Alma Mater Studiorum · Università di Bologna
Campus di Cesena

SCUOLA DI SCIENZE
Corso di Laurea in Ingegneria e Scienze informatiche

Integrazione della tecnologia di realtà aumentata Meta 2 con il framework MiRAgE e sviluppo di una libreria di oggetti aumentati

Relazione finale in:

SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore:
Prof. ALESSANDRO RICCI

Tesi di laurea di:
DANIELE ROSSI

Co-relatore:
Ing. ANGELO CROATTI

Prima Sessione di Laurea

Anno accademico: 2017-2018

PAROLE CHIAVE

Augmented Reality
Mixed Reality
Augmented World
MiRAgE
Meta 2
Unity

Ai miei genitori, Marina e Paolino, che per permettermi di essere qui hanno conosciuto il volto di Fatica e Sacrificio.

Indice

Introduzione

La velocità dell'evoluzione tecnologica ha consegnato il futuro nelle nostre mani: i dispositivi negli anni sono diventati sempre più piccoli tanto da poter essere, addirittura, indossati; lo sviluppo della rete internet ha permesso il collegamento di questi indipendentemente dalla loro posizione. Se fino a qualche anno fa il concetto di tecnologia era strettamente legato al computer ora non è più così, un chiaro esempio di ciò è l'Internet of Things (Iot).

Grazie all'IoT i più disparati oggetti quotidiani possono essere collegati e comunicare tra di loro; ciò è una vera e propria rivoluzione che ha cambiato radicalmente le nostre vite, iniettando all'interno delle cose una sorta d'intelligenza in modo che queste possano essere di supporto all'essere umano.

È evidente come la tecnologia sia diventata pervasiva e la massima espressione di questo fenomeno è rappresentata dalle tecnologie di Realtà Virtuale (VR), Realtà Aumentata (AR) e Realtà Mixata (MR). La nascita e lo sviluppo della VR ha introdotto un nuovo modo di interagire con le informazioni, dando vita a software che creano, nell'utilizzatore, un vero senso di presenza all'interno di mondi digitali. AR e MR hanno introdotto un'ulteriore rivoluzione creando uno strato digitale per aumentare il mondo fisico consentendo all'essere umano di interagire con oggetti virtuali come se fossero reali. Queste due tecnologie sono una delle più grandi invenzioni dalla nascita dei personal computer perché cambiano il modo di vedere l'informazione digitale.

Pur utilizzando la tecnologia tutti i giorni non la sentiamo "vicina", ma percepiamo un senso di distaccamento dagli altri quando, ad esempio, utilizziamo il nostro smartphone; effettivamente è così, i dispositivi con cui fruiamo dell'informazione digitale ci separano dal mondo fisico facendo sembrare quest'ultima qualcosa di cattivo e pericoloso. Ecco perché AR e MR stanno rivoluzionando il nostro mondo: consentono un utilizzo dell'informazione digitale senza separare le persone; ciò apre nuovi e interessanti scenari sopratutto nell'ambito del lavoro cooperativo assistito da computer (CSCW).

In questo contesto si inserisce MiRAgE un framework per lo sviluppo di mondi aumentati, basati su agenti, in cui oggetti computazionali vengo-

no posizionati nel mondo fisico ed eventualmente visualizzati mediante un ologramma.

Da qui nasce l'obiettivo cooperativo della tesi: integrare un dispositivo di visualizzazione MR, Meta 2, con MiRAgE che utilizza principalmente Vuforia per la visualizzazione olografica.
Inoltre, per semplificare il lavoro di sviluppatori che vogliono realizzare AW e rendere l'implementazione il più indipendente possibile dall'engine utilizzato per la creazione e la gestione degli ologrammi, la tesi si prefigge lo scopo di studiare e creare una libreria di oggetti utili al CSCW basato su MR.
In particolare, la creazione della libreria, prevederà lo sviluppo di uno degli oggetti individuati; quest'ultimo progetto verrà svolto singolarmente dallo studente.

Sommario

Il presente elaborato si sviluppa in quattro capitoli.

I primi due capitoli forniscono le nozioni tecniche e concettuali utili per lo sviluppo della parte progettuale. Gli ultimi due sono dedicati alla descrizione dell'analisi e della progettazione dei sistemi sviluppati.

Capitolo 1 Nel primo capitolo, vengono introdotti i concetti chiave di Realtà Virtuale (VR), Realtà Aumentata (AR), Realtà Mixata (MR) e le principali differenze tra questi. Inoltre, viene proposta una panoramica sul software e i dispositivi hardware per la AR.

Capitolo 2 Il secondo capitolo definisce cosa si intende per Augmented World (AW) e il modello concettuale di quest'ultimo. Tale capitolo descrive, anche, la struttura di MiRAgE : un framework per la creazione di AW.

Capitolo 3 Nel terzo capitolo si descrive la tecnologia utilizzata per la parte progettuale cooperativa entrando nel merito di quest'ultima. In particolare, si analizza il problema introducendo i concetti matematici e tecnici per affrontare poi la parte di progettazione in cui viene descritta la struttura del sistema implementato.

Capitolo 4 L'ultimo capitolo entra nel merito del progetto svolto individualmente. Si introducono i limiti delle GUI e i vantaggi delle interfacce basate su AR effettuando poi una piccola analisi su oggetti utili in queste interfacce in ambito lavorativo. La parte finale è legata alla progettazione e allo sviluppo di due oggetti, post-it e lavagna, che compongono questa libreria.

Capitolo 1

Augmented Reality e Mixed Reality

Un'ampia definizione di AR è la seguente: "*tecnologia in grado di aumentare il feedback naturale dell'utilizzatore con segnali simulati*", in maniera più specifica la AR può essere definita come: "*una forma di realtà virtuale in cui il dispositivo di visualizzazione indossato dall'utente consente di vedere il mondo fisico*" [1].

Nella VR l'utente è completamente immerso in un mondo virtuale che può imitare o meno le proprietà del mondo fisico: nel mondo sintetico le leggi fisiche che regolano la gravità, il tempo e la materia potrebbero non essere rispettate. La AR, invece, consente all'utente di vedere il mondo reale, ma gli oggetti fisici coesistono con quelli virtuali.

Per evitare di limitare la AR alla specifica tecnologia di visualizzazione è possibile definire un sistema AR in base alle 3 caratteristiche che deve avere [2][3]:

1. Combinare il reale e il virtuale.
2. Interazione in tempo reale.
3. Registrazione con il mondo fisico.

Fino ad ora abbiamo parlato di AR e VR come se fossero due concetti separati, in realtà, Paul Milgram e Fumio Kishino in un paper del 1994 [1], collocano le due tecnologie agli estremi di ciò che viene definito come Reality-Virtuality (RV) continuum [1.1]. L'estremo di sinistra definisce un ambiente composto solamente da oggetti reali ed include qualsiasi cosa possa essere osservata da una persona nel mondo.
L'estremo di destra definisce un ambiente composto esclusivamente da oggetti virtuali (ambiente VR). Con virtualità aumentata (AV) si fa riferimento

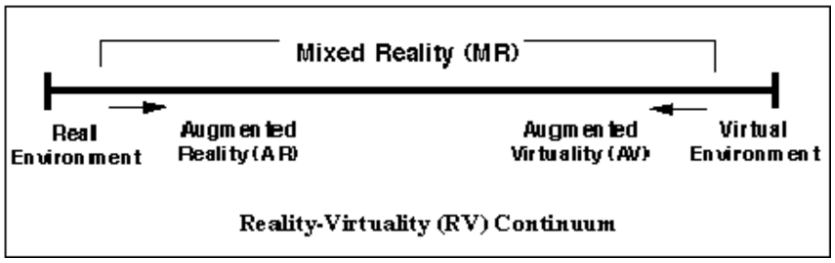

Figura 1.1: Rappresentazione semplificata del RV continuum.

ad un ambiente completamente immersivo in cui, in certe situazioni, vengono introdotti oggetti reali per manipolare qualcosa appartenente alla scena virtuale; ad esempio, possono essere introdotte le mani dell'utente per modellare azioni di afferramento. La parte situata tra i due estremi può essere definita come Mixed Reality (MR) e rappresenta ambienti in cui oggetti reali e virtuali sono presentati insieme in una singola visualizzazione.

Mentre nella AR gli oggetti virtuali sono in primo piano rispetto al mondo fisico, nella MR gli oggetti virtuali sono integrati con il mondo fisico combinando aspetti della VR e della AR [1.2].

Nella MR viene quindi introdotto il concetto di scannerizzazione dell'ambiente, indispensabile per ottenere informazioni utili al posizionamento degli oggetti digitali.

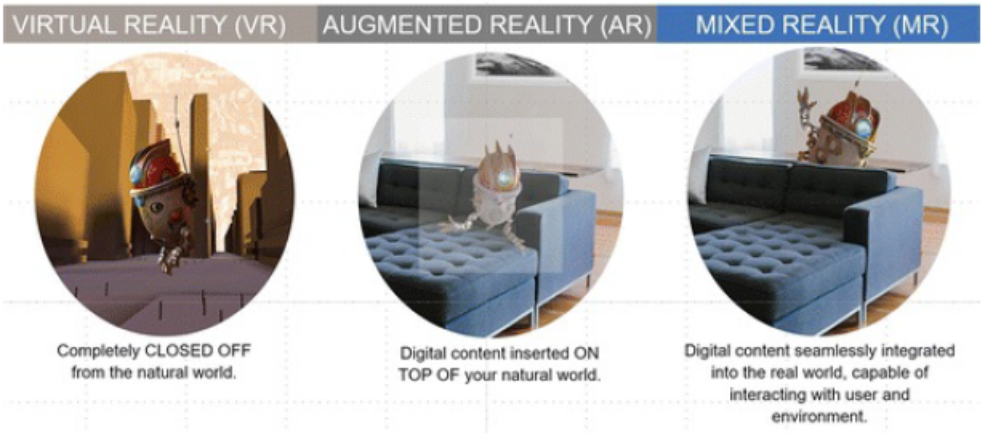

Figura 1.2: Differenza tra VR, AR e MR

1.1 Hardware per la realtà aumentata

Come già accennato nelle pagine precedenti è possibile utilizzare diversi dispositivi per sperimentare la AR.

Ad alto livello, questi ultimi, possono essere classificati in[4]: dispositivi non indossabili e dispositivi indossabili, definiti anche head-mounted displays (HMD) [1.3]. Fanno parte del primo gruppo: dispositivi mobili (ad esempio smartphone), dispositivi stazionari (ad esempio tv o pc) e head-up displays (dispositivo tipicamente utilizzato nelle auto che proietta informazioni circa la navigazione). Il secondo gruppo, il più interessante dal punto di vista della AR, è caratterizzato dai dispositivi dedicati: elmetti e occhiali smart.

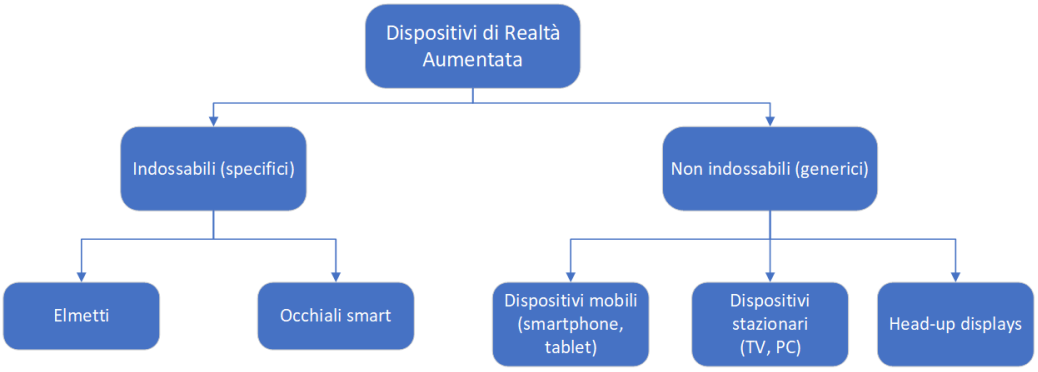

Figura 1.3: Tassonomia dispositivi AR

1.1.1 Elmetti

Un device può essere classificato come elmetto se copre la testa dell'utente, le orecchie e gran parte del viso. Un esempio è l'elmetto smart di Daqri, [1.4] caschetto antinfortunistico, composto da plastica, fibra di carbonio, alluminio e dotato di visiera protettiva antigraffio.

Il dispositivo è pensato come dotazione per operatori specializzati e tecnici; le applicazioni incluse aiutano l'utente a svolgere lavori complessi come riconoscere tubature e ottenere informazioni in tempo reale sulla loro pressione.

Figura 1.4: elmetto smart di Daqri

1.1.2 Occhiali smart

Gli occhiali smart, che possono essere categorizzati in consumer o commercial, sono un dispositivo più compatto dell'elmetto e tipicamente integrano: lenti, camera, casse per l'audio e vari sensori.

1.1.2.1 Consumer

Questi dispositivi hanno un piccolo sistema operativo real time (RTOS) che comunica via bluetooth low energy (BTLE) con qualsiasi smartphone o tablet e utilizza i sensori a bordo di quest'ultimo come GPS, giroscopio e accelerometro.

Questi sono alimentati da una batteria e sono molto compatti tanto da sembrare un classico occhiale da vista [1.5].

Sono pensati per una presentazione 2D di contenuti come ad esempio, riquadri contenenti informazioni utili all'utente: ricezione di una chiamata, informazioni di navigazione, risultati di computazioni...

(a) Vuzix M100 (b) GlassUP's UNO

Figura 1.5: Esempio di occhiali smart consumer

1.1.2.2 Commercial

Questi dispositivi, pensati per uso commerciale, scientifico e ingegneristico, sono più potenti di quelli precedenti e tipicamente sono connessi ad un pc via cavo. Esempi di visori che rientrano in questa categoria sono: Meta 2, Microsoft Hololens e i futuri Magic Leap One [1.6].

Rispetto ai precedenti sono in grado di fornire un'esperienza molto più immersiva: gli ologrammi sono 3D e, grazie ad una vasta gamma di sensori, sono integrati con il mondo fisico; ad esempio, gli ologrammi possono essere nascosti da oggetti fisici grazie alla funzionalità di depth occlusion fornita dall'occhiale.

(a) Microsoft Hololens (b) Magic Leap One

Figura 1.6: Esempio di occhiali smart commercial

1.2 Software per realtà aumentata

Diverse compagnie offrono strumenti per aiutare gli sviluppatori nella creazione di applicazioni basate su AR.

Di seguito si entrerà nel merito delle principali infrastrutture software e dei principali strumenti di sviluppo.

1.2.1 Piattaforme software

1.2.1.1 Vuforia

Una delle più famose infrastrutture software per AR è Vuforia usata in più di 50000 applicazioni; la sua popolarità è aumentata quando, nel 2016, ha siglato un accordo di partnership con Unity per l'integrazione dell'infrastruttura con quest'ultimo.

Vuforia permette il posizionamento di contenuti digitali tridimensionali nel mondo fisico; il cuore di tutto è rappresentato dal *Vuforia Engine* che fornisce delle funzionalità di computer vision per il riconoscimento degli oggetti e per la ricostruzione dell'ambiente.

Le funzionalità principali di riconoscimento offerte da Vuforia sono:

- **VuMark:** non è altro che un marker personalizzabile [1.7]; questa caratteristica, che può sembrare banale, permette di avere un marker unico ed espressivo per ogni oggetto. Un altro vantaggio che offre il VuMark è la possibilità di realizzare design dal look and feel affascinante, caratteristica essenziale per applicazioni consumer.

- **Riconoscimento immagini:** rappresenta la possibilità di riconoscere immagini anziché marker [1.8]. A differenza del classico fiducial marker, le immagini da riconoscere non sono contraddistinte da regioni bianche e nere; Vuforia, infatti, è in grado di riconoscere le caratteristiche peculiari dell'immagine comparando queste con quelle di alcune immagini target contenute in un database.

- **Riconoscimento oggetti:** permette di rilevare oggetti tridimensionali fornendone una rappresentazione digitale delle caratteristiche e della geometria [1.9]. Il cuore del riconoscimento oggetti è il *Vuforia Object Scanner* che permette di ottenere un file contenente i dati relativi alle caratteristiche dell'oggetto e caricarlo sul gestore dei target.
  Per far si che ciò funzioni, l'oggetto deve essere opaco, rigido e avere un buon contrasto.

