ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

STUDIO DEL MODELLO AUGMENTED WORLDS PER L'INGEGNERIA DI SISTEMI SOFTWARE BASATI SU REALTÀ AUMENTATA: ESPLORAZIONI BASATE SU UNITY E ANDROID

Elaborato in
PROGRAMMAZIONE DI SISTEMI EMBEDDED

Relatore
Prof. ALESSANDRO RICCI

Co-relatore
Ing. ANGELO CROATTI

Presentata da
EDOARDO ANTONINI

Prima Sessione di Laurea
Anno Accademico 2015 – 2016

PAROLE CHIAVE

Augmented Reality

Augmented Worlds

Unity

Vuforia

Android

Dedicato a mia zia Paola e a mia nonna Adriana

Indice

Introduzione

Negli ultimi anni il crescere della capacità di calcolo dei dispositivi e il diminuire delle loro dimensioni ha permesso di far nascere idee innovative e di esplorare più in dettaglio alcuni settori. Uno di questi è sicuramente quello della realtà aumentata (Augmented reality), infatti, la discussione su questo argomento nasce già negli anni 40 del novecento, ma, per mancanza di mezzi tecnologici adeguati, solo ora si iniziano a realizzare le prime applicazioni che si basano su questa idea e il grande pubblico inizia ad interessarsi all'argomento.

Il concetto della realtà aumentata, come mezzo che permette di aggiungere informazioni e funzionalità al mondo reale, ha infinite possibilità di applicazione e potrebbe rivoluzionare in modo massiccio molti settori lavorativi. Le grandi potenzialità di questo strumento hanno stimolato l'immaginazione di molte persone ed ora che iniziano a comparire strumenti, come gli smart-glasses, che permettono di implementare questa idea utilizzando dispositivi wearable, molti sviluppatori si stanno dando da fare per rendere l'AR un mezzo fruibile nella quotidianità.

La costruzione di applicazioni di realtà aumentata, al momento, è basata sull'utilizzo di alcuni framework che mettono a disposizione dello sviluppatore alcune funzioni molto comuni in questi software, come il tracking di marker e l'utilizzo di bottoni virtuali. Questi strumenti, seppur comodi, non garantiscono sempre la buona progettazione dell'applicazione e tendono a unire insieme parti di logica applicativa e di grafica. Per questo motivo, anche nella ricerca, si stanno cercando di studiare dei metodi in grado di permettere una divisione ottimale dei compiti in modo da ottenere un software riusabile e facilmente mantenibile, ma che permetta anche di sfruttare appieno le potenzialità dell'AR attraverso, per esempio, sistemi distribuiti. Un framework concettuale che rientra in questa categoria è sicuramente quello degli Augmented Worlds, mondi virtuali collegati a quello fisico che ne incrementano le caratteristiche e le possibilità tramite la presenza di entità aumentate.

La mia tesi, quindi, si propone di sviluppare un prototipo di un framework con le caratteristiche sopra citate di estendibilità, utilizzando le piattaforme in questo momento a disposizione e ispirandosi alla visione degli Augmented Worlds.

Il documento sarà composto di quattro capitoli. Nel capitolo uno si definiranno alcuni concetti chiave dell'argomento, tra cui, per esempio, Augmented Reality, Virtual Reality e Mixed Reality, mostrando anche alcuni esempi di applicazioni e dispositivi per il deploy, come, per esempio, gli Head Mounted Displays (HMDs). Nel secondo capitolo, invece, si entrerà più nel dettaglio dei componenti necessari per sviluppare un'applicazione AR e delle possibili architetture utilizzabili, accennando anche ad alcuni framework ora disponibili. Nella terza parte si darà un introduzione al concetto di Augmented World per la costruzione di AR-application, discutendo anche dell'idea dei Mirror Worlds come possibile implementazione Agent-Oriented dello stesso. Nel capitolo quattro si discuterà della parte di progetto realizzata, esaminando in dettaglio le scelte architetturali riguardanti la struttura ed entrando nel merito dello sviluppo di un'applicazione utilizzando il framework creato.

Capitolo 1

Augmented reality

In questa prima sezione cercheremo di definire in modo preciso, ma conciso, il significato di "Augmented reality" e di alcuni termini ad essa legati.

1.1 Concetti introduttivi

Negli ultimi anni il dibattito sulla realtà aumentata in generale è stato molto acceso, anche grazie ai primi tentativi di commercializzazione di alcuni apparecchi (come i Google Glass) che hanno fomentato l'interesse del grande pubblico, questo però ha anche portato confusione su alcune delle definizioni cardine dell'argomento.

1.1.1 Mixed reality

Solitamente viene definita "Mixed reality" una qualsiasi situazione in cui siano presenti sia componenti del mondo reale che elementi del mondo virtuale. Questa definizione copre un numero molto vasto di applicazioni, per esempio possiamo marcare anche un semplice navigatore GPS come un' applicazione di mixed reality, come anche un' app che permette di vedere la rappresentazione virtuale di un divano in casa per decidere se acquistarlo o no.

Una definizione cosi vasta va necessariamente specializzata, infatti, si considerano come sottoparti della mixed reality sia l'augmented virtuality che l'augmented reality come possiamo vedere dalla figura 1.1. Solitamente, si considera come augmented virtuality il caso in cui il mondo virtuale è aumentato da entità del mondo reale, mentre, nel caso opposto, si può parlare di augmented reality.

Figura 1.1: Distribuzione delle tecnologie nel range da totalmente fisico a totalmente virtuale, da [2]

1.1.2 Augmented reality

L' augmented reality, o realtà aumentata, è quindi una sezione della mixed reality con caratteristiche particolari.

Definiremo meglio la realtà aumentata partendo dalla definizione tratta dal libro di A.B. Craig [2]:

" *La realtà aumentata è un mezzo di comunicazione dove l'informazione digitale è sovrapposta al mondo fisico e collegata sia in modo spaziale che temporale, permettendo un interazione in real-time* "

Questa frase riassume vari aspetti importanti della realtà aumentata, in primis quello di essere un mezzo di comunicazione, allo stesso livello di TV o libri per esempio, ma con la possibilità aggiuntiva di poter favorire interazioni tra computer e uomini oltre che tra uomini ed altri uomini.

Inoltre si specifica come il mondo virtuale debba essere strettamente collegato a quello fisico in modo sia **spaziale** che **temporale**. La correlazione spaziale implica che un oggetto virtuale sarà relativo ad una specifica posizione sia che qualcuno lo stia guardando sia in caso contrario, e che l'oggetto rimarrà in quella posizione fino a che un' azione (fisica o virtuale) non ne provocherà lo spostamento.

Con l'allineamento temporale, invece, si cerca di far "scorrere" nello stesso modo il tempo nella realtà fisica e in quella virtuale. L'allineamento temporale è scarso quando, per esempio, un' azione nel mondo reale (la spinta verso l'ologramma di un vaso) genera un effetto nel mondo virtuale (la caduta e rottura del vaso) che però viene visualizzato con un ritardo eccessivo dall'azione iniziale.

La capacità di ottenere applicazioni di realtà aumentata con alte precisioni relative all'allineamento spaziale e temporale, dipende molto dall'hardware utilizzato. Al giorno d'oggi non è sempre possibile fornire rappresentazioni di alta qualità per la poca memoria o velocità di elaborazione dei componenti, tuttavia, la rapida crescita delle tecnologie permetterà in poco tempo di sopperire a queste mancanze.

Un altro punto cardine della definizione è l'interazione, cioè la possibilità di comunicare, modificare o in generale avere un contatto con il mondo virtuale.

Questa definizione copre comunque molte possibili applicazioni, ma ne scarta molte altre, come per esempio quelle che semplicemente sovrappongono del testo virtuale al mondo reale.

1.1.3 Virtual reality

La virtual reality, o realtà virtuale, può essere considerata un caso speciale di realtà aumentata dove l'utente non vede il mondo reale ma ne vede uno sintetico, infatti, in genere, un'applicazione di realtà virtuale si svolge in un ambiente completamente fittizio dove gli stimoli visivi, sonori e, in casi più rari, relativi agli altri sensi, sono virtuali.

Le applicazioni di realtà virtuale cercano spesso di "ingannare" l'utente cercando di farlo sentire "immerso e presente" nel mondo virtuale, nella realtà aumentata, invece, il senso di presenza è garantito dal fatto di vedere il mondo reale. Un'altra differenza tra questi due tipi di sistemi è che nella realtà virtuale non c'è bisogno di tracciare la posizione dell'utente nel mondo, perché un'esperienza del genere può, teoricamente, aver luogo ovunque, non necessitando di riferimenti spaziali.

Figura 1.2: Oculus Rift - il più famoso dispositivo per la realtà virtuale, da www.forbes.com

1.1.4 Telepresence

La telepresence, anche detta telepresenza, è solitamente definita come la tecnologia che permette di agire e ricevere stimoli da un ambiente diverso da quello in cui ci si trova fisicamente. Per esempio, un simulatore di volo che permette di guidare sopra un certo luogo e vedere dal finestrino quel luogo può essere definito come un'applicazione di telepresenza, come anche una teleconferenza dove i partecipanti si vedono dentro una stanza insieme agli altri,

pur essendo distanti migliaia di chilometri. Solitamente in base allo schema in figura 1.1, la telepresenza è definita augmented virtuality.

1.1.5 Cyberspace

Il termine Cyberspace è stato coniato nel libro Neuromancer di William Gibson, ed è definito come *"un'allucinazione consensuale di dati e interazioni in uno spazio tecnologico"*, dove l'attività più frequente è la comunicazione tra persone e tra dati. In base a questo concetto molte tecnologie potrebbero essere definite come cyberspace, dalla semplice comunicazione telefonica alla rete internet in generale. Da questo punto di vista anche la realtà aumentata può essere considerato un concetto relativo al cyberspace, pur non essendo in effetti la stessa cosa.

1.1.6 Metaverse

Il termine Metaverse, invece, si riferisce ad un cyberspace dove tutti sono interconnessi, come, per esempio, il nostro Internet acceduto tramite realtà virtuale (figura 1.3). La parola, tuttavia, ha una connotazione generale e comprende tutta quella serie di ambienti e mondi virtuali a cui si pensa riferendosi al futuro.

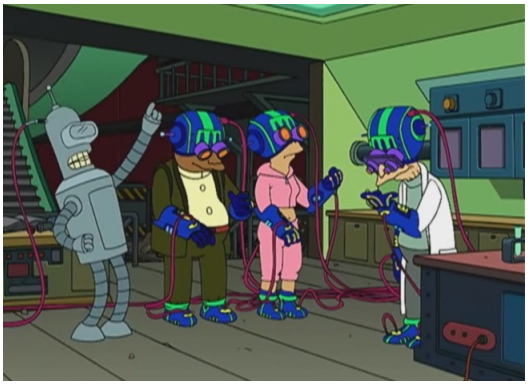


Figura 1.3: Episodio di Futurama in cui i protagonisti accedono ad Internet tramite realtà virtuale, da hackertrips.wordpress.com

1.2 Esempi di AR-application

Uno dei modi migliori per capire cos'è un'applicazione di realtà aumentata è studiare degli esempi, in seguito ne mostreremo alcuni particolarmente rilevanti che identificano le classi di applicazioni più frequenti in questo campo.

1.2.1 The Ethnobotany Workbook

"The Ethnobotany Workbook" è un libro scritto dallo staff del centro medico SEPA dell'Università del Nebraska per diffondere le informazioni sulle piante native della zona, esso è nato come un normalissimo scritto, ma poi si è deciso di trasformarlo in un "magic book", cioè un libro che potesse mostrare modelli virtuali della flora della zona. Per fare ciò, sono stati creati da un'artista i modelli 3D delle piante, che sono stati poi associati a dei semplici marker in bianco e nero. Originariamente il software fu sviluppato su dei laptop posizionati su dei chioschi che, avendo abbastanza spazio per memorizzare tutti i modelli utili, non richiedevano alcuna connessione alla rete. Successivamente, però, si è deciso di realizzare una versione totalmente mobile dell'applicazione attraverso l'utilizzo di smart tablets (come gli iPad) connessi ad Internet. Gli studenti e gli utilizzatori del libro, a questo punto, per fruire dell'applicazione, dovevano solamente attaccare i marker sulle pagine dello stesso e inquadrarle mediante un dispositivo per poter vedere gli ologrammi.

Figura 1.4: Visualizzazione del modello di una pianta tramite il marker posizionato sul libro, da [2]

Bisogna notare che questa applicazione può essere utilizzata per molti altri scopi semplicemente cambiando l'associazione tra i marker e il contenuto, per esempio si potrebbero mostrare modelli 3D di molecole in un libro di chimica utilizzando la stessa architettura.

