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 del sistema di gestione dell'input dell'utente

Relazione finale in:
Sistemi Embedded e Internet-of-Things

Relatore:
Prof. Alessandro Ricci

Presentata da:
Emanuele Pancisi

Correlatore:
Dott. Ing. Angelo Croatti

I sessione di laurea
Anno accademico: 2017-2018

PAROLE CHIAVE

Augmented Reality
Mixed Reality
Augmented World
MiRAgE
Meta 2
Unity

alla mia famiglia

Indice

Introduzione **v**

1 Augmented e Mixed Reality **1**
1.1 Mixed Reality 2
1.2 Dispositivi per realtà aumentata 3
1.2.1 Microsoft HoloLens 4
1.2.2 Meta 2 5
1.2.3 Magic Leap One 6
1.3 Software per realtà aumentata 7
1.3.1 Piattaforme software 7
1.3.2 Tools di sviluppo 10

2 Augmented World **15**
2.1 Introduzione 15
2.2 Modello concettuale 17
2.3 MiRAgE: Augmented World framework 19
2.3.1 Architettura logica 20
2.3.2 Agenti e primitive 21

3 Integrazione Meta 2 con MiRAgE **23**
3.1 Introduzione 23
3.2 Approfondimento Meta 2 24
3.2.1 Specifiche tecniche 24
3.2.2 Unity SDK 2.7.0 25
3.3 Supporto per mondi condivisi in MiRAgE mediante Meta 2 . . 32
3.3.1 Analisi 32
3.3.2 Progettazione e sviluppo 39
3.3.3 Integrazione del sistema 48
3.4 Considerazioni e risultati ottenuti 50
3.4.1 Pregi e difetti del sistema 50
3.4.2 Test di precisione del sistema 51

4 Sistema di gestione degli eventi e degli input **54**
4.1 Gestione dell'input in Meta 2 54
4.1.1 Dispositivo di interesse: Meta 2 55
4.1.2 Eventi di input 55
4.2 Integrazione in MiRAgE 56
4.2.1 Aspetti architetturali 56
4.2.2 Gerarchia degli eventi 57
4.2.3 Rilevazione degli eventi 59
4.2.4 Comunicazione degli eventi 62
4.3 Considerazioni 63

Conclusioni e sviluppi futuri **66**

Ringraziamenti **68**

Introduzione

La realtà aumentata è diventata, negli ultimi anni, una delle maggiori aree di interesse nello studio e nello sviluppo di applicazioni e tecnologie innovative in ambito ICT. Questo successo deriva dal grande potenziale e dai grandi benefici che questa tecnologia può portare nella vita di tutti i giorni.

La possibilità di estendere il mondo reale sovrapponendovi informazioni e oggetti virtuali apre grandi scenari di sviluppo nei contesti applicativi più disparati. La realtà aumentata può avere un grande impatto da un punto di vista applicativo rimodellando l'ambiente in cui le persone lavorano e vivono e cambiando notevolmente il modo in cui interagiscono e collaborano.

Queste potenzialità vengono ulteriormente amplificate se integrate e messe in sinergia con le tecnologie hardware emergenti in ambito Internet of Things (IoT) e Web of Things(WoT) andando a delineare dei veri e propri ambienti intelligenti.

L'obiettivo di questa tesi di laurea è duplice: integrare il dispositivo per realtà aumentata Meta 2 con il framework per lo sviluppo di mondi aumentati MiRAgE e sviluppare successivamente una astrazione per la gestione degli eventi e degli input che prescinda dal particolare dispositivo utilizzato.

Inizialmente viene fatta una panoramica sul contesto dell'intera trattazione: vengono definiti i concetti chiave di realtà aumentata e realtà mixata, vengono categorizzati e descritti i vari tipi di dispositivi con particolare accezione ai dispositivi dedicati (Microsoft HoloLens, Meta2, Magic Leap One) e vengono analizzate le più importanti piattaforme e i principali strumenti di sviluppo disponibili.

Successivamente viene presentato il framework per lo sviluppo di sistemi di realtà mixata basati su agenti, MiRAgE; in particolare viene analizzato il modello concettuale di Augmented World sul quale si basa, vengono trattate le principali funzionalità e caratteristiche e vengono analizzati i componenti architetturali.

Nel terzo capitolo figura la prima parte di progetto realizzata in collaborazione con Daniele Rossi che si pone come obiettivo quello di integrare il dispositivo Meta 2 con il framework MiRAgE. Inizialmente vengono descritte

in modo approfondito le caratteristiche e le funzionalità del dispositivo per entrare, poi, nel merito del sistema sviluppato effettuando un'analisi del problema con particolare interesse al concetto di registrazione e sottolineando le problematiche che derivano dallo stato di sviluppo attuale del dispositivo. In questa parte di analisi vengono riportati, inoltre, i concetti matematici basilari necessari per comprendere le trasformazioni tra diversi sistemi di riferimento.

La parte di progettazione entra nel merito e mostra come è stato sviluppato il sistema: inizialmente viene descritta la procedura di registrazione progettata evidenziando alcuni problemi e le relative soluzioni, viene esplicitata la necessità delle nozioni di matematica, precedentemente introdotte, per la costruzione della matrice di trasformazione tra sistemi di riferimento e viene introdotto il problema della rotazione e la sua risoluzione attraverso un algoritmo apposito.

Nel capitolo quattro viene descritta la seconda parte del progetto, realizzata in modo autonomo, che vuole sviluppare una libreria che permetta la gestione degli eventi e degli input in modo astratto e indipendente dal dispositivo utilizzato. A livello di analisi vengono identificate le diverse tipologie di input, le interazioni e le funzionalità del SDK di Meta 2 e viene contestualizzata la posizione di questo componente all'interno della infrastruttura MiRAgE.

Nella sezione di progettazione viene identificata la gerarchia degli eventi sviluppata, viene affrontata la rilevazione di tali eventi su Meta 2 sottolineando i limiti del dispositivo e viene descritto il sistema di comunicazione verso MiRAgE.

Capitolo 1

Augmented e Mixed Reality

Un sistema di realtà aumentata arricchisce il mondo reale con oggetti virtuali che coesistono e condividono lo stesso spazio fisico con oggetti reali. In particolare un sistema di realtà aumentata deve [RA01]:

- combinare oggetti reali e virtuali nello stesso ambiente reale
- essere interattivo e real-time
- permettere di allineare oggetti reali e virtuali l'un l'altro

Questa definizione di AR non è ristretta ad un tecnologia in particolare in quanto potenzialmente può essere applicata ad ogni senso e percezione dell'uomo come l'udito, l'olfatto o il tatto. La definizione di realtà aumentata è comunque ancora fonte di confusione tra i ricercatori in quanto i suoi confini non sono ben definiti.

Milgram[PM94] definisce uno spazio continuo che unisce ambiente reale e virtuale, in cui AR è un sottoinsieme della più generale mixed reality MR [1.1]. Sia nella virtualità aumentata AV, in cui gli oggetti fisici vengono aggiunti a quelli virtuali, che nella realtà virtuale VR l'ambiente circostante è virtuale mentre nella AR è il mondo fisico.

Figura 1.1: Milgram's reality–virtuality continuum.

1.1 Mixed Reality

La mixed reality MR [Ped17, p. 288] può essere vista anche come una estensione della AR che non si limita alla sovrapposizione di oggetti virtuali nel mondo fisico ma che cerca di integrarli in modo realistico e consistente [1.2]. Questo può implicare diverse problematiche da gestire ad esempio la corretta e consistente illuminazione degli oggetti virtuali oppure il loro posizionamento sopra o dietro agli oggetti fisici. Per rendere l'esperienza migliore possibile agli utenti è necessario, inoltre, definire e sviluppare modelli di interazione con gli oggetti virtuali innovativi e il più possibile vicini a quelli tradizionali.

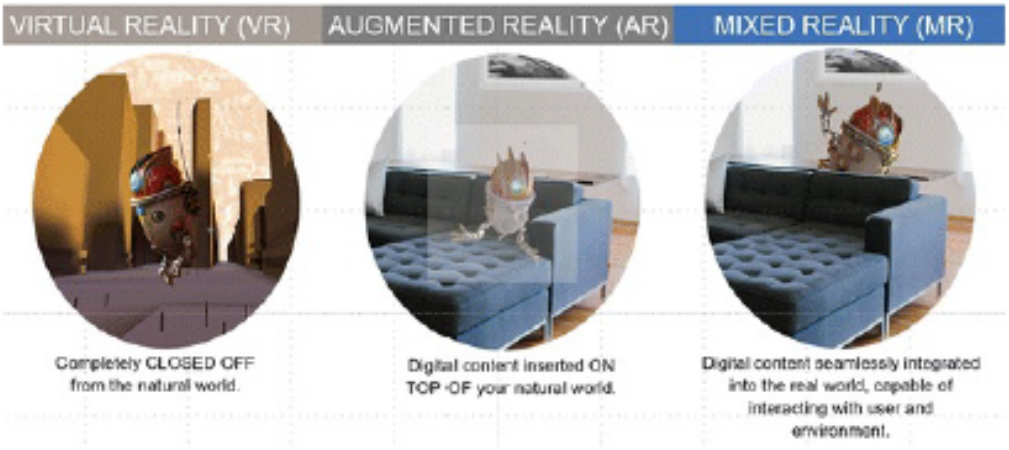

Figura 1.2: Differenze tra realtà virtuale, aumentata e mixata.

1.2 Dispositivi per realtà aumentata

La realtà aumentata AR è un concetto indipendente dalla tecnologia e dal contesto applicativo; in particolare può essere utile in contesti molto diversi: a livello consumatore, commerciale o industriale-scientifico e può essere fruita attraverso dispositivi per la visualizzazione generici o dedicati.

Una tassonomia [1.3] [Ped17, p. 84] può iniziare con una suddivisione tra dispositivi indossabili, definiti anche Head-Mounted Displays (HMD) e non indossabili. In quest'ultima categoria troviamo i dispositivi più generici che vanno da quelli mobili, come smartphone e tablet, a quelli stazionari come TV e PC. Nella categoria dei dispositivi indossabili ci sono i dispositivi dedicati alla realtà aumentata tra cui:

- **Helmets**: sono dispositivi indossabili che ricoprono gran parte della testa e delle orecchie. Un esempio sono i caschi da moto che mostrano le informazioni sul veicolo durante la guida.
- **Headset**: sono dispositivi indossabili di dimensioni ridotte e meno invasivi rispetto agli Helmets. Figurano in questa categoria la maggioranza dei prodotti commerciali come visori e smart glasses.

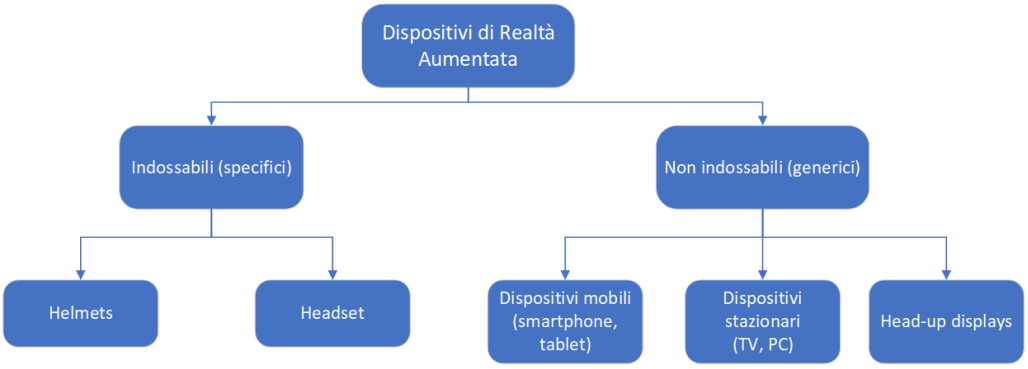

Figura 1.3: Tassonomia dei dispositivi per realtà aumentata.

Gli headset sono la categoria principale per quanto riguarda i dispositivi per realtà aumentata e sono la parte di interesse di questa trattazione. Sul mercato ci sono diversi modelli con caratteristiche diverse.

1.2.1 Microsoft HoloLens

I Microsoft HoloLens [1.4] sono senza dubbio i dispositivi con il più alto profilo nel campo della realtà aumentata. Sono dispositivi non cablati con autonomia della batteria di circa 3 ore. Non essendo collegati ad un computer contengono al loro interno un coprocessore olografico chiamato HPU (Microsoft Holografic Processing Unit). L'HPU riceve informazioni dalla inertial measurement unit (IMU) che comprende accelerometri, giroscopi e magnetometri e le combina con il tracking della testa e dell'ambiente per mostrare correttamente gli oggetti virtuali. Si occupa inoltre del riconoscimento vocale e delle gesture.

Attraverso i dati percepiti dai sensori gli HoloLens riescono a costruire continuamente ed in tempo reale la mesh 3D dell'ambiente che li circonda e integrano così in modo realistico oggetti virtuali e oggetti fisici.

Microsoft fornisce una piattaforma di supporto per gli sviluppatori chiamata "Microsoft Mixed Reality" che è direttamente integrata nel loro sistema operativo Windows 10.

Figura 1.4: Microsoft HoloLens.

1.2.2 Meta 2

I Meta 2 [1.5] sono stati sviluppati da Meta, una azienda della Silicon Valley che ha lanciato questo prodotto sul mercato nel 2017 dopo aver ottenuto un grande successo in una campagna su Kickstarter.

Sono costruiti intorno all'idea di essere un dispositivo sostitutivo ai monitor tradizionali e per questo sono cablati (thethered). Devono essere collegati alla scheda video del PC che si occupa di processare i dati e le immagini ottenuti dai sensori.

Possiedono un largo field-of-view (FOV) che può raggiungere i 90°, a differenza di Microsoft HoloLens (che si ferma a solo 35°), che permette una esperienza immersiva all'utente.

Sono in grado di ricostruire l'ambiente circostante e forniscono delle interazioni per muovere oggetti virtuali in modo naturale e intuitivo.

Sono i dispositivi utilizzati nel progetto e per questo verranno analizzati più nei dettagli in una apposita sezione.

Figura 1.5: Dispositivo Meta 2.

1.2.3 Magic Leap One

Magic Leap One [1.6] è un dispositivo in sviluppo, dell'omonima azienda, che verrà portato sul mercato nel corso di questo anno.

Pur non essendo ancora in commercio ha creato un grande interesse nella comunità per le sue particolarità e differenze rispetto agli altri dispositivi. In un certo senso, Magic Leap One è un ponte tra HoloLens e Meta 2. L'headset è cablato ma diversamente da Meta 2 è collegato ad un piccolo computer, che viene portato dall'utente all'altezza dell'anca, che si occupa di processare e gestire i dati principali.Seguendo questa filosofia si ottiene un headset dalle dimensioni ridotte rispetto alla concorrenza che risulta esteticamente apprezzabile e più comodo.

Un'altra caratteristica molto particolare di Magic Leap One, trapelata dalle prime presentazioni, è la presenza di un dispositivo fisico di input impugnabile e utilizzabile dall'utente.

Figura 1.6: Magic Leap One.

1.3 Software per realtà aumentata

Diverse aziende offrono strumenti per aiutare gli sviluppatori nello sviluppo di applicazioni ben funzionanti in ambienti di realtà aumentata.

Di seguito verrà fatta una breve panoramica su alcune tra le principali piattaforme e tra i principali strumenti di sviluppo utilizzabili.

1.3.1 Piattaforme software

Vuforia

Vuforia [Ped17, p. 255] è la più conosciuta infrastruttura software per realtà aumentata. Dal 2016 è direttamente integrata con il motore grafico cross-platform Unity 3D che permettere di sviluppare sulla maggior parte delle piattaforme software e hardware.

Vuforia permettere di piazzare un contenuto 3D virtuale in corrispondenza ad un marker nell'ambiente fisico grazie alle funzionalità di visione artificiale con le quali riconosce gli oggetti e ricostruisce l'ambiente. Oltre ai classici marker, Vuforia, è in grado di riconoscere:

- **VuMarks** [1.7]: sono particolari marker offerti da Vuforia. Possono avere un design completamente personalizzabile ma allo stesso tempo permettono di codificare informazioni e funzionare come AR target su cui piazzare il contenuto virtuale.

- **Image targets** [1.8]: sono immagini arbitrarie che possono essere create tramite lo strumento Vuforia Target Manager. Per riconoscerle Vuforia ricava le caratteristiche peculiari dell'immagine e le confronta con quelle contenute nel suo database.

- **Object targets** [1.9]: sono rappresentazioni 3D ottenute scansionando un oggetto fisico utilizzando lo strumento Vuforia Object Scanner. In modo simile alle immagini, l'oggetto deve essere inserito all'interno del database tramite Vuforia Target Manager. Per funzionare correttamente l'oggetto deve essere opaco, rigido e con poche parti dinamiche.

Vuforia fornisce API per diversi linguaggi e ambienti di sviluppo tra cui: Java, C++ e Unity 3D.

Figura 1.7: Esempi di Vuforia VuMarks.

Figura 1.8: Esempio di Image target.

