Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

ESTENSIONE DI UN'INTERFACCIA DI GIOCO IN REALTA' MISTA: IL CASO DI UN SIMULATORE DI VOLO

Relatore:
Prof. Gustavo Marfia

Presentata da:
Giuseppe Di Maria

Correlatore:
Dott. Vincenzo Armandi
Dott. Lorenzo Stacchio

Sessione V
2022-2023

Sommario

Nel contesto delle tecnologie di realtà virtuale (VR), realtà aumentata (AR) e realtà mista (MR), l'uso di esperienze tridimensionali sta crescendo in diversi settori, architettura, ingegneria, costruzione, istruzione e formazione, intrattenimento e gaming. Tuttavia, nonostante la disponibilità di hardware VR più accessibile, l'integrazione di queste tecnologie nei software procede lentamente. Mentre la VR si sta lentamente diffondendo dai laboratori specializzati ai dispositivi di consumo, il passaggio dalla rappresentazione 2D all'ambiente 3D rimane una sfida aperta.
Questo lavoro mira a illustrare gli strumenti e gli approcci enunciati fino ad adesso, che sfruttano interfacce immersive, offrendo un quadro generale sull'implementazione per supportare la transizione da ambienti esclusivamente 2D a quelli 3D, estendendo la stessa transizione e rendendo le stesse interfacce immersive interattive secondo un approccio in MR. Si cerca dunque di rendere interattivo in MR un applicativo non pensato per essere utilizzato in un ambiente virtuale immersivo. L'obiettivo è aprire nuove prospettive per l'adattamento di applicativi desktop in ambienti virtuali, passando da spazi bidimensionali 2D a spazi tridimensionali 3D, estendendo quindi il supporto di tecnologie VR con un sistema in MR.
Utilizzeremo Microsoft Flight Simulator (MFS) come caso di studio per creare quest'estensione dell'applicazione che supporta solo VR, in una nuova esperienza in MR.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Nel panorama in continua evoluzione delle tecnologie di realtà virtuale (VR), realtà aumentata (AR) e realtà mista (MR), le esperienze tridimensionali stanno acquisendo sempre maggiore popolarità e rilevanza. Negli ultimi tempi si è verificata una significativa adozione di dispositivi MR in diverse aree applicative, tra cui architettura, ingegneria, costruzione, istruzione e formazione, intrattenimento e gaming. Tuttavia, anche se le applicazioni di tecnologie VR, AR e MR stanno guadagnando terreno, molti si interrogano sul futuro in cui le interfacce bidimensionali (2D) e tridimensionali (3D) convivranno armoniosamente nei sistemi informatici per offrire prestazioni ed esperienze migliorate.
Nonostante la diminuzione dei prezzi dei componenti hardware per VR, che incoraggia la diffusione di massa di dispositivi compatibili al grande pubblico, il processo di integrazione di questi paradigmi di interfaccia nelle piattaforme software procede a passo più lento, a causa di questo non è possibile osservare un aumento significativo del numero di applicazioni che aggiungono esperienze immersive.
Nel tempo sono state esplorate differenti soluzioni, ma che sono risultate costose e poco accessibili.
La mancanza di un approccio generale e semplice per il passaggio da ambienti 2D in un ambiente 3D, potrebbe ostacolare questa transizione, rischiando che l'hardware diventi abbordabile ma la disponibilità di soluzioni software adeguate rimanga limitata [6].
Alcune soluzioni proposte per l'implementazione di visualizzazioni VR includono la creazione di applicazioni native per la realtà virtuale, l'aggiunta di plugin per la VR a software esistenti e il porting di applicazioni 2D in ambienti VR tramite l'intercettazione di chiamate di rendering, per eseguirle successivamente in un ambiente virtuale che le supporti. Ad esempio, i plugin aggiuntivi consentono all'utente di interagire con le piattaforme software preesistenti o, in alcuni sistemi VR, esistono dei metodi per intercettare le chiamate alla libreria di OpenGL, e queste chiamate vengono utilizzate per acquisire ed eseguire il rendering di un'applicazione desktop all'interno di un display immersivo [23].

In questo elaborato l'idea fondamentale è di estendere applicativi desktop per spazi bidimensionali e portarli in spazi tridimensionali, sfruttando anche eventuali funzionalità VR supportate, come nel caso di MFS, trasferendole in un contesto MR. Si studia dunque un sistema che consenta una transizione in uno spazio tridimensionale e che permetta di ottenere un approccio MR su applicativi nativi per desktop, old-computing, che non sono stati pensati per essere usati in maniera così immersiva.

Il caso d'uso di particolare interesse è quello di un'esperienza di simulazione di volo da una realtà virtuale a una realtà mista, all'interno di un ambiente tridimensionale controllato. Con gli ultimi aggiornamenti infatti, MFS offre giocabilità in VR per consentire agli utenti di vivere una nuova esperienza di gioco più coinvolgente, utilizzando dispositivi compatibili. Tuttavia, lo stesso discorso non vale per le funzionalità MR, che all'interno del gioco non sono supportate. MFS offre agli utenti finali, di vivere solo un'esperienza completamente virtuale, immergendoli all'interno dello spazio di gioco in maniera interattiva. Tuttavia, il fatto che MFS non offra di vivere esperienze di gioco in MR, non impedisce di creare tale esperienza a partire dalla sua modalità VR già esistente, passando prima da un contesto bidimensionale a uno tridimensionale, e successivamente all'implementazione di questa estensione MR in uno spazio virtuale 3D controllato. Esploreremo le tecnologie di VR, AR e MR, le loro differenze e la loro evoluzione, e parleremo anche di OpenXR e di come quest'ultimo abbia migliorato l'approccio a queste tecnologie per gli sviluppatori, offrendo la compatibilità universale per i dispositivi che supportano queste tecnologie con diversi strumenti di sviluppo. Inoltre, le soluzioni adottate per il passaggio di applicativi da spazi bidimensionali a contesti tridimensionali e delle soluzioni proposte.
Esploreremo le motivazioni che conducono alla scelta di un simulatore di volo tramite MFS, preso come caso di studio per creare la transizione di applicativi desktop, con supporto alla VR in questo caso, portandoli da spazi bidimensionali a tridimensionali per provare a estenderlo alla realtà mista, e anche le tecnologie e i dispositivi utilizzati come il Varjo XR3, visore compatibile con la MR. Analizzeremo anche gli aspetti essenziali che un simulatore di volo deve avere e che deve soddisfare per essere considerato qualificato.

Il cuore di questa tesi risiede nell'architettura sviluppata, come vedremo più avanti in "Architettura del Simulatore in MR", per permettere questa transizione dalla VR alla MR, portando allo stesso tempo un applicativo desktop in un ambiente 3D. Particolare attenzione va fatta sul calcolo delle mappe di profondità che conferiscono al mondo virtuale una percezione tangibile di profondità e realismo. Questo approccio permette di offrire un'esperienza MR più coinvolgente e innovativa.
Esploreremo infine l'architettura sviluppata, che potrebbe aprire nuove possibilità sull'uso di applicazioni originariamente pensate per desktop, concepite solo per ambienti 2D, per renderle interattive in MR, spostandole in contesti tridimensionali coinvolgenti.

Capitolo 2

Stato dell'Arte

Con l'avvento della Realtà Mista (MR), della Realtà Aumentata (AR) e della Realtà Virtuale (VR), il modo in cui interagiamo con il mondo digitale e reale è profondamente cambiato. Queste tecnologie offrono esperienze immersive e coinvolgenti, consentendo agli utenti di sperimentare ambienti virtuali e di interagire con oggetti digitali all'interno del mondo reale. Secondo una delle definizioni più esplicative [3], illustrate nella Figura 2.1 che rappresenta il continuum realtà e virtualità, la tecnologia di Realtà Aumentata (AR) arricchisce la nostra percezione del mondo reale, ciò che percepiamo senza l'uso della tecnologia, aggiungendo elementi virtuali in tempo reale. La Realtà Virtuale (VR), invece, offre la simulazione di un'esperienza immergendo completamente gli utenti in un ambiente completamente virtuale. La Realtà Mista (MR) combina elementi reali e virtuali, consentendo l'interazione con oggetti digitali all'interno del mondo reale.

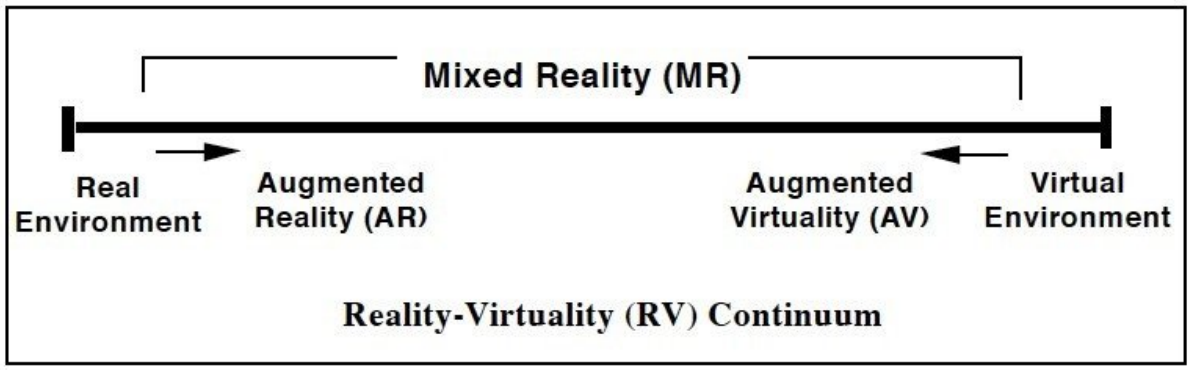


Figura 2.1: Reality-Virtuality Continuum

Questo documento si concentra sulle tecnologie che operano in un punto intermedio tra ambienti reali e ambienti virtuali completi e che sono collettivamente chiamate Realtà Mista. Esploreremo i progressi, gli sviluppi e le applicazioni di queste tecnologie, concentrandoci in particolare sull'utilizzo della Realtà Mista MR. Analizzeremo i vantaggi,

le sfide e le innovazioni che caratterizzano l'applicazione della Realtà Mista usando come caso di studio MFS come simulatori di volo, con un'attenzione particolare all'utilizzo del visore Varjo. Attraverso una panoramica esaustiva delle tecnologie di Realtà Mista, Realtà Aumentata e Realtà Virtuale, ci prepareremo a esplorare in dettaglio le ultime ricerche e gli sviluppi, offrendo una visione completa delle opportunità e delle sfide che questa tecnologia presenta.

2.1 Realtà Aumentata, Realtà Virtuale e Realtà Mista

La realtà virtuale e la realtà aumentata sono state ampiamente adottate in diversi ambiti, inclusi quelli aziendali e di intrattenimento. La VR offre un'esperienza completamente immersiva in un ambiente virtuale, consentendo agli utenti di interagire con oggetti e scenari digitali in modo realistico. È spesso utilizzata nell'industria dei giochi per creare mondi virtuali coinvolgenti e emozionanti. D'altra parte, l'AR arricchisce la realtà fisica sovrapponendo elementi digitali interattivi. Gli utenti possono vedere e interagire con contenuti virtuali direttamente nel contesto del mondo reale. L'AR ha trovato applicazioni in diversi settori, come l'architettura, l'ingegneria, la medicina e l'istruzione. Ad esempio, nella formazione aziendale, l'AR può essere utilizzata per simulare situazioni reali e fornire addestramento pratico senza i rischi associati. Tuttavia, una tecnologia emergente che sta guadagnando sempre più attenzione è la realtà mista. La MR combina gli elementi della VR e dell'AR, offrendo un'esperienza in cui gli oggetti virtuali sono integrati nel mondo reale in modo più realistico e interattivo. A differenza dell'AR, che sovrappone semplicemente elementi digitali alla realtà fisica, la MR permette agli utenti di interagire con gli oggetti virtuali in modo più naturale e realistico. Per interagire con queste tecnologie, esistono diverse piattaforme e dispositivi disponibili sul mercato. Per la VR, ci sono dispositivi come Oculus Rift, HTC Vive e PlayStation VR, che offrono un'esperienza di realtà virtuale completa. Per l'AR, abbiamo Microsoft HoloLens, Magic Leap e smartphone dotati di funzionalità AR. Per quanto riguarda la MR, il visore Varjo in Figura 3.1 è un esempio di dispositivo che combina VR e AR per fornire un'esperienza di realtà mista di alta qualità. L'utilizzo della VR, AR e MR può avere un impatto significativo nei contesti educativi e aziendali. Nel settore educativo, queste tecnologie offrono opportunità uniche per l'apprendimento immersivo, consentendo agli studenti di esplorare concetti complessi in modo pratico e coinvolgente. Nell'ambito aziendale, la VR, l'AR e la MR possono migliorare l'addestramento del personale, consentire la simulazione di scenari di lavoro reali e migliorare la collaborazione tra i dipendenti. In conclusione, la VR, l'AR e la MR rappresentano tecnologie in continua evoluzione che offrono nuove possibilità in diversi settori. La MR, in particolare, con la sua capacità di integrare in modo realistico con oggetti virtuali nel mondo reale, apre nuove prospettive per l'interazione e l'esperienza degli utenti. Sia nei contesti educativi che aziendali, queste tecnologie possono migliorare l'apprendimento, l'addestramento e la collaborazione, portando benefici tangibili e aprendo nuove opportunità di sviluppo.

