МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра дискретного аналізу та інтелектуальних систем
(повна назва кафедри)

ДИПЛОМНА РОБОТА

РОЗРОБКА ТА ОПТИМІЗАЦІЯ AR ЗАСТОСУНКУ
З ДОПОМОГОЮ UNITY 3D

Виконав: студент IV курсу, групи ПМі-43с
напряму підготовки (спеціальності)
122 «Комп'ютерні науки»
(шифр і назва спеціальності)


(підпис) Кіс Ю. О. (прізвище та ініціали)

Керівник _____
(підпис) (прізвище та ініціали)

Рецензент _____
(підпис) (прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики
Кафедра Дискретного аналізу та інтелектуальних систем
Спеціальність 122 «Комп'ютерні науки»
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____

"31" серпня 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Кісові Юрієві Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та оптимізація AR застосунку з допомогою Unity 3D

керівник роботи професор кафедри дискретного аналізу та інтелектуальних систем Щербина Юрій Миколайович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від **"13" вересня 2022 року № 15**

2. Строк подання студентом роботи **13.06.2023р.**

3. Вихідні дані до роботи документація по мові програмування C#, документація Unity, відеоігровий рушій Unity, набір для розробки AR застосунків ARDK, документація Lightship ARDK, вбудовані пакети Unity: Rendering, UI, Physics, Animation, Audio, Video; DOTween, PermissionPlugin, TextMesh Pro, інтернет-ресурси

4. Зміст дипломної роботи (перелік питань, які потрібно розробити) Ознайомитись з технічними можливостями Unity, ознайомитись з технологією доповненої реальності, вибір одного із існуючих видів доповненої реальності, створення застосунку з гнучкою архітектурою в Unity з технологією доповненої реальності та його подальша оптимізація, опис використаних методів оптимізації та архітектури проекту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
ілюстрації циклу опрацювання об'єктів Unity, ілюстрації для візуалізації різних видів доповненої реальності, ілюстрації для пояснення роботи засобів відлагодження та наочного зображення дії методів оптимізації, порівняльна таблиця функціоналу для доповненої реальності різних наборів розробки, рисунки побудови архітектурних компонент проекту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання **31 серпня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Дослідження теоретичних відомостей	30.09.2022	
2.	Освоєння роботи з Unity	28.10.2022	
3.	Вибір SDK для роботи з доповненою реальністю	18.11.2022	
4.	Освоєння роботи із засобами відлагодження	30.11.2022	
5.	Ознайомлення із базовими методами оптимізації	16.12.2022	
6.	Підготовка основи структури проекту	28.01.2023	
7.	Інтеграція SDK доповненої реальності	18.02.2023	
8.	Удосконалення об’єкту гравця та реалізація керування	18.03.2023	
9.	Організаця сцени для роботи з AR	01.04.2023	
10.	Побудова системи квестів	22.04.2023	
11.	Наповнення сцени контентом	29.04.2023	
12.	Застосування методів оптимізації та відлагодження	28.05.2023	

Студент _____ Кіс Ю. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Щербина Ю. М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Робота складається із вступу, термінології, чотирьох розділів, висновків, джерел та одного додатку. Обсяг дипломної роботи: 43 сторінки, 1 таблиця та 37 рисунків. Список використаних джерел складається з 10 найменувань.

Мета даної роботи – розробка та дослідження процесу розробки AR застосунка із використанням відеоігрового рушія Unity 3D. Ця робота є вагомою, оскільки технологія доповненої реальності на даний момент стрімко розвивається, проте без слідуванню конкретним методам розробки. Різні команди виконують одні і ті ж задачі з метою створення якісного продукту, проте через хаотичність процесу допускаються великої кількості помилок. Окресливши правила роботи з доповненою реальністю в майбутньому буде можливість їх покращувати та доповнювати, що зрештою призведе до підвищення середньостатистичної якості створеної продукції та загального розвитку ринку технології доповненої реальності. Вказавши на базові помилки, яких допускаються молоді команди незалежних розробників, успіх їхніх продуктів стане стрімко зростати, що зрештою приверне ще більше уваги великих технологічних компаній, які в результаті будуть готові виділити значні кошти на ефективний розвиток індустрії. Фактично для створення застосунку використовується Unity 3D із інтеграцією ARDK, який дозволяє використовувати технологію доповненої реальності у проекті. Мова програмування – C#.

Новизна роботи полягає у визначенні найбільш критичних помилок, яких допускаються при розробці AR застосунків та опрацюванні повного спектру можливостей взаємодії з віртуальним світом, які надають сучасні смартфони.

ЗМІСТ

ТЕРМІНОЛОГІЯ 6
ВСТУП 8
1 ПРИНЦИПИ РОБОТИ ІЗ UNITY 10
2 ДОПОВНЕНА РЕАЛЬНІСТЬ 13
2.1 Система розпізнавання 14
2.2 Види доповненої реальності 16
2.3 Вибір SDK для доповненої реальності 21
3 МЕТОДИ ОПТИМІЗАЦІЇ 22
3.1 Засоби відлагодження 24
3.1.1 Профайлер 24
3.1.2 Вікно статистики 25
3.1.3 Відлагоджувач кадрів 25
3.2 Оптимізація роботи керованої пам’яті та коду 26
3.3 Оптимізація рендерингу 28
3.3.1 Рівень деталізації 28
3.3.2 Occlusion culling 29
3.3.3 Оптимізація асетів 30
4 РЕАЛІЗАЦІЯ 33
4.1 Об’єкт гравця 33
4.2 Керування 36
4.3 Побудова сцени 37
4.4 Побудова квестової системи 38
4.5 Результат 39
ВИСНОВКИ 40
ДЖЕРЕЛА 41
Додаток А 43

ТЕРМІНОЛОГІЯ

Unity 3D (також Unity, Юніті) – мультиплатформний інструмент, відеоігровий рушій для розробки ігор та застосунків для рендерингу фізичних симуляцій, тривимірних презентацій та ін.

fps (frames per second, кадрова частота, фреймрейт, фпс) – кількість кадрів (зображень), які відображаються на екрані користувача за одну секунду.

Mesh (polygon mesh, полігональний меш, полігональна сітка) – спосіб зберігання багатостороннього об’єкту в комп’ютерній тривимірній графіці за допомогою набору ребер, граней та вершин. Серед підтипів полігональних мешів найбільше використовується сітка трикутників.

Triangle mesh (сітка трикутників) – включає в себе набір трикутників, які мають спільні ребра або вершини.

Triangles (trigs, трикутники) – в контексті цієї дипломної роботи, це ключова складова сітки трикутників. Будівельна компонента для опису форми будь-якого можливого багатогранного об’єкту.

AR (augmented reality, доповнена реальність) – інтерактивний досвід, який об’єднує реальний світ та віртуальний контент.

VR (virtual reality, віртуальна реальність) – інтерактивний досвід, який поміщає користувача у повністю тривимірний віртуальний світ.

UX (user experience, досвід користувача) – спосіб у який користувач взаємодіє із застосунком, а застосунок впливає на дії користувача.

Рендеринг – процес генерування фотореалістичної чи не фотореалістичної картинки із двовимірного зображення або тривимірних моделей під час роботи комп’ютерної програми.

SDK (software development kit) – набір засобів розробки та відповідної документації, який надає програмістам можливість створювати прикладні програми за визначеною технологією.

Point Cloud (хмара точок) – дискретний набір точок даних у тривимірному просторі.

Ассет (ігровий ресурс, англ. game asset) – цифровий об’єкт, який представляє частину ігрового контенту. До ассетів належать всі об’єкти, які використовуються в застосунку: починаючи від коду та звуків і закінчуючи моделями об’єктів та діалогами.

Спрайт – двовимірний графічний об’єкт. Найчастіше це растрове зображення.

UI (user interface) – інтерфейс користувача, який надає додаткові відомості про ігровий світ у вигляді двовимірної графіки, також може бути використаний як ігровий елемент.

ВСТУП

Останнім часом найбільш впливові технічні компанії починають приділяти все більше уваги технологіям доповненої та віртуальної реальності. Завдяки цим технологіям уможливлюється навчання спеціалістів без будь-якого реального обладнання – шоломи віртуальної реальності дозволяють освоїти базові навички та принципи будь-якої комплексної роботи. За свій досвід я зустрів проекти, які були націлені на тренування медперсоналу, барист, барменів та навіть працівників металургійної професії. Окрім цього, застосунки доповненої реальності дозволяють добавляти інтерактивності різним виставкам, картинам, надають можливість оглянути тривимірну модель ще не збудованої споруди через дисплей звичайного смартфона. Особисто я слідкував за проектом фотографа, який створював панорамні фотографії, і надавав контексту цим фотографіям віртуальними вказівниками з допомогою технології доповненої реальності.

Проте основними продуктами, які розробляються з допомогою цих технологій звісно ж є відеоігри, а враховуючи виключно позитивний ріст відеоігрового ринку протягом останніх десяти років, стає цілком зрозуміло, чому великі компанії готові ризикнути і почати вкладати гроші в досі нові технології, які все ще розвиваються і мають непевне майбутнє.

Окрім великих компаній, стає все більше незалежних розробників (indie developers), які фокусуються на створенні менших, але все ще надзвичайно прибуткових, як на свої вкладення, ігор. Не в останню чергу це зумовлено доступністю відкритих відеоігрових рушіїв, таких як Unity, Godot та Unreal Engine. Проте, якщо великі команди мають вдосталь фінансування для належного циклу розробки, з мінімізацією проблем оптимізації та багів, то менші команди постійно стикаються з вищенаведеними проблемами.

Окрім згаданих проблем з оптимізацією, зачасту малі команди також досить погано розуміють важливість UX – досвіду користувача. Багато речей, які розробляються малими командами просто-напросто нелогічні. Багато речей, як от неякісні ассети та прогалини в історії побачити насправді набагато простіше, ніж

проблеми в UX, в результаті користувач попросту не знатиме, як використовувати застосунок. Як розробник, я чудово розумію, чому ось цей ефект «замиленого ока» стається – створивши систему так, як собі її уявляєш, часто можна не усвідомлювати, що для інших людей така система навпаки контрінтуїтивна.

Враховуючи те, що VR та AR продовжують набирати популярності, великі компанії (до прикладу Meta) щиро зацікавлені та щедро фінансують продовження розвитку цих технологій, а менші команди ще просто не набралися досвіду у створенні застосунків та відеоігор з доповненою та віртуальною реальностями, я вирішив поекспериментувати з розробкою AR застосунка власноруч. Звісно ідея створити VR застосунок була набагато більш захоплюючою, проте причина відмовитись від цієї ідеї була досить банальною – відсутність VR шолому. З іншого боку, створення та використання AR застосунку потребує лише ПК (чи ноутбук) та більш-менш сучасний телефон, який працює на системі Android чи iOS.

1 ПРИНЦИПИ РОБОТИ ІЗ UNITY

Unity здебільшого використовується для створення ігор. Відповідно, він має працювати в певних циклах, у яких опрацьовуються користувацькі вводи даних (це можуть бути натискання клавіш на клавіатурі, рух мишкою, дотики до екрану смартфона і навіть відповідь на зміни орієнтації девайсу в просторі чи просто різкі зміни швидкості, які реєструються акселерометром), щоб відтворювати послідовно набір кадрів, у відповідь на дії користувача.