1.2.2 Daqri anatomy application

Un altro esempio che mostra bene l'utilizzo di applicazioni di realtà aumentata per l'istruzione è quello riguardante l'app di anatomia sviluppata da Daqri. Tramite questo software l'utente può vedere un corpo umano virtuale in scala con il marker, potendo scegliere alcuni parametri tramite dei comandi presenti sul device. Dei virtual buttons (bottoni virtuali) permettono di scegliere che apparati e sistemi mostrare (muscolare, scheletrico, cardio-circolatorio), mentre uno slider permette di scegliere l'opacità della pelle dell'individuo mostrato, per decidere se visualizzarne le parti interne.

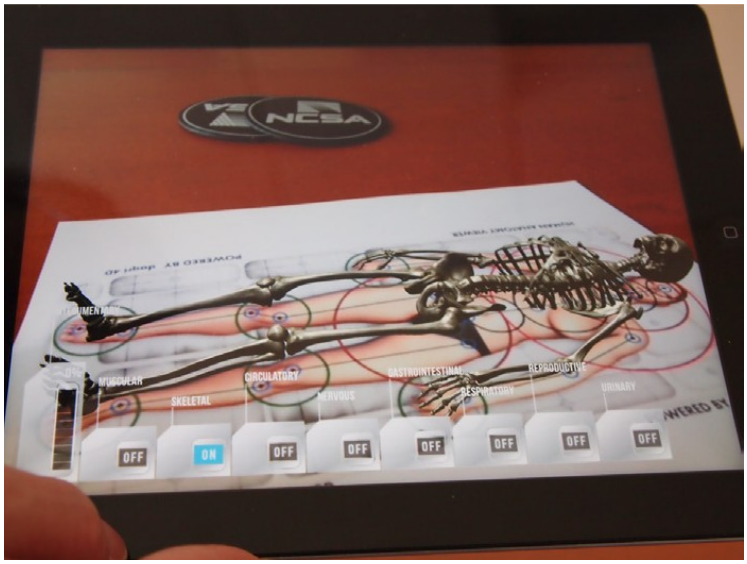


Figura 1.5: Applicazione di anatomia Daqri, in questa foto è visualizzato solo lo scheletro perché solo quel button è stato selezionato, da [2]

Questo genere di applicazioni potrebbero avere una grande crescita nei prossimi anni, tanto da poter permettere lo svolgimento di vere e proprie operazioni virtuali e sostituire, ove possibile, l'utilizzo di cadaveri reali, umani o animali.

1.3 Sistemi hands free per AR-application

Negli ultimi tempi tutti gli wearable-computer hanno cercato di rendersi "hands-free", questa parola viene spesso confusa con "non ingombrante". Un sistema che viene controllato tramite gesture, pur non essendo ingombrante, non può venir definito hands-free perchè usa effettivamente le mani dell'uten-

te, mentre un HMD (Head-Mounted-Display) che utilizza comandi vocali è effettivamente hands-free.

Il maggior beneficio di questi sistemi sta nel non dover manipolare uno smartphone o una tastiera per poter comunicare con il computer, tuttavia le alternative alle tastiere, come per esempio il riconoscimento vocale, non sono ancora del tutto efficienti e possono essere utilizzate solo in un numero limitato di campi.

Nella prossima sezione analizzeremo alcuni tipi di HMDs utilizzabili per applicazioni di realtà aumentata.

1.3.1 Differenze tra optical-see-through e video-see-through display

Solitamente gli HMDs per la realtà aumentata vengono divisi in *optical-see-through* e *video-see-through*, la differenza sostanziale tra i due è che nel primo l'utente vede il mondo reale direttamente attraverso una lente ottica, mentre nel secondo la visuale dell'utente è bloccata ed esso vede il mondo esteriore tramite la proiezione, in un display interno, della registrazione fatta da una telecamera.

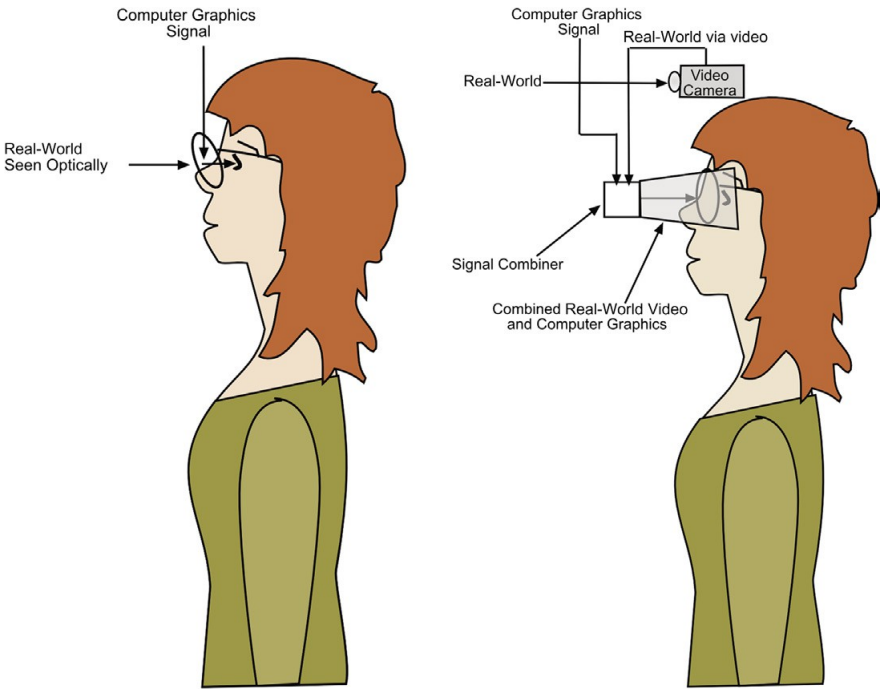


Figura 1.6: Differenze tra optical-see-through, a sinistra, e video see-through, a destra, da [2]

I video-see-through display sono sicuramente più complessi e ingombranti, ma hanno dei vantaggi, infatti, essendo il mondo reale mediato e registrato in un video, esso può essere modificato con tecniche di image processing. Un altro importante vantaggio è il fatto di poter introdurre ritardi di visualizzazione programmati per poter compensare la latenza dell'applicazione, ciò tuttavia genera dei disturbi all'utente. Con l'avanzare della tecnologia e quindi con la diminuzione della latenza stanno prendendo sempre più piede gli optical-see-through display che hanno sicuramente un maggior grado di usabilità e garantiscono una miglior esperienza del partecipante. Al momento sono in commercio molti tipi di HMDs con optical-see-throught, come per esempio gli smart helmets Daqri[1](1.7a) o gli Epson Moverio BT-200[7] (1.7b), quest'ultimi sono degli smart glass utilizzati soprattutto in ambito accademico e sono stati usati per testare l'applicazione del progetto di tesi.

(a) Daqri smart-helmets

(b) Epson Moverio BT-200 smart-glasses

Figura 1.7: Esempi di Head Mounted Displays

[1]daqri.com

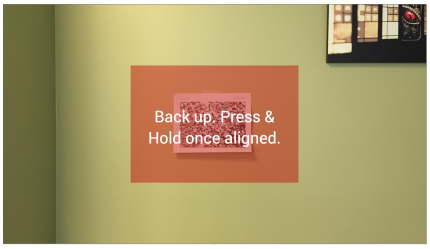
Tecniche di calibrazione HMDs

Solitamente, tutti gli HMDs necessitano di una procedura di calibrazione prima di essere utilizzati, infatti c'è bisogno di un modo per poter allineare geometricamente mondo reale e mondo virtuale, questo processo generalmente permette di ricavare i parametri non intrinsechi della camera (come rotazione e traslazione) da quelli intrinsechi (come lunghezza focale e distorsione) che tuttavia non sono sempre fissi. Il metodo più usato per la calibrazione con parametri intrinsechi variabili è quello marker-based, che viene solitamente diviso in 3 fasi: *marker detection*, *marker identification* e *camera parameter estimation*.

Si individua il marker nel campo d'azione utilizzando tecniche di elaborazione d'immagine come la binarizzazione, lo si confronta con i marker conosciuti e, dopo aver completato questa fase, si associa la posizione 3D del marker nel mondo reale con la posizione 2D dell'immagine catturata, questo permette di stimare i parametri della camera.

Qui di seguito mostriamo un esempio di calibrazione di un generico optical-see-through HMD, da [6]

(a) Allineamento ravvicinato



(b) Allineamento da lontano

Figura 1.8: Calibrazione di un eye-wear utilizzando l'app di Vuforia, il procedimento si svolge in due fasi, un allineamento a distanza ridotta e uno dopo aver fatto qualche passo indietro. L'operazione va effettuata due volte, con l'occhio destro e poi con il sinistro.

Capitolo 2

Progettazione e sviluppo di applicazioni di realtà aumentata

Questo capitolo tratterà delle varie modalità e architetture di sviluppo di applicazioni di realtà aumentata, con un focus sui software utilizzati per la parte sperimentale.

2.1 Software utilizzati

Generalmente per sviluppare un sistema di realtà aumentata servono almeno tre tipi di software:

- Software coinvolti direttamente nell'applicazione,
- Software per creare l'applicazione,
- Software per creare il contenuto 2D-3D dell'applicazione.

2.1.1 Software coinvolti direttamente nell'applicazione

Questi software vengono utilizzati per poter garantire le funzionalità dell'applicazione e sono spesso divisi in:

- Acquisizione dati dai sensori,
- Application engine,
- Software per il rendering.

Come possiamo vedere dalla figura 2.1 i software per l'acquisizione dati, sono coloro che comunicano direttamente con i sensori utilizzati nell'applicazione come per esempio la fotocamera, il GPS, i sensori di temperatura etc. In particolare, in molti sistemi, la fotocamera è fondamentale perché permette di individuare i marker su cui posizionare gli ologrammi. Inoltre questi programmi possono essere utilizzati anche per registrare delle azioni di input dell'utente, come per esempio la pressione di un bottone.

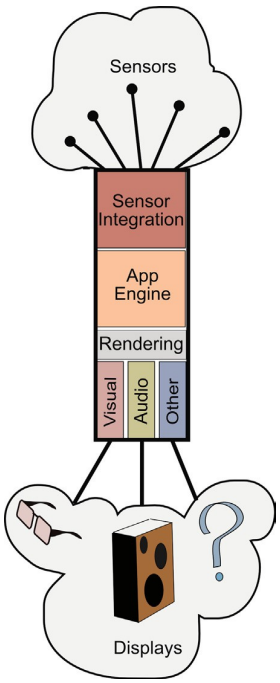


Figura 2.1: Divisione delle parti di un software per AR, da [2]

I dati generati, vengono poi passati all'application engine che rappresenta il vero *core* dell'applicazione, infatti è colui che prende i dati delle interfacce di input, li elabora e li trasmette alle interfacce di output, come per esempio i display. Inoltre questo componente specifica anche tutto ciò che deve accadere nel mondo aumentato e come avverranno le interazioni tra il mondo reale e quello virtuale. Per fare un esempio possiamo dire che, se nel mondo virtuale è presente un'entità, come per esempio un palloncino, l'application engine è colui che controlla la reazione dell'entità al "tocco" dell'utente, che potrebbe provocare un'esplosione oppure un semplice spostamento.

Solitamente l'engine lavora in un ciclo continuo come in figura 2.2 in modo simile a quello che succede nei game engine, la durata di questo loop deve essere tuttavia limitata per garantire la miglior User Experience possibile.

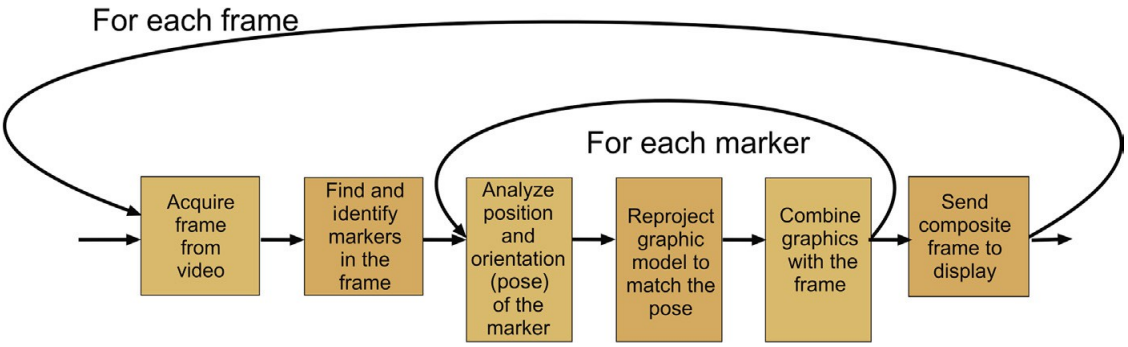


Figura 2.2: Ciclo di lavoro di un'application engine, da [2]

L'engine però non si occupa direttamente della visualizzazione e del rendering degli elementi aumentati, questi infatti sono gestiti da particolari software o driver in comunicazione con le interfacce di output, che possono essere visuali, sonore, tattili e, in alcuni rari casi, anche olfattive o riguardanti il senso del gusto.