2.2 OpenXR

L'industria della realtà virtuale (VR) è in continua evoluzione, con l'emergere di nuovi visori e applicazioni ogni anno. In passato, ogni produttore di visori VR ha sviluppato il proprio runtime e le proprie librerie, il che ha reso difficile per gli sviluppatori creare applicazioni compatibili con più di un tipo di visore. Ciò ha comportato una frammentazione del mercato VR, con diversi visori che utilizzano standard e API diverse. [21]

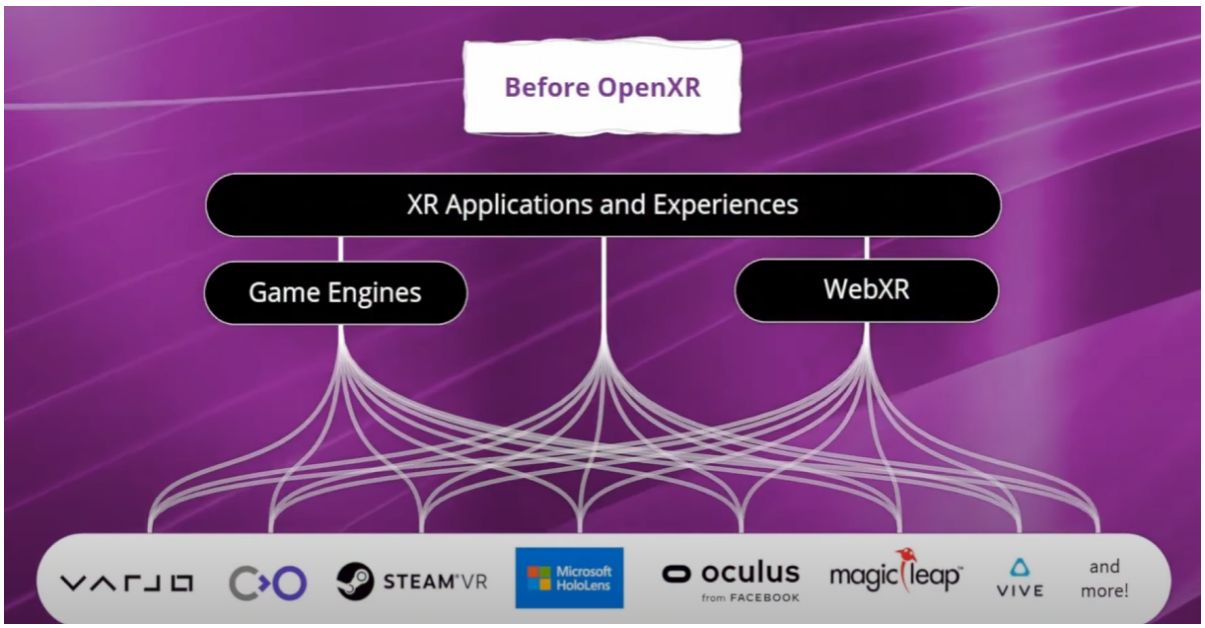


Figura 2.2: Scenario tecnologico esistente prima dell'avvento di OpenXR

Nel 2017, il Khronos Group ha annunciato OpenXR, un nuovo standard per lo sviluppo di applicazioni VR e AR. OpenXR mira a unificare lo sviluppo delle applicazioni VR, in modo che gli sviluppatori possano creare applicazioni che funzionino su qualsiasi visore VR compatibile con OpenXR. OpenXR offre una serie di vantaggi rispetto ai precedenti standard VR. Innanzitutto, è uno standard aperto, il che significa che è gratuito da usare e può essere implementato da qualsiasi produttore di visori VR. In secondo luogo, è un modello multipiattaforma, il che significa che le applicazioni OpenXR possono essere eseguite su Windows, macOS e Linux. In terzo luogo, è un modello flessibile, che consente agli sviluppatori di personalizzare le applicazioni in base alle specifiche esigenze dei propri visori. Da quando è stato annunciato, OpenXR ha ricevuto un ampio sostegno da parte dell'industria VR. Molti produttori di visori VR, tra cui HTC, Valve, Oculus e Microsoft, hanno annunciato il loro impegno a supportare OpenXR. Anche gli sviluppatori di giochi e applicazioni VR stanno adottando lo stesso approccio, con sempre più giochi e applicazioni che supportano lo standard. L'adozione di OpenXR ha il potenziale per

avere un impatto significativo sull'industria VR. Lo standard infatti ha il fine di unificare il mercato VR, rendendo più facile per gli sviluppatori creare applicazioni compatibili con più di un tipo di visore. Ciò potrebbe portare a un aumento della domanda di visori VR, poiché i consumatori avranno un'ampia scelta di applicazioni tra cui scegliere.

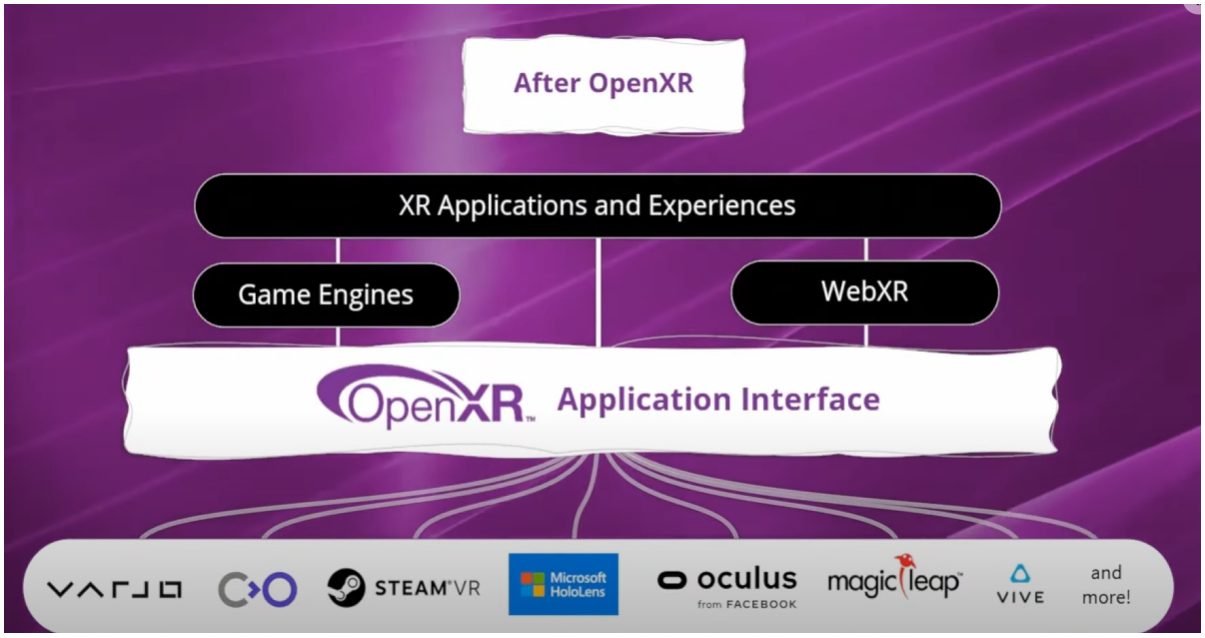


Figura 2.3: Scenario tecnologico esistente dopo all'avvento di OpenXR

OpenXR potrebbe aiutare a rendere la VR più accessibile, poiché è un modello flessibile, che consente agli sviluppatori di personalizzare le applicazioni in base alle specifiche esigenze dei propri visori. Ciò potrebbe portare a una maggiore varietà di applicazioni VR, progettate per diversi tipi di visori e rendere la VR più accessibile a un pubblico più ampio. In conclusione, OpenXR è uno standard importante che ha il potenziale per avere un impatto significativo sull'industria VR e potrebbe aiutare a unificare il mercato VR, rendere la VR più accessibile e promuovere lo sviluppo di nuove applicazioni VR.

2.3 Transizione da VR a MR

Fino a oggi, abbiamo assistito a un significativo progresso nella transizione da ambienti bidimensionali (2D) a quelli tridimensionali (3D) nell'ambito della realtà virtuale. Questa evoluzione ha coinvolto una vasta gamma di settori, ad esempio la visualizzazione scientifica, che utilizza la grafica per rappresentare dati, insiemi di dati e forme tridimensionali in modo sempre più immersivo [6], proprio per manipolare questi dati in maniera immersiva all'interno di una scena virtuale. Diversi approcci sono stati esplorati per portare l'immersione 3D in ambienti precedentemente 2D. Questi approcci includono la creazione di applicazioni native per la realtà virtuale, l'aggiunta di plugin per la VR a software esistenti e il porting di applicazioni 2D in ambienti VR tramite l'intercettazione di chiamate di rendering [22]:

- VR Nativo: l'interfaccia utente è progettata in 3D;
- Plugins per il VR: permettono all'utente di interagire con le piattaforme software tradizionali supportando il VR [2] [1];
- Porting in VR: la soluzione consiste nell'intercettare le chiamate effettuate dalla libreria di rendering (ad es. DirectX, OpenGL) e reindirizzarle a un'applicazione che le gestisca, eseguendole in un ambiente VR [7];

Alcune aziende come come Techviz o Moreviz hanno sviluppato soluzioni di porting basate su meccanismi di intercettazione di chiamate alla libreria di rendering OpenGL, per eseguirle in un ambiente virtuale, ma spesso queste soluzioni sono risultate troppo costose per il mercato e per i laboratori di ricerca accademici. Questi strumenti sono progettati per consentire a software sviluppati in un contesto 2D di funzionare in un ambiente 3D. Si studia quindi un nuovo modo che sia efficiente per portare le applicazioni in un ambiente 3D, soprattutto considerando l'evoluzione continua della realtà virtuale e della realtà mista.
Nel frattempo, anche l'industria del gaming ha fatto progressi significativi nell'integrazione della realtà virtuale nei suoi titoli, come nel caso di MFS, che è stato creato per raggiungere tre obiettivi chiave: realismo, precisione e autenticità del volo, e adesso supporta anche la modalità VR. MFS offre agli utenti finali, di vivere solo un'esperienza completamente virtuale, immergendoli all'interno dello spazio di gioco in maniera interattiva. Tuttavia, MFS natio non offre lo stesso supporto per vivere un'esperienza di gioco in MR, permettendo così all'utente di interagire con componenti virtuali all'interno dell'ambiente reale.
Il nostro problema dunque non solo si focalizza sul passare un applicativo da un contesto 2D a uno 3D, ma anche di renderlo compatibile con la tecnologia di MR. La nostra soluzione propone una comunicazione client-server, per spostare l'esecuzione di MFS in modalità VR all'interno di un ambiente virtuale come Unity, dove si ha il controllo dello spazio 3D per l'implementazione delle funzionalità MR. Quindi il contributo dello studio

di questa tesi mira a rendere compatibili in MR applicativi che in realtà non sono stato pensati per essere utilizzati secondo quest'approccio, scegliendo MFS come caso d'uso al fine di estendere l'esperienza di simulazione di volo, passando dalla VR alla MR.
Questo lavoro spinge verso una realtà mista, unendo il meglio della realtà virtuale e delle interfacce 2D per offrire esperienze ancora più immersive ed efficaci. Ci sposteremo all'interno di Unity per ottenere il controllo dell'ambiente tridimensionale, come vedremo in seguito, per estendere l'esperienza di MFS in VR alla MR.

2.4 Applicazione dell'Industria Aeronautica

Con l'espansione crescente della tecnologia VR (Virtual Reality) e MR (Mixed Reality) nel mercato, la necessità di migliorare la sicurezza nella formazione dei nuovi piloti è emersa come un punto critico all'interno dell'industria dell'aviazione.

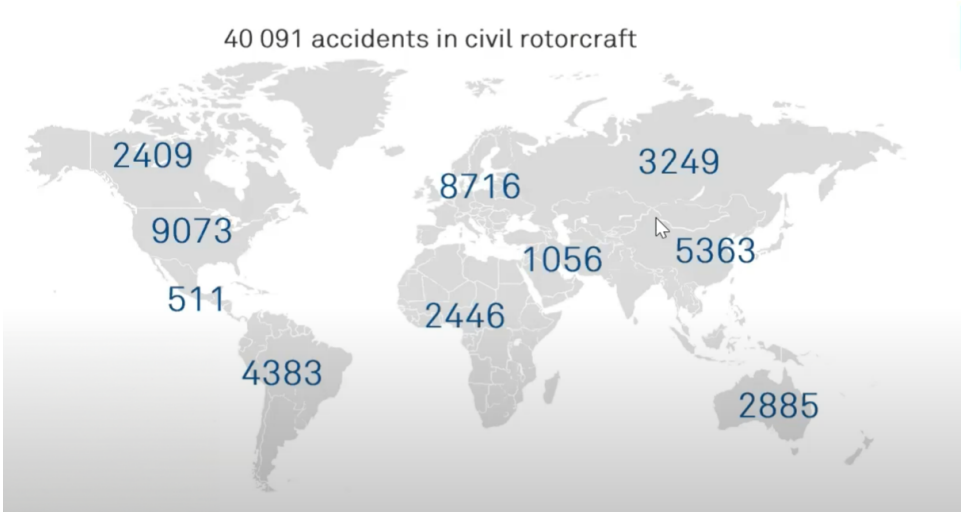


Figura 2.4: Numero di incidenti aerei avvenuti negli ultimi due anni

Come evidenziato dalla Figura 2.4, secondo le analisi svolte dalla EASA negli ultimi due anni hanno rivelato che il numero di incidenti aerei rimane ancora significativamente elevato. Nonostante l'evoluzione dei simulatori di volo, finora non si è registrata una significativa riduzione di tali incidenti. Tuttavia, lo sviluppo di simulatori di volo, non ha ancora decrementato molto i numeri di questi dati, ma con il tempo, grazie alla combinazione della tecnologia VR e MR con simulatori di volo qualificati offre prospettive promettenti per ridurre in modo sostanziale questi dati negli anni a venire. La chiave per migliorare la sicurezza e la preparazione dei piloti risiede nell'offrire esperienze di simulazione estremamente realistiche e al contempo accessibili. Questo obiettivo richiede lo sviluppo di simulatori di volo certificati, i quali rivestono un ruolo cruciale nel garantire

la qualità e l'efficacia dell'addestramento dei piloti. In questo contesto, l'acquisizione di una qualifica certificata diventa un elemento fondamentale per l'intero prodotto, contribuendo a plasmare il futuro della formazione dei piloti e la sicurezza delle operazioni aeree.

Figura 2.5: Simulatore di Volo in VR

Nell'esempio in Figura 2.5 è illustrato uno screen di un simulatore di volo in VR. Come si può osservare, l'utente si trova immerso nel mondo virutale all'interno della cabina di pilotaggio, dove può interagire con i comandi della cabina di pilotaggio e girarsi con la testa, percependo anche il senso di profondità della scena. Con la realtà mista invece l'esperienza della simulazione di volo viene resa ancora più realistica per i nuovi piloti, garantendo l'immersione nell'ambiente virtuale, integrando gli elementi che ne fanno parte con l'ambiente reale e rendendoli interattivi. I vantaggi di un simulatore in MR come in Figura 2.6, rispetto ad un simulatore in VR, consentono dunque di combinare un mondo virtuale, altamente personalizzabile, con un flusso video dell'ambiente reale. Si possono, ad esempio, mostrare le mani del pilota e parti rilevanti del mockup fisico della cabina di pilotaggio, arricchito con elementi e la vista virtuali fuori dal finestrino. In una simulazione in MR, i piloti possono interagire in modo naturale e diretto con dispositivi di input convenzionali immergendosi allo stesso tempo nell'ambiente virtuale [8]. Grazie all'utilizzo di simulatori di volo dunque aumenta l'accessibilità e la disponibilità per la formazione in sicurezza. Per esempio, non c'è più bisogno di viaggiare in una località specifica dove le spese possono risultare molto onerose. L'avvento dei simulatori di volo qualificati fa si che l'addestrameto risulti anche più veloce ed efficiente, e questo garantisce all'utente una formazione che non è solo una simulazione, ma una vera e propria esperienza.