Для отримання плавної та повноцінно якісної гри із задовільною швидкістю роботи, посекундна кількість кадрів має бути як мінімум на рівні 30 фпс, а в ідеалі на рівні 60 фпс та вище [1]. До того ж, для опрацювання одного єдиного кадру, система має пройти надзвичайно великий і комплексний цикл опрацьовуючи кожен об’єкт на сцені [2]. Спрощену модель цього циклу наведено на рис. 1.1, який подано нижче:

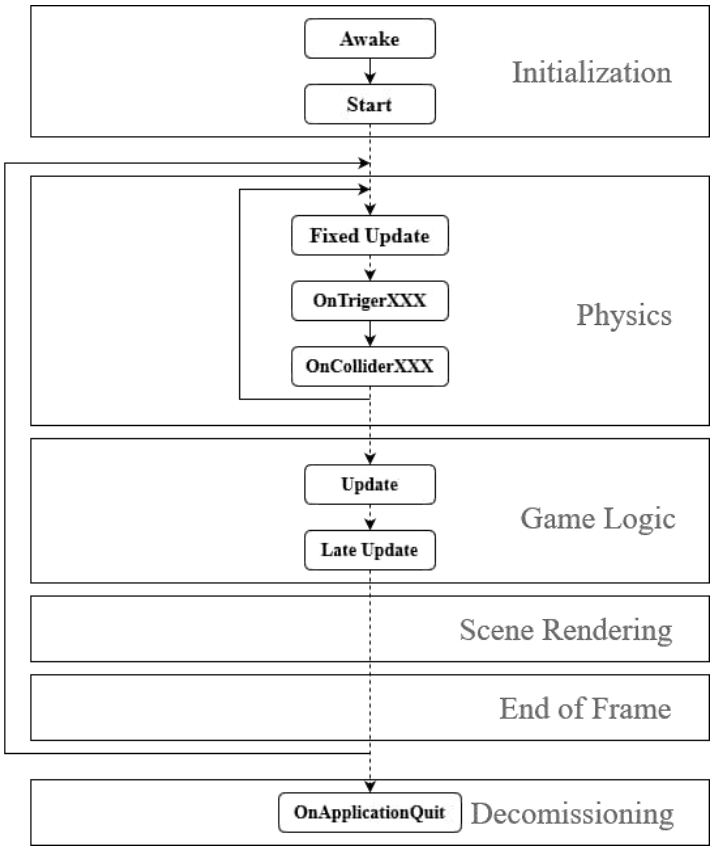

Рисунок 1.1. Цикл роботи, який Unity проходить,
опрацьовуючи кожен об’єкт сцени

Виходячи з графіка на рис. 1.1, та з того, що кожної секунди Unity опрацьовує як мінімум до 30 разів всі задіяні об’єкти, переобчислюючи фізику та проводячи рендеринг сцени заново, можна дійти висновку, що навіть незначні зміни в коді, спрямовані на оптимізацію, зрештою додаються і мають усі шанси значно допомогти у прискоренні роботи застосунку.

Unity використовує концепцію ієрархії між батьками та дітьми (рис. 1.2). Це важливо зауважити, оскільки для утворення посилань між різними об’єктами (до прикладу, камера - персонаж чи персонаж - контролер гри) використовується цілий ряд функцій, кожна з яких має свою швидкість виконання.

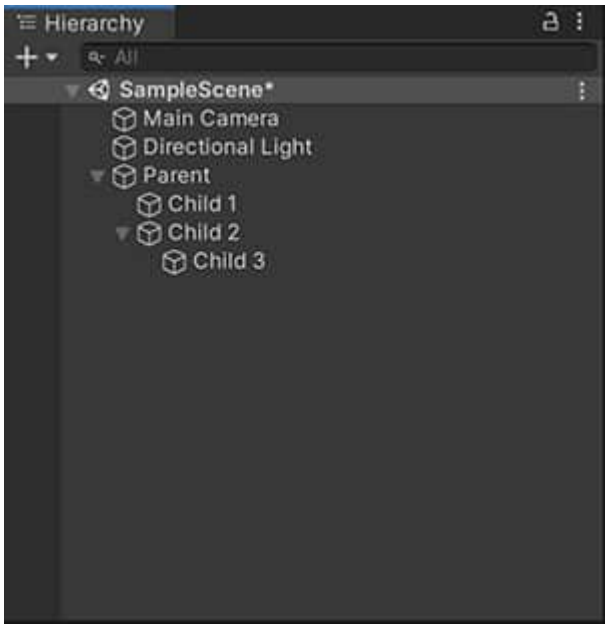

Рисунок 1.2. Ієрархічна структура Unity

Також важливо згадати, що кожен об’єкт, який міститься на сцені, може містити багатьо компонентів (рис. 1.3). Створення посилань між компонентами одного об’єкту чи різних об’єктів також є однією з задач програміста, яку можна оптимізовувати.

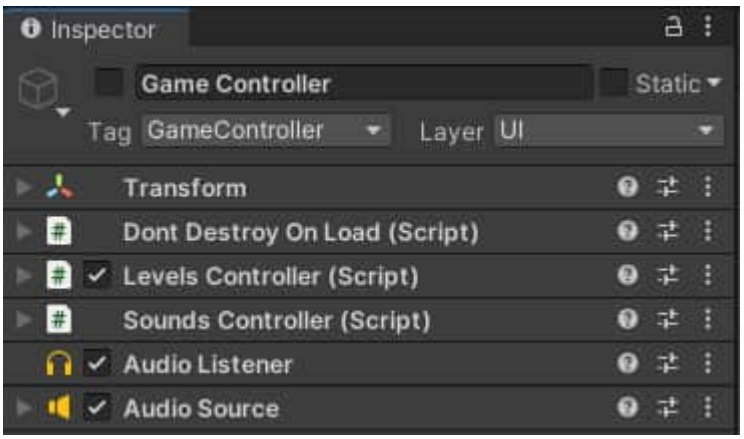

Рисунок 1.3. Інспектор об’єкта

Ознайомлюючись з можливостями Юніті я натрапив на так званий «вбудований список пакетів» (рис. 1.4), що вказує на те, що Юніті використовує модульну систему. Окрім того, Юніті очевидно має широкий вибір не вбудованих пакетів, а також надає можливість користувачам створювати власні пакети, і в подальшому поширювати їх як SDK. Це може знадобитись в майбутньому при пошуку необхідних компонент для створення застосунку, в тому числі AR плагінів чи SDK.

- AI: com.unity.modules.ai
- Android JNI: com.unity.modules.androidjni
- Animation: com.unity.modules.animation
- Asset Bundle: com.unity.modules.assetbundle
- Audio: com.unity.modules.audio
- Cloth: com.unity.modules.cloth
- Director: com.unity.modules.director
- Image Conversion: com.unity.modules.imageconversion
- IMGUI: com.unity.modules.imgui
- JSONSerialize: com.unity.modules.jsonserialize
- NVIDIA: com.unity.modules.nvidia
- Particle System: com.unity.modules.particlesystem
- Physics: com.unity.modules.physics
- Physics 2D: com.unity.modules.physics2d
- Screen Capture: com.unity.modules.screencapture
- Terrain: com.unity.modules.terrain
- Terrain Physics: com.unity.modules.terrainphysics
- Tilemap: com.unity.modules.tilemap
- UI: com.unity.modules.ui
- UIElements: com.unity.modules.uielements
- Umbra: com.unity.modules.umbra
- Unity Analytics: com.unity.modules.unityanalytics
- Unity Web Request: com.unity.modules.unitywebrequest

Рисунок 1.4. Вбудований список пакетів

2 ДОПОВНЕНА РЕАЛЬНІСТЬ

Augmented reality, також AR або доповнена реальність – інтерактивний досвід, який об’єднує реальний світ та комп’ютерний (віртуальний) контент. Цей контент зазвичай є візуальним, проте окрім цього він може бути у вигляді аудіо чи вібрацій, які продукує девайс. Основні характеристики доповненої реальності [3]:

- Комбінація реального та віртуального світу;
- Можливість взаємодіяти з віртуальним світом в живому часі;
- Точна реєстрація та відображення в тривимірному просторі віртуального світу, тісно зв’язаного із реальним;

Одним із найбільш відомих проектів, який використовує доповнену реальність є Pokemon Go, гра випущена компанією Niantic у 2016 році (рис. 2.1).

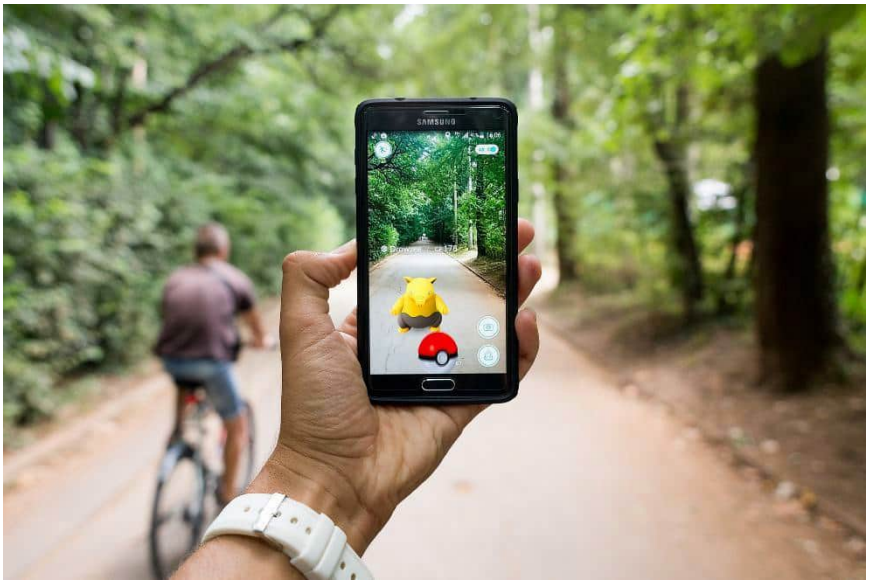

Рисунок 2.1. Приклад роботи Pokemon Go

Особлива цінність доповненої реальності це те, як вона вливається у сприйняття реального світу людиною. Доповнена реальність відчувається не просто відображенням певного контенту на дисплей, а як натуральне доповнення справжнього світу, зокрема завдяки інтеграції іммерсивних відповідей на певні дії: аудіо-відповіді на дії користувача та відчуття дотику, яке на мобільних пристроях симулюється вібраціями. Цей ефект зокрема чітко видно, коли людина виконує певну роботу, використовуючи допомогу доповненої реальності [4].

2.1 Система розпізнавання

Хоч в цій роботі уже й було згадано смартфони, як девайси, які в основному використовують доповнену реальність, вони лиш недавно стали достатньо потужними, а алгоритми достатньо оптимізованими, щоб в прямому часі опрацьовувати необхідні кількості інформації. Взагалом, пристрій, здатний на відображення доповненої реальності, повинен мати дисплей, потужний процесор, якісну камеру та певні засоби вводу. Років з десять назад для цього створювали спеціалізовані пристрої, які працювали виключно із доповненою реальністю.

В сьогоденні сучасного смартфона справді цілком хватає. Камери надзвичайно якісні, є можливість вводу інформації у вигляді тач-дисплею, а також наявні додаткові мікроелектромеханічні системи, як от акселерометр, GPS та цифровий компас. Все це в сукупності дозволяє розглядати сучасні смартфони як девайси доповненої реальності [5].

Зокрема, якість відслідковування значно зростає завдяки тому, що в найновіших моделях інженери добавляють додаткові камери, які обчислюють глибину зображення, що дозволяє набагато точніше проектувати віртуальні об’єкти на реальний світ та використовувати додатковий функціонал.

Безпосередньо щодо системи розпізнавання – це основа технології доповненої реальності. Системи розпізнавання можна поділити на дві категорії:

- Локаційно-залежні
- Об’єктно-залежні

Взагалом, локаційно-залежні системи використовують SLAM – групу методів одночасної локалізації та картографування. SLAM використовується для побудови карти в невідомому просторі з одночасним контролем поточного місцезнаходження і пройденого шляху.

