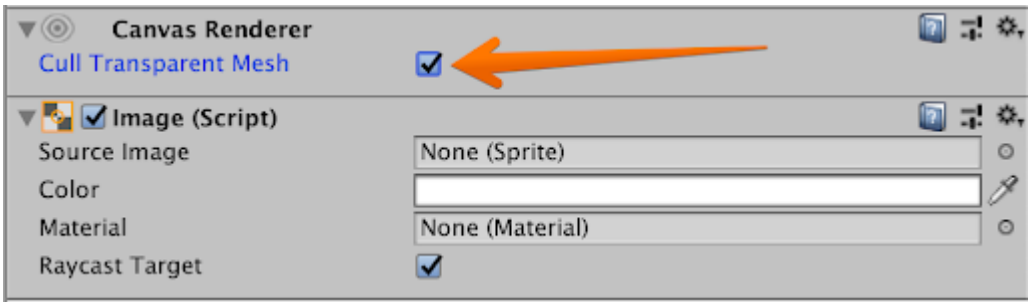

Рис. 3.10. Вимкнення Cull Transparent Mesh

3) вимкнення об'єктів світу, прихованих непрозорим інтерфейсом. Якщо інтерфейс перекриває не весь світ, можна зберегти його у Render Texture, а світову камеру вимкнути;

4) мінімізація кількості пікселів для відображення. Необхідно об'єднувати по можливості кілька зображень в одне. Наприклад, варто робити кнопки одним спрайтом, а не окремими шарами з підкладкою, обведенням, тілом кнопки і т.п. Це зменшить гнучкість роботи з такими елементами і може призвести до засмічення ресурсів, тому треба шукати компроміс;

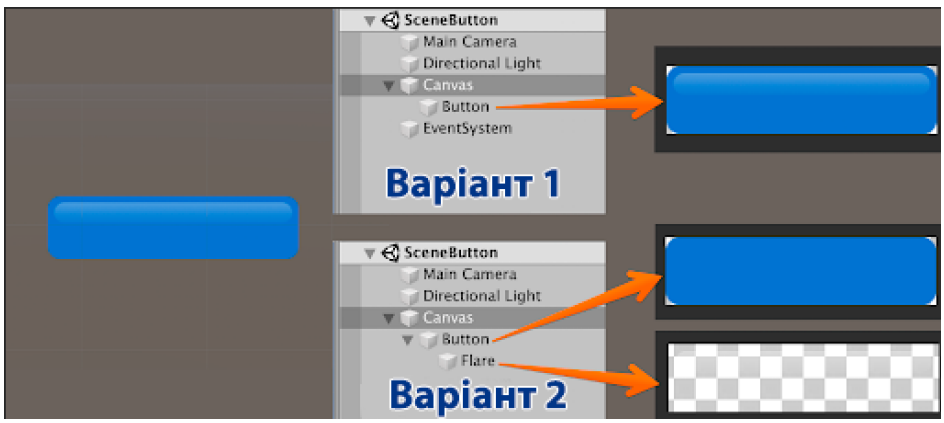

Рис. 3.11. Приклад використання спрайтів з об’єднаними елементами

5) уникайте порожніх елементів, які служать лише для організації структури (не використовуйте елементи як назви папок в ієрархії документа);

6) уникайте перетину об'єктів, які не можуть бути «запеченими» один з одним. Якщо це можливо, краще змінити положення в ієрархії, розмір контейнера або положення перекриття елементів;

7) використовуйте окремі або вкладені полотна для динамічних елементів. Таким чином, ви мінімізуєте витрати на сортування та реструктуризацію структури полотна, що містить велику кількість елементів. Вкладені полотна більш зручні, тому що легше успадкувують налаштування батьківського полотна. У той же час, коли змінюється батьківський Canvas, будуть перебудовані і всі вкладені. Це досить рідко, але виникає (наприклад, при зміні роздільної здатності екрана). Треба мати на увазі, що об'єкти з різних полотен або вкладеного полотна не запікаються для спільного відтворення. Рекомендується розділити полотна за регулярністю оновлення елементів. Статичні елементи повинні бути розміщені в окремому полотні, потім вони будуть намальовані лише один раз. Якщо є елементи, які постійно змінюються, то краще об'єднати їх на іншому полотні, тому що вони все ще будуть призводити до реструктуризації Canvas-у. Об'єкти, що змінюються, також можна розділити на кілька полотен за частотою оновлення. Наприклад, елементи, які оновлюються кожним кадром, покладаються на одне полотно, а елементи, які оновлюються рідше, в іншому;

8) вимкнення Pixel Perfect значно підвищить продуктивність. Особливо це стосується об'єктів, що постійно оновлюються з великою кількістю елементів (наприклад, інвентар із скроллом);

