ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

In

Elettronica T-1

REALTÀ VIRTUALE MULTIUTENTE

CANDIDATO:

Pierfrancesco Soffritti

RELATORE:

Chiar.mo Prof. Bruno Riccò

CORRELATORE:

Ing. Marco Marchesi

Anno Accademico 2015/2016

Sessione I

Sommario

1 Introduzione

Nel corso degli ultimi anni, in particolare in quest'ultimo, abbiamo assistito ad un notevole sviluppo tecnologico nell'ambito della Virtual Reality (VR). Sebbene in passato siano stati fatti tentavi per diffondere la VR, le tecnologie ancora acerbe ne hanno impedito il successo. Quest'anno invece, con la commercializzazione dei visori di nuova generazione (ad esempio Oculus Rift e HTC Vive, per citare i più famosi) e gli investimenti fatti dalle principali aziende del settore (Google, Sony, Facebook, Microsoft) potrebbe essere l'inizio di una svolta, portando VR di qualità alla portata di tutti.

Essendo ancora ai suoi albori però, la VR ha tre principali problematiche:

- un prezzo per molti proibitivo
- modalità d'interazione ancora acerbe
- isolamento dell'utente

Per poter godere di un'esperienza gradevole sono richieste macchine estremamente potenti e visori di alta qualità, entrambi dal prezzo elevato, il che costituisce una forte limitazione alla diffusione della tecnologia.
L'alternativa al problema del costo finora è stata una: Google Cardboard [1]. Un progetto open source portato avanti da Google, che propone di utilizzare uno smartphone come dispositivo di elaborazione, display e input/output. Inserendo lo smartphone all'interno di un apposito supporto (più o meno sofisticato) è possibile avere tutto ciò che serve per accedere alla VR.
Questa soluzione, sebbene estremamente economica, comporta alcune significative problematiche. I telefoni non sono ancora abbastanza potenti per elaborare VR di qualità e i sensori al loro interno in molti casi mancano dell'accuratezza necessaria.
Finora la VR di Google è stata un esperimento più che un vero prodotto commerciale. Recentemente però l'azienda ha presentato una nuova piattaforma, Daydream [2], con l'intenzione di definire standard precisi per la VR mobile. In questo modo gli sviluppatori e i produttori hardware avranno la possibilità di sviluppare applicazioni e dispositivi di qualità in ambito VR.

Il problema relativo all'isolamento dell'utente è molto sentito. Quasi tutte le esperienze in VR finora disponibili sono esperienze solitarie, inoltre, il fatto di indossare un visore che separa l'utente dall'ambiente circostante non fa che accentuare questo aspetto. Nonostante ciò, le applicazioni più interessanti per la VR potrebbero proprio essere quelle con fini sociali, in cui più persone possono condividere lo stesso spazio virtuale e interagire come se fossero nella stessa stanza.
Parecchi esperimenti sono stati svolti in questo ambito, molti dei quali hanno avuto esiti positivi ed evidenziato le potenzialità sociali di questa tecnologia. Il fatto che Facebook abbia acquistato Oculus è già di per sé prova dell'interesse nei confronti della social VR.

Relativamente alle modalità d'interazione, ogni produttore sta proponendo le proprie soluzioni al problema, sia dal punto di vista hardware che software.
Per ora la soluzione hardware più efficace è quella di HTC Vive [3], con tracciamento spaziale del giocatore e tracking della posizione delle mani tramite appositi controller.

Le problematiche relative all'interazione non sono argomento di questa tesi, lo sono invece quelle relative al rapporto prezzo/qualità dell'esperienza e quelle relative all'isolamento.

1.1 Obbiettivi

L'obbiettivo di questa tesi è realizzare il prototipo di un'applicazione client-server che permetta di utilizzare in remoto applicazioni in VR, con supporto alla multiutenza.

L'applicazione in realtà virtuale dovrà girare sul server, dispositivo con capacità di calcolo notevolmente superiori rispetto a quelle del client. Più utenti dovranno avere la possibilità di connettersi contemporaneamente e condividere lo stesso spazio virtuale.

Il client sarà, in questo caso, un'applicazione Android che si connetterà al server e avrà il compito di mostrare all'utente l'output dell'applicazione VR e allo stesso tempo ricevere l'input da inviare al server.

Un altro obbiettivo durante lo sviluppo del prototipo è quello di realizzare una libreria che offra le funzionalità sopraelencate, facilmente integrabile in nuovi progetti o in progetti già esistenti.

Un requisito importante è quello di mantenere il rendering vicino ai 60 frame per secondo, in modo da offrire un'esperienza il più possibile priva di motion sickness.

Utilizzando questa struttura client-server sarà possibile sviluppare applicazioni che permettano a più persone di condividere lo stesso spazio virtuale, ognuno dal proprio punto di vista, utilizzando visori e sistemi operativi diversi.

Quindi l'oggetto di questa tesi è proporre una soluzione alternativa ai problemi relativi al costo e all'isolamento della VR.

2 Fondamenti

Questo capitolo ha lo scopo di introdurre le nozioni fondamentali necessarie al lettore per capire gli argomenti trattati e il lavoro svolto in questa tesi.