Спрощено – SLAM використовується для розпізнавання оточення шляхом розкладання картинки на геометричні об’єкти та лінії. Після цього кожній окремій формі та лінії система присвоює одну або кілька точок, так званих якорів, фіксуючи

їх розташування в просторових координатах на послідовних кадрах відеопотоку. Це зокрема допомагає виділяти різноманітні площини. В найпростіших прикладах – тільки підлогу, в складніших ще й різноманітні точки інтересу, які дозволяють більш точно позиціонувати камеру у віртуальному просторі.

Математично, нехай дана послідовність даних спостереження сенсору за дискретні проміжки часу. Задачею SLAM є розрахувати і визначити розташування агента і мапу оточення. Всі величини зазвичай ймовірнісні [6], тому необхідно обчислити (2.1.1):

$$P(m_t, x_t | o_{1:t}) \quad (2.1.1)$$

де o_t — дані спостереження сенсору, x_t — розташування агента, m_t — мапа оточення та t — дискретні проміжки часу. Застосування правила Байєса дає основу для послідовного оновлення апостеріорного розташування (2.1.2), при мапі і функції переходу $P(x_t \mid x_{t-1})$,

$$P\left(x_t \middle| o_{1:t,} m_t\right) = \sum_{m_{t-1}} P(o_t | x_t, m_t) \sum_{x_{t-1}} P(x_t | x_{t-1}) P(x_{t-1} | m_t, o_{1:t-1}) / Z \quad (2.1.2)$$

Аналогічно мапа може оновлюватися послідовно наступним чином (2.1.3):

$$P(m_t | x_t, o_{1:t}) = \sum_{x_t} \sum_{m_t} P(m_t | x_t, m_{t-1}, o_t) P(m_{t-1}, x_t | o_{1:t-1}, m_{t-1}) \quad (2.1.3)$$

Як для більшості задач наближення, рішення можна знайти при наближенні двох змінних, до локального оптимального рішення, шляхом почергового оновлення двох рівнянь у формі EM-алгоритму.

На даний момент існує більше десятка різних методів SLAM, які відрізняються конкретними напрямками застосування та швидкістю, з якою вони працюють.

Об’єктно-залежні системи натомість намагаються знайти точки інтересу певної картини, об’єкту, чи навіть локації, узагальнено – маркеру. Головна різниця від локаційно-залежної системи в тому, що об’єктно-залежна система наперед знає, що їй треба шукати. Перший крок роботи – опрацювання отриманого зображення з відеопотоку, для розпізнавання шуканих точок інтересу [7]. Зокрема для цього

використовують популярні методи обробки зображень: виявлення кутів, виявлення плям, виявлення контурів, порогування та інші. З цього опису стає зрозуміло, що маркер має бути насичений деталями, без симетричних візерунків та й взагалом якомога різноманітнішим. Другим кроком стає позиціювання камери у просторі відносно маркеру.

2.2 Види доповненої реальності

Після того як ми розібрались з технічними деталями роботи технології доповненої реальності настав час обрати який саме тип трекінгу використовуватиметься у застосунку. Найпопулярніші з них наведені у списку нижче:

- Plane Tracking (відслідковування поверхні)

Відслідковування поверхонь дозволяє слідкувати за плоскими поверхнями у прямому часі (рис. 2.2.1). Використовується локаційно-залежна система слідкування за оточенням. Це найпростіший приклад роботи такої системи. Взагалом дозволяє використовувати одну чи кілька площин для різноманітних інтеракцій визначених програмістом. Не найбільш інтерактивний вид відслідковування, тож його я вирішив не використовувати. Окрім того, він також використовується в інших видах відслідковування.

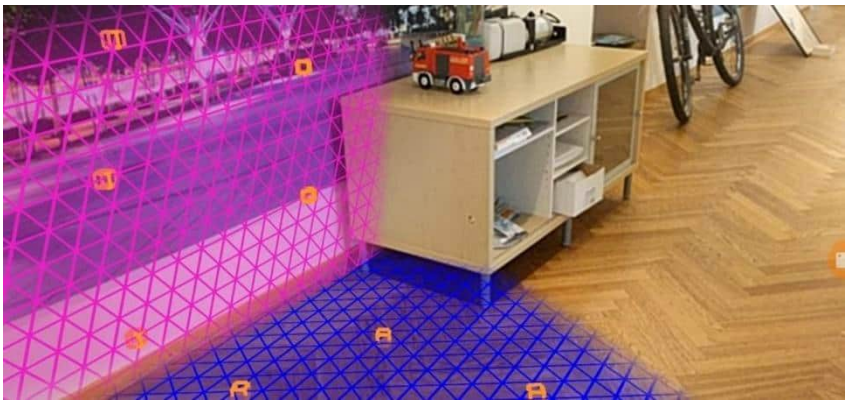

Рисунок 2.2.1. Приклад роботи відслідковування поверхонь

- Face Tracking (відслідковування обличчя)

Відслідковування обличчя (рис. 2.2.2) дозволяє накладати маски – візуальні модифікації лиця, прикладом яких можуть бути маски в ТікТоці чи Інстаграмі. Окрім того, ця технологія використовується у FaceID для перевірки обличчя людини, яка намагається увімкнути телефон. Не надто інтерактивний досвід, як на мене, та й цікавої архітектури застосунку побудувати не вдастся.

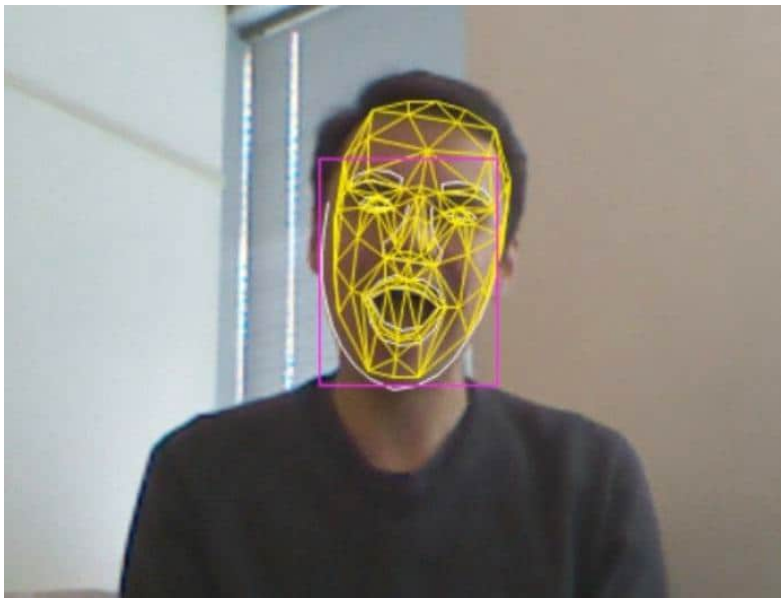

Рисунок 2.2.2. Накладання трикутників на розпізнане обличчя

- 2D Image Tracking (відслідковування двовимірного об’єкту)

Для цього типу доповнених досвідів використовується об’єктно-залежна система розпізнавання. Приклад шуканого маркера зображено на рис. 2.2.3. Після розпізнавання центр світу встановлюється в центрі розпізнаного двовимірного зображення, а віртуальні елементи розміщуються поверх нього так, як зазначено в коді програми (рис. 2.2.4).

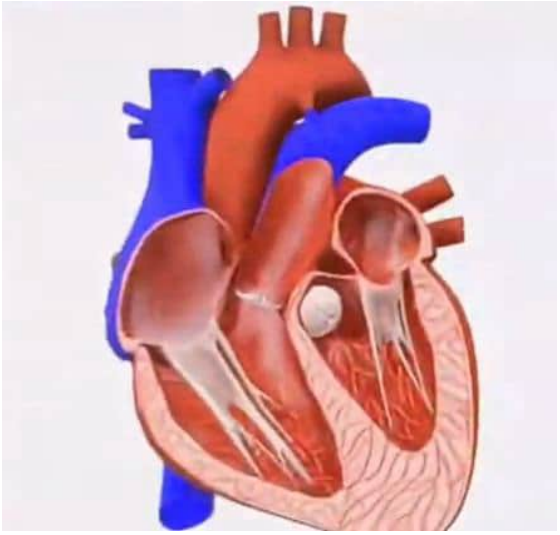

Рисунок 2.2.3. Приклад маркеру

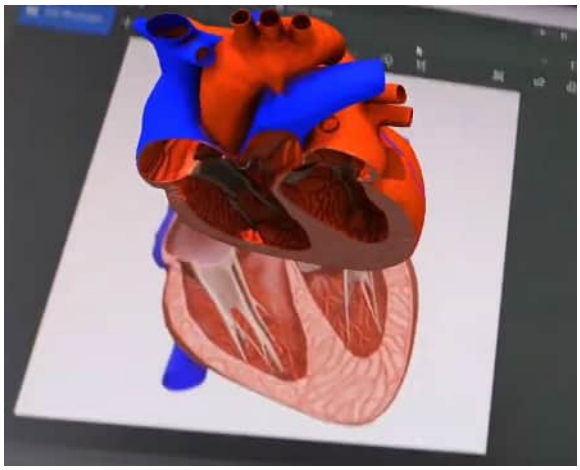

Рисунок 2.2.4. Приклад розташування віртуального елементу
поверх розпізнаного маркеру

- 3D Object Tracking (відслідковування тривимірного об’єкту)

Як і в попередньому випадку використовується об’єктно-залежна система розпізнавання, проте шуканим маркером у цьому випадку є не двовимірний об’єкт, а так званий point cloud – хмара точок, яка відображає особливі шукані точки тривимірного об’єкту (рис. 2.2.5). Схожий алгоритм використовується при Location Tracking, з тією різницею, що в трекінгу локації point cloud охоплює не конкретний об’єкт, а цілу локацію.

Рисунок 2.2.5. Point Cloud горнятка

- 2D & 3D Body Tracking (відслідковування людського тіла)

Цей тип трекінгу надзвичайно схожий до Face Tracking, тільки в ньому шукається не лице, а радше ціле тіло людини (рис. 2.2.6). Різниця між двовимірним та тривимірним видом трекінгу людського тіла у додаванні третьої координа для відслідковування глибини розташування окремих частин тіла. Нерідко два види трекінгу (Face Tracking, Body Tracking) об’єднують і відслідковують одночасно лице та тіло (рис. 2.2.7).