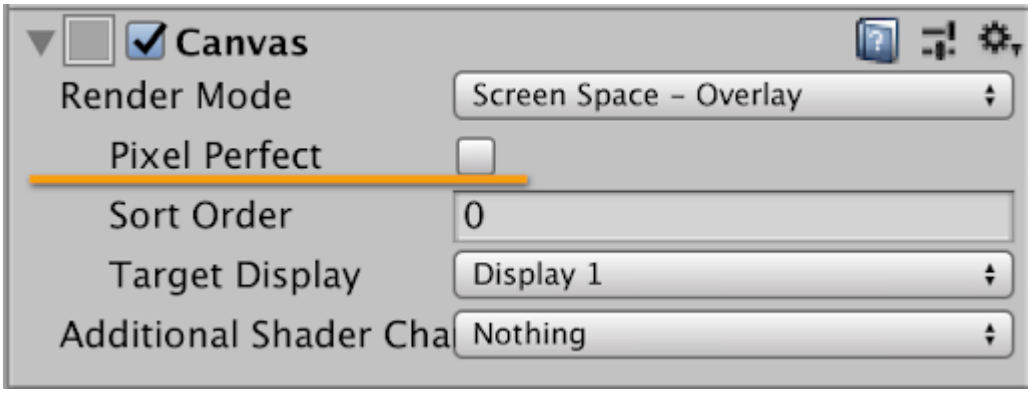

Рис. 3.12. На основному полотні

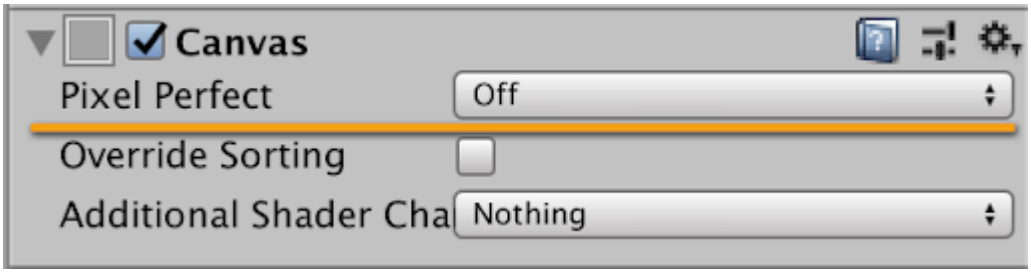

Рис. 3.13. На вкладенному полотні

9) якщо необхідно, вимкнути полотно, не відключайте об'єкт, що його містить (через функцію SetActive). Наступного разу, коли ви включаєте полотно, усі елементи будут позначені як змінені та відбудеться перебудова полотна. Краще вимкнути сам компонент полотна, тоді вся структура та запечені дані не зміняться, і наступного разу полотно просто починає відтворення;

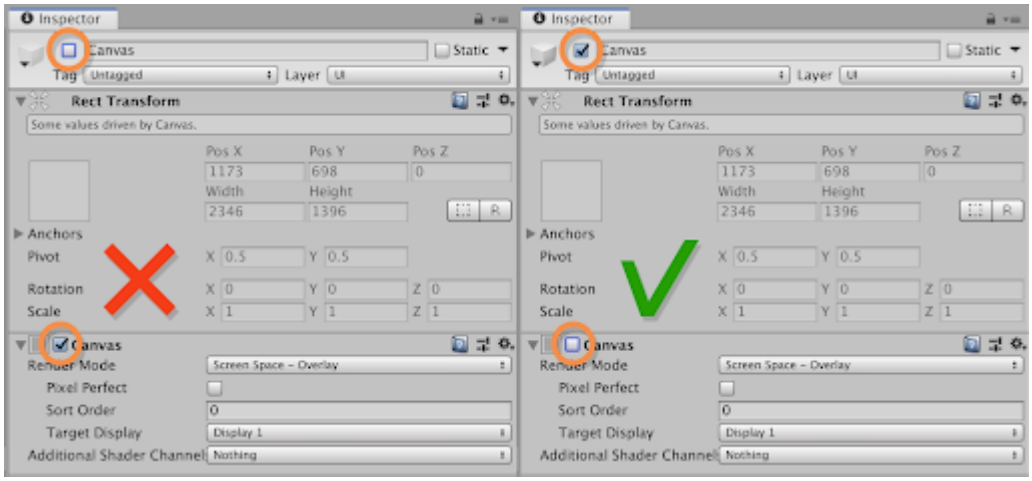

Рис. 3.14. Іллюстрація корректного відключення полотна

10) у полотнах, у яких у режимі Render Mode вибрано Screen Space — Camera або World Space, завжди встановлюйте камеру. Якщо її не встановити, то

система інтерфейсу користувача в кожному кадрі буде здійснювати її пошук через Object.FindObjectWithTag, щоб знайти Camera.main, а це позначиться на продуктивності;

11) щоб не створювати кілька однакових текстур, можна створити одну у відтінках сірого та “фарбувати” її через компонент Image, вибравши потрібний колір. Цей метод використовується на більшості проектів, над якими мені довелося попрацювати. Економія ресурсів атласу та фізичної пам’яті додатку/гри;

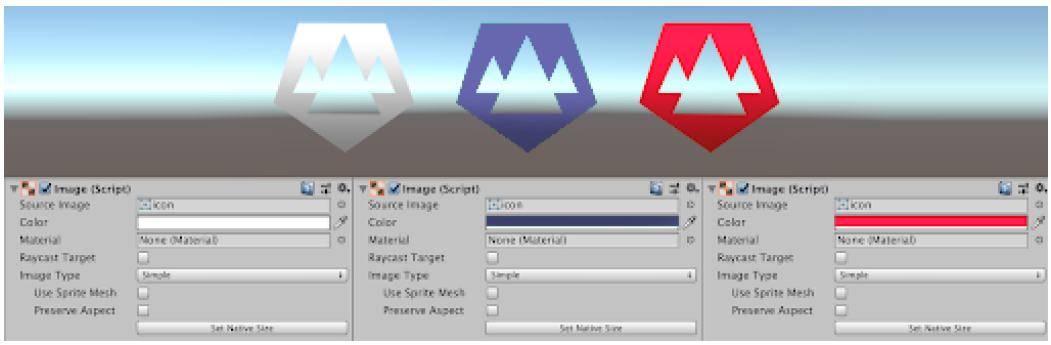

Рис. 3.15. Приклад застосування кольору у рушії на зображення

12) залишайте прапор Raycast Target тільки на елементах, що потребують подій, і знімайте у інших. За замовчуванням Raycast target увімкнено на багатьох елементах (Image, Text і т.д.). Це ускладнює та уповільнює роботу Raycaster-компонента, який обробляє події введення в UI Unity. При кліку або тапі він проходить по всій ієрархії елементів і шукає всі графічні компоненти з виставленим прапором Raycast Target, потім перевіряє їх на можливість подій введення та після успішного проходження перевірок додає до списку влучень. Після цього список влучень сортується за глибиною, відкидаються об'єкти, що знаходяться поза екраном. В результаті залишається остаточний список влучень;