Solitamente questi componenti sono distribuiti assieme in una libreria, tuttavia la scelta di quest'ultima può impattare molto sull'applicazione, infatti, ogni libreria ha determinati limitazioni prestazionali e funziona solo con determinate applicazioni e linguaggi.

2.1.2 Software per creare l'applicazione

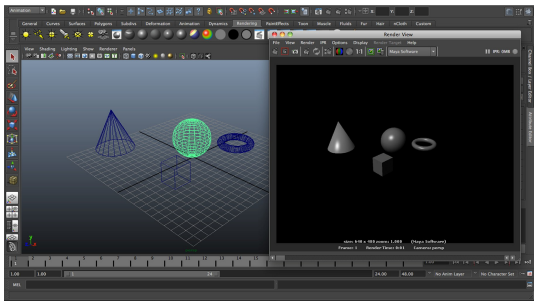
Sviluppare un'applicazione di realtà aumentata non è molto diverso da sviluppare qualsiasi altro software, infatti molto spesso, nei due casi, vengono utilizzati gli stessi strumenti. Ovviamente, come la scelta delle librerie, anche la scelta dell' IDE di sviluppo influenza e limita le possibilità dell'applicazione e quindi deve essere fatta con attenzione.

2.1.3 Software per generare il contenuto

Il contenuto è una delle parti più importanti di una AR-application, infatti arricchisce l'esperienza e contribuisce ad aumentare l'immersione dell'utente.

Ogni applicazione di una certa complessità contiene, grafiche 2D, grafiche 3D e suoni, ognuna di queste parti và creata (o acquistata) e poi integrata nell'applicazione.

Esistono software che possono generare e modificare tutti questi tipi di contenuto, ma solitamente per avere un prodotto di alta qualità si utilizzano software specializzati, come per esempio Autodesk Maya per le grafiche 3D (2.3a) o Crystal softsynth (2.3b) per la musica.



(a) Screenshot da Autodesk Maya

(b) Screenshot da Crystal softsynth

Figura 2.3: Immagini prese da [2]

2.2 Esempi di framework per lo sviluppo di un AR-Application

I framework per sviluppare AR-application stanno aumentando velocemente in complessità offrendo sempre più soluzioni, tra i più utilizzati vale la pena nominare Wikitude, Vuforia e ARToolkit, i primi due necessitano di una licenza pur offrendo una versione di prova con funzioni limitate e/o watermark, mentre il terzo è open-source. Tutti gli strumenti citati, seppur con leggere differenze, forniscono dei meccanismi utili in quasi tutte le applicazioni di realtà aumentata, come il riconoscimento di marker e l'utilizzo di bottoni virtuali, inoltre permettono di sviluppare app su Android e iOS in modo nativo o cross-platform, utilizzando Unity, un noto game-engine. Di seguito daremo un introduzione a Unity[5] e Vuforia[6] in moda da facilitare, successivamente, la comprensione della parte progettuale.

2.2.1 Unity

Unity è un game-engine e un IDE di sviluppo molto utilizzato, che permette di sviluppare applicazioni cross-platform e ha meccanismi di supporto alla distribuzione del gioco in rete. Essendo un programma molto ampio e complesso il suo apprendimento completo può richiedere molto tempo, qui di seguito si spiegheranno alcune caratteristiche base dell'engine per permettere di creare una semplice applicazione di realtà aumentata utilizzando questo strumento.

Per maggiori dettagli si rimanda alla documentazione di Unity.

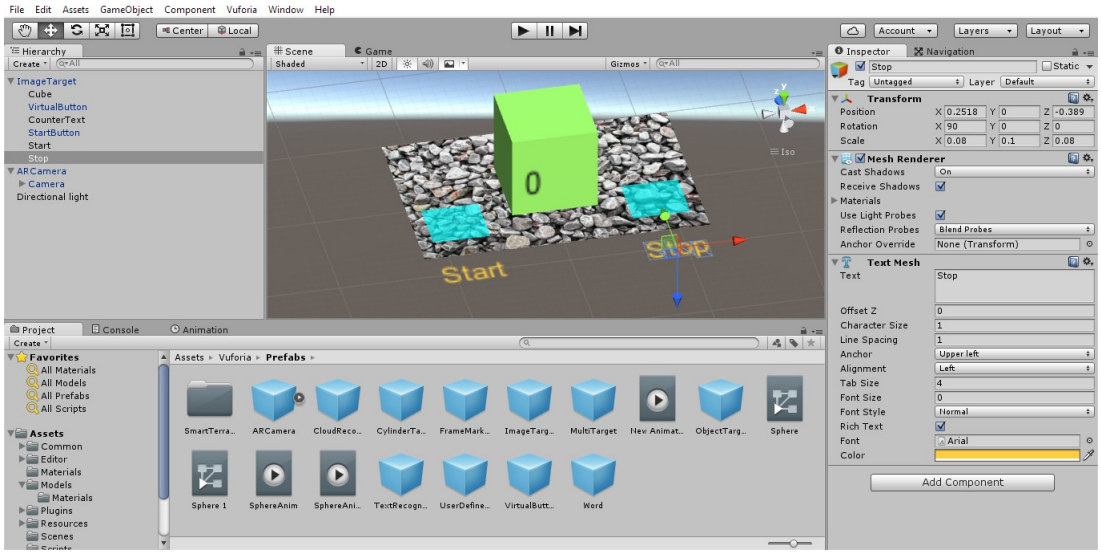


Figura 2.4: Schermata di progettazione Unity

Le viste

La piattaforma di sviluppo si divide in una serie di viste, le principali sono:

- Scene View
- Game View
- Hierarchy View
- Project View
- Inspector View

La Scene View (figura 2.5) è la vista che permette di posizionare gli oggetti all'interno del gioco/applicazione e creare le varie scene, infatti, in questa sezione si possono inserire i componenti 2D o 3D che poi dovranno essere visualizzati. Una volta inseriti, gli oggetti possono essere spostati in base ai 3 assi, ruotati e ridimensionati.

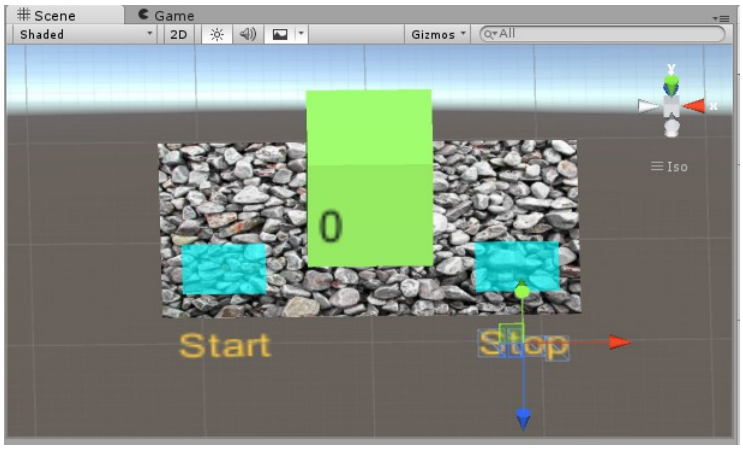


Figura 2.5: Scene View di un'applicazione, il testo selezionato può essere spostato secondo i tre assi visualizzati

La Game View (figura 2.6) è la vista utilizzata durante i test e il debugging del gioco, mostra l'applicazione che viene eseguita, che può essere messa in pausa e riavviata attraverso i tasti nella barra superiore dell'IDE.

Figura 2.6: Dettaglio della Game View, in questo caso, essendo un AR-application, si usa la camera del pc per individuare il marker

La Hierachy View (figura 2.7) è una vista molto importante in quanto definisce tutti gli oggetti che sono presenti all'interno del mondo del gioco. Ogni componente può avere un padre e diversi figli, quindi la gerarchia diventa un albero. Facendo doppio clic su un qualsiasi elemento di questa vista, la Scene View centra la propria visualizzazione rispetto al componente selezionato. Gli elementi possono essere spostati liberamente lungo l'albero.

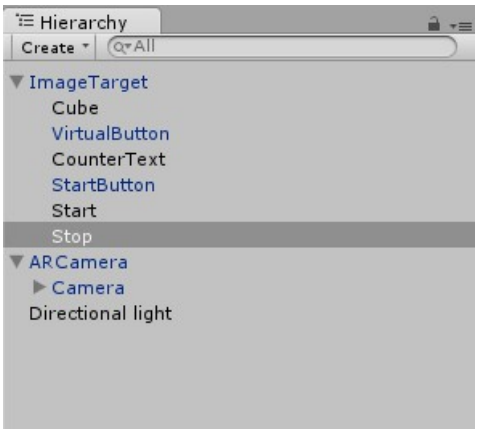


Figura 2.7: Hierarchy view, si notano in particolare alcuni oggetti fondamentali per ogni applicazione, la camera, che rappresenta il punto di vista dell'utente (in questo caso ARCamera) e la *Directional light* cioè l'oggetto che specifica da dove viene la luce all'interno del mondo del gioco

La Project View (figura 2.8) è la vista che permette di visualizzare e organizzare le cartelle del progetto, in particolare contiene tutti gli script, i modelli 3D e le scene utilizzate nell'applicazione

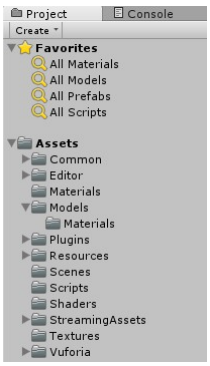


Figura 2.8: Dettaglio della project view

L'Inspector View (figura 2.9) mostra tutte le caratteristiche di un certo elemento selezionato permettendo di modificarle, inoltre dà la possibilità di aggiungere altri *Components* collegati a quell'oggetto come per esempio degli script che ne rappresentino il comportamento.

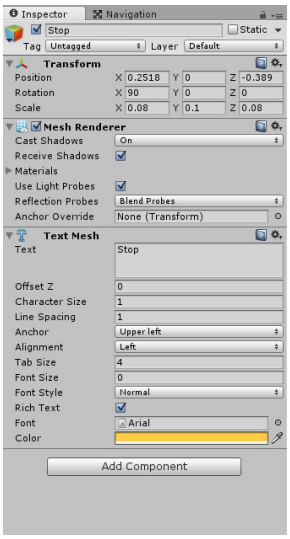


Figura 2.9: In questo particolare della Inspector View, si notano tutte le proprietà di un elemento 3DText selezionato, da notare che i valori nella sezione *Transform* sono visualizzati rispetto alla scala dell'elemento padre nella gerarchia

Il comportamento

In Unity il comportamento degli oggetti viene gestito mediante l'uso di script, che possono essere scritti in Javascript o C# in base alle preferenze dello sviluppatore. Gli script sono composti solitamente almeno da due callback principali, Start e Update. Start è la funzione che viene richiamata all'inizializzazione dell'oggetto, mentre Update viene chiamata per ogni frame del gioco. Da uno script è possibile anche cambiare dinamicamente le proprietà dell'oggetto al quale esso è collegato, inoltre si possono anche ricavare i riferimenti ad altri componenti del mondo e agire su di essi.

```
public class CounterTextController : MonoBehaviour {

    private TextMesh textMesh;
    private int count;

   // Use this for initialization
   void Start ()
    {
       textMesh = GetComponent<TextMesh>();
   }

   // Update is called once per frame
```

```
void Update (){}

 public void Inc()
 {
     this.count++;
     textMesh.text=this.count.ToString();
 }
```

Listato 2.1: Esempio di un semplicissimo script che, tramite la funzione Inc, modifica il testo di una TextMesh (associata ad un 3DText) in base al valore di una variabile

Materiali

Gli oggetti 3D inseriti in Unity possono essere associati a dei materiali (cartella *Materials*) questi ultimi consentono di modificare la grafica dei componenti e il modo in cui essi riflettono la luce e creano ombre. Alcuni framework per realtà aumentata o virtuale mettono a disposizione dei materiali appositi che cambiano colore, per esempio, quando l'oggetto è inquadrato.