Figura 1.7: esempi di VMark

Figura 1.8: Riconoscimento immagini di Vuforia

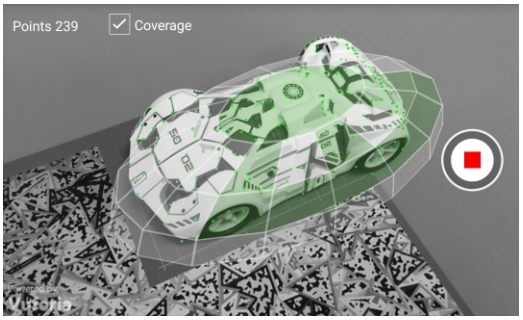

Figura 1.9: Vuforia object tracking

1.2.1.2 ARToolKit

ARToolKit è una libreria open-source che permette di sviluppare applicazioni AR.

Per capire quale sia il punto di vista dell'utente, ARToolKit usa algoritmi di computer vision.

La libreria calcola la posizione della camera e il suo orientamento rispetto ad un marker in tempo reale; questo permette di realizzare facilmente una vasta gamma di applicazioni AR.

Alcune caratteristiche di ARTollkit sono:

- calcolo di posizione e orientamento della camera;
- possibilità di usare qualsiasi fiducial marker pattern;
- semplicità di calibrazione della camera;
- supporto per Windows, Linux, MacOS e SGI Irix;
- completamente open-source;

1.2.1.3 Augment

Augment è una piattaforma dell'omonima azienda leader nella visualizzazione di prodotti commerciali basata su AR e nasce con l'idea di eliminare le ambiguità sull'aspetto del prodotto create dalle foto descrittive: il prodotto ha un colore leggermente diverso da quello in foto, è troppo grande...

Le tecnologie offerte dall'azienda sono:

- **Augment manager:** consente di caricare pubblicamente o privatamente dei modelli 3D e dei marker personalizzati e gestirli. Nel caso in cui i modelli fossero resi pubblici sarebbe possibile visualizzare statistiche legate alla visualizzazione di questi: quando e dove sono stati visualizzati.
- **Augment desktop:** permette di creare i propri modelli 3D e di aggiungergli texture, materiali e animazioni.
- **Augment app:** compatibile sia con iOS che con Android offre la possibilità di visualizzare i modelli 3D dei vari prodotti nel mondo reale.
- **Augment sdk:** permette di integrare la visualizzazione di prodotti commerciali basata su AR nel proprio sito di e-commerce.

1.2.2 Strumenti di sviluppo

1.2.2.1 OpenCV

OpenCV (Open Source Computer Vision Library) è una libreria open-source pensata per fornire un'infrastruttura comune per applicazioni basate su computer vision e accelerare l'uso di macchine percettive per la realizzazione di prodotti commerciali.

La libreria ha più di 2500 algoritmi ottimizzati tra cui: riconoscimento volti, identificazione oggetti, estrazione di modelli 3D, riconoscimento marker...

OpenCV è stata progettata per essere multi-piattaforma e per questo è stata scritta in C; dalla versione 2.0 gli algoritmi della libreria sono stati scritti in C++ e sono stati sviluppati wrapper per linguaggi come Python e Java aumentando in questo modo la community degli utilizzatori.

Nel 2010 è stato aggiunto ad OpenCV un modulo per l'accelerazione GPU sviluppato utilizzando CUDA; in questo modo è possibile eseguire algoritmi sofisticati in tempo reale e ottenere un output ad elevata risoluzione consumando meno energia[1.10].

OpenCV è eseguibile anche in ambiente mobile; infatti nel 2010 è stato aggiunto il supporto per Android e nel 2012 quello per iOS.

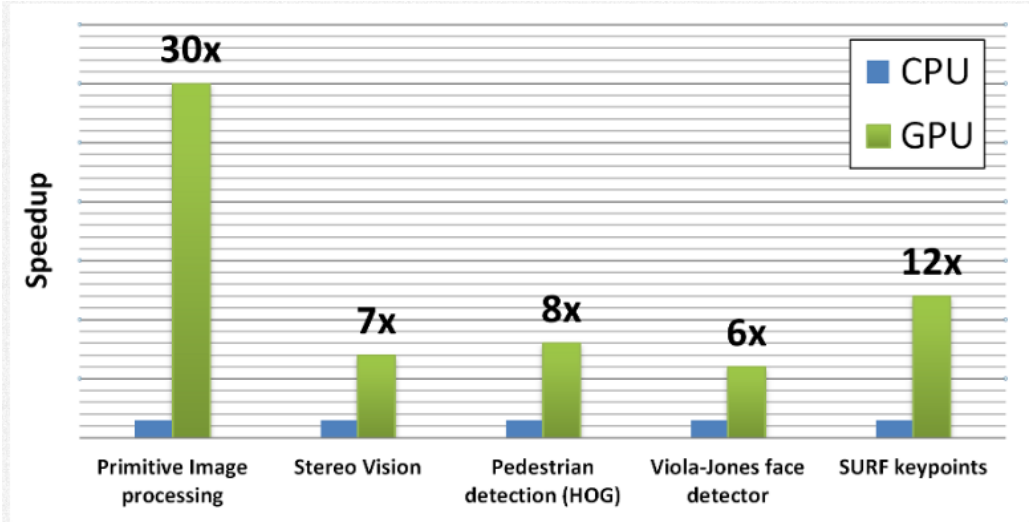

Figura 1.10: Tesla C2050 GPU VS Core i5-760 2.8Ghz, SSE, TBB performance

1.2.2.2 Unity3D

Unity (comunemente chiamato Unity3D) è un game engine e ambiente di sviluppo integrato (IDE) per la creazione di media interattivi, in particolare videogiochi, ed è famoso per la sua prototipazione rapida e per la vasta gamma di dispositivi supportati.

Di seguito vengono introdotte e descritte alcune delle componenti base di Unity.

Assets Un asset è una rappresentazione di un qualsiasi oggetto che può essere utilizzato in un progetto Unity; esso può provenire da un file esterno (non creato all'interno di Unity), come ad esempio un modello 3D, oppure può essere creato internamente a Unity, come ad esempio un controller di animazioni.

Gli assets sono i blocchi base dei progetti realizzati con Unity ed è per questo che in ogni progetto tutti i file sono contenuti nella cartella Assets (figlia di Project) [1.11].

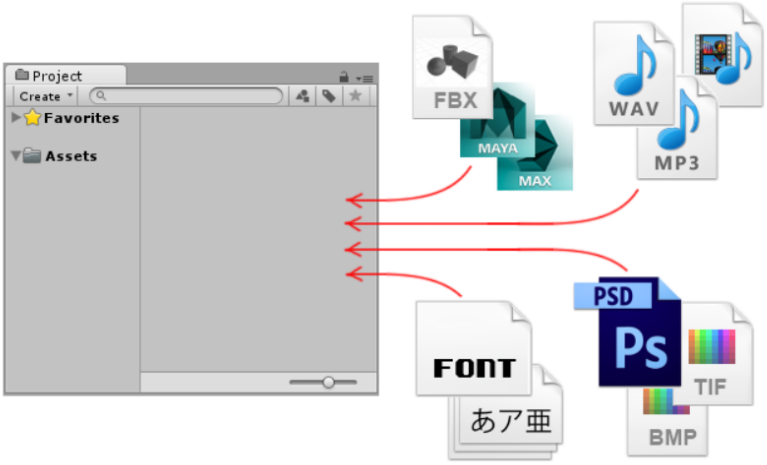

Figura 1.11: Assets in unity

Scenes Una scena è un livello unico contenente l'ambiente o il menù di gioco.

All'interno di una scena è possibile piazzare i propri oggetti di gioco che verranno poi renderizzati dalla camera.

Ogni scena può essere salvata e riaperta in un momento successivo.

Game Objects Quando un asset viene usato in una scena diventa un Game Object (GO).

Ogni Game Object ha almeno un componente di default denominato transform che indica la sua posizione, rotazione e scala descritte in coordinate X, Y, Z.

Per poter assegnare ad un GO delle proprietà, in modo che questo possa svolgere un determinato ruolo, è necessario aggiungergli dei componenti.

È possibile pensare ad un GO come un recipiente da cucina vuoto ed ai componenti come gli ingredienti per realizzare la ricetta desiderata [1.12].

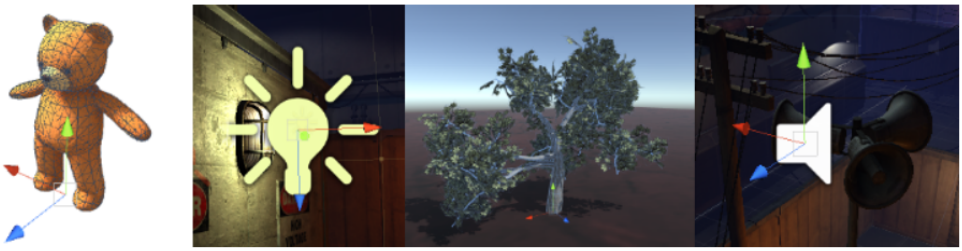

Figura 1.12: Esempi di Game Object dotati di componenti differenti

Components Un componente può svolgere ruoli diversi: definire un comportamento, definire l'aspetto di un oggetto [1.12] o influenzare altri aspetti di quest'ultimo nel gioco.

Molti componenti base utilizzati nella produzione di videogiochi sono già integrati in Unity, tra questi vi è il rigidbody, che indica a Unity di utilizzare l'engine fisico per l'oggetto ed i componenti che trasformano un oggetto in una lampada, telecamera, emettitore di particelle e molto altro.

Ogni componente è dotato di particolari parametri che consentono di controllare l'effetto che quest'ultimo ha sull'oggetto a cui appartiene [1.13].

Un esempio di componente è il *Material* che permette di definire il colore di un oggetto, la trasparenza, la lucentezza...

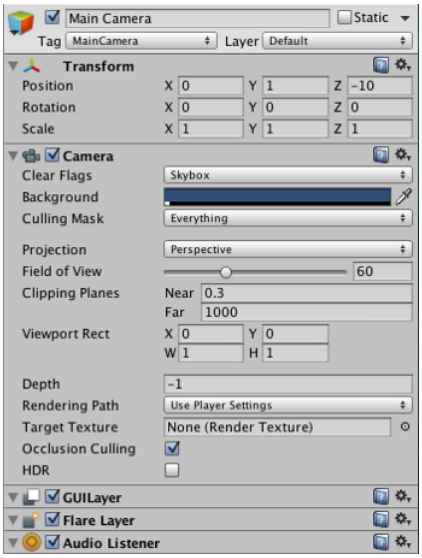

Figura 1.13: Esempi di component attaccati ad un oggetto e relativi parametri

Scripts Per dotare un oggetto di funzionalità non implementate nelle componenti base sarà necessario scrivere un apposito script.

In Unity è possibile scrivere script utilizzando C# o Unityscript, linguaggio di programmazione derivato da Javascript, progettato specificatamente per essere utilizzato in Unity; in aggiunta a questi due è possibile utilizzare altri linguaggi .NET se danno la possibilità di compilare una DLL compatibile.

Il flusso di controllo di Unity è basato su un ciclo ad eventi al verificarsi dei quali vengono chiamate diverse funzioni contenute nella classe *MonoBehavior* da cui ogni script deriva.

Alla sua creazione ogni script è composto dalle due funzioni base della classe MonoBehavior: *Update()* che contiene il codice che verrà eseguito ad ogni frame e *Start()* che viene chiamato una sola volta e contiene le inizializzazioni [1.14].