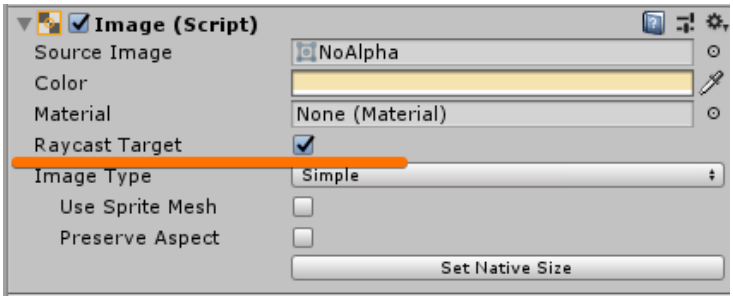

Рис. 3.16. Raycast Target у компонента зображення

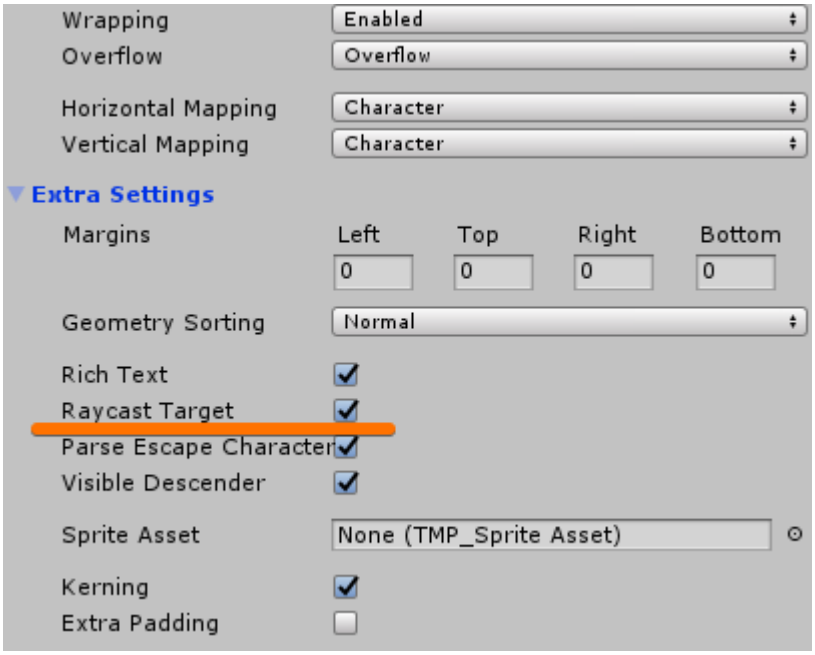

Рис. 3.17. Raycast Target у налаштуваннях TextMeshPro

Висновки до розділу 3

У розділі було детально описано скриптування модулів системи частинок. Описані такі етапи як: доступ до модулів, поняття лерпу та принципи його використання, властивості модуля MinMaxCurve та її режими.

Було продемонстровано різницю між оптимізованою та неоптимізованою кривими классу, а також зміна кривої на свою.

Після проведених тестів продуктивності було виявлено слабкі місця кожного із режимів, та час, що витрачається

Можна зробити висновки, що використання деяких режимів споживає більший ресурс системи черз додаткову вибірку значень, яка генерується самостіно при програванні ефекту

ВИСНОВКИ

Підсумовуючи результати магістерської роботи, перш за все хочеться виділити декілька моментів. Для виконання подібного завдання необхідно бути мультифункціональним професіоналом, тому що оптимізація інтерфейсу вміщує у собі два напрямки: технічний і художній, де можна максимально розкрити свій потенціал.

Оптимізація завжди про вибір, про пошук балансу між різними, а іноді навіть і суперечливими один одному рішеннями. Немає універсальної інструкції на «всі випадки життя», не можна брати кожен із зазначених методів без попереднього аналізу та впевненості, що ваші дії гарантовано призведуть до позитивного результату.

Не треба негайно сплющувати усю графіку і робити дуже мінімалістичний але оптимізований сірий квадрат на сірому фоні, відмовившись від приємної візуальної складової інтерфейсу. Тим не менш, кожен раз, поглиблюючись, з’являється можливість уникнути збільшення кількості проблем навіть на початку. Ситуації варіюються від проекту до проекту, від мети до мети, але загальні принципи завжди залишаються однаковими.

Розробники Unity, звичайно, також не сидять. Офіційні гайди на Unity Learn поповнюються хорошими статтями щодо оптимізації інтерфейсу, а нові версії рушія повільно, але виправляють проблеми старих. Судячи з останніх аноносів, Unity вже готує абсолютно новий уніфікований інструмент для редагування користувацьких інтерфейсів - на перший погляд, дуже нагадує CSS для веб-розробки. Нова система обіцяє повністю переробляти інтерфейс і значно підвищити продуктивність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інформаційний портал з матеріалами для створення ефектів. *RealTimeVFX* : веб-сайт. URL: https://realtimevfx.com/

2. VFX для наукових ігор. *80LV*: веб-сайт. URL:https://80.lv/articles/vfx/

3. Створення ефектів для великомасштабних проектів. *RenderRu* : веб-сайт. URL:https://render.ru/

4. Оптимізація VFX для ігрових платформ. *DTF*: веб-сайт. URL: https://dtf.ru/

5. Оптимізація Particle System. *Unity3D*: веб-сайт. URL: https://unity3d.com/

6. Програмування модулів ПС. *BlogsUnity* : веб-сайт. URL: https://blogs.unity3d.com

7. Види VFX та їх оптимізація. *Pinterest* : веб-сайт. URL: https://www.pinterest.ca

8. Jeffrey A. Okun, Susan Zwerman, The VES Handbook of Visual Effects: Industry Standard VFX Practices and Procedures : підручник. Каліфорнія, 2010, 922 с. URL: https://www.goodreads.com/book/show/7886959-the-ves-handbook-of-visual-effects

9. Ron Brinkmann, The Art and Science of Digital Compositing: Techniques for Visual Effects, Animation and Motion Graphics: підручник. Берлінгтон, 2008, 704 с. URL: https://www.goodreads.com/book/show/3368031-the-art-and-science-of-digital-compositing

10. Steve Wright, Digital Compositing for Film and Video: Production Workflows and Techniques: посібник. Лондон, 2017, 576 с. URL: https://www.goodreads.com/book/show/34703063-digital-compositing-for-film-and-video

11. Eran Dinur, The Filmmaker's Guide to Visual Effects: The Art and Techniques of VFX for Directors, Producers, Editors and Cinematographers:

підручник. Лондон, 2017, 205 с. URL: https://www.goodreads.com/book/show/34852894-the-filmmaker-s-guide-to-visual-effects

12. Charles Finance, Susan Zwerman, The Visual Effects Producer: Understanding the Art and Business of Vfx: підручник. Берлінгтон, 2009, 387 с. URL: https://www.goodreads.com/book/show/6852025-the-visual-effects-producer

13. Robh Ruppel, Graphic L.A.: підручник. Лос-Анджелес, 2014, 144 с. URL: https://www.goodreads.com/book/show/22115277-graphic-l-a

14. Ken A. Priebe, The Advanced Art of Stop-Motion Animation: посібник. Мічіган, 2011, 747 с.

15. Закон України «Про охорону навколишнього природного середовища» –К.: Україна. – 1991. – 59 с. (з усіма редакціями до 2017 року);

16. НПАОП 0.00-1.28-10 Правила охорони праці під час експлуатації електронно-обчислювальних машин/Зареєстровано в Міністерстві юстиції України 19 квітня 2010 р. за №293/17588;

17. Правила улаштування электроустановок. ПУЕ.– Харків.: Форт – 2011 –728 с.;

18. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу. Гігієнічні нормативи ГН 3.3.5-8-6.6.1 2002 р. Видання офіційне Київ, 2001 рік – 46 с.