2.2.2 Vuforia

Il framework Vuforia mette a disposizione alcuni componenti Unity per lo sviluppo di applicazioni di realtà aumentata, tra i tanti, i più importanti sono gli ImageTarget che rappresentano i marker, l'ARCamera e i VirtualButtons. Una volta importato il package di Vuforia all'interno dell'engine, questi oggetti saranno visibili nella cartella Vuforia/Prefabs

L'ImageTarget è un componente che non ha rappresentazione nell'applicazione, ma è il punto di partenza di essa, infatti, permette l'allineamento tra mondo reale e mondo virtuale facendo da sistema di riferimento. La posizione degli oggetti, quindi, è definita in base ad esso.

L'ARCamera è il componente al quale ci si riferisce per individuare il marker, esso ovviamente dipende da quale dispositivo viene usato, per cui ha tre opzioni:

- None, se il dispositivo non è definito come digital eyewear (es. WebCam del pc o smartphone)
- Optical-see-throught (es. Moverio)
- Video-see-throught

I VirtualButtons, invece, sono dei bottoni virtuali che non sono visibili dall'applicazione, ma vengono attivati quando qualcosa passa al di sopra della zona dove sono stati collocati.

Bisogna specificare, inoltre, che ogni applicazione che utilizza Vuforia deve avere una licenza, quest'ultima può essere gratuita o a pagamento, nel primo caso ci sono limitazioni nell'uso delle funzionalità e la presenza di un watermark. La licenza free, comunque, permette di definire una serie di marker personalizzati salvati nel cloud per creare target custom. Al salvataggio dell'immagine tramite il sito, quest'ultimo indicherà anche la bontà del marker risultante, intesa come facilità di riconoscimento.

Figura 2.10: Dettaglio di un target caricato sul sito di Vuforia con relativo rating

Script relativi a Vuforia

I componenti Vuforia sono collegati automaticamente a degli script necessari per il funzionamento dell'applicazione, che sono contenuti nella cartella Vuforia/Scripts del progetto. Il più importante è sicuramente *DefaultTrackableEventHandler*, che gestisce gli eventi relativi al tracking del marker. Nell'implementazione standard, al riconoscimento del marker si rendono visibili tutti gli elementi figli dello stesso, mentre alla perdita del tracking essi vengo disabilitati. Tuttavia, basandosi sullo stesso template si possono definire handler alternativi, in funzione alle necessità dello sviluppatore.

2.2.3 Esempio di applicazione con Unity e Vuforia

Una semplicissima applicazione possibile con questi due strumenti, potrebbe essere quella che permette la visualizzazione di un contatore posto davanti ad un cubo che viene avviato premendo un bottone virtuale e messo in pausa attraverso un altro, come in figura 2.11.

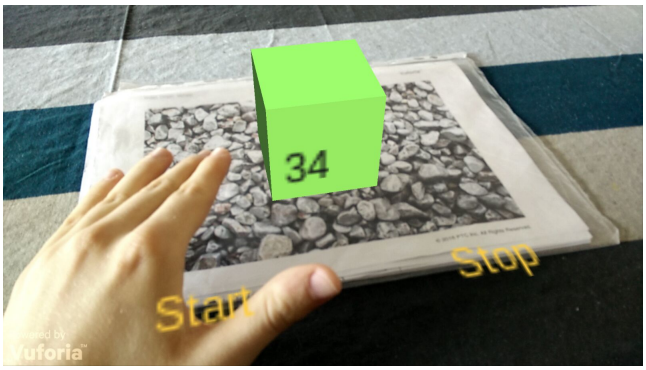


Figura 2.11: Cubo con contatore

Per ottenere questo risultato è stato necessario innanzitutto posizionare nella scena l'ARCamera scegliendo il tipo di dispositivo su cui distribuire l'applicazione. Si è poi piazzato l'ImageTarget scegliendo, dalla sezione *ImageTargetBehaviour* dell' Inspector View, il tipo di marker desiderato per il riconoscimento.

Successivamente, si sono dovuti inserire tutti i componenti da visualizzare come figli del marker (come osservabile nella precedente figura 2.7) insieme ai due Virtual Buttons. Per gestire la logica di pressione dei bottoni si è aggiunto poi uno script all'ImageTarget, riportato di seguito.

```
public class VirtualButtonHandler : MonoBehaviour,
    IVirtualButtonEventHandler
{

    private CounterController counterController;

    void Start()
    {
        this.counterController =
            GetComponentInChildren<CounterController>();
        VirtualButtonBehaviour[] vbs =
            GetComponentsInChildren<VirtualButtonBehaviour>();

        for (int i = 0; i < vbs.Length; ++i)
        {
            vbs[i].RegisterEventHandler(this);
        }
    }

    public void OnButtonPressed(VirtualButtonAbstractBehaviour vb) {
        switch(vb.VirtualButtonName) {

        case "startButton":
                counterController.StartCounting();
                break;
        case "stopButton":
                counterController.StopCounting();
                break;
        default:
            throw new UnityException("Button not supported: " +
                vb.VirtualButtonName);
        }
    }
}
```

Listato 2.2: Handler dei virtual buttons

Da questo codice si può capire come, in base al bottone premuto, si richiami la funzione che avvia o mette in pausa il contatore, da cui poi verrà modificato il testo visualizzato.

2.3 Architetture AR-application

Un'applicazione di realtà aumentata puoi venire eseguita su molti tipi di architetture, tuttavia possiamo dividere queste ultime semplicemente in client-only e client-server. Queste categorie hanno vantaggi e svantaggi che spiegheremo di seguito.

2.3.1 Applicazione client-only

Al giorno d'oggi molti dispositivi hanno già una capacità di calcolo tale da poter eseguire un'applicazione di realtà aumentata di medio livello, questo permette di non aver bisogno di una connessione sempre presente e quindi ne garantisce la completa portabilità. La mancanza di connessione però può diventare un problema con il crescere della complessità dell'applicazione, infatti se sia ha la necessità di svolgere simulazioni complesse o visualizzare una grande quantità di oggetti 3D, la memoria e la cpu di un normale pc o di uno smartphone possono risultare insufficenti o causare dei ritardi non trascurabili. Bisogna specificare, però, che nei prossimi anni l'avanzata tecnologica dei device potrebbe risolvere quasi del tutto quest'ultimo problema.

Una questione più difficile da oltrepassare riguarda la creazione di applicazioni multiutente senza connessione, infatti, se due persone si trovano molto lontane tra loro sarà impossibile farle comunicare senza ritardi pesanti non utilizzando una connessione client-server. Se invece tutti gli utenti sono all'interno di un raggio limitato, si potrebbe pensare di costruire una rete peer-to-peer tra loro per la comunicazione, ciò potrebbe risolvere il problema, ma aumenterebbe di molto il grado di complessità dell'applicazione quindi, solitamente, anche in questi casi si opta per una connessione client-server.

2.3.2 Applicazioni client-server

La crescita tecnologica oltre a incrementare la potenza dei dispositivi sta anche portando la connettività veloce ad essere disponibile sempre in più zone, in molte aree si può dire in buona approssimazione che la connessione è disponibile ovunque, ed è in questi luoghi che può risultare conveniente sviluppare un'applicazione client-server.

Il server in questo caso svolge funzioni quali la gestione dei contenuti, il logging, il riconoscimento dei marker e l'elaborazione degli algoritmi più complessi. A volte sul server, o sul cloud, si specifica l'intera logica dell'applicazione, in questo modo il device client diventa solo un interfaccia utente che riceve dati dalla rete (thin client application).

La connessione risolve molti dei problemi definiti in precedenza, ma non tutti, infatti se gli oggetti da visualizzare sul client sono molto pesanti, la memoria del dispositivo può comunque rappresentare un vincolo e sarà necessario utilizzare apposite strategie di caching e data swapping per limitare il delay.

Come specificato anche in precedenza, le architetture client-server sono le più adatte per sviluppare applicazioni multi-utente con minimo sforzo, come ad esempio videogame online.

Capitolo 3

Il modello Augmented Worlds

Come si è accennato nel capitolo 1 e come si fa notare in [3, 4], il grande sviluppo dell'hardware, e la comparsa degli HMDs sta riducendo il divario tra mondo reale e mondo virtuale. Nel prossimo futuro la progettazione di sistemi software che estendano il mondo fisico con entità computazionali, per esempio visualizzabili tramite AR, potrà diventare un'interessante prospettiva di programmazione.

In questo ambito in [3] si introduce il concetto di *Augmented World*, un software che aumenta le funzionalità del mondo fisico attraverso oggetti *collocati nello spazio* e percepibili dagli utenti. Questi oggetti, o *entità aumentate*, potranno modellare estensioni di oggetti reali, in questo caso, essi saranno soggetti ai cambiamenti di stato della loro controparte fisica, mentre quest'ultima dovrà reagire agli eventi del mondo virtuale.

3.1 Augmented Worlds

La differenza principale tra un Augmented World e una qualsiasi applicazione di realtà aumentata è che, solitamente, gli oggetti contenuti in quest'ultime (come per esempio gli ologrammi) sono utili soprattutto a riferire informazioni all'utente e non hanno senso se non possono essere visualizzati dallo stesso. Un'entità aumentata contenuta in un AW, invece, è un elemento computazionale che ha un comportamento e uno stato particolari che possono essere indipendenti dagli utenti, proprio come avviene per esempio nei personaggi o BOT contenuti negli AR-games.

Nella ricerca si sta cercando di utilizzare quest'ultimo paradigma andando oltre i videogiochi e concentrandosi sulla costruzione di veri e propri framework general-purpose.

3.1.1 Caratteristiche principali negli AW

In questo paragrafo si cercheranno di definire in modo chiaro le caratteristiche di un AW, cercando di astrarre il più possibile dal modello di programmazione che potrebbe essere utilizzato nello sviluppo.

Allineamento spaziale

In un AW le entità aumentate sono sempre localizzate in un qualche punto del mondo fisico (figura 3.1), che può essere specificato direttamente tramite coordinate geografiche o in riferimento ad un particolare oggetto. Il sistema di riferimento per le coordinate può essere assoluto (latitudine e longitudine) o relativo all' AW stesso (e.g. con dei marker). Bisogna notare che un'entità aumentata localizzata in un certo punto dello spazio, non deve essere necessariamente eseguita da un componente hardware che si trova in quel punto, ma può essere gestita da uno o più nodi in rete, per esempio nel cloud.

Per poter essere visualizzata, l'entità, deve avere una propria definizione geometrica che può essere rappresentata attraverso un vettore di coordinate tridimensionali rispetto al sistema di riferimento, come accade in una qualsiasi AR-application. Sia la posizione che la forma geometrica dell'entità possono essere modificate a run-time in base alle interazioni che avvengono nell' AW.

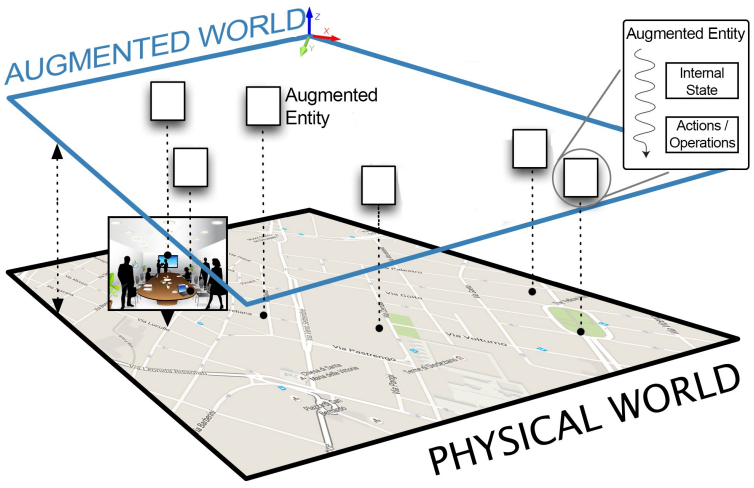


Figura 3.1: Rappresentazione di un'AW con particolare riferimento all' allineamento spaziale, da [3]

Approcci relativi all'interazione con le entità

La spazialità delle entità rende possibili diversi approcci riguardanti l'interazione tra esse e con gli utenti. Per poter interagire con un'entità bisogna

innanzitutto individuarla e percepirla, per fare ciò si possono utilizzare due modalità: pull e push. All'interno di un AW è possibile, infatti, ottenere il riferimento a tutte le entità che si trovano in una determinata regione, questo può essere fatto utilizzando funzioni di *lookup* messe a disposizione dal mondo stesso. L'ottica descritta può essere definita *pull-based discovery*. I metodi push/event-driven possono risultare altrettanto utili poiché non hanno bisogno di effettuare richieste, infatti, definita un entità come osservabile, tutti gli oggetti osservatori potranno, in modo asincrono, percepire i suoi cambiamenti di stato. Per poter implementare questo approccio si devono definire due proprietà di osservazione dette *Observability radius* (raggio di osservabilità) e *Observation radius* (raggio di osservazione) (figura 3.2a). La prima definisce la distanza da cui un'entità può essere percepita da un osservatore, mentre la seconda specifica il raggio in cui un entità può percepire altri oggetti. Per esempio, un'entità aumentata utilizzata per contare tutte le persone che entrano in un teatro, dovrà avere un raggio di osservazione di una certa misura (la larghezza della porta del teatro), ma potrebbe avere un raggio di osservabilità nullo, per non essere percepito da nessuno.