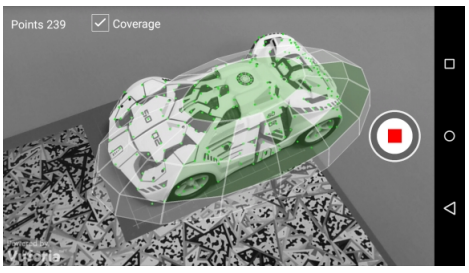

Figura 1.9: Esempio di scansione di un oggetto fisico.

ARToolKit

ARToolKit [Ped17, p. 253] è una libreria open-source per la creazione di applicazioni di realtà aumentata.

ARToolKit utilizza algoritmi di visione artificiale per ottenere la posizione e la rotazione della camera rispetto ad un marker fisico in tempo reale. Le principali funzionalità di ARToolKit sono:

- calcolo della posizione e orientazione della camera
- possibilità di utilizzare marker di forma quadrata
- semplice calibrazione della camera
- velocità di sviluppo per applicazioni AR real-time

ARToolKit supporta la maggior parte dei sistemi operativi e fornisce un plug-in per Unity 3D.

Augment

Augment [Ped17, p. 256], a differenza delle precedenti, è una piattaforma che si occupa nello specifico della visualizzazione dei prodotti in realtà aumentata. L'idea nasce da quelle che sono le difficoltà dell'utente nell'immaginare un prodotto (ad esempio in termini di dimensioni o colori) vedendolo solo attraverso semplici foto descrittive.

Augment offre diversi servizi per la creazione di applicazioni dedicate o per l'integrazione della realtà aumentata sui propri sistemi tra cui:

- **Augment Manager**: permette di caricare e gestire, pubblicamente o privatamente, i propri modelli 3D e marker.
- **Augment Desktop**: permette di costruire modelli 3D e fornisce una grande quantità di colori e materiali.
- **Augment App**: permette di visualizzare i propri modelli 3D e marker, oltre a quelli pubblici, dal proprio smartphone attraverso l'apposita applicazione Augment (Android e iOS).
- **Augment SDK**: permette di integrare la visualizzazione dei modelli 3D nei propri sistemi web e applicazioni. Un esempio può essere l'integrazione in un sistema di e-Commerce.

1.3.2 Tools di sviluppo

OpenCV

OpenCV (Open Source Computer Vision Library) [Ped17, p. 252] è la più famosa libreria open source di computer vision e machine learning. Comprende algoritmi ottimizzati che possono essere utilizzati per riconoscere facce, identificare e tracciare oggetti, classificare gesti, riconoscere marker per realtà aumentata [1.10], etc...

OpenCV è stata progettata per essere cross-platform ed è stata quindi scritta in C e C++. OpenCV può essere eseguita sia su ambienti desktop (Windows, Linux, MacOS) che mobile (Android, iOS) e su infrastrutture hardware diverse.Per rendere la libreria ancora più accessibile sono stati sviluppati diversi wrapper per i linguaggi più conosciuti tra cui Python e Java.

Una parte significativa degli algoritmi di visione artificiale necessita di processare immagini e quindi dal 2010 OpenCV integra l'utilizzo della GPU tramite le primitive CUDA. Con questo modulo, ancora in sviluppo attivo, possono essere eseguiti algoritmi più accurati e sofisticati in tempo reale e consumando meno energia.

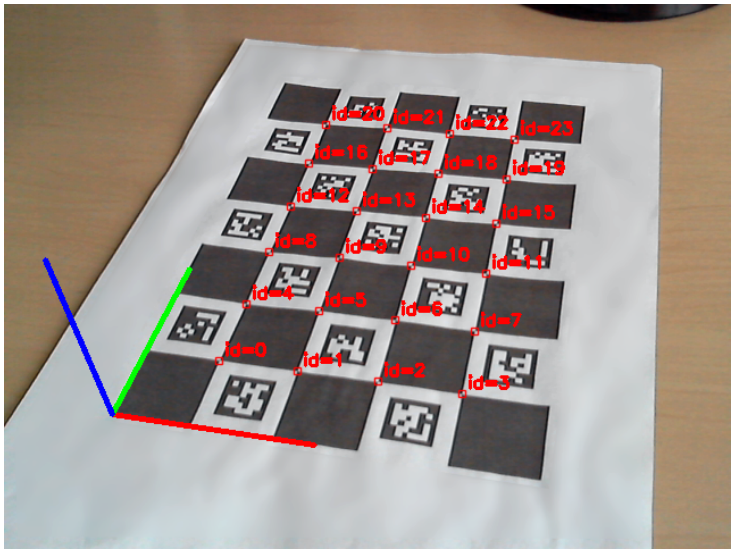

Figura 1.10: Esempio di riconoscimento marker tramite OpenCV.

Unity 3D

Unity 3D è un motore grafico e ambiente di sviluppo integrato (IDE) multipiattaforma per la creazione di contenuti interattivi, in particolare videogiochi 3D.

Unity permette una prototipazione rapida perché gestisce la maggior parte dei componenti necessari per lo sviluppo di videogiochi tra cui: grafica, audio, fisica, interazioni e anche networking.

La sua struttura modulare e il suo largo utilizzo lo rendono il motore grafico più adatto anche in altri contesti applicativi, compresa la realtà aumentata. La maggior parte dei dispositivi e delle infrastrutture AR forniscono SDK specifici per agevolare lo sviluppo delle applicazioni; in particolare per il progetto è stato utilizzato il Meta 2 Unity 3D SDK.

Di seguito vengono riportati e discussi i principali componenti che caratterizzano la struttura del motore.

Assets Un asset [1.11] è una rappresentazione di un oggetto qualsiasi che può essere utilizzata all'interno di un progetto. Un asset può essere costruito all'interno di Unity o attraverso un programma esterno e può essere un qualsiasi file ad esempio un modello 3D, una traccia audio, una immagine, un font, etc... Per questo motivo i file di un progetto Unity sono contenuti in una cartella chiamata *Assets*.

Collezioni di assets possono essere inserite e rese pubbliche, gratuite o a pagamento, nello "Unity Asset Store" che costruisce un package compresso, simile ad un file zip, che può essere scaricato e importato all'interno di un progetto.

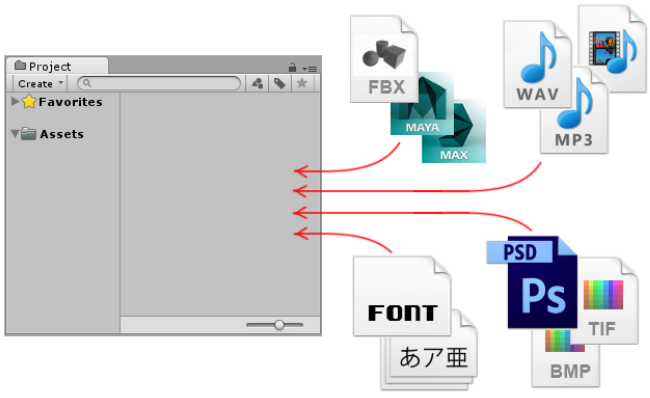

Figura 1.11: Esempi di assets differenti.

Scenes Le scene contengono gli ambienti e i menu del gioco. Un'unica scena può essere pensata come un unico livello in cui possono essere piazzati i contenuti di gioco.

Lo sviluppo in scene permette di separare logicamente l'applicazione, di distribuire il tempo di caricamento e testare le diverse parti individualmente aumentando l'estendibilità, la modularità e la manutenibilità del progetto.

GameObjects Il GameObject GO [1.12] è il concetto fondamentale di Unity. Quando un asset viene inserito in una scena di gioco diventa un particolare GameObject.

Ciascun GameObject può contenere al suo interno diversi Component che descrivono le sue proprietà e i suoi comportamenti; in particolare ciascun GO contiene almeno il componente *Transform* che descrive la sua posizione, rotazione e scala all'interno della scena in coordinate (X,Y,Z). Tramite questa struttura modulare possono essere costruiti GameObject estremamente diversi e complessi semplicemente aggiungendo nuovi componenti.

Unity fornisce alcuni GameObject predefiniti con le principali primitive geometriche come il cubo, la sfera, cilindro, etc.

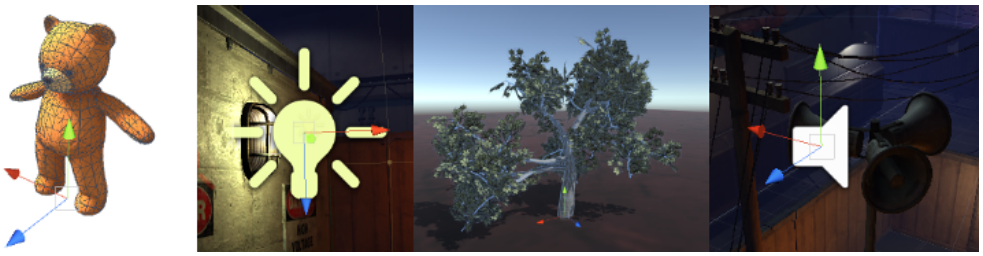

Figura 1.12: Diversi tipi di GameObject: un personaggio, una luce, un albero e una sorgente audio.

Components Un componente è un elemento che può essere agganciato ad un GameObject e che può avere diverse forme e svolgere diversi ruoli.

I componenti più comuni sono costruiti e offerti direttamente da Unity; un esempio è il *Rigidbody* [1.13] che applica ad un oggetto le proprietà necessarie per essere affetto dal motore fisico oppure il *Mesh Renderer* che si occupa di visualizzare l'oggetto in base alla sua forma e posizione.

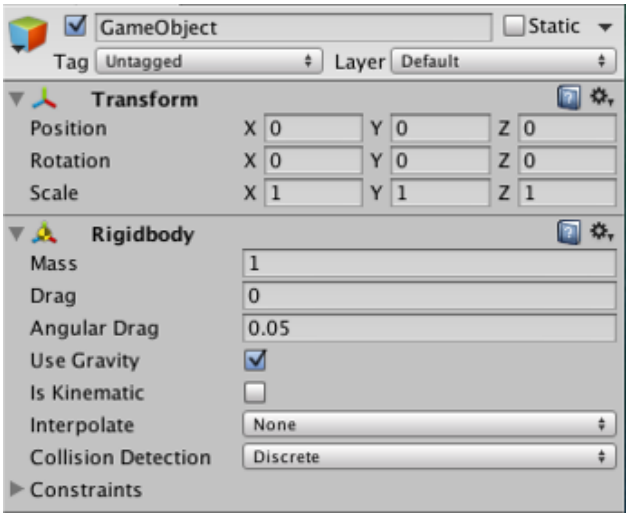

Figura 1.13: Esempio di componente RigidBody.

Scripts I componenti offerti da Unity sono numerosi e versatili ma spesso è necessario agganciare ad un GameObject una funzionalità specifica.

Unity permette di costruire dei componenti personalizzati utilizzando gli scripts. Gli scripts [1.1] permettono di lanciare eventi, modificare proprietà di altri componenti e rispondere agli input ricevuti.

Unity supporta nativamente due linguaggi di programmazione:

- **C#**: linguaggio di programmazione ad oggetti simile a Java o C++ (linguaggio usato nel progetto).
- **UnityScript**: linguaggio di scripting derivato da JavaScript progettato per Unity.

Il flusso di controllo di Unity è basato su un articolato Event-Loop. Unity esegue le funzioni implementate nella classe `MonoBehavior`, che è la classe base dalla quale devono estendere gli scripts. Le due funzioni base sono `Start()`