19. ДБН В.2.5-67:2013. Опалення, вентиляція та кондиціювання. – К.: Мінрегіон України, 2013. – 147 с.;

20. ДБН.В.2.5 – 28-2006. Природне і штучне освітлення. – К.: Мінбуд України, - 2008 – 74 с.;

21. НАПБ Б.03.002 – 2007 Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою. Наказ МНС від 03.12.2007 №883;

22. ДСанПін 3.3.2.007– 98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. – К.: ГСЕУ України, 1998 – 21 с.;

23. ДБН В.1.1 – 7- 2002. Захист від пожежі. Пожежна безпека об’єктів будівництва. – К.: 2002. – 41 с.;

24. ДСТУ ISO14001 – 97 – 14012-97. Система управления окружающей средой – К.: Держстандарт України – 225 с.;

ДОДАТОК А

Програмний код для ініціалізації VFX у UI

```
#pragma kernel CSMain
#define NB_THREADS_PER_GROUP 64
#define HAS_ATTRIBUTES 1
#define VFX_PASSDEPTH_ACTUAL (0)
#define VFX_PASSDEPTH_MOTION_VECTOR (1)
#define VFX_PASSDEPTH_SELECTION (2)
#define VFX_USE_LIFETIME_CURRENT 1
#define VFX_USE_SEED_CURRENT 1
#define VFX_USE_POSITION_CURRENT 1
#define VFX_USE_DIRECTION_CURRENT 1
#define VFX_USE_VELOCITY_CURRENT 1
#define VFX_USE_ALIVE_CURRENT 1
#define VFX_USE_AGE_CURRENT 1
#define VFX_LOCAL_SPACE 1
#include
"Packages/com.unity.visualeffectgraph/Shaders/RenderPipeline/Universal/VFXDefines.hlsl"


CBUFFER_START(parameters)
    float3 ArcSphere_sphere_center_b;
    float A_a;
    float B_a;
    float ArcSphere_sphere_radius_b;
    float ArcSphere_arc_b;
```

```
    uint PADDING_0;
CBUFFER_END

struct Attributes
{
    float lifetime;
    uint seed;
    float3 position;
    float3 direction;
    float3 velocity;
    bool alive;
    float age;
};

struct SourceAttributes
{
};

#define USE_DEAD_LIST (VFX_USE_ALIVE_CURRENT && !HAS_STRIPS)

RWByteAddressBuffer attributeBuffer;
ByteAddressBuffer sourceAttributeBuffer;

CBUFFER_START(initParams)
#if !VFX_USE_SPAWNER_FROM_GPU
    uint nbSpawned;
spawned
```

```
    uint spawnIndex;
spawned
    uint dispatchWidth;
#else
    uint offsetInAdditionalOutput;
        uint nbMax;
#endif
        uint systemSeed;
CBUFFER_END

#if USE_DEAD_LIST
RWStructuredBuffer<uint> deadListIn;
ByteAddressBuffer deadListCount;
#endif

#if VFX_USE_SPAWNER_FROM_GPU
StructuredBuffer<uint> eventList;
ByteAddressBuffer inputAdditional;
#endif

#if HAS_STRIPS
RWBuffer<uint> stripDataBuffer;
#endif

#include
"Packages/com.unity.visualeffectgraph/Shaders/Common/VFXCommonCompute.hls
l"
#include "Packages/com.unity.visualeffectgraph/Shaders/VFXCommon.hlsl"
```

```
void SetAttribute_F01429A3(inout float lifetime, inout uint seed, float A, float B) /*attribute:lifetime Composition:Overwrite Source:Slot Random:Uniform channels:XYZ */
{
   lifetime = lerp(A,B,RAND);
}
void PositionSphere_0(inout float3 position, inout uint seed, inout float3 direction, float3 ArcSphere_sphere_center, float ArcSphere_sphere_radius, float ArcSphere_arc, float volumeFactor) /*positionMode:Surface spawnMode:Random */
{
   float cosPhi = 2.0f * RAND - 1.0f;float theta = ArcSphere_arc * RAND;
   float rNorm = pow(volumeFactor + (1 - volumeFactor) * RAND, 1.0f / 3.0f);

   float2 sincosTheta;
   sincos(theta, sincosTheta.x, sincosTheta.y);
   sincosTheta *= sqrt(1.0f - cosPhi * cosPhi);

   direction = float3(sincosTheta, cosPhi);
   position += direction * (rNorm * ArcSphere_sphere_radius) + ArcSphere_sphere_center;

}

#if HAS_STRIPS
bool GetParticleIndex(inout uint particleIndex, uint stripIndex)
```

```
    {
        uint relativeIndex;
        InterlockedAdd(STRIP_DATA(STRIP_NEXT_INDEX, stripIndex), 1, relativeIndex);
        if (relativeIndex >= PARTICLE_PER_STRIP_COUNT) // strip is full
        {
            InterlockedAdd(STRIP_DATA(STRIP_NEXT_INDEX, stripIndex), -1); // Remove previous increment
            return false;
        }

        particleIndex = stripIndex * PARTICLE_PER_STRIP_COUNT + ((STRIP_DATA(STRIP_FIRST_INDEX, stripIndex) + relativeIndex) % PARTICLE_PER_STRIP_COUNT);
       return true;
    }
    #endif
    [numthreads(NB_THREADS_PER_GROUP,1,1)]
    void CSMain(uint3 groupId          : SV_GroupID,
                uint3 groupThreadId    : SV_GroupThreadID)
    {
       uint id = groupThreadId.x + groupId.x * NB_THREADS_PER_GROUP;
    #if !VFX_USE_SPAWNER_FROM_GPU
       id += groupId.y * dispatchWidth * NB_THREADS_PER_GROUP;
    #endif
    #if VFX_USE_SPAWNER_FROM_GPU
       uint maxThreadId = inputAdditional.Load((offsetInAdditionalOutput * 2 + 0) << 2);
```

```
        uint currentSpawnIndex = inputAdditional.Load((offsetInAdditionalOutput * 2 + 1) << 2) - maxThreadId;
    #else
        uint maxThreadId = nbSpawned;
        uint currentSpawnIndex = spawnIndex;
    #endif

    #if USE_DEAD_LIST
        maxThreadId = min(maxThreadId, deadListCount.Load(0x0));
    #elif VFX_USE_SPAWNER_FROM_GPU
        maxThreadId = min(maxThreadId, nbMax); //otherwise, nbSpawned already clamped on CPU
    #endif

        if (id < maxThreadId)
        {
    #if VFX_USE_SPAWNER_FROM_GPU
            int sourceIndex = eventList[id];
    #endif
                    uint particleIndex = id + currentSpawnIndex;
    #if !VFX_USE_SPAWNER_FROM_GPU
            int sourceIndex = 0;
            /*//Loop with 1 iteration generate a wrong IL Assembly (and actually, useless code)
            uint currentSumSpawnCount = 0u;
            for (sourceIndex=0; sourceIndex<1; sourceIndex++)
            {
```

```
            currentSumSpawnCount                                    +=
uint(asfloat(sourceAttributeBuffer.Load((sourceIndex * 0x1 + 0x0) << 2)));
            if (id < currentSumSpawnCount)
            {
               break;
            }
         }
         */
      #endif
                  Attributes attributes = (Attributes)0;
                  SourceAttributes sourceAttributes = (SourceAttributes)0;
         attributes.lifetime = (float)1;
         attributes.seed = (uint)0;
         attributes.position = float3(0, 0, 0);
         attributes.direction = float3(0, 0, 1);
         attributes.velocity = float3(0, 0, 0);
         attributes.alive = (bool)true;
         attributes.age = (float)0;
      #if VFX_USE_PARTICLEID_CURRENT
          attributes.particleId = particleIndex;
      #endif
      #if VFX_USE_SEED_CURRENT
         attributes.seed = WangHash(particleIndex ^ systemSeed);
      #endif
      #if VFX_USE_SPAWNINDEX_CURRENT
         attributes.spawnIndex = id;
      #endif
      #if HAS_STRIPS
```

```
#if !VFX_USE_SPAWNER_FROM_GPU
#else
    uint stripIndex = sourceIndex;
#endif
            stripIndex = min(stripIndex, STRIP_COUNT);
    if (!GetParticleIndex(particleIndex, stripIndex))
      return;
    const StripData stripData = GetStripDataFromStripIndex(stripIndex, PARTICLE_PER_STRIP_COUNT);
            InitStripAttributes(particleIndex, attributes, stripData);
            // TODO Change seed to be sure we're deterministic on random with strip
#endif
    SetAttribute_F01429A3( /*inout */attributes.lifetime, /*inout */attributes.seed, A_a, B_a);
    {
      PositionSphere_0( /*inout */attributes.position, /*inout */attributes.seed, /*inout */attributes.direction, ArcSphere_sphere_center_b, ArcSphere_sphere_radius_b, ArcSphere_arc_b, (float)1);
    }

#if VFX_USE_ALIVE_CURRENT
    if (attributes.alive)
#endif
    {
#if USE_DEAD_LIST
        uint deadIndex = deadListIn.DecrementCounter();
      uint index = deadListIn[deadIndex];
```

```
        #else
                uint index = particleIndex;
        #endif
                attributeBuffer.Store((index * 0x1 + 0x0) <<
2,asuint(attributes.lifetime));
                attributeBuffer.Store3((index * 0x8 + 0xC380) <<
2,asuint(attributes.position));
                attributeBuffer.Store3((index * 0x8 + 0xC384) <<
2,asuint(attributes.velocity));
                attributeBuffer.Store((index * 0x8 + 0xC383) <<
2,uint(attributes.alive));
                attributeBuffer.Store((index * 0x8 + 0xC387) <<
2,asuint(attributes.age));
            }
        }
    }
```