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PlayerMovement : MonoBehaviour {
6
7     // Use this for initialization
8     void Start () {
9
10    }
11
12    // Update is called once per frame
13    void Update () {
14
15    }
16 }
```

Figura 1.14: Struttura di uno script in Unityscript alla creazione

Prefabs I prefab consentono di salvare completamente un oggetto includendo i suoi componenti e la configurazione di questi.
Un prefab funziona come un template da cui è possibile creare nuove istanze dell'oggetto in una scena.
Ogni modifica effettuata ad un prefab viene immediatamente riflessa a tutte le istanze prodotte partendo dallo stesso; in alternativa, è possibile modificare esclusivamente le proprietà di una specifica istanza.

Capitolo 2

Augmented World

2.1 Introduzione

Grazie al recente sviluppo delle tecnologie ed a componenti hardware prestanti, che permettono ai dispositivi mobili e wearable di sfruttare meglio il web delle cose, è possibile creare ambienti intelligenti integrando la realtà con un'intelligenza artificiale in grado di elaborare i dati provenienti da una rete di sensori.

Questo scenario può essere arricchito dalla AR e MR grazie alle quali è possibile aumentare il mondo con informazioni digitali che possono essere percepite utilizzando appositi device.

L'evoluzione della rete, il cloud e l'IoT consentono di sviluppare spazi smart di grandi dimensioni capaci di ricoprire intere città.

Tutto questo implica lo sviluppo di applicazioni, eseguite a livello di città, mediante opportuno hardware e software distribuito, in grado non solo di estendere le capacità dell'essere umano, ma anche di influenzare le sue decisioni e i suoi piani.

Una delle sfide più grosse è riuscire a definire un modello concettuale di base per descrivere questi spazi; la visione di un Mirror World [5] presenta un possibile modello per fare ciò.

Nell'ottica del Mirror World gli ambienti smart sono modellati come una città digitale, che ha la stessa forma della città fisica con cui è accoppiata, popolata da organizzazioni di agenti capaci di realizzare comportamenti complessi.

Il concetto di specchio interviene quando le cose fisiche, che possono essere percepite dagli essere umani nel mondo fisico, hanno una controparte digitale nel Mirror World, in questo modo possono essere osservate e modificate dagli agenti. Vice versa, un'entità nel Mirror World con cui gli agenti possono

interagire, può prendere forma nel mondo fisico, ad esempio attraverso l'AR, ed essere osservata dagli esseri umani.

Tutto questo implica una forma di accoppiamento per cui un'azione su un oggetto fisico ha un effetto anche sulla corrispondente entità virtuale e può essere osservato dagli agenti; similmente, un'azione di un agente su un'entità virtuale può avere effetto sull'oggetto fisico corrispondente ed essere percepito dagli esseri umani.

Per comprendere meglio questi concetti è possibile osservare l'esempio riportato in figura [2.1] che rappresenta la struttura di Augmented World (AW) [6] ispirato al modello Mirror World.

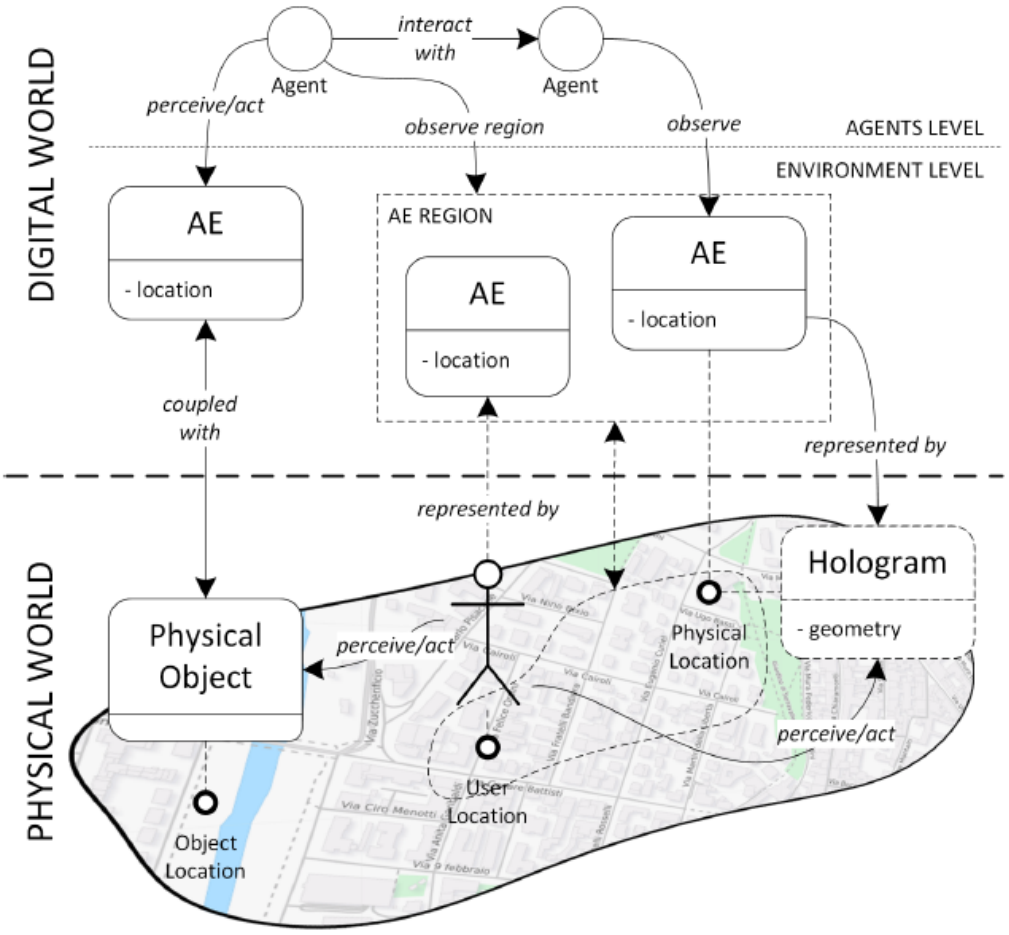

Figura 2.1: Modello AW

Dalla figura è possibile notare la distinzione tra mondo fisico e mondo ri-

flesso e come questi sono collegati tra loro mediante entità aumentate (AE). In particolare, le AE possono essere associate ad oggetti fisici, tra cui anche esseri umani, fornendo una rappresentazione e una descrizione di questi all'interno del mondo digitale. Nel mondo fisico le AE sono rappresentate da ologrammi che gli esseri umani possono vedere e con cui possono interagire.

Inoltre è possibile definire dei confini spaziali, denominati regioni, costituiti da oggetti reali all'interno del mondo fisico e da AE all'interno del mondo digitale.

Nel mondo digitale viene definito un ulteriore livello concettuale, oltre a quello dell'ambiente, appartenente agli agenti. Questi possono osservare, interagire e percepire determinate AE e/o osservare determinate regioni in modo indipendente o cooperativo.

2.2 Modello concettuale di un AW

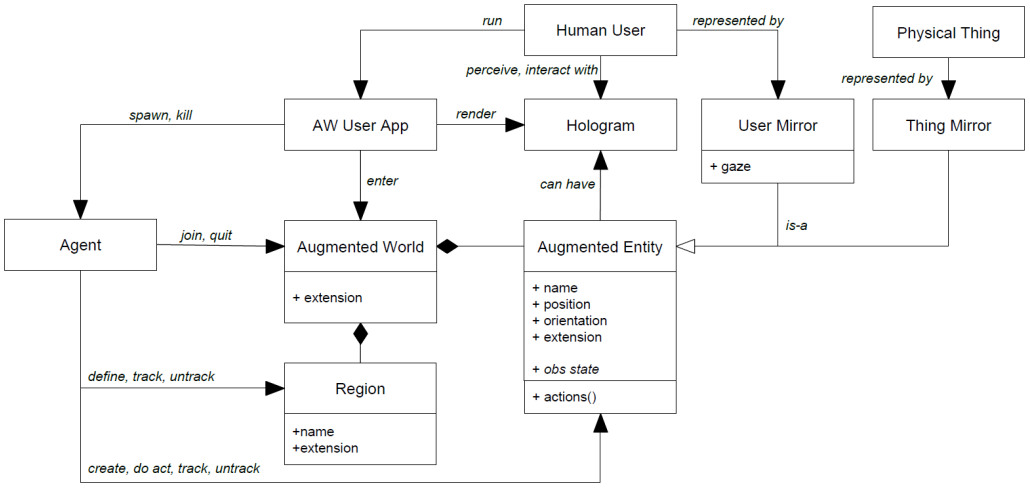

Figura 2.2: Modello concettuale AW

Il concetto principale di AW è formalizzato nel diagramma UML mostrato in figura [2.2].

Un Augmented World è un mondo computazionale mappato su alcune specifiche regioni fisiche.

Un AW è composto da un insieme di entità aumentate che rappresentano gli oggetti computazionali ed hanno una specifica posizione, orientazione ed estensione nel mondo fisico (attributi *position, orientation ed extension*) specificate in relazione alla regione di appartenenza e al sistema di riferimento definito dall'AW.

Le AE sono le componenti base dell'ambiente applicativo basato su agenti; un'AE espone uno stato osservabile (attributo *obs state*), in termini di un insieme di proprietà osservabili che possono cambiare in qualsiasi momento, e un' interfaccia (operazioni *actions()*) che definisce un insieme di azioni. Le AE possono essere dinamicamente create e posizionate dagli agenti e possono essere mosse da questi ultimi in punti differenti dell'AW.

Per poter lavorare all'interno di un AW, un agente deve entrarci (procedura di *join*) ed instaurare una sessione di lavoro. Una volta entrato, un agente può agire sulle varie AE utilizzando le azioni esposte dalle interfacce di queste ultime e osservarle (*track*) per percepire il loro stato e il verificarsi di determinati eventi.

Oltre ad osservare una specifica AE, gli agenti possono osservare regioni all'interno dell'AW; non appena un'AE entra o esce da una regione osservata

da un agente, quest'ultimo percepisce un evento contenente l'informazione dell'AE che lo ha generato.

Il collegamento con la AR è fornito dagli ologrammi: un'AE può avere associato un ologramma, quest'ultimo è la rappresentazione, 3D o 2D, dell'AE nel mondo fisico che la rende percepibile dagli umani.

Il modello dell'ologramma dipende dallo stato della corrispondente AE ed è aggiornato ogni qual volta questa ne subisca un cambio.

La vista dell'ologramma dipende dal suo modello e dal dispositivo AR utilizzato; indipendentemente dalla rappresentazione, l'ologramma è il componente che permette l'interazione con l'utente.

Un utente inizia una sessione all'interno di un AW tramite un'applicazione, *AW UserApp*, che crea un agente in grado di agire sull'AW.

Dal punto di vista dell'interfaccia grafica, l'AW UserApp è responsabile della creazione di una visualizzazione AR coerente con la posizione e l'orientamento dell'utente. Inoltre, questa è la parte che si occupa di rilevare gli input dell'utente (sguardi, gesti e comandi vocali) che possono anche riguardare specifici ologrammi (afferrare un ologramma, guardare un ologramma).

Un ologramma è, quindi, una fonte osservabile di eventi che possono essere percepiti dagli agenti che osservano l'AE corrispondente.

All'interno dell'AW possono esserci più esseri umani che utilizzano, possibilmente, AWUserApp differenti.

Come detto in precedenza, alcune AE possono essere usate per rappresentare oggetti fisici (che fanno parte dell'ambiente fisico) all'interno dell'AW; in questo modo, lo stato dell'AE rappresenta il modello dello stato fisico degli oggetti nel mondo reale e può essere osservato dagli agenti (concetto *Thing Mirror* nello schema).

Tra gli oggetti fisici che possono essere accoppiati con le AE vi è anche il corpo umano dell'utente, ciò è rappresentato nello schema concettuale dalla classe *User Mirror*, che come *Thing Mirror*, è una specializzazione di AE.

L'accoppiamento tra oggetti fisici e AE può funzionare anche in direzione opposta: a seguito di un'azione su un'AE può essere utile avere un effetto su un oggetto fisico utilizzando appositi attuatori; in questo caso un agente può avere un effetto su un oggetto fisico agendo sulla corrispettiva AE.

2.3 Un framework per AW: MiRAgE

MiRAgE (**M**ixed **R**eality based **Aug**mented **E**nvironments)[6] è un framework per lo sviluppo di AW basato strettamente sul modello concettuale mostrato in precedenza [2.2] e fornisce un insieme di funzionalità:

- **Mondi condivisi:** il framework permette a più umani di "immergersi" nello stesso mondo, condividendo e possibilmente interagendo con le stesse entità aumentate.
- **Arricchimento delle entità reali e virtuali:** il framework si basa sul concetto di arricchire il mondo fisico con elementi computazionali che possono essere visualizzati mediante AR/MR; questo concetto è distante dalla semplice definizione di queste due tecnologie.
- **Sistema aperto:** il framework rende possibile lo sviluppo di sistemi AR/MR basati su agenti in cui questi possono entrare ed uscire dinamicamente dal mondo aumentato e possono cambiare la struttura di quest'ultimo, in termini di entità/ologrammi virtuali, a runtime.
- **Tecnologie di programmazione agenti eterogenee:** il design del framework permette di sviluppare gli agenti che lavoreranno all'interno dell'AW utilizzando differenti tecnologie di programmazione agenti (ad esempio: ASTRA, Jason, JaCaMo).
- **Supporto a tecnologie eterogenee:** il design del framework permette di utilizzare diverse tecnologie AR sia software (Unity3D, Vuforia...) che hardware (Meta2, Hololens...); inoltre è integrabile con tecnologie e stack Internet of Things e Web of Things.
- **Generalità:** il framework punta ad essere sufficientemente generale da supportare lo sviluppo di mondi aumentati per qualsiasi dominio applicativo includendo sia scenari indoor (stanza, casa...) sia outdoor.

Il framework è stato progettato con l'idea di nascondere il più possibile i dettagli implementativi e di funzionare con le tecnologie di AR/MR. In questo modo, lo sviluppatore può esclusivamente concentrarsi sulla logica applicativa.

I principali benefici di MiRAgE sono:

- Possibilità di progettare template per AE (definizione di proprietà osservabili e azioni per ogni AE).
- Possibilità di progettare i modelli degli ologrammi o utilizzare modelli già fatti.

- Possibilità di progettare la logica applicativa per agenti che popolano l'AW.
- Possibilità di progettare la logica applicativa legata alle interazioni dell'essere umano con l'AW.

2.3.1 Componenti architetturali

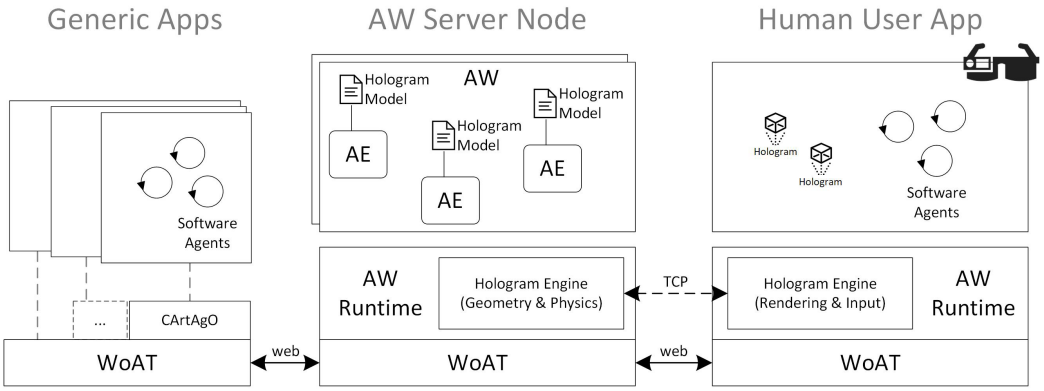

Figura 2.3: Architettura logica di MiRAgE

Da un punto di vista strutturale, è possibile suddividere l' architettura logica di MiRAgE [2.3] in tre componenti principali: AW-Runtime, Hologram Engine, WoAT layer.

2.3.1.1 AW-Runtime

Fornisce l'infrastruttura per eseguire istanze di AW e gestisce l'esecuzione di AE fornendo un'interfaccia comune agli agenti per osservare ed interagire con esse.

Il runtime è progettato per essere eseguito su un nodo server, possibilmente distribuito su un'architettura cloud, in grado di ospitare uno o più AW.

2.3.1.2 Hologram Engine

Supporta la visualizzazione di ologrammi e mantiene aggiornata la loro geometria in tutti i dispositivi indossabili considerando anche gli input degli utenti sugli ologrammi.

Il compito principale dell' HE è quello di mantenere le proprietà e la geometria degli ologrammi aggiornata e consistente su tutte le applicazioni utente che vogliono interagire con l'AW.

L'HE è in esecuzione sia sul nodo server sia su ogni device utente; queste due istanze sono in grado di comunicare direttamente utilizzando un canale TCP. Lato server l'HE fornisce un supporto per la progettazione della geometria degli ologrammi e l'aggiornamento di questa. Lato utente fornisce tutte le funzionalità necessarie per la visualizzazione degli ologrammi, la gestione della fisica (ad esempio, collisione tra due ologrammi) e la gestione degli input e delle interazioni con gli ologrammi.

2.3.1.3 WoAT layer

WoAT (**W**eb **o**f **A**ugmented **T**hings) garantisce piena interoperabilità a livello applicativo; per fare ciò e per garantire scalabilità e integrazione con il mondo IoT adotta il modello Web of Things proposto nel contesto IoT.

In questo modo l' AW e le AE, dal punto di vista degli agenti, sono risorse raggiungibili via rete che forniscono delle REST API per l'interazione.

Ogni AE è raggiungibile attraverso uno specifico indirizzo (URL), relativo all'URL root dell'AW, e tramite operazioni HTTP GET è possibile ottenere proprietà osservabili mentre tramite HTTP POST è possibile inviare azioni da eseguire sull'AE.

2.3.2 Agenti e primitive

Dal punto di vista degli agenti, un AW è un ambiente applicativo che fornisce un insieme di azioni per poter agire all'interno di esso.

L'insieme di azioni può essere suddiviso in due principali categorie:

- **azioni predefinite:** insieme prestabilito di primitive che forniscono funzioni per entrare/uscire dal mondo; creare,gestire e osservare AE; definire e osservare regioni [2.1].

- **azioni specifiche:** entrando in un AW l'insieme delle possibili azioni è dato da tutte le operazioni offerte dalle interfaccie delle AE. Un agente può utilizzare la primitiva doAct(aeID,op,args) che innesca l'esecuzione di un'operazione (op) sull'AE (aeID). Le azioni possono essere processi a lungo termine, per questo l'esecuzione è delegata all'entità stessa; dal punto di vista dell'agente tale azione è eseguita in modo asincrono.

Primitive	**Descrizione**
joinAW(name, location) : awID	per entrare in un AW esistente, ritorna l'id della sessione di lavoro
quitAW(awID)	per uscire da una sessione di lavoro di un AW
createAE(awID, name, template, args, config) : aeID	per creare una nuova AE in uno specifico AW specificando il suo nome, template, parametri(che dipendono dallo specifico template) e una configurazione iniziale
disposeAE(aeID)	elimina un'AE esistente
trackAE(aeID)	per iniziare ad osservare un'AE esistente
stopTrackingAE(aeID)	per smettere di osservare un'AE esistente
moveAE(aeID, position, orientation)	per modificare, se permesso, la posizione e l'orientazione di un'AE
defineRegion(awID, name, region)	per definire una regione specificando nome e estensione
trackRegion(awID, name)	per iniziare ad osservare una regione
StopTrackingRegion(awID, name)	per smettere di osservare una regione

Tabella 2.1: Primitive MiRAgE .

Capitolo 3

Integrazione MiRAgE con Meta2

3.1 Obiettivo

L'obiettivo di questo progetto è integrare una tecnologia emergente per la visualizzazione di MR, Meta 2, con il framework per la creazione di AW MiRAgE .

Meta 2 offre un'esperienza di visualizzazione particolarmente immersiva grazie all'ampio Field Of View (FOV) ed alla sua vasta gamma di sensori consentendo all'utente di interagire in maniera naturale con gli oggetti virtuali. Proprio per questi motivi l'integrazione con MiRAgE potrebbe aprire molti scenari, soprattutto in ambito lavorativo.

Considerando, ad esempio, quest'ultimo come campo applicativo è possibile estendere l'ambiente di lavoro con entità computazionali che possono essere di supporto all'utente. Questa possibilità si fa ancora più interessante quando più utenti possono utilizzare il mondo aumentato ed interagire tra di loro sfruttando Meta2 e MiRAgE come mezzo per il lavoro cooperativo basato su computer (CSCW). Naturalmente, quest'ultima possibilità risulta molto interessante anche in ambiti applicativi diversi da quello lavorativo come, ad esempio, quello ludico.

Per la realizzazione di un AW multi-utente, però, è necessario che questi ultimi abbiano una visione consistente alla propria prospettiva di ciò che li circonda. Tutti i dispositivi devono "registrarsi" con il sistema di riferimento dell'AW; questa procedura, spesso, impiega i marker come mezzo per identificare l'origine e l'orientazione del sistema di riferimento.

Sfortunatamente, Meta 2 non offre nessun supporto al riconoscimento dei marker; da qui è nata la volontà di realizzare un sistema che permetta al

dispositivo di registrarsi con l'AW.

3.2 Meta 2

Meta 2 [3.1] è un occhiale di tipo commercial per la AR sviluppato dall'azienda californiana Meta.

Questo occhiale, a differenza di Hololens e Magic Leap One, è dotato di cavo e necessita di essere connesso alla scheda video del computer; ciò significa che sono stati pensati per lavorare in uno spazio ridotto.

I Meta 2 offrono una collezione molto vasta di sensori in grado di creare una mappa dell'ambiente e riconoscere le varie gesture che permettono all'utente di interagire con gli oggetti virtuali.

Uno dei punti forti di Meta 2 è il display che presenta un ampio Field of View (FOV) e un'elevata risoluzione.

Figura 3.1: Occhiale Meta 2

3.2.1 Specifiche tecniche

Specifiche del dispositivo:

Risoluzione: 2550x1440
Refresh rate: 60 Hz
Field of view: 90°
Audio: 4 surround speaker
Microfoni: 3 microfoni
Camera: RGB camera frontale 720p
Sensori: IMU 6-assi, array di sensori per interazioni con le mani e tracking posizionale
Cablaggio: HDMI 1.4b per video, USB 3.0 per dati, alimentatore
Peso: 500g

Requisiti minimi computer:

Hardware

Processore: Intel Core Intel 7-4770 or AMD FX 9370, o superiori
RAM: 8GB DDR3
Scheda grafica: NVIDIA GTX 960 o AMD Radeon R9 290.
Output video: HDMI 1.4b
Connessione dati: USB 3.0
Hard disk: 2 GB di spazio libero

Software

Motore grafico: Unity 5.6 o superiore
Sistema Operativo: Windows 10 (64-bit) (in futuro supporto per MacOs)

3.2.2 Meta SDK 2.7.0 per Unity

Meta mette a disposizione un SDK per Unity per tutti gli sviluppatori che vogliono realizzare applicazioni AR basate sul suddetto engine.

All'interno dell'SDK sono contenuti alcuni prefab che implementano funzionalità di base dell'occhiale e alcune scene di esempio per prendere familiarità con il dispositivo.

3.2.2.1 Sensori e camera

Meta camera rig Prefab che si occupa del rendering e tracking all'interno dell'ambiente Unity; per funzionare correttamente, la scena deve contenere esattamente un'istanza attiva del prefab.

Per default all'avvio della scena la posizione della camera viene resettata a (0,0,0); per poter posizionare la camera in un punto diverso dall'origine è necessario includere il MetacameraRig all'interno di un altro Game Object.

SLAM Localizer Si occupa della localizzazione e del mapping simultaneo utilizzando esclusivamente i sensori dell'occhiale.

Per fare ciò, combina i dati provenienti dalle camere con quelli provenienti dalle unità di misurazione inerziale (IMUs) che utilizzano accelerometri per misurare l'accelerazione lineare e giroscopi per quanto riguarda la rotazione.

Il visore può lavorare in due modalità differenti:

- **Limited tracking:** utilizza 3 gradi di libertà; gli oggetti sono posizionati tutti alla stessa distanza utilizzando la rotazione fornita dai giroscopi.
- **Full tracking:** utilizza 6 gradi di libertà; gli oggetti vengono posizionati a seconda dei dati provenienti dagli accelerometri e ruotati in base a quelli provenienti dai giroscopi.

Lo script genera diversi eventi che possono essere intercettati dai Game Object iscritti come ad esempio: *OnSlamMappingComplete()* che segnala la fine della procedura di mapping dell'ambiente [3.2].

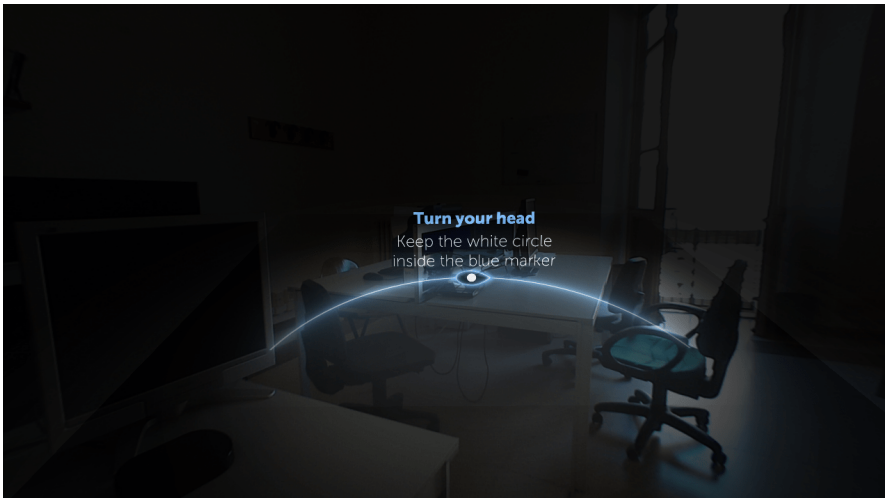

Figura 3.2: Procedura di mapping dell'ambiente

Compositor Si occupa del rendering delle immagini aggiornandole ogni qual volta ottiene informazioni dallo SLAM.

Utilizza tecniche di predizione per ridurre la percezione della latenza del rendering degli oggetti virtuali quando si muove la testa.

Permette anche l'attivazione del depth occlusion [3.3] che fa si che gli oggetti reali, rilevati dal sensore di profondità dei Meta 2, oscurino gli oggetti virtuali che si trovano dietro.

Virtual webcam Permette di visualizzare e registrare ciò che vede l'utente che indossa l'occhiale.

I Meta 2 espongono due stream appartenenti alla webcam:

- **Meta2 webcam + holographic overlay** - si compone sia delle informazioni provenienti dalla camera sia degli oggetti virtuali visualizzati nell'occhiale.
- **Meta2 webcam** - non contiene gli oggetti virtuali, ma solo le informazioni provenienti dalla webcam.

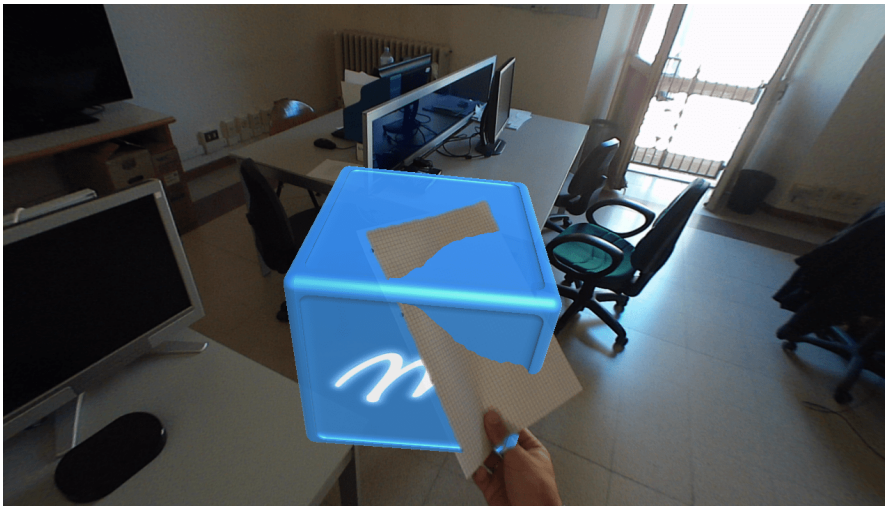

Figura 3.3: Depth occlusion Meta 2

Environment configuration Crea una mesh 3D dell'ambiente fisico [3.4], utilizzando i sensori dei Meta 2, che può essere utilizzata per facilitare le interazioni tra oggetti virtuali e reali. Ad esempio, è possibile utilizzare la ricostruzione di superfici per far rimbalzare una pallina virtuale su un tavolo reale.

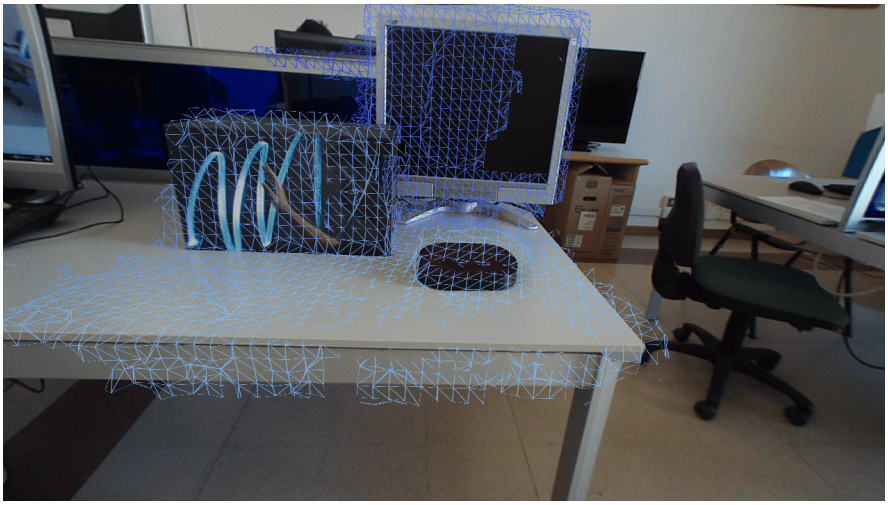

Figura 3.4: Ricostruzione mesh Meta2

Meta locking Consente di bloccare un oggetto relativamente allo sguardo dell'utente; ad esempio è possibile bloccare un oggetto in una porzione del campo visivo dell'utente in modo che possa essere di supporto a quest'ultimo.

3.2.2.2 Interazioni

Per poter interagire con un oggetto virtuale è necessario che questo possieda i componenti rigidbody e collider.

Nella parte sottostante verranno introdotti i diversi tipi d'interazione messi a disposizione dal Meta SDK.

Meta hands Questo prefab fornisce informazioni relative alle mani dell'utente e offre diverse componenti che implementano interazioni base dell'SDK, ad esempio:

- **GrabInteraction** - permette di afferrare e traslare un oggetto.
- **TwoHandGrabRotateInteraction** - permette di traslare e ruotare un oggetto afferrandolo con due mani.
- **TwoHandGrabScaleInteraction** - permette di traslare e scalare un oggetto afferrandolo con due mani.

Vi sono poi altre interazioni supplementari derivate dalla classe base *Interaction*.

Meta mouse Consente al puntatore mouse del computer di essere usato all'interno dell'ambiente AR tridimensionale.

Inserendo il prefab *MetaInputModule* è possibile interagire con gli oggetti virtuali visualizzati utilizzando il mouse del computer.

Per poter interagire con i Game Object è necessario agganciarvi alcuni script:

- **DragRotate** - permette di afferrare e ruotare l'oggetto.
- **DragScale** - permette di afferrare e scalare l'oggetto.
- **DragTranslate** - permette di afferrare e muovere l'oggetto.

Per poter svolgere le diverse funzionalità è possibile assegnare a ciascuna un tasto diverso del mouse.

Meta gaze Permette di interagire con gli oggetti che si trovano al centro della vista dell'utente.

Per funzionare emette dei raycast (raggi) dal centro del *MetaCameraRig* nella direzioni in cui l'utente sta guardando e verifica se questi collidono con qualche oggetto che dovrà essere dotato del componente *Box Collider* per la rilevazione delle collisioni.

Meta canvas Consente una prototipazione rapida di interfacce 2D che supportano interazioni sia *Meta Hands* che *Meta Mouse*.

Per fare ciò utilizza componenti base di Unity, come il canvas, e del Meta SDK.

