UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMNTO DI INFORMATICA

Corso di Laurea Triennale in Informatica

Tesi di Laurea

IDENTIFICAZIONE DEGLI OGGETTI MEDIANTE MIXED REALITY

Relatore:

Prof. Andrea Francesco Abate

Candidato:

Muto Giuliana

Mat:0512104598

Anno Accademico 2018/2019

Il Signore sarà per te luce eterna, il tuo Dio sarà il tuo splendore.

Is 60,19

Indice

1. Introduzione

Immergersi nella realtà virtuale comporta pericoli non trascurabili come l'allontanamento dall'ambiente circostante, creando situazioni pericolose per il proprio benessere fisico.

Lo scopo di questa tesi è quello di ridurre questo tipo di pericoli, sviluppando un'applicazione che permetta all'utente di poter godere liberamente della propria permanenza nel mondo virtuale, ma allo stesso tempo di individuare gli oggetti che si trovano anche nell'ambiente reale.

Quindi per non recare disturbo all'utente nell'area di gioco, gli oggetti che si trovano nell'ambiente reale diventano oggetti tridimensionali con cui interagire anche nel mondo virtuale.

L'applicativo sviluppato può essere anche usato per costruire dei videogames di tipo **ARG** (**Alternate Reality Game**), cioè videogiochi dove gli ostacoli sono gli oggetti che si trovano nella stanza.

Per sviluppare questa applicazione si è studiato a fondo il plug-in SRWorks SDK, in modo tale da poter capire il funzionamento della **segmentazione semantica 3D** e **dell'object detection**. Successivamente, è stato sviluppato l'algoritmo per lo scanner della scena, del salvataggio degli oggetti identificati, della creazione delle box mesh con relative texture e infine il caricamento di questi ultimi nel mondo virtuale.

Ad oggi l'applicazione riesce a scannerizzare e identificare: sedie, tavoli, muri, soffitti, pavimenti e pareti.

La tesi è suddivisa in quattro capitoli:

- Il primo capitolo è composto da un'introduzione sulla **Virtual Reality**, **Augmented Reality** e **Mixed Reality**, delle differenze che ci sono tra di esse e delle tecnologie che si trovano in commercio per sviluppare applicazioni in MR;
- Il secondo capitolo tratta le tecnologie, strumenti e software usate per questo progetto;
- Nel terzo capitolo viene descritto il lavoro svolto, si parlerà **della object detection**, dello scan della scena, delle modifiche apportate al plug in, degli script scritti e della creazione degli oggetti inseriti nella scena;
- Nel quarto capitolo verranno descritti i possibili sviluppi futuri;
- La tesi si chiude con l'ultimo capitolo con le conclusioni

1.1 La Virtual Reality

Già dalla seconda metà del XX secolo si è parlato di "Realtà Virtuale" (in inglese Virtual Reality o VR), ma solo da pochi decenni, questo termine si è insediato nel nostro vocabolario, soprattutto da quando, nel 2014, il colosso Facebook ha comprato Oculus, nota azienda per le tecnologie in realtà virtuale, portando così una nuova frontiera nelle case di milioni di persone. Infatti, secondo un'attenta analisi del Digi-Capital [1], nel 2020 l'era del virtuale sarà un business da oltre 150 miliardi di dollari. In figura 1.1 possiamo notare l'andamento del mercato della Realtà Aumentata e della VR e di come dall'anno 2016 all'anno 2022 aumenta a dismisura. Ad oggi, oltre a Facebook, altri grandi colossi si sono interfacciati in questo campo, come ad esempio Valve e HTC con HTC Vive, Samsung con Gear VR, Sony con PlayStation VR, Nintendo con Nintendo Labo Kit VR, Microsoft con HoloLens, Acer con Acer VR Headset, Asus con Asus Windows Mixed Reality Headset, Lenovo con Lenovo Explorer o Hauwei con Hauwei VR Glass.

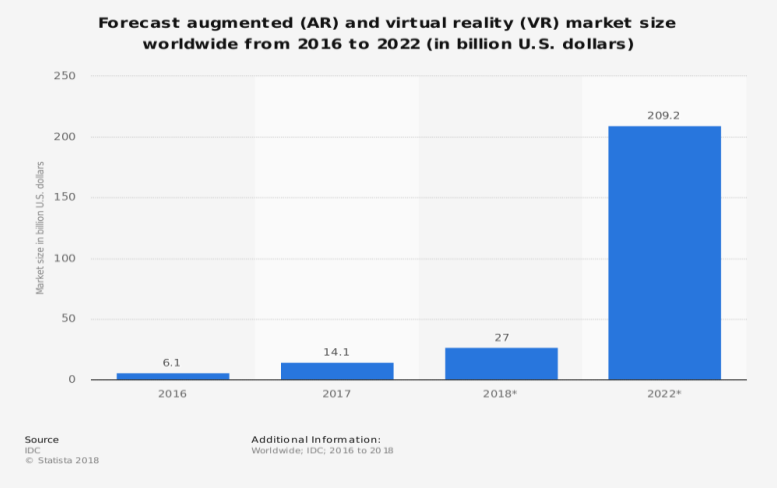

Fig. 1.1 *Dimensione del mercato di realtà aumentata (AR) e realtà virtuale (VR) prevista in tutto il mondo dal 2016 al 2022 (in miliardi di dollari USA)*

Con il termine Virtual Reality (VR) si intende una realtà simulata, costruita al computer, all'interno della quale è possibile muoversi liberamente. L'accesso al mondo virtuale è reso possibile dai Visori VR e dagli accessori, quali joypad, guanti e altro, sviluppati per interagire al meglio con il mondo virtuale, infatti è possibile esplorare ogni singolo centimetro e direzione. Dunque, viene creato un mondo simulato e tridimensionale, proprio come se fosse il mondo reale.

La Virtual Reality ad oggi è utilizzata in tantissimi campi. Come:

- **Interior design**: per ricreare ambienti di ogni tipo;
- **Psicoterapia**: offre al paziente di partecipare attivamente al riconoscimento e alla presa di consapevolezza di pensieri ed emozioni;
- **Medicina**: con la possibilità di esercitazioni mediche o simulazioni di interventi;
- **Militare**: per l'addestramento militare con rischi ridotti al minimo;
- **Turismo**: per esplorare luoghi inaccessibili;

- **Arte**: per la possibilità di osservare opere di tutto il mondo;
- **Gaming Pubblicità**: la pubblicità diventa immersiva, una vera e propria esperienza per l'utente;
- **Test di prodotti**: permette di creare ambienti dove inserire prodotti e testarli;
- **Videogames:** lo sviluppo di giochi in VR è un mercato in continuo aumento perché riesce a portare l'esperienza di gioco a livelli nettamente superiore. In figura 1.2 è riportato l'esempio di Beat Saber, un gioco in cui bisogna colpire dei cubi sospesi in aria con delle spade laser, il tutto al ritmo di musica. Questo tipo di esperienze non sarebbero realizzabili se non in un contesto di realtà virtuale.

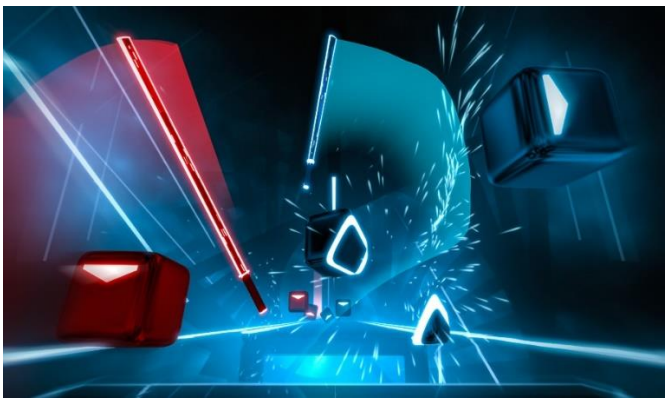

Fig. 1.2 *Beat Saber [2]*

L'obiettivo principale della programmazione in VR è avere una perfetta corrispondenza tra i movimenti della testa, del corpo e degli occhi. Ed è proprio questo quello che rende l'ambiente virtuale realistico e piacevole. I problemi sorgono infatti quando c'è un ritardo tra le azioni della persona e la risposta del sistema che va a disturbare l'esperienza dell'utente.

1.2 Augmented Reality

Per Realtà Aumentata (in inglese Augmented Reality o AR) si intende un arricchimento, con strumenti digitali, in tempo reale della percezione sensoriale dell'utente. L' AR è destinata, già nei prossimi anni, a cambiare le nostre abitudini, il nostro modo di vedere il mondo. Un principio dell'AR è quello dell'overlay: la fotocamera legge l'oggetto nell'inquadratura, il sistema lo riconosce e attiva un nuovo livello di comunicazione che si va a sovrapporre e a integrare perfettamente alla realtà, potenziando la quantità di dati di dettaglio in relazione a quell'oggetto.

In figura 1.3 è mostrato Pokemon Go, un gioco che ha spopolato tra i giovanissimi (e non) nel 2016. Il gioco sfrutta la fotocamera per dare l'illusione al giocatore di interagire con personaggi virtuali nel suo mondo reale, questo rappresenta uno dei primi utilizzi di successo dell'AR.

Fig. 1.3 *Pokemon Go [3]*

L'AR può essere fruita attraverso diversi tipi di device:

- **Sistemi di proiezione ottica**: pc, tablet e smartphone;
- **Sistemi indossabili**: occhiali, guanti, cuffie e altro.

L'AR rappresenta una rivoluzione silenziosa che sta producendo risultati interessanti, già in diversi settori: dalla medicina all'ambito militare, dallo sport al marketing, dal turismo all'intrattenimento. Tra i brand del retail che sono stati dei precursori a utilizzare l'Augmented Reality troviamo Lego, Tesco e Ikea.

Ad esempio, per evitare che i clienti più curiosi aprissero le confezioni e per far capire meglio l'utilizzo del gioco contenuto, Lego usa l'AR come forma di smart packaging: in un corner dedicato, inquadrando una confezione dal monitor si può vedere partire una sorta di trailer con i personaggi in animazione coinvolti in una gag.