ДОДАТОК Б

Програмний код оптимізації UI нотіфікацій

```
using System;
using GameDevWare.Serialization;

[TypeSerializer(typeof(ExtendedSaveItemSerializer<NotifierSaveData>))]
public class NotifierSaveData
{
   [ExtendedDataMember("nt", typeof(int))]  public NotifierType Type;
   [ExtendedDataMember("npk")] public string Key;
}

[Flags]
public enum NotifierType
{
   JOURNAL = (1 << 0),
   QUESTS = (1 << 1),
   SHOP = (1<<2),
   CUSTOMIZATIONSKININSIDE = (1<<3),
   CUSTOMIZATIONSKINOUTSIDE = (1<<4),
   PETS = (1<<5),
   HUMANSKINS = (1<<6),
   ACHIEVEMENTS = (1<<7),
   STORYPHOTOS = (1<<8)
}
using GameDevWare.Serialization;
using System;
using System.Collections.Generic;

[TypeSerializer(typeof(ExtendedSaveItemSerializer<UINotificationBlock>))]
public class UINotificationBlock
{
   [ExtendedDataMember("nsd")] public List<NotifierSaveData> NotifierSaveDatas { get; private
set; } = new List<NotifierSaveData>();

   //public event Action<NotifierSaveData> OnAdded;
   public event Action OnRemoved;

   public int GetNotifierCountByType(NotifierType notifierType)
   {
      var data = NotifierSaveDatas.FindAll(x => (notifierType & x.Type) != 0);
      return data.Count;
   }
   public bool IsNotifierAvailable(string key)
   {
      for (int i = 0; i < NotifierSaveDatas.Count; i++)
```

```
        {
            if (NotifierSaveDatas[i].Key == key)
                return true;
        }

        return false;
    }

    public void AddNotifierToView(NotifierType type, string parentKey)
    {
        for (int i = 0; i < NotifierSaveDatas.Count; i++)
        {
            if (NotifierSaveDatas[i].Key == parentKey)
                return;
        }
        NotifierSaveDatas.Add(new NotifierSaveData() { Key = parentKey, Type = type });
    }

    public void NotifierWasViewed(NotifierType type, string parentKey)
    {
        for (int i = 0; i < NotifierSaveDatas.Count; i++)
        {
            if (NotifierSaveDatas[i].Key == parentKey)
            {
                NotifierSaveDatas.RemoveAt(i);
                OnRemoved?.Invoke();
            }
        }
    }
}
using System.Collections;
using UnityEngine;
using UnityEngine.UI;
public class UINotificationElement : MonoBehaviour
{
    private const float BORDER_X = 2f;
    private const float BORDER_Y = 2f;
    [SerializeField] private NotifierType _type;
    [SerializeField] private Image _image;
    private bool _isViewed;
    public string ParentKey { get; set; }
    public IEnumerator OnVisible(bool forceDisable)
    {
        Vector3 elementPos;
        if (!forceDisable)
        {
            yield return new WaitUntil(() =>
            {
```

```
            elementPos = transform.parent.position -
UIMainController.Instance.UICanvas.transform.position;

            return elementPos.x > -BORDER_X && elementPos.x < BORDER_X && elementPos.y
> -BORDER_Y && elementPos.y < BORDER_Y;
        });

    }

    yield return new WaitForSeconds(1);

    if (!_isViewed)
    {
        _isViewed = true;
        Managers.SaveManager.userState.UINotificationBlock.NotifierWasViewed(_type,
ParentKey);
        _image.enabled = false;
    }
  }

  public void RefreshView(bool forceDisable = false)
  {
    var isAvailable =
Managers.SaveManager.userState.UINotificationBlock.IsNotifierAvailable(ParentKey);

    _image.enabled = isAvailable;
    _isViewed = !isAvailable;

    if (isAvailable)
        StartCoroutine(OnVisible(forceDisable));
  }
}
using TMPro;
using UnityEngine;

public class UINotificationGroupCountView : MonoBehaviour
{
  [SerializeField] private NotifierType _notifierType;
  [SerializeField] private GameObject _countViewerGo;
  [SerializeField] private TMP_Text _counter;

  private void OnEnable()
  {
    Managers.SaveManager.userState.UINotificationBlock.OnRemoved += OnElementRemoved;
  }

  private void OnDisable()
  {
```

```
        Managers.SaveManager.userState.UINotificationBlock.OnRemoved -= OnElementRemoved;
    }

    private void OnElementRemoved()
    {
        UpdateCounter();
    }

    public void UpdateCounter()
    {
        var count =
Managers.SaveManager.userState.UINotificationBlock.GetNotifierCountByType(_notifierType);
        _countViewerGo.SetActive(count > 0);

        _counter.text = count.ToString();
    }
}
```