che viene eseguita solo una volta e può essere utilizzata per l'inizializzazione e `Update()` che viene chiamata ad ogni frame.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class MainPlayer : Monobehaviour
5  {
6
7      void Start()
8      {
9          //Use this for initialization
10     }
11
12     void Update()
13     {
14         //Update is called once per frame
15     }
16
17 }
```

Listing 1.1: Scheletro di script Unity in C#

Prefabs Un prefab permette di memorizzare un GameObject completo con i suoi componenti e le sue proprietà. Il prefab è un template dal quale possono essere generate delle istanze all'interno delle scene. Modificando il prefab vengono aggiornate di conseguenza tutte le istanze prodotte aumentando così la manutenibilità e riusabilità dei GameObject nella scena.

É comunque possibile modificare un oggetto ben specifico per caratterizzarlo rispetto al suo prefab di appartenenza.

Capitolo 2

Augmented World

2.1 Introduzione

Grazie ai recenti sviluppi hardware sono aumentati gli scenari futuristici che coinvolgono gli ambienti intelligenti.

Il concetto di ambiente intelligente nasce dalla definizione di computazione onnipresente e promuove l'idea di un mondo fisico che viene arricchito da elementi computazionali, inseriti all'interno degli oggetti fisici di tutti i giorni, che comunicano tra loro.

La realtà aumentata può essere utilizzata per arricchire questi ambienti intelligenti; si può creare un livello digitale che fornisce informazioni e servizi che possono essere percepiti da dispositivi indossabili; si definisce così un mondo virtuale considerato uno specchio di quello fisico.

Nella visione dei mondi riflessi, gli spazi intelligenti sono modellati in termini di ambienti digitali costruiti intorno al mondo fisico con il quale sono accoppiati, abitati da organizzazioni di agenti software che possono agire e implementare comportamenti complessi. Vi è un accoppiamento tra questi due mondi; una azione nel mondo fisico, effettuata da un umano, causa un cambiamento nelle entità del mondo riflesso e una azione nel mondo virtuale, effettuata da un agente software, può causare una modifica del mondo fisico. In questo scenario viviamo in entrambi i mondi contemporaneamente in quella che è, a tutti gli effetti, una realtà aumentata.

Nella figura [2.1] viene rappresentato il modello di Augmented World [AC] (AW) fortemente basato sul concetto di Mirror World [AR15]. Si può subito notare la distinzione tra mondo fisico e mondo digitale e la loro stretta correlazione; le entità aumentate, definite nel mondo riflesso, possono essere rappresentate e osservate nel mondo reale attraverso gli ologrammi, in modo simmetrico entità reali presenti nel mondo fisico, compresi gli esseri umani,

possono essere accoppiati e rappresentati da entità aumentate nel mondo digitale. L'essere umano può interagire con gli ologrammi e gli oggetti fisici e modificare di conseguenza il loro stato nel mondo riflesso.

L'accoppiamento tra i mondi viene definito anche a livello spaziale; le entità aumentate possiedono la locazione corrispondente nel mondo fisico: questo permette di costruire regioni spaziali che possono essere osservate da agenti software. Gli agenti software sono definiti nel mondo digitale ad un livello concettuale diverso dall'ambiente e possono interagire e percepire entità e regioni di entità aumentate in modo indipendente o cooperativo.

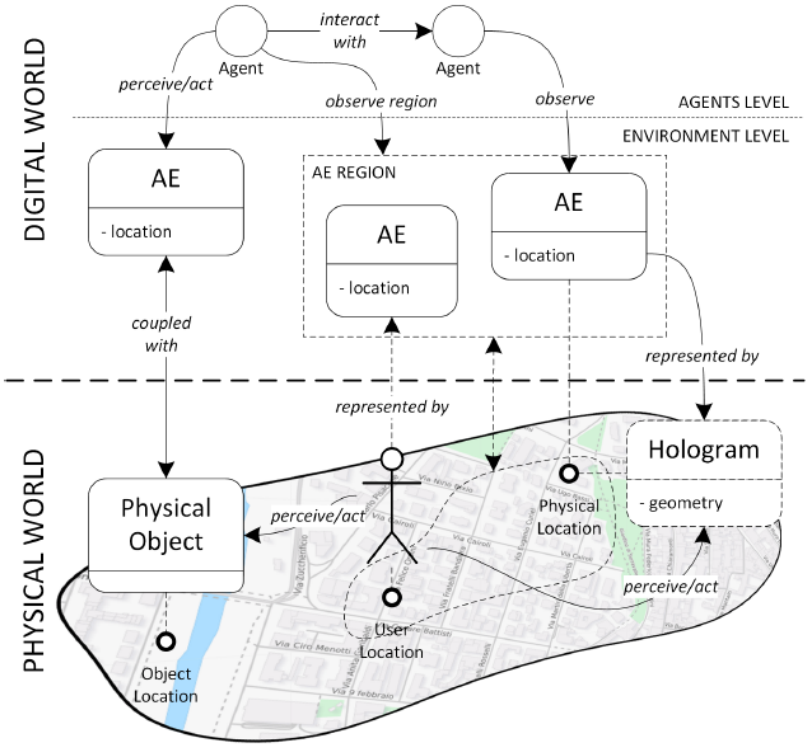

Figura 2.1: Modello Augmented World.

2.2 Modello concettuale

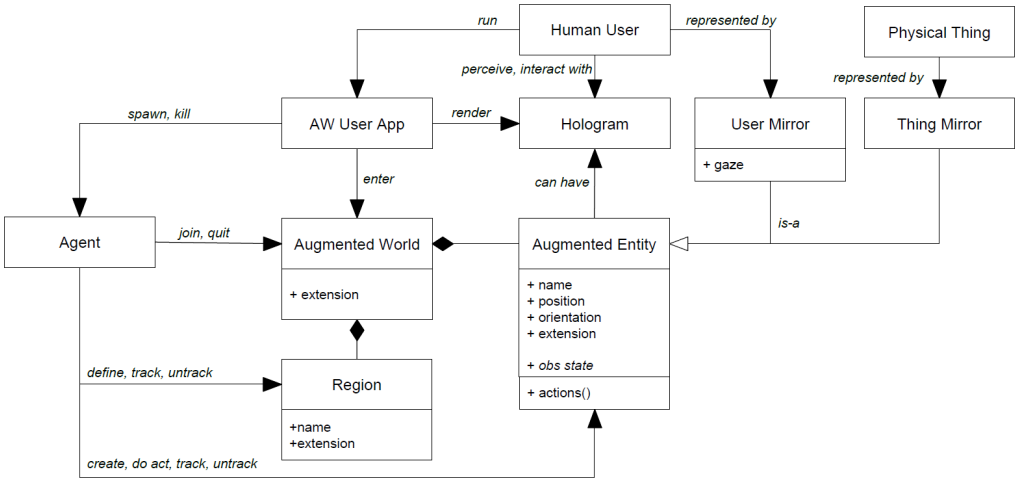

Figura 2.2: Modello concettuale Augmented World AW.

Nel diagramma UML [2.2] viene rappresentato il modello concettuale di un Augmented World (AW) e i suoi principali elementi. Un `Augmented World` è un mondo virtuale accoppiato a livello spaziale ad una specifica regione fisica. É composto da `Augmented Entity` che rappresentano gli oggetti reali nel mondo digitale. Una entità aumentata ha delle proprietà specifiche: posizione, orientamento e estensione definite in relazione alla regione e al sistema di riferimento del mondo aumentato ed espongono uno stato osservabile, in termini di proprietà che possono cambiare nel tempo, e delle azioni.

Le entità aumentate possono essere create, posizionate e spostate dinamicamente dagli `Agents`. Gli agenti per operare all'interno di un mondo aumentato devono effettuare l'operazione di *join* e avviare una sessione; una volta entrati possono *tracciare* entità aumentate, osservando il loro stato e percependo i loro eventi, e agire su di esse attraverso le azioni che rendono disponibili. Oltre a osservare specifiche entità, gli agenti possono tracciare porzioni di spazio fisico modellate dalle `Regioni` e percepire eventi sulla relativa entrata o uscita di entità.

Una entità aumentata può avere un `Hologram` associato che la rappresenta e la rende percepibile da un utente umano. La vista dell'ologramma dipende dal suo stato e dalle capacità e caratteristiche del dispositivo utilizzato ma, indipendentemente dalla rappresentazione, è il componente che permette l'iterazione con l'utente.

L'utente, rappresentato dallo `Human User`, entra all'interno del mondo aumentato attraverso una applicazione (`AW User App`). L'applicazione si occupa di creare l'agente che può agire all'interno del mondo aumentato ed è responsabile della corretta visualizzazione degli ologrammi e dell'aggiornamento del rispettivo `User Mirror` all'interno del sistema fornendo posizione e rotazione dell'utente. Questa applicazione si può occupare inoltre della registrazione degli eventi di input (sguardo, gesti e comandi vocali) che possono riguardare specifici ologrammi (afferrare un ologramma, guardare un ologramma). Questi eventi di input vengono propagati dalle rispettive entità aumentate all'interno dell'infrastruttura agli agenti interessati. Più utenti possono accedere contemporaneamente al mondo aumentato, utilizzando anche applicazioni e dispositivi diversi, e possono osservare e interagire con gli stessi oggetti.

Gli oggetti fisici (`Physical Thing`) possono essere rappresentati da entità aumentate specifiche (`Thing Mirror`) così da essere percepiti e modificabili dagli agenti software.

2.3 MiRAgE: Augmented World framework

MiRAgE (**Mi**xed **R**eality based **Aug**mented **E**nvironments) è un framework che modella e gestisce mondi aumentati basato strettamente sul modello concettuale di Augmented World (AW) mostrato in precedenza [2.2]. MiRAgE fornisce un insieme di funzionalità:

- **Mondi condivisi**: il framework permette l'ingresso di più utenti umani all'interno dello stesso AW che condividono e interagiscono con le stesse entità aumentate.
- **Arricchimento delle entità virtuali e reali**: il framework, oltre ad aumentare la realtà attraverso entità virtuali, è in grado di integrare il concetto di pervasive computing e di arricchire il mondo fisico con elementi con capacità computazionale.
- **Sistema aperto**: il framework rende possibile lo sviluppo di agenti che possono, dinamicamente, entrare e uscire da un AW e modificare le entità al suo interno.
- **Tecnologie di programmazione ad agenti eterogenee**: il framework permette l'utilizzo di tecnologie ad agenti eterogenee per la programmazione degli agenti all'interno del AW. Gli agenti possono essere scritti in Java o tramite linguaggi appositi come ASTRA, Jason, JaCaMo.
- **Supporto a tecnologie eterogenee**: il design del framework permette l'utilizzo di tecnologie di realtà aumentata differenti, sia software (Unity 3D, Vuforia) che hardware (Hololens, Meta2). Il framework permette inoltre l'integrazione a tecnologie Internet of Things (IoT) e Web of Things (WoT).
- **Generalità**: il framework mira ad essere generale per supportare scenari applicativi diversi, inclusi scenari indoor e outdoor.

MiRAgE è progettato con l'obiettivo di nascondere il più possibile i dettagli implementativi per il funzionamento della realtà aumentata. Lo sviluppatore può così focalizzarsi solo sulla progettazione della logica applicativa:

- Design delle entità aumentate (definendone proprietà e azioni)
- Design della geometria degli ologrammi
- Design della logica applicativa degli agenti
- Design della logica applicativa legata alle interazioni dell'utente

2.3.1 Architettura logica

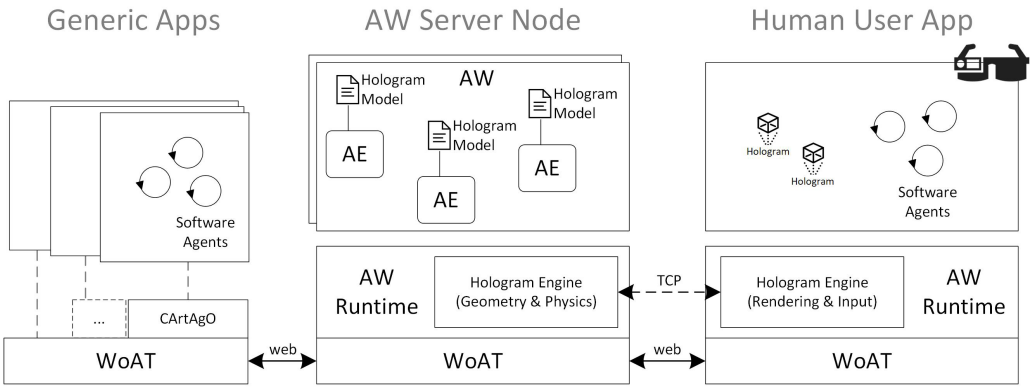

Figura 2.3: Architettura logica di MiRAgE.

Dalla architettura logica di MiRAgE [2.3] si possono riconoscere tre componenti principali:

- **AW-Runtime**: è il componente che esegue le istanze di Augmented World (AW), gestisce le entità aumentate e fornisce una interfaccia comune agli agenti software per l'interazione. AW-Runtime è progettato per essere eseguito su uno nodo (eventualmente su architetture cloud) e per gestire ed eseguire più mondi aumentati contemporaneamente.

- **Hologram Engine**: è il componente che si occupa della visualizzazione degli ologrammi agli utenti. Il compito principale del HE è quello di mantenere la visualizzazione degli ologrammi consistente con il modello concettuale. L'hologram engine viene eseguito sia all'interno del nodo AW che all'interno del dispositivo e le due istanze comunicano direttamente attraverso un canale TCP. L'hologram engine gestisce la fisica (collisioni tra ologrammi, visualizzazione in prospettiva degli ologrammi rispetto ad oggetti fisici e alla distanza) e le interazioni da parte dell'utente (rilevamento dello sguardo e dei gesti).

- **WoAT layer**: Web of Augmented Things (WoAT) è un livello software che garantisce la massima interoperabilità alle applicazioni. Adotta il modello del Web of Things proposto nel contesto IoT e rappresenta, quindi, le entità aumentate come risorse di rete accessibili attraverso una Web REST API. Una applicazione, o semplicemente un agente, può accedere alle entità aumentate attraverso uno specifico indirizzo