Secondo le indagini della Gartner Inc [4], già nel 2020 AR e VR potranno rivoluzionare la Customer Experience permettendo ai clienti di visualizzare i prodotti in contesti "reali" e di ricevere offerte personalizzate. Un sondaggio svolto tra luglio e agosto 2018 su 97 retailer in Europa, Usa, Canada e Cina ha rivelato che, entro il 2020, il 46% dei negozianti intende adottare soluzioni di AR o VR e ora la società di ricerche stima che, nel 2020, 100 milioni di consumatori faranno shopping in realtà aumentata sia in-store che online.

1.3 La Mixed Reality

Le novità tecnologiche sono certamente in continua evoluzione; oggi si parla molto di VR e AR, ma non tutti sanno che nell'ultimo anno Google Trends ha riscontrato un aumento nelle ricerche sulla "Mixed Reality" (in italiano Realtà Mista o MR) di cinque volte in più rispetto agli scorsi anni.

La MR è un termine coniato nel 1994 da un articolo di ricerca scritto da Paul Milgram and Fumio Kishino [5], viene descritta come "una particolare sottoclasse di tecnologie legate alla realtà virtuale che comportano la fusione di mondi reali e virtuali". Più specificamente, affermano che la Mixed Reality è una fusione di mondi reali e mondi virtuali da qualche parte lungo la "reality-vituality continuum" che collega ambienti completamente reali con ambienti completamene virtuali. Possiamo vederla come un potenziamento della Realtà Virtuale, una sorta di Realtà Aumentata 4.0.

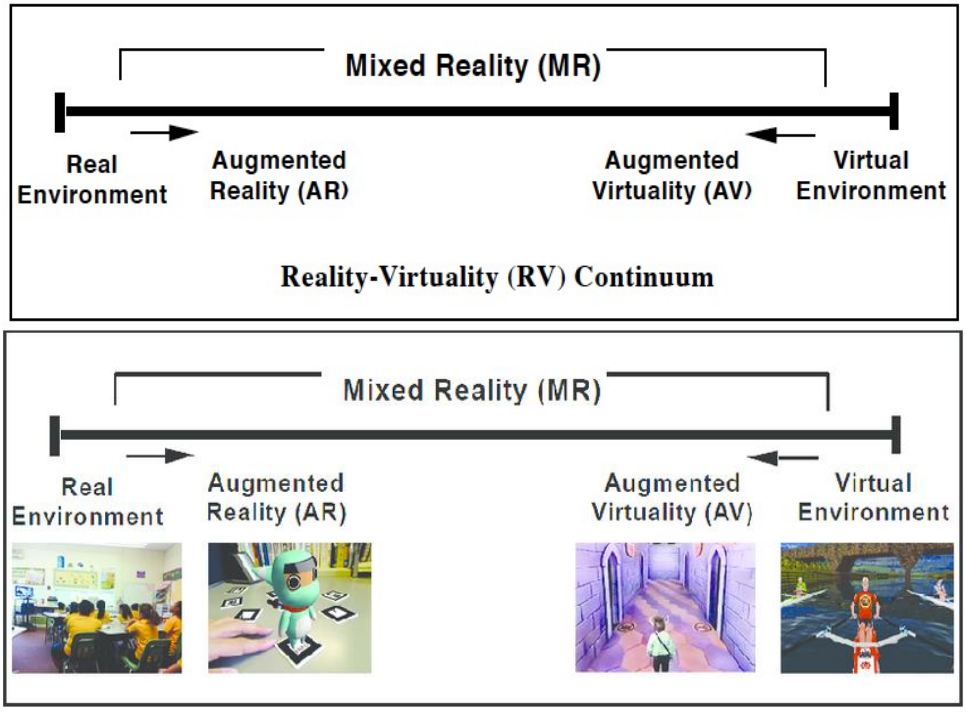

Fig. 1.4 *Reality-vituality continuum*

In figura 1.4 vediamo come la MR si posiziona al di sopra della VR e AR, creando un legame tra le due. Ciò significa che l'area tra i due estremi (Fig.1.4), in cui si mescolano sia il reale che il virtuale, è chiamata realtà mista, appunto Mixed Reality. A sua volta è costituito dalla Realtà Aumentata (Augmented Reality) dove il virtuale aumenta la realtà reale e dalla Virtualità Aumentata (Virtual Environment), dove il reale aumenta la realtà virtuale.

I campi di utilizzo della Mixed Reality sono innumerevoli, tra cui:

- **Addestramento militare:** per simulare la realtà di combattimento e per addestrare il personale militare in una varietà di senari. Viene addestrato anche il personale medico in situazioni in cui i soldati non sarebbero probabilmente sopravvissuti. Questo comporta molti vantaggi, come la

riduzione della quantità di munizioni spese durante l'addestramento, la possibilità di simulare qualsiasi tipo di campo da battaglia;

- **Training e formazione:** la possibilità di formare i dipendenti di un'azienda con un'esperienza immersiva ed interattiva.
- **Lavoro a distanza:** la MR consente ad una forza lavoro di team remoti di poter lavorare insieme e affrontare sfide aziendali.
- **Mockup funzionale:** la MR può essere utilizzata per creare modelli che combinano elementi fisici e digitali. Con l'uso della localizzazione e mappatura simultanea (SLAM), i modelli possono interagire con il mondo fisico per utilizzare funzionalità come la permanenza degli oggetti.
- **Campo medico:** è possibile combinare gli smartglass con i processi chirurgici.
- **Ottimizzazione degli spazi:** piccoli uffici possono diventare multifunzionali a costi estremamente bassi. Si può vedere un ufficio piccolo e vuoto pieno di lavagne bianche su tutte le pareti, e scriverci sopra con le dita.
- **Operazioni di riparazione:** è possibile implementare anche le operazioni di riparazione mostrando ogni parte componente della macchina al dipendente e le istruzioni per ripararla.

In sostanza, la MR consente all'utente di vedere il mondo reale (come succede in AR) ma anche oggetti virtuali credibili (come succede in VR).

La caratteristica fondamentale è che è possibile ancorare gli oggetti virtuali in un posto nello spazio reale. Quando l'utente si sposta, l'oggetto tridimensionale generato al computer sparisce dallo spazio visivo, ma se si ritorna con lo sguardo nel punto in

cui questo era stato posizionato lo si ritrova proprio in quel luogo. Questo permette di trattare degli oggetti virtuali come se fossero reali.

Attenzione a non confondere i tre tipi di “realtà”: anche se sono tecnologie immersive (tecnologia che integra elementi virtuali e mondo reale) e hanno somiglianze, ci sono delle differenze.

Fig. 1.5 *Differenza tra VR, AR e MR*

Nella VR gli utenti sono completamente immersi in una realtà generata al computer. Quando un utente indossa visore VR o un auricolare VR, vede e sente lo spostamento tra oggetti virtuali su uno schermo e il suo cervello crede alla realtà che vede e sente. L’AR è la tecnologia che sovrappone le informazioni digitali al mondo reale, piuttosto che offrire un'esperienza virtuale completamente immersiva, la realtà aumentata migliora il mondo reale con immagini, testo e altre informazioni virtuali tramite dispositivi come display heads-up, smartphone, tablet, obiettivi intelligenti e occhiali AR.

Invece, un ambiente MR va oltre la realtà aumentata perché gli utenti possono interagire in tempo reale con oggetti virtuali posizionati nel mondo reale. Questi oggetti virtuali risponderanno e reagiranno agli utenti come se fossero oggetti reali. È la tecnologia di riconoscimento di gesti / sguardi / voce attraverso un paio di controller di movimento o attraverso l'auricolare MR che aiuta a offrire un'esperienza credibile in realtà mista. Ci vuole molta più potenza di elaborazione per consentire un'esperienza di realtà mista rispetto a un'esperienza di realtà virtuale o aumentata.

1.4 Tecnologie attuali per la Mixed Reality

Durante il Mobile World Congress 2019 di Barcellona Microsoft ha presentato la seconda generazione del proprio visore dedicato alla Mixed Reality: HoloLens 2, mostrato in figura 1.6

Nato per scopi industriali, ma utilizzabile anche in campo medico, HoloLens 2 permette di vedere più ologrammi allo stesso tempo grazie al campo visivo molto più ampio, di leggere il testo e riconoscere anche i più piccoli dettagli sulle immagini 3D in modo semplice e comodo con una risoluzione leader del settore. Basta alzare la visiera per tornate nel mondo reale. È possibile muoversi liberamente nella stanza senza l'ausilio di cavi o dispositivi esterni. L'headset HoloLens 2 ha un computer integrato con interfaccia WLAN [6].

Fig. 1.6 *HoloLens 2*

Legato a questo visore è la piattaforma Windows Mixed Reality, infatti HoloLens è nato appunto per sfruttare le potenzialità del sistema. La piattaforma permette di scaricare una serie di applicazioni, come Minecraft, Skype, software sviluppati dalla NASA e altro ancora, il tutto sfruttando la Mixed Reality.

Oltre a HoloLens, ci sono una vasta gamma di visori che sfruttano questo sistema, come il visore VR della Acer Mixed Reality Headset, mostrato in figura 1.7, è compatibile con una varietà di computer portatili e desktop che soddisfano i requisiti di Windows Mixed Reality, l'auricolare Acer consente di guardare spettacoli in un ambiente virtuale, giocare a giochi di realtà virtuale, viaggiare per il mondo attraverso holo-tour, guardare immersive 3D, a 360 ° e video 4K e sfogliare i siti Web [7].

Fig. 1.7 *Acer Mixed Reality Headset*

2. Tecnologie utilizzate

Per questo progetto di tesi è stato utilizzato un visore che permettesse all'utente di immergersi in un mondo virtuale, in cui sono stati inseriti gli oggetti del mondo reale. È stato usato **l'HTC Vive Pro** e come ambiente di sviluppo è stato usato **Unity**. Inoltre, si è serviti del plug-in **Vive SRWorks SDK**, perché permette di accedere alle telecamere stereo frontali, così da poter mescolare i due mondi. Inoltre, per utilizzare il visore con Unity, è stato adoperato **SteamVR** e **OpenVR**. La scelta di tale attrezzatura è stata decisa perché Unity è totalmente gratuito e perché è un ambiente di sviluppo multipiattaforma: i contenuti possono essere creati usando un PC con Windows o un Mac, ma possono poi essere installati su qualsiasi dispositivo presente sul mercato (Smartphone Android e IOS, console Psp, Xbox o switch e altro ancora).