In un AW anche queste ultime due proprietà potrebbero cambiare durante lo svolgimento dell'applicazione.

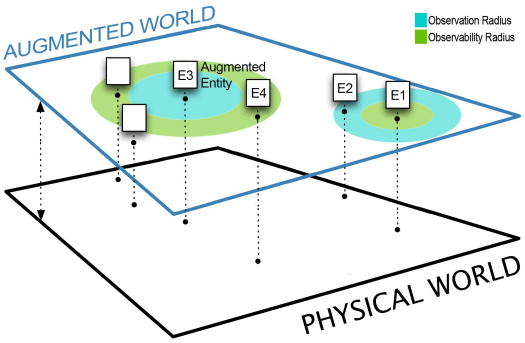
Modellazione degli utenti e interazione

Uno degli aspetti principali in un AW è l'interazione con l'utente, infatti, attraverso questa idea, si cercano di modellare applicazioni dove gli utenti e le entità aumentate svolgono una qualche attività in un ambiente fisico interagendo tra loro. Un possibile modo per modellare queste interazioni è creare, per ogni utente, un *augmented body* nel mondo virtuale dotato di uno stato rappresentante le informazioni dell'utente (per esempio la sua posizione o temperatura corporea) e di alcune funzioni per modellare l'input/output. L'utente, in questo modo, può sfruttare le proprietà di osservazione descritte sopra per percepire lo stato delle entità e, in caso sia necessario, visualizzarle, per esempio attraverso un ologramma basato sulle loro proprietà geometriche (figura 3.2b). Bisogna anche considerare che, come un' utente può osservare le entità, anche queste ultime possono percepire la presenza dell'utente e reagire in base al suo operato.

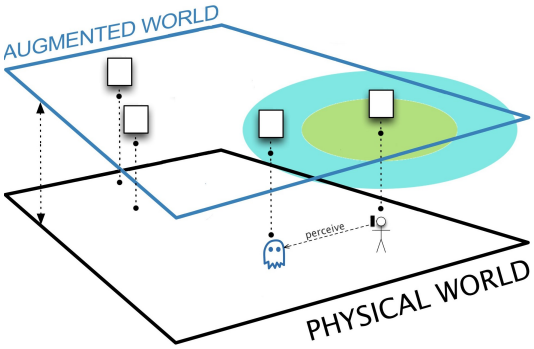
Estendere le funzionalità degli oggetti fisici

Il concetto di "augmentation" si riferisce anche all'incremento delle potenzialità e delle funzionalità degli oggetti fisici. Questa estensione può essere fatta semplicemente fornendo delle interfacce virtuali dalle quali controllare lo stato degli oggetti fisici, ma anche sfruttando le capacità computazionali dell'

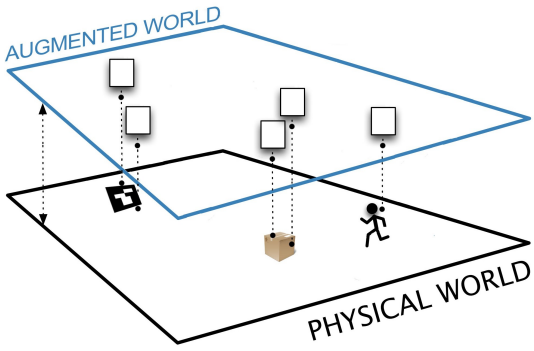
AW per fornire più funzioni all'oggetto (figura 3.2c). Per esempio, un semplice vaso di fiori da esterno, potrebbe disporre di un'interfaccia virtuale per controllare i dati riguardanti lo stato del vegetale (temperatura, umidità, luminosità, pressione), ma anche utilizzare l'AW per incrociare i dati e avvisare l'utente dell'arrivo di un imminente precipitazione, in modo che esso non annaffi. Come già accennato in precedenza questo tipo di "augmentation" necessita di un accoppiamento tra le entità aumentate e i loro corrispettivi fisici, in modo che il mondo virtuale sia a conoscenza dei cambiamenti che avvengono nel mondo fisico. Se questo accoppiamento viene interrotto, per esempio a causa di mancanza di connessione, sarà necessario disporre di un meccanismo per fare in modo che il mondo aumentato si renda conto che i dati di cui dispone non sono più attendibili. Una possibile soluzione a questo problema consiste nell'utilizzo di meta-dati che potrebbero specificare, per esempio, la scadenza delle informazioni.



(a) In questa immagine si nota come E1 possa percepire E2, ma non possa essere osservato da altre entità dato il suo limitato observability radius. L'entità 3 invece può essere osservata dalla E4 e dalle altre entità all'interno del suo raggio di osservabilità



(b) Qui di sopra si mostra come un utente dotato di un HMD o di uno smartphone possa visualizzare il fantasma raffigurato, poichè all'interno del suo raggio di osservazione



(c) Accoppiamento 1-N di oggetti fisici con entità aumentate

Figura 3.2: Immagini prese da [3]

3.1.2 Modelli di programmazione per AW

In base ai concetti descritti sopra, si possono pensare di utilizzare più sistemi per sviluppare AWs. La programmazione object-oriented (OOP) potrebbe essere un metodo efficace per rappresentare i mondi aumentati, infatti, si potrebbero utilizzare semplici oggetti per modellare le entità aumentate, incapsulando il loro stato e comportamento. In questo modo, inoltre, sarebbe possibile costruire gerarchie di entità tramite i meccanismi di ereditarietà. In questo paradigma, il mondo, potrebbe essere rappresentato come un semplice contenitore di entità con funzioni in grado di gestirle. Le proprietà di osservazione potrebbero essere modellate attraverso il pattern Observer (figura 3.3).

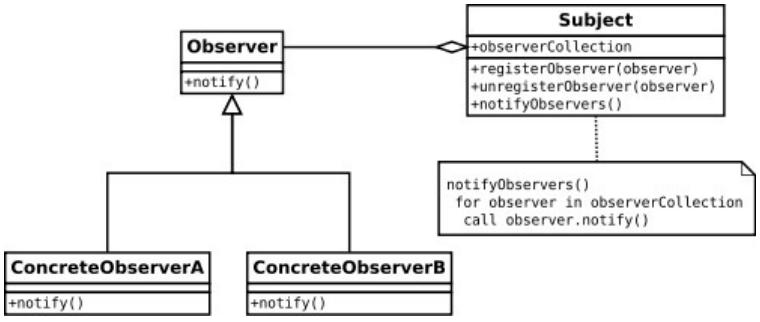


Figura 3.3: Diagramma UML della struttura del pattern observer, da Wikipedia

L'OOP, però, non ha alcun meccanismo veramente efficace per rappresentare modelli concorrenti e con interazioni asincrone, fondamentali nei mondi aumentati. Gli AW, infatti, sono intrinsecamente sistemi concorrenti, dove le entità aumentate rappresentano oggetti indipendenti le cui funzioni sono scatenate da eventi asincroni, come le azioni dell'utente o l'interazione con le altre entità. Questi aspetti possono essere catturati da modelli di programmazione più avanzati come quelli basati sugli attori, in questo modo, ogni interazione viene modellata come un passaggio diretto di messaggi asincroni tra entità. Le caratteristiche di osservazione precedentemente citate, in questo caso, possono essere implementate in un framework sottostante, oppure direttamente catturate estendendo il modello base ad attori in uno ad agenti. In quest'ultimo caso, gli agenti diventerebbero personaggi del mondo virtuale con la possibilità di agire su tutte le entità aumentate. Qui di seguito si mostra un primo esempio di agent-oriented augmented world, basato sull'idea dei *Mirror Worlds*.

3.1.3 Mirror Worlds

In [4] si definisce l'idea di Mirror World come "*un mondo digitale formato come un sistema multi agente, che vive in uno spazio virtuale accoppiato ad un ambiente fisico, in modo da aumentare le funzionalità e le capacità delle persone che vivono e lavorano al suo interno*", secondo questa visone, quindi nel mondo virtuale "vivono" insiemi di agenti software che aumentano alcune caratteristiche della porzione del mondo fisico a cui sono collegati. Il nome mirror world suggerisce una forma di "rispecchiamento" tra il mondo fisico e quello virtuale, infatti, in questa idea, ogni entità fisica ha una controparte digitale che può essere percepita dagli abitanti del mondo virtuale (i.e. gli agenti), e ogni entità virtuale ha una controparte fisica, come un ologramma percepibile tramite AR, in modo da permettere l'interazione con gli umani.

Un semplice esempio di questo concetto può essere esaminato nel gioco di realtà aumentata Ghost in the city (figura 3.4) dove il MW è composto da una serie di tesori e un gruppo di fantasmi distribuiti in una zona della città. Il gioco si compone anche di due squadre formate da giocatori con smart glasses e smartphone, con questi ultimi che vengono utilizzati come "bacchetta magica". Lo scopo del gioco è collezionare più tesori possibili senza essere toccati dai fantasmi, che sono gestiti da agenti e si muovono liberamente nel MW. Gli utenti visualizzano i fantasmi tramite gli smart glasses e quest'ultimi iniziano a inseguire i giocatori quando entrano all'interno di un certo raggio d'azione. Quando un fantasma tocca un giocatore esso riceve uno stimolo fisico, in questo caso la vibrazione dello smartphone. Gli spettri, inoltre, possono avere delle preferenze sui luoghi da attraversare, per esempio in base alla temperatura o luminosità della zona.

Questo esempio, seppur relativo a un semplice gioco, può riassumere bene le caratteristiche di un MW.

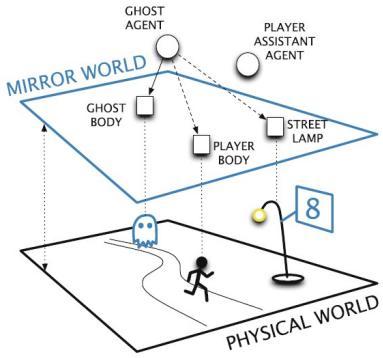


Figura 3.4: Rappresentazione astratta di un mirror world, in riferimento al gioco ghost in the city, da [4]

I MWs si basano sul modello Agent and Artifacts (A&A), un astrazione introdotta per la modellazione di sistemi multi agente. Gli Artifacts, o artefatti, possono modellare qualsiasi risorsa che può essere utilizzata da un'agente durante la sua attività, infatti, hanno uno stato osservabile e un insieme di operazioni mediante le quali gli agenti possono modificare il mondo. L'osservazione degli agenti rispetto agli artefatti è totalmente asincrona, infatti, successivamente ad una modifica dello stato, l'artefatto notifica tutti i suoi osservatori. Gli insiemi di artefatti vengono detti workspace. Maggiori dettagli su questo paradigma possono essere trovati in [3, 4]

Mirror world come Augmented worlds

Un mirror world può essere quindi descritto come un augmented world sviluppato utilizzando la filosofia agent-oriented, infatti, esso implementa tutte le caratteristiche definite precedentemente, come spiegheremo qui di seguito.

L'allineamento spaziale degli artefatti avviene definendo il concetto di *mirror workspace* (figura 3.5), che estende il normale workspace del modello A&A aggiungendo una collocazione fisica degli artefatti, implementabile, per esempio, attraverso una mappa.

L'osservazione degli artefatti in un MW può avvenire secondo due modalità, la prima, detta focusing, equivale ad un'azione di subscribe come quella che avviene in un qualsiasi sistema publish-subscribe. In questo caso il subscriberer, che potrebbe essere rappresentato dall'agente, si "registra" agli aggiornamenti del publisher, l'artefatto. La seconda modalità, caratteristica dei sistemi che hanno una componente fisica, è la percezione in base alla posizione nel mondo reale, utilizzando i concetti di observability radius e observation radius definiti in precedenza. Per poter utilizzare questo metodo ogni agente deve avere un "corpo" nel mondo reale che ne identifichi la posizione. Gli agenti con questa caratteristica vengono detti *mirror agents*.