(URL) e può effettuare operazioni HTTP: operazioni di GET possono essere utilizzate per ottenere le proprietà osservabili mentre operazioni POST possono essere utilizzate per compiere azioni sulle entità.

2.3.2 Agenti e primitive

Dal punto di vista degli agenti, MiRAgE è una infrastruttura che offre un insieme di primitive che può essere suddivisa in due categorie:

- **Funzionalità predefinite**: insieme prefissato di primitive che permettono ad un agente di: entrare o uscire da un AW, creare e gestire AE, definire e osservare regioni [2.1].
- **Funzionalità specifiche**: insieme delle azioni (*actions*) offerte dalle entità aumentate all'interno del AW. L'agente può utilizzare la primitiva `doAct(aeID,op,args)` per richiamare una determinata operazione (`op`) di una specifica AE (`aeID`). Le azioni possono essere processi a lungo termine e per questo motivo vengono eseguite dalla entità stessa; dal punto di vista dell'agente l'azione è eseguita in modo asincrono e può eventualmente fallire.

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 una AE esistente
trackAE(aeID)	per iniziare ad osservare una AE esistente
stopTrackingAE(aeID)	per smettere di osservare una AE esistente
moveAE(aeID, position, orientation)	per modificare, se permesso, la posizione e l'orientazione di una 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 Meta 2 con MiRAgE

3.1 Introduzione

In questo capitolo viene effettuata l'integrazione tra il dispositivo Meta 2 e l'infrastruttura MiRAgE. Da un lato Meta 2, allo stato attuale, non è utilizzato e non fornisce nessun supporto per realizzare applicazioni multi-utente, dall'altro, MiRAgE è stata utilizzata solamente con dispositivi di visualizzazione generici (smartphone). Risulta, pertanto, molto interessante la possibilità di realizzare applicazioni in cui più utenti condividono e percepiscono lo stesso mondo aumentato con gli stessi oggetti virtuali attraverso questo dispositivo.

Meta 2 fornisce una visualizzazione immersiva all'utente completamente diversa e molto più realistica rispetto ai dispositivi tradizionali. L'utilizzo di questo dispositivo all'interno di applicazioni multi-utente potrebbe avere un grande impatto, soprattutto in ambito lavorativo, aumentando notevolmente le possibilità di interazione e lo scambio di contenuti e informazioni tra le persone.

Inoltre l'integrazione con MiRAgE permetterebbe, a livello teorico, l'utilizzo di questa tecnologia di realtà aumentata in combinazione con altri dispositivi delineando scenari in cui le persone partecipano e interagiscono all'interno dello stesso mondo aumentato attraverso dispositivi completamente differenti.

3.2 Approfondimento Meta 2

In questa sezione vengono analizzate le specifiche dei Meta 2 e vengono approfondite alcune tra le principali funzionalità offerte agli sviluppatori dal SDK 2.7.0 in ambiente Unity 3D.

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:

Sistema Operativo: Windows 10 (64-bit) (MacOS verrà supportato in futuro)
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
Motore grafico: Unity 5.6 o superiore
Hard disk: 2 GB di spazio libero

3.2.2 Unity SDK 2.7.0

Meta 2 fornisce numerosi componenti e funzionalità per aiutare gli sviluppatori nell'implementazione di applicazioni in realtà aumentata. In particolare all'interno dello Unity SDK 2.7.0, importabile direttamente nel progetto, vi sono diversi prefab utilizzabili e diverse scene che illustrano le principali funzionalità offerte.

Sensori e Camera

Meta camera Rig Prefab che rappresenta la camera del dispositivo e si occupa del **rendering** e del **tracking** nell'ambiente Unity. Per default la camera è posizionata a (0,0,0) quando viene eseguita la scena. Per cambiarne la posizione è necessario inglobarla all'interno di un nuovo GameObject. Le coordinate Unity vengono impostate dopo la calibrazione del visore. In seguito vengono elencati i principali Script contenuti all'interno del prefab.

Tracking: SLAM Localizer Si occupa della localizzazione e della mappatura simultanei (SLAM) [3.1] dell'ambiente fisico utilizzando solamente i sensori presenti nel dispositivo.

Lo script genera diversi eventi agli eventuali GameObject iscritti per comunicare lo stato della procedura (es. `OnSlamMappingComplete()`). Il sistema SLAM di Meta2 fornisce misurazioni di tipo IMU (inertial measurment units) tramite accelerometri e giroscopi che permettono di rilevare posizione e rotazione del dispositivo. Il visore può lavorare in due modalità differenti:

- **Limited tracking (3DOF)**: utilizzando solo 3 gradi di libertà; così facendo viene utilizzata solo la rotazione ottenuta dai giroscopi. Gli oggetti virtuali vengono quindi visualizzati tutti alla stessa distanza.
- **Full tracking (6DOF)**: utilizzando tutti e 6 i gradi di libertà; vengono utilizzati i giroscopi per ottenere la rotazione e gli accelerometri per la posizione. Gli oggetti virtuali vengono visualizzati a distanza diversa.

Figura 3.1: Procedura SLAM.

Rendering: Compositor Si occupa del rendering degli oggetti virtuali nella scena. Permette di aggiustare i parametri Warp2D/3D per ridurre la latenza di rendering quando vengono fatti movimenti bruschi.

Permette di abilitare/disabilitare il **Depth Occlusion** (Shader occlusion) [3.2] che nasconde/visualizza gli oggetti virtuali in base all'ambiente fisico rilevato dai sensori di profondità.

Virtual Webcam Permette di simulare, e registrare, su PC quello che si vede dal dispositivo, compresi gli oggetti virtuali. La camera del dispositivo fornisce due flussi video:

- **Meta2 Webcam + Holographic Overlay**: il video è composto sia dai dati ottenuti dalla camera RGB sia dagli oggetti virtuali visibili attraverso il dispositivo.
- **Meta2 Webcam**: il video è composto unicamente dai dati ottenuti dalla camera.

Dalla versione 2.7.0 la funzionalità supporta anche il depth occlusion.

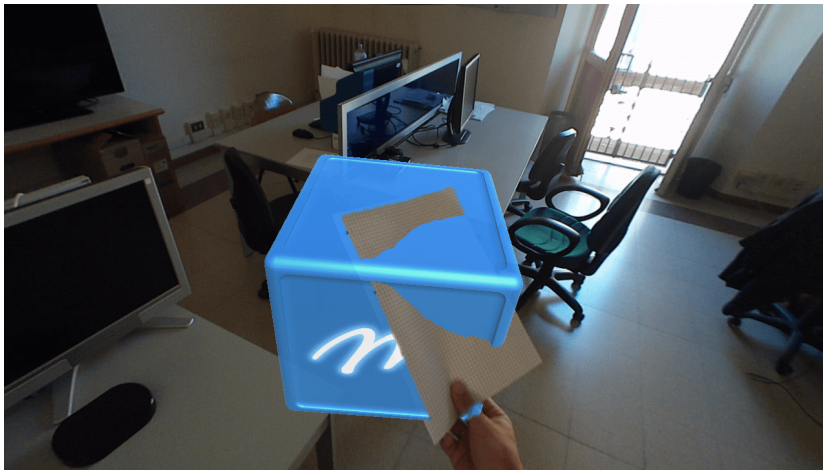

Figura 3.2: Funzione depth occlusion.

Environment Configuration Si occupa della creazione di una mesh 3D dell'ambiente fisico [3.3] utilizzando il sensore di profondità. Con i dati ottenuti dalla ricostruzione può essere creato un GameObject Unity che può interagire con gli altri oggetti virtuali; ad esempio si può creare una palla virtuale che rimbalza su un tavolo reale.

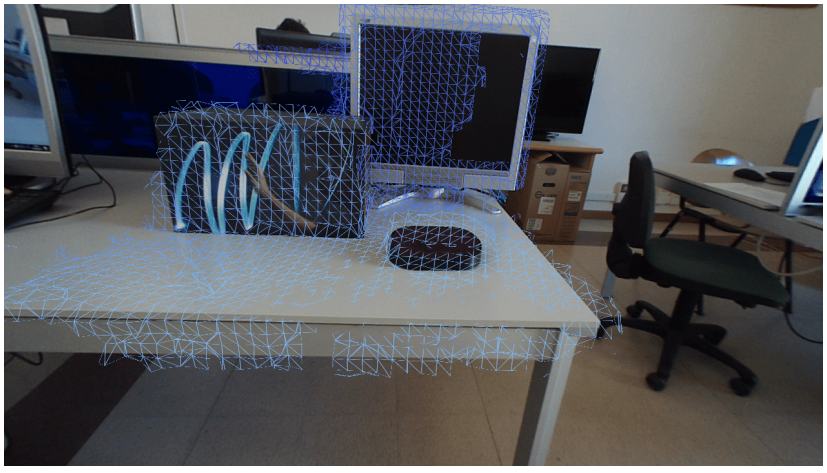

Figura 3.3: Ricostruzione mesh dell'ambiente.

Meta Locking Meta Locking è uno script che permette di bloccare un oggetto relativamente allo sguardo dell'utente.

Viene utilizzato per creare degli Head-up-displays (HUD); ad esempio si può bloccare un oggetto nella parte alta del campo visivo che rimane immobile e di supporto all'utente.

Interazioni

Meta 2 fornisce diversi tipi di interazione per operare con oggetti virtuali.

I GameObjects con i quali si può interagire devono avere agganciati un Rigidbody e un Box Collider.

Meta Hands Prefab che si occupa della gestione delle interazioni attraverso le mani. Incapsula lo script **HandsProvider** [3.4] che inizializza il template delle mani (facendo visualizzare un pallino azzurro all'utente) e possiede eventi base ai quali ci si può agganciare. Questi eventi vengono catturati da `CenterHandFeature : HandFeature`.

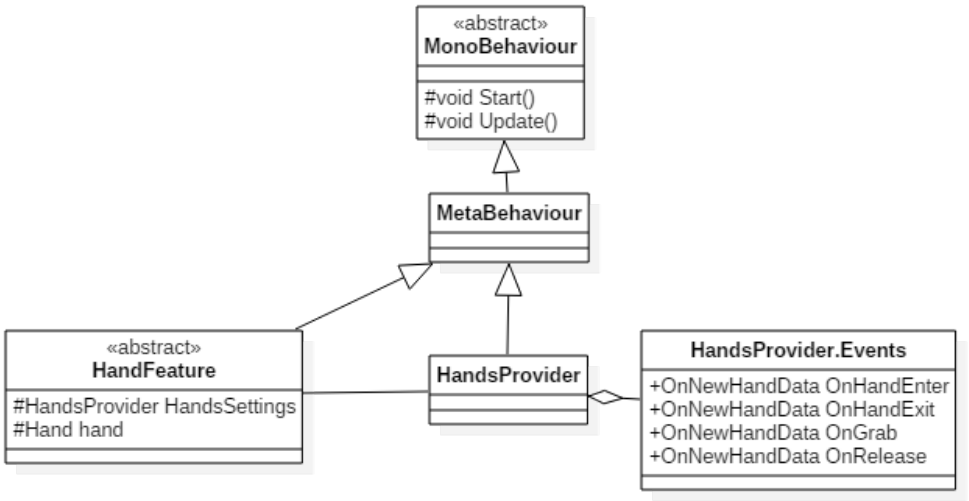

Figura 3.4: Diagramma delle classi rappresentante il componente HandsProvider, gli eventi che genera e il suo utilizzo da parte dalla classe HandFeature.

Le interazioni che possono essere agganciate come componenti ai GameObject derivano dalla classe base astratta `Interation` [3.5] che imple-

menta l'interfaccia `IInteractibleObject` ed espone diversi metodi astratti da implementare per la gestione delle interazioni. Vi sono diverse interazioni built-in offerte dal SDK, tutte derivate dalla classe base `Interaction`, ad esempio:

- GrabInteraction
- TwoHandGrabRotateInteraction
- TwoHandGrabScaleInteraction

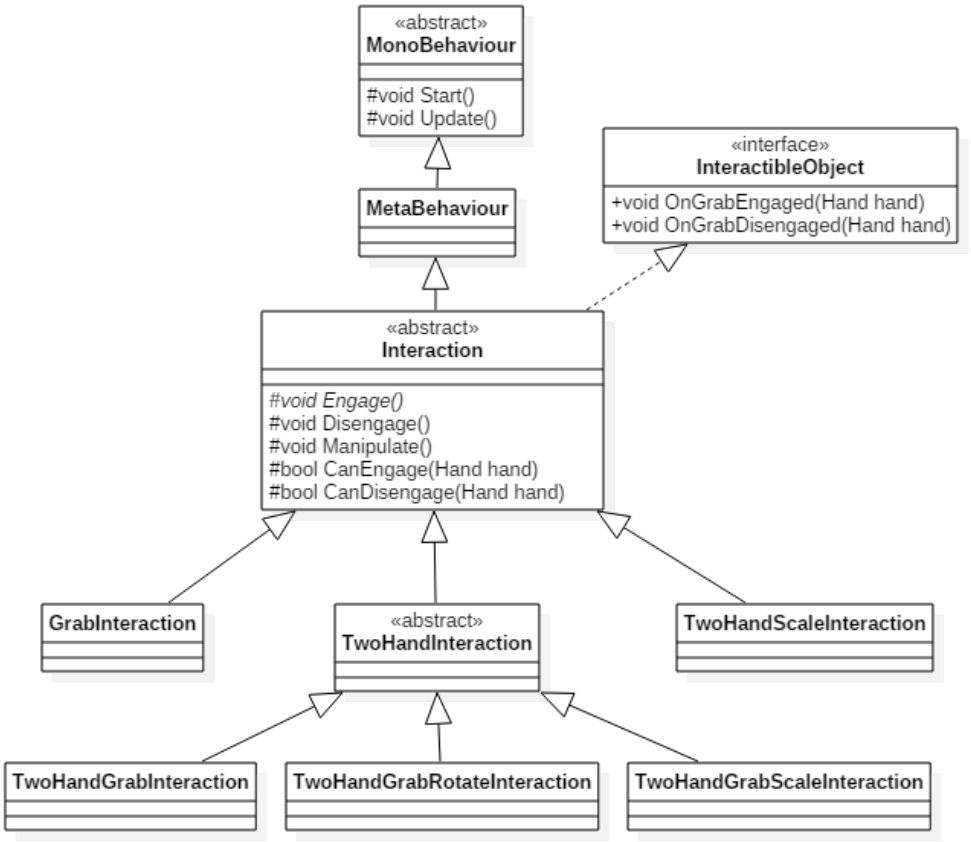

Figura 3.5: Diagramma delle classi rappresentante la gerarchia di interazioni fornite da Meta 2.

Le interazioni derivate dalla classe `Interaction` contengono al loro interno le informazioni relative alle mani che stanno toccando (*hovering hands*)

o tenendo (*grabbing hands*) un oggetto. Queste informazioni sono contenute nella classe astratta `HandFeature` [3.6] che incapsula la mano (`Hand`) e le sue proprietà (`HandsData`). Tra le informazioni che si possono trovare all'interno della classe `HandsData` vi sono:

- posizione corrente del palmo e del dito della mano.
- informazioni sulla presenza della mano (rilevata o non rilevata).
- informazioni sullo stato della mano (aperta o chiusa).
- informazioni sul tipo di mano (destra o sinistra).

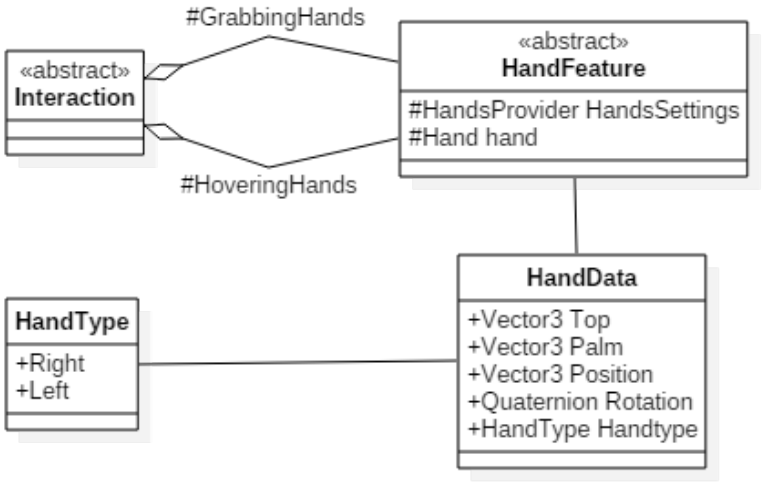

Figura 3.6: Diagramma delle classi rappresentante la relazione tra Interaction e HandFeature e le proprietà disponibili all'interno di HandData.

Meta Mouse Inserendo il prefab **MetaInputModule** nella scena è possibile simulare il mouse all'interno del nostro ambiente 3D. Per interagire con i GameObject si possono semplicemente agganciare alcuni script:

- **DragTranslate**: permette di traslare un oggetto.
- **DragRotate**: permette di ruotare un oggetto.
- **DragScale**: permette di scalare un oggetto.

Gli script possono essere agganciati a diversi tasti del mouse: tasto sinistro, tasto destro o tasto centrale.

Meta Gaze Permette di invocare un metodo ad un oggetto nel caso lo stessimo guardando grazie ai Raycast emessi dal centro della camera. L'oggetto deve possedere necessariamente un Box Collider agganciato per rilevare la collisione. E' sufficiente implementare i due metodi `OnGazeStart()` e `OnGazeEnd()` delle due rispettive interfacce `IGazeStartEvent` e `IGazeEndEvent` [3.1].