3.3 AW multi-utente in MiRAgE basati su Meta2

3.3.1 Analisi del problema

Uno dei problemi principali nelle applicazioni AR è quello della registrazione [7]: gli oggetti nel mondo reale e nel mondo virtuale devono essere propriamente allineati, altrimenti viene persa l'illusione che i due mondi coesistano.

In molte applicazioni, sopratutto in ambito medico, è richiesta un registrazione molto accurata in assenza della quale l' applicazione AR non può essere eseguita.

Il problema della registrazione assume ulteriore importanza nell'ambito del CSCW: il requisito base per poter fare lavoro cooperativo utilizzando visori MR è che tutti gli utenti abbiano una visione consistente del mondo mixato in base alla loro posizione.

I metodi di registrazione possono essere classificati nelle seguenti categorie[8]: registrazione basata sui sensori, registrazione basata sui marker e registrazione senza marker.

3.3.1.1 Registrazione basata sui sensori

Questo tipo di registrazione non utilizza sensori visivi per reperire le informazioni utili alla registrazione. I tipi di sensori che possono essere utilizzati sono: GPS, sensori magnetici, sensori ad ultrasuoni...

I vantaggi legati all'utilizzo di tali sensori sono la facilità di implementazione, in quanto questi sono già integrati in molti dispositivi AR e accessibili mediante API, e il basso consumo di energia e di capacità computazionale. Per contro, alcuni sensori possono essere utilizzati solo in particolari ambienti, ad esempio il GPS può essere utilizzato esclusivamente all'aperto, e non consentono di rilevare i cambi di vista dell'utente.

A causa di questi svantaggi la registrazione basata su sensori può essere utilizzata solo in alcuni sistemi AR; il trend principale è quello di utilizzarla in combinazione a tecniche di registrazione che utilizzano sensori visivi per garantire una completa esperienza di AR.

3.3.1.2 Registrazione basata sui marker (vision-based)

I metodi di registrazione basati su sensori visivi sono attualmente i più utilizzati e usano algoritmi di pattern matching per individuare particolari target

(marker in questo caso) posizionati nel mondo fisico e determinare posizione e rotazione di chi li guarda rispetto a questi.

I marker più comuni hanno forma quadrata e presentano un pattern asimmetrico bianco su sfondo nero [3.5]; oltre a questi esistono marker circolari, marker basati su punti, marker innestati...

I vantaggi che derivano dall'utilizzo di marker sono legati al grado di precisione della registrazione (l'errore di posizione è nel grado del centimetro e quello di rotazione in quello del grado) e all'inferiore utilizzo di capacità computazionale rispetto alla tecnica markerless. Per contro, questa tecnica risulta vulnerabile a movimenti bruschi della camera, occlusione del marker e condizioni di scarsa illuminazione.

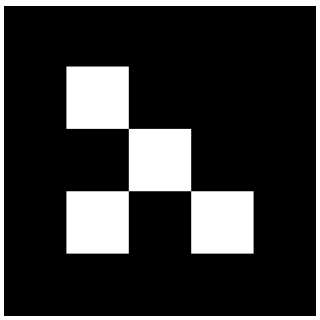

Figura 3.5: Esempio di marker.

3.3.1.3 Registrazione senza marker (vision-based)

Grazie allo sviluppo di algoritmi avanzati per la visione artificiale è possibile utilizzare una registrazione di tipo marker-less che consente di individuare particolari come punti, bordi o spigoli e di utilizzarli come marker per la registrazione di oggetti virtuali.

Esistono due approcci principali per effettuare una registrazione markerless:

- **Model-based tracking:** questi metodi stimano la posizione della camera utilizzando corrispondenze che vengono calcolate comparando le caratteristiche di un frame iniziale e del frame attuale.
- **Move-matching tracking:** questi metodi stimano la posizione della camera in base al movimento di un insieme di caratteristiche tra un frame e l'altro. Per l'implementazione di questi metodi è spesso utilizzata la tecnica SLAM (**S**imultaneous **L**ocalization **A**nd **M**apping) in

grado di generare una mappa 3D dell'ambiente e localizzare l'utente all'interno di questa.

Rispetto ai metodi marker-based questa tecnica permette di lavorare in ambienti sconosciuti e su cui non è necessario apporre marker: le applicazioni AR basate su registrazione marker-less, potenzialmente, possono funzionare in ogni luogo. Per contro, richiede l'impiego di algoritmi avanzati e computazionalmente onerosi e non è in grado di raggiungere la precisione ottenuta mediante l'impiego di marker.

3.3.1.4 Problematiche e considerazioni iniziali

Per poter integrare Meta 2 con MiRAgE è necessario registrare ogni dispositivo Meta con le coordinate dell'AW di MiRAgE .

Mentre Meta 2 segue una filosofia di tipo marker-less e non offre nessun supporto nativo al riconoscimento e alla gestione dei marker, MiRAgE è stato utilizzato, fino ad ora, utilizzando i marker e, in particolare, Vuforia. È quindi necessario sviluppare un sistema di registrazione ad-hoc.

Inoltre, MiRAgE per semplicità sfrutta la funzionalità multiplayer offerta da Unity che fornisce un'interfaccia ad alto livello (HLAPI: High Level API) per la realizzazione di multiplayer in rete.

Meta 2, nativamente, non fornisce nessun supporto a questa funzionalità, per questo motivo è necessario avere in esecuzione su ogni dispositivo un'istanza client, indipendente dalle altre, che comunica con l'AW Server Node. Ogni qualvolta un client esegue una modifica su un'AE questa non deve essere effettuata localmente, ma deve essere notificata al server che ha il compito di propagarla a tutti i client, compreso quello che la eseguita.

3.3.1.5 Trasformazione sistemi di riferimento

Per poter effettuare la registrazione con l' AW di MiRAgE è necessaria una tecnica che permetta di convertire le coordinate espresse in relazione al sistema di riferimento dell'AW in coordinate espresse in relazione al sistema di riferimento del dispositivo e viceversa.

All'interno di uno spazio R^n, n ∈ N è possibile individuare due entità:

- **punto:** possiede un unico attributo che rappresenta la sua posizione rispetto ad un sistema di riferimento.
- **vettore:** possiede due attributi che rappresentano la sua lunghezza e la sua direzione.

Ogni vettore w dello spazio può essere rappresentato univocamente in termini di tre vettori linearmente indipendenti; questi vettori sono usati come base di quello spazio (3.2). In uno spazio vettoriale V i vettori v1, v2, . . . , vk, k ∈ N, si dicono linearmente indipendenti se il solo modo di scrivere il vettore nullo $\mathbf{0_v}$ come loro combinazione lineare è a coefficienti nulli (3.1).

$$0_v = \alpha_1 v_1 + ... + \alpha_k v_k \leftrightarrow \alpha_1 + ... + \alpha_k = 0 \tag{3.1}$$

$$B = \{v_1, v_2, v_3\} \rightarrow w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$$

In forma matriciale:

$$\mathbf{a} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \qquad w = a^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{3.2}$$

Cambio sistemi di coordinate In uno spazio vettoriale, date due basi è possibile esprimere una nei termini dell'altra definendo la matrice M, 3x3, di cambiamento di base (3.3).

$$B_1 = \{v_1, v_2, v_3\} \;\; B_2 = \{u_1, u_2, u_3\}$$

$$\begin{aligned} u_1 &= \sigma_{11} v_1 + \sigma_{12} v_2 + \sigma_{13} v_3 \\ u_2 &= \sigma_{21} v_1 + \sigma_{22} v_2 + \sigma_{23} v_3 \\ u_3 &= \sigma_{31} v_1 + \sigma_{32} v_2 + \sigma_{33} v_3 \end{aligned} \tag{3.3}$$

$$M = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \qquad \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \mathbf{o} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = M^{-1} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

Dato un vettore w è possibile ottenere la sua rappresentazione, w', nell'altro sistema di coordinate usando la matrice M e viceversa (3.4):

$$\begin{gathered} w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 \\ a = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \qquad w = a^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \\ w' = \beta_1 u_1 + \beta_2 u_2 + \beta_3 u_3 \\ b = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \qquad a = M^T b \text{ e } b = (M^T)^{-1} a \end{gathered} \tag{3.4}$$

È importante notare che si sta parlando di vettori e non di punti: questi cambi di base lasciano l'origine immutata, in altre parole, rappresentano solo rotazioni e scalature.

Una base, B, non basta per definire la posizione di un punto nello spazio, occorre anche un punto di riferimento: l'origine [3.6]; per poter effettuare un cambio di sistema di riferimento è necessario eseguire anche un cambio di tale punto.

Sistema di riferimento — Tre vettori linearmente indipendenti

Figura 3.6: Differenza tra sistema di riferimento e tre vettori linearmente indipendenti

Cambio sistemi di riferimento Un sistema di riferimento (frame) necessita, quindi, di un punto di origine P_0 e di una base; in esso è possibile rappresentare univocamente un punto:

$$P = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + P_0 \; con \; \alpha_1, \alpha_2, \alpha_3 \; scalari \tag{3.5}$$

Per poter effettuare trasformazioni di coordinate geometriche in forma matriciale è necessario utilizzare il metodo matematico delle coordinate omogenee.

In questo modo il punto P (3.5) può essere rappresentato come:

$$P = [\alpha_1, \alpha_2, \alpha_3, 1] \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} \tag{3.6}$$

Per poter rappresentare un vettore si sostituisce l'1 nell'equazione (3.6) con uno 0.

Dati due sistemi di riferimento, è possibile esprimere uno nei termini dell'altro utilizzando una matrice M, 4x4, di cambiamento di frame (3.7).

$$S_1 = \{v_1, v_2, v_3, P_0\} \;\; S_2 = \{u_1, u_2, u_3, Q_0\}$$

$$\begin{aligned} u_1 &= \sigma_{11} v_1 + \sigma_{12} v_2 + \sigma_{13} v_3 \\ u_2 &= \sigma_{21} v_1 + \sigma_{22} v_2 + \sigma_{23} v_3 \\ u_3 &= \sigma_{31} v_1 + \sigma_{32} v_2 + \sigma_{33} v_3 \\ Q_0 &= \sigma_{41} v_1 + \sigma_{42} v_2 + \sigma_{43} v_3 + P_0 \end{aligned}$$

$$M = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} & 0 \\ \sigma_{21} & \sigma_{22} & \sigma_{23} & 0 \\ \sigma_{31} & \sigma_{32} & \sigma_{33} & 0 \\ \sigma_{41} & \sigma_{42} & \sigma_{43} & 1 \end{bmatrix} \tag{3.7}$$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix} = M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} \;\; \textbf{e} \;\; \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = M^{-1} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix}$$

Date due rappresentazioni *a,b*, sia di un vettore che di un punto, in coordinate omogenee appartenenti a differenti frame vale:

$$b^T \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix} = b^T M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = a^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = b^T M = a^T \rightarrow a = M^T b \; \textbf{e} \; b = (M^T)^{-1} a \tag{3.8}$$

3.3.2 Progettazione e sviluppo del sistema

3.3.2.1 Processo di registrazione

Idea Per effettuare la registrazione con l'AW di MiRAgE è possibile sfruttare la funzionalità di depth occlusion offerta dai Meta 2 e creare un mirino, posto ad una distanza fissa rispetto agli occhi di chi indossa l'occhiale, da allineare con 3 punti d'interesse collocati precedentemente nel mondo fisico [3.7]. I 3 punti devono essere presi in un ordine specifico:

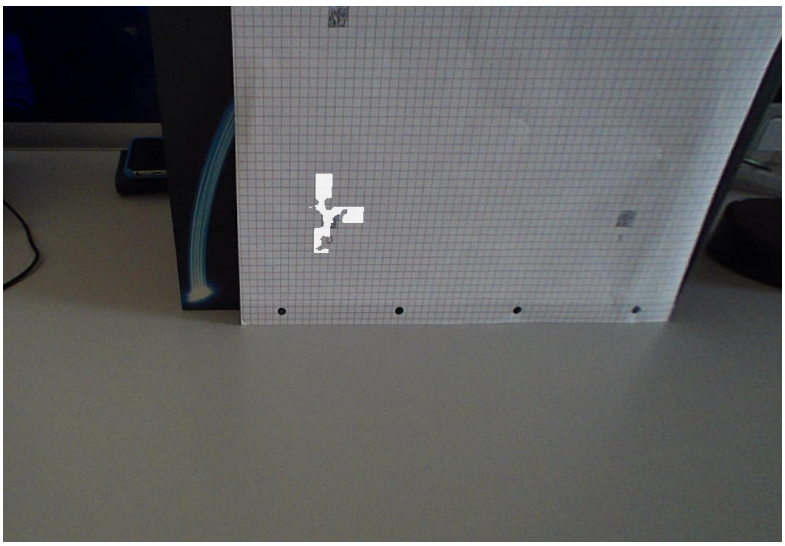

Figura 3.7: Allineamento del mirino ad un punto

1. Origine
2. Punto asse X
3. Punto asse Y

I vettori degli assi X e Y vengono calcolati effettuando la sottrazione tra il punto del rispettivo asse e il punto dell'origine; il vettore dell'asse Z può essere calcolato semplicemente effettuando il prodotto vettoriale tra i due assi X e Y (3.9).

$$\begin{aligned} Xvector &= Xpoint - Origin \\ Yvector &= Ypoint - Origin \\ Zvector &= Xvector \times Yvector \end{aligned} \tag{3.9}$$

Un'idea iniziale potrebbe essere quella di prendere la posizione dell'origine e la direzione degli assi rispetto alla prima; le direzioni vengono indicate dall'utente ruotando la testa in modo che lo sguardo segua l'asse desiderato.

Il vantaggio di questo tipo di soluzione è la facilità di calibrazione: l'utente può fissare l'orientamento degli assi del mondo ruotando esclusivamente la testa. Purtroppo, in questo modo si identificano gli assi del mondo come se fossero paralleli o perpendicolari a quelli del visore e non è possibile supporre ciò visto che questi ultimi vengono fissati dopo la SLAM calibration che richiede che l'occhiale venga indossato.

Per ovviare a questo problema è possibile prendere la posizione dei 3 punti fissati dall'utente lasciando a quest'ultimo il compito di effettuare un allineamento preciso tra il mirino virtuale e il punto fisico. Questo tipo di calibrazione è più scomoda della precedente in quanto richiede il movimento dell'utente lungo gli assi e introduce due problemi legati alla precisione con cui questi vengono fissati: non è detto che gli assi siano ortogonali tra loro e non è detto che il piano XZ sia parallelo al suolo. Per ovviare alla prima situazione si utilizza la seguente procedura:

1. L'utente definisce origine, asse X e asse Y (non necessariamente perpendicolari tra loro) individuando così un piano.

2. L'asse Z viene calcolato come prodotto vettoriale tra X e Y risultando quindi ortogonale al piano definito.

3. L'asse Y viene corretto e calcolato come prodotto vettoriale tra X e Z.

Per evitare il secondo problema si può sfruttare il funzionamento dei Meta 2: ad ogni esecuzione Meta 2 richiede la calibrazione SLAM del dispositivo all'utente al termine della quale vengono definiti gli assi di riferimento e l'origine del sistema. In particolare, la camera del dispositivo diventa l'origine, l'asse Y è verso l'alto, l'asse X è verso destra e l'asse Z perpendicolare ad esse ed uscente. Il dispositivo, a prescindere dalla posizione della testa dell'utente a fine calibrazione, è in grado di definire il piano XZ parallelo al suolo. Grazie a ciò e supponendo che il "marker" contenente i 3 punti necessari per la registrazione sia posizionato verticalmente, è possibile affermare che, il fatto che il piano XZ individuato dall'utente non sia parallelo al suolo sia dovuto ad errori nella "selezione" dei punti ed applicare, quindi, un'ulteriore correzione. Viene calcolata la linea dei nodi, che è la retta in cui si intersecano i piani XZ, come prodotto vettoriale tra l'asse Y individuato dall'utente e l'asse Y dei meta (0,1,0); se tale prodotto non dà come risultato 0, ovvero i piani non sono paralleli, si effettua una rotazione degli assi, lungo la linea dei nodi, di un angolo θ che è l'angolo tra le due Y [3.8].

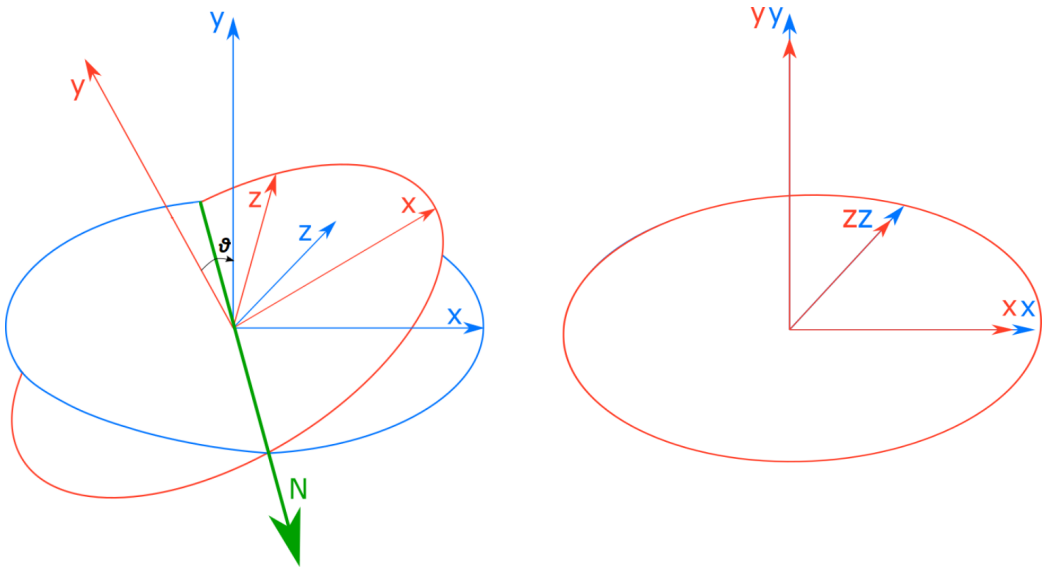

Figura 3.8: A sinistra i due sistemi di riferimento disallineati, a destra i sistemi di riferimento allineati dopo la correzione.

Procedura di registrazione La procedura di registrazione è un insieme ordinato di fasi ognuna delle quali, al termine, produce informazioni necessarie ad identificare il sistema di riferimento del mondo; è naturale, quindi, descrivere il suo comportamento mediante una macchina a stati finiti (FSM) [3.9].

L'implementazione prevedere le seguenti fasi:

- **SettingOrigin:** stato di inizio della procedura; viene abilitato il depth occlusion in modo che l'utente possa allineare il mirino con il punto di origine del mondo e premere un pulsate per confermare. Si entra in questo stato solo al termine della procedura SLAM (evento OnSlamMappingComplete() generato dai Meta 2).
- **SettingX:** l'utente deve allinearsi con il punto relativo all'asse X.
- **SettingY:** l'utente deve allinearsi con il punto Y.
- **CalibrationEnd:** la registrazione è conclusa, l'utente può decidere se ricominciare la procedura o salvare il risultato ottenuto. I vettori vengono corretti utilizzando la procedura illustrata precedentemente [3.8].
- **CalibrationFailed:** nel caso in cui gli assi siano paralleli e non sia possibile ottenere un asse Z si entra in uno stato di fallimento e viene

richiesta la pressione di un pulsante per riniziare la procedura. Questa situazione può derivare da una cattiva calibrazione da parte dell'utente.

- **CalibrationSuccess:** i dati necessari alla calibrazione sono stati ottenuti, la procedura è terminata e andata a buon fine. Viene resettato il depth occlusion allo stato iniziale.

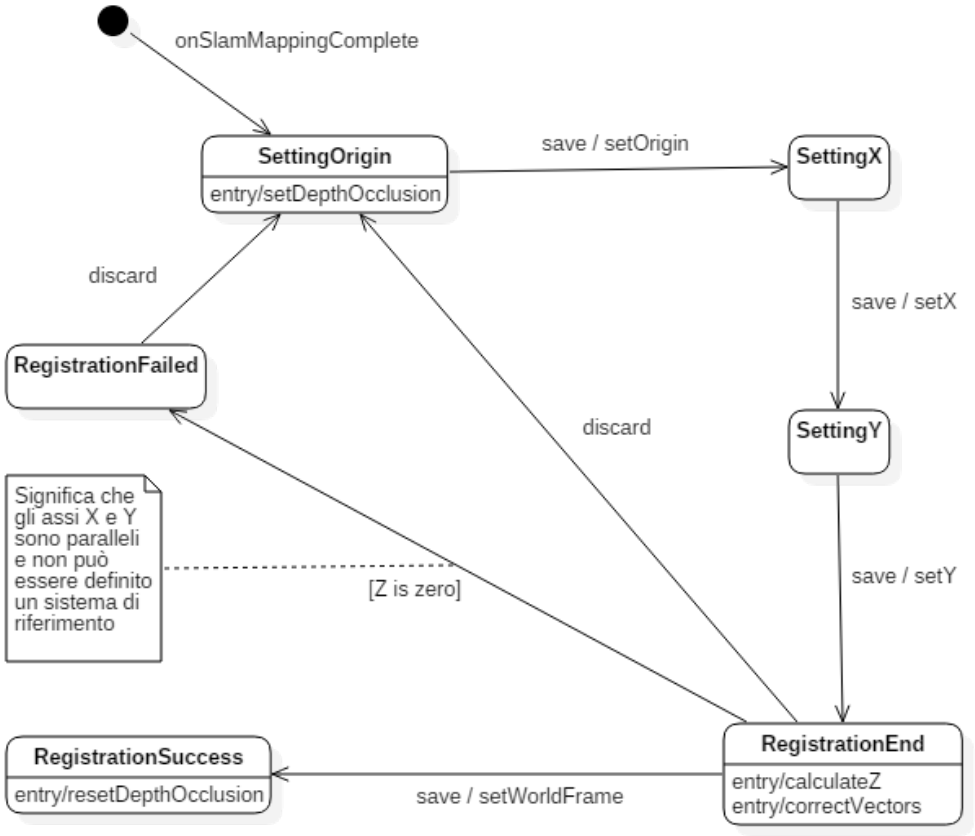

Figura 3.9: FSM procedura di registrazione.

Durante la registrazione l'utente è supportato da una semplice interfaccia grafica che gli indica le operazioni da effettuare.

I vettori ottenuti in questa procedura devono essere normalizzati; la normalizzazione consiste nella correzione del vettore in modo che questo preservi la sua direzione, ma abbia norma euclidea (lunghezza) unitaria.

$$\|v\| = \sqrt{v_1^2 + \ldots + v_n^2} = 1$$

In questo modo, qualsiasi punto che giace sulla semi-retta individuata dall'origine e dalle coordinate del vettore (x,y,z) assume lo stesso valore (x',y',z'). Questo significa che non importa a che distanza viene preso un punto dall'origine, ma solamente la direzione in cui viene preso. Ad esempio, dati due vettori:

$$v = \begin{pmatrix} 3 & 0 & 0 & 0 \end{pmatrix}$$
$$v' = \begin{pmatrix} 1/4 & 0 & 0 & 0 \end{pmatrix}$$

si ottiene lo stesso vettore normalizzato:

$$v'' = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$$

Al termine della procedura di registrazione un apposito componente si occupa della costruzione della matrice di trasformazione.

Matrice di trasformazione La costruzione della matrice può semplicemente avvenire incolonnando i vettori x, y, z rilevati nel processo di calibrazione e il punto di origine. Questo è possibile perché i vettori del sistema di riferimento del visore sono i vettori della base canonica (C); ciò comporta che i vettori x, y, z siano già espressi come combinazione lineare di C.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Supponendo che la fase di registrazione abbia prodotto i seguenti risultati:

$$\begin{aligned} x &= \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \\ y &= \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \\ z &= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \\ o &= \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \end{aligned}$$

Dall'equazione (3.7) si ottiene:

$$\begin{aligned} [0,1,0,0] &= \theta_{11}[1,0,0,0] + \theta_{12}[0,1,0,0] + \theta_{13}[0,0,1,0] + \theta_{14}[0,0,0,1] \\ [1,0,0,0] &= \theta_{21}[1,0,0,0] + \theta_{22}[0,1,0,0] + \theta_{23}[0,0,1,0] + \theta_{24}[0,0,0,1] \\ [0,1,0,0] &= \theta_{31}[1,0,0,0] + \theta_{32}[0,1,0,0] + \theta_{33}[0,0,1,0] + \theta_{34}[0,0,0,1] \\ [1,1,1,1] &= \theta_{41}[1,0,0,0] + \theta_{42}[0,1,0,0] + \theta_{43}[0,0,1,0] + \theta_{44}[0,0,0,1] \end{aligned}$$

Da questa ricaviamo:

$$\begin{aligned} \theta_{11} = 0 \;\; \theta_{12} = 1 \;\; \theta_{13} = 0 \;\; \theta_{14} = 0 \\ \theta_{21} = 1 \;\; \theta_{22} = 0 \;\; \theta_{23} = 0 \;\; \theta_{24} = 0 \\ \theta_{31} = 0 \;\; \theta_{32} = 0 \;\; \theta_{33} = 1 \;\; \theta_{34} = 0 \\ \theta_{41} = 1 \;\; \theta_{42} = 1 \;\; \theta_{43} = 1 \;\; \theta_{44} = 1 \end{aligned}$$

è evidente, leggendoli per riga, che i coefficiente θ corrispondano ad x, y, z ed o.

La matrice del cambiamento di frame $\mathrm{M_{B,C}}$, che trasforma le coordinate dell'AW in coordinate dei Meta 2, sarà quindi:

$$M_{\mathrm{B,C}} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Per effettuare l'operazione inversa è necessario, semplicemente, invertire la matrice:

$$M_{\mathrm{C,B}} = {M_{\mathrm{B,C}}}^{-1} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Queste due matrici permettono la conversione di un qualsiasi punto o vettore da un sistema di riferimento all'altro moltiplicando la matrice opportuna per il vettore o il punto corrispondente:

$$v_{\mathrm{C}} = M_{\mathrm{B,C}} * v_{\mathrm{B}}$$
$$v_{\mathrm{B}} = M_{\mathrm{C,B}} * v_{\mathrm{C}}$$

Nel sistema è implementata una classe, *FrameTransformation*, per la creazione delle matrici sopraelencate che permette in modo semplice di effettuare la trasformazione di un punto tra i due sistemi di riferimenti richiamando i metodi `ToLocal(Vector3 v)` e `ToWorld(Vector3 v)`. La classe è di carattere globale e implementata come singleton in modo da avere un' unica istanza; la procedura di calibrazione si occupa della sua inizializzazione.

Rotazione tra i sistemi di riferimento La letteratura suggerisce l'utilizzo della matrice del cambio di frame per applicare trasformazioni affini quali scala, rotazione e traslazione ad un oggetto; per fare ciò, la matrice deve essere applicata ad ogni suo vertice.

In Unity i vertici dell'oggetto sono contenuti nella sua mesh applicando la matrice a questi punti si ruota la mesh e quindi la visualizzazione dell'oggetto, ma non si modifica la proprietà transform di questo (La transform descrive posizione, rotazione e scala dell'oggetto [1.2.2.2]). Per effettuare la rotazione degli oggetti dell'AW modificando la corrispondente proprietà della transform è stato necessario sviluppare un algoritmo apposito.

Algoritmo di rotazione L'algoritmo è in grado di allineare il sistema di riferimento dell'AW con quello dei Meta ruotando i vari oggetti; per fare ciò è necessario che i due sistemi di riferimento siano entrambi ortogonali. Questo nel nostro caso è garantito dalla procedura di correzione degli assi nel processo di registrazione. Oltre a ciò, la procedura è in grado di allineare gli assi Y [3.8], quindi, data l'ortogonalità dei sistemi, è necessario allineare esclusivamente gli assi X o Z [3.10].

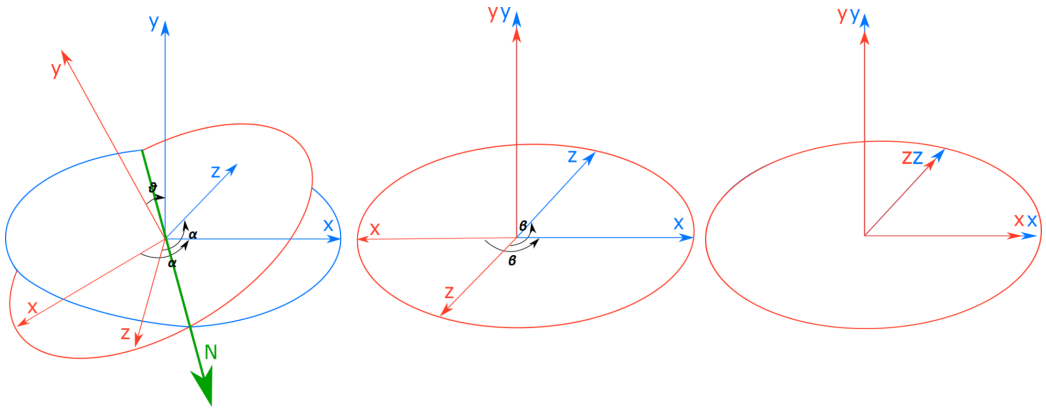

Figura 3.10: A sinistra i due sistemi di riferimento completamente disallineati. Al centro i due sistemi di riferimento con asse Y allineato dalla procedura di registrazione. A destra i sistemi allineati dall'algoritmo di rotazione.

Questo algoritmo risente di un problema legato all'orientamento dell'utente al termine della procedura di SLAM. Quest'ultimo infatti, non è detto che effettui tale procedura davanti al "marker" di registrazione inserendo in questo modo un angolo di rotazione iniziale sull'asse Y. Per ovviare a ciò è necessario salvare l'angolo θ di rotazione della camera dei Meta 2 quando viene fissata l'origine e utilizzare questo angolo per ruotare gli assi X e Z, calcolati nella procedura di registrazione, (di $-\theta$) [3.11] utilizzando gli assi ottenuti per l'esecuzione dell'algoritmo. Così facendo l'utente può effettuare la registrazione indipendentemente dalla rotazione iniziale della sua camera rispetto al "marker".Per migliorare la precisione dell'algoritmo è possibile salvare un angolo che corrisponda alla media delle rotazioni su Y dell'utente quando fissa origine, punto X e punto Y.