Figura 2.6: Simulatore di Volo in MR

Motion Compensation

Come si può vedere dalla Figura 2.7, in un contesto aziendale, il simulatore di volo è fornito di una piattaforma di simulazione che emula la cabina di pilotaggio, che è dinamica e che segue i movimenti del simulatore durante la sessione di addestramento del pilota. La piattaforma utilizzata per la fase di addestramento, non è solo un dispositivo, con i suoi diversi gradi di movimento, ma come già documentato per il simulatore di volo, dev'essere anche qualificato e soddisfare speifici requisiti. Un problema riscontrato durante la ricerca è che risulta difficile per l'headset capire dove il pilota sta effettivamente guardando durante la simulazione. Ecco perché il Varjo ha anche un sistema di tracciamento che viene definito "Motion Compensation", che consiste nel sottrarre il movimento della piattaforma di simulazione dal movimento della testa

Figura 2.7: Benefici per i Piloti

2.5 Requisiti Essenziali dei Simulatori di Volo

Benefici per i Piloti

I sistemi di simulazioni di volo qualificati consentono agli utenti di acquisire le competenze necessarie ad apprendere la fasi di volo, al fine di poter volare in sicurezza in un contesto reale, e far sì che questi avvertano effettivamente la sensazione di pilotaggio in modo molto dettagliato e coerente, come se si trovassero realmente su un aereomobile. Ciò significa che deve esistere un vero e proprio modello fisico all'interno del sistema di simulazione, che permetta ad esempio il volo ad alta quota, il volo a bassa quota e le operazioni di trasporto di un carico con un imbracatura esterna, ecc. Però esistono allo stesso tempo delle limitazioni, ad esempio come quella di non poter avvertire effettivamente il peso di un eventuale carico da trasportare da un punto A a un punto B durante la simulazione. Proprio per questo motivo, ogni velivolo che si desidera emulare dev'essere opportunatamente calibrato per garantire la più realistica esperienza di volo per i nuovi piloti.

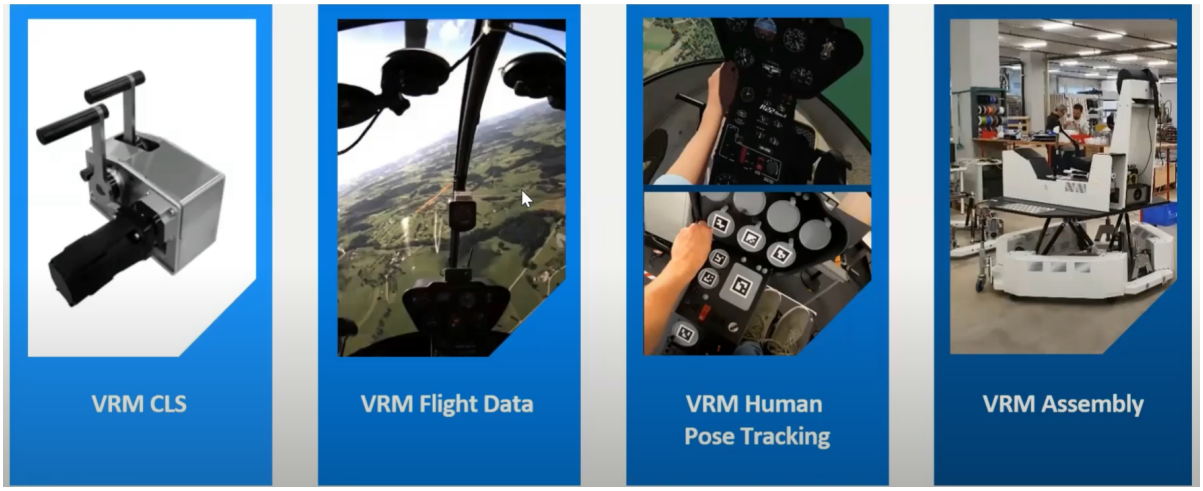


Figura 2.8: Componenti per un simulatore di volo

Inoltre, la possibilità di registrare e analizzare le sessioni di volo consente ai piloti di rivedere le proprie prestazioni e identificare le aree di miglioramento della loro esperienza. Infine, l'interattività con altri utenti consente ai nuovi piloti di esercitarsi anche nei voli di gruppo. I simulatori di volo quindi devono soddisfare una serie di requisiti essenziali per essere utilizzati per l'addestramento di nuovi piloti.
Alcuni di questi requisiti includono:

- Realismo: Il simulatore deve essere in grado di simulare l'ambiente di volo in modo realistico, in modo che i piloti possano sviluppare le competenze necessarie per affrontare situazioni reali. Ciò include la capacità di simulare le condizioni atmosferiche, i sistemi dell'aereo, le condizioni e i movimenti di volo.

- Sicurezza: Il simulatore garantisce sicurezza per i piloti in addestramento. Ciò significa che deve essere progettato per prevenire danni a persone o proprietà nel mondo reale.

- Affidabilità: Il simulatore deve essere affidabile e funzionare correttamente durante le sessioni di addestramento. Ciò è importante per garantire che i piloti ricevano l'addestramento di cui hanno bisogno [9].

- Flessibilità: Il simulatore deve essere in grado di simulare una varietà di aerei e situazioni di volo. Ciò è importante per garantire che i piloti possano essere addestrati per vari tipi di attività.

Oltre a questi requisiti essenziali, i simulatori di volo possono includere anche altre funzionalità che possono migliorare l'efficacia dell'addestramento. Queste funzionalità includono:

- Simulare situazioni di emergenza: La possibilità di simulare situazioni di emergenza consente ai piloti di acquisire le competenze necessarie per gestire tali situazioni di pericolo in modo sicuro e rapido.

- Registrare e analizzare le sessioni di volo: La registrazione delle sessioni di volo consente ai piloti di rivedere le proprie prestazioni e identificare le aree di miglioramento della loro esperienza. Inoltre, i dati raccolti possono essere utilizzati per sviluppare nuovi metodi di addestramento.

- Interattività con altri piloti: L'interattività con altri piloti consente ai nuovi piloti di esercitarsi in situazioni di volo di gruppo, come il decollo e l'atterraggio in formazione. Ciò è importante per sviluppare le capacità di comunicazione e collaborazione.

. L'utilizzo di simulatori di volo per l'addestramento di nuovi piloti offre una serie di vantaggi rispetto all'addestramento in volo reale. I simulatori sono più economici, e offrono un ambiente più sicuro in cui i piloti possono acquisire le competenze necessarie per gestire molteplici situazioni in un contesto reale. In conclusione, i simulatori di volo sono strumenti essenziali per la formazione di nuovi piloti. I simulatori che soddisfano i requisiti essenziali e includono funzionalità aggiuntive possono migliorare l'efficacia dell'addestramento e aiutare a formare nuovi piloti in modo più sicuri.

Capitolo 3

Tecnologie Utilizzate

Questa sezione fornirà una panoramica approfondita delle componenti chiave del sistema, delle interazioni tra di esse e dei processi coinvolti nella realizzazione di tale esperienza di volo immersiva. Per questo scopo, una sfida particolare è stata quella di rendere l'ambiente virtuale della simulazione con MFS in grado di fornire, all'utente finale, anche un senso di profondità della scena finale. Infatti, è stato utilizzato il modello di Machine Learning detto MiDaS (Multiple Depth Estimation Accuracy with Single Network). Stimare la profondità in modo accurato da una singola immagine non è semplice. I recenti lavori basati sulle reti neurali convoluzionali profonde (Deep Learning) mostrano grandi progressi con risultati plausibili. Le reti neurali convoluzionali sono generalmente composte da due parti: un codificatore per l'estrazione di caratteristiche dense e un decodificatore per la previsione della profondità desiderata. MiDaS si basa sull'apprendimento profondo e rappresenta una soluzione avanzata per stimare mappe di profondità da immagini monoculari. Una volta passati da un contesto bidimensionale a uno tridimensionale per MFS, si procede, dall'elaborazione della scena di gioco con il contributo sul calcolo della profondità, allo sviluppo della transizione per passare dalla modalità VR alla modalità MR.

3.1 Varjo XR3

Il Varjo XR3, illustrato in Figura 3.1 è tra i dispositivi di realtà virtuale e realtà mista basata su *pass-through*, con una delle risoluzioni più alte sul mercato e consente di vedere i dettagli delle scene virtuali come nelle situazioni reali. Le specifiche tecniche del Varjo sono:

- display integrato uOLED, con risoluzione di 1920 x 1920 px. per ciascun occhio;
- area periferica a oltre 30 PPD LCD, 2880 x 2720 px. per ciascun occhio;
- colori: 99% sRGB, 93% DCI-P3.

Figura 3.1: Varjo XR3

Questi display oculari ad alta risoluzione sono in grado di renderizzare immagini di alta qualità, aumentando il fattore di immersione per l'utente. Il Varjo XR-3 in Figura 3.1 ha un sensore di tracciamento incorporato che opera a 200 Hz. Ciò significa che è in grado di annotare e registrare la posizione degli occhi dell'utente, il diametro delle pupille, il punto di sguardo, la posizione degli occhi, il diametro della pupilla, la stima del punto di sguardo nell'ambiente virtuale, nonché la posizione della testa e la direzione della visuale. Inoltre, Varjo XR-3 è dotato di un supporto integrato per i sensori di tracciamento posizionale di SteamVR 2.0. che può essere incluso come parte del sistema di calibrazione, illustarto in Figura 3.2. I sensori di tracciamento di localizzazione posizionale, noti anche come stazioni di base, funzionano a 100 Hz, hanno una risoluzione di 160°orizzontale e 115°verticale e sono in grado compiere il tracciare fino a 7 metri. Da qualche anno sono in corso diversi progetti di ricerca, proprio mirati allo sviluppo di simulatori di volo per l'addestramento e la formazione di nuovi piloti, come ad esempio un progetto il cui obbiettivo era quello di raccogliere e monitorare dati psico-biologici, o monitorare il battito cardiaco dei piloti durante le simulazioni [19]. Con l'utilizzo di questa tecnologia nell'ambito dell'aviazione, è stato sorpassato un importante problema relativo alla sicurezza degli aereomobili. Nonostante questo, nel 2018 l'autorità aeronautica europea, e molti altri enti, hanno segnalato l'esistenza di alcune lacune che potrebbero in futuro essere causa di incidenti, ed è per questa ragione che è nata la necessità di avere simulatori di volo realistici. Ecco il motivo per cui la ricerca in questo settore coinvolge anche piloti esperti, al fine di ottenere una importante qualifica per lo sviluppo di questi prodotti.

Figura 3.2: Headset con sensori spaziali (SteamVR)

Se si costruisce un simulatore di volo è facile pensare che tutti potrebbero usarlo solo come videogioco, o anche per vivere una semplice esperienza di volo in VR, tuttavia se il fine è quello di addestrare e formare nuovi piloti, è necessario soddisfare standard molto specifici, per evitare quello che viene definito *Training Negativo* [20]. *Training Negativo* significa che si prova una sensazione di "piacere" durante la fase di addestramento con il simulatore, tramite il quale il pilota svilupperà una propria memoria muscolare, programmata per queste simulazioni ma che potrebbe non rispecchiarsi efficientemente in un contesto reale. I movimenti fisici "lontani" dalla realtà potrebbero avere un impatto negativo quando l'utente viene trasferito dall'interazione nella simulazione, all'ambiente reale, soprattutto nel caso in cui sia richiesta una forte interazione con gli oggetti fisici, e che non porterà alcun beneficio all'utente in un contesto reale [4]. Ed è per questo motivo che si deve passare attraverso un enorme lavoro di regolamentazione per questo tipo di tecnologia. Quindi ogni simulatore di volo, una volta sviluppato, neccessita di essere qualificato. Se il simulatore di volo può essere qualificato per il training dei nuovi piloti, professionalmente viene chiamato "Dispositivo di Addestramento di Simulatore di Volo", o FSTD (Flight Simulator Training Device). L'utilizzo di simulatori di volo risulta convieniente nei contesti privati poiché ha un ottimo impatto sui costi economici per la formazione e sulla sicurezza, sia per gli utenti che per gli aereomobili, Uno dei vantaggi più evidenti dello sviluppo di simulatori di volo in VR o in MR è rappresentato dalla riduzione dei costi che questi comportano. In genere, la maggior parte delle componenti di un simulatore di volo di alta qualità e qualificato, si aggirano intorno ai 4000$ (2022). È dunque evidente che un simulatore di volo completo e qualificato dalla FAA, presenti un netto vantaggio in termini di costi. Poiché l'addestramento di nuovi

piloti richiede molto tempo e denaro (in genere un istruttore con due studenti) ed è molto costoso da gestire. Dunque, attraverso simulatori di volo qualificati, il tempo del lavoro di preparazione farebbe risparmiare tempo e denaro. È inoltre molto più facile condividere e registrare l'esperienza virtuale di un utente con altri studenti [5].

3.2 Realtà Mista con Varjo XR3

Il Varjo XR-3 combina le funzionalità VR e AR in un unico dispositivo. Due telecamere sulla piastra frontale dell'headset consentono di vedere il mondo reale con oggetti virtuali posizionati al suo interno. Al contrario, è possibile scegliere di inserire una porzione più piccola del mondo reale in un ambiente virtuale più ampio. Alcune delle features principali utilizzate del Varjo, relative alla tecnologia MR:

- Masking;
- Varjo Markers;

Attualmente Varjo fornisce implementazioni per i suoi dispositivi XR per Unreal e Unity con il suo SDK nativo. Se il progetto esistente è stato scritto in Unreal o Unity, è possibile eseguirne il porting sui dispositivi XR di Varjo, poiché entrambi i motori di rendering sono supportati. Tuttavia, se l'applicazione è stata sviluppata in modo nativo, è necessario riscriverla per supportare l'API Varjo.