```
1  using Meta;
2
3  public class GazeExample : Monobehaviour,
       IGazeStartEvent, IGazeEndEvent
4  {
5      public void OnGazeStart()
6      {
7          /* This method is called when the user
8          begins to gaze at this GameObject */
9      }
10
11     public void OnGazeEnd()
12     {
13         /* This method is called when the user
14         looks away from  this GameObject */
15     }
16 }
```

Listing 3.1: Esempio di classe che implementa la funzione di gaze.

Meta Canvas Il MetaCanvas prefab è uno strumento per la prototipazione rapida di interfacce utente 2D che supporta interazioni sia tramite Meta Hands che Meta Mouse.

3.3 Supporto per mondi condivisi in MiRAgE mediante Meta 2

3.3.1 Analisi

Uno dei problemi principali nelle applicazioni di realtà aumentata è il problema della registrazione [Azu]. Gli oggetti del mondo reale e quelli del mondo virtuale devono essere correttamente allineati, altrimenti l'illusione che i due mondi co-esistano sarebbe compromessa. Il metodi di registrazione possono essere classificati[Yan15] in: registrazione basata su sensori o registrazione basata su visione artificiale.

Registrazione basata su sensori

Utilizza i sensori, esclusi quelli di visione, per ottenere le informazioni di registrazione. Questi sensori includono sensori meccanici, sensori magnetici, GPS, sensori ad ultrasuoni e sensori inerziali. Questo tipo di registrazione è semplice da implementare in quanto questi sensori sono solitamente integrati in tutti i dispositivi AR e sono facilmente accessibili dai programmatori attraverso apposite interfacce. Inoltre è una tecnica a basso costo energetico e computazionale. Tra gli svantaggi ci sono la bassa precisione e la difficoltà di rilevamento dello sguardo dell'utente.

Solitamente viene utilizzata per correggere e assistere tecniche di registrazione basate su visione artificiale.

Registrazione basata su marker (visione artificiale)

Le tecniche di registrazione basate su visione artificiale sono le più utilizzate. Vengono utilizzati dei punti di riferimento, nel mondo reale, per determinare la posizione e la rotazione rispetto a quest'ultimi. Il modo più robusto per identificare questi punti di riferimento è l'utilizzo dei fiducial marker [3.7]. I fiducial marker sono delle immagini particolari che sono facilmente riconoscibili attraverso algoritmi di visione artificiale; i più comuni sono di forma quadrata con un pattern asimmetrico bianco su sfondo nero.

L'utilizzo dei marker garantisce una alta precisione (nell'ordine del centimetro) e una bassa complessità computazionale rispetto a registrazioni marker-less. Risultano, però, vulnerabili a condizioni di luce e movimento e devono essere esplicitamente posizionati nell'ambiente fisico.

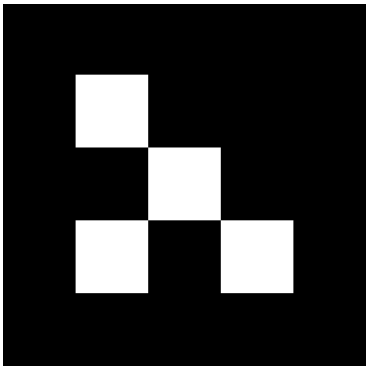

Figura 3.7: Esempio di fiducial marker.

Registrazione senza marker (visione artificiale)

La registrazione marker-less utilizza algoritmi di visione artificiale più complessi; in generale vengono estratte informazioni, quali punti, bordi, angoli e segmenti direttamente dagli oggetti rilevati nelle immagini della camera. Esistono due approcci per effettuare una registrazione marker-less:

- **Model-based tracking**: viene stimata la posizione della camera confrontando le caratteristiche dei diversi frame da quello iniziale.
- **Move-matching tracking**: la posizione della camera viene stimata in base al movimento delle caratteristiche dell'immagine tra un frame e l'altro. Il Simultaneous Localization and Mapping (SLAM), è solitamente utilizzato in questo tipo di approccio. SLAM genera una mappa 3D dell'ambiente e stima, contemporaneamente, la posizione dell'utente.

Il grande vantaggio di questo tipo di registrazione è l'indipendenza dai marker che aumenta notevolmente la flessibilità. D'altra parte queste tecniche sono ancora in sviluppo e sono lontane dal raggiungere il pieno potenziale; in particolare non hanno gli stessi gradi di precisione e robustezza delle soluzioni con marker e sono computazionalmente molto più onerose.

Problematiche e considerazioni iniziali

Il problema principale da risolvere per effettuare l'integrazione tra il dispositivo Meta 2 e MiRAgE deriva dalle due differenti filosofie. Da una parte Meta 2 nasce con una filosofia di tipo marker-less che utilizza la tecnica SLAM per effettuare la registrazione e ottenere la posizione della camera; d'altra parte MiRAgE utilizza, nelle versioni presentate fin ora, i marker con Vuforia.

Il riconoscimento di marker su Meta 2 non è attualmente supportato e l'integrazione con librerie di visione artificiale non è banale. Per questo motivo è necessario progettare un sistema di registrazione manuale effettuabile dall'utente. Ovviamente una registrazione demandata all'utente ha una precisione nettamente inferiore rispetto a una registrazione effettuata tramite sensori o visione artificiale pertanto sarebbe opportuno, nel momento in cui sia disponibile un supporto ufficiale ai marker da parte di Meta 2, sostituire la procedura.

MiRAgE è per definizione una infrastruttura di mondi aumentati condivisi e nella prima versione utilizza l'infrastruttura multiplayer offerta da Unity3D che fornisce un'astrazione ad alto livello High Level API (HLAPI) per lo sviluppo di applicazioni multi-giocatore. Meta 2 non offre ufficialmente supporto a questa infrastruttura ed è quindi necessario utilizzare un approccio alternativo: ogni istanza di Unity su Meta 2 è totalmente indipendente e deve comunicare in modo esplicito le modifiche con MiRAgE tenendo conto della sua posizione e rotazione rispetto al mondo aumentato.

Trasformazione tra sistemi di riferimento

La registrazione con il mondo aumentato definito da MiRAgE equivale alla definizione di un ulteriore sistema di riferimento; diviene necessario quindi effettuare delle trasformazioni di coordinate tra i due sistemi.

Ogni vettore w dello spazio tridimensionale può essere rappresentato univocamente utilizzando tre vettori linearmente indipendenti. I tre vettori in questione definiscono una base dello spazio tridimensionale.

Dati n vettori $v_1, v_2, \ldots, v_n$ di uno spazio vettoriale V, si dice che essi sono linearmente indipendenti se:

$$\sum_{i=1}^{n} a_i v_i = a_1 v_1 + a_2 v_2 + ... + a_n v_n = 0$$

è verificata solo se gli elementi $a_1, a_2, \ldots a_n$ sono tutti uguali a zero.

Data una base $B = \{v_1, v_2, v_3\}$, un qualsiasi vettore w può essere definito nel seguente modo:

$$w = a_1 v_1 + a_2 v_2 + a_3 v_3 \tag{3.1}$$

in forma matriciale:

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad w = a^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

Cambio di sistemi di coordinate In uno spazio vettoriale, date due basi $B_1 = \{v_1, v_2, v_3\}$ e $B_2 = \{u_1, u_2, u_3\}$, possiamo esprimere una in termini dell'altra:

$$\begin{aligned} u_1 &= \gamma_{11} v_1 + \gamma_{12} v_2 + \gamma_{13} v_3 \\ u_2 &= \gamma_{21} v_1 + \gamma_{22} v_2 + \gamma_{23} v_3 \\ u_3 &= \gamma_{31} v_1 + \gamma_{32} v_2 + \gamma_{33} v_3 \end{aligned} \tag{3.2}$$

Questo sistema di equazioni definisce una matrice M 3x3 del cambiamento di base:

$$M = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \qquad \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = M^{-1} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

Dato un vettore w definito nel sistema di coordinate descritto dalla base B_1:

$$w = a_1 v_1 + a_2 v_2 + a_3 v_3 \tag{3.3}$$

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad w = a^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

si può ottenere la sua rappresentazione (coefficienti) b nel sistema di coordinate con base B_2 usando la matrice $(M^T)^{-1}$ e viceversa utilizzando M^T:

$$w' = b_1 u_1 + b_2 u_2 + b_3 u_3 \tag{3.4}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad a = M^T b \qquad b = (M^T)^{-1} a$$

É importante notare che nei sistemi di coordinate si parla di vettori e non di punti. Questi due elementi hanno, concettualmente, significati molto diversi:

- **Punto**: è una entità in cui l'unico attributo è la sua posizione rispetto ad un sistema di riferimento.
- **Vettore**: è una entità in cui gli attributi sono lunghezza e direzione.

Questi cambi di base modificano solo vettori lasciano l'origine immutata; in altre parole rappresentano solo rotazioni e scalature. Per rappresentare quindi la posizione di un punto (e le traslazioni) occorre definire anche un punto di riferimento, l'origine [3.8].

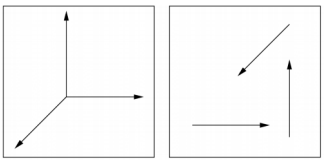

Figura 3.8: A sinistra un sistema di riferimento con origine, a destra gli stessi vettori senza origine.

Cambio di sistemi di riferimento (frame) Per definire un sistema di riferimento (frame F) sono necessari quindi un punto di origine P_0 e una base. In un frame è possibile rappresentare in modo univoco un qualsiasi punto P dello spazio tridimensionale:

$$\begin{aligned} F &= \{v_1, v_2, v_3, P_0\} \\ P &= a_1 v_1 + a_2 v_2 + a_3 v_3 + P_0 \end{aligned} \tag{3.5}$$

Per descrivere le trasformazioni di punti e vettori in forma matriciale si utilizza l'artificio matematico delle **coordinate omogenee**:

- un punto P è rappresentato dalla matrice colonna p:

$$p = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ 1 \end{bmatrix}$$

  dalla equazione (3.3):

$$P = p^T F^T = \begin{bmatrix} a_1 & a_2 & a_3 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = a_1 v_1 + a_2 v_2 + a_3 v_3 + P_0$$

- un vettore w è rappresentato dalla matrice colonna a:

$$a = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ 0 \end{bmatrix}$$

  dalla equazione (3.3):

$$w = a^T F^T = \begin{bmatrix} \sigma_1 & \sigma_2 & \sigma_3 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = a_1 v_1 + a_2 v_2 + a_3 v_3$$

Così facendo vengono rappresentati in modo univoco sia i punti che i vettori e le loro trasformazioni; se la trasformazione riguarda un punto P viene

aggiunta la eventuale traslazione dall'origine P_0 mentre se riguarda un vettore w no.

In modo analogo al caso precedente, dati due sistemi di riferimento $F_1 = \{v_1, v_2, v_3, P_0\}$ e $F_2 = \{u_1, u_2, u_3, Q_0\}$, possiamo esprimere uno in termini dell'altro:

$$\begin{aligned} u_1 &= \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3 \\ u_2 &= \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3 \\ u_3 &= \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3 \\ Q_0 &= \gamma_{41}v_1 + \gamma_{42}v_2 + \gamma_{43}v_3 + P_0 \end{aligned} \tag{3.6}$$

Questo sistema di equazioni definisce una matrice M 4x4 del cambiamento di frame:

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

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

Date le due rappresentazioni (coefficienti dei vettori) a, b rispetto ai due frame F_1 e F_2 si può ricavare a utilizzando la matrice M^T:

$$b^T \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix} = b^T M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = a^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = b^T M = a^T \quad => \quad a = M^T b$$

in modo analogo si può ricavare b utilizzando la matrice inversa $(M^T)^{-1}$:

$$b = (M^T)^{-1} a$$

3.3.2 Progettazione e sviluppo

Processo di registrazione

Il processo di registrazione sviluppato è di tipo manuale ed è demandato all'utente all'avvio della applicazione.

Idea Per sviluppare un processo di registrazione manuale si possono sfruttare i sensori di profondità dei Meta 2 e la funzionalità di depth occlusion. Sfruttando questa possibilità si può creare un oggetto, che potrebbe assomigliare ad un mirino, ad una distanza prefissata (es. $p_{(x,y,z)} = (0, 0, 0.5)$) e richiedere all'utente di allinearlo con quelli che sono i punti di allineamento collocati precedentemente e appositamente nel luogo fisico. Per la costruzione bastano 3 punti presi in uno specifico ordine:

1. Origine
2. Asse X
3. Asse Y

I vettori degli assi X e Y si ottengono sottraendo l'origine al punto rilevato:

$$vector = point - origin$$

Per definire il sistema di riferimento si può, a questo punto, calcolare il vettore dell'asse Z semplicemente effettuando il prodotto vettoriale tra i due vettori X e Y.

Inizialmente era stata sviluppata una procedura, più semplice e veloce, nella quale l'utente andava a definire il punto di origine e la sola direzione degli assi X e Y semplicemente ruotando la testa. In questo modo, però, si identificano gli assi del mondo come se fossero unicamente paralleli o perpendicolari a quelli del visore. Questo introduce dei problemi perché il dispositivo, dopo la calibrazione SLAM iniziale, non è detto che crei gli assi paralleli/perpendicolari a quelli definiti nello spazio fisico.

Prendendo invece 3 punti va all'utente l'onere di effettuare una registrazione precisa per evitare che gli assi rilevati non combacino con quelli definiti. Demandando la registrazione all'utente vengono inevitabilmente introdotti degli errori e sorgono due problemi principali: definizione di un sistema di riferimento non ortogonale, definizione di un sistema non parallelo con il piano.

Sistema di riferimento non ortogonale I punti ottenuti dall'utente potrebbero generare, nella stragrande maggioranza dei casi, un sistema di riferimento con assi non ortogonali introducendo una distorsione della forma degli oggetti. Per evitare questa situazione si utilizza una procedura, che corregge gli assi per renderli ortogonali, di questo tipo:

1. L'utente, attraverso la procedura di registrazione, definisce origine, asse X e asse Y (non necessariamente perpendicolari tra loro) definendo così il piano XY.

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

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

Sistema di riferimento non parallelo con il piano Supponendo il posizionamento del "marker", su cui effettuare la registrazione, su una superficie verticale e perpendicolare al pavimento (es. muro) è naturale definire il piano XZ parallelo al pavimento stesso. Un'altra situazione, generalmente non desiderata, è dunque quella della definizione di un sistema di riferimento non parallelo al piano definito dal pavimento in quanto è sintomo di errori introdotti nella registrazione da parte dell'utente.

Ad ogni esecuzione Meta 2 richiede la calibrazione SLAM del dispositivo all'utente e successivamente definisce gli assi di orientamento e l'origine del sistema. In particolare il centro degli occhi dell'utente diventa l'origine, l'asse Y il vettore verticale verso l'alto, l'asse X quello orizzontale verso destra e l'asse Z perpendicolare ad essi ed uscente. Meta 2 riesce sempre a costruire il piano definito da X e Z in modo che sia parallelo al pavimento.

Gli assi rilevati dall'utente vengono quindi ulteriormente corretti in modo che il piano XZ del mondo aumentato risulti parallelo al piano XZ definito da Meta 2. Per effettuare questa operazione si calcola la retta di intersezione tra i due piani XZ di Meta 2 e XZ individuato dall'utente. Questa retta, chiamata linea dei nodi N, viene calcolata semplicemente facendo il prodotto vettoriale tra i due vettori normali ai due piani ossia i due assi Y. Se questa retta ha valore 0 significa che i due piani non si intersecano e sono quindi già paralleli tra loro. Nel caso in cui il valore di questa retta sia diverso da 0 si effettua una rotazione di tutti gli assi (X,Y,Z) di un angolo θ, calcolato come l'angolo tra i due assi Y, attorno alla linea dei nodi N [3.9].

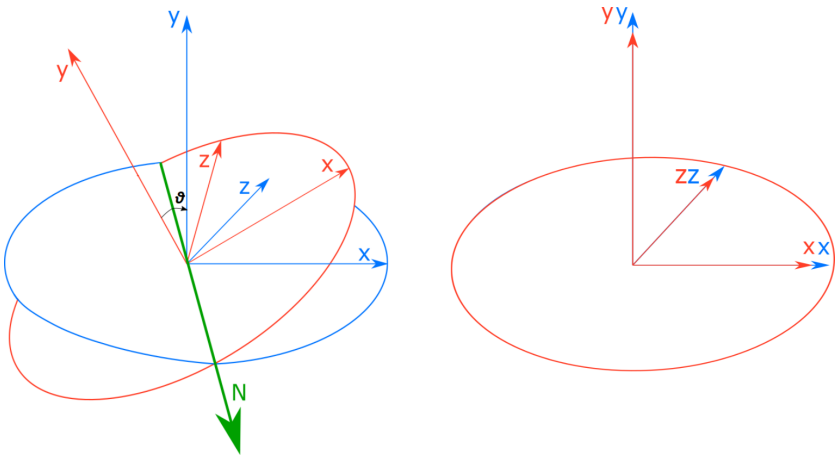

Figura 3.9: A sinistra i due sistemi di riferimento su piani diversi, a destra gli stessi sistemi allineati dalla correzione.

Procedura di registrazione La procedura di registrazione è un insieme ordinato di fasi dopo le quali si ottengono le informazioni necessarie all'identificazione del sistema di riferimento del mondo. Risulta molto semplice definire questo comportamento attraverso una macchina a stati finiti (FSM) [3.10]. In particolare l'implementazione prevedere le seguenti fasi:

- **SettingOrigin**: stato di inizio della procedura; l'utente deve allineare il mirino con il punto di origine del mondo e premere un pulsante per confermare. Per permettere l'allineamento del mirino deve essere attivato necessariamente il depth occlusion. La macchina a stati inizia solamente al termine della procedura SLAM di Meta 2 ricevendo l'evento `onSlamMappingComplete`.
- **SettingX**: l'utente deve allinearsi con il punto relativo all'asse X.
- **SettingY**: analogamente all'asse X l'utente deve allinearsi con il punto relativo all'asse Y.
- **RegistrationEnd**: la registrazione è conclusa, l'utente può decidere, se non soddisfatto, di ricominciare la procedura o confermare i punti scelti. I vettori vengono corretti utilizzando la procedura illustrata precedentemente [3.9].
- **RegistrationFailed**: nel caso in cui gli assi X e Y siano paralleli e non sia possibile ottenere un asse Z e quindi definire un sistema di

riferimento si entra in uno stato di fallimento. Questa situazione può derivare da una cattiva registrazione da parte dell'utente. Richiede la pressione di un pulsante per riniziare la procedura.

- **RegistrationSuccess**: i dati necessari alla registrazione sono stati ottenuti e vengono comunicati al componente che si occupa della costruzione del sistema di riferimento del mondo. É necessario a questo punto ripristinare il depth occlusion alla situazione iniziale.

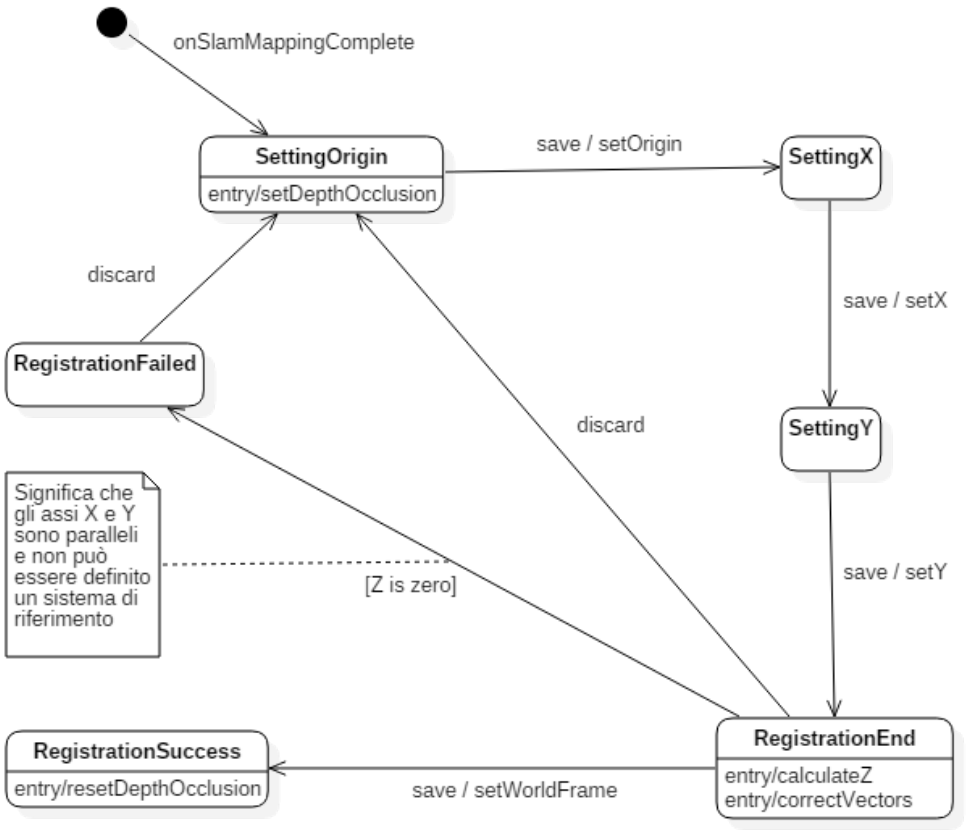

Figura 3.10: Macchina a stati finiti della procedura di registrazione.

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

Terminata la procedura gli assi rilevati vengono comunicati ad un componente specifico che si occupa della costruzione del sistema di riferimento del mondo e della matrice di trasformazione. I vettori ottenuti nel processo

devono essere normalizzati. La procedura di normalizzazione consiste nel correggere la dimensione del vettore in modo che preservi la sua direzione ma che abbia lunghezza (norma euclidea) unitaria.

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

Così facendo un qualsiasi punto che giace nella semiretta identificata dalle coordinate del vettore *(x,y,z)*, rispetto all'origine, 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'* e *v"* (notare l'aggiunta di un quarto elemento a 0):

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

si ottiene lo stesso vettore normalizzato v:

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

Matrice di trasformazione Una volta effettuata la procedura di registrazione viene definito il sistema di riferimento del mondo. Per effettuare la trasformazione di un punto o di un vettore da un sistema di riferimento all'altro è necessario costruire la matrice del cambiamento di frame.

É importante sottolineare che i vettori del sistema di riferimento di Meta 2 su Unity sono i vettori della base canonica con origine in 0. Indichiamo questo sistema di riferimento con la lettera C:

$$C = \{v_1, v_2, v_3, P_0\}$$

$$v_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$$
$$v_2 = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$$
$$v_3 = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$$
$$P_0 = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$$

Indicando con $B = \{u_1, u_2, u_3, Q_0\}$ il sistema di riferimento del mondo possiamo costruire la matrice del cambiamento di frame da questo sistema B al sistema di Meta 2 C semplicemente mettendo in colonna i vettori u_1, u_2, u_3 e l'origine Q_0 rilevati dall'utente in fase di registrazione. Definendo:

$$C = \{u_1, u_2, u_3, Q_0\}$$

$$u_1 = \begin{pmatrix} u_{11} & u_{12} & u_{13} & 0 \end{pmatrix}$$

$$
\begin{aligned}
u_2 &= \begin{pmatrix} u_{21} & u_{22} & u_{23} & 0 \end{pmatrix} \\
u_3 &= \begin{pmatrix} u_{31} & u_{32} & u_{33} & 0 \end{pmatrix} \\
Q_0 &= \begin{pmatrix} Q_{01} & Q_{02} & Q_{03} & 1 \end{pmatrix}
\end{aligned}
$$

La matrice del cambiamento di frame $M_{B,C}$ che trasforma le coordinate espresse dal sistema di riferimento del mondo B a coordinate nel sistema del visore C è:

$$
M_{B,C} = \begin{bmatrix} u_{11} & u_{21} & u_{31} & Q_{01} \\ u_{12} & u_{22} & u_{32} & Q_{02} \\ u_{13} & u_{23} & u_{33} & Q_{03} \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Questo è corretto perché i vettori del sistema di riferimento di Meta 2 sono i vettori della base canonica. La matrice che si occupa della trasformazione contraria $M_{C,B}$ da sistema di riferimento di Meta 2 a quello del mondo è semplicemente l'inversa:

$$
M_{C,B} = {M_{B,C}}^{-1} = \begin{bmatrix} u_{11} & u_{21} & u_{31} & Q_{01} \\ u_{12} & u_{22} & u_{32} & Q_{02} \\ u_{13} & u_{23} & u_{33} & Q_{03} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}
$$

Ad esempio ponendo:

$$
B = \{u_1, u_2, u_3, Q_0\}
$$

$$
\begin{aligned}
u_1 &= \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \\
u_2 &= \begin{pmatrix} 2 & 0 & 0 & 0 \end{pmatrix} \\
u_3 &= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \\
Q_0 &= \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}
\end{aligned}
$$

Si ottengono le seguenti matrici di trasformazione:

$$
M_{B,C} = \begin{bmatrix} 0 & 2 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
M_{C,B} = {M_{B,C}}^{-1} = \begin{bmatrix} 0 & 2 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1/2 & 0 & 0 & -1/2 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

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

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

Nel sistema è implementata una classe, chiamata `FrameTrasformation`, che modella i concetti sopra elencati e che permette in modo semplice di effettuare la trasformazione di un punto tra i due sistemi di riferimento richiamando i metodi `ToLocal(Vector3 v)` e `ToWorld(Vector3 v)`. La classe è di carattere globale e quindi implementata come singleton. La procedura di registrazione si occupa della sua inizializzazione.

Rotazione tra i sistemi di riferimento La matrice di cambio di frame definita precedentemente permette di trasformare tutti i punti da un sistema di riferimento all'altro e viceversa. Applicando la matrice ad ogni punto dell'oggetto vengono effettuate tutte le trasformazioni affini sull'oggetto: traslazione, rotazione e scala.

Unity permette di accedere a tutti i vertici di un oggetto attraverso la sua mesh. L'applicazione della matrice di trasformazione ad ogni punto della mesh va a modificare correttamente la visualizzazione dell'oggetto ma non modifica la sua proprietà `transform`. La transform è il componente di un GameObject di Unity che rappresenta le informazioni sulla geometria dell'oggetto. Nella transform vengono memorizzati posizione e scala sotto forma di vettori in 3 dimensioni e la rotazione dell'oggetto sotto forma di quaternione. I quaternioni forniscono una notazione matematica conveniente per la rappresentazione di orientamenti e rotazioni di oggetti in tre dimensioni in quanto risolvono il problema del *Gimbal Lock* che affligge la rappresentazione con gli angoli di Eulero. Nel progetto viene comunque utilizzata quest'ultima rappresentazione in quanto risulta più semplice.

Per effettuare la rotazione degli oggetti modificando la proprietà rotazione della loro transform è stato necessario progettare un algoritmo apposito.

Algoritmo di rotazione L'algoritmo di rotazione progettato va ad allineare (idealmente), ruotando l'oggetto di interesse, il sistema di riferimento del mondo e quello di Meta 2. Per poter effettuare un allineamento tra sistemi di riferimento è necessario che entrambi siano ortogonali. Questo nel nostro caso particolare è garantito dalla procedura di correzione degli assi durante il processo di registrazione. La procedura di registrazione garantisce inoltre l'allineamento degli assi Y; questo significa che per allineare i sistemi di riferimento è sufficiente allineare uno tra i due assi X e Z [3.11].

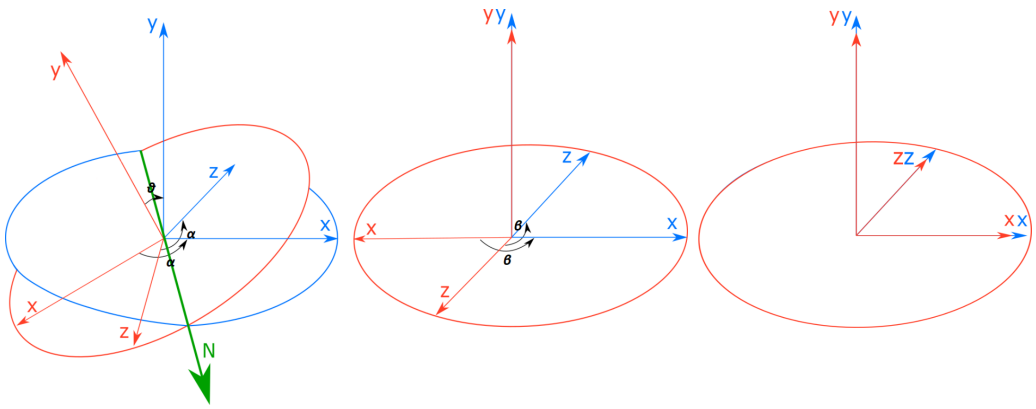

Figura 3.11: A sinistra i due sistemi di riferimento su piani diversi, al centro i due assi Y allineati durante la registrazione, a destra i due sistemi allineati durante la rotazione.

Questa procedura non considera però quella che è la rotazione iniziale dell'utente in fase di registrazione rispetto al sistema di riferimento di Meta 2 definito alla fine della procedura di SLAM. Di conseguenza l'algoritmo di rotazione deve essere applicato considerando il sistema di riferimento di Meta 2 e un sistema di riferimento del mondo ruotato attorno all'asse Y in base alla rotazione iniziale dell'utente.

Per costruire il sistema di riferimento ruotato è necessario salvare l'angolo α di rotazione su Y della camera quando viene fissata l'origine e ruotare gli assi X e Z del sistema di riferimento del mondo di $-\alpha$ intorno all'asse Y [3.12]. Così facendo l'utente può effettuare la registrazione indipendentemente da come è inizialmente ruotato rispetto al "marker" di registrazione.

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

Figura 3.12: Procedura di creazione degli assi ruotati in base alla rotazione iniziale della camera.

Per comprendere meglio il funzionamento dell'algoritmo viene riportata la sua implementazione [3.13]. Inizialmente viene costruito un oggetto fittizio al quale viene assegnata la rotazione ricevuta in input tenendo conto però dell'orientamento iniziale della camera. Per allineare idealmente i due sistemi

di riferimento è necessario conoscere l'angolo β che intercorre tra l'asse del mondo ruotato X e l'asse X di Meta 2. Questo angolo può essere calcolato ruotando intorno a un qualsiasi asse Y in quanto quest'ultimi sono già stati allineati dalla procedura di registrazione. A questo punto si può ottenere la rotazione dell'oggetto ruotandolo dell'angolo β rilevato attorno all'asse Y utilizzando la funzione `Rotate` di Unity.

Viene mostrata la funzione che effettua la trasformazione tra il sistema di riferimento del mondo aumentato e il sistema di riferimento locale di Meta 2; nella trasformazione inversa la rotazione della camera deve essere sottratta e l'angolo β deve essere calcolato dall'asse X di Meta verso quello del mondo ruotato.