Le interazioni con l'utente in un mirror world avvengono attraverso l'agente collegato alla persona coinvolta nell'esperienza, quest'ultimo può essere accoppiato, per esempio, all' HMD indossato. Quando questo agente percepisce un artefatto lo può segnalare all'utente mostrando un messaggio, o un ologramma, attraverso il dispositivo al quale è collegato.

Gli artefatti in un MW potrebbero essere completamente virtuali oppure essere accoppiati ad un oggetto fisico realmente esistente ed estenderne le funzionalità. Nel primo caso la posizione geografica dell'oggetto deve essere definita alla creazione dell'artefatto e cambierà in base alle azioni svolte su di esso, mentre nel secondo bisognerà tener sincronizzati i dati del *mirror artifact* in base alla posizione dell'oggetto fisico e ai suoi movimenti.

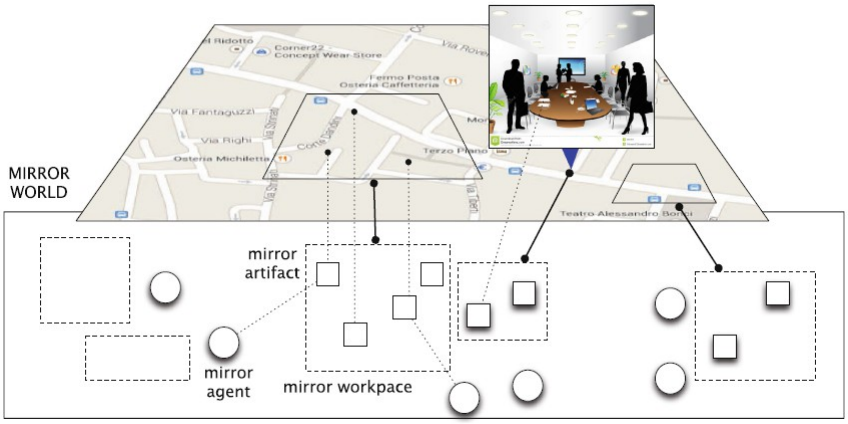


Figura 3.5: Rappresentazione di mirror agents, mirror workspaces e mirror artifacts all'interno di un mirror world, da [4]

3.2 Future direzioni della ricerca

Lo sviluppo di applicazioni basate su AW o MW fa emergere problemi particolari ancora non del tutto risolti. Una delle sfide principali riguarda il "grado di real-time" nella sincronizzazione tra mondo fisico e mondo aumentato, ciò ha una grande importanza in quanto modifica il modo in cui le persone coinvolte nell'applicazione percepiscono la realtà e quindi influenza le loro azioni.

In tutti i mondi aumentati, compresi i più semplici, è essere presente sempre una parte di rete, ciò fa in modo che tutta una serie di problemi collegata al network, come la latenza e la mancanza di connessione, debbano essere affrontati anche in queste applicazioni.

Inoltre è bene specificare che, come nei sistemi distribuiti in generale, nei mondi aumentati solitamente non è presente un orologio globale e collettivo, anzi, si può assumere che ogni entità abbia il suo riferimento temporale locale e ciò può provocare problemi nell'ordinamento degli eventi. In questo contesto bisogna garantire la consistenza delle relazioni causa-effetto, per esempio, se un artefatto produce una sequenza di eventi relativi ad un cambio dello stato, la stessa sequenza deve essere percepita da tutti gli osservatori dello stesso.

Un'altra analisi interessante che deve essere ancora sviluppata è quella riguardo agli strumenti di supporto allo sviluppo e al debugging di queste applicazioni, che dovranno avere caratteristiche tali da affrontare gli attributi di un AW sopra descritti.

Capitolo 4

Studio di un framework per Augmented Worlds basato su Unity e Android

4.1 Definizione del problema

Il progetto di tesi si basa sulla necessità di trovare la maniera di dividere in modo netto view, model e controller di un'applicazione di realtà aumentata, in modo che ci sia un componente che si occupi solo degli ologrammi, uno solo della logica dell'applicazione e uno solo dello stato delle entità. Molto spesso, infatti, sviluppando queste applicazioni utilizzando i framework a disposizione, non si riesce ad ottenere il grado di interdipendenza tra le parti necessario per produrre un software decentemente riusabile e mantenibile.

L'obiettivo del progetto, quindi, è la definizione di un'architettura che separi chiaramente le parti del software, in particolare la view dal model.

La parte di framework da sviluppare si baserà sulle tecnologie illustrate nel nel capitolo 2, cioè Unity e Vuforia, cercando però di adattarle alla visione di Augmented World mostrata in precedenza. Si prenderà come riferimento il caso in cui ogni entità abbia un solo ologramma, eventualmente composto da più oggetti. Prima di entrare nel dettaglio della struttura del software realizzato esaminiamo le piattaforme definite come case-study.

4.2 Case - study: Unity-Vuforia-Epson Moverio BT-200

L'applicazione sviluppata durante la tesi, ha utilizzato uno stack abbastanza particolare. Utilizzando gli Epson Moverio è stato necessario generare

un'app Android, ma questo non è stato fatto mediante l'utilizzo di Android Studio e gli strumenti per la generazione nativa android, bensì utilizzando Unity e il framework Vuforia per la realtà aumentata (figura 4.1). Per la parte *core* è poi stato aggiunto un plugin Android al progetto Unity come spiegheremo successivamente.

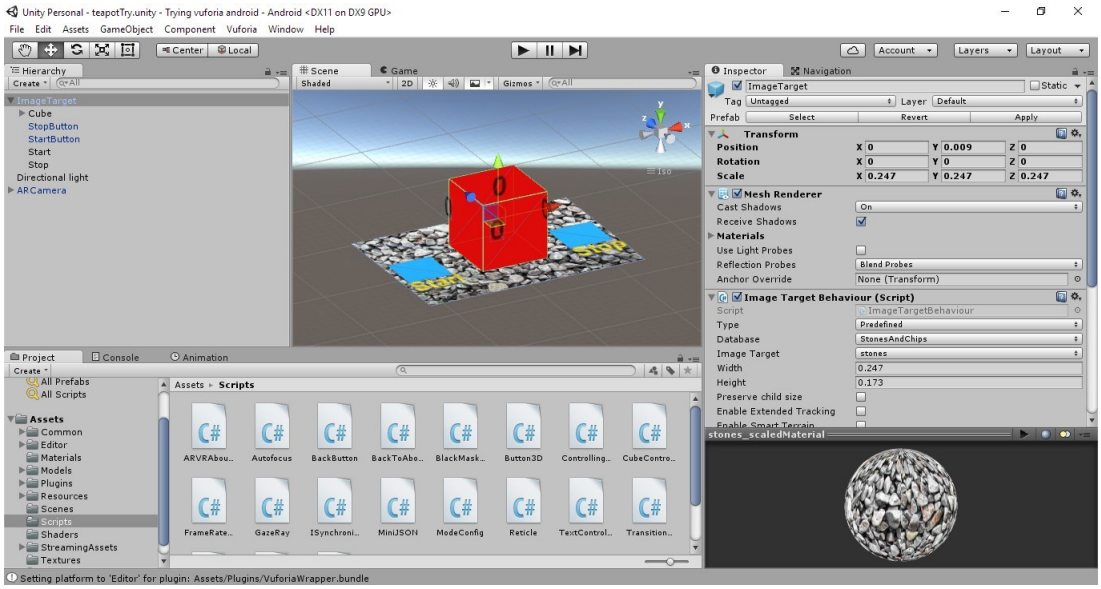


Figura 4.1: Screenshot di un progetto Unity che utilizza Vuforia

4.3 Progettazione del software

Per la parte sperimentale della tesi è stato deciso di creare un prototipo che possa servire come base per la costruzione di un framework di realtà aumentata in Java, utilizzando Unity solo per la gestione degli ologrammi da visualizzare.

4.3.1 Entità aumentate e ologrammi

In una qualsiasi applicazione di realtà aumentata sono presenti entità aumentate e ologrammi associati ad esse, ma molto spesso questa associazione è troppo forte, tanto che l'uno dipende in gran parte dall'altro. Nel nostro prototipo si è deciso di distinguere in modo netto queste due parti. Le entità aumentate, come spiegato nel capitolo 3, sono dei componenti che vivono e hanno un comportamento all'interno del mondo virtuale, esse poi possono essere associate ad un punto nel mondo reale in vari modi, per esempio utilizzando un marker per allineare i sistemi di riferimento. Un'entità non deve essere forzatamente visibile all'utilizzatore dell'applicazione, infatti, essa potrebbe

manifestarsi tramite un ologramma, come una melodia, oppure non mostrarsi affatto e raccogliere dati dal mondo reale senza che l'utente se ne renda conto. Si nota, quindi, come gli ologrammi siano solo una possibile rappresentazione di un'entità aumentata e quindi debbano essere del tutto separati dalla logica applicativa, per fare in modo che si possa semplicemente cambiare la View per modificare l'aspetto dell'applicazione.

4.3.2 Architettura generale del prototipo

Il prototipo si basa sulla definizione di due "mondi", uno relativo agli ologrammi e uno relativo alle entità aumentate. La comunicazione tra i due mondi avviene per sincronizzazione e basandosi su un sistema di riferimento condiviso, cioè il marker principale, che permette l'allineamento.

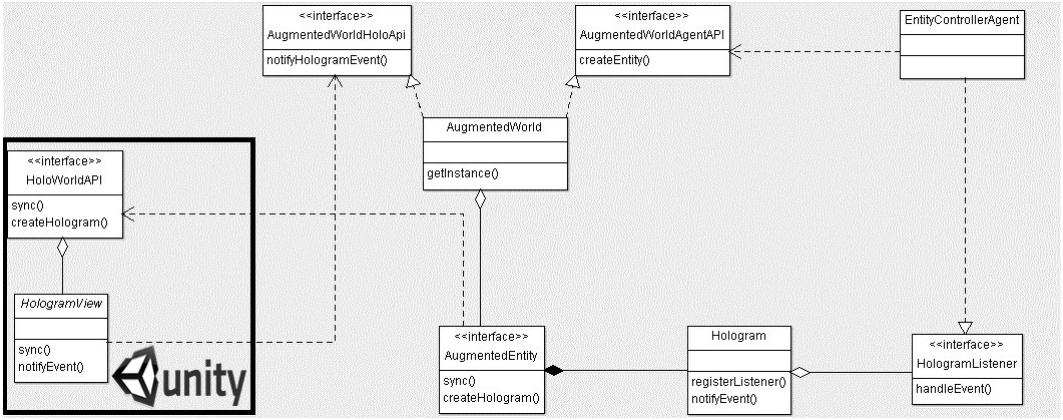


Figura 4.2: Struttura dell'applicazione

Come visibile dalla figura 4.2, il "mondo aumentato"(AugmentedWorld) contiene delle entità che possono chiamare i metodi del "mondo degli ologrammi"(HoloWorldAPI), per creare oggetti 3D ad esse associati o per sincronizzarli in base al nuovo stato. Si noti, inoltre, come anche gli ologrammi siano parte del mondo aumentato e solo la loro visualizzazione sia demandata all' HoloWorld.

HologramView è la classe astratta, lato Unity, simmetrica a quella del model. Questa viene estesa dai vari ologrammi in modo da ricevere la sincronizzazione del mondo aumentato e propagarla ai componenti 3D visualizzabili, contenuti al loro interno. HoloWorldAPI, quindi, deve avere i riferimenti a tutti questi elementi per poter chiamare la *sync* su di essi.

La gestione degli eventi generati dagli ologrammi parte dall'HologramView, come è possibile vedere dallo schema, quest'ultima notifica l'evento tramite

l'interfaccia messa a disposizione dall'AugmentedWorld, (AugmentedWorldHoloAPI) da cui viene spedito all'ologramma corrispondente. Se l'oggetto 3D da cui parte l'evento ha, nel mondo aumentato, un listener, viene scatenato l'handler relativo a quell'evento.

Per esempio, per fare in modo che al "tocco" di un utente su un ologramma di un cubo, esso inizi a ruotare, bisognerebbe fare in modo che il suo listener abbia un handler per quell'evento che abiliti la rotazione dell'entità aumentata associata, la rotazione effettiva verrà visualizzata alla *sync* successiva.