Masking per la MR

Dopo aver abilitato il plug-in di Varjo SDK nell'ambiente di un progetto in Unity, siamo pronti per implementare una prima scena in MR. Per rendere un applicazione in MR è necessario abilitare il video-pass trhough del dispositivo, e per questo è necessario utilizzare una maschera all'interno della scena di Unity. Per fare questo possiamo creare un nuovo materiale. Per il materiale è necessario assegnare uno shader in grado di esportare l'alfa e di scrivere nel buffer di profondità. È possibile utilizzare lo shader Unlit predefinito di HDRP (High Definition Rendering Pipeline) o crearne uno personalizzato. Quando si usa HDRP e Unlit, assicurarsi che il Tipo di superficie sia Opaco e impostare il Colore su RGBA(0,0,0,0). Poiché i livelli Varjo utilizzano l'alfa premoltiplicato, il solo canale alfa a 0 non è sufficiente. Assicurarsi inoltre che il colore RGB sia nero con lo shader predefinito Unlit, poiché non contribuisce e non è influenzato da eventuali contributi luminosi presenti nella scena virtuale. Una volta che la maschera VST è pronta, si può vedere il mondo reale attraverso la mesh della scena virtuale a cui il materiale è associato. Si noti che l'immagine VST non è visibile nell'Editor Unity. È possibile vederla all'interno dell'headset grazie al video-pass through del Varjo.

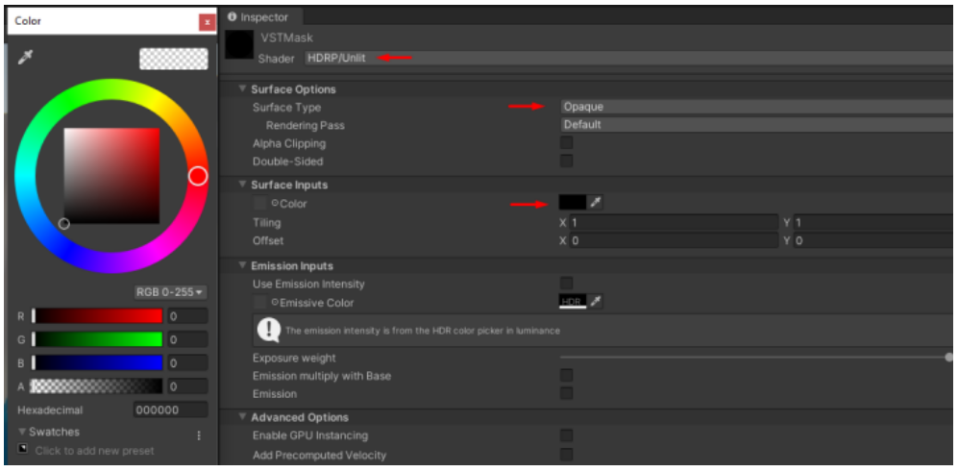


Figura 3.3: VSTMask Material

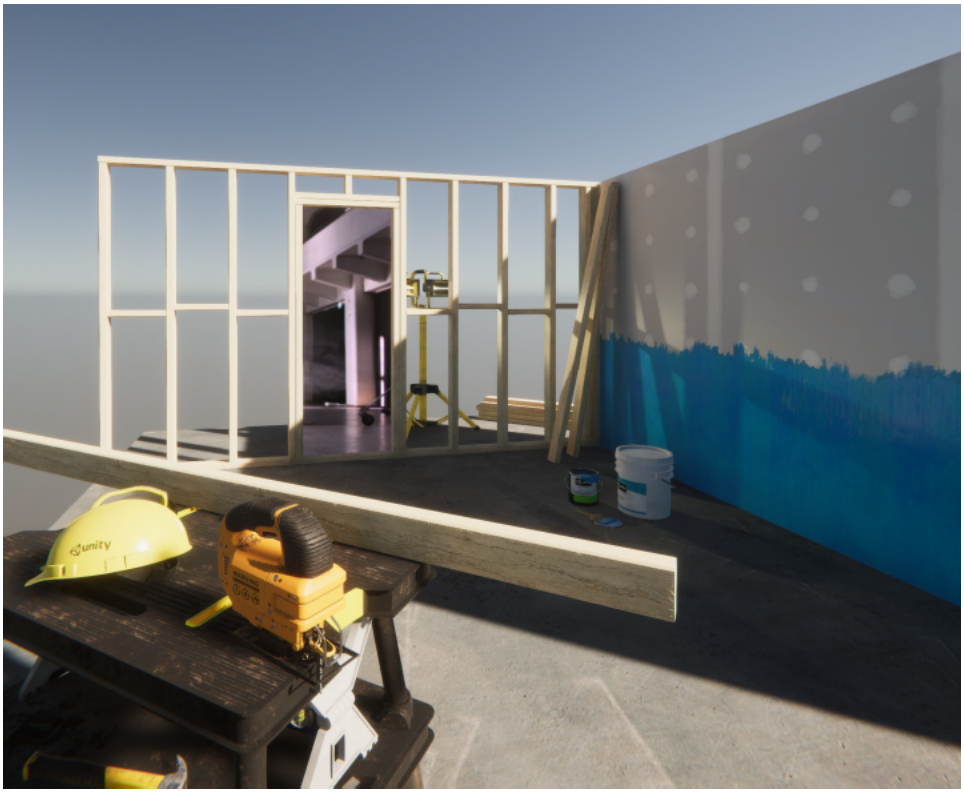
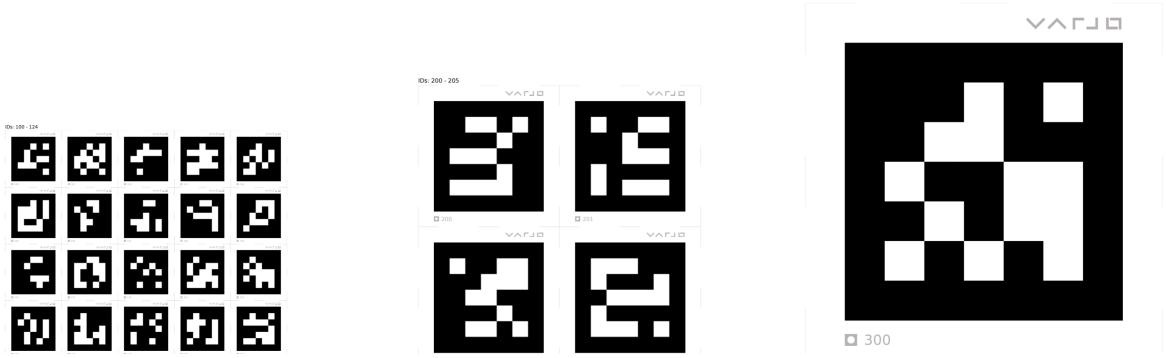


Figura 3.4: Masking con Varjo

Varjo Markers

I marcatori Varjo sono marcatori fisici, tracciati dalle videocamere passanti del Varjo XR-3. Questi oggetti vengono utilizzati per tracciare oggetti statici o dinamici nell'ambiente utente. È possibile utilizzare i marcatori di oggetti sia nelle applicazioni XR che in quelle VR. Ognuno di questi oggetti marcatori ha un ID univoco ad esso associato, e non si dovrebbe usare lo stesso marcatore più di una volta in un determinato ambiente. Per una maggiore precisione, un'applicazione può utilizzare più marcatori per tracciare un singolo oggetto. Ad esempio, è possibile tracciare un monitor, posizionando un marcatore diverso in ogni angolo. I marcatori utilizzano il tracciamento statico per impostazione predefinita, ma è possibile impostare, a seconda del caso, anche tracciamenti dinamici per oggetti in movimento. In generale, il tracciamento funziona meglio con marcatori di grandi dimensioni e soprattutto quando i marcatori sono più lontani dall'headset. Le dimensioni dei marcatori e le rispettive distanze di tracciamento sono elencate nella tabella 3.1



(a) Small size (b) Medium size (c) Large size

Figura 3.5: Esempi di Marcatori

Marker size	Tracking Distance
Small	Up to 0.5 m (1.5 ft)
Medium	Up to 1 m (3 ft)
Large 5	Up to 3 m (9 ft)

Tabella 3.1: Distanze di Tracciamento

Allineare una maschera ad un oggetto del mondo reale

Uno degli usi più comuni dei marcatori Varjo è l'allineamento di una maschera video pass-through con un oggetto fisico nell'ambiente, in questo caso illustreremo l'esempio di una tasiera. Ciò è utile per portare nell'ambiente virtuale oggetti come dispositivi di input o strumenti dal mondo reale. Una volta creato un nuovo *GameObject* in Unity, e gli si associa il materiale della maschera, si deve aggiungere lo stesso oggetto nell'array degli oggetti tracciati con l'ID del marcatore che si sta utilizzando, come illustrato in Figura 3.6 Posizionare il marcatore stampato accanto alla tastiera. In genere, si consiglia di collegare il marcatore all'oggetto fisico, in modo che si muovano insieme, come in Figura **??**. Si aggiunge un nuovo *GameObject* in Unity (si consiglia con la stessa forma dell'oggetto reale), come figlio del *GameObject* tracciato, precedentemente creato con il materiale della VST Mask associato. All'avvio l'oggetto reale, che seguirà il centro marcatore, sarà quindi visibile all'interno del mondo virtuale.

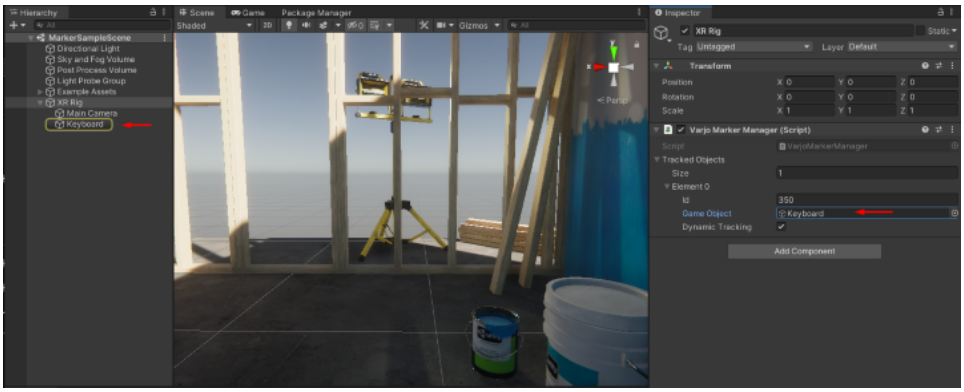


Figura 3.6: GameObject tracciato

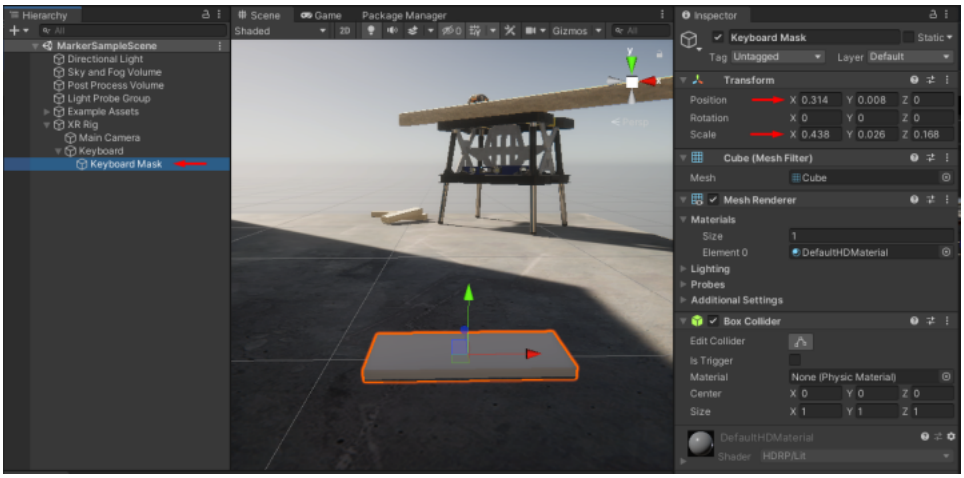


Figura 3.7: Nuovo GameObject

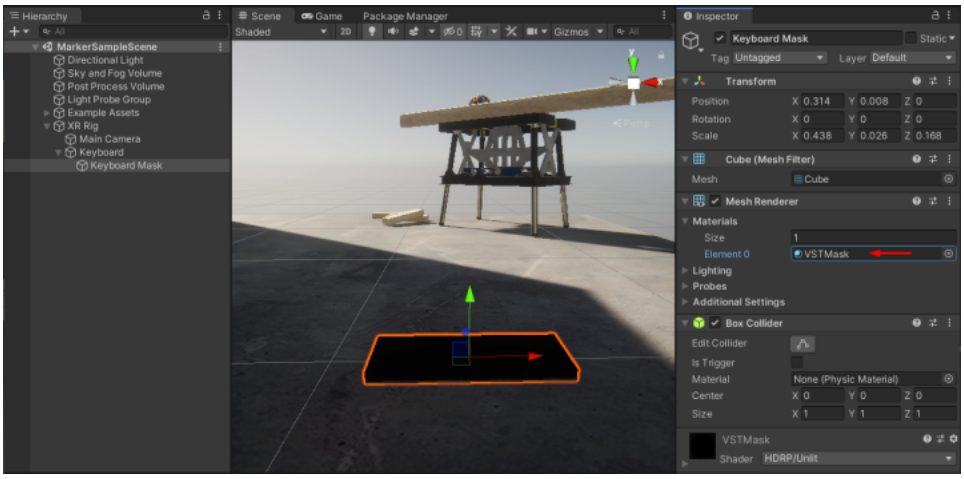


Figura 3.8: Oggetto Reale nell'Ambiente Virtuale

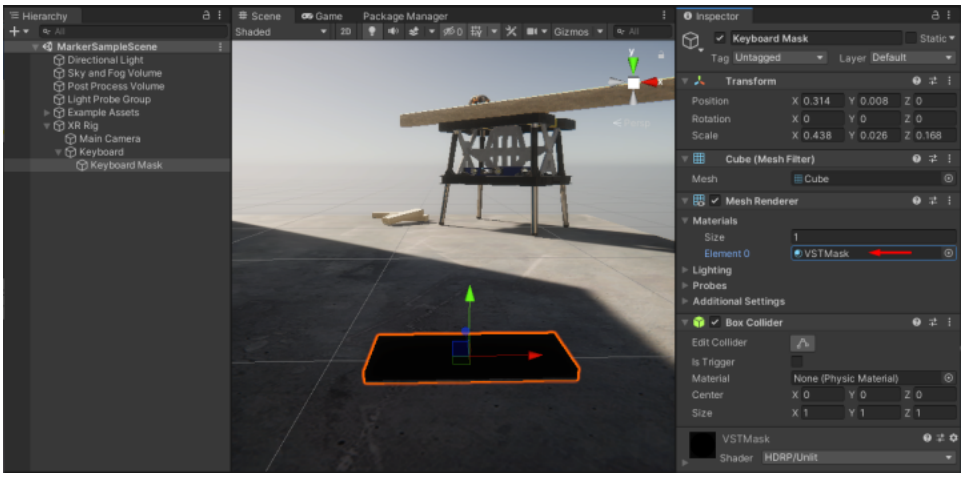


Figura 3.9: Oggetto Reale nell'Ambiente Virtuale

Figura 3.10: Primo Esempio di una scena in MR con un Oggetto Reale

3.3 MiDaS

La stima della profondità da immagini bidimensionali è stata oggetto di studio nell'ambito della visione artificiale per molto tempo ed è oggi utilizzata in settori come la robotica, la guida autonoma, la comprensione delle scene e la ricostruzione tridimensionale. Di solito, queste applicazioni utilizzano, per eseguire la stima della profondità, molteplici istanze della stessa scena, come coppie di immagini stereo [18] o multipli fotogrammi da una telecamera in movimento [16]. Poiché la stima della profondità da osservazioni multiple ha fatto notevoli progressi, naturalmente ha portato alla stima della profondità con un'immagine singola, poiché richiede meno costo e meno vincoli. Tuttavia, stimare con precisione la profondità da un'immagine singola non è semplice, in quanto infinite scene tridimensionali possono proiettarsi sulla stessa scena bidimensionale [14]. Per comprendere la configurazione geometrica da un'immagine singola, gli esseri umani considerano non solo indizi locali, come l'aspetto della texture in diverse condizioni di illuminazione e occlusione, la prospettiva o la scala relativa agli oggetti noti, ma anche il contesto globale, come la forma o la disposizione completa della scena [10]. All'interno dell'ambiente virtuale di Unity, un elemento cruciale è la percezione della profondità degli oggetti virtuali rispetto all'ambiente circostante.