ДОДАТОК В

Програмний код генерації модулів UI

```
using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.VFX;

using Object = UnityEngine.Object;
using System.Text.RegularExpressions;

namespace UnityEditor.VFX
{
    static class VFXCodeGenerator
    {
        private static string GetIndent(string src, int index)
        {
            var indent = "";
            index--;
            while (index > 0 && (src[index] == ' ' || src[index] == '\t'))
            {
                indent = src[index] + indent;
                index--;
            }
            return indent;
        }
```

```
        //This function insure to keep padding while replacing a specific string
        private static void ReplaceMultiline(StringBuilder target, string targetQuery, StringBuilder value)
        {
            string[] delim = { System.Environment.NewLine, "\n" };
            var valueLines = value.ToString().Split(delim, System.StringSplitOptions.None);
            if (valueLines.Length <= 1)
            {
                target.Replace(targetQuery, value.ToString());
            }
            else
            {
                while (true)
                {
                    var targetCopy = target.ToString();
                    var index = targetCopy.IndexOf(targetQuery);
                    if (index == -1)
                    {
                        break;
                    }

                    var indent = GetIndent(targetCopy, index);
                    var currentValue = new StringBuilder();
                    foreach (var line in valueLines)
                    {
                        currentValue.Append(indent + line + '\n');
                    }
                    target.Replace(indent + targetQuery, currentValue.ToString());
                }
            }
        }

        static private VFXShaderWriter GenerateLoadAttribute(string matching, VFXContext context)
        {
            var r = new VFXShaderWriter();

            var regex = new Regex(matching);
            var attributesFromContext = context.GetData().GetAttributes().Where(o => regex.IsMatch(o.attrib.name)).ToArray();
            var attributesSource = attributesFromContext.Where(a => context.GetData().IsSourceAttributeUsed(a.attrib, context)).ToArray();
            var attributesCurrent = attributesFromContext.Where(a => context.GetData().IsCurrentAttributeUsed(a.attrib, context) || (context.contextType == VFXContextType.Init && context.GetData().IsAttributeStored(a.attrib))).ToArray();

            //< Current Attribute
```

```
                foreach (var attribute in attributesCurrent.Select(o => o.attrib))
                {
                    var name = attribute.GetNameInCode(VFXAttributeLocation.Current);
                    if (attribute.name != VFXAttribute.EventCount.name)
                    {
                        if (context.contextType != VFXContextType.Init && context.GetData().IsAttributeStored(attribute))
                        {
                            r.WriteAssignement(attribute.type, name, context.GetData().GetLoadAttributeCode(attribute, VFXAttributeLocation.Current));
                        }
                        else
                        {
                            r.WriteAssignement(attribute.type, name, attribute.value.GetCodeString(null));
                        }
                    }
                    else
                    {
                        var linkedOutCount = context.allLinkedOutputSlot.Count();
                        r.WriteAssignement(attribute.type, name, attribute.value.GetCodeString(null));
                        for (uint i = 0; i < linkedOutCount; ++i)
                        {
                            r.WriteLine();
                            r.WriteFormat("uint {0}_{1} = 0u;", VFXAttribute.EventCount.name, VFXCodeGeneratorHelper.GeneratePrefix(i));
                        }
                    }
                    r.WriteLine();
                }

                //< Source Attribute (default temporary behavior, source is always the initial current value except for init context)
                foreach (var attribute in attributesSource.Select(o => o.attrib))
                {
                    var name = attribute.GetNameInCode(VFXAttributeLocation.Source);
                    if (context.contextType == VFXContextType.Init)
                    {
                        r.WriteAssignement(attribute.type, name, context.GetData().GetLoadAttributeCode(attribute, VFXAttributeLocation.Source));
                    }
                    else
                    {
                        if (attributesCurrent.Any(o => o.attrib.name == attribute.name))
                        {
                            var reference = new VFXAttributeExpression(new VFXAttribute(attribute.name, attribute.value), VFXAttributeLocation.Current);
```

```
                    r.WriteAssignement(reference.valueType, name, reference.GetCodeString(null));
                }
                else
                {
                    r.WriteAssignement(attribute.type, name, attribute.value.GetCodeString(null));
                }
            }
            r.WriteLine();
        }
        return r;
    }

    private const string eventListOutName = "eventListOut";

    static private VFXShaderWriter GenerateStoreAttribute(string matching, VFXContext context, uint linkedOutCount)
    {
        var r = new VFXShaderWriter();
        var regex = new Regex(matching);

        var attributesFromContext = context.GetData().GetAttributes().Where(o => regex.IsMatch(o.attrib.name) &&
            context.GetData().IsAttributeStored(o.attrib) &&
            (context.contextType == VFXContextType.Init || context.GetData().IsCurrentAttributeWritten(o.attrib, context))).ToArray();

        foreach (var attribute in attributesFromContext.Select(o => o.attrib))
        {
            r.Write(context.GetData().GetStoreAttributeCode(attribute, new VFXAttributeExpression(attribute).GetCodeString(null)));
            r.WriteLine(';');
        }

        if (regex.IsMatch(VFXAttribute.EventCount.name))
        {
            for (uint i = 0; i < linkedOutCount; ++i)
            {