Per sfruttare al massimo il visore, è stato usato un computer più performante, con una GPU di fascia alta con scheda Quadro P4000. I processori richiesti hanno la potenza di calcolo di un Intel i5-4590 e hanno almeno 48 GB di RAM disponibili. Usato con le specifiche raccomandate, il Vive Pro è nettamente superiore rispetto ad altri visori in commercio. L'aumento della risoluzione offre una maggiore chiarezza nella scena generale, più evidente negli oggetti in lontananza, nei dettagli e soprattutto nei testi.

2.1 HTC Vive Pro

HTC Vive Pro è un visore che ha aiutato gli sviluppatori a creare mondi virtuali con cui l'utente può interagire usando mani e piedi. Progettato per utenti di livello professionale, include VIVE Pro HMD. Ha una vestibilità raffinata ed un ottimo equilibrio e comfort per un uso prolungato.

Fig. 2.1 *HTC Vive Pro: visore*

Ha una risoluzione totale di 2880x1600 pixel (1400x1600 pixel ogni occhio), ha una doppia telecamera frontale, usata per il tracciamento delle mani, per un miglioramento del sistema di rilevamento dei confini della stanza reale (miglioramento di Chaperone System di SteamVR) o per un migliore tracciamento di oggetti nella stanza. Raymond Pao, vicepresidente di HTC Vive, ha spiegato che le telecamere rilevano i dati di profondità da uno a due metri di distanza e segnalano

la presenza di oggetti del mondo reale così che gli utenti possano evitarli. Pao fa anche sapere che HTC sta distribuendo dei kit per sviluppatori per la fotocamera, così da vedere quali altri usi si potrebbero fare nei giochi e nelle esperienze, appunto SRWorks SDK, plugin usato per questo progetto.

HTC Vive Pro ha anche le cuffie integrate; include controlli del volume e offre una qualità del suono straordinariamente ricca e potente. Inoltre, il Vive Pro supporta la possibilità di regolare la distanza delle lenti dagli occhi, il che lo rende più comodo per chi indossa gli occhiali.

Configurare un visore come Vive richiede un po' di tempo ed è ancora necessario un PC abbastanza potente per eseguire l'operazione, insieme a una coppia di controller Vive e un paio di stazioni base di HTC per sbloccare la capacità di tracciamento dei movimenti. Il collegamento del visore al PC richiede lo stesso procedimento, anche la periferica di collegamento che unisce i due è migliorata nel Pro, essendoci un cavo in meno di cui preoccuparsi e un pulsante di accensione apposito.

Fig. 2.2 *HTC Vive Pro Visore: controller Vive e stazioni base*

Essendo necessarie le stazioni base, lo spazio richiesto per utilizzare entrambi i visori è ancora un minimo di 2 metri x 1,5 metri e il massimo è di circa 5 metri. Questo è destinato a cambiare, almeno nel caso di Vive Pro che supporterà le stazioni base di prossima generazione che verranno lanciate entro la fine dell'anno 2019. Ciò consentirà ai proprietari di Vive Pro di collegarne più di due per estendere l'area di gioco a 100 metri quadrati, anche se per adesso entrambi sono limitati alla stessa quantità di spazio VR libero [8]

Per chi ha disponibilità economica e un ampio spazio dedicato alla VR, HTCVive Pro è una combinazione allettante di tracciamento affidabile, visione nitida e libertà di movimento.

2.2 Unity

Unity è diventato col tempo il riferimento di tutte le produzioni “indie” (piccoli gruppi di sviluppo senza un grosso publisher alle spalle) anche in virtù del suo costo veramente accessibile, infatti è totalmente gratuito.

L’hardware VR ad ora ha ancora dei limiti specialmente nel rendering in tempo reale delle luci. Dobbiamo pensare che, se per vedere un videogioco in 4K con tutti gli shader, ombre, dettagli, etc., abbiamo bisogno di macchine potenti, per la VR bisogna considerare che il computer deve renderizzalo ad una qualità buona, ma soprattutto, per avere una stereoscopia e la precisione della profondità perciò “raddoppia” il lavoro di processing delle immagini. Se per i videogiochi normali bastano schede video di fascia media, per la VR, le caratteristiche della GPU devono sottostare a degli standard medio-alti in termini di memoria e velocità per poter visualizzare degnamente un videogioco. Per questo i programmatori di Unity nell’ultima versione hanno creato dei modelli o template dedicati alla VR. Questi template sono adattati ai visori perciò creano tutti gli elementi di rendering adattati alla visualizzazione su dispositivi VR.

2.3 HTC Vive SRWorks SDK

Dopo il lancio di HTC Vive Pro, gli sviluppatori hanno avuto accesso alle telecamere stereo frontali in modo da poter rilevare la profondità con i sensori stereo RGB e di eseguire la percezione 3D. L'SDK espande le capacità e le funzionalità della tecnologia del Vive VR. Con questo, gli sviluppatori saranno in grado di portare il mondo reale nella realtà virtuale.

Le immagini della doppia fotocamera frontale vengono elaborate tramite i seguenti moduli Vive SRWorks, mostrato in figura 2.2, dopodiché l'output viene renderizzato tramite Unity e poi consegnato a SteamVR:

- **See-through module** (modulo trasparente): consente di visualizzare l'ambiente reale al di fuori della realtà virtuale. Inoltre, questo modulo fornisce anche effetti di materiali e texture.
- **Depth module** (modulo di profondità): fornisce il rilevamento della profondità degli oggetti nel mondo reale. È possibile avere informazioni come profondità di campo, lunghezza focale o distanza di un oggetto.
- **AI Vision**: fornisce informazioni sulla segmentazione di oggetti umani o interni nel mondo reale. Permette di riconoscere scene 3D per applicazioni in Mixed Reality.
- **3D reconstruction module** (modulo di ricostruzione 3D): supporta il salvataggio di scene 3D, incluse mesh, materiali, trame, colori, collider e rilevamento di collisioni. Se questo modulo supporta l'AI Vision, esporterà

oggetti semantici con estensione ".obj" e le informazioni extra in un file ".xml".

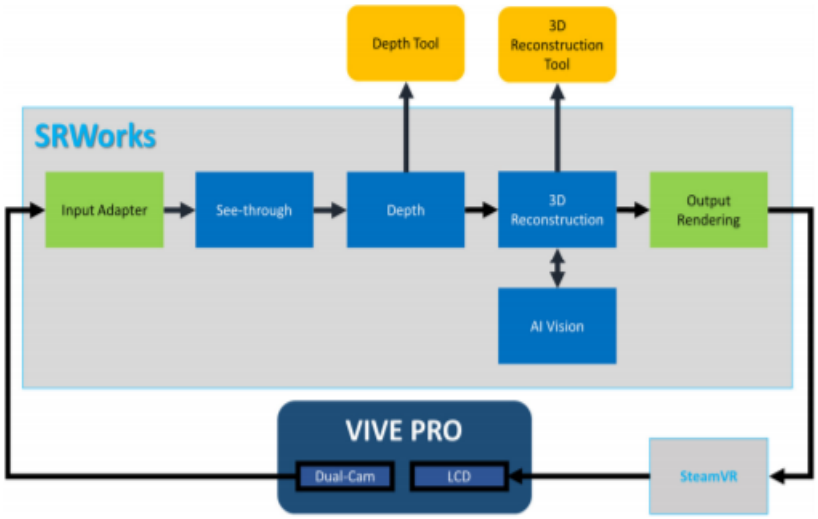

Fig. 2.3 *Schema SRWorks*

La doppia fotocamera del Vive Pro offre una visione stereo del mondo reale, imitando la visione umana tramite due obiettivi ciascuno con il proprio sensore di immagine. Per gli sviluppatori che intendono utilizzare la tecnologia see-through con i loro contenuti VR, la doppia fotocamera offre due vantaggi principali:

- **Percezione 3D**: poiché ci sono telecamere separate utilizzate per gli occhi sinistro e destro, la percezione 3D è possibile quando si guarda il mondo reale.
- **Rilevamento della profondità:** le informazioni sulla profondità possono essere utilizzate per sviluppi avanzati, ad esempio per il tracciamento del

corpo, della mano e del viso. Questo richiede un'ulteriore implementazione e test eseguiti dagli sviluppatori.

Utilizzando l'SDK SRWorks, gli sviluppatori possono creare contenuti VR in grado di allineare contemporaneamente il mondo reale e la realtà virtuale. La funzione di scoperta, supportata dalla ricostruzione 3D e dal rilevamento della profondità [9].

Per il progetto sono stati utilizzati i moduli **AI Vision** e **3D Reconstruction**, poiché danno la possibilità di riconoscere soffitto, pavimento, parete, sedia, tavolo e letto; così è stato possibile mescolare con continuità, il mondo reale e virtuale in uno stile di vita naturale.

2.4 SteamVR e OpenVR

Per usare il plug-in è stato importato la versione precedente di SteamVR, ossia pacchetto SteamVR Unity plugin 1.2.3, reperibile in un repository GitHub [10], rilasciato il 10 gennaio 2018. Utilizza il vecchio stile di input con ID pulsante anziché le azioni e i set di azioni di OpenVR. Valve mantiene il plug-in di Unity SteamVR, è una libreria ufficiale che rende più facile lo sviluppo delle applicazioni VR.

Open VR, la cui struttura è presentata in figura 2.4, è un kit di sviluppo software (SDK) e una libreria (API) sviluppata da Valve per supportare la piattaforma SteamVR. Open VR viene utilizzata come interfaccia tra l'hardware e il software. Open VR permette ad un gioco di interagire con gli HMD.