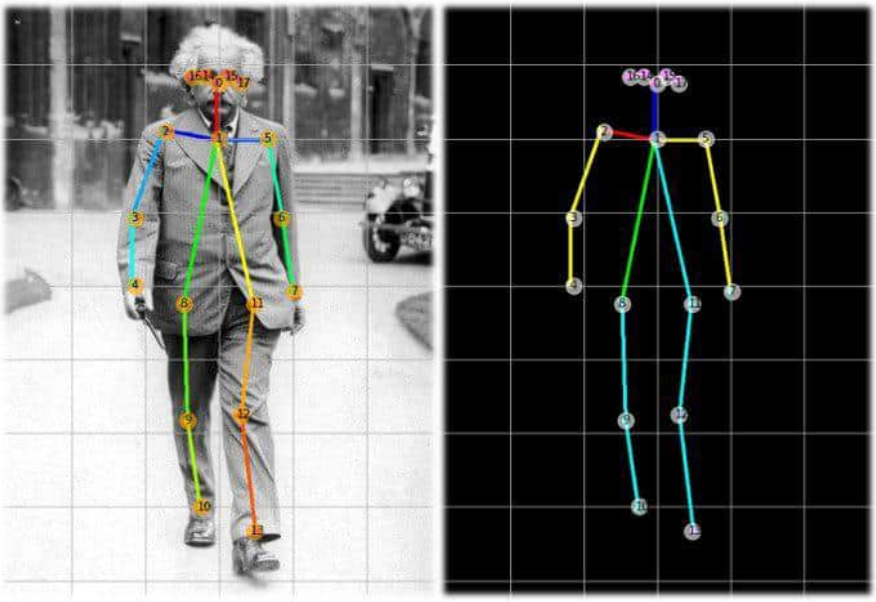

Рисунок 2.2.6. Відслідковування пози людини на зображенні

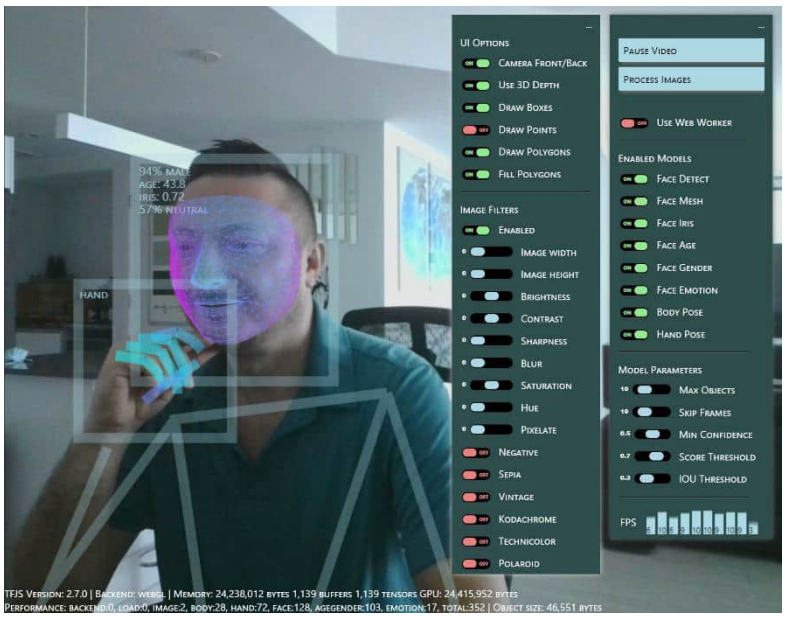

Рисунок 2.2.7. Відслідковування пози та лиця людини у відеопотоці

- Real-Time Mapping (відслідковування в просторі)

Цей тип трекінгу найбільш ресурсо-затратний, використовується локаційно-залежна система розпізнавання навколишнього оточення без конкретних даних, які варто шукати. Основне ускладнення навідміну від Plane Tracking-у походить з того, що відслідковуються не лише площини, а ще й утворюється point cloud, який в подальшому зберігаються та постійно оновлюється для збільшення точності трекінгу.

Серед усіх вищенаведених видів доповненої реальності мені найбільше імпонує Real-Time Mapping, який надає найбільш широкий спектр функцій для створення інтерактивних досвідів [8]. Окрім цього, він найбільш ресурсо-затратний, що означає, що для його коректної роботи необхідно буде займатись оптимізацією застосунку. Разом з тим, в довгостроковій перспективі він найбільш точний, оскільки point cloud, який регулярно розширюється та оновлюється матиме найбільш точні дані. Саме цей вид доповненої реальності я буду використовувати у своєму застосунку.

2.3 Вибір SDK для доповненої реальності

Ознайомлюючись із робочими можливостями Unity, я зауважив, що він працює за модульною системою, отже варто пошукати модулі, які дозволяють використовувати доповнену реальність у застосунках. Також існує можливість створити свою власну систему SLAM, яка опрацьовуватиме дані в реальному часі, проте існуючі альтернативи практично гарантовано більш оптимізовані і працюватимуть на обох основних платформах, які можуть цікавити розробника застосунку доповненої реальності – Android та iOS.

Серед доступних варіантів я натрапив на такі:

- ARFoundation з використанням ARCore та ARKit; сам по собі ARFoundation надає лише основу для опрацювання даних, які видає SLAM, в той час як ARCore для Android та ARKit для iOS витягують дані з девайсів та виконують основну частину математичних обчислень для створення досвіду з доповненою реальністю. Це офіційний модуль Юніті, тож потенційно він найбільш стабільний.
- EasyAR, китайський SDK для роботи з доповненими реальностями. Цей варіант не безкоштовний, та й не має завершеної підтримки Real-Time Mapping-у.
- Vuforia, має безкоштовний для студентів плагін для Unity, який може генерувати та зберігати в хмарному сховищі маркери; спеціалізується на об’єктно-залежні системи розпізнавання.
- Wikitude, має безкоштовний пробний період, спеціалізується в трекінгу одного або й кількох маркерів, а також у відслідковуванні світу в живому часі, тобто Real-Time Mapping.
- ARDK, від творців Pokemon Go, спеціалізується в основному на Real-Time Mapping, а також має вбудовані системи для мультиплеєру та семантичної сегментації людей поверх віртуального світу.

Функціональний перелік видів трекінгу, який надають вищезгадані SDK наведено у табл. 2.3.1.

Таблиця 2.3.1. Перелік функціоналу різноманітних SDK для доповненої реальності

	ARCore	ARKit	EasyAR	Vuforia	Wikitude	ARDK
Plane Tracking	Yes	Yes	Yes			Yes
Face Tracking	Yes	Yes				
2D Image Tracking	Yes	Yes	Yes	Yes	Yes	Yes
3D Object Tracking		Yes	Yes	Yes	Yes	
Body Tracking		Yes				Yes
Point Clouds	Yes	Yes	Yes	Yes	Yes	Yes
Real-Time Mapping	Yes	Yes	Yes		Yes	Yes
Networking		Yes				Yes
Multiple Trackers	Yes	Yes		Yes	Yes	
Cloud Recognition			Yes	Yes	Yes	

Серед вищенаведених SDK чітко виділяється ARDK, який зосереджується конкретно на Real-Time Mapping [9], що зокрема й робить його найбільш ефективним при використанні із цим видом доповненої реальності. Його я й оберу як основу для проекту.

3 МЕТОДИ ОПТИМІЗАЦІЇ

Як добре відомо, мобільні девайси значно менш потужні, ніж персональні комп’ютери чи консолі. Поверх цього, додаткового навантаження на потужності смартфону утворюватиме система розпізнавання, тож питанням оптимізації не можна нехтувати ніяк. Додатковим ускладненням є й те, що різноманітні телефони мають різне апаратне забезпечення та різні характеристики деталей. Зачасту найбільш очевидним та найбільш банальним показником потужності є рік випуску, оскільки чим новіший девайс, тим кращі деталі в ньому використовуються. Хоча

варто зауважити, що деякі бюджетні телефони 2023 року мають досить слабкі процесори та не таку вже й швидку оперативну пам’ять.

Загалом, цитуючи британського інформатика Майкла А. Джексона:

“Перше правило програмної оптимізації - не займайтесь нею. Друге правило програмної оптимізації (лише для професіоналів!): не займайтесь нею вже зараз”

Думка, яку він хотів донести цією фразою, була такою. Зважаючи на швидкодію, з якою працюють наші комп’ютери і якими темпами вона росте, є високий шанс того, що створивши неефективний код, він все одно працюватиме достатньо швидко, щоб ніяк не впливати на систему взагалом. Крім того, оптимізовуючи код забагато є високий шанс, що в подальшому це буде обмежувати програміста чи утворюватиме баги відслідкувати які надзвичайно важко.

Тим не менш, це зовсім не означає, що оптимізацією займатись немає сенсу. Друга частина фрази пропонує оптимізувати програму від самого початку розробки, слідкуючи за тим, щоб жодна система не використовувала забагато ресурсів платформи, ще в процесі розробки цієї системи.

Я орієнтуватимусь на телефон Galaxy S9, 2018 року випуску, він є достатньо старим, але все ще підтримує використання доповненої реальності. Якщо створений застосунок працюватиме на цьому телефоні із задовільною кадровою частотою - на інших мобільних девайсах він працюватиме бездоганно.

Щодо того, що саме можна оптимізовувати у застосунках, створених використовуючи Юніті, я би виділив три конкретні категорії:

- Оптимізація керованої пам’яті;
- Оптимізація рендерингу;
- Оптимізація коду;

Інші речі, як от вбудовані модулі чи використовувані SDK оптимізувати попросту немає змоги, їхній код по-перше закритий у бібліотеках, а по-друге уже достатньо оптимізований і навіть не раз оптимізований на рівні компілятора, цим займається команда Unity, тож в цьому плані я їм довірюся.

3.1 Засоби відлагодження

Нижче описано ті засоби відлагодження, які я активно використовував при розробці програми. Вміння ефективно користуватись ними – важливий аспект розробки кожного застосунку, який використовує Unity.

3.1.1 Профайлер

Профайлер (рис. 3.1.1.1) – надзвичайно потужний інструмент, який надає детальну інформацію про те, як працює проект. Під час запуску проекту в режимі редактора, профайлер деталізує використання проектом CPU, оперативної пам’яті, обчислень, які займає опрацювання фізики, навантаження на GPU, використання інтернет мережі та ін. Якщо проект має певні проблеми, такі як низький фреймрейт або високе використання оперативної пам’яті – профайлер допоможе відслідкувати джерело проблеми аж до виклику конкретних функцій.

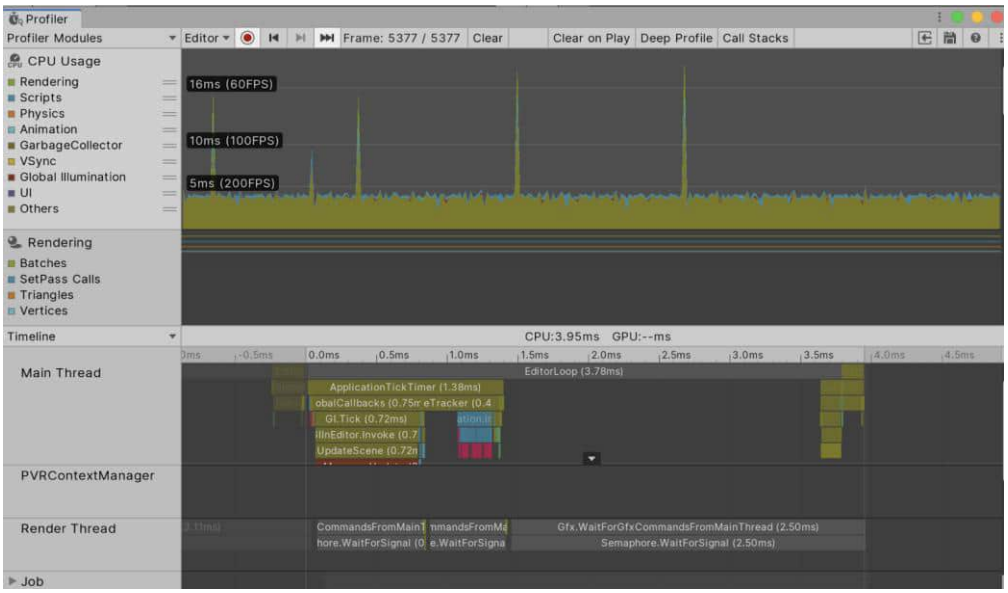

Рисунок 3.1.1.1. Вікно профайлеру

На рисунку 3.1.1.2 бачимо, що профайлер відслідкував всього 11536 кадрів, а обраний кадр (11493) сягнув позначки в 120 фпс (при середньому значенні в ~300 фпс). Опрацювання цього кадру на CPU зайняло ~9.2мс і основна частка роботи припадає на Garbage Collection (~3.4мс), що спричинило просідання кадрової частоти, отже для вирішення таких просідань варто попрацювати з Garbage Collection.