                var prefix = VFXCodeGeneratorHelper.GeneratePrefix(i);
                r.WriteLineFormat("for (uint i = 0; i < {1}_{0}; ++i) {2}_{0}.Append(index);", prefix, VFXAttribute.EventCount.name, eventListOutName);
            }
        }
        return r;
    }
```

```
        static private VFXShaderWriter GenerateLoadParameter(string matching, VFXNamedExpression[] namedExpressions, Dictionary<VFXExpression, string> expressionToName)
        {
            var r = new VFXShaderWriter();
            var regex = new Regex(matching);

            var filteredNamedExpressions = namedExpressions.Where(o => regex.IsMatch(o.name) &&
                !(expressionToName.ContainsKey(o.exp) && expressionToName[o.exp] == o.name));  // if parameter already in the global scope, there's nothing to do

            bool needScope = false;
            foreach (var namedExpression in filteredNamedExpressions)
            {
                r.WriteVariable(namedExpression.exp.valueType, namedExpression.name, "0");
                r.WriteLine();
                needScope = true;
            }

            if (needScope)
            {
                var expressionToNameLocal = new Dictionary<VFXExpression, string>(expressionToName);
                r.EnterScope();
                foreach (var namedExpression in filteredNamedExpressions)
                {
                    if (!expressionToNameLocal.ContainsKey(namedExpression.exp))
                    {
                        r.WriteVariable(namedExpression.exp, expressionToNameLocal);
                        r.WriteLine();
                    }
                    r.WriteAssignement(namedExpression.exp.valueType, namedExpression.name, expressionToNameLocal[namedExpression.exp]);
                    r.WriteLine();
                }
                r.ExitScope();
            }

            return r;
        }

        static public StringBuilder Build(VFXContext context, VFXCompilationMode compilationMode, VFXContextCompiledData contextData)
        {
            var templatePath = string.Format("{0}.template", context.codeGeneratorTemplate);
            return Build(context, templatePath, compilationMode, contextData);
        }
```

```
        static private void GetFunctionName(VFXBlock block, out string functionName, out string comment)
        {
            var settings = block.GetSettings(true).ToArray();
            if (settings.Length > 0)
            {
                comment = "";
                int hash = 0;
                foreach (var setting in settings)
                {
                    var value = setting.value;
                    hash = (hash * 397) ^ value.GetHashCode();
                    comment += string.Format("{0}:{1} ", setting.field.Name, value.ToString());
                }
                functionName = string.Format("{0}_{1}", block.GetType().Name, hash.ToString("X"));
            }
            else
            {
                comment = null;
                functionName = block.GetType().Name;
            }
        }

        static private string FormatPath(string path)
        {
            return Path.GetFullPath(path)
                .TrimEnd(Path.DirectorySeparatorChar, Path.AltDirectorySeparatorChar)
                #if !UNITY_STANDALONE_LINUX
                .ToLowerInvariant()
                #endif
                ;
        }

        static IEnumerable<Match> GetUniqueMatches(string regexStr, string src)
        {
            var regex = new Regex(regexStr);
            var matches = regex.Matches(src);
            return matches.Cast<Match>().GroupBy(m => m.Groups[0].Value).Select(g => g.First());
        }

        static private VFXShaderWriter GenerateComputeSourceIndex(VFXContext context)
        {
            var r = new VFXShaderWriter();
            var spawnCountAttribute = new VFXAttribute("spawnCount", VFXValueType.Float);
```

```
            if (!context.GetData().dependenciesIn.Any())
            {
                var spawnLinkCount = context.GetData().sourceCount;
                r.WriteLine("int sourceIndex = 0;");

                if (spawnLinkCount <= 1)
                    r.WriteLine("/*//Loop with 1 iteration generate a wrong IL Assembly (and actually, useless code)");
                r.WriteLine("uint currentSumSpawnCount = 0u;");
                r.WriteLineFormat("for (sourceIndex=0; sourceIndex<{0}; sourceIndex++)", spawnLinkCount);
                r.EnterScope();
                r.WriteLineFormat("currentSumSpawnCount += uint({0});", context.GetData().GetLoadAttributeCode(spawnCountAttribute, VFXAttributeLocation.Source));
                r.WriteLine("if (id < currentSumSpawnCount)");
                r.EnterScope();
                r.WriteLine("break;");
                r.ExitScope();
                r.ExitScope();
                if (spawnLinkCount <= 1)
                    r.WriteLine("*/");
            }
            else
            {
                /* context invalid or GPU event */
            }
            return r;
        }

        static private StringBuilder GetFlattenedTemplateContent(string path, List<string> includes, IEnumerable<string> defines)
        {
            var formattedPath = FormatPath(path);

            if (includes.Contains(formattedPath))
            {
                var includeHierarchy = new StringBuilder(string.Format("Cyclic VFXInclude dependency detected: {0}\n", formattedPath));
                foreach (var str in Enumerable.Reverse<string>(includes))
                    includeHierarchy.Append(str + '\n');
                throw new InvalidOperationException(includeHierarchy.ToString());
            }
            includes.Add(formattedPath);
            var templateContent = new StringBuilder(System.IO.File.ReadAllText(formattedPath));
            foreach (var match in GetUniqueMatches("\\${VFXInclude(RP|)\\(\\\"(.*?)\\\"\\)(,.*)?}", templateContent.ToString()))
            {
```

```
                var groups = match.Groups;
                var renderPipelineInclude = groups[1].Value == "RP";
                var includePath = groups[2].Value;

                if (groups.Count > 3 && !String.IsNullOrEmpty(groups[2].Value))
                {