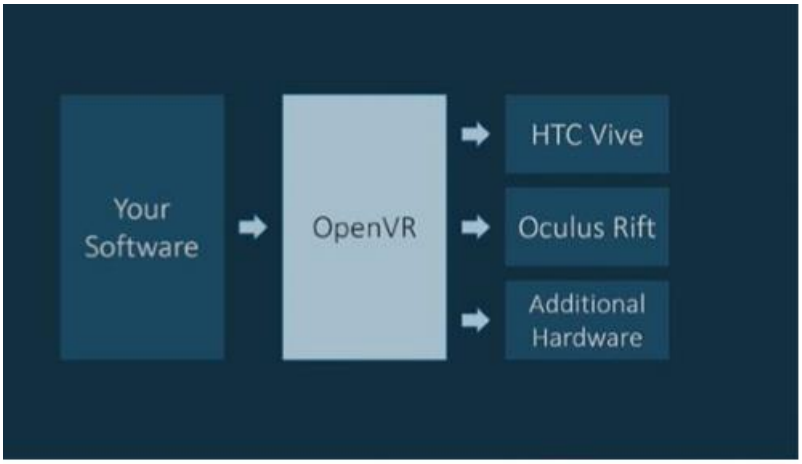

Fig. 2.4 *Architettura OpenVR [10]*

3. Caso di studio

In questo capitolo verranno descritti tutti i passaggi che hanno portato alla conclusione del progetto. Si parlerà della object detection, di come è stata implementata e della costruzione della scena di gioco. Verrà descritto anche la Semantic Segmentation e di come sono stati usati i suoi script, delle modifiche apportate agli script delle telecamere frontali e dello spatial scan che ha permesso di scannerizzare la scena. Infine, parleremo di come sono state create le mesh cube e i game object e della sperimentazione dell'applicazione.

Poiché il caso di studio preso in esame non è stato ancora trattato, non ci sono stati spunti per la costruzione degli algoritmi. L'unico contribuito è stato il plug-in che ha permesso di scannerizzare gli oggetti e ha facilitato il lavoro per la distinzione di tali oggetti.

Le figure presenti in questo capitolo sono state prese dal progetto.

3.1 Object Detection

L’Object detection (in italiano rilevamento di oggetti) nella visione artificiale è la capacità di trovare un determinato oggetto in una sequenza di immagini o video.

Ogni oggetto in un'immagine, ha molte caratteristiche che possono essere estratte in modo da fornire una descrizione "caratteristica" dell'oggetto. Questa descrizione estratta da una immagine campione può poi essere utilizzata per identificare l'oggetto durante il tentativo di individuare l'oggetto in una immagine di test contenente più oggetti. È importante che l'insieme di caratteristiche estratte dall'immagine campione sia insensibile a variazioni di scala delle immagini, ai disturbi, all'illuminazione e alle distorsioni geometriche, in modo da rendere affidabile il riconoscimento. Quindi si occupa di identificare e localizzare oggetti di determinate classi. L’interpretazione dell’oggetto può essere eseguita in vari modi e in questo progetto è stato creato un riquadro di delimitazione attorno all’oggetto, come mostrato in figura 3.1, che successivamente è stato reso trasparente.

Fig.3.1 *Object Detection di un oggetto*

Gli oggetti diversi o anche dello stesso tipo, possono avere proporzioni e dimensioni diverse a seconda della dimensione della stanza e della distanza dell'oggetto dalla telecamera. È stato deciso di utilizzare questa tecnica perché per raggiungere lo scopo di questo progetto è necessario avere un mezzo che permetta di poter scannerizzare gli oggetti del mondo reale per poi poterli inserire nel mondo virtuale. Per ottenere il riquadro di delimitazione del game object in base al tipo di oggetto della scena, è stato utilizzato il metodo **GetElementsBoundingBoxMeshes** mostrato in figura 3.2**,** che si trova nello script **ViveSR_SceneUnderstanding.**

```
2 riferimenti
public void GetElementsBoundingBoxMeshes(int tagObj, Element tagIdElement, ref List<GameObject> boxObj)
{
    List<int> MeshDataIndices = new List<int>();

    //top lines
    MeshDataIndices.Add(0); MeshDataIndices.Add(1);
    MeshDataIndices.Add(1); MeshDataIndices.Add(2);
    MeshDataIndices.Add(2); MeshDataIndices.Add(3);
    MeshDataIndices.Add(3); MeshDataIndices.Add(0);
    //bottom lines
    MeshDataIndices.Add(4); MeshDataIndices.Add(5);
    MeshDataIndices.Add(5); MeshDataIndices.Add(6);
    MeshDataIndices.Add(6); MeshDataIndices.Add(7);
    MeshDataIndices.Add(7); MeshDataIndices.Add(4);
    //vertical lines
    MeshDataIndices.Add(0); MeshDataIndices.Add(4);
    MeshDataIndices.Add(1); MeshDataIndices.Add(5);
    MeshDataIndices.Add(2); MeshDataIndices.Add(6);
    MeshDataIndices.Add(3); MeshDataIndices.Add(7);

    //foreach (Element each in GetElements(enums[tagObj]))
    {
        GameObject Obj = new GameObject("Box_" + tagIdElement.tag + "_" + tagIdElement.id);
        MeshFilter mf = Obj.AddComponent(typeof(MeshFilter)) as MeshFilter;
        MeshRenderer mr = Obj.AddComponent(typeof(MeshRenderer)) as MeshRenderer;

        mr.shadowCastingMode = UnityEngine.Rendering.ShadowCastingMode.Off;
        mr.material.shader = Shader.Find("Transparent/Diffuse");
        mr.material.color = Color.clear;
```

```
mf.mesh = new Mesh();

mf.mesh.MarkDynamic();
List<Vector3> MeshDataVertices = new List<Vector3>();


MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMinPoint.x, tagIdElement.bBoxMinPoint.y, tagIdElement.bBoxMinPoint.z)); //0
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMinPoint.x, tagIdElement.bBoxMinPoint.y, tagIdElement.bBoxMaxPoint.z)); //1
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMinPoint.x, tagIdElement.bBoxMaxPoint.y, tagIdElement.bBoxMaxPoint.z)); //2
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMaxPoint.x, tagIdElement.bBoxMinPoint.y, tagIdElement.bBoxMinPoint.z)); //3
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMinPoint.x, tagIdElement.bBoxMaxPoint.y, tagIdElement.bBoxMinPoint.z)); //4
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMinPoint.x, tagIdElement.bBoxMaxPoint.y, tagIdElement.bBoxMaxPoint.z)); //5
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMaxPoint.x, tagIdElement.bBoxMaxPoint.y, tagIdElement.bBoxMaxPoint.z)); //6
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMaxPoint.x, tagIdElement.bBoxMaxPoint.y, tagIdElement.bBoxMinPoint.z)); //7




mf.sharedMesh.Clear();
mf.sharedMesh.SetVertices(MeshDataVertices);
mf.sharedMesh.SetIndices(MeshDataIndices.ToArray(), MeshTopology.Lines, 0);
Obj.SetActive(false);
boxObj.Add(Obj);
```

```
BoxCollider box = Obj.AddComponent<BoxCollider>();
Vector3 position = box.center;
Vector3 scale = box.size;
instanziatePrefab.InitializationPrefabs(tagIdElement.tag, position, scale, Obj);
```

Fig.3.2 *Object Detection di un oggetto*

Prima di spiegare il funzionamento del metodo, dobbiamo avere chiaro un concetto fondamentale, ovvero come funzionano le **Mesh Cube** in Unity.

Le **Mesh** sono create da elementi, punti e linee. I punti sono chiamati vertici e un singolo punto è chiamato vertice. I vertici definiscono i punti nello spazio 3D. In Unity, tre vertici collegati formano un triangolo e questi triangoli definiscono le mesh degli oggetti. In questo script, è stata creata la mesh dinamicamente nella lista **MeshDataIndices,** come è possibile vedere in figura 3.3.

```
public void GetElementsBoundingBoxMeshes(int tagObj, Element tagIdElement, ref List<GameObject> boxObj)
{
    List<int> MeshDataIndices = new List<int>();

    //top lines
    MeshDataIndices.Add(0); MeshDataIndices.Add(1);
    MeshDataIndices.Add(1); MeshDataIndices.Add(2);
    MeshDataIndices.Add(2); MeshDataIndices.Add(3);
    MeshDataIndices.Add(3); MeshDataIndices.Add(0);
    //bottom lines
    MeshDataIndices.Add(4); MeshDataIndices.Add(5);
    MeshDataIndices.Add(5); MeshDataIndices.Add(6);
    MeshDataIndices.Add(6); MeshDataIndices.Add(7);
    MeshDataIndices.Add(7); MeshDataIndices.Add(4);
    //vertical lines
    MeshDataIndices.Add(0); MeshDataIndices.Add(4);
    MeshDataIndices.Add(1); MeshDataIndices.Add(5);
    MeshDataIndices.Add(2); MeshDataIndices.Add(6);
    MeshDataIndices.Add(3); MeshDataIndices.Add(7);
```

Fig.3.3 *Creazione dinamica degli indici delle mesh*

In figura 3.4 è mostrato come sono posizionati i vertici di un mesh cube.

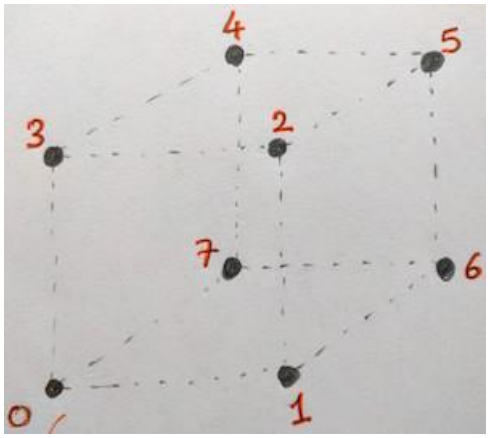

Fig.3.4 *Vertici del mesh cube*

L'ordine dei vertici di ciascun triangolo è chiamato **ordine di avvolgimento**. L'ordine di avvolgimento, come mostrato in figura 3.5, può essere utilizzato per determinare se il triangolo viene visto dalla parte anteriore o posteriore. Unity3D utilizza l'ordine di avvolgimento in **senso orario** per determinare i triangoli rivolti in avanti.