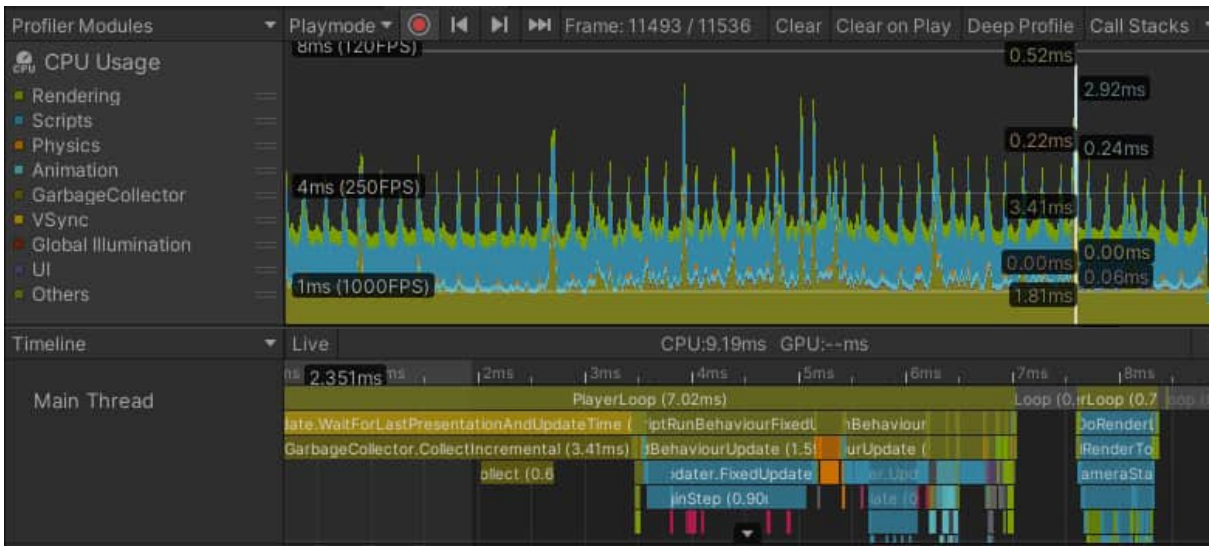

Рисунок 3.1.1.2. Приклад роботи профайлеру у робочому проекті

3.1.2 Вікно статистики

Вікно статистики (рис. 3.1.2.1) містить значну кількість інформації в компактній формі, яка також може допомогти при відлагодженні застосунку. Основний спосіб використання вікна статистики - слідкування за кадровою частотою. Також зручно слідкувати за опрацьованою в поточному кадрі кількістю трикутників.

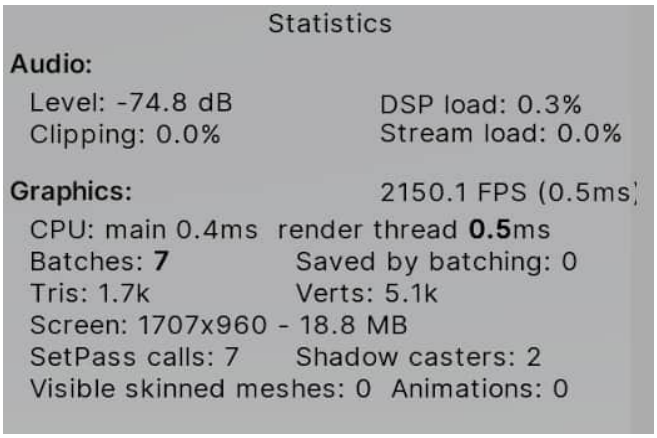

Рисунок 3.1.2.1. Вікно статистики

3.1.3 Відлагоджувач кадрів

Відлагоджувач кадрів (рис. 3.1.3.1) – зручний інструмент для дослідження роботи графічного ядра. Він дозволяє призупинити відтворення проекту на будь-якому кадрі, щоб оглянути кожен виклик функції промалювання для кожного окремого елемента на екрані. Також дозволяє поступово оглянути процес формування картинки на екрані відділяючи кожен етап будування кадру.

Якщо профайлер вказує на проблеми саме з графічною частиною роботи застосунку – в першу чергу ми звертаємося до дебагера кадрів. Якщо проблемою є певний елемент на сцені і він займає багато часу для відтворення (до прикладу модель з надзвичайно великою кількістю трикутників) - буде зрозуміло, з чим надалі варто працювати для покращення швидкодії (у випадку неоптимізованої моделі – зменшення кількості трикутників, за рахунок погіршення вигляду моделі).

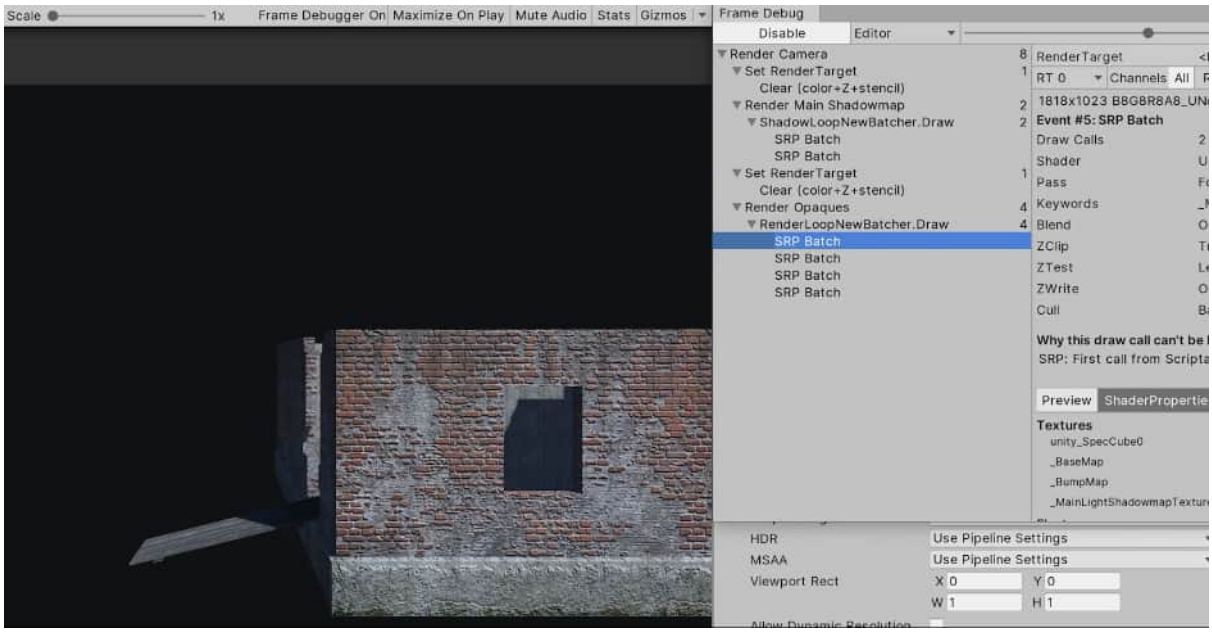

Рисунок 3.1.3.1. Вікно дебагера кадрів

3.2 Оптимізація роботи керованої пам’яті та коду

Багато методів, які використовуються для оптимізації роботи керованої пам’яті зачасту оптимізують і загальну роботу коду, тож цей пункт буде спільним для обох. Детальніше про керовану пам’ять викладено у додатку А.

Перше, і напевне найважливіше правило, це використання кешування. Як згадувалось раніше і як зображено на рис. 1.3, кожен об’єкт може містити декілька компонент. Для отримання посилання на конкретну компоненту існує декілька функцій, до прикладу GetComponent<T>, де T - тип шуканої компоненти. Процес пошуку компоненти відбувається по списку усіх компонент цього об’єкту, тож логічно, що ітерувати по усіх елементах списку в кожному циклі функції Update, яка відбувається кожного кадру, просто-напросто не ефективно. Рішенням для цієї проблеми є вищезгадане кешування, до прикладу, пошук компоненти у функції

Awake(), яка викликається для кожного об’єкту лише раз при ініціалізації як вказується на рис. 1.1. Іншим рішенням є використання атрибуту [SerializedField], що надає змогу додати посилання на компоненту у вікні інспектора (рис. 3.2.1).

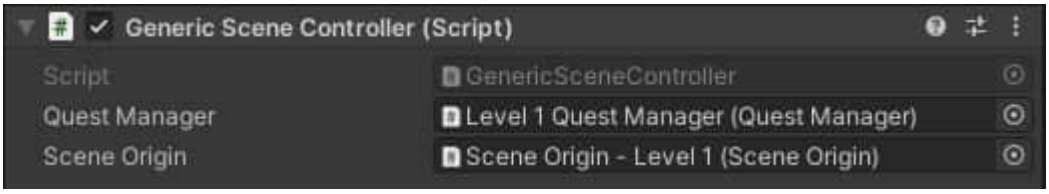

Рисунок 3.2.1. Серіалізовані поля компоненти GenericSceneController в інспекторі

Іншим методом оптимізації є пул об’єктів. Пул об’єктів - це набір ініціалізованих і готових до використання об'єктів. В плані швидкодії набагато дешевше вимкнути об’єкт та увімкнути його тоді коли знову потрібно його використати, ніж щоразу видаляти та створювати цей об’єкт. Таким чином, не витрачається зайвий час на ініціалізацію нового об’єкта та на очищення видаленого об’єкта з керованої пам’яті.

Останнім методом для оптимізації коду про який я згадаю, це обережне поводження з функціями Update. Якщо об’єкт має таку функцію - вона буде відпрацьовувати для нього кожен кадр. Якщо таких об’єктів, до прикладу десять, буде відбуватися десять однакових циклів Update методу. Як приклад візьмемо об’єкт, який чекає, поки гравець його підбере, навівши камеру на об’єкт. Правильним рішенням буде створити один скрипт, прив’язаний до гравця, який викликатиме івент. Якщо об’єкт підпишеться до цього івенту, то зможе відповідати на ввід користувача (в нашому випадку наведення камери на об’єкт). Зрештою, ми опрацьовуватимемо лише один цикл цього виду взаємодії користувача зі світом для усіх десяти об’єктів.

3.3 Оптимізація рендерингу

3.3.1 Рівень деталізації

Рівень деталізації (level of detail) – ця техніка застосовується в Unity та й в розробці відеоігор для зменшення кількості операцій, які має виконати GPU при рендерингу дальніх об’єктів. Суть полягає в тому, щоб показувати різні тривимірні моделі в залежності від відстані до камери (рис. 3.3.1.1).

Рисунок 3.3.1.1. Налаштування рівня деталізації

LOD 0 (рис. 3.3.1.2) - повна тривимірна модель, така, якою її має бачити гравець зблизька. LOD 1 (рис. 3.3.1.3) та LOD 2 (рис. 3.3.1.4) - спрощені моделі, які містять менше трикутників, відповідно займають менше часу для рендерингу. Режим Culled — коли гравець достатньо далеко рендеринг об’єкта не відбуватиметься.

Рисунок 3.3.1.2. LOD 0 має 1536 трикутників

Рисунок 3.3.1.3. LOD 1 має 922 трикутники

Рисунок 3.3.1.4. LOD 2 має 614 трикутників

Користь такого методу оптимізації не можна заперечувати. Звісно, це займає додатковий час у людей, які роблять моделі, щоб їх спрощувати, проте деякі моделі можна спрощувати автоматично, особливо для LOD 2. Попри це, беззаперечним залишається факт, що робота GPU оптимізується, оскільки кількість трикутників

для відтворення надзвичайно зменшується, крім того зменшується кількість викликів рендерингу об’єктів, що в свою чергу зменшує навантаження ще й на CPU.

3.3.2 Occlusion culling

Occlusion culling - процес, який запобігає виконанню обчислень для рендерингу об’єктів, які повністю приховані від камери. Кожен кадр, камери досліджують усі об’єкти на сцені, та виключають ті, які не треба відображати. Стандартним алгоритмом в цьому процесі Unity є використання піраміди огляду (рис. 3.3.2.1), проте її недолік в тому, що об’єкти, які приховані іншими об’єктами все ще відображаються. Перевага occlusion culling в тому, що навіть ті об’єкти, які приховані іншими, також виключаються з рендерингу (рис. 3.3.2.2).