                    var allDefines = groups[3].Value.Split(new char[] {',', ' ', '\t'},
StringSplitOptions.RemoveEmptyEntries);
                    var neededDefines = allDefines.Where(d => d[0] != '!');
                    var forbiddenDefines = allDefines.Except(neededDefines).Select(d =>
d.Substring(1));
                    if (!neededDefines.All(d => defines.Contains(d)) || forbiddenDefines.Any(d =>
defines.Contains(d)))
                    {
                        ReplaceMultiline(templateContent, groups[0].Value, new StringBuilder());
                        continue;
                    }
                }
                string absolutePath;
                if (renderPipelineInclude)
                    absolutePath = VFXLibrary.currentSRPBinder.templatePath + "/" +
includePath;
                else
                    absolutePath = VisualEffectGraphPackageInfo.assetPackagePath + "/" +
includePath;

                var includeBuilder = GetFlattenedTemplateContent(absolutePath, includes,
defines);
                ReplaceMultiline(templateContent, groups[0].Value, includeBuilder);
            }

            includes.Remove(formattedPath);
            return templateContent;
        }

        static private void SubstituteMacros(StringBuilder builder)
        {
            var definesToCode = new Dictionary<string, string>();
            var source = builder.ToString();
            Regex beginRegex = new Regex("\\${VFXBegin:(.*)}");

            int currentPos = -1;
            int builderOffset = 0;
            while ((currentPos = source.IndexOf("${")) != -1)
            {
                int endPos = source.IndexOf('}', currentPos);
                if (endPos == -1)
                    throw new FormatException("Ill-formed VFX tag (Missing closing brace");
```

```
                var tag = source.Substring(currentPos, endPos - currentPos + 1);
                // Replace any tag found
                string macro;
                if (definesToCode.TryGetValue(tag, out macro))
                {
                    builder.Remove(currentPos + builderOffset, tag.Length);
                    var indentedMacro = macro.Replace("\n", "\n" + GetIndent(source, currentPos));
                    builder.Insert(currentPos + builderOffset, indentedMacro);
                }
                else
                {
                    const string endStr = "${VFXEnd}";
                    var match = beginRegex.Match(source, currentPos, tag.Length);
                    if (match.Success)
                    {
                        var macroStartPos = match.Index + match.Length;
                        var macroEndCodePos = source.IndexOf(endStr, macroStartPos);
                        if (macroEndCodePos == -1)
                            throw new FormatException("${VFXBegin} found without ${VFXEnd}");

                        var defineStr = "${" + match.Groups[1].Value + "}";
                        definesToCode[defineStr] = source.Substring(macroStartPos, macroEndCodePos - macroStartPos);

                        // Remove the define in builder
                        builder.Remove(match.Index + builderOffset, macroEndCodePos - match.Index + endStr.Length);
                    }
                    else if (tag == endStr)
                        throw new FormatException("${VFXEnd} found without ${VFXBegin}");
                    else // Remove undefined tag
                        builder.Remove(currentPos + builderOffset, tag.Length);
                }

                builderOffset += currentPos;
                source = builder.ToString(builderOffset, builder.Length - builderOffset);
            }
        }
        static private StringBuilder Build(VFXContext context, string templatePath, VFXCompilationMode compilationMode, VFXContextCompiledData contextData)
        {
            if (!context.SetupCompilation())
                return null;
            var stringBuilder = GetFlattenedTemplateContent(templatePath, new List<string>(), context.additionalDefines);

            var allCurrentAttributes = context.GetData().GetAttributes().Where(a =>
                (context.GetData().IsCurrentAttributeUsed(a.attrib, context)) ||
```

```
                (context.contextType == VFXContextType.Init && context.GetData().IsAttributeStored(a.attrib)));
                var allSourceAttributes = context.GetData().GetAttributes().Where(a => (context.GetData().IsSourceAttributeUsed(a.attrib, context)));

                var globalDeclaration = new VFXShaderWriter();
                globalDeclaration.WriteCBuffer(contextData.uniformMapper, "parameters");
                globalDeclaration.WriteLine();
                globalDeclaration.WriteAttributeStruct(allCurrentAttributes.Select(a => a.attrib), "Attributes");
                globalDeclaration.WriteLine();
                globalDeclaration.WriteAttributeStruct(allSourceAttributes.Select(a => a.attrib), "SourceAttributes");
                globalDeclaration.WriteLine();
                globalDeclaration.WriteTexture(contextData.uniformMapper);

                var linkedEventOut = context.allLinkedOutputSlot.Where(s => ((VFXModel)s.owner).GetFirstOfType<VFXContext>().CanBeCompiled()).ToList();
                globalDeclaration.WriteEventBuffer(eventListOutName, linkedEventOut.Count);

                //< Block processor
                var blockFunction = new VFXShaderWriter();
                var blockCallFunction = new VFXShaderWriter();
                var blockDeclared = new HashSet<string>();
                var expressionToName = context.GetData().GetAttributes().ToDictionary(o => new VFXAttributeExpression(o.attrib) as VFXExpression, o => (new VFXAttributeExpression(o.attrib)).GetCodeString(null));
                expressionToName = expressionToName.Union(contextData.uniformMapper.expressionToCode).ToDictionary(s => s.Key, s => s.Value);

                int cpt = 0;
                foreach (var current in context.activeFlattenedChildrenWithImplicit)
                {
                    BuildBlock(contextData, linkedEventOut, blockFunction, blockCallFunction, blockDeclared, expressionToName, current, ref cpt);
                }
```