```
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 3.13: Algoritmo di trasformazione di rotazione tra sistema di riferimento del mondo e sistema di riferimento di Meta 2.

3.3.3 Integrazione del sistema

Per permettere una integrazione agile con MiRAgE il sistema è stato organizzato in prefab Unity che possono essere direttamente inclusi nella scena. In particolare il prefab che si occupa di tutto il processo di registrazione si chiama `RegistrationManager` [3.14]. Questo componente contiene al suo interno tutti gli elementi necessari per effettuare l'intero processo tra cui:

- **Crosshair**: rappresenta graficamente il mirino che l'utente deve allineare con i punti di registrazione.
- **UIHelpText**: rappresenta l'interfaccia grafica che indica all'utente come comportarsi e lo stato del processo di registrazione.
- **MetaButtons**: prefab che contiene lo script `MetaButtonEventBroadcaster` che permette di registrarsi agli eventi prodotti dai pulsanti fisici presenti nel dispositivo attraverso il pattern observer. Viene utilizzato per permettere all'utente l'uso dei bottoni del visore durante il processo di registrazione.

Per utilizzare Meta 2 è necessario inserire nella scena l'oggetto che rappresenta la camera `MetaCameraRig`.

Figura 3.14: Scena di esempio con RegistrationManager e i suoi componenti.

MiRAgE fornisce delle funzionalità su Unity che possono essere utilizzate per gestire la comunicazione verso il Server Node che gestisce il mondo aumentato. Tra questi componenti viene fornito, in particolare, il prefab chiamato `MiRAgE_hologram_engine` al quale è agganciato lo script `AWEngine` che all'avvio del sistema si occupa dell'apertura del canale TCP verso l'infrastruttura mediante il metodo `StartEngine()`.

I messaggi di creazione e di aggiornamento degli ologrammi ricevuti da questo componente vengono delegati e gestiti da una classe statica specifica `HologramFactory`.

Per integrare le due componenti è necessario apportare delle piccole modifiche in queste classi. In particolare, per prima cosa, è necessario creare il

canale TCP ed effettuare la connessione verso MiRAgE solamente una volta finito il processo di registrazione. Se non è presente l'oggetto RegistrationManager all'interno della scena il collegamento viene creato normalmente all'avvio altrimenti il RegistrationManager si occupa di richiamare il metodo pubblico StartEngine() di AWEngine al termine della registrazione.

La seconda modifica riguarda l'inserimento della trasformazione tra i due sistemi di riferimento nella creazione e aggiornamento di rotazione e posizione degli ologrammi all'interno della classe statica HologramFactory. In particolare posizione e rotazione vengono trasformati attraverso gli appositi metodi del singleton `FrameTransformation` [3.2].

La creazione e la modifica degli ologrammi utilizza sempre le funzioni che trasformano da sistema di riferimento del mondo a sistema di riferimento di Meta 2; le funzioni inverse vengono utilizzate solamente nella comunicazione degli eventi verso l'infrastruttura.

```
1 localPos = FrameTransformation.Instance.
2     ToLocalPosition(position);
3
4 localRot = FrameTransformation.Instance.
    ToLocalRotation(rotation);
```

Listing 3.2: Trasformazione di posizione e rotazione.

Una volta apportate le modifiche specificate in precedenza si può utilizzare il dispositivo Meta 2 con l'infrastruttura MiRAgE utilizzando una scena con al suo interno gli oggetti `RegistrationManager`, `MetaCameraRig` e `MiRAgE_hologram_engine` [3.15].

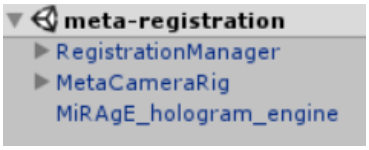

Figura 3.15: Scena Unity base per utilizzare Meta 2 con MiRAgE.

3.4 Considerazioni e risultati ottenuti

L'obiettivo di integrare la tecnologia per realtà aumentata Meta 2 con l'infrastruttura MiRAgE è stato raggiunto con successo. Gli utenti possono registrarsi, tramite la procedura sviluppata, con i punti di allineamento precedentemente collocati nello spazio fisico ed avere una visione consistente del mondo aumentato. Questo risultato è notevole in quanto Meta 2 è una tecnologia emergente e non vi sono, sul web, sistemi che permettono lo sviluppo di applicazioni multi-utente con questo dispositivo.

Il sistema sviluppato è ben funzionante sia per quanto riguarda la posizione degli oggetti sia per la loro rotazione in base alla prospettiva di vista. Quest'ultimo risultato, in particolare, ha richiesto moltissimo tempo e lavoro per essere perfezionato e ha richiesto diverse considerazioni importanti tra cui: allineamento degli assi Y dei due sistemi di riferimento (piano XZ), costruzione del sistema di riferimento ruotato in base alla rotazione iniziale della camera.

Queste considerazioni sono il frutto di un lavoro iterativo e incrementale durante il quale sono state sviluppate diverse versioni e sono stati aggiunti vincoli. Inizialmente, infatti, non era stato previsto l'allineamento degli assi Y del sistema di riferimento; si supponeva che i sistemi potessero essere completamente diversi. Questo aumentava notevolmente la difficoltà e per questo è stato necessario fare la supposizione, sensata, che il "marker" fosse posizionato su una superficie verticale (es. muro) e avesse l'asse Y perpendicolare al suolo. Questa supposizione risulta accettabile per almeno due motivi:

- la procedura di registrazione sviluppata obbliga l'utente ad allineare il mirino virtuale con il punto di interesse muovendo la testa; risulta sicuramente più comodo effettuarla su una superficie verticale.
- non vi è nessun motivo logico per posizionare il marker sul muro con il piano definito da X e Z non parallelo al pavimento.

3.4.1 Pregi e difetti del sistema

É fondamentale analizzare in modo critico pregi e difetti dell'implementazione. Per prima cosa è importante sottolineare che il sistema ha un compito decisamente arduo, che è un prototipo sperimentale e che non vi sono, almeno al momento, sistemi simili su questo dispositivo.

Il sistema sviluppato ha il grande pregio di essere generale e indipendente dallo specifico dispositivo e dal motore grafico in quanto è costruito su una base solida e concettuale che prescinde in linea di principio dalla tecnologia sottostante. Questo è vero in parte perché sono state sfruttate alcune

funzionalità di Meta 2 come ad esempio il depth occlusion e il fatto che gli assi X e Z formano un piano parallelo con il pavimento. In altri dispositivi queste considerazioni non potrebbero essere fatte ma generalmente questi sistemi funzionano in modo simile e forniscono funzionalità simili. L'integrazione con tecnologie di realtà aumentata o engine differenti richiederebbe sicuramente l'apporto di modifiche ma non stravolgerebbe il funzionamento generale.

Un altro vantaggio è la struttura modulare e organizzata del sistema. Meta 2 può essere integrato molto velocemente con MiRAgE inserendo semplicemente, oltre alla camera `MetaCameraRig`, il prefab `RegistrationManager` all'interno della scena. Il RegistrationManager si occupa di tutto il processo di registrazione e della comunicazione con l'infrastruttura in modo trasparente all'utente.

Un altro grosso vantaggio deriva dall'infrastruttura MiRAgE. Fino a questo momento su Meta 2 non sono state sviluppate applicazioni multi-utente e persistenti. L'utilizzo di MiRAgE permette entrambe le cose: più utenti possono agire e percepire lo stesso mondo aumentato e possono collegarsi e scollegarsi liberamente consapevoli del fatto che il mondo aumentato continua ad esistere e ad aggiornarsi. In questo scenario un utente potrebbe ricollegarsi dopo diverso tempo al mondo aumentato e vedere i cambiamenti e le modifiche apportate dagli altri utenti. Anche MiRAgE, d'altro lato, si arricchisce con la prima integrazione di un dispositivo di realtà aumentata dedicato.

Lo svantaggio maggiore risulta essere sicuramente la bassa precisione rispetto all'utilizzo di approcci di visione artificiale. La procedura di registrazione progettata è manuale e demandata all'utente; questo introduce inevitabilmente imperfezioni e inconsistenze. Gli utenti effettuano la registrazione in modo differente ottenendo visualizzazioni scostate e differenti del mondo aumentato. Questi problemi si manifestano quando più utenti partecipano all'interno dello stesso AW e per questo motivo è stato necessario effettuare diversi test sperimentali.

3.4.2 Test di precisione del sistema

I test effettuati sono stati tutti di carattere sperimentale. Non è stato possibile sviluppare test di tipo automatico in quanto i sistemi di riferimento locali sono differenti e posizione/rotazione degli oggetti non possono essere confrontati. Solo nel caso in cui entrambi i dispositivi venissero avviati nello stesso identico punto si potrebbe fare un confronto di questo tipo; realisticamente parlando questa è una situazione irrealizzabile.

Dai numerosi test effettuati è emerso che la differenza media di visualizzazione di un oggetto da parte di due utenti risulta nell'ordine di qualche centimetro.

In particolare per effettuare i test abbiamo utilizzato due dispositivi Meta 2 con due istanze in esecuzione su macchine diverse all'interno della stessa rete locale. Entrambi abbiamo effettuato la registrazione sul medesimo "marker" posizionato al centro della stanza e abbiamo azionato un agente all'interno di MiRAgE con il compito di istanziare una entità aumentata in una specifica posizione del mondo aumentato. Per effettuare la misurazione ci siamo avvicinati entrambi al cubo e posizionando il dito su uno degli spigoli abbiamo ottenuto la differenza di posizione.

Interessante è comprendere meglio da cosa derivano queste differenze e questi errori. Per prima cosa bisogna evidenziare le difficoltà di Meta 2 nel rendering degli oggetti a distanze superiori al metro. Ci siamo resi conto, infatti, che maggiore è la distanza e maggiore è la sensazione di distinzione tra i diversi utenti. Avvicinandosi questa sensazione si riduce notevolmente e l'oggetto discosta di pochi centimetri come detto in precedenza.

Questi errori sono dovuti alla differente registrazione da parte degli utenti e in particolare nei punti di origine e asse X. I punti di origine e asse X possono essere infatti presi a profondità differenti e possono dare origine quindi a sistemi di riferimento leggermente diversi. Questa differenza anche se piccola (es. 1°) genera assi che a grandi lunghezze differiscono notevolmente.

Possiamo quindi evidenziare due errori distinti: uno costante e uno variabile. L'errore costante risulta misurabile sugli oggetti che si trovano all'origine del mondo aumentato ed è probabilmente dovuto alla diversa registrazione del punto di origine e ai dati diversi di posizione rilevati dai sensori di ciascun dispositivo.

L'errore variabile dipende invece dalla distanza dell'oggetto rispetto all'origine. Supponendo che i sistemi di riferimento registrati siano differenti, e che quindi gli assi che li compongono non siano esattamente paralleli, si crea un errore variabile che risulta tanto maggiore quanto più ci si allontana dall'origine.

Nel caso applicativo specifico di scenario indoor in cui le distanze sono comunque limitate e sottolineando che Meta 2 è a livello filosofico una semplice estensione di un ambiente desktop l'errore variabile non risulta comunque proibitivo.

La misurazione dell'errore di rotazione è stata invece effettuata in maniera meno approfondita. In particolare sono stati effettuati test osservando la visualizzazione delle facce di un cubo numerato dai diversi punti di vista. Questi test volevano verificare il corretto funzionamento dell'algoritmo

realizzato senza andare a misurare effettivamente le differenze tra i diversi dispositivi.

Un test interessante non approfondito per mancanza di tempo è quello con altri dispositivi. Effettuando dei test utilizzando due dispositivi Meta 2 l'errore misurato potrebbe essere dovuto alla registrazione di un utente, dell'altro o addirittura di entrambi. Facendo dei test con dispositivi che utilizzano marker e che garantiscono la correttezza della posizione degli oggetti si potrebbe stimare meglio l'errore del singolo visore Meta 2.

A seguire viene riportata una immagine che ritrae le due differenti visualizzazioni dello stesso oggetto utilizzando Meta 2 [3.16]. L'oggetto costruito nello specifico esempio è un semplice cubo con le facce numerate per poter apprezzare la corretta visualizzazione di rotazione dai due punti di vista distinti. Le due riprese sono a distanze diverse come si può notare dalla dimensione del cubo ma rendono comunque l'idea del risultato ottenuto. Nel caso specifico il cubo non è posizionato all'origine ma 0.5 m indietro lungo l'asse Z e 0.5 m in alto lungo Y e la differenza misurata risultava essere 2.5 cm.