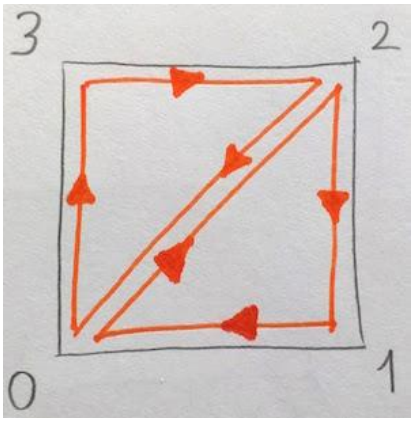

Fig.3.5 *Funzionamento dell'ordine di avvolgimento*

Il cubo ha sei facce composte da dodici triangoli. Nello script, mostrato in figura 3.6, i triangoli sono stati definiti dalla lista dei Vector3 **MeshDataVertices**.

```
mf.mesh = new Mesh();

mf.mesh.MarkDynamic();
List<Vector3> MeshDataVertices = new List<Vector3>();


MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMinPoint.x, tagIdElement.bBoxMinPoint.y, tagIdElement.bBoxMinPoint.z)); //0
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMinPoint.x, tagIdElement.bBoxMinPoint.y, tagIdElement.bBoxMaxPoint.z)); //1
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMaxPoint.x, tagIdElement.bBoxMinPoint.y, tagIdElement.bBoxMaxPoint.z)); //2
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMaxPoint.x, tagIdElement.bBoxMinPoint.y, tagIdElement.bBoxMinPoint.z)); //3
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMinPoint.x, tagIdElement.bBoxMaxPoint.y, tagIdElement.bBoxMinPoint.z)); //4
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMinPoint.x, tagIdElement.bBoxMaxPoint.y, tagIdElement.bBoxMaxPoint.z)); //5
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMaxPoint.x, tagIdElement.bBoxMaxPoint.y, tagIdElement.bBoxMaxPoint.z)); //6
MeshDataVertices.Add(new Vector3(tagIdElement.bBoxMaxPoint.x, tagIdElement.bBoxMaxPoint.y, tagIdElement.bBoxMinPoint.z)); //7
```

Fig.3.6 *Creazione dei vertici del mesh cube*

Sia i vertici che i triangoli son stati poi settati. Inoltre, come mostrato in figura 3.7, per ogni mesh cube creato viene assegnato un nome, definito dalla parola Box, il tag dell'oggetto (Chair, Table, Wall, Ceiling, Floor) e dall'id che è un numero progressivo. Inoltre, viene anche assegnato un **MeshFilter** (prende una mesh dagli asset e la passa al mesh per il rendering sullo schermo) e un **MeshRenderer** (esegue il rendering delle mesh inserite da MeshFilter). Inoltre, è stato anche dotato di un **BoxCollider**, ma di questo ne parleremo più avanti.

```
GameObject Obj = new GameObject("Box_" + tagIdElement.tag + "_" + tagIdElement.id);
MeshFilter mf = Obj.AddComponent(typeof(MeshFilter)) as MeshFilter;
MeshRenderer mr = Obj.AddComponent(typeof(MeshRenderer)) as MeshRenderer;

mr.shadowCastingMode = UnityEngine.Rendering.ShadowCastingMode.Off;
mr.material.shader = Shader.Find("Transparent/Diffuse");
mr.material.color = Color.clear;
```

```
mf.sharedMesh.Clear();
mf.sharedMesh.SetVertices(MeshDataVertices);
mf.sharedMesh.SetIndices(MeshDataIndices.ToArray(), MeshTopology.Lines, 0);
Obj.SetActive(false);
boxObj.Add(Obj);

BoxCollider box = Obj.AddComponent<BoxCollider>();
Vector3 position = box.center;
```

Fig.3.7 *Attributi del mesh cube*

3.2 Costruzione della scena

Per rendere confortevole la permanenza dell'utente nel mondo virtuale, è stato costruito un prato e un cielo con le nuvole utilizzando un asset preso nell'asset store di Unity [12] come è mostrato in figura 3.8.

Fig.3.8 *Vista dell'utente durante l'esecuzione*

È stato poi importato il plug-in di SRWorks SDK e il plug-in SteamVR. In questi plug-in, ci sono molti contenuti per lo sviluppo VR, AR e MR, per tanto è stato risparmiato tempo sull'implementazione di alcune componenti, come ad esempio, è stato utilizzato il sistema di interazione, composto da una serie di script, prefabbricati e altre risorse. In particolar modo si è serviti del prefab del **Player**, che permette all'utente di svolgere le azioni di input necessarie. In figura 3.9 è mostrato come è composto il prefab del Player.

Fig.3.9 *Prefab Player*

Il Player si trova al centro dell'intero sistema e si comporta come un singleton, il che significa che deve esserci solo un oggetto di questo tipo nella scena. Il Player non fa molto, tranne tenere traccia delle mani e dell'hmd. È possibile accedervi a livello globale durante tutto il progetto e molti aspetti del sistema di interazione presuppongono che l'oggetto Player esista sempre nella scena. La classe del giocatore è impostata per utilizzare le icone per mostrare i piedi e le mani nella vista della scena dell'editor, ma a causa del modo in cui Unity funziona, queste icone, chiamate **Gizmos,** (mostrate in figura 3.10) devono trovarsi in una cartella specifica per funzionare.

Fig.3.10 *icone Gizmos*

Invece, il prefab **ViveSR,** la cui struttura è mostrata in figura 3.11, permette l'interazione con le videocamere e della ricostruzione 3D, parleremo di entrambe più avanti.

Fig.3.11 *Prefab ViveSR*

3.3 See-through

Esistono tre telecamere per See-through. Una è responsabile del rendering solo dell'immagine dell'occhio sinistro, un'altra per il rendering solo dell'immagine dell'occhio destro e l'altra per il rendering di tutti gli oggetti di gioco tranne due piani di immagini. Per impostare le telecamere stereo frontali del visore, è stato modificato lo script **ViveSR_DualCameraRig**, mostrato in figura 3.12, che si trova all'interno del prefab **ViveSR** nel game object **DualCamera** come mostrato precedentemente in figura 3.11**,** in particolare è stato modificato il metodo **SetMode,** che permette di impostare il tipo di vista della telecamera. Le lenti della camera, **DualCameraLeft** e **DualCameraRight**, responsabili del rendering dei game object, sono state settate a **true**, mentre **TrackedCameraLeft e TrackedCameraRight** che restituiscono il riferimento della videocamera tracciata rispettivamente a sinistra e destra, sono state settate a **false.** Ciò significa che, se ci troviamo nel mondo virtuale, le telecamere stereo frontali sono sempre attive, il che ci assicura il rendering dinamico del mondo reale, anche se noi non possiamo vedere il mondo reale.

```
DualCameraLeft.enabled = true;
DualCameraRight.enabled = true;
TrackedCameraLeft.gameObject.SetActive(false);
TrackedCameraRight.gameObject.SetActive(false);
```

Fig.3.12 *Modifiche apportate al metodo SetMode*

3.4 3D Reconstruction

Questo modulo consente di ottenere informazioni sulla geometria del mondo reale. È necessario mantenere abilitato il modulo di ricostruzione 3D. Per fare ciò viene utilizzato lo script **ViveSR_RigidReconstructionRenderer** inserito nel prefab **ViveSR** nel game object **RigidReconstructor** come mostrato in figura 3.11**.**

Ci sono alcune operazioni elencate nello script**ViveSR_RigidReconstructionRenderer**

Le diverse modalità di visualizzazione sono:

- **Full Scene Point:** nuvola dei punti completamente ricostruita, è possibile vedere il funzionamento in figura 3.13.
- **Field of View:** punti 3D nel tronco della cornice.
- **Adaptive Mesh:** triangolazione adattabile alla superfice curva.

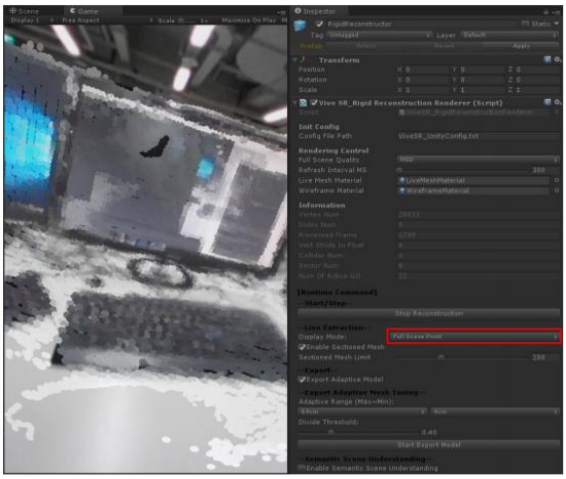

Fig.3.13 *Display settato in modalità Full Scene Point*

È stato deciso di usare **l'Adaptive Mesh**, usando come materiale il **LiveMeshMaterial** e lo **SpatialMappingWireframe**, un mesh scan che cambia colore quando più si è vicini all'oggetto, l'esempio del funzionamento è mostrato in figura 3.14.