```
private void CreateRotatedAxis()
{
    rotatedWorldX = Quaternion.AngleAxis(-cameraRotation.eulerAngles.y, metaY) * worldX;
    rotatedWorldZ = Quaternion.AngleAxis(-cameraRotation.eulerAngles.y, metaY) * worldZ;
    rotatedWorldY = worldY;
}
```

Figura 3.11: Codice per garantire l'indipendenza del processo di registrazione dalla rotazione della camera.

Per comprendere meglio il funzionamento dell'algoritmo, nella figura sottostante [3.12] è riportato l' implementazione in C# del metodo di rotazione dal sistema di riferimento dell' AW a quello dei Meta 2.

Inizialmente viene costruito un oggetto fittizio ruotandolo del valore ricevuto dall'AW Server Node e il valore di rotazione su Y viene compensato aggiungendo quello della camera. A questo punto si calcola l'angolo β tra l'asse X del mondo, precedentemente ruotato [3.11], e l'asse X dei Meta 2, utilizzando la Y di questi ultimi come asse di rotazione (gli assi Y sono stati allineati precedentemente nella procedura di registrazione [3.8] quindi utilizzare l'asse Y del mondo produrrebbe lo stesso risultato). L'oggetto viene ruotato di β sull'asse Y, ottenendo la sua rotazione nel sistema di riferimento dei Meta 2 che viene restituita dopo averlo distrutto.

Per effettuare il cambio di sistema di riferimento inverso è necessario sottrarre, anziché sommare, la rotazione della camera nella seconda riga di codice e cambiare l'ordine di *rotatedWorldX* e *metaX* nel calcolo di β (terza linea di codice).

Questi due metodi sono contenuti nella classe *FrameTransformation*