Figura 3.16: Visualizzazione contemporanea dello stesso oggetto virtuale attraverso Meta 2 utilizzando MiRAgE.

Capitolo 4

Sistema di gestione degli eventi e degli input

Le applicazioni di realtà aumentata sono un campo di sviluppo innovativo e in continua crescita. In particolare c'è grande interesse nello sviluppo di interazioni con gli oggetti virtuali da parte dell'utente.

La realtà aumentata è una tecnologia che si allontana molto dagli strumenti informatici classici; le interazioni sono quindi profondamente diverse e più simili a quelle tra uomo e mondo reale.

Idealmente un utente vorrebbe poter interagire con gli oggetti virtuali come se fossero oggetti reali così da eliminare totalmente la loro distinzione e risultare, di conseguenza, completamente immerso nel mondo aumentato.

4.1 Gestione dell'input in Meta 2

Per andare a definire gli eventi e gli input che devono essere modellati è necessario andare ad analizzare le principali categorie dettate dallo stato dell'arte. In particolare possiamo identificare tre categorie di input principali [Ped17, p. 370]: riconoscimento vocale, riconoscimento dei gesti e riconoscimento dello sguardo.

Gli eventi da modellare devono essere il più astratti possibile così da poter essere utilizzati in dispositivi diversi. Se la struttura e la gerarchia degli eventi è ben modellata, cambiando dispositivo è necessario implementare solamente la parte di rilevazione di tali eventi.

4.1.1 Dispositivo di interesse: Meta 2

Nel caso specifico, lavorando sul dispositivo Meta 2, è stato necessario effettuare una analisi delle funzionalità offerte dal SDK. In particolare Meta 2 non integra, per il momento, il riconoscimento vocale e non fornisce accesso al microfono del dispositivo. Per effettuare un'integrazione del controllo vocale sarebbe necessario l'utilizzo di un microfono esterno e implementarlo manualmente; per questi motivi il riconoscimento vocale non sarà modellato. Meta 2 fornisce, invece, supporto al riconoscimento delle mani e dello sguardo:

- **Riconoscimento sguardo**: il dispositivo non effettua un vero e proprio riconoscimento dello sguardo in quanto si limita a rappresentarlo come il vettore uscente dalla camera andando così a rilevare l'orientamento del visore e non quello degli occhi. Viene rimossa così la possibilità di guardare un oggetto utilizzando lo sguardo periferico.

- **Riconoscimento delle mani**: Meta 2 è in grado di riconoscere le mani; in particolare è in grado di determinare la posizione del palmo della mano, il tipo di mano (destra o sinistra) e il suo stato. Lo stato in particolare fornisce diverse informazioni utili che possono rappresentare degli eventi di input: se la mano è molto vicina ad un oggetto (hover) oppure se sta tenendo un oggetto (grab).

4.1.2 Eventi di input

É importante sottolineare che devono essere modellati degli eventi di input e non delle interazioni; questo perché i dispositivi, come anche Meta 2, forniscono interazioni diverse sviluppate però sugli stessi eventi basilari. Modellando degli eventi di input basilari si garantisce flessibilità e riusabilità demandando poi la costruzione delle interazioni complesse alla infrastruttura di realtà aumentata.

Gli eventi di input possono essere quindi suddivisi in eventi generati dalle mani o dallo sguardo [4.1]. Essendo eventi di input vi è sempre un oggetto (ologramma) di interesse.

Attraverso lo sguardo possiamo identificare due eventi basilari: lo sguardo (Gaze) generato quando si osserva un oggetto e l'opposto (ReleaseGaze) generato quando si smette di osservare una entità.

Riguardo le mani gli eventi basilari possono essere: tocca e smetti di toccare (Hover, ReleaseHover) e, afferra e lascia (Grab, ReleaseGrab).

Sguardo	Mani
guarda, smetti di guardare	tocca, smetti di toccare, afferra, lascia

Tabella 4.1: Eventi base di input categorizzati per sguardo e mani.

4.2 Integrazione in MiRAgE

4.2.1 Aspetti architetturali

La gestione degli eventi è necessaria per completare l'integrazione di Meta 2 con l'infrastruttura MiRAgE.

A livello architetturale questa componente si trova all'interno del Hologram Engine (HE), che nel caso particolare è Unity3D, nella applicazione utente [4.1].

Gli eventi sono elementi architetturali e come tali vengono comunicati al AW Server Node attraverso il canale TCP principale e non utilizzando l'interfaccia web fornita da WoAT.

L'integrazione con MiRAgE necessita anche la comunicazione delle informazioni relative all'utente, rappresentato dall'avatar all'interno del mondo aumentato. In particolare MiRAgE deve essere informata di ogni cambiamento relativo allo stato dell'utente: posizione, orientamento, direzione dello sguardo e stato delle mani. Queste informazioni possono essere rappresentate anch'esse da eventi che non riguardano ologrammi ma l'avatar dell'utente.

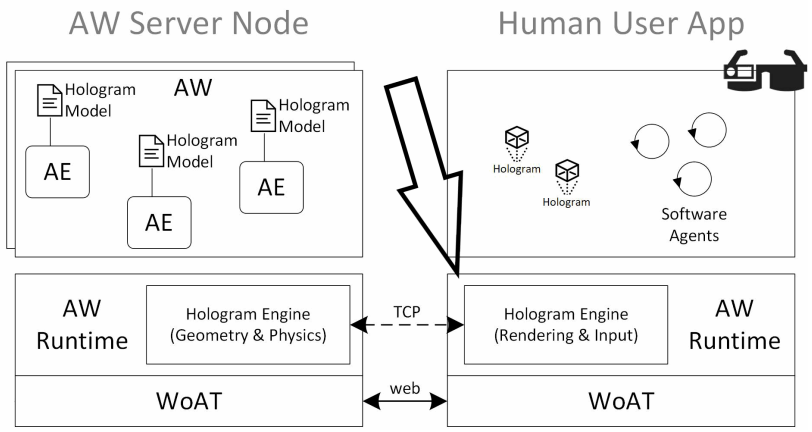

Figura 4.1: La freccia indica dove è situato il componente di gestione degli eventi all'interno di MiRAgE.

4.2.2 Gerarchia degli eventi

Con le informazioni ottenute in fase di analisi i vari tipi di eventi sono stati categorizzati e inseriti all'interno di una gerarchia [4.2].

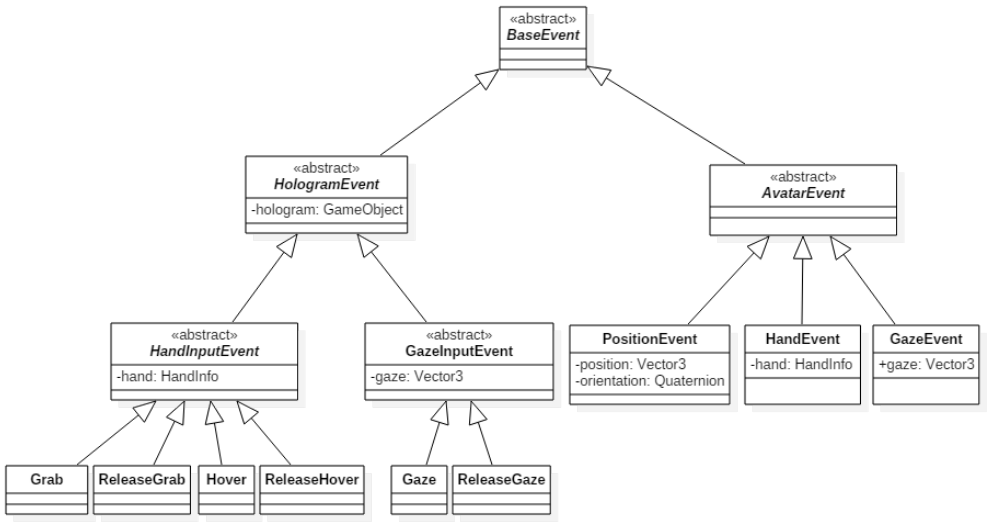

Figura 4.2: Gerarchia completa degli eventi ottenuta al termine dell'analisi.

Alla radice si trova la classe astratta `BaseEvent` che rappresenta in modo generico il concetto di evento. La prima fondamentale distinzione è data dagli eventi che riguardano un ologramma `HologramEvent` e gli eventi che riguardano l'avatar dell'utente `AvatarEvent`. Gli HologramEvent vengono poi ulteriormente suddivisi in due sotto-categorie: `HandInputEvent` e `GazeInputEvent` che rappresentano rispettivamente gli eventi di input generati con le mani e quelli generati dallo sguardo.

Queste due sotto-categorie di eventi relativi agli ologrammi rappresentano tutti gli eventi di input; deve essere aggiunto quindi un ulteriori livello che discrimini il tipo specifico. In particolare figurano gli eventi precedentemente elencati in fase di analisi: `Grab`, `ReleaseGrab`, `Hover` e `ReleaseHover` per quanto riguarda le mani e `Gaze`, `ReleaseGaze` per lo sguardo.

Oltre agli eventi relativi agli ologrammi figurano gli eventi relativi all'avatar, modellati dalla classe astratta AvatarEvent. Questi eventi sono necessari per aggiornare lo stato dell'avatar dell'utente all'interno del mondo aumentato di MiRAgE. In particolare vengono identificate tre tipologie di eventi: `PositionEvent` per notificare il cambiamento di posizione o di orientazione

da parte dell'utente, `HandEvent` per notificare il cambiamento dello stato di una mano (ad esempio la posizione), `GazeEvent` per notificare un cambiamento di direzione dello sguardo.

Nel diagramma figura un oggetto `HandInfo` [4.3] che rappresenta le informazioni relative alla mano tra cui: il tipo (destra o sinistra), la posizione e l'orientamento. É stata implementata una classe apposita, anche se Meta 2 fornisce già una classe `Hand` molto più completa, per rendere gli eventi relativi alla mano indipendenti da Meta e permettere il loro utilizzo anche con altri dispositivi.

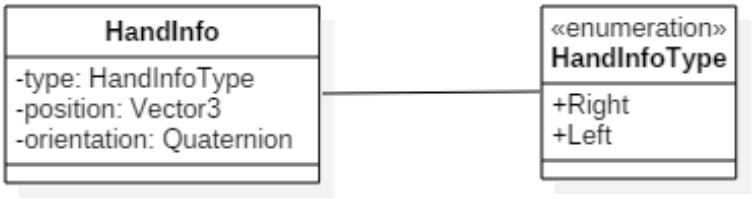

Figura 4.3: Classe che rappresenta le informazioni della mano.

L'ultimo livello di gerarchia degli eventi relativi agli ologrammi non aggiunge informazioni ma definisce solamente il tipo di evento; per questo motivo e per evitare la proliferazione di classi sono state utilizzate due enumerazioni: `HandInputEventType` e `GazeInputEventType` [4.4].

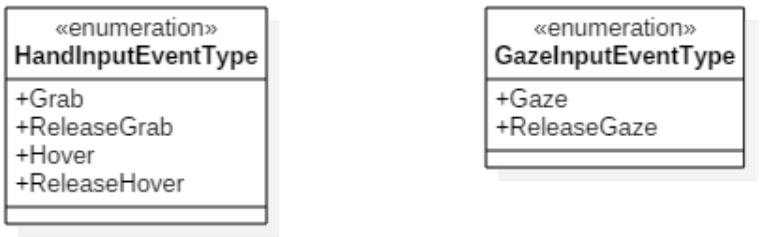

Figura 4.4: A sinistra l'enumerazione che definisce le tipologie di eventi di input con le mani a destra con lo sguardo.

La gerarchia degli eventi implementata risulta così semplificata e meno profonda [4.5]:

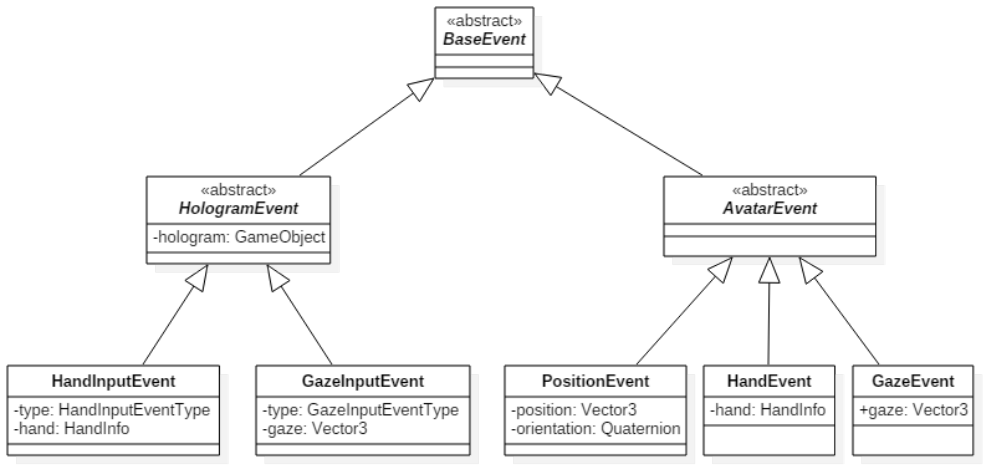

Figura 4.5: Gerarchia degli eventi implementata.

4.2.3 Rilevazione degli eventi

La rilevazione degli eventi non può prescindere dalla tecnologia utilizzata. In base al dispositivo utilizzato e in base alle funzionalità offerte è necessario implementare un componente apposito che si occupi di rilevare gli eventi e notificarli a MiRAgE.

Per nascondere, anche per quando riguarda gli sviluppi futuri, la necessità di inviare i messaggi all'infrastruttura è stata progettata una classe astratta `BaseEventNotifier` che deriva da `MonoBehavior` e implementa l'interfaccia `IEventNotifier` che prevede il metodo `NotifyEvent(BaseEvent ev)` [4.6].

Questa classe si occupa, in modo trasparente, di convertire gli eventi in oggetti JSON e di comunicarli al componente specifico (Heb: Hologram engine bridge) che si occuperà del loro invio verso MiRAgE. In questo modo, senza conoscere i dettagli dell'invio del messaggio verso l'infrastruttura, è possibile implementare il proprio gestore degli eventi modellando una classe che estende da BaseEventNotifier e richiamando il metodo NotifyEvent(BaseEvent ev).

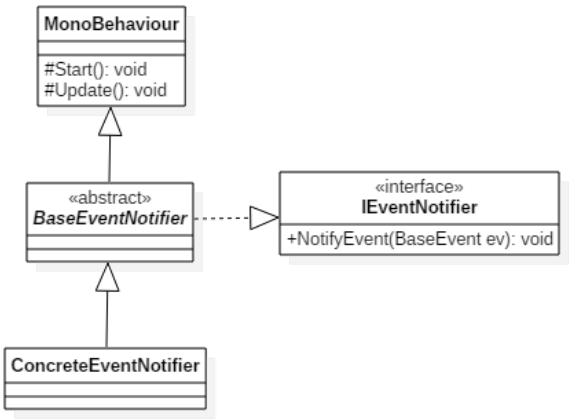

Figura 4.6: Struttura degli EventNotifier e implementazione concreta.

Rilevazione degli eventi su Meta 2

Un parte corposa del progetto riguarda i componenti per la rilevazione degli eventi nel dispositivo Meta 2. La rilevazione dei cambi di stato delle proprietà di interesse può essere effettuata generalmente in due modi:

- **Campionamento periodico (Polling)**: la lettura del valore di interesse viene fatto periodicamente ad una frequenza di campionamento ben definita. Questo è l'approccio generalmente più semplice ma anche il più problematico. La scelta del periodo di campionamento è un aspetto molto importante: il periodo deve essere sufficientemente piccolo da evitare la perdita eventi ma allo stesso tempo sufficientemente grande da non richiedere troppo tempo di calcolo al processore. In particolare per evitare la perdita di eventi è necessario scegliere un periodo di campionamento inferiore al MEST (Minimum event separation time) che è l'intervallo di tempo più piccolo che può intercorrere tra due eventi nella specifica applicazione.

- **Utilizzo degli eventi**: il componente che deve essere osservato si occupa della notifica dei cambiamenti di stato al suo interno agli osservatori interessati. Questo tipo di gestione implementa il pattern observer: la sorgente mette a disposizione un'interfaccia per registrare gli osservatori interessati a ricevere gli eventi, mentre gli osservatori (listeners) di una sorgente mettono a disposizione una interfaccia per la notifica dell'evento. Utilizzando questa tipologia di gestione si risolvono le problematiche relative alla scelta del periodo di campionamento.

Nell'implementazione dei componenti che si occupano della rilevazione degli eventi su Meta 2 è stato necessario utilizzare un approccio misto. Questo perché lo Unity SDK di Meta 2 fornisce un componente (`HandsProvider`) che è in grado di notificare solamente gli eventi `OnHandEnter`, `OnHandExit`, `OnGrab`, `OnRelease` [3.4].