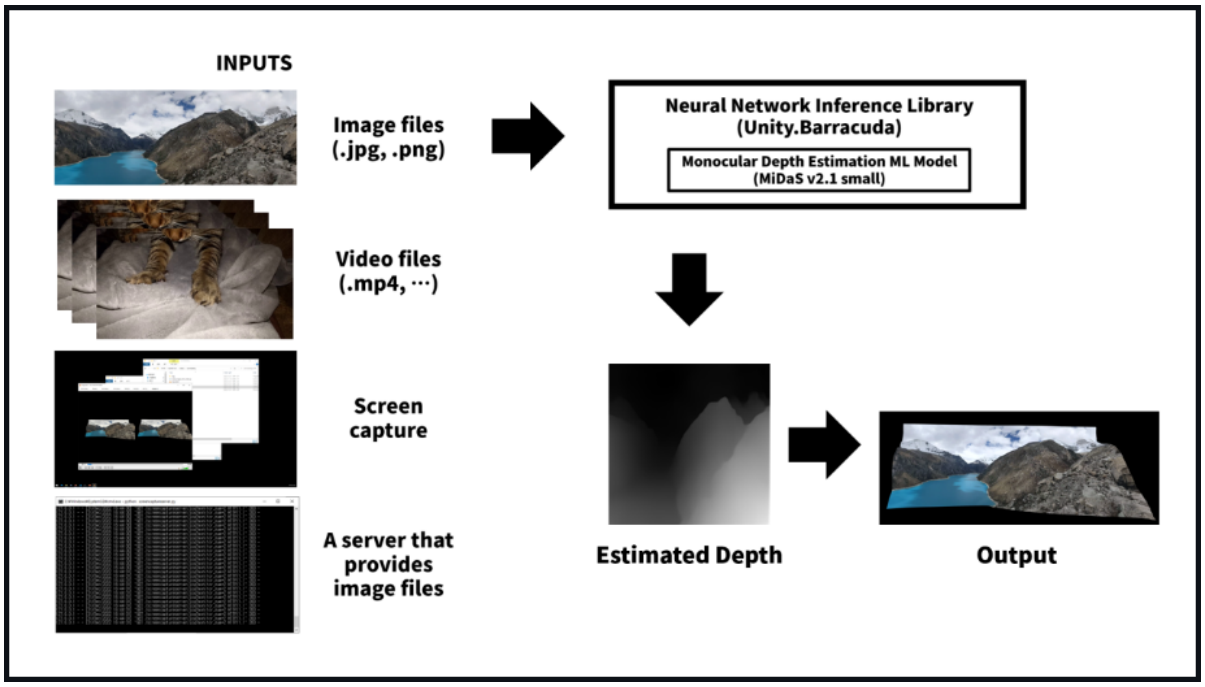


Figura 3.11: MiDaS per il calcolo della Depth Map

Nell'ambito dell'esperienza di MR, il modello MiDaS sarà integrato nella parte del Server, permettendo la generazione delle mappe di profondità per gli oggetti virtuali al-

l'interno dell'ambiente Unity. Questa implementazione consentirà agli oggetti virtuali di essere adeguatamente posizionat, allineati e tracciati con l'ambiente reale, contribuendo a migliorare l'immersione e la percezione dell'utente nell'ambiente di Mixed Reality. In definitiva, l'adozione del modello MiDaS per il calcolo delle mappe di profondità rappresenta un passo significativo nel garantire un'esperienza di MR accurata e coinvolgente, in cui gli oggetti virtuali sono resi in modo realistico all'interno dello spazio fisico e consentono una percezione coerente della profondità.

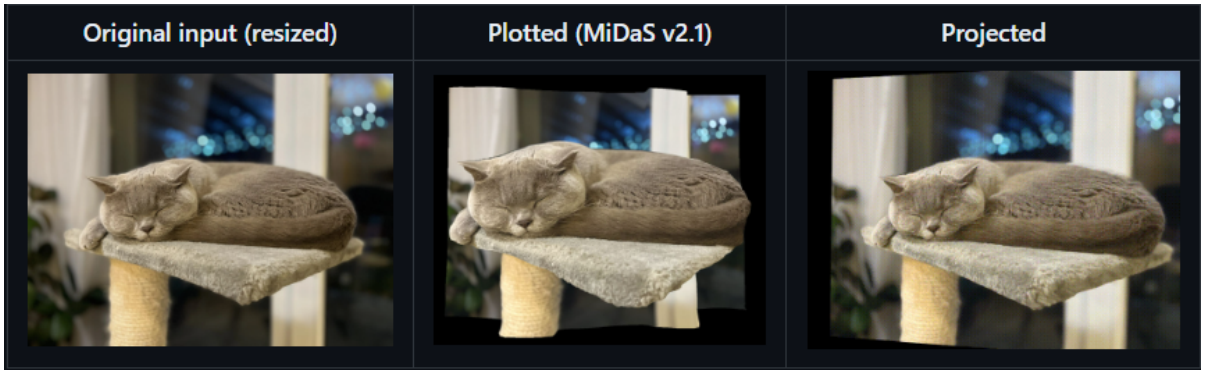


Figura 3.12: MiDaS esempio

3.4 Shader Unlit

Lo Shader *Unlit* all'interno della High Definition Render Pipeline di Unity è un componente essenziale per la creazione di scene virtuali realistiche e coinvolgenti. Questo shader offre una soluzione versatile per la resa di oggetti all'interno di una scena, eliminando gli effetti di illuminazione e ombreggiatura tipici dei materiali più complessi. Con lo shader Unlit, è possibile ottenere una resa piatta e priva di ombre, ideale per oggetti o superfici che non devono risentire delle dinamiche di illuminazione dell'ambiente circostante. Questo è particolarmente utile nelle applicazioni di Mixed-Reality (MR), in cui l'integrazione di oggetti virtuali con l'ambiente reale richiede una resa uniforme e priva di effetti di luce e ombra indesiderati. L'utilizzo dello shader Unlit è semplice e flessibile. Permette di definire il colore e la texture di un oggetto senza dover preoccuparsi della simulazione delle interazioni luminose. Questo è particolarmente utile quando si desidera mascherare la scena virtuale in modo da consentire la sovrapposizione dell'ambiente reale, come nel contesto del nostro progetto di Mixed-Reality con Varjo XR-3. In sintesi, lo shader Unlit della HDRP di Unity rappresenta uno strumento chiave per la creazione di esperienze Mixed-Reality realistiche e coerenti, consentendo agli sviluppatori di controllare in modo preciso l'aspetto visivo degli oggetti all'interno della scena virtuale.

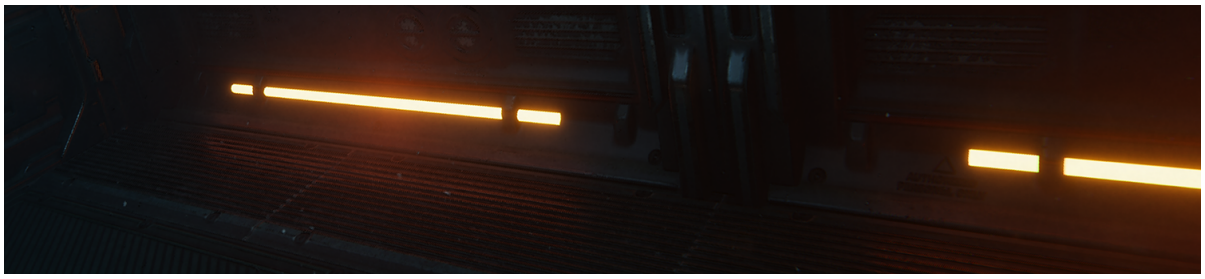


Figura 3.13: Esempio di Unlit Shader

Capitolo 4

Caso di Studio

Nel contesto in continua evoluzione della VR e della MR, sempre più applicazioni e piattaforme cercano di offrire esperienze tridimensionali coinvolgenti. Un caso di studio affascinante per questo caso è rappresentato da MFS come simulatore di volo, al fine di utilizzare il gioco, già realistico così com'è, per migliorare i processi di formazione e di allenamento di nuovi piloti, su quelli che possono essere simulatori di volo qualificati e certificati per questi processi. Un simulatore di volo è sfruttato per velocizzare le fasi dell'addestramento dei nuovi piloti, per esempio dando loro la possibilità di registrarsi in diverse situazioni durante le sessioni, offrendo la possibilità di migliorare le loro abilità e condividendo l'esperienza con altri utenti nel caso di simulazioni di volo di coppia. Le fasi di training quindi possono svolgersi in posti sicuri e garantire affidabilità per le esperienze vissute da parte degli utenti, assicurando l'utilizzo di meno risorse economiche allo stesso tempo.
Dunque la scelta ricade su MFS poiché include funzionalità VR, consentendo agli utenti di immergersi completamente nel mondo virtuale dell'aviazione. Tuttavia, MFS non presenta supporto per la giocabilità, con dispositivi compatibili, in realtà mista.
Il lavoro propone di andare oltre la VR e spingersi verso un'esperienza MR, in cui gli utenti possono interagire con il mondo virtuale in modi più profondi e naturali.
L'architettura di questa tesi è concepita per estendere l'esperienza di MFS da una realtà virtuale a una realtà mista, all'interno di un ambiente tridimensionale controllato, come Unity. Un'ulteriore aspetto chiave di questa estensione è la creazione di mappe di profondità che conferiscono al mondo virtuale una percezione tangibile di profondità, rendendo l'esperienza MR ancora più coinvolgente.
Nella seguente tesi, esploreremo in dettaglio l'architettura che rende possibile questa transizione da VR a MR, aprendo nuove possibilità per l'utilizzo di applicazioni inizialmente progettate per l'ambiente bidimensionale in contesti tridimensionali coinvolgenti. Analizzeremo le singole parti di cui quest'architettura è composta, a partire dal client in cui vive l'esecuzione di MFS, e il server all'interno dell' engine Unity.
All'interno dell'engine (o qualunque altro ambiente virtuale 3D controllato) sarà visualiz-

zato il risultato finale della trasmissione client-server tramite socket, per ottenere il gioco all'interno di un contesto tridimensionale controllato e implementare infine l'esperienza in MR con il contributo sul calcolo della profondità sulla scena.
Inoltre l'utilizzo dell'architeturra di MiDaS, contribuisce al calcolo ottenere la mappa di profondità per la scena virtuale.

4.1 Architettura del Simulatore in MR

Nel presente paragrafo, esploreremo l'architettura alla base dell'implementazione di un'esperienza immersiva di Realtà Mista, attraverso la fruizione di MFS con il dispositivo Varjo XR3. L'architettura del sistema è progettata per catturare il mondo virtuale di MFS attraverso il dispositivo Varjo XR3 e renderlo accessibile all'interno dell'ambiente di sviluppo Unity. L'architettura, illustarta in Figura 4.1 si compone di diversi componenti interconnessi, ognuno dei quali svolge un ruolo specifico nell'esperienza di MR. Il sistema quindi si basa su due componenti principali: il Client (PC con MFS) e il Server (PC con Unity). Il Client si occupa della cattura delle schermate di gioco e della loro invio al Server attraverso richieste HTTP. Il Server, a sua volta, elabora le schermate ricevute e le integra in un ambiente virtuale Unity, dove verrà sviluppata l'esperienza di volo in MR per l'utente. La struttura che descrive l'architettura di questo sistema:

- **Client con MFS:**
  Il Client è il sistema in cui gira il gioco di MFS. Quando il gioco è in esecuzione avviene la cattura della schermata per ottenere l'output visivo del gioco. Le schermate catturate vengono compressate per ridurre la dimensione dei dati e preparate per la trasmissione attraverso richieste HTTP;

- **Comunicazione tra Client e Server**
  La comunicazione tra il Client e il Server è gestita attraverso il protocollo HTTP. Quando il Client cattura una schermata, essa viene inviata al Server. L'URL di destinazione per queste richieste è configurato all'interno del Server. Ogni richiesta include i frame di MFS;

- **Il Server in Unity:**
  Il server è basato su Unity, un ambiente di sviluppo per creare esperienze virtuali e interattive. Il Server riceve le schermate catturate dal Client. Una volta ricevute, le schermate vengono elaborate per essere integrate nell'ambiente Unity. Questo processo coinvolge la decompressione delle schermate e il loro posizionamento all'interno dello spazio virtuale.