```
public Quaternion ToLocalRotation(Quaternion rotation)
{
    /* Create dummy object to rotate */
    GameObject g = new GameObject();

    /* Set initial rotation adjusting Y object rotation due to camera initial rotation */
    g.transform.rotation = Quaternion.Euler(rotation.eulerAngles.x,
                                            rotation.eulerAngles.y + cameraRotation.eulerAngles.y,
                                            rotation.eulerAngles.z);

    /* Calculate angle between rotated world X axis and Meta 2 X axis around Y */
    float beta = Vector3.SignedAngle(rotatedWorldX, metaX, metaY);

    /* Rotate object of angle beta around Y */
    g.transform.Rotate(metaY, beta, Space.World);
    var result = g.transform.rotation;
    Object.Destroy(g);

    return result;
}
```

Figura 3.12: Algoritmo di rotazione

3.3.3 Integrazione del sistema

Per poter facilmente utilizzare Meta 2 come tecnologia olografica in MiRAgE , il sistema è stato organizzato in un prefab direttamente importabile nella scena Unity. In particolare, il prefab che si occupa della registrazione con l'AW, è il *RegistrationManager*[3.13]; quest'ultimo contiene tutti i componenti necessari per il suddetto processo:

- **Crosshair:** è la rappresentazione grafica del mirino.
- **UIHelpText:** si occupa della visualizzazione dei diversi messaggi dell'interfaccia grafica che guideranno l'utente durante il processo di registrazione.
- **MetaButtons:** prefab contenente il *MetaButtonEventBroadcaster* che mette a disposizione un metodo *Subscribe()* per potersi registrare come osservatore ai bottoni dell'occhiale; in questo modo è possibile utilizzare questi ultimi per la registrazione con l'AW.

Inoltre, l'utilizzo di Meta 2 richiede l'importazione del prefab che rappresenta la camera: *MetaCameraRig*.

Figura 3.13: Componenti della scena Unity per gestire la registrazione con AW

MiRAgE , lato Unity, fornisce già delle funzionalità per poter instaurare un canale TCP con l'AW Server Node e gestire lo scambio dei messaggi. In particolare, il prefab adibito a tale compito è il *MiRAgE_hologram_engine* a cui è agganciato lo script *AWEngine* che all'avvio del sistema di occupa di instaurare la connessione mediante il metodo *StartEngine()*. Oltre a ciò, tale script, si occupa della ricezione dei messaggi provenienti dal server che possono riguardare: creazione del mondo, creazione di AE, aggiornamento di proprietà sull'AE ed esecuzione di azioni sull'AE.

Per la gestione dei messaggi riguardanti le AE, l'AWEngine utilizza la classe statica *HologramFactory*.

Per integrare Meta 2 con MiRAgE è necessario modificare questi ultimi componenti. Il canale TCP deve essere instaurato solo dopo la procedura di registrazione, per questo, è stato modificato lo script *AWEngine* in modo che richiami il metodo *StartEngine()* solo se il prefab *RegistrationManager* non è presente; in caso contrario, sarà quest'ultimo a richiamare il metodo suddetto. Per garantire una visione consistente degli ologrammi è stata modificata la classe *HologramFactory*: i valori di rotazione e posizione ricevuti dal server vengono trasformati utilizzando i metodi *ToLocalPosition()* e *ToLocalRotation()* della classe *FrameTransformation* prima di essere assegnati all'AE [3.1].