Abbiamo trascurato finora ciò che nello schema è chiamato EntityControllerAgent, esso forma una delle parti più importanti dell'applicazione, rappresenta, infatti, il flusso di controllo centrale della stessa, colui che gestisce e crea le entità aumentate e fa da listener per gli eventi relativi agli ologrammi. L'agente interagisce con il mondo tramite l'interfaccia AugmentedWorldAgentAPI, da cui ricava i riferimenti alle entità aumentate. Attraverso questi, poi, chiama i metodi che ne provocano un cambiamento dello stato.

Nel prototipo è presente un solo agente, ma per applicazioni di più grande calibro sarà necessario crearne altri, divisi per scopo o categoria, per separare meglio i compiti. Questo porterà anche alla necessità di gestire con più attenzione la concorrenza tra i vari flussi di controllo.

Bisogna far notare che, solitamente, un'agente è un'entità puramente attiva e che quindi non riceve chiamate sui propri metodi, per semplicità, però, nella nostra applicazione all'arrivo di un evento relativo ad un ologramma viene richiamata una funzione dell'agente stesso. Il comportamento di cui sopra potrà essere migliorato in futuro.

4.3.3 Architettura dettagliata

Qui di seguito si vanno a mostrare dei diagrammi utili a esplorare meglio i dettagli dell'architettura.

Dettaglio parte in Unity

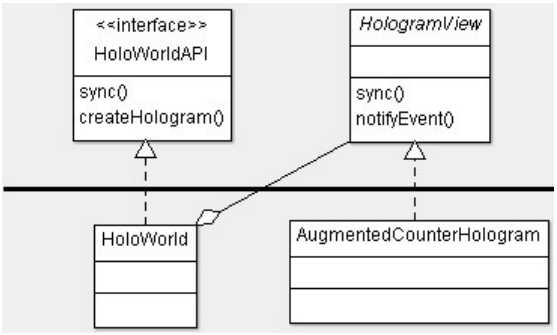


Figura 4.3: Dettaglio parte in Unity

Come è possibile vedere dalla figura 4.3 il mondo è rappresentato da un' unica classe (HoloWorld) che implementa i metodi dell'interfaccia, quest'ultima, inoltre, è l'unica che ha la funzione per creare effettivamente un ologramma all'interno della View di Unity.

Il mondo contiene una mappa chiave-valore che specifica l'associazione tra le entità aumentate e il loro ologramma, essa viene popolata durante la *createHologram*. All'arrivo di una *sync* il mondo cerca l'HologramView relativo all'entità chiamante e, se presente, avvia la sincronizzazione su di esso. La logica di rendering degli ologrammi è incapsulata dentro le classi che estendono HologramView, questo potrebbe facilitare la divisione dei compiti nello sviluppo dell'applicazione, infatti, la scrittura di questi componenti potrebbe essere demandata ad artisti o esperti di Unity. Nell'immagine si mostra la separazione tra le classi di framework e la loro effettiva implementazione relativa al prototipo, in questo caso è presente una sola classe di tipo HologramView, ma, in una applicazione reale potrebbe essere costruita una più complessa gerarchia di classi per definire tutti gli ologrammi da mostrare nell'ambiente.

Dettaglio parte Android/Java

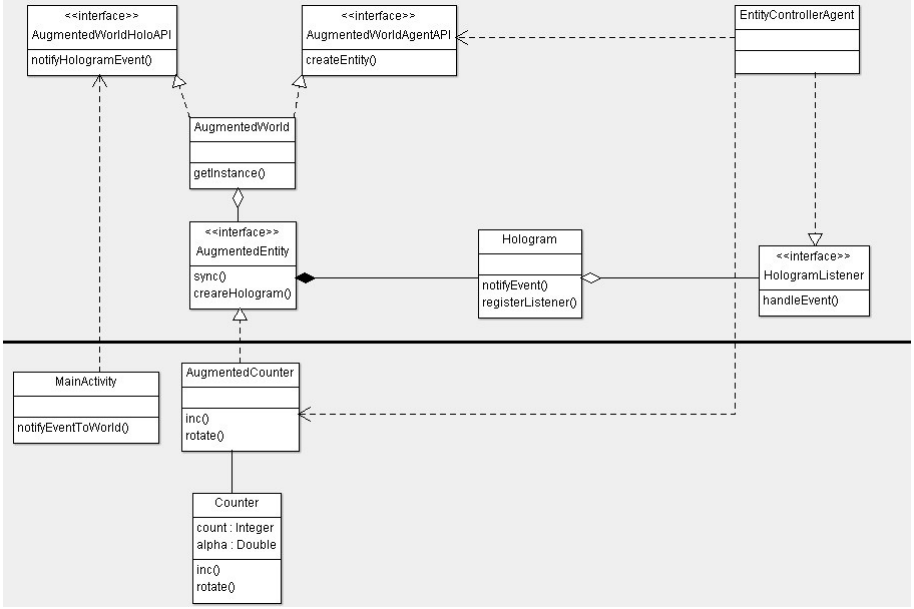


Figura 4.4: Dettaglio parte Android/Java

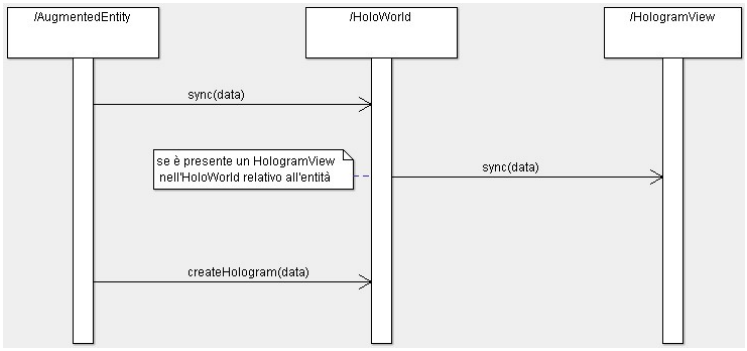
Nella parte Java mostrata qui sopra, si nota, innanzitutto, che l'agente opera direttamente sulle entità aumentate chiamando i metodi che esse espongono, senza mai chiamare direttamente la *sync* per aggiornare la View. La logica di chiamata del metodo di sincronizzazione, infatti, è incapsulata dentro le entità aumentate stesse, per esempio, il metodo *inc* dell'AugmentedCounter dovrà chiamare la *sync* solo nel caso in cui sia presente un ologramma che possa visualizzare il valore del conteggio.

In figura 4.4 si può vedere anche come l'AugmentedCounter sia formato da un semplice contatore con due campi: count e alpha; il primo rappresenta il valore del conteggio, mentre il secondo un angolo di rotazione. Questi due attribuiti formano lo stato dell'entità che verrà sincronizzato con gli ologrammi. Come ultimo dettaglio facciamo notare la presenza della MainActivity, questa classe è l'unica caratteristica di Android ed è utile alla ricezione dei messaggi dalla parte Unity come spiegheremo più in dettaglio successivamente.

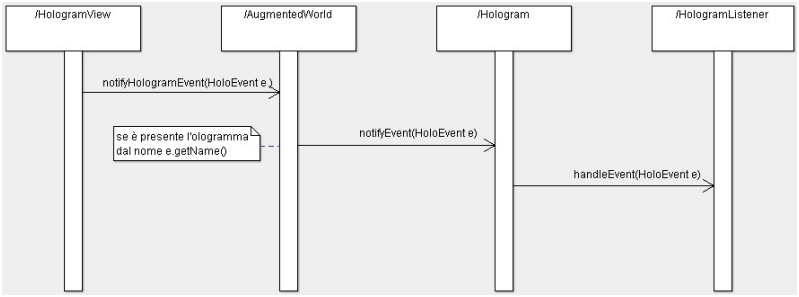
Anche qui si nota come, per poter aggiungere nuove entità, basti implementare le interfacce di framework senza dover modificare altre parti del progetto.

Comunicazione

In questo paragrafo discuteremo ad alto livello di come sia stata gestita la comunicazione nel progetto, i dettagli implementativi saranno riferiti nelle note tecniche.



(a) Invio di messaggi per sincronizzazione e creazione ologrammi



(b) Notifiche di eventi sugli ologrammi

Figura 4.5: Gestione della comunicazione

Nelle figure 4.5 si nota come la comunicazione all'interno dell'applicazione avvenga tramite uno scambio di messaggi totalmente asincrono tra la parte Unity e la parte Java. In 4.5a si mostra l'invio dei messaggi relativi alla sincronizzazione della view e alla creazione degli ologrammi. L'AugmentedEntity può, infatti, chiamare i metodi *sync* e *createHologram* del mondo degli ologrammi, nel primo caso HoloWorld provvederà a propagare la *sync* all'ologramma relativo all'entità, mentre nel secondo genererà dinamicamente un nuovo HologramView in base ai dati passati.

La figura 4.5b evidenzia, invece, la gestione degli eventi relativi agli ologrammi. In questo caso, la classe lato Unity specifica dell'ologramma notifica l'evento al mondo aumentato che lo invierà all' Hologram corretto (se presente). L'oggetto selezionato segnalerà l'evento a tutti i suoi listener (pattern Observer) che provvederanno a gestirlo e a scatenare i vari handler.

4.4 Note tecniche

4.4.1 Applicazioni utilizzate

Durante lo sviluppo del progetto si sono utilizzate varie applicazioni:

- Android Studio per la generazione del plugin android contenente la parte *core* dell'applicazione
- Unity per la gestione degli oggetti 3D e l'integrazione con Vuforia
- Microsoft Visual Studio per l'editing degli script C# di Unity

Tra gli strumenti solo Android Studio è completamente open-source, mentre Unity è utilizzabile in modo gratuito solo per scopi non professionali. Visual Studio invece ha bisogno di una licenza, per evitare questa complicazione si può utilizzare il meno potente, ma gratuito, MonoDevelop, l' IDE integrato in Unity.

4.4.2 Generazione plugin Android per Unity

Dopo aver verificato la possibilità di generare con Unity un'app per i Moverio, ci siamo posti il problema di come aggiungere il modulo Java con la parte *core* dell'applicazione. Per fare ciò si è costruita una libreria Android strutturata in modo da poter comunicare con Unity, cioè con una MainActivity senza layout che estende UnityNativePlayerActivity. La libreria è stata successivamente compilata in un jar e posta nella sezione \Assets\Plugins\Android del progetto Unity. Qui di seguito lo script Gradle per generare il jar da AndroidStudio.

```
task deleteOldJar(type: Delete) {
    delete 'libs/AndroidPlugin.jar'
}

task exportJar(type: Copy) {
    from('build/intermediates/bundles/release/')
    into('libs/')
    include('classes.jar')
    include('Vuforia.jar')
    include('VuforiaUnityPlayer.jar')
    rename('classes.jar', 'ControllingUnityPlugin.jar')
}
exportJar.dependsOn(deleteOldJar, build)
```

Listato 4.1: Script Gradle

4.4.3 Comunicazione Unity - Android

La comunicazione tra Unity e il plugin avviene in modo differente a seconda della direzione, infatti, per chiamare metodi di script Unity(C#) da Android si usa uno scambio di messaggi, mentre per il caso opposto si usa una sorta di Reflection con API messe a disposizione da Unity.

Da Android a Unity

Per poter invocare metodi di script si utilizza il metodo statico *UnityPlayer.UnitySendMessage* che ha bisogno del nome del component associato allo script, del metodo dello script e di un parametro per il metodo. Per garantire interoperabilità tra i linguaggi è stato utilizzato un JSON come ultimo parametro in tutti questi messaggi, in questo modo si può anche superare la limitazione dell'unico parametro richiesto semplicemente impacchettando più informazioni nello stesso JSON

```
UnityPlayer.UnitySendMessage(
        StringUtilities.WORLD,
        StringUtilities.SYNC_METHOD,
        createCounterJSON(this.counter)
        .toString());
```

Listato 4.2: Invio di un messaggio contente lo stato dell'entità aumentata

Da Unity ad Android

Tutta la comunicazione tra le classi C# e il plugin deve forzatamente passare attraverso la MainActivity, infatti, da script, si può ottenere il riferimento solo a quest'ultima. Nello snippet mostrato di seguito si spiega come effettuare una chiamata di metodo sull'Activity

```
    private static AndroidJavaClass jc;
    private AndroidJavaObject jo;

    void Start()
    {
        jc = new AndroidJavaClass("com.unity3d.player.UnityPlayer");
        jo = jc.GetStatic<AndroidJavaObject>("currentActivity");
        jo.Call("stampaDiDebug","Hello World");
    }
```

Listato 4.3: Esempio di chiamata di metodo sulla MainActivity

Struttura dei messaggi

Come già descritto, i messaggi tra le due parti del software sono stati strutturati come dei JSON. Per la parte Java sono state utilizzate le librerie di default del package *org.json*, mentre per Unity, a causa di particolari problemi con la versione del .NetFramework, si è scelto di adottare la classe *MiniJSON*[1].