L'architettura delineata in questo capitolo rappresenta il fondamento su cui si basa l'implementazione di un'esperienza di realtà mista con il visore Varjo XR3(insieme al sistema di calibrazione di SteamVR) e MFS. La comunicazione tra il Client e il Server,

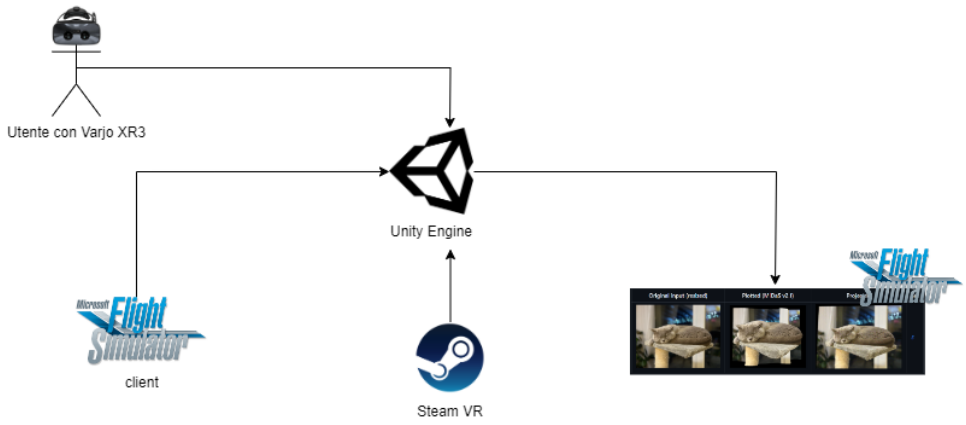


Figura 4.1: Architettura del Sistema di Simulazione

insieme alla conversione delle schermate in contenuti virtuali all'interno di Unity, consentirà agli utenti di immergersi in un ambiente coinvolgente in cui il mondo virtuale e il mondo reale si fondono in un'esperienza unica e realistica. Inoltre, L'integrazione della Depth Map all'interno dell'esperienza, arricchisce notevolmente la sensazione di immersione dell'utente, come vedremo in seguito, creando un'esperienza visiva più coinvolgente e realistica.

4.2 Architettura di MiDaS

Il modello MiDaS sfrutta una struttura di rete neurale convoluzionale profonda che consente di catturare informazioni a diverse scale, ottenendo una stima precisa della profondità sia per oggetti vicini che lontani [12], come illustrato in Figura 4.2. Inoltre, la rete è progettata per affrontare sfide come la ricerca di profondità e l'occlusione, contribuendo a migliorare l'accuratezza complessiva del calcolo della mappa di profondità finale. La panoramica generica dell'architettura di MiDaS:

- Architettura Encoder-Decoder:
  MiDaS si basa su un'architettura di tipo encoder-decoder, in cui la parte dell'encoder è responsabile dell'estrazione di caratteristiche di alto livello e il decoder genera la mappa di profondità da queste caratteristiche attraverso il processo di up-sampling (aumento della risoluzione);

- Architettura di Base:
  MiDaS utilizza tipicamente una rete residuale (ResNet-50 o ResNet-101) per l'estrazione delle caratteristiche, poiché è robusta ai gradienti che tendono a scomparire durante l'addestramento. Questo consente a MiDaS di estrarre mappe di caratteristiche multi-canale dalle immagini in ingresso, catturando informazioni gerarchiche a diverse scale;

- Fusione di caratteristiche Multi-scala:
  MiDaS incorpora connessioni di salto (skip connections) e fusione di caratteristiche per consentire una stima accurata della profondità. Le mappe di caratteristiche da livelli precedenti sono collegate ai livelli successivi tramite connessioni di salto per accedere ai dettagli a basso livello durante l'up-sampling. Con la fusione delle caratteristiche, le mappe di caratteristiche multi-scala vengono combinate per garantire un'efficace sfruttamento sia delle informazioni locali che globali per la stima della profondità;

- Up-Sampling e Affinamento:
  La mappa di profondità finale viene generata utilizzando l'up-sampling. Le tecniche generali utilizzate per l'up-sampling sono l'interpolazione bilineare o le convoluzioni trasposte per aumentare la risoluzione spaziale delle mappe di caratteristiche. La fusione delle caratteristiche viene impiegata per combinare le mappe di profondità con le relative connessioni di salto al fine di raffinare la stima della profondità [13].

Figura 4.2: Calcolo di una Depth Map utilizzando MiDaS

Capitolo 5

Implementazione

Nel corso dei capitoli precedenti, abbiamo esaminato con attenzione la progettazione e l'architettura del nostro sistema di simulazione che si basa su MFS. Abbiamo esplorato la sue parti in Client con MFS in esecuzione, la comunicazione HTTP tra client e Server in unity con il calcolo della *Depth Map*. L'obiettivo principale di questo progetto è offrire agli utenti un'esperienza coinvolgente in MR utilizzando MFS come mondo virtuale per il simulatore di volo. Tuttavia, sono stati affrontati diversi problemi per rendere l'applicazione direttamente compatibile con la tecnologia MR, dato che su MFS non c'è possibilità di avere il controllo diretto dell'ambiente di gioco e supporta soltanto esperienze in VR. La soluzione adottata consiste nel "proiettare" MFS in VR all'interno di un ambiente virtuale, come Unity, per ottenere il controllo completo sulla scena e consentire la creazione di un'esperienza di volo in MR altamente personalizzata. Il sistema è basato su una comunicazione tramite socket, che permette la trasmissione in tempo reale dei frame del gioco da un client (dove MFS è in esecuzione) a un server all'interno dell'ambiente virtuale in Unity. È stata posta particolare attenzione all'ottimizzazione della trasmissione dei dati, al fine di ridurre al minimo la latenza e garantendo un'esperienza fluida e reattiva. Per risolvere il problema della percezione "piatta" dello streaming della scena di gioco all'interno di Unity, è stata adottata una soluzione basata su Machine Learning. Un modello di Machine Learning denominato MiDaS è stato utilizzato per calcolare una mappa di profondità (Depth Map) della scena virtuale. Questa mappa di profondità viene applicata alla mesh sulla quale è proiettato lo streaming di MFS, che si deforma in corrispondenza dei valori specifici della Depth Map dello specifico frame calcolato da MiDaS, creando per l'utente finale un senso realistico di profondità. Questo approccio migliora notevolmente l'esperienza MR dell'utente, consentendogli di percepire l'ambiente virtuale in modo tridimensionale e immersivo. In questo capitolo esploreremo il codice che alimenta il nostro progetto, analizzando in particolare le funzioni fondamentali che consentono la registrazione dello schermo, la compressione e la conversione dei frame e la gestione delle richieste HTTP per la loro tasmissione. Esamineremo il codice a cominciare dalla funzione `main()`, il punto di ingresso del nostro software, che

coordina il flusso delle operazioni. Scopriremo come essa inizializza il registratore video per catturare la finestra desktop di MFS, e anche come avvia il server HTTP, preparando così il terreno per queste operazioni di registrazione e trasmissione dei frame. Alla fine di questo capitolo, avremo completato una tappa fondamentale nell'implementazione del nostro progetto, avendo trasformato i frame catturati in un formato ottimizzato per la trasmissione e pronti per essere inviati al nostro server tramite il protocollo HTTP. Quindi esamineremo il processo di streaming per MFS.

5.1 Funzione principale

La funzione principale `main()` riportata sotto, rappresenta il punto di ingresso dell'applicazione e svolge un ruolo fondamentale nel coordinare l'esecuzione del nostro sistema di registrazione e streaming per MFS. La funzione inizia dichiarando le variabili necessarie per gestire il contesto del client WebSocket e del server HTTP. Questi contesti saranno fondamentali per la comunicazione e la gestione delle richieste dai client. Per prima cosa, il `main` chiama la funzione `initRecorder()`, che è responsabile dell'inizializzazione del registratore video. Tra le sue operazioni, essa cerca la finestra di MFS ed esegue le istruzioni necessarie per la cattura dei frame dallo schermo. Dopo aver inizializzato il registratore video, passiamo alla funzione `initServer()`, che configura e avvia il server HTTP. Il server sarà responsabile della gestione delle richieste dei client, in questo caso della richiesta di streaming dei frame dello schermo. Con la stessa chiamata quindi il server HTTP entra in esecuzione e pronto a ricevere richieste dai client. Il nostro sistema è ora configurato e pronto per catturare e trasmettere i dati dei frame di gioco.

```
1 int main() {
2     struct lws_context* context;
3     struct lws_context_creation_info info;
4     struct lws_client_connect_info cinfo;
5     struct lws* wsi;
6     initRecorder();
7     initServer();
8 }
```

Codice 5.1: Funzione Principale

5.2 Inizializzazione del Registratore Video

La funzione `initRecorder()` si occupa di inizializzare il registratore video, configurando l'ambiente necessario per catturare i frame dal gioco. La funzione `initRecorder()` in Figura **??** inizia dichiarando variabili per la finestra `(hWnd)` e i contesti dei dispositivi DC `(hWindowDC e hMemDC)`. Un "contesto del dispositivo" o "Device Context" (DC) è un oggetto che rappresenta un ambiente grafico per gestire operazioni di disegno e grafica su un dispositivo di output (schermo, stampanti, ecc.):

- `hWindowDC`: recupera un handle per un contesto di dispositivo (DC) per l'area client di una finestra specificata o per l'intero schermo. Consente di effettuare operazioni grafiche come la cattura dei frame di una finestra;
- `hMemDC`: Rappresenta crea un contesto di dispositivo di memoria (DC) compatibile con il dispositivo specificato, ad esempio `hWindowDC`.
  `hMemDC` sarà utilizzato per eseguire operazioni di cattura dell'immagine senza influire direttamente sullo schermo;

```
1  void initRecorder() {
2      hWnd = NULL;
3      EnumWindows(EnumWindowsProc, (LPARAM)&hWnd);
4      SwitchToThisWindow(hWnd, TRUE);
5      displayWidthOffset = 5;
6      displayHeightOffset = 45;
7      displayWidth = 1895 - displayWidthOffset;
8      displayHeight = 1050 - displayHeightOffset;
9      hBitmap = CreateCompatibleBitmap(hWindowDC,
10     displayWidth,
11     displayHeight);
12 }
```

Codice 5.2: Inizializzazione del Recorder

La finestra è essenziale per catturare i frame dal gioco, e `hWnd` sarà utilizzata per memorizzare l'handle della finestra. Successivamente, la funzione utilizza la funzione `EnumWindows` per enumerare tutte le finestre attive nel sistema, utilizzando la callback `EnumWindowsProc`. L'obiettivo è trovare la finestra di MFS, tra quelle disponibili in esecuzione, e ottenere il suo handle. Una volta ottenuto l'handle della finestra di gioco, la funzione chiama `SwitchToThisWindow` per assicurarsi che la finestra sia effettivamente quella attiva e che sia quella di MFS al momento della cattura dei frame. Successivamente, vengono ottenuti i contesti dei dispositivi DC. `hWindowDC` rappresenta

il contesto del dispositivo per l'intero schermo, mentre `hMemDC` è creato come un contesto del dispositivo compatibile, e funge da buffer temporaneo in cui i dati vengono elaborati prima di essere catturati e utilizzati per ulteriori scopi, come la conversione dell'immagine o la trasmissione ai client. Sono definiti offset e dimensioni che determinano l'area da catturare dallo schermo del gioco. Questi valori possono essere regolati in base alle esigenze specifiche del progetto. Con questa inizializzazione completa, il registratore video è pronto a catturare i frame dal gioco MFS. Nelle sezioni successive, esamineremo il processo di cattura dei frame e la loro preparazione per la trasmissione attraverso il server HTTP.

5.3 Inizializzazione Server

La funzione `initServer()` è responsabile dell'inizializzazione del server HTTP, che gestirà le richieste provenienti dai client per la trasmissione dei frame di gioco. La funzione `initServer()` inizia dichiarando variabili per il server HTTP e un indirizzo IP. La struttura `addr` viene inizializzata con le informazioni sull'indirizzo IP e sulla porta a cui il server HTTP sarà associato. Nell'esempio riportato sotto, l'indirizzo IP è impostato su "130.136.148.18" e la porta su 5000. Questi valori possono essere personalizzati in base alle esigenze specifiche del progetto. La funzione chiama '`MHD_start_daemon()`' per creare il server HTTP. Dopo la creazione del server HTTP, la funzione verifica se il server è stato avviato correttamente. Se il server HTTP non può essere avviato, viene stampato un messaggio di errore, altrimenti, se il server HTTP viene avviato con successo, la funzione stampa un messaggio di conferma che indica l'indirizzo IP e la porta su cui il server è in esecuzione. Successivamente, utilizziamo '`getchar()`' per mantenere il server in esecuzione fino a quando l'utente non specifica l'input per terminare il programma. Infine, quando il server viene terminato, la funzione chiama '`MHD_stop_daemon()`' per arrestare il server HTTP in modo pulito. Con l'esecuzione di '`initServer()`', il nostro server HTTP è ora in esecuzione e pronto a ricevere richieste dai client per la ricezione dei frame di MFS. Nelle sezioni successive, esamineremo la gestione delle richieste HTTP e la trasmissione dei dati dei frame ai client.

```
void initServer() {
    struct MHD_Daemon* http_server;
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(struct sockaddr_in));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(5000);
    inet_pton(AF_INET, "130.136.148.18", &(addr.sin_addr));
    http_server = MHD_start_daemon(
        MHD_USE_SELECT_INTERNALLY,
        5000,
        NULL, NULL, &request_handler, NULL,
        MHD_OPTION_SOCK_ADDR, &addr,
        MHD_OPTION_END);
    if (!http_server) {
        printf("Failed to start HTTP server.\n");
        return;
    }
    printf(``HTTP server is running on %s:%d...\n'',
    ``130.136.148.18'', 5000);
    getchar();
    MHD_stop_daemon(http_server);
}
```

Codice 5.3: Inizializzazione del Server

5.4 Gestione delle Richieste HTTP

La funzione `request_handler` serve a gestire le richieste dei client. Questa funzione è responsabile di inviare i frame di gioco ai client. La funzione, raffigurati in Figura **??**, viene chiamata ogni volta che il server riceve una richiesta HTTP. Il cuore del processo sta nel confrontare l'URL della richiesta HTTP con `/get_frame`. Quando un client richiede un frame di gioco, se l'URL corrisponde, il server risponde inviando il frame desiderato. La risposta è creata utilizzando la funzione `acquireViewAndSendIt`, che si occupa di catturare il frame dalla finestra di gioco e inviarlo al client. Per fornire una risposta valida, la funzione `request_handler`, crea un oggetto `struct MHD_Response` che contiene i dati dei frame da inviare e imposta l'heandle "Content-Type" come "application/octet-stream", per indicare che i dati sono in formato binario. Una volta che la risposta è stata preparata, viene accodata e inviata al client, soddisfacendo la richiesta di streaming del frame di gioco. In questo modo, `request_handler` gioca un ruolo centrale nell'interazione tra il server e i client, consentendo loro di ottenere in modo efficiente i frame di gioco richiesti per l'esperienza finale in MR. Con una gestione adeguata delle richieste HTTP, il nostro sistema offre una risposta tempestiva e fluida alle esigenze dei client, evitando framerate bassi e limitando la latenza nel flusso dei dati.

```
1  static int request_handler(void* cls, struct MHD_Connection*
2  connection, const char* url, const char* method,
3  const char* version, const char* upload_data, size_t*
       upload_data_size,
4  void** ptr)
5   {
6      if (strcmp(url, ``/get_frame'') == 0) {
7          const char* response = ``Hello, World! This is
               /get_frame.'';
8          struct BitsAndSize bitsAndSize =
               acquireViewAndSendIt();
9          BYTE* bits = bitsAndSize.bits;
10         struct MHD_Response* http_response =
11         MHD_create_response_from_data(bitsAndSize.size,
               bitsAndSize.bits,
12         MHD_YES, MHD_NO);
13         MHD_add_response_header(http_response, "Content-Type",
14         "application/octet-stream");
15         int ret = MHD_queue_response(connection,
16         MHD_HTTP_OK, http_response);
```

```
17          MHD_destroy_response(http_response);
18          return ret;
19  }
```