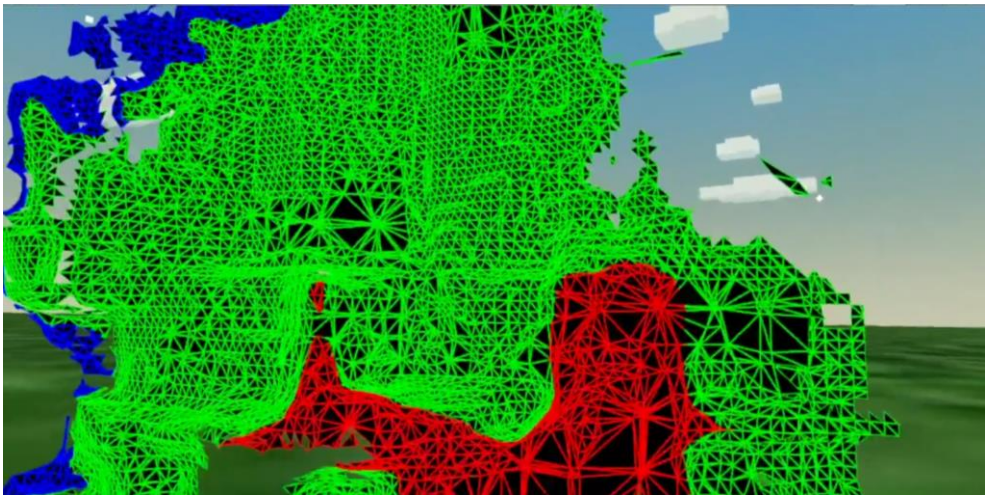

Fig.3.14 *Scan Mesh della scena*

Lo scan mesh genera i collider con l'Adaptive Mesh e perette ai collider di evitare i buchi senza collisione fisica. Inoltre, dà anche informazioni sulla mesh generata, come l'orientamento (verticale, orizzontale e obliqui), la forma (convessa, rettileo) e l'aria (approssimativa). Per fare questo vengono utilizzati due script, e sono: **ViveSR_RigidReconstruction** e **ViveSR_RigidReconstructionRenderer.**

RigidReconstruction si occupa della ricostruzione globale della scena, utilizza metodi come **IsScaning** che restituisce **true** se si sta scannerizzando la scena corrente, **IsExportingMesh** che restituisce **true** se si sta esportando il modello

corrente. **ExportModel** che inizia ad esportare il modello e lo salva con il nome specificato nella cartella **Recons3DAsset,** il contenuto della cartella mostrato mostrata in figura 3.15**.**

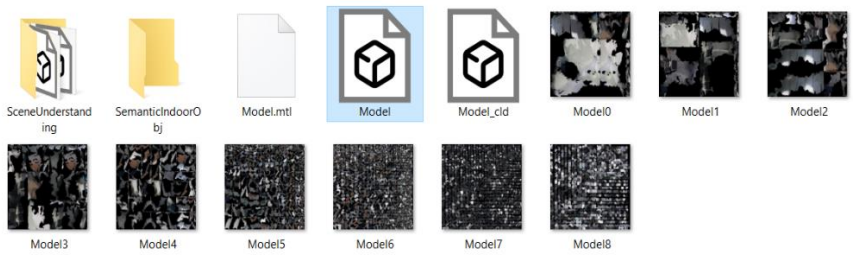

Fig.3.15 *Catella Recons3DAsset*

RigidReconstructionRenderer invece gestisce le funzioni di ricostruzione, ovvero ne gestisce il ciclo di vita. In particolar modo vengono usati due attributi **LiveMeshDisplayMode** che imposta la modalità di visualizzazione dell'estrazione live e si trova nello script **ViveSR_Enums** e **Instance** che restituisce l'istanza di questa classe singleton.

3.5 Semantic Segmentation

La Semantic Segmentation ha il compito di classificare ognuno dei pixel in un'immagine in una classe. Ogni oggetto della stessa classe avrà una etichetta diversa. Il passo successivo della Semantic Segmentation è la rilevazione, che fornisce non solo le classi ma anche informazioni aggiuntive della posizione spaziale di tali classi. L'algoritmo del Semantic Segmentation è situato nello script **Sample9_SemanticSegmentation**, che si trova all'interno del game object **Sample9_SemanticSegmentation**, la cui struttura è mostrata in figura 3.16.

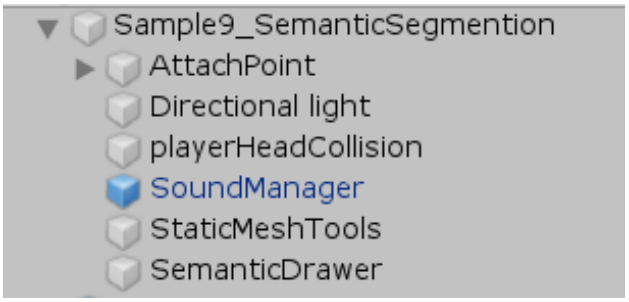

Fig.3.16 *Game object Sample9_SemanticSegmentation*

Analizziamo lo script più nel dettaglio:

- **Awake (figura 3.17):** utilizzato per inizializzare qualsiasi variabile all'inizio dell'esecuzione, viene chiamato solo una volta durante il ciclo di vista dell'istanza dello script. Sono stati definiti due oggetti che fanno riferimento due script diversi: **ViveSR_Experience_StaticMesh** e **ViveSR_Experience_SemanticDrawer.**

```
private void Awake()
{
    StaticMeshScript = FindObjectOfType<ViveSR_Experience_StaticMesh>();
    SemanticDrawer = FindObjectOfType<ViveSR_Experience_SemanticDrawer>();
}
```

Fig.3.17 *Metodo Awake*

- **Start (figura 3.18):** viene chiamato quando lo script è abilitato poco prima che uno dei metodi di aggiornamento viene chiamato per la prima volta. Viene chiamato **l'instance** di **CheckHandStatus** che controlla se le telecamere sono abilitate e poi chiama il metodo **Scanning**.

```
private void Start()
{
    ViveSR_Experience.instance.CheckHandStatus(() =>
    {
        ViveSR_RigidReconstructionRenderer.LiveMeshDisplayMode = ReconstructionDisplayMode.ADAPTIVE_MESH;
        Scanning();
    });
}
```

Fig.3.18 *Metodo Start*

- **Scanning (figura 3.19):** questo metodo fa partire lo scanner della scena, prima controlla che lo scanner sia spento e che non ci siano mesh e modelli in esportazione, poi elimina gli oggetti trovati in una precedente scannerizzazione e infine parte lo scanner.

```
void Scanning()
{
    if (!ViveSR_RigidReconstruction.IsScanning && !StaticMeshScript.ModelIsLoading && !StaticMeshScript.SemanticMeshIsLoading)
    {
        SemanticDrawer.DestroyAllObjects();
        if (StaticMeshScript.CheckModelLoaded()) StaticMeshScript.LoadMesh(false);
        StaticMeshScript.ActivateSemanticMesh(false);
        StaticMeshScript.SetScanning(true);
        StaticMeshScript.SetSegmentation(true);
    }
}
```

Fig.3.19 *Metodo Scanning*

- **Update (figura 3.20):** è l'aggiornamento che avviene ad ogni frame. Per far stoppare lo scanner e per far continuare l'algoritmo, è stato costruito un countdown, che dopo un certo tempo dato dall'utente nell'inspector, come mostrato in figura 3.21, viene invocato il metodo **Saving** che fa stoppare lo scanner.

```
private void Update()
{
    timeLeft -= Time.deltaTime;

    if (timeLeft < 0)
    {
        Saving();
    }
}
```

Fig.3.20 *Metodo Update*

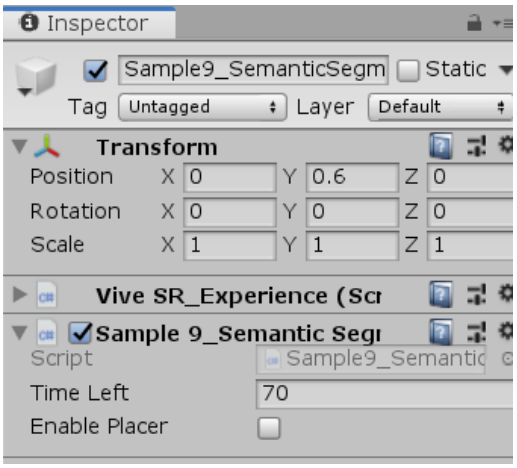

Fig.3.21 *Inserimento secondi per il countdown*

- **Saving (figura 3.22):** il metodo viene invocato da **Update**, controlla se lo scanner è in esecuzione; se è in esecuzione, con il metodo **SetSegmentation** stoppa lo scanner e con il metodo **SetAllCustomSceneUnderstadingConfig** imposta la configurazione di compressione personalizzate per tutti gli stessi parametri, in questo caso è stato scelto il numero dieci. In fine, dopo aver esportato le Mesh e i Modelli

con i metodi **Export** (che sono stati spiegati precedentemente) invoca il metodo **CheckMesh**.

```
void Saving()
{
    if (ViveSR_RigidReconstruction.IsScanning)
    {

        StaticMeshScript.UnloadSemanticMesh();
        ViveSR_SceneUnderstanding.SetAllCustomSceneUnderstandingConfig(10, true);
        StaticMeshScript.SetSegmentation(false);

        StaticMeshScript.ExportSemanticMesh(UpdateSegmentationPercentage,
            () =>
            {
                StaticMeshScript.ExportModel(UpdateModelPercentage,
                 () =>
                 {
                    //Debug.Log("Mesh Saved");
                    CheckMesh();
                 }
                 );
            }
        );

    }

}
```

***Fig.3.22** Metodo Saving*

- **CheckMesh (figura 3.23):** invocato dal metodo **Saving**, si occupa di controllare se i modelli delle mesh esistono o meno. Infatti, il metodo fa un controllo: se non si sta scannerizzando, se esistono dei modelli e se i modelli non sono ancora stati caricati, chiama il metodo **SwitchMode**, ne discuteremo più avanti, e carica la Mesh con il metodo **StaticMeshScript**.**LoadMesh,** che controlla se i file dei modelli esistono o meno. In caso affermativo invoca il metodo **LoadSemanticMesh** e attiva la collisione del game object con **SetActive.**

```
void CheckMesh()
{

    if (!ViveSR_RigidReconstruction.IsScanning && StaticMeshScript.CheckModelExist() && !StaticMeshScript.CheckModelLoaded())
    {
        SwitchMode();
        StaticMeshScript.LoadMesh(
            true,
            () =>
            { // step 1
                //Debug.Log( "Loading Scene...");
            },
            () =>
            { // step 2
                if (StaticMeshScript.CheckSemanticMeshDirExist())
                {
                    LoadSemanticMesh();
                    StaticMeshScript.collisionMesh.SetActive(false);
                }
                else
                {
                  // Debug.Log("No Object is Found.\nPlease Rescan!");
                }
            }
        );
    }

}
```