I JSON relativi alla *sync* sono formati dalle caratteristiche dell'entità aumentata compreso l'identificativo della stessa (entityName), mentre quelli relativi alla *createHologram* contengono il nome della classe per la gestione dell'ologramma (name), l'entità chiamante (entityName) e le informazioni per il rendering. Di seguito mostriamo degli esempi.

```
{
   "counter":
   {
      "count":42,
      "alpha":0
   },
   "entityName":"AugmentedCounter0"
}
```

Listato 4.4: JSON per la sincronizzazione del contatore

```
{"renderInfo":
   {
      "scaleX":0.30,
      "scaleY":0.30,
      "scaleZ":0.30,
      "positionX":0,
      "positionY":0.15,
      "positionZ":-0.10
   },
   "name":"AugmentedCounterHologram",
   "entityName":"AugmentedCounter0"
}
```

Listato 4.5: JSON per la generazione dell'ologramma

[1] presa da gist.github.com/darktable/1411710

4.4.4 Particolari implementativi in Unity

In questa sezione si riporta il codice di alcune parti importanti per il funzionamento dell'applicazione.

Generazione oggetti di tipo HologramView

La *createHologram* dell'HoloWorld riceve i dati dall'entità aumentata e, per prima cosa, estrae dal JSON il nome della stessa e il nome della classe relativa all'ologramma, successivamente istanzia via Reflection l'HologramView specifica passando il Dictionary con i dati per il rendering. Infine associa, in un ulteriore Dictionary, l'ologramma appena creato all'entità relativa. Per maggiori dettagli si allega una parte di codice.

```
public void CreateHologram(string jsonHologramData)
{
   Dictionary<string,object> dict =
       MiniJSON.Json.Deserialize(jsonHologramData) as
       Dictionary<string,object>;
   string entityName = dict["entityName"] as string;
   string hologramName = dict["name"] as string;
   HologramView holo =
       Activator.CreateInstance(Type.GetType(hologramName),dict) as
       HologramView;
   holoList.Add(entityName,holo);
}
```

Listato 4.6: Generazione oggetto HologramView

Rendering ologrammi

La *createHologram* dell'HoloWorld passa quindi i dati per la strutturazione dell'ologramma alla classe specifica che li interpreta e elabora.

```
public void RenderComponents(Dictionary<string, object> dict)
{
   Dictionary<string, object> renderInfo = dict["renderInfo"] as
       Dictionary<string, object>;
   string name = dict["name"] as string;
   float scaleX = float.Parse(renderInfo["scaleX"].ToString());
   float scaleY = float.Parse(renderInfo["scaleY"].ToString());
   float scaleZ = float.Parse(renderInfo["scaleZ"].ToString());
   float positionX = float.Parse(renderInfo["positionX"].ToString());
   float positionY = float.Parse(renderInfo["positionY"].ToString());
   float positionZ = float.Parse(renderInfo["positionZ"].ToString());

   PrimitiveType type = getPrimitiveType("Cube");
   hologram = GameObject.CreatePrimitive(type);
   hologram.name = name;
   hologram.transform.localScale = new Vector3(scaleX, scaleY,
       scaleZ);
   hologram.transform.localPosition = new Vector3(positionX,
       positionY, positionZ);
   textObject.AddComponent<TextMesh>();
   textObject.AddComponent<MeshRenderer>();
   textMesh = textObject.GetComponent<TextMesh>();
   textMesh.text="0";
   textObject.GetComponent<MeshRenderer>().material.color=Color.black;
   textObject.name = "CounterText";
   textObject.transform.localScale = new Vector3(0.05f, 0.1f, 0.05f);
   textObject.transform.localPosition = new Vector3(positionX,
       positionY + 0.1f, positionZ - 0.3f);
   hologram.AddComponent(this.GetType());
}
```

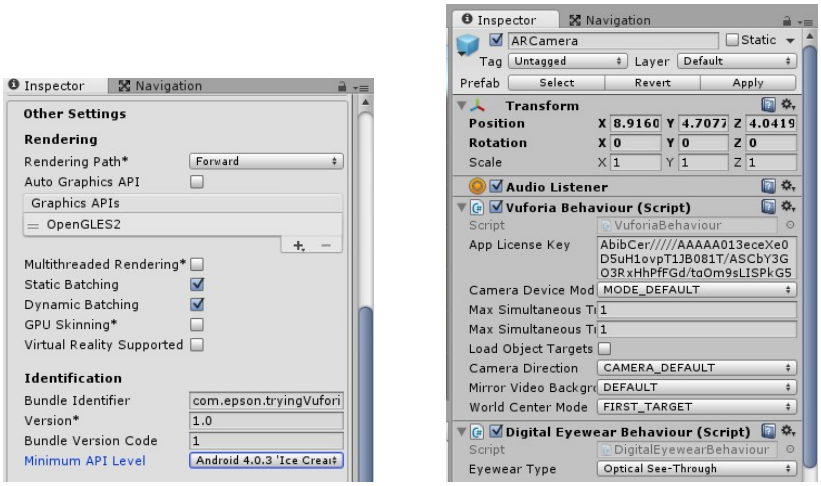
Listato 4.7: Setting delle impostazioni di rendering di un ologramma, in questo caso si generano un cubo e un oggetto di tipo testo

4.4.5 Debugging

Durante lo sviluppo del progetto si sono dovute utilizzare delle strategie di debugging particolari, infatti, essendo l'apk generato da Unity non è stato possibile utilizzare nè il debugger di AndroidStudio nè quello di VisualStudio. Un'ulteriore ostacolo è stato rappresentato dal fatto che non tutti i messaggi di log generati da Unity venivano mostrati nell'Android console. Per risolvere questi problemi si è deciso di utilizzare un semplice *print debugging* facendo in modo che i messaggi di Unity venissero stampati richiamando una funzione apposita nella MainActivity, utilizzando il metodo mostrato precedentemente.

4.4.6 Deploy sui Moverio

Per poter utilizzare l'applicazione sui Moverio sono stati necessari dei piccoli, ma fondamentali, accorgimenti (figura 4.6). I BT-200 utilizzano l' API 15 di Android quindi da Unity si è dovuto specificare di generare un apk di quel livello, inoltre, come già specificato più volte in precedenza, i Moverio sono un dispositivo optical-see-throught e quindi l'ARCamera è stata impostata secondo questa opzione. Successivamente per facilitare l'installazione è stato caricato l'apk in una cartella Dropbox condivisa con gli occhiali.



(a) UnityPlayer API settings (b) ARCamera settings

Figura 4.6

4.5 Valutazione del risultato

Il framework sviluppato, seppur semplice, propone un modo per sviluppare un'applicazione di realtà aumentata separando nettamente i concetti relativi al Model e alla View e, in particolare, le entità aumentate dai loro ologrammi. Questa divisione consente di poter modificare a piacimento la parte grafica dell'applicazione senza apportare modifiche alla logica, e quindi aumenta la riusabilità e manutenibilità del progetto.

Le applicazioni sviluppate soltanto con Unity, invece, sono molto concentrate sulla struttura degli oggetti 2D-3D e le loro interazioni, come per esempio le collisioni. Questa logica di sviluppo può andare bene per il game-design, ma per delle applicazioni con un'ampia parte non grafica e di logica pura, come quelle distribuite, può risultare sconveniente e poco mantenibile.

Durante le ultime fasi dello sviluppo del prototipo si è notato che le modifiche alla parte Unity provocavano cambiamenti minimi o nulli sulla parte Java, per cui credo che si possa considerare il lavoro svolto, almeno in parte, riuscito.

Conclusioni

L'Augmented Reality sarà molto probabilmente un ambito che risulterà molto importante negli anni a venire, portando innovazione negli ambiti più disparati, come quello medico, militare, scolastico e turistico e rivoluzionando il nostro modo di vivere. Gli smart-glasses e in generale i dispositivi wearable continueranno a diminuire nelle dimensioni fino ad diventare quasi invisibili e a confondersi con il corpo dell'utente, in questo modo le persone percepiranno sempre di più l'allineamento tra mondo reale e virtuale e, quando si potranno renderizzare in real-time ologrammi ad alta definizione, essi diventeranno quasi indistinguibili e probabilmente riusciranno ad ingannare la mente umana.

Lo sviluppo di applicazioni di realtà aumentata, quindi, diventerà un importantissimo aspetto nell'ambito della programmazione e si studieranno vari metodi e modelli per strutturare bene ed ingegnerizzare il processo. Credo che i framework per AR diventeranno sempre più potenti e semplici da usare fornendo supporto a vari paradigmi di programmazione, come per esempio l'Object-Oriented o quello ad agenti, in modo da poter essere utilizzati per qualsiasi esigenza.

Il mio progetto di tesi, in questo contesto, si poneva come base l'obiettivo di sviluppare un framework riusabile che distinguesse la logica dell'applicazione dalla gestione degli ologrammi, in modo da poter lavorare sulle entità aumentate in modo totalmente indipendente. Credo che l'obiettivo sia stato in buona parte raggiunto anche se, sicuramente, il progetto potrà essere migliorato e reso più elegante ed estendibile.

Questa tesi, in conclusione, ha cercato di introdurre i concetti relativi alla realtà aumentata e alle sue tecnologie di rifermento, cercando di dare un idea su come ora si sviluppano le applicazioni e come si pensa che verranno sviluppate in futuro, proponendo alcune visioni possibili e implementando un semplice prototipo di framework.

Sviluppi Futuri

Durante lo sviluppo del progetto sono stati incontrati alcuni problemi e limitazioni che potrebbero essere migliorati in futuro, di seguito alcuni esempi.

Il meccanismo di scripting di Unity per gli ologrammi non si adatta molto bene a una gestione a oggetti del mondo aumentato, per cui nel prototipo si è deciso di rendererizzare gli oggetti 3D in modo totalmente dinamico (tramite codice) ricevendo i dati necessari dall'entità, questa modalità, però, limita la complessità degli ologrammi generabili e dovrà essere raffinata in futuro per poter utilizzare il framework per la costruzione di applicazioni reali.

Il prototipo al momento utilizza Unity, Vuforia e Java, in futuro, si potrebbe estendere il supporto ad altri framework come AR-Toolkit e Wikitude e a altri engine che li utilizzano, comunque in questo caso la parte Java dovrebbe rimanere pressoché invariata, eccezion fatta per la comunicazione.

Al momento, inoltre, il framework si basa su una semplice estensione ad agenti di un modello classico ad oggetti, per poter costruire in modo migliore un sistema distribuito si potrebbe convertire questo modello in uno più orientato agli agenti come nell'idea dei Mirror Worlds.

Per concludere credo che il prototipo, seppur semplice, potrebbe essere esteso in modo da ottenere qualcosa di effettivamente utilizzabile per costruire applicazioni di complessità tale da diventare effettivamente competitive.

Ringraziamenti

Vorrei ringraziare il Prof. Alessandro Ricci per avermi permesso di affrontare questo progetto e l'Ing. Angelo Croatti per il supporto e la disponibilità forniti durante la stesura della tesi.

Un ringraziamento speciale a tutti quelli che mi hanno sostenuto durante il percorso accademico, in particolare:
La mia famiglia: Celestina, Leda, Delio, Patrizia, Piergy, Ale, Giuli, Franci, Lelli, Alberto e Pinolo
I miei amici: Edoardo, Jacopo, Jin, Michele, Nicola, Riccardo, Samuele, Tommaso, Alberto G., Alberto M., Alessio, Chiara, Christian, Emanuele, Marco, Simone, Stefano.

Bibliografia

[1] W. Barfield, *Fundamentals of Wearable Computers and Augmented Reality (2nd edition)*, CRC Press, 2016.

[2] A.B. Craig, *Understanding Augmented Reality - Concepts and Applications*, Morgan Kaufmann, 2013.

[3] A.Croatti, A.Ricci, *Programming Abstractions for Augmented Worlds (unpublished)*, 2015.
soft.vub.ac.be/AGERE15/papers/AGERE_2015_paper_5.pdf

[4] A.Ricci, A.Croatti, P.Brunetti, M.Viroli, *Programming Mirror Worlds, An Agent-Oriented Programming Perspective*, 2015.

[5] Unity3D, *Sito ufficiale Unity, unity3d.com.*

[6] Vuforia, *Sito ufficiale Vuforia, www.vuforia.com.*

[7] Epson Moverio, *Sito ufficiale Epson Moverio, moverio.epson.com.*