```
localPos = FrameTransformation.Instance.ToLocalPosition(position);
localRot = FrameTransformation.Instance.ToLocalRotation(rotation);
```

Listing 3.1: Modifica dei valori di rotazione e posizione in C#

I metodi che effettuano la trasformazione inversa, *ToWorldPosition()* e *ToWorldRotation()*, vengono utilizzati solo per la comunicazione degli eventi all'infrastruttura server.

Una volta apportate le modifiche precedenti è possibile utilizzare Meta 2 su MiRAgE; la scena dovrà contenere i tre componenti mostrati nella figura sottostante[3.14].

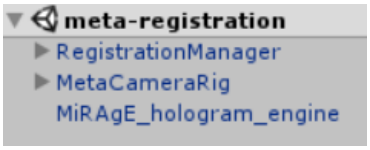

Figura 3.14: Componenti della scena Unity per utilizzare Meta2 su MiRAgE

3.3.4 Valutazione e test del lavoro svolto

L'obiettivo che si prefiggeva questa parte di progetto è stato raggiunto; Meta 2 è stato integrato con MiRAgE e permette agli utenti di registrarsi al mondo ed avere una visione consistente di ciò che li circonda.

Di per sé questo risultato è molto soddisfacente in quanto Meta 2 è una tecnologia emergente e poco esplorata; infatti, anche sulla community ufficiale degli sviluppatori, i pochi utenti che si sono cimentati in questo compito hanno utilizzato dei metodi di registrazione che richiedono di modificare manualmente i parametri di posizione e rotazione degli oggetti.

Il sistema implementato, pur non utilizzando tecniche di computer vision per rilevare i 3 punti del sistema di riferimento dell'AW, non è affatto banale, sopratutto per quanto riguarda l'algoritmo di rotazione. L'implementazione di quest'ultimo ha richiesto tante letture ed approfondimenti in ambito computer graphics. Per poterlo perfezionare è stato necessario effettuare molti test cercando ogni volta di capire cosa poteva essere migliorato. Inizialmente l'algoritmo non teneva conto della rotazione della camera su Y al momento dell'allineamento con i tre punti; questo permetteva di registrarsi correttamente solo nel caso in cui la procedura di SLAM venisse eseguita davanti al "marker". Nelle versioni successive è stata migliorata la precisione dell'algoritmo sfruttando la capacità dei Meta 2 di costruire un piano XZ parallelo al suolo.

Nel corso dell'implementazione è stato necessario aggiungere alcuni vincoli; in particolare, per potersi calibrare è necessario che il marker sia posto verticalmente, ad esempio su una parete, e quindi l'asse Y sia perpendicolare al suolo: registrazioni non verticali richiedono di compensare, oltre alla rotazione sull'asse Y, quella sulla X aumentando la complessità concettuale del problema e diminuendo la precisione di visualizzazione.

3.3.4.1 Pregi e difetti del sistema implementato

Entrando nel merito di quanto svolto è possibile sottolineare pregi e difetti. Per quanto riguarda i pregi:

(i) Sicuramente è necessario risottolineare che il sistema sviluppato è una novità e al momento, su Meta 2, non esiste nulla di simile.

(ii) Uno dei punti forti del sistema è la portabilità; lo sviluppo è **pressoché** indipendente da funzionalità peculiari dei Meta 2 e del game engine, Unity 3D; questo permette, concettualmente, di utilizzare la procedura di registrazione con altri device di visualizzazione MR o con altri engine senza dover stravolgere il codice.

(iii) Organizzando il tutto all'interno di un prefab (RegistrationManager) è possibile integrare Meta 2 con MiRAgE inserendo nella scena, oltre alla camera dei Meta 2, il solo prefab [3.14].

(iv) L'integrazione permette di fondere due grandi vantaggi offerti da Meta 2 e MiRAgE : è possibile sperimentare l'esperienza di MR multi-utente entrando in un AW e arricchirla sfruttando l'immersività offerta dai Meta 2. Inoltre, la visualizzazione degli oggetti è persistente: il mondo aumentato esiste anche in assenza di utenti; ciò significa che un utente può entrare in un AW, creare delle AE, effettuare delle modifiche visibili a tutti ed uscire senza compromettere l'esistenza dell'AW. Al suo rientro il mondo sarà esattamente come lo ha lasciato al netto di eventuali modifiche fatte dagli altri utenti.

Il sistema non è esente da difetti:

(i) Il vantaggio (ii) non è del tutto corretto. In realtà per l'implementazione del sistema è stata utilizzata la funzionalità di depth occlusion di Meta, perciò non è possibile utilizzare il sistema su dispositivi che non offrono tale funzionalità (tipicamente è integrata in tutti gli occhiali smart di tipo commercial [1.1.2.2]). Inoltre, negli occhiali che non sono in grado di costruire un piano XZ parallelo al suolo la visualizzazione risente maggiormente di errori umani nella selezione dei punti.

(ii) La procedura di registrazione è manuale e per questo motivo risulta poco precisa; essendo demandato all'utente il compito di prendere i 3 punti per l'individuazione del sistema di riferimento, il meccanismo è soggetto ad errori ed inconsistenze. In un AW multi-utente le diverse registrazioni effettuate da questi ultimi causano un errore di posizionamento e rotazione che potrebbe compromettere l'esperienza. Per stimare questo errore sono stati eseguiti test sperimentali descritti nella sezione successiva.

3.3.4.2 Test sperimentali del sistema

È importante sottolineare che l'esecuzione di test, per verificare la precisione del sistema, accurati e automatizzati sono difficilmente realizzabili. Questo perché la procedura di SLAM dei due occhiali dovrebbe essere effettuata nella stessa posizione e con la stessa rotazione rispetto al "marker". In questo caso, visto che i due occhiali avrebbero lo stesso sistema di riferimento, sarebbe possibile confrontare i valori di posizione e rotazione degli oggetti per stimarne l'errore dovuto alle diverse registrazioni.

Tale procedura però è di tipo manuale e deve essere fatta ruotando la testa a destra e sinistra, motivo per cui non si può essere sicuri di terminarla con lo stesso angolo di rotazione rispetto al "marker".

I test eseguiti, quindi, sono stati di carattere sperimentale. In particolare, è stata eseguita un'istanza di un AW e mediante un agente è stato creato un cubo in una determinata posizione. A questo punto abbiamo effettuato la registrazione con il centro del mondo e avvicinandoci al cubo abbiamo individuato con un dito lo stesso vertice, stimando in questo modo l'errore. Questo tipo di test è stato effettuato più volte e ha rilevato un errore medio di qualche centimetro (3/5 cm).

Per verificare la rotazione è stato creato un ologramma che rappresenta un cubo numerato; la composizione asimmetrica di quest'ultimo permette in modo semplice di rilevare la correttezza di rotazioni non troppo complesse [3.15].

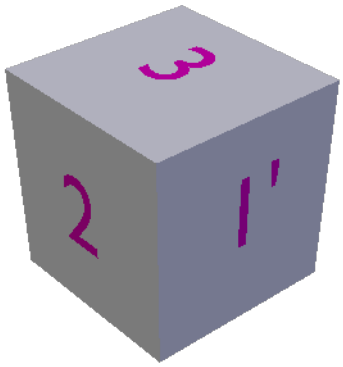

Figura 3.15: Cubo numerato per comprendere l'errore di rotazione

Dai numerosi test eseguiti abbiamo notato che:

- Nel caso in cui gli oggetti siano lontani, l'esperienza immersiva risulta meno appagante. In particolare, dopo un paio di metri di distanza dall'oggetto si ha l'impressione che questo si muova con l'occhiale; ciò probabilmente è dovuto ad errori nel sistema di tracking del dispositivo. In un contesto di visualizzazione multi-utente questo errore si traduce in inconsistenza di visualizzazione.

- Un altro errore che aumenta con la distanza è quello di posizionamento; questo è dovuto alle diverse registrazioni degli utenti. Anche differenze di pochi gradi nella definizione dei sistemi di riferimento possono portare ad errori di posizionamento visibili nel caso in cui l'oggetto sia molto distante dall'origine. Per questo motivo abbiamo identificato due tipologie di errore di registrazione: **errore costante** ed **errore variabile**. Il primo è dovuto alle diverse posizioni in cui gli utenti fissano l'origine e rimane costante all'allontanarsi dell'oggetto. Il secondo è dovuto alla differenza con cui gli utenti fissano gli assi ed aumenta proporzionalmente alla distanza dell'oggetto dall'origine.

Va sottolineato che Meta 2 è ancora in fase prototipale ed essendo dotato di cavo non permette di raggiungere distanze elevate dal centro dell'AW. Inoltre, tale dispositivo, è pensato per una visualizzazione MR in un ambiente confinato come ad esempio una scrivania.

Nell'immagine sottostante [3.16] è riportato un caso di esecuzione in cui viene visualizzato un cubo numerato a -50 cm sull'asse Z e 50 cm sull'asse Y. Le due foto sono state scattate a distanze diverse dall'oggetto e non è comprensibile l'errore di posizionamento che comunque era nell'ordine dei 2.5 cm. È possibile però notare il risultato ottenuto e la coerenza delle due visualizzazioni.

Figura 3.16: Visualizzazione da parte di due utenti di un cubo numerato da prospettive diverse

Capitolo 4

Sviluppo libreria di oggetti aumentati per AW

La scienza della progettazione di interfacce è tra le più antiche attività umane e risale alla nascita del primo attrezzo utilizzato dagli uomini primitivi.
Da allora lo sviluppo di interfacce si è esteso creando strumenti di interazione con il mondo fisico sempre più complessi, fino al picco massimo avvenuto nell'ultimo secolo con la nascita e lo sviluppo dei computer.

Tuttavia, tutte le competenze legate alle interfacce acquisite nei secoli, non vengono sfruttate per l'interazione con il mondo digitale in cui quest'ultima è sostanzialmente limitata alle Graphical User Interface (GUI). Le GUI si sono dimostrate un modello per l'interazione uomo-macchina di successo e di lunga durata ed hanno dominato l'ultimo decennio nel mondo del design delle interfacce. Ciò nonostante, tale modello, ha delle mancanze soprattutto per quanto riguarda l'interazione tra le persone e gli ambienti fisici che le circondano.
La AR e la MR hanno iniziato ad affrontare questo problema spostando lo spazio delle interfacce digitali dal desktop all'ambiente fisico. Purtroppo, la progettazione di interfacce basate su queste tecnologie è rimasta molto legata alle GUI: si è cercato di esportare il paradigma di queste ultime all'interno di interfacce AR/MR perdendo la ricchezza delle interazioni fisicospaziali che invece dovrebbe aumentare.

È necessario individuare quali tipi di oggetti potrebbero essere utili in queste nuove interfacce cercando di non sfruttare analogie con quelle già conosciute e studiate; ad esempio, il classico bidone delle interfacce desktop potrebbe essere sostituito da una gesture che simula il lancio di un oggetto virtuale.

Gli studi legati agli oggetti utilizzati nelle interfacce grafiche sono molto vasti e complessi e in ambito di AR/MR ancora molto "acerbi".

4.1 Analisi del problema

Per rendere lo sviluppatore il più indipendente possibile dall'engine utilizzato per sviluppare applicazioni utente per AW, nel nostro caso Unity, è stata creata una libreria che semplifica la creazione di un'AE con relativo ologramma associato.

Prima di tutto, è necessario individuare quali oggetti potrebbero essere utili in interfacce AR/MR di supporto al CSCW; di seguito è riportata la lista di alcuni oggetti individuati:

- **Post-it:** Mediante questo oggetto è possibile annotare appunti o reminder che possono essere associati in maniera solidale ad oggetti fisici. Ad esempio, il posizionamento di un post-it sopra ad un libro in grado di muoversi con quest'ultimo.

- **Nodi e archi:** Mediante nodi e archi è possibile nell'ambito dell'ITC schematizzare: programmi, struttura di rete, mappa di siti... questi oggetti potrebbero essere usati anche come strumento di misurazione posizionando nodi agli estremi di un oggetto per conoscerne la lunghezza.

- **Tastiera virtuale:** In alcuni casi potrebbe essere impossibile utilizzare dispositivi fisici di input come tastiere o smartphone e dato che non tutti gli occhiali supportano interazioni vocali è necessario disporre di un dispositivo di input come una tastiera virtuale.

- **Linee per disegno a mano libera:** Mediante questo oggetto è possibile disegnare nello spazio linee di diversi colori. Questa funzionalità risulta molto utile per focalizzare l'attenzione degli utenti su un oggetto o un particolare di quest'ultimo circondandolo, ad esempio, con una linea di colore rosso.

Gli oggetti individuati sono di carattere generico ed utilizzabili in tutti i contesti applicativi, ovviamente la lista potrebbe essere estesa con specifici oggetti per applicazioni differenti. Ad esempio, nel mondo dell'ingegneria urbana si potrebbero definire oggetti per la costruzione di edifici, oggetti per l'illuminazione che consentano lo studio delle ombre...

4.2 Progettazione e sviluppo della libreria

In MiRAgE le AE sono definite all'interno dell'AW Server e possono avere associato un ologramma gestito dal componente HE (Hologram Engine) [2].

Mentre lato server è necessario implementare una classe che descriva l'AE in termini di proprietà ed azioni e permetta, agli agenti, la modifica di queste, lato client è necessario sviluppare un componente che si occupi di mantenere la visualizzazione dell'ologramma consistente con lo stato dell'entità, ovvero l'insieme dei valori delle sue proprietà in un determinato istante.

4.2.1 Definizione AE lato AW Server Node

Per definire un'entità aumentata sul server è necessario implementare una classe (che chiameremo classe *Template*), che estenda dalla classe AE e contenga l'insieme di proprietà e azioni (custom) peculiari dell'oggetto descritto. La classe AE è utilizzata per descrivere una generica entità e contiene:

- informazioni utili ad identificarla;
- delle proprietà legate alla sua geometria;
- una mappa di proprietà custom;
- una mappa di azioni custom;
- due mappe per memorizzare gli observer dell'entità; una memorizza gli observer interessati alle singole proprietà, l'altra quelli interessati all'entità stessa.

L'interfaccia completa della classe AE è riportata nell'immagine sottostante [4.1].

```java
public interface AEInterface {
    /* GETTERS */
    String id();
    String tag();
    String type();
    String hologramRes();

    /* PROPERTY GETTERS */
    Location location();
    Orientation orientation();
    Extension extension();
    JsonArray things();
    Set<String> customProperties();
    Object customProperty(String name) throws PropertyNotFoundException;

    /*PROPERTY SETTERS*/
    void location(Location location);
    void orientation(Orientation orientation);
    void extension(Extension extension);
    void things(JsonArray things);
    void customProperty(String name, Object newValue)
            throws PropertyNotFoundException;

    /*ACTION*/
    Map<String, String[]> actions();
    <T extends AE> void executeAction(T ae, String name, Object... params)
            throws ActionNotFoundException;

    /*OBSERVERS METHOD*/
    void addAEObserver(String sessionID, ServerWebSocket ws);
    void addAEPropertyObserver(String ip, ServerWebSocket ws, String propertyName)
            throws PropertyNotFoundException;
    void removeObserver(String sessionID);

    /*JSON REPRESENTATION OF AE'S ACTION AND PROPERTY*/
    JsonObject getJSONRepresentation();
}
```

Figura 4.1: Interfaccia classe AE

All'interno della classe Template le proprietà sono di tipo stringa e identificate mediante la notazione *@PROPERTY* e le azioni mediante la notazione *@ACTION*; inoltre la classe deve specificare mediante *@HOLOGRAM* il nome dell' ologramma associato all' AE.
Nel caso specifico, sono state implementate le classi Template per gli oggetti postit e lavagna [4.2].

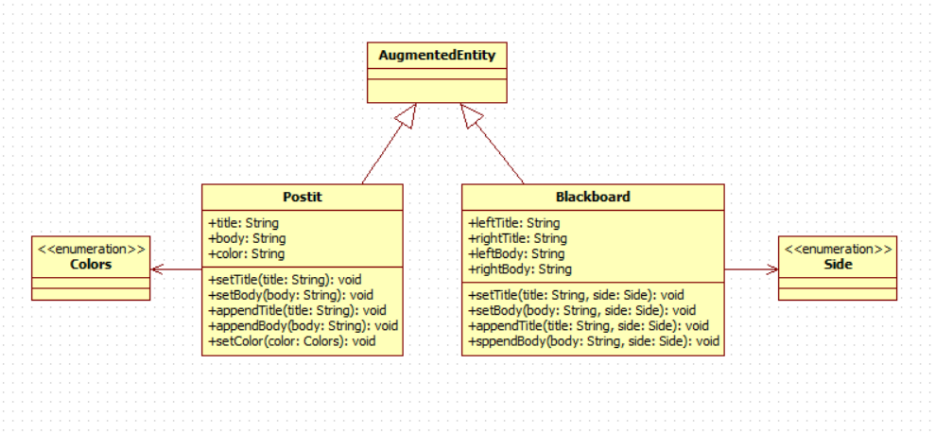

Figura 4.2: Diagramma delle classi relativo alle AE implementate

La classe *Postit* memorizza come proprietà titolo, testo del corpo, colore e implementa le azioni per settare/aggiornare le proprietà [4.3].

La classe *Blackboard* è analoga a quella precedente, ma possiede due titoli e due corpi (uno a sinistra e uno a destra) e non possiede il colore [4.3].

```
public interface PostitInterface {
    void setTitle(String text);
    void setBody(String text);
    void appendTitle(String text);
    void appendBody(String text);
    void setColor(Colors color);
}
```

```
public interface BlackboardInterface {
    void setTitle(String text, Side s);
    void setBody(String text, Side s);
    void appendTitle(String text, Side s);
    void appendBody(String text, Side s);
}
```

Figura 4.3: A sinistra l'interfaccia della classe Postit, a destra l'interfaccia della classe Blackboard

Tutte le azioni, per modificare il valore della proprietà custom, utilizzano il metodo setter della classe padre, *customProperty(String name, object value)*; se non utilizzassero tale metodo la modifica verrebbe apportata agli attributi locali e non alle proprietà dell'AE.

Per gestire il colore del post-it è stata implementata l'enumerazione Colors, in questo modo si limitano i possibili valori che il campo color può assumere evitando problemi di rendering lato client (mancanza del Material che rappresenta il colore). Siccome le proprietà sono di tipo stringa il metodo *setColor()* effettuerà un *toString()* sul campo color preso in input.

In modo analogo, per una gestione più semplice, il lato della lavagna è rappresentato da un enumeratore *Side*.

Grazie alla definizione delle classi Template è possibile creare AE che rappresentano postit e lavagna utilizzando il metodo *createAE()* fornito dal componente *WoATMiddleware*. Tale metodo richiede in input i seguenti parametri:

- **String sessionId:** identificativo della sessione di lavoro.
- **String awId:** identificativo dell' AW su cui si vuole creare l'entità.
- **String entityName:** nome dell'entità da creare.
- **String template:** nome della classe template.
- **String tag:** tag da assegnare all'oggetto quando verrà istanziato nell'engine lato client.
- **LocationAEProperty location:** posizione dell'AE.
- **ExtensionAEProperty extension:** scala dell'AE.
- **OrientationAEProperty orientation:** rotazione dell'AE.
- **JsonArray things:** Array in formato Json per l'inizializzazione delle proprietà custom.
- **Handler<AsyncResult<String>> handler:** handler asincrono del risultato.

Di seguito è riportato il codice Java per creare un post-it in posizione (0,0,0) e con rotazione 0 su tutti gli assi [4.4].