Fig.3.23 *Metodo CheckMesh*

- **LoadSemanticMesh (figura 3.24):** metodo invocato da **CheckMesh**, permette il caricamento della scena di gioco tramite **StaticMeshScript.LoadSemanticMesh**, viene disattivata la collisione del Game object con **setActive** e poi vengono chiamati i metodi **SwitchMode** e **SwitchActionMode.**

```
private void LoadSemanticMesh()
{
    StaticMeshScript.LoadSemanticMesh(
        () =>
        {
          // Debug.Log("Loading Objects...");
        },
        () =>
        {
            StaticMeshScript.collisionMesh.SetActive(false);

          // Debug.Log("Mesh Loaded!");
            SwitchMode();
            SwitchActionMode(ActionMode.SemanticObjectControl);
        }
    );
}
```

Fig.3.24 *Metodo LoadSemantcMesh*

- **SwitchMode (figura 3.26):** è un metodo che permette di definire la variabile **ActionMode**, è una variabile di tipo enum definita all'inizio dello script, la definizione di questa variabile è mostrata in figura 3.25. Il valore calcolato viene passato al metodo **SwitchActionMode**

```
enum ActionMode
{
    MeshControl,
    SemanticObjectControl,
    MaxNum
}
```

Fig.3.25 *Metodo CheckMesh*

```
public void SwitchMode()
{
    if (!ViveSR_RigidReconstruction.IsExportingMesh && !ViveSR_RigidReconstruction.IsScanning)
    {
        int mode_int = (int)actionMode;
        ActionMode mode = (ActionMode)((++mode_int) % ActionModeNum);
        SwitchActionMode(mode);
    }

}
```

Fig.3.26 *Metodo SwitchMode*

- **SwitchActionMode (figura 3.27):** a questo metodo viene passato un valore di tipo **ActionMode**, e fa un controllo, se questo valore è uguale a zero allora **SemanticDrawer** verrà posto a **false**, altrimenti **true**. In entrambi i casi viene invocato **SemanticObjOperation**.

```
void SwitchActionMode(ActionMode mode)
{
    actionMode = mode;
    if (mode == ActionMode.MeshControl) // 0
    {
        SemanticDrawer.enabled = false;
        SemanticObjOperation();
        StaticMeshScript.SwitchShowCollider(ShowMode.None);

    }
    else if (mode == ActionMode.SemanticObjectControl) // 1
    {
        SemanticDrawer.enabled = true;
        SemanticObjOperation();
        StaticMeshScript.SwitchShowCollider(ShowMode.None);

    }
}
```

Fig.3.27 *Metodo SwitchActionMode*

- **SemanticObjOperation (figura 3.28):** tramite ShowAllObjects permette di disegnare nella scena tutti gli oggetti.

```
private void SemanticObjOperation()
{
    SemanticDrawer.ShowAllObjects();
    Debug.Log("Show all");
}
```

Fig.3.28 *Metodo SemanticObjOperation*

3.6 Creazione oggetti

Fino ad ora abbiamo parlato di come funziona l'algoritmo, quindi di come vengono create le mesh cube, come vengono esportati i modelli, come viene modificata la telecamera e di come funziona lo scan. Una fase importante di questo progetto, l'ultima ma non meno importante, è quella della creazione degli oggetti 3D e del loro posizionamento all'interno della scena al posto dei mesh cube.

Per parlare di questo, riprendiamo lo script analizzato nel capitolo 3 paragrafo 3.1, ovvero **ViveSR_SceneUnderstanding**. Come detto in precedenza, è stato aggiunto un **BoxCollider** che è servito per calcolare la posizione effettiva del mesh cube.

Ma perché non usare la posizione dei vertici, degli indici e dei triangoli del mesh cube?

La risposta è semplice: la posizione del mesh cube è una posizione fittizia. Nel senso che, il cubo viene inserito nella posizione giusta all'interno della scena, quindi viene posizionato all'interno nella scena virtuale dove è situato il rispettivo oggetto reale nella scena reale, però le coordinate dell'asse x, y e z sono sempre zero: il vettore del mesh cube (x, y, z) è (0 ,0, 0). Per adempiere a questo problema, è stato inserito un BoxCollider e ne è stata calcolata la posizione.

Analizziamo del dettaglio la porzione di codice in figura 3.30: è stato aggiunto al mesh cube il **BoxCollider** con **AddComponent.** Sono stati definiti poi due Vector3:

- **position** a cui viene assegnata il valore **center**, ovvero il centro del BoxCollider, misurato nello spazio locale dell'oggetto;
- **scale** a cui viene assegnata la **size**, ovvero la dimensione della scatola, misurata nello spazio locale dell'oggetto.

```
BoxCollider box = Obj.AddComponent<BoxCollider>();
Vector3 position = box.center;
Vector3 scale = box.size;
instanziatePrefab.InitializationPrefabs(tagIdElement.tag, position, scale, Obj);
```

Fig.3.30 *Inserimento del BoxCollider*

Infine, viene invocato il metodo **InitializationPrefabs** che si trova nello script **InstanziatePrefabs** dove stono stati passati, il tag dell'oggetto trovato, position, scale e l'oggetto.

Lo script **InstaziatePrefabs** è stato inserito all'interno di un empty object che si trova all'interno della scena, come è possibile vedere in figura 3.31. Questa strategia è stata attuata per facilitare la creazione degli oggetti.

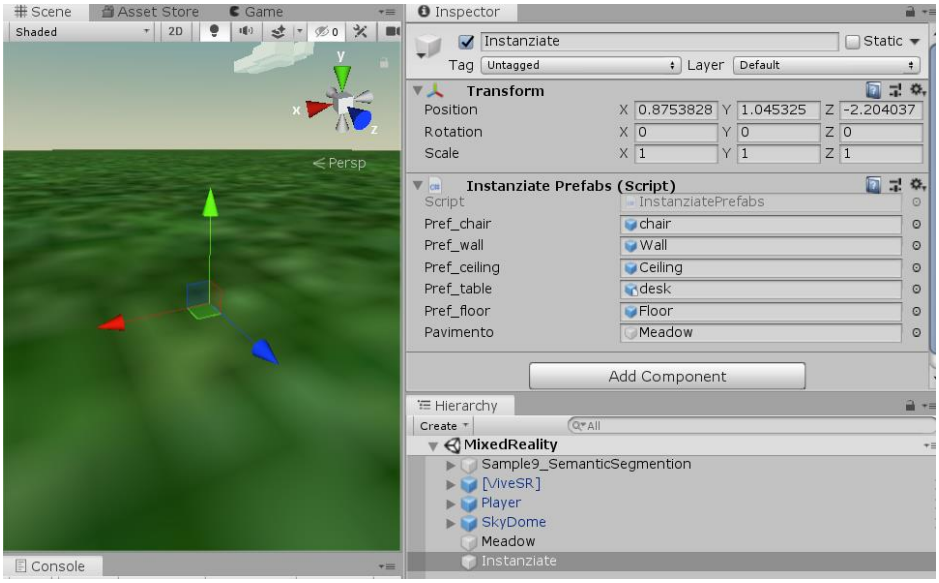

Fig.3.31 *Vista di Instanziate nell'inspector*

In questo script sono stati resi pubblici i game object, come in figura 3.32, ed è bastato fare drag e drop degli oggetti 3D nelle corrispettive celle.

```
public GameObject pref_chair, pref_wall, pref_ceiling, pref_table, pref_floor;
public GameObject meadow;
```

Fig.3.33 *definizione dei prefab all'iterno dello script Instanziate*

Con il tag dell'oggetto passato è stato controllato il tipo di oggetto passato, perché in base all'oggetto trovato, viene assegnato il prefab corrispondente. È stato preso in figura 3.34, il caso della sedia. È stato fatto il controllo se l'element (il tag dell'oggetto) è "Chair" è stata settata la posizione della sedia con le coordinate di position, poi è stato creato il prefab nella scena, è stata data la grandezza da scale e

infine è diventato un oggetto figlio del mesh cube corrispondente. Nell'immagine 3.35 sono presenti nella scena tutti gli oggetti che sono stati scannerizzati.

```
if (element.Equals("Chair"))
{

    position = new Vector3(position.x, position.y, position.z);
    pref_chair = Instantiate(pref_chair, position, new Quaternion(0, 0, 0, 0));
    pref_chair.transform.localScale = scale;
    pref_chair.transform.SetParent(obj.transform);

}
```

Fig.3.34 *esempio di creazione dell'oggetto*

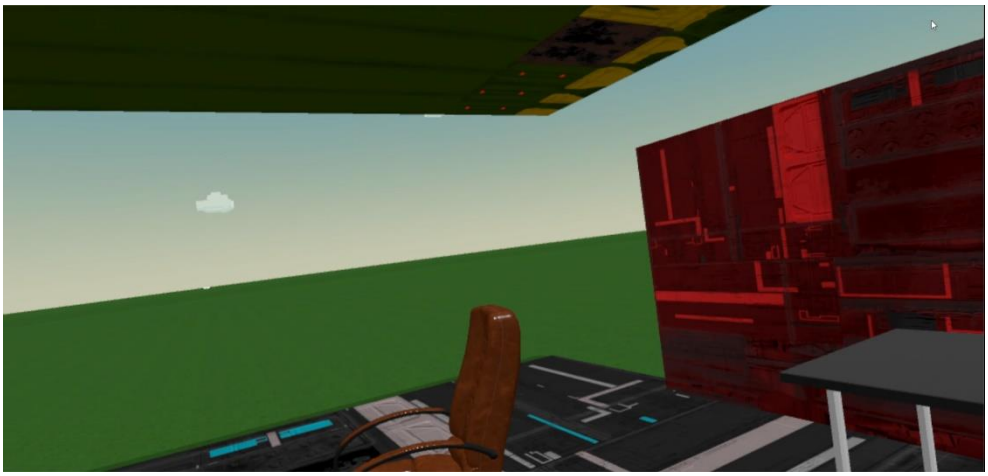

Fig.3.35 *tutto gli oggetti scannerizzati e posizionati nella scena*

A seguire sono mostrati i modelli degli oggetti 3D utilizzati:

- Soffitto (figura 3.36);
- Parete (figura 3.37);
- Pavimento (figura 3.38);

- Tavolo (figura 3.39);
- Sedia (figura 3.40).