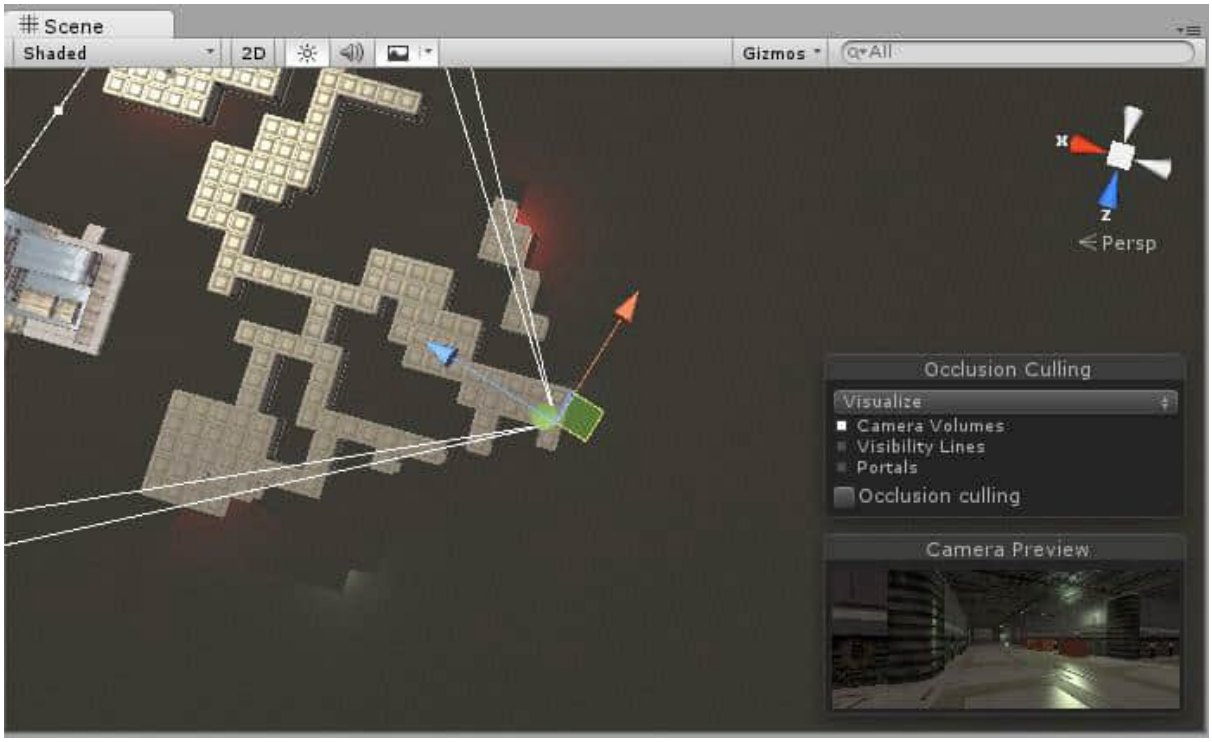

Рисунок 3.3.2.1. Піраміда огляду, всі об’єкти в радіусі огляду камери промальовуються

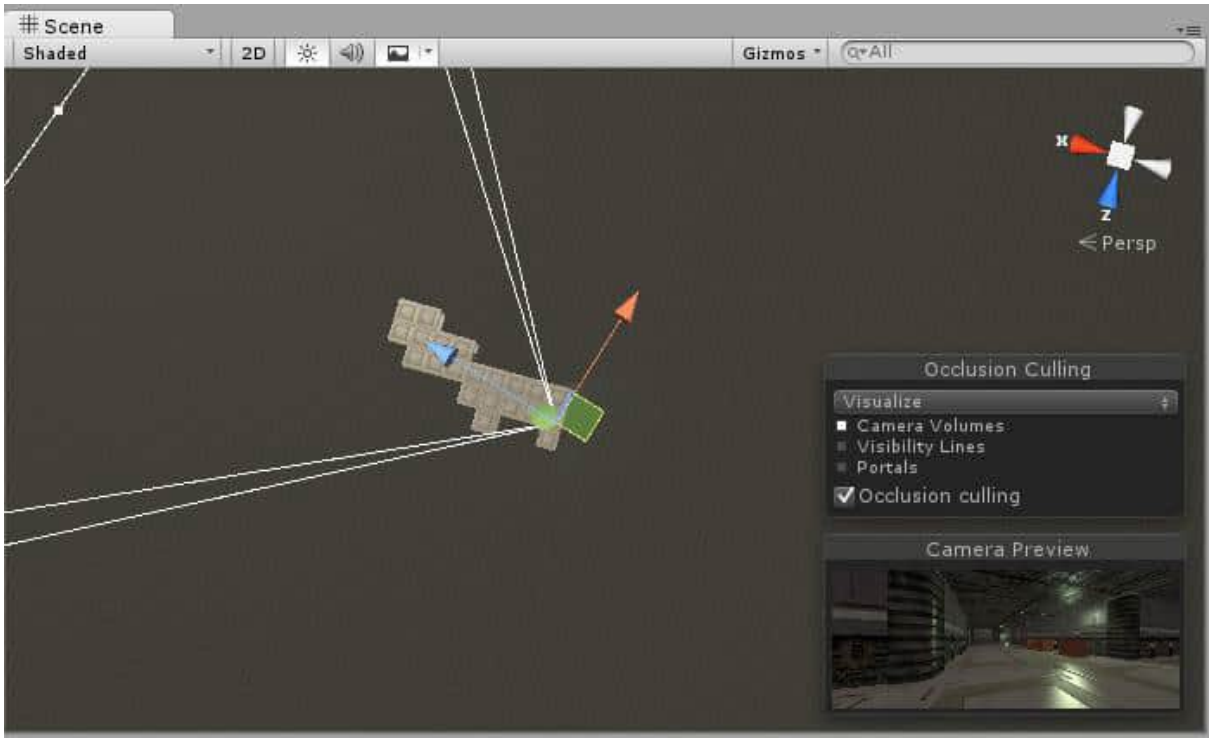

Рисунок 3.3.2.2. Occlusion culling в роботі, обчислюються лише об’єкти в безпосередній видимості камери

Для використання occlusion culling, необхідно запекти сцену, тобто провести складні обчислення на рівні розробки проекту, які у вигляді файлу будуть знаходитись в грі, що значно зменшить час подальших обчислень для виключення зайвих об’єктів з рендерингу.

3.3.3 Оптимізація асетів

Надзвичайно важливо слідкувати за ассетами, які надсилають інші члени команди розробки - розробники візуальних ефектів, аніматори, UI розробники, 3D розробники та інші.

Оптимізація UI компонент (двовимірної графіки), зазвичай передбачає використання розмірів зображення кратним четвірці, тобто текстури та спрайти повинні мати розширення 128x128, 256x256, 1024x1024 і т.д. Таким чином формат ETC2 може ефективно компресувати зображення, зменшуючи їх розмір та пришвидшуючи роботу з ними. Порівнявши некомпресоване зображення (рис. 3.3.3.1) та компресоване (рис. 3.3.3.2) - бачимо, що попри збільшення розширення зображення, його розмір зменшився вдвічі.

Рисунок 3.3.3.1. Зображення із неправильними розмірами, які не є кратними четвірці

Рисунок 3.3.3.2. Зображення із правильними розмірами 256x256

Оптимізація тривимірних об’єктів, які надсилають 3D дизайнери, зазвичай потребує більше уваги. Нехай 3D розробники надіслали файл із такими стендами (рис. 3.3.3.3).

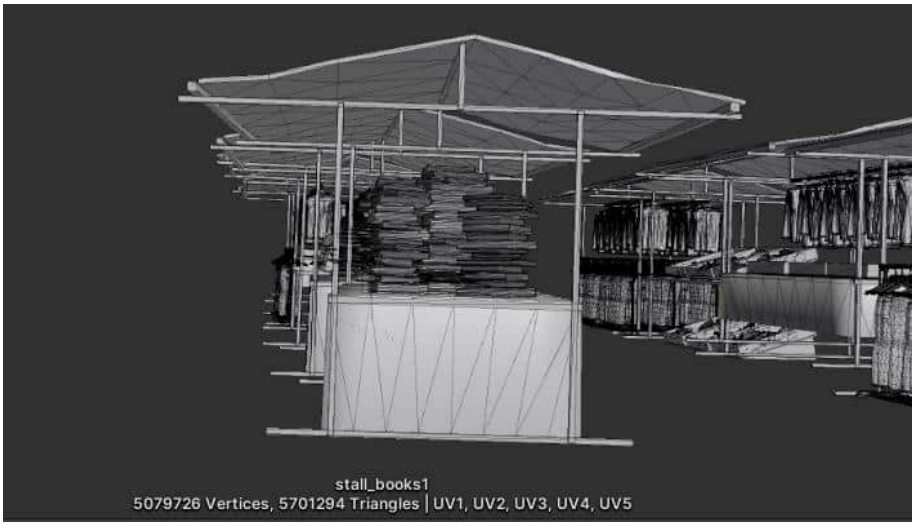

Рисунок 3.3.3.3. Приклад надісланого 3D об’єкту

Ця модель складається із практично ідентичних стендів, які мають ідентичні об’єкти, які накладені так, як це має бути у фінальному вигляді на сцені. Кількість трикутників - 5 мільйонів у лише цій моделі, а це ж ще далеко не усі об’єкти, які

мали би бути на сцені. До відома, рекомендована кількість трикутників у одній моделі - 3-4 тисячі для гри на персональні комп’ютери, і значно менше для гри на смартфони.

Набагато кращим підходом буде створення однієї моделі для стенду (рис. 3.3.3.4) та окремих моделей для кожного типу об’єктів (рис. 3.3.3.5), які будуть на цих стендах. Після цього можна буде вручну виставити декілька об’єктів такого типу на сцені і це працюватиме набагато ефективніше.

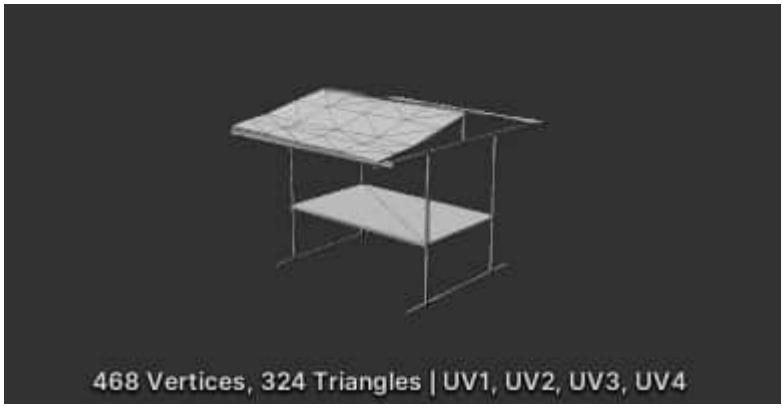

Рисунок 3.3.3.4. Оптимізована модель стенду

Рисунок 3.3.3.5. Оптимізована модель об’єкту, який знаходитиметься на стенді

Додатковою перевагою такого метода є можливість використання GPU Instancing у матеріалі об’єкта, де матеріал - це набір текстур та налаштувань для відображення кольорів об’єкта. Використання GPU Instancing дозволить опрацювати один матеріал, та використати його на абсолютно усіх об’єктах такого типу за єдиний виклик функції малювання.