Codice 5.4: request_handler

5.5 Conversione dei Frame Catturati

Nel nostro sistema di registrazione e streaming per MFS, la conversione dei frame catturati è una fase essenziale prima della trasmissione ai client. Abbiamo sviluppato due funzioni chiave, `acquireViewAndSendIt()`, in Figura **??** e `SaveBitmapAsJPEGToBuffer()`, in figura **??** che collaborano per catturare e preparare i frame per la trasmissione. La funzione `acquireViewAndSendIt()` è responsabile della cattura dei frame di MFS dalla sua finestra. Utilizzando operazioni grafiche, questa funzione acquisisce il contenuto visibile nella finestra di gioco e lo prepara per la successiva elaborazione. Dopo la cattura, i dati del frame vengono immagazzinati nella struttura dati `BitsAndSize`. Per la trasmissione, la funzione `SaveBitmapAsJPEGToBuffer()`) entra in esecuzione. Questa funzione converte il frame acquisito in formato JPEG per ridurre le dimensioni dei dati e la larghezza di banda necessaria per la trasmissione. Il processo di conversione coinvolge diverse fasi, tra cui la creazione di un oggetto `BITMAPINFO` per descrivere l'immagine, l'allocazione di memoria per i dati del frame, e l'utilizzo della libreria `libjpeg` per la compressione dei dati in formato JPEG. La dimensione complessiva dei dati compressi viene calcolata, e i dati vengono copiati nell'array `result.bits` all'interno della struttura `BitsAndSize`. Questi dati convertiti e compressi saranno quindi pronti per essere trasmessi ai client tramite il server HTTP. Questa combinazione di funzioni consente la cattura, la conversione e la preparazione dei frame catturati per la trasmissione. Questo processo ottimizzato garantisce che i dati dei frame vengano inviati ai client in modo efficiente, riducendo al minimo la latenza e migliorando l'esperienza di streaming complessiva.

```
1   struct BitsAndSize acquireViewAndSendIt() {
2       HGDIOBJ hOldBitmap = SelectObject(hMemDC, hBitmap);
3       BitBlt(hMemDC, 0, 0, displayWidth, displayHeight,
4       hWindowDC, 0 + displayWidthOffset,
5       0 + displayHeightOffset, SRCCOPY);
6
7       SelectObject(hMemDC, hOldBitmap);
8
9       BITMAPINFO bitmapInfo;
10      bitmapInfo.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
```

```
    bitmapInfo.bmiHeader.biWidth = displayWidth;
    bitmapInfo.bmiHeader.biHeight = -displayHeight;
    bitmapInfo.bmiHeader.biPlanes = 1;
    bitmapInfo.bmiHeader.biBitCount = 32;
    bitmapInfo.bmiHeader.biCompression = BI_RGB;
    bitmapInfo.bmiHeader.biSizeImage = 0;
    bitmapInfo.bmiHeader.biXPelsPerMeter = 0;
    bitmapInfo.bmiHeader.biYPelsPerMeter = 0;
    bitmapInfo.bmiHeader.biClrUsed = 0;
    bitmapInfo.bmiHeader.biClrImportant = 0;
    int bitmapSize = displayWidth * displayHeight * 4;
    BYTE* bits = (BYTE*)malloc(bitmapSize);
    if (!bits) {
        printf("Failed to allocate memory for bitmap bits\n");
        return;
    }
    if (!GetDIBits(hWindowDC, hBitmap,
    0, displayHeight, bits, &bitmapInfo, DIB_RGB_COLORS)) {

        printf("Failed to get bitmap bits\n");
        free(bits);
        return;

    }
    struct BitsAndSize bitsAndSize;
    bitsAndSize = SaveBitmapAsJPEGToBuffer(hBitmap,
    displayWidth, displayHeight);
    BYTE* bitToSend = bitsAndSize.bits;
    int size = bitsAndSize.size;
    return bitsAndSize;
}
```

Codice 5.5: acquireViewandSendIt

5.6 Visualizzazione Profonda in Unity

Nel nostro progetto, non ci limitiamo a trasmettere semplicemente i frame di gioco all'interno di una scena virtuale di Unity, per generare un'esperienza in MR all'interno di MFS. Va oltre, e mostra la creazione di un'esperienza di MR ancora più coinvolgente e realistica attraverso l'uso di MiDaS, un modello di Machine Learning per il calcolo della profondità. MiDaS, di cui un esempio in Figura 5.1, è il cuore della nostra soluzione per il calcolo della Depth Map. Questo modello di Machine Learning avanzato è progettato per estrarre informazioni di profondità da immagini bidimensionali, ricostruendo la percezione tridimensionale della scena. Questo modello offre mappe di profondità (dette Depth Map) a partire da immagini monoculari, poi calibrate a vista singola con due architetture di deep learning, reti neurali convoluzionali e transformers. In sostanza MiDaS può generare la mappa di profondità e il file poligonale corrispondente a diverse risoluzioni dell'immagine4 [11].

Figura 5.1: MiDaS esempio

All'interno della scena di Unity, utilizziamo MiDaS per calcolarne la Depth Map, che rappresenta la distanza tra gli oggetti nella scena e l'osservatore. Questo processo è fondamentale per creare un senso di "profondità" nella scena del gioco, migliorando l'esperienza di MR. La Depth Map ottenuta da MiDaS viene quindi applicata alla mesh sulla quale verrà proiettato lo stream di MFS, consentendo un'interazione più accurata tra gli oggetti virtuali e il mondo reale. Grazie a MiDaS, i giocatori possono godere di un'esperienza di MR più coinvolgente, in cui gli oggetti virtuali si integrano in modo più realistico con l'ambiente circostante, creando una percezione tridimensionale autentica e un'immersione completa. Nella nostra configurazione, la scena virtuale di Unity serve come ambiente dove l'utente vivrà l'esperienza in MR, interagendo con MFS. Questa scena virtuale agisce come una sorta di "ponte" tra il mondo del gioco e l'utente, consentendo una fusione fluida tra la realtà virtuale e il mondo reale. In questa scena virtuale, MFS viene "proiettato" su una mesh tridimensionale. Questa mesh rappresenta la superficie virtuale sui cui si svolge lo streaming del gioco. L'importante di questo concetto è che

la mesh possa deformarsi e adattarsi dinamicamente in base a ciò che accade all'interno di MFS.

Figura 5.2: Impostazioni di MiDaS

Ad esempio, se l'aereo nel gioco si inclina o si muove, la mesh si adeguerà, fornendo il giusto senso di profondità e di movimento all'utente che indossa il visore MR. Per abilitare lo streaming dei frame all'interno di questa scena virtuale, viene specificato un indirizzo IP, come in Figura 5.2. Questo indirizzo IP è fondamentale perché consente a Unity di ricevere e visualizzare i frame provenienti dal server di streaming, che cattura l'output di MFS. Ciò significa che l'utente, indossando il visore MR, viene letteralmente "immerso" all'interno della scena virtuale, dove può percepire la profondità dell'ambiente del gioco in modo realistico. Sono presenti anche ulteriori slider di controllo, al fine di gestire il risultato finale che viene calcolato dal modello, al fine di ottimizzare il senso di profondità percepito sulla scena finale. In questo modo, l'utente può sperimentare diverse combinazioni, impostando diversi valori per questi parametri, andando alla ricerca di una fusione perfetta tra il mondo virtuale di MFS e la profondità percepita della scena, grazie alla deformazione dinamica della mesh e alla trasmissione accurata dei frame, minimizzando la latenza dello streaming. Questo approccio offre un'esperienza di MR coinvolgente e autentica, in cui la profondità dell'ambiente virtuale è percepita in modo naturale e dinamico, consentendo all'utente finale di vivere un'esperienza di volo realistica. In Figura 5.2 si può notare la presenza di un ulteriore oggetto, colorato completamente nero. Quest'ultimo rappresenta un piccolo game object presente in scena, con applicato lo shader "Unlit" della High Definition Rendering Pipeline (HDRP) di Unity, per effettuare quello che viene chiamato "mascheramento" della scena virtuale,

al fine di combinare gli elementi virtuali con gli elementi reali, e fornire così agli utenti l'esperienza in MR.

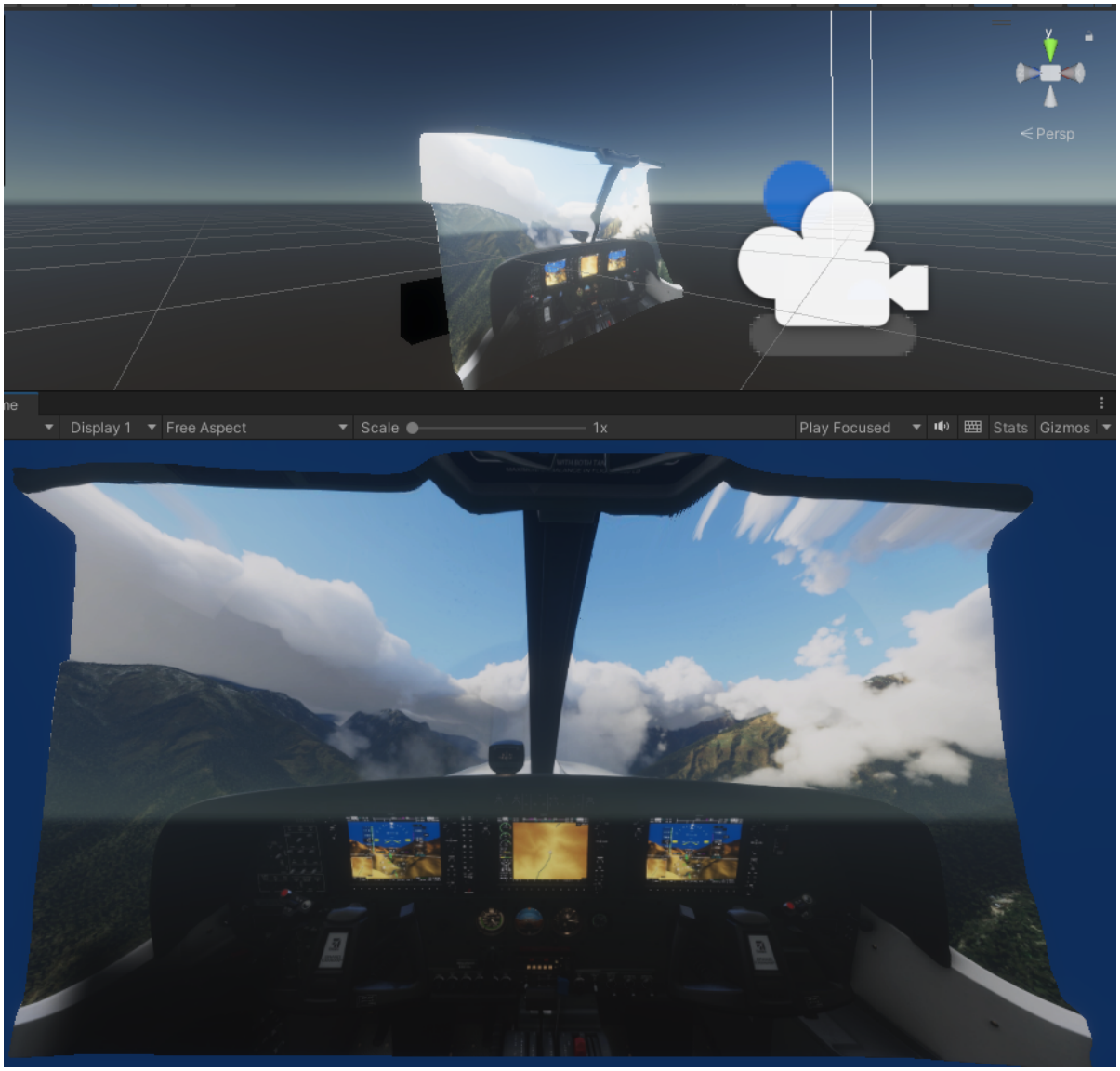


Figura 5.3: Streaming MFS su MiDaS

5.7 Mixed-Reality

Nel contesto di questo progetto, il MR prende vita all'interno di una scena virtuale in Unity, dove MFS viene proiettato e integrato con una serie di tecnologie per offrire un'esperienza coinvolgente agli utenti. In questo progetto, un aspetto chiave dell'esperienza in MR, come già detto, è basata sull'uso di un modello di machine learning, noto come MiDaS, per calcolare e modellare la forma della superficie della mesh. Questa mesh rappresenta la superficie tridimensionale su cui si svolge lo streaming del gioco e viene opportunamente deformata per garantire una percezione realistica della profondità nella scena. Questo processo di deformazione della mesh è fondamentale per rendere l'esperienza di volo in MR il più autentica possibile, consentendo agli utenti di percepire il movimento dell'aereo e l'ambiente circostante in modo dinamico. Per visualizzare questa scena virtuale in MR, è stato configurato un oggetto "XRRig Camera", progettato per essere utilizzato con il visore MR Varjo XR-3. Questa configurazione consente agli utenti di indossare il visore e immergersi completamente nella scena virtuale creata in Unity, che include lo streaming di MFS sulla mesh enerata da MiDaS. Grazie a questa integrazione, l'utente può percepire il senso della profondità in modo dinamico, in base alle manovre effettuate durante il gioco e a ciò che sta guardando o inquadrando all'interno dell'ambiente di volo.
In Figura 5.4 si può vedere un primo esempio di un primo prototipo del sistema presentato. MFS viene reso in MR all'interno di Unity utilizzando l'SDK di Varjo, l'HDRP e il dispositivo RealSimGear G1000 (RSG) come secondo schermo. In pratica, grazie all'installazione dell'opportuno plugin (XPlane12) [15] [17] che funziona e lavora su MFS, ci viene permesso di trascinare alcune parti in-game come finestre esterne all'interno sullo schermo del RSG. Si procede con il mascheramento della scena virtuale, che visualizza la proiezione di MFS, visualizzando allo stesso tempo anche la componente della scena reale, con il RSG inquadrato e tracciato tramite opportuno marcatore (parzialmente visibile in Figura 5.4). Per consentire l'effetto MR all'interno della scena, è stato utilizzato uno shader specifico denominato "Unlit" della HDRP di Unity, per creare il materiale come in Figura 3.3. Questo shader ha la particolarità di mascherare la scena virtuale, consentendo la sovrapposizione dell'ambiente reale durante l'esecuzione, tramite il video pass through. L'oggetto che utilizza questo shader è stato posizionato in modo strategico all'interno della scena virtuale, cercando di allineare la sua posizione relativa con quella del RSG reale, tramite il tracciamento del marcatore, perché risulti correttamente sovrapposto ai display nella scena del gioco, per consentire agli utenti di vedere il contesto reale circostante attraverso il visore che supporti il MR. Inoltre, è importante notare che lo shader "Unlit" di HDRP, non fornisce e non contribuisce al calcolo di alcun contributo di illuminazione presente nella scena virtuale. I nuovi materiali in HDRP utilizzano lo shader "Lit" per impostazione predefinita. Per creare un materiale non illuminato, è necessario creare un nuovo materiale e fargli utilizzare lo shader corrispondente "Unlit", come già fatto precedentemente in *Masking con Varjo*. La realizzazione di questo