```
WoATMiddleware.getInstance().createAE(
        sessionID,
        awName,
        "postit001",
        "Postit",
        "postit",
        new LocationAEProperty(new CartesianLocation(0, 0, 0)),
        new ExtensionAEProperty(new BasicExtension(0)),
        new OrientationAEProperty(new AngularOrientation(0, 0, 0)),
        new JsonArray(),
        res -> {
            if(res.succeeded()) {
                System.out.println("Post-it created");
            } else {
                System.out.println(res.cause().getMessage());
            }
        });
```

Figura 4.4: Codice Java per creare un post-it

4.2.2 Gestione ologramma lato client

Per poter mantenere aggiornata la visualizzazione dell'ologramma associato ad un'AE è necessario sviluppare ed agganciare a quest'ultimo un controller specifico che deriva dalla classe *HologramController*. Questa classe gestisce la geometria dell'ologramma (posizione, locazione, estensione) e definisce un metodo, che deve essere implementato dalle classi figlie, per la rilevazione e le gestione degli aggiornamenti delle proprietà custom. Nel caso specifico sono stati implementati due controller: PostitController e BlackboardController. Questi ultimi hanno un comportamento molto simile; all'arrivo di un aggiornamento controllano a quale proprietà esso si riferisce: nel caso in cui si riferisca al titolo ottengono la mesh, contenente il testo di quest'ultimo, associata all'oggetto e la aggiornano; nel caso in cui si riferisca al body il procedimento è analogo, ma cambia la mesh che viene aggiornata.

Il controller del post-it si occupa anche dei cambiamenti di colore; in particolare, a seconda della stringa che riceve ("cyan", "yellow" o "pink") associa il corrispondente materiale al post-it.

I modelli 3D degli ologrammi sono stati disegnati utilizzando Blender in quanto Unity 3D non consente modellazioni complesse degli ogetti base [4.5][4.6]. Siccome questi due software lavorano con sistemi di riferimento diversi è stato necessario apportare al modello Blender delle rotazioni aggiuntive in modo che la visualizzazione a (0,0,0) fosse frontale. Inoltre, eventuali texture associate al modello Blender non vengono importate su Unity, quindi è necessario definirle su quest'ultimo e associarle ai modelli dopo averli importati. A seguito dell'importazione del modello gli è stato aggiunto un canvas contenente le mesh 2D del titolo e del corpo.

Le mesh non sono quelle base offerte da Unity, ma è stato utilizzato un plugin *TextMeshPro* che, opportunamente configurato, consente una gestione articolata dell'overflow e dell'allineamento del testo.

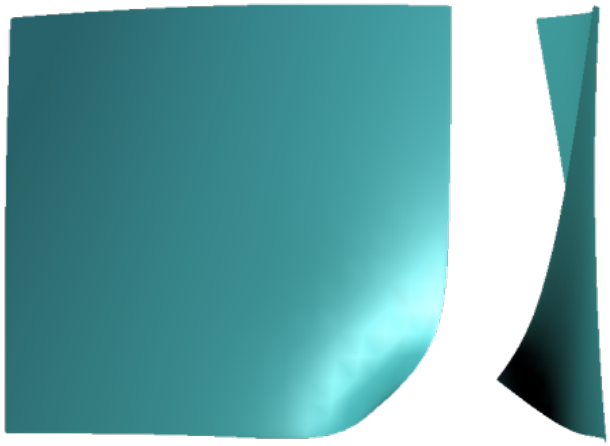

Figura 4.5: Modello post-it con materiale color ciano visto da due prospettive differenti

Figura 4.6: Modello lavagna visto frontalmente

4.2.3 Valutazione e test del lavoro svolto

L'obiettivo di questa parte di progetto è stato raggiunto: il post-it è stato implementato ed è stata aggiunta anche una sua variante, la lavagna; la creazione di questi oggetti risulta rapida e non richiede di disegnare e implementare il codice per la gestione dell'ologramma. Questo è sicuramente un grande vantaggio per gli sviluppatori di mondi aumentati che utilizzano tali AE. La parte grafica è stata curata particolarmente per aumentare la user experience parametro fondamentale di valutazione delle interfacce odierne; ciò non è stato banale ed ha richiesto tempo: non essendo esperto di model-

lazione 3D è stato necessario prendere confidenza con Blender ed apprendere le nozioni utili a disegnare gli oggetti.

Gli studi legati alle interfacce ed agli oggetti utili all'interno di queste sono molto vasti e complessi; inoltre, in ambito AR/MR tali studi sono ancora molto "acerbi". Non potendo usufruire di informazioni date dalla letteratura l'individuazione degli oggetti non è stata semplice e questi potrebbero rivelarsi inadeguati.

Purtroppo, gli oggetti implementati non sono in grado di adattarsi al testo, infatti nel caso in cui questo sia troppo lungo viene troncato. Per consentire all'utente di inserire testi più lunghi è stata sviluppata la lavagna, che però non risolve completamente questo problema.

Per poter testare gli oggetti implementati è stato sviluppato un agente che crea un post-it ed effettua qualche modifica. L'agente non è altro che un thread Java che all'interno del metodo run() effettua delle chiamate ai metodi offerti dal componente *WoATMiddleware*. Di seguito è riportata l'implementazione in Java [4.8].

I metodi utilizzati non fanno altro che richiamare le funzionalità offerte dal *WoATMiddleware*, questo per mantenere più pulito il corpo di *run()*. Non sono stati inseriti degli sleep() tra la creazione e la scrittura del titolo e del corpo in modo da poter stimare la latenza di visualizzazione, dovuta alla comunicazione e gestione dell'azione, che si aggira intorno ai 2 secondi.

```
@Override
public void run() {
    super.run();
    try {
        joinAW();
        sleep(1500);
        createAE(aeID , new CartesianLocation(0.5, 0.5, 0));
        getAEproperty("title");
        invokeAction("setTitle", new JsonArray().add("Titolo"));
        getAEproperty("title");
        invokeAction("setBody", new JsonArray().add("Lorem Ipsum"));
        getAEproperty("body");
        sleep(1500);
        quitAW();
    }catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Figura 4.7: Codice Java dell'agente che crea e modifica un post-it

Il risultato, visualizzato su Meta 2, dell'esecuzione dell'agente all'interno dell'AW è riportato nella figura sottostante [4.8].

Per testare il comportamento delle mesh del titolo e del corpo nel caso in cui il testo fosse troppo lungo è stato modificato l'agente sopra riportato. In particolare, più il testo è lungo più la dimensione del font viene ridotta fino al raggiungimento di una dimensione minima, che comunque ne consenta una buona visualizzazione, dopo la quale il testo viene troncato [4.8].

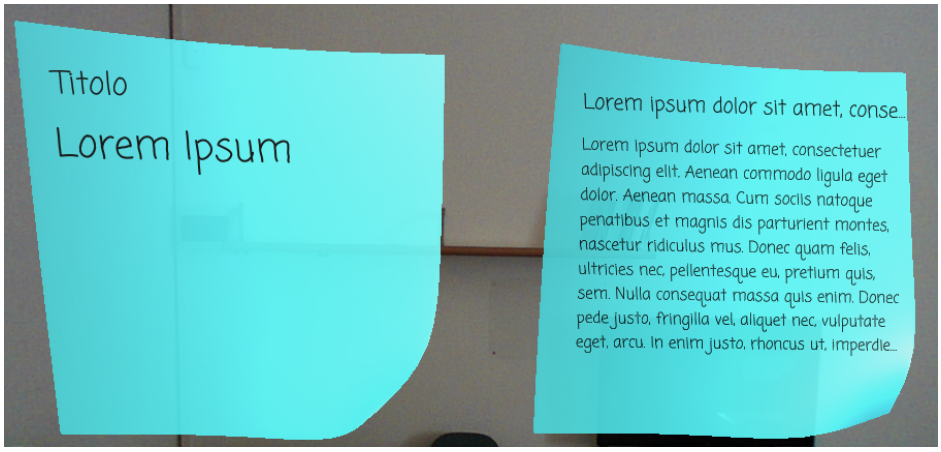

Figura 4.8: Visualizzazione di due post-it creati da un agente. Quello di sinistra contiene un titolo e un corpo corto, quello di destra contiene un titolo e un corpo lungo che viene troncato.

Siccome l'oggetto deve essere visto nello stesso modo a prescindere dal dispositivo di visualizzazione è stato eseguito anche un test utilizzando Vuforia che attualmente è il metodo di visualizzazione utilizzato da MiRAgE. In particolare, il test si è concentrato solo sulla visualizzazione, non ha coinvolto, quindi, l'utilizzo di MiRAgE .

L'applicazione è stata installata ed eseguita su uno smartphone Android; la figura sottostante riporta il risultato ottenuto [4.9].

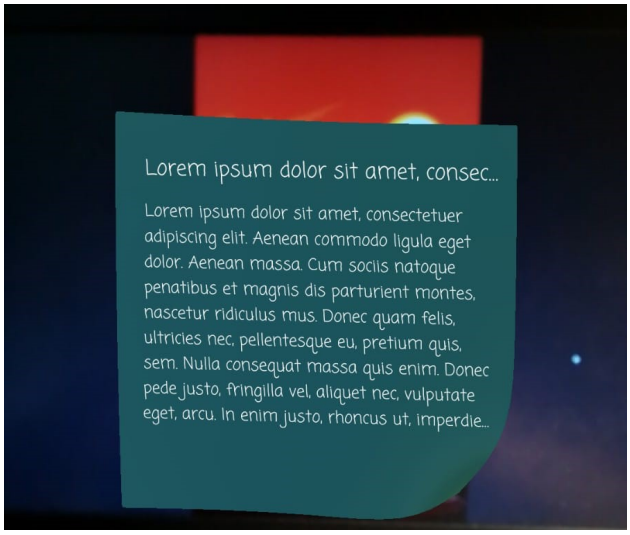

Figura 4.9: Visualizzazione di un post-it su smartphone Android tramite Vuforia.

Ovviamente, siccome le dimensioni del post-it sono le stesse il testo viene troncato e visualizzato allo stesso modo sia su Meta 2 [4.8] sia su Vuforia [4.9]; lo stesso materiale, invece, viene renderizzato in maniera differente dai due dispositivi.

Conclusioni e sviluppi futuri

Questa tesi si prefiggeva un duplice obiettivo: integrare un occhiale per AR, Meta 2, con il framework per la creazione di mondi aumentati (AW) MiRAgE e sviluppare una libreria di oggetti per consentire al programmatore di AW una creazione di entità aumentate semplice e indipendente dall'engine utilizzato per la gestione degli ologrammi. Il primo traguardo è stato raggiunto in maniera cooperativa con l'aiuto di Emanuele Pancisi, il secondo, invece, è stato oggetto di uno studio ed un lavoro individuale.

La parte di progetto cooperativa è stata molto complessa ed ha richiesto la maggior parte del tempo; questo a causa del suo carattere prettamente algebrico/geometrico e legato a nozioni di computer graphics che non avevamo mai approfondito. Inoltre, dato che Meta non fornisce nessun supporto per riconoscimento dei marker è stato necessario sviluppare una procedura di registrazione apposita. Un'altra difficoltà significativa è stata quella di non poter applicare, come suggerisce la letteratura, la matrice di cambio frame ai vertici di un oggetto per ruotarlo, questo ha richiesto lo sviluppo di un algoritmo di rotazione ad hoc.

Il punto forte della nostra implementazione è la portabilità: non sono state sfruttate funzionalità peculiari né dei Meta 2 né di Unity e questo consente, dal punto di vista concettuale, un'integrazione semplice con altri occhiali o engine.

Non mancano però punti deboli: il più significativo è legato alla precisione della procedura di registrazione; essendo demandato all'utente il compito di prendere 3 punti per l'identificazione del sistema di riferimento, il meccanismo è soggetto ad errori ed inconsistenze. In base a come gli utenti effettuano la registrazione avranno una visione differente del mondo aumentato; alcuni test sperimentali hanno mostrato un'imprecisione nell'ordine dei centimetri. Il risultato è comunque molto buono, ma in futuro sarà necessario sostituire la procedura di registrazione manuale con una basata su rilevazione marker. Inoltre, è necessario sottolineare che il processo di registrazione è di tipo statico: viene effettuato solo una volta all'avvio; una registrazione di tipo dinamico viene eseguita anche durante l'esecuzione dell'applicazione e

consente una visualizzazione più precisa.

La parte individuale è di carattere più pratico rispetto alla precedente; in particolare, per aumentare la user experience sono stati disegnati modelli dal design accattivante, ciò ha richiesto l'impiego e lo studio di una piattaforma di modellazione, Blender, diversa da quella fornita da Unity.

A livello concettuale la parte più impegnativa è stata comprendere il funzionamento di MiRAgE in modo da poter implementare correttamente la libreria.

Anche in questo caso l'obiettivo prefissato è stato raggiunto e il risultato è molto buono, ma la parte è sicuramente soggetta a numerosi sviluppi futuri, tra cui, i due più importanti sono:

- **Studio più approfondito legato alle interfacce AR/MR:** la disciplina legata allo studio di interfacce e oggetti utili all'interno di queste è molto vasta e complessa; purtroppo, essendo MR e AR due tecnologie abbastanza emergenti, in questo ambito tale studio è ancora molto "acerbo". Per questo motivo, l'individuazione di oggetti utili in queste interfacce sarà da ripetere cercandone di nuovi.

- **Migliorare l'indipendenza da Unity:** La libreria non potrà mai essere abbastanza vasta da fornire tutti gli oggetti utili e alcuni programmatori dovranno sviluppare le proprie AE. La gestione degli ologrammi lato client è ancora molto dipendente dalla struttura di Unity, per cui sarebbe utile cercare di costruire un livello di astrazione che prescinda da questa.

Per concludere, sono molto soddisfatto del lavoro svolto per la realizzazione di questa tesi in quanto, insieme a Pancisi, sono riusciti a realizzare qualcosa di nuovo e ancora inesistente. Inoltre, la parte di progetto singola mi ha permesso di cimentarmi in qualcosa di più pratico ottenendo buoni risultati.

Ringraziamenti

Il ringraziamento più grande va ai miei genitori, Marina e Paolino, che hanno permesso tutto questo. Lungo il mio percorso di vita sono stati saggi consiglieri sempre pronti a tendermi la mano nel momento del bisogno.

In secondo luogo, vorrei ringraziare mia sorella, Deborah, e mio fratello, Davide, che mi hanno sempre coccolato e sostenuto, esprimendo ammirazione per i risultati da me ottenuti.

Grazie alle mie splendide nipoti che ogni giorno mi ricordano la bellezza dei piccoli gesti, la magia e il valore di un sorriso e l'importanza della curiosità.

Un grazie speciale va alla mia fidanzata, Martina, il cui amore mi fa sentire capace di grandi cose; in questi duri mesi è sempre stata al mio fianco spronandomi e motivandomi per permettermi di raggiungere questo risultato.

Grazie agli amici di una vita con cui ho riso, pianto e che a volte vengono messi da parte, ma in realtà sono sempre pronti ad aiutarti nel momento del bisogno.

Grazie ai miei compagni di studio, Giacomo e Manuele, che hanno reso, e renderanno, questo percorso indimenticabile.

Un enorme ringraziamento va ad un amico conosciuto grazie a questo lavoro, Emanuele, con cui ho condiviso tutti i giorni degli ultimi due mesi e di cui ho apprezzato la purezza e la genuinità. Insieme abbiamo trovato la forza per raggiungere questo traguardo.

Un ultimo sentito ringraziamento è rivolto al professore Alessandro Ricci e al dottore Angelo Croatti che in questi mesi hanno dedicato parte del loro tempo al nostro progetto dandoci preziosi consigli di cui farò tesoro.

Bibliografia

[1] Akira Utsumi Fumio Kishino Paul Milgram, Haruo Takemura. In *Augmented Reality: A class of displays on the reality-virtuality continuum*, ATR Communication Systems Research Laboratories, Japan, 1994.

[2] Reinhold Behringer Steven Feiner Simon Julier Blair MacIntyre Ronald Azuma, Yohan Baillot. In *Recent Advances in Augmented Reality*, 2001.

[3] Ronald Azuma. In *A Survey of Augmented Reality*, Hughes Research Laboratories,Malibu, CA 90265, 1997.

[4] Jon Peddie. In *Augmented Reality: Where We Will All Live*, 2017.

[5] Luca Tummolini Cristiano Castelfranchi Alessandro Ricci, Michele Piunti. In *The Mirror World: Preparing for Mixed-Reality Living*, 2015.

[6] Alessandro Ricci Angelo Croatti. In *A Model and Platform for Building Agent-Based Pervasive Mixed Reality Systems*, DISI, University of Bologna, Via Sacchi 3, Cesena, Italy.

[7] Ronald T. Azuma. In *A Survey of Augmented Reality*, Hughes Research Laboratories, Malibu Canyon Road, MS RL96.

[8] Yan Yan. Registration issues in augmented reality. University of Birmingham, 2014-2015.

[9] Hirokazu Kato Mark Billinghurst. In *Collaborative Mixed Reality*, Human Interface Technology Laboratory, Seattle, WA 98195, USA, 1999.

[10] Will Goldstone. In *Unity Game Development Essentials*, 2009.

Sitografia

[1] "brevetto lenti sony." https://www.cnet.com/news/sony-patents-contact-lens-that-records-what-you-see/.

[2] "discorso di Meron Gribetz al Ted." https://www.ted.com/talks/meron_gribetz_a_glimpse_of_the_future_through_an_augmented_reality_headset.

[3] "Vuforia." https://www.vuforia.com/features.html.

[4] "ARToolKit." https://www.hitl.washington.edu/artoolkit/.

[5] "Augment." http://www.augment.com/technology/.

[6] "Opencv." https://opencv.org/about.html.

[7] "Unity." https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf.

[8] "Unity." https://docs.unity3d.com/Manual/.

[9] "Meta." https://docs.metavision.com/external/doc/latest/.

[10] "Cambio sistemi di riferimento." http://vcg.isti.cnr.it/~cignoni/FGT0708/FGT_04_Trasformazioni.pdf.

[11] "Sviluppo di interfacce." https://www.politesi.polimi.it/bitstream/10589/4122/1/2010_10_Cibien.pdf.