4 РЕАЛІЗАЦІЯ

Основними кроками у реалізації будуть:

- налаштування доповненої реальності;
- організація об’єкту гравця;
- визначення та налаштування способів за допомогою яких гравець може взаємодіяти з різними об’єктами;
- визначення ієрархії сцени, оскільки вона має бути прив’язаною до AR компоненти, яка визначатиме позицію гравця та його розташування по координаті Y (висота гравця);
- побудова системи квестів, для створення якогось роду прогресії.

В результаті отримаємо застосунок з, як мінімум, одним робочим рівнем та широкими можливостями для подальшого розширення ігрового досвіду.

4.1 Об’єкт гравця

Після деяких налаштувань об’єкту гравця, я дійшов до такої ієрархії основного контролюючого компоненту в грі (рис. 4.1.1)

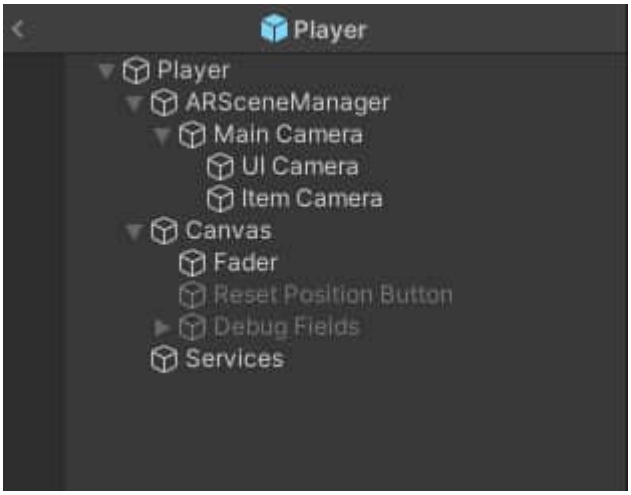

Рисунок 4.1.1. Ієрархія вкладень об’єкту гравця

Фактично найважливішим об’єктом у нас є ARSceneManager, який вміщує у собі усі компоненти відповідальні за запуск та роботу сесії доповненої реальності (рис. 4.4.2).

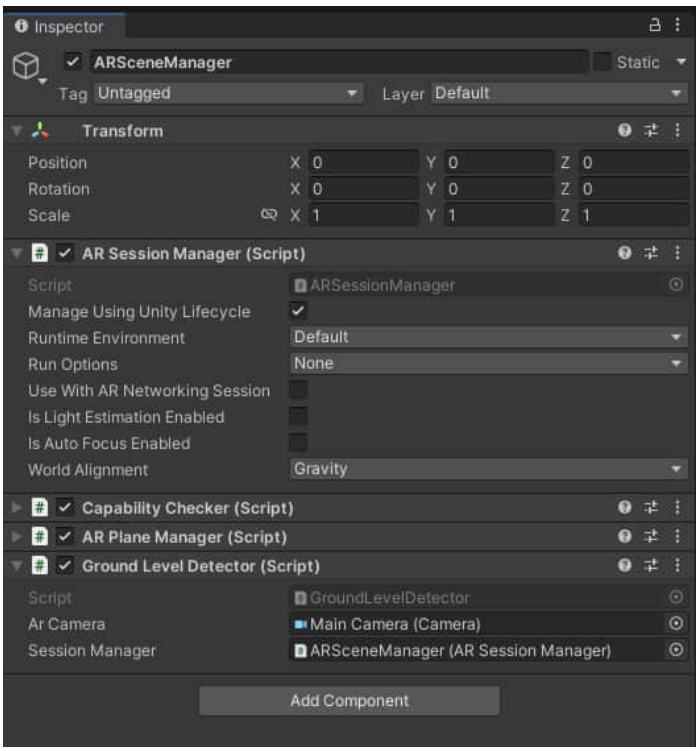

Рисунок 4.4.2. Об’єкт ARSceneManager

У об’єкті ARSceneManager (рис. 4.4.1) маємо вкладений об’єкт Main Camera, який в свою чергу має два інші об’єкти - UI Camera та Item Camera. Main Camera напряму керується сесією доповненої реальності. Дві вкладені камери використовуються окремо для тих елементів, які мають бути поверх решти світу, тобто для UI та підібраних об’єктів.

Далі маємо канвас, який вміщує декілька компонент з базовим функціоналом, спільним для всього застосунку: Fader (об’єкт для затемнення екрану, який потрібний, наприклад, для переходу між рівнями), Reset Position Button, який допомагає гравцю повернутись на стартову точку віртуального світу, не рухаючись у реальному та дебаг поля, які в результуючому продукті будуть неактивними. В подальшому можна інстанціювати інші UI елементи в процесі роботи застосунку.

Останньою компонентою гравця є сервіси - це усі можливі способи взаємодії з віртуальним світом, зібрані в одному місці. Коли умова використання сервісу виконується - він надсилає івент (напр. OnShakeDetected, коли акселерометр розпізнає швидкий струс телефоном). Якщо жоден об’єкт не прив’язувався до

цього івенту - нічого не відбувається, а виклик йде впусту. Детальніше про способи взаємодії гравця з віртуальним світом у пункті 4.2.

Об’єкт, який огортає решту компонент є сам Player (рис. 4.4.3).

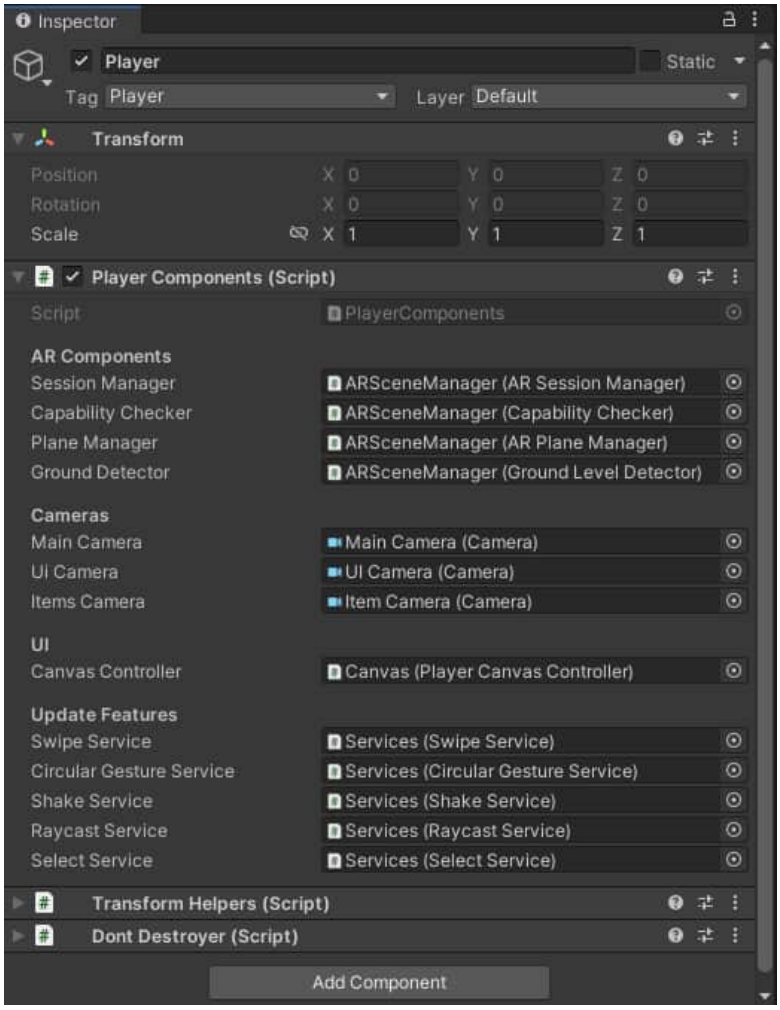

Рисунок 4.4.3. Об’єкт гравця

Цей компонент має тег “Player”, що дозволяє знайти його з будь-якого скрипту на сцені за допомогою FindGameObjectWithTag(string tag). Це дорога у виконанні функція, але за розумного використання вона дуже корисна. Окрім цього вона містить скрипт PlayerComponents, використовуючи який можна отримати посилання на будь-який компонент, вміщений у об’єкті гравця. Сам об’єкт Player є синглтоном, тобто він може існувати лише в одному екземплярі. Додатково він має скрипт DontDestroyer, який маркує його як такий об’єкт, який не має бути знищеним при зміні сцени, тож навіть змінюючи рівні ми залишатимемся з тією самою сесією доповненої реальності.

4.2 Керування

Пропоновані мною способи взаємодії гравця із віртуальністю:

- Фізичний рух у просторі
- Свайпи
- Використання акселерометра для визначення струшування телефоном
- Наведення девайсу на об’єкти та тригери
- Тапання по об’єктах та тригерах
- Складні свайпи (рух пальцем по екрану колом)
- Використання UI (user interface)

Такий набір інтеракцій є достатньо повним для виконання практично будь-яких дій користувачем. Проте, варто зауважити, що деякі з цих взаємодій можуть бути суперечливими у їх призначенні. До прикладу, якщо деякі об’єкти піднімаються навівши на них телефоном, другі тапаючи на них, а треті, якщо до них підійти достатньо близько - це збиватиме гравця з пантелику, оскільки іноді треба буде пробувати усі три підходи. Тож я пропоную використовувати лише наведення камерою на об’єкт.

Як уже згадувалось у пункті 4.1 усі ці способи інтеракцій є реалізованими у скриптах у об’єкті Services (рис. 4.2.1). Select Service ми не будемо використовувати, оскільки це потенційно може заплутати користувача.

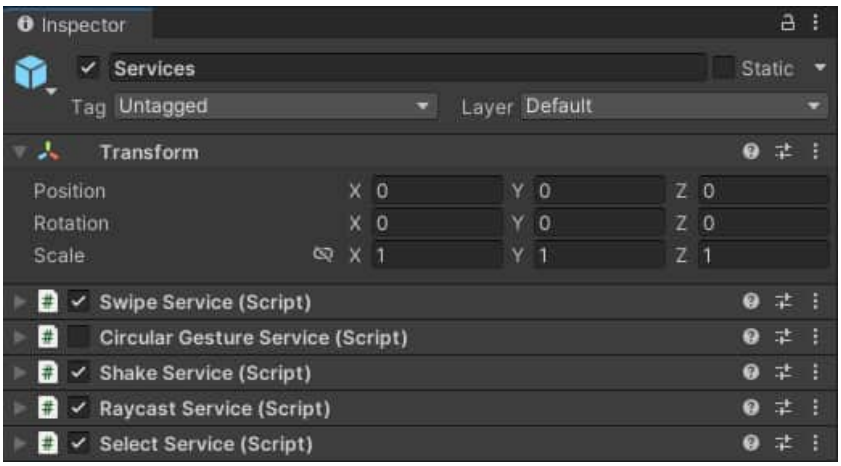

Рисунок 4.2.1. Сервіси взаємодії користувача

4.3 Побудова сцени

Оскільки ми працюємо з доповненою реальністю - центр сцени має бути визначений у живому часі відносно справжнього простору. Відповідно для цього треба розумно побудувати ієрархічну структуру сцени (рис. 4.3.1).

Рисунок 4.3.1. Базова ієрархія сцени

Власне об’єкт “Scene Origin - Level 1” з рисунку 4.3.1 передбачає використання уже знайденого центру сцени або очікування на ARSessionManager, який в процесі своєї роботи визначить центр сцени. Зокрема для визначення центру сцени (так би мовити, центру віртуального світу) у нас буде окремий рівень, який буде призначений чисто для цього початкового скану навколишнього простору.

Вкладеними в цей об’єкт ми маємо усі елементи, для яких важливо мати конкретне розташування всередині сцені - будівлі, елементи інтер’єру та об’єкти, з якими може взаємодіяти гравець.