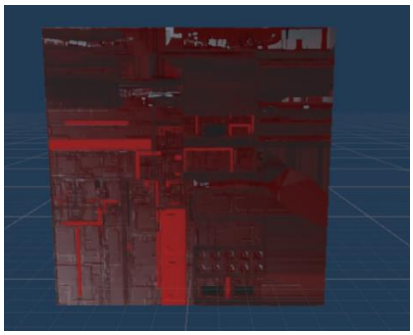

Fig.3.36 *modello 3D del soffitto*

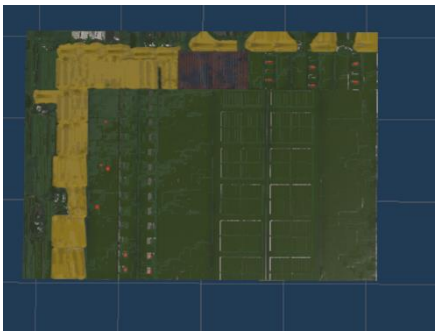

Fig.3.37 *modello 3D della parete*

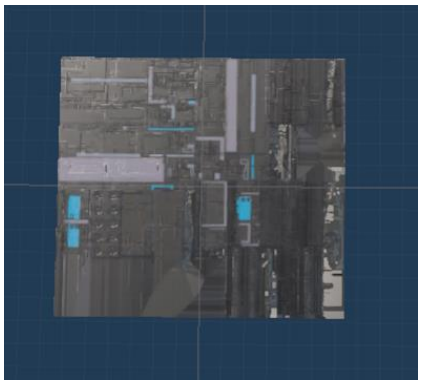

Fig.3.38 *modello 3D del pavimento*

Fig.3.39 *modello 3D del tavolo*

Fig.3.40 *modello 3D della sedia*

3.6 Caso Sedia

A seguire, un esempio di come viene vista la sedia nella realtà (figura 3.41) e di come viene vista nel mondo virtuale (figura 3.42) dopo essere stata scannerizzata.

Fig.3.41 *sedia nel mondo reale*

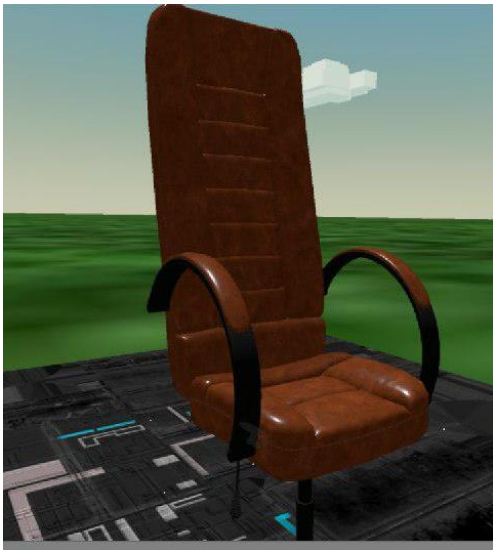

Fig.3.41 *sedia nel mondo virtuale*

4 Sviluppi futuri

Questo progetto si basa sul riconoscimento di oggetti base (quali sedia, tavolo, parete, soffitto e pavimento), ma si potrebbero fare delle migliorie all'algoritmo, come:

- Migliorare l'algoritmo di scannerizzazione rendendolo più preciso;
- Possibilità di poter riconoscere ulteriori oggetti (ad esempio la televisione, finestra, etc.);
- Possibilità di riconoscere le persone;
- Possibilità di riconoscere oggetti in movimento.

Inoltre, grazie alle nuove tecnologie, può essere migliorato il tracking della scena, ad esempio, in figura 4.1 è mostrato il nuovo **HTC Vive Cosmos,** che grazie alle sei telecamere esterne presenti sul visore, ottiene un accurato tracciamento tramite un ampio campo visivo.

Fig.4.1 *HTC Vive Cosmos*

5 Conclusioni

Essendo la Mixed Reality una tecnologia giovane, ancora non si è riusciti ad usare tutte le sue potenzialità. Infatti, ad oggi, non tutto ancora funziona perfettamente. Si basti pensare alla prova di HoloLens dove gli oggetti hanno avuto degli sfarfallii, purtroppo non è semplice collocare l'oggetto nella scena. Comunque, le potenzialità della MR sono smisurate e soprattutto applicabili in tantissimi ambiti; per questa ragione, tutti i brand che producono visori e dispositivi, a partire da Oculus, HTC e Google, stanno investendo cifre immense in questa tecnologia. È un dato di fatto che tutti i visori standalone attualmente o prossimamente in commercio offriranno funzionalità di mixed reality.

Incredibile ma vero, c'è molto in serbo per il futuro che la MR ha da offrire. Ecco alcuni dei punti salienti:

- Le proiezioni olografiche e un display virtuale interattivo guadagneranno maggiore attenzione;
- MR offrirà strutture di potenza di elaborazione convenienti e competenze di riconoscimento di immagini / visioni;
- I big data e il cloud computing svolgeranno un ruolo significativo nelle tecnologie MR avanzate;
- La realtà mista cambierà la comunicazione tra uomo e computer e la porterà ad un nuovo livello;
- L'MR diventerà una delle industrie globali più ambite nei prossimi cinque / sette anni.

La Mixed Reality ha reso reale ciò che anni prima abbiamo potuto ammirare solo nei film come in figura 5.1 e che potevamo immaginare come una realtà molto lontana dalla nostra. Ad oggi, non è più solo fantascienza da film.

Fig5.1 *Esempio di Mixed Reality nel film Avatar*

Ringraziamenti

Ho sempre pensato che il successo di una persona sia tale anche grazie all'aiuto degli altri. L'allunaggio non è avvenuto solo per mano di uomo e Hyrule non è stata salvata solo grazie all'aiuto di un eroe.

Per questo mio successo devo ringraziare soprattutto la mia famiglia, i miei genitori: Grazie Babbo, per aver scommesso su di me, per avermi dato la possibilità di studiare ciò che amo, senza di te non avrei mai potuto iniziare questa carriera. Grazie Mamma, sei stata la prima a credere in me ancora prima di me stessa, mi hai sempre spronato a dare il massimo. Grazie a mio fratello Nello, per essermi stato sempre a fianco, fin da piccoli mi hai sempre protetto e sostenuto. Grazie alla mia piccola Nana, per il tuo affetto smisurato. Grazie perché senza di voi non sarei mai arrivata fino in fondo. Questa tesi la dedico a voi che siete la mia famiglia, vi amerò per sempre.

Grazie Checco, per essere il mio punto di riferimento, grazie perché hai sempre creduto profondamente in me dicendomi sempre che ce l'avrei fatta. Abbiamo affrontato tante sfide, ma le abbiamo sempre superate, nonostante le difficoltà, mi hai aiutato a crescere, e devo dirti grazie anche perché se oggi sono così forte e sicura di me è anche grazie a te, sei una persona fantastica!

Grazie Gaia, la mia amica di sempre, per essermi stata sempre a fianco.

Grazie Marianna, mia preziosa compagna in questi tre anni di avventure (e disavventure), ne abbiamo passate tante insieme, sei diventata una sorella!

Un grosso grazie a tutti i cari amici che in modi diversi, attraverso parole, gesti, messaggi, risate, camminate e chiacchierate mi hanno incoraggiata! Grazie Ben, Ale, Carmen, Peppe, Goku e Danilo! Vi voglio davvero bene ragazzi.

Grazie alla mia numerosa famiglia: le mie nonne, nonna Antonietta e nonna Teresa, i miei zii e i miei cugini, in particolar modo a Imma, Lucia, Pupo e Pieri, grazie per avermi supportato (e sopportato).

Grazie al mio piccolo fan club di sostenitori, siete parte della mia famiglia: Mena, Luigi, Annalisa e la piccola Martina. Grazie per tutte le belle parole che mi avete rivolto.

Grazie ai miei colleghi universitari con i quali ho condiviso tutti questi giorni e grazie a chi in questi anni, anche solo con un sorriso mi è stato vicino.

Grazie al professore Abate per avermi dato la possibilità di lavorare su un argomento che amo tanto e Grazie alla dottoressa Paola Barra per avermi seguito in questi mesi di lavoro con costanza, disponibilità e tanta professionalità.

Grazie alle mie stelle polari, Nonna Immacolata, Nonno Aniello e Nonno Luigi, so che da lassù mi proteggete sempre.

Mi hanno sempre etichettato come una "strana", mi hanno detto che non avrei concluso niente, che ero un'illusa, per questo, tre anni fa, quando entrai per la prima volta in F8, ero una ragazza che non avrebbe mai scommesso un euro su sé stessa. In questi tre anni sono cresciuta, ho affrontato tante sfide senza mai avere paura del risultato, ho imparato a difendere le mie scelte, ho capito che non devo mai smettere di inseguire i miei sogni grandi come l'America, e mai lo farò, li inseguirò sempre con **passione**, **determinazione**, e **coraggio**. Oggi ho raggiunto il punto di partenza per iniziare a fare ciò che quella ragazzina tra i banchi di scuola immaginava di diventare un giorno.

Che il gioco abbia inizio!

Giuliana

Bibliografia

[1] DigiCapital. (2018). Tratto da https://www.perkinscoie.com/images/content /1/8/v2/187785/2018-VR-AR-Survey-Digital.pdf

[2] Beat Game. Tratto da https://beatsaber.com/

[3] Niantic. Tratto da https://pokemongolive.com/it/

[4] Garner Customer. Tratto da https://www.Gartner.com/en/conferences/emea/ customer-experience-uk

[5] Milgram.P, & Kishino.F. (1994). Taxonomy of mixed reality visual displays. IEICE TRANSACTIONS on Information and Systems.

[6] Microsoft. Tratto da Microsoft: https://www.microsoft.com/it-it/hololens

[7] Acer. Tratto da https://www.acer.com/ac/it/IT/content/windows-mixed-reality-home

[8] Vive. Tratto da https://www.vive.com/us/

[9] Vive. Tratto da Developer Vive: https://developer.vive.com/resources/ knowledgebase/intro-vive-srworks-sdk/

[10] Zite.Tratto da GitHub: https://github.com/Valve Software/steamvr_unity _plugin /releases/tag/1.2.3

[11] Valve. Steam. Tratto da https://store.steampowered.com/?l=italian

[12] Unity Asset Store. (s.d.). Tratto da https://assetstore.unity.com/