prototipo dunque segue gli stessi passaggi precedentemente documentati. Tuttavia, per ottenere risultati ancora più realistici, è importante considerare le limitazioni da superare di questo sistema e gli ulteriori passaggi come la modellazione accurata della mesh che funge da "maschera" all'interno della scena virtuale, adattandola alla forma della piattaforma di simulazione (cockpit), con l'obbiettivo di farla aderire con la più alta precisione e garantendone un allineamento il più possibile realistico. Questo migliorerebbe notevolmente il tracciamento degli oggetti e l'esperienza di volo in MR, fornendo agli utenti una sensazione di immersività ancora maggiore durante il gioco. In questo primo caso l'esperienza in MR, per mancato accesso alle risorse, è stata implementata consentendo all'utente di interagire con il dispositivo RSG come componente reale da sovrapporre alla scena virtuale del simulatore, insieme al calcolo della profondità.

Figura 5.4: Esecuzione di MFS con un approccio in MR

5.8 Limitazioni

Nonostante i significativi progressi raggiunti nell'implementazione dell'approccio MR per ottenere mappe di profondità all'interno della scena virtuale di Unity attraverso lo streaming di dati frame tra client-server, è importante riconoscere alcune limitazioni chiave associate a questa metodologia.
In primo luogo, è essenziale sottolineare che la configurazione attuale non permette un controllo completo del gioco per un implementazione diretta di funzionalità MR, neanche a livello di comandi.
La nostra implementazione si concentra principalmente sulla visualizzazione degli elementi di gioco in uno spazio virtuale tridimensionale progettato per l'integrazione di funzionalità di realtà mista. Tuttavia, l'interazione diretta dell'utente con il mondo virtuale è attualmente limitata. Infatti, tramite l'utilizzo di opportuno visore, l'utente può visualizzare la scena di MFS proiettata nell'applicativo, percependone anche la profondità, ma allo stesso tempo, non ha possibilità di controllare direttamente il simulatore.
Gli utenti potrebbe interagire con il gioco utilizzando dispositivi di input come controller, mouse e tastiera per comandare la sessione di gioco. Tuttavia, l'esperienza interattiva potrebbe risultare meno intuitiva e immersiva rispetto a soluzioni più avanzate di interazione uomo-macchina.
Un'altra considerazione rilevante riguarda il processo di visualizzazione MR.
La visualizzazione degli elementi di gioco in realtà mista richiede un calcolo accurato della profondità della scena di gioco visualizzata in un dato momento. Tuttavia, poiché facciamo affidamento su un modello pre-addestrato di MiDaS per questo calcolo, l'accuratezza delle mappe di profondità può influenzare direttamente la qualità complessiva della visualizzazione. In altre parole, le variazioni nella precisione del modello possono riflettersi direttamente sulla percezione dell'utente in merito alla profondità e alla tridimensionalità degli oggetti virtuali nel contesto MR, non risultando abbastanza realistici quanto si vorrebbe.
In conclusione, nonostante le attuali limitazioni e restrizioni associate all'approccio MR che stiamo utilizzando, si studia un modo che ci permetta di superarle. Guardando al futuro, il nostro obiettivo è espandere l'applicazione di questa tecnologia a una vasta gamma di applicazioni desktop, consentendo agli utenti di interagire in modo più fluido, intuitivo e immersivo con il mondo virtuale, tramite i vantaggi offerti dalla realtà mista.

Capitolo 6

Conclusioni e Lavori Futuri

In questa tesi, abbiamo presentato il contesto e l'obiettivo principale del progetto realizzato. L'obiettivo principale è stato delineato come la creazione di un'esperienza in MR coinvolgente, utilizzando MFS come soggetto protagonista di un simulatore di volo. Sono stati identificati i principali problemi da affrontare, tra cui la mancanza di controllo diretto dell'ambiente di gioco di MFS, e per ovviare a questa mancanza di controllo dello spazio tridimensionale, è stato sviluppato un sistema di trasmissione dati dei frame di MFS all'interno di un ambiente virtuale come Unity, per procedere con la realizzazione del simulatore in MR. Quindi, un altro problema da affrontare, è stato quello di trovare una soluzione per ridurre la latenza della trasmissione dati frame di MFS. Siamo entrati nel dettaglio delle sfide tecniche e abbiamo delineato la soluzione generale del progetto, esaminando l'architettura di sistema adottata. Abbiamo descritto come la comunicazione tramite socket che consente la trasmissione in tempo reale dei frame di gioco, quindi da MFS a Unity. Abbiamo enfatizzato l'importanza dell'ottimizzazione della trasmissione dei dati per ridurre al minimo la latenza e garantire un'esperienza fluida per l'utente. E' stato esaminata l'implementazione di due componenti chiave del progetto: il recorder e il server. Il recorder si occupa di catturare i frame di MFS, mentre il server gestisce la trasmissione dei dati al server in Unity. Abbiamo esaminato il codice di queste funzioni e descritto il loro ruolo nel contesto generale del progetto. Ci siamo concentrati sulla gestione delle richieste HTTP all'interno del server e sulla conversione dei frame catturati da MFS, esaminando il codice della funzione che gestiscono le richieste HTTP e invia i frame al client e il processo della loro conversione in formato JPEG, per ridurre la quantità di dati trasmessi. L'aspetto chiave del progetto rimane sempre lo stesso, ovvero garantire l'esperienza in MR. Abbiamo descritto come un modello di machine learning MiDaS, venga utilizzato per generare la forma della superficie della mesh all'interno della scena virtuale in Unity. Questa mesh viene deformata per garantire una percezione realistica della profondità nella scena. Abbiamo anche discusso dell'uso di uno shader chiamato "Unlit" della High Definition Render Pipeline (HDRP) di Unity per mascherare parte della scena virtuale e consentire la sovrapposizione del-

l'ambiente reale attraverso il visore MR Varjo XR-3, grazie anche alla tecnologia di video pass-through che quest'ultimo fornisce. Abbiamo affrontato le sfide di comunicazione in tempo reale, cattura dei frame e gestione della profondità, ottenendo un risultato che offre un senso realistico di profondità all'utente durante il volo. La combinazione dell'utilizzo di MiDaS e dello shader "Unli" ha reso l'esperienza MR più immersiva. Per i futuri progetti correlati, ci sono diverse direzioni possibili. Prima di tutto, è possibile migliorare ulteriormente la modellazione della mesh "maschera" all'interno della scena virtuale per una migliore corrispondenza e allineamento con il cockpit del simulatore. Inoltre, potrebbe essere esplorata l'integrazione di ulteriori sensori o marcatori per il tracciamento e il riconoscimento più preciso e performante degli oggetti nella scena MR. Infine, potrebbe essere interessante esplorare la possibilità di estendere questa tecnologia a simulatori di volo diversi da MFS o a altri tipi di giochi e applicazioni MR.
In conclusione, il progetto ha dimostrato il potenziale dell'implementazione di MR in ambienti di simulazione complessi come MFS, aprendo la strada a future innovazioni e sviluppi nell'ambito dell'esperienza di volo immersiva in Mixed-Reality.

Capitolo 7

Ringraziamenti

In questo momento di gioia e realizzazione personale, desidero esprimere la mia profonda gratitudine a coloro che hanno reso possibile il mio percorso di studio universitario. La mia famiglia merita un riconoscimento speciale per il loro costante sostegno, affetto e incoraggiamento, insieme anche ai miei coinquilini e ai fratelli di comunità, in modo particolare a Benedetto, Elena e Margherita per essermi stati vicini e per avermi supportato durante questo lungo e difficile percorso accademico.
Vorrei estendere il mio sincero ringraziamento al Prof. Gustavo Marfia per la sua disponibilità e l'inestimabile assistenza fornita fin dal primo giorno, così come ai miei colleghi del Gruppo "VarLAB". In particolare, desidero ringraziare Vincenzo Armandi, Giacomo Vallasciani e Lorenzo Stacchio. La loro guida, la condivisione di conoscenze e l'ambiente di apprendimento stimolante hanno contribuito in modo significativo al mio sviluppo accademico. Sono grato per l'opportunità di essere stato parte di questa comunità di apprendimento. Senza di loro, questo traguardo non sarebbe stato raggiungibile, e sono eternamente grato per tutto ciò che hanno fatto per me.
Inoltre, dedico un ringraziamento anche al Dott. Simone Badioli per la generosa fornitura del dispositivo del RealSimGear.
Infine, desidero ringraziare l'intera comunità accademica, compresi i professori, i colleghi e gli amici, che hanno contribuito alla mia crescita e al mio apprendimento durante questi anni di studio universitario. Ognuno di voi ha giocato un ruolo importante nel mio percorso e sono grato per il vostro continuo sostegno e ispirazione.
In questo momento di felicità e di nuovi inizi, porto con me il ricordo di tutte le persone meravigliose che hanno influenzato positivamente la mia vita accademica. Grazie a tutti voi per avermi aiutato a raggiungere questo traguardo. Guardo con ottimismo al futuro e sono grato per tutte le esperienze che mi hanno portato fin qui.

Bibliografia

[1] Autodesck. *3d design software — sketchup*. URL: https://www.sketchup.com/.

[2] Autodesck. *Vr software for virtual reality design*. URL: https://www.autodesk.com/solutions/extended-reality.

[3] Ryan Anthony J de Belen et al. "A systematic review of the current state of collaborative mixed reality technologies: 2013–2018". In: *AIMS Electronics and Electrical Engineering* 3.2 (2019), pp. 181–223.

[4] Meredith Bricken e Chris M Byrne. "Summer students in virtual reality: A pilot study on educational applications of virtual reality technology". In: *Virtual reality*. Elsevier, 1993, pp. 199–217.

[5] Jamie Ian Cross et al. "Using extended reality in flight simulators: a literature review". In: *IEEE Transactions on Visualization and Computer Graphics* (2022).

[6] Lorenzo Donatiello, Lorenzo Gasparini e Gustavo Marfia. "Laying the path to consumer-level immersive simulation environments". In: *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE. 2020, pp. 1–4.

[7] Lorenzo Donatiello, Lorenzo Gasparini e Gustavo Marfia. "Towards an immersive visualization of consumer-level simulations of vehicular traffic". In: *2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE. 2021, pp. 1–4.

[8] Johannes Maria Ernst, Tim Laudien e Sven Schmerwitz. "Implementation of a mixed-reality flight simulator: blending real and virtual with a video-see-through head-mounted display". In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications V*. Vol. 12538. SPIE. 2023, pp. 181–190.

[9] MATTIA GIOVANNONI. "Sviluppo di un modello di wind shear valido nella fase di decollo per un simulatore di volo da addestramento piloti". In: (2018).

[10] Ian P Howard. *Perceiving in depth, volume 1: basic mechanisms*. Oxford University Press, 2012.

[11] S Howells e O Abuomar. “Depth Maps Comparisons from Monocular Images by MiDaS Convolutional Neural Networks and Dense Prediction Transformers”. In: *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE. 2022, pp. 1–6.

[12] S. Howells e O. Abuomar. “Depth Maps Comparisons from Monocular Images by MiDaS Convolutional Neural Networks and Dense Prediction Transformers”. In: *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. 2022, pp. 1–6. DOI: 10.1109/ICECCME55909.2022.9987767.

[13] Nabeel Khan. *Getting Started with Depth Estimation using MiDaS*. URL: https://medium.com/artificialis/getting-started-with-depth-estimation-using-midas-and-python-d0119bfe1159#:~:text=MiDaS%20is%20based%20on%20an,these%20features%20via%20up%2Dsampling..

[14] Jin Han Lee et al. “From big to small: Multi-scale local planar guidance for monocular depth estimation”. In: *arXiv preprint arXiv:1907.10326* (2019).

[15] XForcePC Micheal Brown. *Using the RealSimGear G1000 Suite in FS 2020*. URL: https://www.youtube.com/watch?v=Nm2EY0tAPz0.

[16] Rene Ranftl et al. “Dense monocular depth estimation in complex dynamic scenes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4058–4066.

[17] RealSimGear. *Download XPlane RSG plugIns*. URL: https://help.realsimgear.com/en/collections/2466132-downloads.

[18] Daniel Scharstein e Richard Szeliski. “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms”. In: *International journal of computer vision* 47 (2002), pp. 7–42.

[19] Stefan Stavrev e Dimitar Ginchev. “IMPLEMENTATION OF A SYSTEM FOR MONITORING BIO-PHYSIOLOGICAL DATA OF PILOTS VIA A FLIGHT SIMULATOR”. In: ().

[20] Varjo. *VR that’s officially certified for pilot training*. URL: https://www.youtube.com/watch?v=aHRbfnbX-q8&t=1466s.

[21] Back To VR. *Diciamo ADDIO a STEAM VR... OpenComposite con OpenXR Toolkit (guida)*. URL: https://www.youtube.com/watch?v=bstfF-_imkY&t=269s.

[22] David J Zielinski et al. “Enabling closed-source applications for virtual reality via opengl intercept-based techniques”. In: *2014 IEEE 7th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. IEEE. 2014, pp. 59–64.

[23] David J Zielinski et al. “Intercept tags: enhancing intercept-based systems”. In: *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*. 2013, pp. 263–266.