Per rilevare gli altri eventi è stato utilizzato un approccio polling con periodo di campionamento fissato a 50ms per garantire una buona reattività e per evitare la perdita degli eventi.

A livello pratico lo spostamento dell'utente, inteso come cambiamento della sua posizione, avviene continuamente in quanto la posizione è determinata da sensori molto precisi. Anche utilizzando un periodo di campionamento così piccolo questi eventi vengono persi ma a livello applicativo risulta un buon compromesso tra reattività ed efficienza.

Dal punto di vista implementativo per effettuare un campionamento periodico è stato utilizzato il metodo `InvokeRepeating(string methodName, float time, float repeatRate)` della classe `MonoBehaviour` in quanto il metodo `Update()` viene eseguito ad ogni frame e quindi non periodicamente.

Per rendere il codice più strutturato sono stati sviluppati due componenti separati che gestiscono gli eventi: `AvatarHandler` che si occupa degli eventi relativi all'avatar e `InputHandler` che si occupa della rilevazione degli eventi di input. Questi due componenti non appena rilevato un evento lo comunicano attraverso il metodo della classe base `NotifyEvent(BaseEvent ev)` a MiRAgE.

Meta 2 presenta però, soprattutto per quanto riguarda la rilevazione delle mani, diverse imperfezioni; spesso le mani vengono rilevate quando non dovrebbero.

Una migliore implementazione dei due componenti potrebbe effettuare dunque degli aggiustamenti (input conditioning) per cercare di rilevare solamente i segnali significativi.

Una prima tecnica di filtraggio potrebbe consistere nella riduzione del tempo di campionamento. Nel nostro caso applicativo questa soluzione non è ottimale in quanto è fondamentale la reattività del sistema.

Una soluzione più generica e appropriata consiste nel richiedere che l'evento rilevato abbia una certa durata minima; è probabile che la ricezione di più eventi molto ravvicinati temporalmente sia un errore di rilevazione.

Per integrare gli input con Meta 2 e MiRAgE è sufficiente inserire un'istanza del prefab `MetaHands` e un'istanza del prefab sviluppato `InputHandler` che manda in esecuzione i due rilevatori di eventi descritti in precedenza. L'oggetto con cui si vuole interagire deve avere agganciato lo script `MetaInteraction`.

4.2.4 Comunicazione degli eventi

Una volta rilevati gli eventi devono essere trasmessi all'infrastruttura MiRAgE. Come detto in precedenza la comunicazione di questi eventi al AW Server Node non avviene attraverso l'interfaccia WoAT ma attraverso il canale TCP principale.

MiRAgE fornisce un insieme di funzionalità lato Hologram Engine (Unity) che permettono la comunicazione verso l'infrastruttura. In particolare è presente una classe `HebEndPoint` (Heb: Hologram engine bridge) che mantiene il collegamento TCP al server. Questa classe fornisce un metodo asincrono `sendMessage(JSONObject message)` che permette di inviare gli eventi a MiRAgE.

Per lo scambio di dati viene utilizzato il formato JSON (JavaScript Object Notation). JSON è un formato di tipo testuale facile da leggere e scrivere per le persone e facile da analizzare e generare dai calcolatori. É molto utilizzato perché è completamente indipendente dal linguaggio di programmazione e si basa su due strutture che sono comuni in tutti i linguaggi: coppie nome/valore e elenco ordinato di valori (vettori). Utilizzando solo due strutture dati così semplici JSON è comunque in grado di definire strutture arbitrariamente complesse annidando più oggetti.

Per poter inviare gli eventi a MiRAgE è necessario che essi siano convertibili in oggetti JSON. Per garantirlo la classe base `BaseEvent` implementa l'interfaccia `IJSONFormattable` [4.7] con il metodo `ToJSON()`. Tutti gli eventi derivati devono implementare questo metodo richiamando quello della classe base e aggiungendo le informazioni di competenza. Il sistema di override permette di costruire quindi il messaggio in modo incrementale partendo dalla radice.

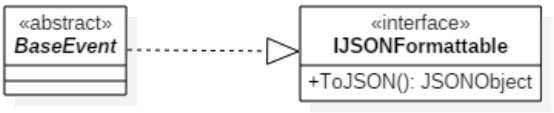

Figura 4.7: Implementazione dell'interfaccia IJSONFormattable da parte degli eventi.

Tra le informazioni dell'ologramma vengono inseriti nel messaggio JSON solamente le informazione principali quali: nome, posizione e orientamento. La classe `HandInfo` implementa anch'essa l'interfaccia `IJSONFormattable` e viene inserita interamente.

A seguire un esempio di messaggio JSON rappresentante un evento relativo al cambiamento di posizione dell'avatar dell'utente [4.1]:

```
{
    "event-category":"avatar event",
    "event-subcategory":"position event",
    "position":{
        "type":"3D/cartesian-location",
        "x":0,
        "y":0,
        "z":0
    },
    "orientation":{
        "type":"3D/angular-orientation",
        "roll":0,
        "pitch":0,
        "yaw":0
    }
}
```

Listing 4.1: Rappresentazione JSON di evento di cambiamento di posizione dell'avatar.

4.3 Considerazioni

É importante ora fare qualche considerazione in merito a quanto sviluppato. Il sistema di gestione degli eventi e degli input costruito fornisce una interfaccia comune e astratta per la comunicazione di eventi e di input verso l'infrastruttura MiRAgE.

Il sistema risulta generale e astratto in quanto la gerarchia degli eventi modellata include al suo interno solamente gli eventi basilari permettendo l'integrazione su un qualsiasi dispositivo.

Ovviamente la rilevazione di tali eventi non può prescindere dalla specifica tecnologia ed è necessario implementare uno specifico componente che effettui tale rilevazione. Viene fornita comunque allo sviluppatore una classe astratta `BaseEventNotifier` dalla quale si può estendere e tramite la quale si può notificare la rilevazione degli eventi in modo trasparente senza conoscere i dettagli implementativi di comunicazione verso MiRAgE.

É fondamentale sottolineare che diversi dispositivi potrebbero presentare eventi basilari leggermente differenti. Ad esempio l'evento di input *Grab*,

che figura in maniera esplicita su Meta 2, potrebbe non essere presente su HoloLens o su un dispositivo per la visualizzazione generica (smartphone). Questo non va a intaccare concettualmente gli eventi modellati in quanto l'evento *Grab* va inteso in modo figurativo e più generale. Ad esempio su uno smartphone il *Grab* potrebbe essere inteso come l'evento generato dalla pressione prolungata su un oggetto virtuale.

Seguendo questa linea di principio si potrebbero accomunare i diversi tipi di input dei vari dispositivi andando a modellare delle interazioni complesse comuni.

Per quanto riguarda la componente che si occupa della rilevazione degli eventi su Meta 2 possono essere apportate diverse modifiche per migliorarne il funzionamento. Dal punto di vista concettuale i due componenti sviluppati rilevano tutti gli eventi modellati ma dal punto di vista funzionale, ogni tanto, rilevano eventi spuri.

In particolare gli eventi relativi alle mani (soprattutto la presenza e posizione) possono risultare poco precisi. Domandando sul forum mi sono state fornite alcune indicazioni che possono migliorare la rilevazione tra cui:

- Rimboccarsi le maniche lunghe (potrebbero interferire con gli infrarossi).
- Effettuare dei gesti di apertura e chiusura della mano ben definiti (per quanto riguarda il *Grab*).
- Guardare in direzione delle mani (non usando la vista periferica) orientando così la camera a infrarossi verso le mani.
- Utilizzare uno spazio libero; un ambiente con molti oggetti potrebbe causare interferenze.
- Rimuovere la gioielleria da mani e polsi (anelli, orologi, braccialetti).

Seguendo alla lettera queste indicazioni migliora notevolmente il risultato ma non viene eliminato in modo definitivo il problema.

Per ottenere questi risultati e osservare questo fenomeno sono stati effettuati dei test sperimentali lato Unity. Non è stato possibile, purtroppo, implementare un agente su MiRAgE che osservasse lo stato, inteso come posizione, orientazione, sguardo e informazioni sulle mani, dell'avatar dell'utente in quanto il supporto agli eventi in questa parte dell'architettura non è ancora completato.

Viene inserito di seguito un esempio di rilevazione [4.8] in cui vengono generati degli eventi di input di *Grab* da entrambe le mani dell'utente. Tali eventi vengono stampati sulla console Unity nel formato JSON con cui vengono inviati all'infrastruttura.

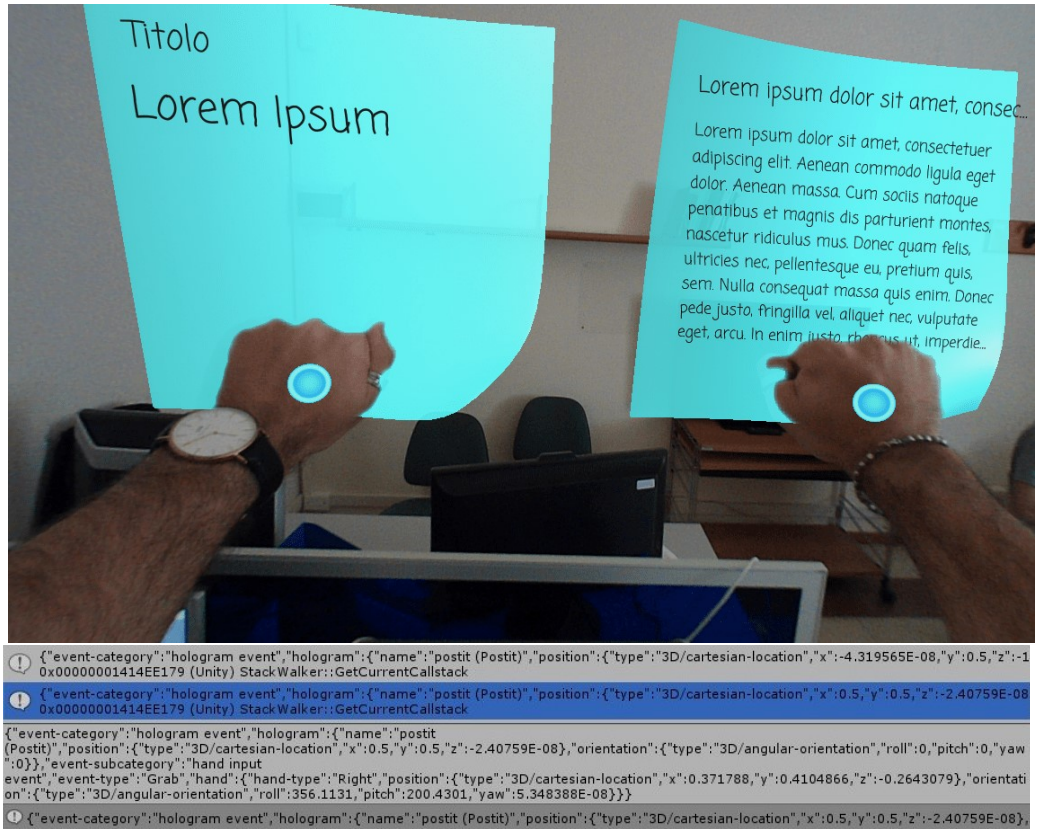

Figura 4.8: Rilevazione di due eventi di Grab su due Postit differenti.

Un aspetto importante non approfondito riguarda la latenza dell'intero sistema. I componenti di rilevazione sviluppati hanno una frequenza di campionamento abbastanza elevata per evitare la perdita di eventi e garantire ottima reattività. Non sono stati effettuati test approfonditi però sulla capacità di gestione di MiRAgE di una quantità elevata di informazioni e sulla latenza di risposta. Nel caso ci fosse una latenza elevata risulterebbe impossibile sviluppare degli agenti che implementano comportamenti complessi utilizzando tali eventi.

Conclusioni e sviluppi futuri

In questa tesi è stato effettuata l'integrazione del dispositivo di realtà aumentata Meta 2 con l'infrastruttura per mondi aumentati MiRAgE sviluppata da Angelo Croatti e Alessandro Ricci.

Il progetto è stato suddiviso in due parti: la prima, che è stata svolta in maniera cooperativa con Daniele Rossi, si poneva l'obiettivo di effettuare l'integrazione prestando particolare attenzione al problema della registrazione con il mondo aumentato mentre la seconda, svolta individualmente, andava a definire un'astrazione per la gestione degli eventi e degli input.

La parte di progetto svolta cooperativamente è stata molto complessa e articolata ed ha richiesto la maggior parte del tempo. É stato necessario approfondire notevolmente le nozioni basilari di computer graphics e le trasformazioni tra matrici e sistemi di riferimento. Inoltre è stato necessario progettare una procedura ad-hoc per effettuare la registrazione con il mondo aumentato così come è stato necessario sviluppare un algoritmo specifico per convertire la rotazione degli oggetti tra i sistemi di riferimento.

Il sistema sviluppato è ben funzionante ed ha il grande merito di essere abbastanza generale e indipendente dal dispositivo di realtà aumentata e dallo specifico engine; il progetto è stato sviluppato con Meta 2 su Unity 3D ma dal punto di vista concettuale può essere implementato, apportando modifiche limitate, su tecnologie differenti.

Tra gli svantaggi maggiori vi è sicuramente la bassa precisione; questo è dovuto alla procedura di registrazione implementata che è demandata all'utente ed è quindi soggetta ad imperfezioni e inconsistenze. In base a come viene effettuata la registrazione da più utenti essi avranno una visione del mondo aumentato leggermente diversa; questo si traduce a livello sperimentale con differenze di posizione degli oggetti nell'ordine di qualche centimetro. Il risultato è comunque apprezzabile e soddisfacente ma tra gli sviluppi futuri l'implementazione di una procedura di registrazione basata su marker è sicuramente la priorità.

É necessario menzionare inoltre che il processo di registrazione sviluppato viene effettuato solamente una volta all'avvio della applicazione. In scenari

realistici questo processo dovrebbe essere fatto continuamente in quella che viene definita registrazione dinamica. Anche questa problematica verrebbe risolta in maniera agile con l'utilizzo di tecniche basate su riconoscimento di marker.

Il progetto individuale ha richiesto uno studio approfondito delle interazioni di input e soprattutto delle funzionalità offerte da Meta 2. La parte principale è stata, a livello concettuale, l'identificazione e la categorizzazione all'interno di una gerarchia delle varie tipologie di eventi. Gli eventi identificati sono basilari e di carattere generale e forniscono quindi una astrazione comune e utilizzabile da dispositivi diversi.

La rilevazione degli eventi, tuttavia, non può prescindere dal particolare dispositivo ed è quindi stato implementato un componente in grado di effettuare tale rilevazione su Meta 2. Lo sviluppo di questo componente ha evidenziato i limiti del dispositivo che non è perfetto nella rilevazione delle mani. Un primo miglioramento della soluzione proposta potrebbe implicare l'utilizzo di filtri a livello software per evitare la rilevazione di eventi spuri. Molto interessante risulterebbe inoltre implementare un componente di rilevazione degli eventi su un dispositivo differente da Meta 2 (es. HoloLens) che usi tuttavia lo stesso insieme di eventi modellato.

Vorrei concludere affermando che sono piuttosto soddisfatto del lavoro svolto e dell'intero percorso universitario.

Ringraziamenti

Vorrei ringraziare in primo luogo i miei genitori, Eleonora ed Eraldo, per il continuo sostegno e amore che mai mi hanno fatto mancare. Dedico a voi questa tesi e vi ringrazio per tutto quello che mi avete insegnato, quello che sono e quello che potrò diventare lo devo solamente a voi.

Ringrazio i miei nonni per avermi sempre spinto ad andare avanti e perché, soprattutto nei momenti più difficili, riescono sempre a farmi capire quali sono le vere preoccupazioni e le cose importanti nella vita.

Ringrazio i miei amici per i bellissimi momenti passati insieme e in particolare i miei compagni di percorso Lorenzo e Edoardo con i quali ho condiviso tanti momenti di studio, lavoro e divertimento negli ultimi 8 anni di scuola e Daniele che ho conosciuto proprio per la realizzazione di questa tesi e con il quale ho trascorso intere giornate negli ultimi indimenticabili mesi. Senza di voi non sarebbe stata la stessa cosa.

Un ringraziamento particolare va poi al Prof. Alessandro Ricci e al Dott. Ing. Angelo Croatti per avermi offerto questa grande possibilità e per avermi aiutato e guidato, con pazienza e disponibilità, nella realizzazione di questa tesi.

Bibliografia

[AC] 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.

[AR15] Luca Tummolini Cristiano Castelfranchi Alessandro Ricci, Michele Piunti. In *The Mirror World: Preparing for Mixed-Reality Living*, 2015.

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

[Ped17] Jon Peddie. In *Augmented Reality: Where We Will All Live*, 2017.

[PM94] F. Kishino P. Milgram. A taxonomy of mixed reality visual displays. In *IEICE Trans. Information Systems, vol. E77-D*, ATR Communication Systems Research Laboratories, Japan, 1994.

[RA01] Reinhold Behringer Steven Feiner Simon Julier Blair MacIntyre Ronald Azuma, Yohan Baillot. In *Recent Advances in Augmented Reality*, 2001.

[Yan15] Yan Yan. Registration issues in augmented reality. University of Birmingham, 2014-2015.