Оскільки я планую будувати систему квестів - всі квести погруповані у вкладку “Interactions” та всі об’єкти, які потенційно можуть бути активними в цих квестах, належать відповідним батьківським об’єктам квестів. Сам по собі менеджер квестів не є тривимірним об’єктом, тож його переміщувати немає сенсу. Аналогічна ситуація з фоновою музикою (Background Audio), вона грає з однаковою гучністю незалежно від позиції гравця у сцені, тож переміщувати її з центром сцени не має ніякого сенсу.

4.4 Побудова квестової системи

На рис. 4.4.1 зображена архітектура системи квестів, яку я розробив для цього застосунку.

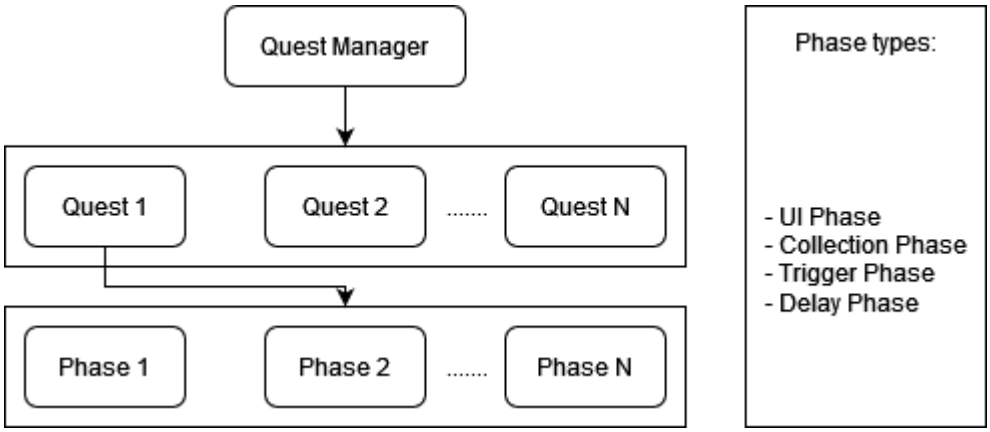

Рисунок 4.4.1. Архітектура системи квестів

Таким чином менеджер квестів може мати довільну кількість квестів, які він же й ініціалізує. Кожен з квестів може мати довільну кількість фаз, які в свою чергу можуть бути одним з чотирьох наведених типів:

- UI Phase - фаза взаємодії з інтерфейсом користувача, це, до прикладу, може бути якась двовимірна міні-гра;
- Collection Phase - фаза збирання та використання різних об’єктів;
- Trigger Phase - фаза, яка зазвичай лише запускає роботу інших фаз, до прикладу UI фази;
- Delay Phase - фаза, яка встановлює таймер очікування на певний час, поки виконується якась інша дія, до прикладу анімація;

Маючи таку систему можна створити квест практично будь-якої складності, зокрема завдяки модульній системі такого підходу.

4.5 Результат

Зрештою, ми маємо архітектуру проекту, готову до побудови практично будь-якої гри у доповненій реальності з використанням Real-Time Mapping. Витративши зовсім незначну кількість часу я запросто створив тестовий рівень, який охоплює практично усі вищенаведені можливості побудованої системи. Результатом є застосунок із:

- головним меню, яке просить доступ до камери і зрештою запускає гру;
- сценою для початкового скану навколишнього середовища, що забезпечує правильну роботу системи доповненої реальності;
- та згаданим вище тестовим рівнем.

Оскільки для створення повноцінної гри потрібні зусилля великого числа фахівців, які відповідають за візуальну частину застосунку я не бачив жодної можливості створення повноцінної гри. Проте засоби, які я описав вище цілком та повністю покривають усі необхідні потреби для потенційної розробки цілої гри.

ВИСНОВКИ

Як висновок, можна сказати, що технологія доповненої реальності достатньо розвинена і в належній мірі доступна, щоб створювати якісні та цікаві проекти уже зараз. За правильного планування та розумного використання наявних ресурсів сучасних систем смартфонів можна створити застосунок, який за насиченістю контентом не поступатиметься великим проектам на консолях чи персональних комп’ютерах.

Навіть команда із десяти людей, лише двоє з яких програмісти, може створити інтерактивний досвід практично будь-якого гатунку. Як показує ця дипломна робота, основну увагу при розробці слід звертати на оптимізацію, планування взаємодії користувача з віртуальним світом та розробку гнучкої архітектури проекту. Оскільки доповнена реальність відносно нова технологія у сфері розваг та й бізнесу - користувачі зачасту не знатимуть як нею користуватись, тож першочергове завдання команди розробки - це передбачати що очікуватимуть користувачі та розробляти систему відповідно до цих очікувань.

Зокрема, особливу увагу доповненій реальності важливо приділяти через все більшу популяризацію доповненої та й віртуальної реальностей. Зараз саме той час, коли можна стати провідним спеціалістом у сфері, яка в майбутньому лише лише набиратиме популярності.

І зрештою, ключова відмінність цього проекту від інших застосунків такого роду - це належне ставлення до оптимізації, що забезпечує приємне проведення часу, користуючись програмою. Окрім того, гнучка архітектура системи квестів дозволяє налаштувати різноманітні та досить унікальні один від одного завдання для проходження рівня, на відміну від інших застосунків, які мають досить одноманітні квести, зациклені на використанні лише одного певного виду керування у доповненій реальності.

ДЖЕРЕЛА

1. Effect of Frame Rate on User Experience, Performance, and Simulator Sickness in Virtual Reality [Electronic resource] / Jialin Wang, Rongkai Shi, Wenxuan Zheng, Weijie Xie, Dominic Kao, Hai-Ning Liang. - 2023. - Abstract – Available from: https://www.computer.org/csdl/journal/tg/2023/05/10049694/1KYopPcDKk8
2. Unity – Manual: Order of execution for event functions [Electronic resource] – Available from: https://docs.unity3d.com/Manual/ExecutionOrder.html
3. A Survey of Augmented Reality [Electronic resource] / Ronald T. Azuma. – 1997. – P. 2-3. – Available from: http://www.cs.unc.edu/~azuma/ARpresence.pdf
4. The use of virtual fixtures as perceptual overlays to enhance operator performance in remote environments [Electronic resource] / Louis B. Rosenberg. - 1992. - P. i – Available from: https://apps.dtic.mil/sti/pdfs/ADA292450.pdf
5. Benchmarking Built-In Tracking Systems for Indoor AR Applications on Popular Mobile Devices [Electronic resource] / Emanuele Marino, Fabio Bruno, Loris Barbieri, Antonio Lagudi. - 2022. - Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9320911/
6. Probabilistic Robotics. The MIT Press / Sebastian Thrun, Wolfram Burgard, Dieter Fox – P. 309
7. Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking [Electronic resource] / Andrei State, Gentaro Hirota, David T. Chen, William F. Garret, Mark A. Livinngston. - Available from: https://www.cs.princeton.edu/courses/archive/fall01/cs597d/papers/state96.pdf
8. A Survey on Augmented Reality Challenges and Tracking [Electronic resource] / Ihsan Rabbi, Sehat Ullah. - 2012. - Available from: https://hrcak.srce.hr/file/150828

9. Niantic Expands Developer Platform and AR Tools with Niantic Lightship [Electronic resource] – Available from: https://nianticlabs.com/news/lightship

10. Unity – Manual: Understanding the managed heap [Electronic resource] – Available from: https://docs.unity3d.com/2020.1/Documentation/Manual/BestPracticeUnderstandingPerformanceInUnity4-1.html

Додаток А

Керована пам’ять в Unity (managed memory) - це система, яка автоматично очищає пам’ять після того, як вона перестає використовуватись. Система керованої пам’яті використовує збирач сміття (garbage collector) та керовану купу (managed heap) для підчищення виділеної пам’яті після того як код перестає тримати посилання на об’єкти, які знаходяться у цій виділеній пам’яті. Ця система забезпечує застосунок від витоків пам’яті [10].

Коли утворюється новий об’єкт, Unity виділяє пам’ять необхідну для його зберігання у так званій керованій купі (рисунок А.1). Коли цей об’єкт більше ніде не використовується - Unity автоматично очищає його з купи, звільняючи місце для наступних об’єктів.

Рисунок А.1. Спрощений вигляд керованої купи. Зона А - дещиця вільної пам’яті

Коли Unity видаляє об’єкт, пам’ять, яку той займав, звільняється. Проте ця звільнена пам’ять не стає частиною великого запасу “вільної” пам’яті. Утворюється свого роду розрив між сусідніми сегментами зайнятої пам’яті, це називається дробленням пам’яті (memory fragmentation) (рисунок А.2).

Рисунок А.2. Приклад дроблення пам’яті, вивільнені елементи позначені сірою переривчастою лінією

Якщо необхідно добавити в купу великий об’єкт, проте там недостатньо вільного або суміжного вивільненого місця (рисунок А.3), керована пам’ять Unity виконує дві операції:

- Спершу, запускається збирач сміття, якщо він ще не запускався, і намагається звільнити достатню кількість пам’яті.
- Якщо після запуску збирача сміття все ще недостатньо суміжно вільного місця для внесення великого об’єкту, керована купа збільшується. Розмір на який купа збільшується залежить від платформи, на якій працює Unity, проте здебільшого вона збільшується вдвічі.

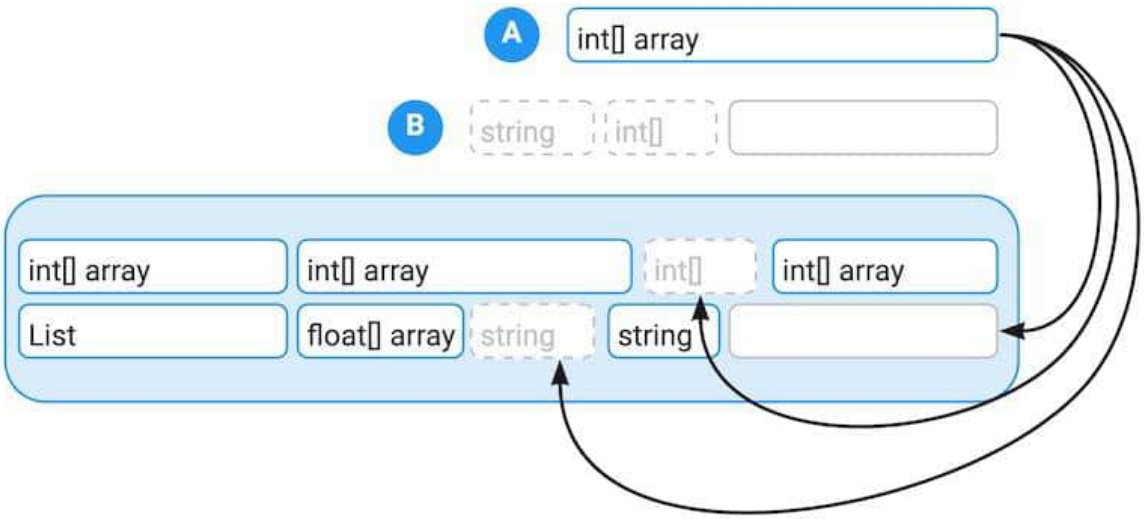

Рисунок А.3. Об’єкт А - новий об’єкт, який треба додати до купи, об’єкти В - вільна та вивільнена пам’ять. Навіть попри те, що загальної вільної пам’яті достатньо, щоб вмістити об’єкт А, ця пам’ять фрагментована і має запуститись збирач сміття.

Варто зазначити кілька деталей при роботі з керованою пам’яттю:

- Якщо з розширеної купи були вилучені усі об’єкти, які там знаходились, купа не зменшується до попереднього розміру. Це заради того, щоб у разі необхідності подальших виділень пам’яті, купа знову не збільшувалась.
- На більшості платформ, Unity зрештою повертає вивільнені фрагменти пам’яті назад до ОС. Проте інтервал такого повернення не є гарантованим.