2.1 Virtual Reality

La realtà virtuale viene collocata all'interno del continuo realtà-virtualità, introdotto da Milgram nel 1994. La realtà aumentata è collocata vicino al mondo reale, mentre la realtà virtuale (virtualità aumentata) è più vicina all'ambiente puramente virtuale.
In altre parole, la realtà aumentata aggiunte informazioni virtuali al mondo reale, con lo scopo di aumentare la percezione e l'interazione dell'utente con il mondo reale. La virtualità aumentata, al contrario, potenzia il mondo virtuale utilizzando informazioni provenienti dal mondo reale, ed è quindi più vicina al concetto di ambiente puramente virtuale.

Al giorno d'oggi il termine VR viene comunemente utilizzato per indicare un ambiente a tre dimensioni generato da un computer, che può essere esplorato da una persona. L'utente diventa parte di questo mondo ed è quindi in grado di manipolarlo o compiere una serie di azioni all'interno di esso.

Esiste un vasto range di dispositivi utilizzati per questo scopo: headsets, guanti, esoscheletri ecc. Tutti vengono usati per ingannare i sensi umani in modo da creare l'illusione che l'esperienza virtuale corrisponda alla realtà.
Le tecnologie per la VR devono quindi tener conto della fisiologia umana, se mancano di accuratezza e sincronismo l'esperienza perde credibilità, rischiando anche di far provare malessere fisico all'utente. Ad esempio se le informazioni raccolte dal sistema vestibolare nell'orecchio umano non coincidono con quelle raccolte dall'apparto visivo si sperimenta la così detta "motion sickness".
È quindi necessario che le informazioni raccolte dai sensori siano tradotte istantaneamente in reazioni nel mondo digitale, con una latenza impercettibile al cervello umano.

La realtà virtuale ha già trovato parecchie applicazioni in diversi ambiti come l'architettura, lo sport, la medicina, l'esercito, le arti e l'intrattenimento. Il numero di impieghi è destinato a crescere con la commercializzazione dei dispositivi in grado di permettere l'accesso alla tecnologia.
Ogni volta che qualcosa è troppo pericoloso, costoso o poco pratico per essere fatto nella realtà, la realtà virtuale potrebbe presentarsi come una buona alternativa. Andando dai programmi di addestramento per piloti di aerei da guerra a quelli per chirurghi, la realtà virtuale permette di correre rischi virtuali in cambio di esperienza relativa al mondo reale.

Mano a mano che queste tecnologie migliorano e i loro prezzi calano ci si può aspettare sempre più utilizzi, con focus rivolti anche verso l'educazione e la produttività.
La realtà virtuale e la realtà aumentata hanno le potenzialità per cambiare radicalmente il modo in cui interagiamo con la tecnologia, continuando il trend della sua umanizzazione.

2.1.1 Headsets

I visori attualmente in commercio (o in sviluppo) sono parecchi. Possono essere suddivisi in due categorie in base al prezzo, più o meno elevato.

Tra i visori costosi quelli di riferimento sono Oculus Rift, HTC Vive e Playstation VR. Questi visori rappresentano lo standard qualitativo più alto per la VR, montando sensori ad alta precisione e display ad alta fedeltà.
Tutti questi dispositivi sono caratterizzati dalla necessità di essere collegati ad un computer per funzionare. È quindi disponibile un'elevata potenza di calcolo che permette molte applicazioni interessanti. Questi visori dispongono di metodi di input specifici, quindi ognuno di essi ha le proprie peculiarità nel modo d'interagire con gli ambienti virtuali.
Il problema che accomuna tutti questi dispositivi è appunto il prezzo. Infatti attualmente il più economico costa 400 dollari. Considerando che per funzionare, tutti necessitano di un computer altamente performante, il costo complessivo diventa notevole.

VR Headsets di qualità

I visori più economici invece sono il Google Cardboard e il Gear VR di Samsung.
Questa categoria di visori è caratterizzata dal fatto che non contengono display e sensori, ma solo le lenti e uno spazio apposito in cui inserire lo smartphone. Quindi la grande differenza rispetto ai visori più costosi è che per funzionare questi non si appoggiano ad un computer, ma unicamente ad uno smartphone.
Vista l'elevata diffusione degli smartphone è chiaro che la spesa per accedere alla VR è, in questo caso, unicamente il prezzo del visore, il quale varia tra i 10 e i 99 dollari.

VR Headsets a basso costo

Particolarmente d'interesse per questa tesi è Google Cardboard. Nominata a seguito del materiale di cui è costruito il visore, la piattaforma è intesa come un sistema a basso costo per incoraggiare l'interesse e lo sviluppo di applicazioni in VR.
La piattaforma è stata creata da due ingegneri Google nel loro 20% project. È stata resa pubblica al Google I/O del 2014.
Il software development kit (SDK) è disponibile sia per Android che iOS, inoltre l'SDK permette agli sviluppatori di integrare contenuti VR all'interno del web.
Le modalità di input possibili con Google Cardboard sono limitate, infatti oltre al movimento della testa, è supportato unicamente un generico tocco sullo schermo.

2.1.2 VR in Android

La realtà virtuale si sta diffondendo anche sui dispositivi mobile, soprattutto su Android.
Alla VR è stata dedicata una sezione apposita nel Google Play Store e a partire dalla N-release di Android alcune funzionalità saranno integrate direttamente all'interno del sistema operativo.

Il problema principale della VR su Android è la limitata capacità di calcolo degli smartphone, i quali sono in grado di fornire solo esperienze qualitativamente scadenti e difficili da considerare come qualcosa di più che semplici demo.
Nonostante questa limitazione le potenzialità sono notevoli: VR senza fili, a basso costo e alla portata di tutti.

2.2 Android

Di seguito sono introdotti alcuni costrutti basilari tipici di Android, fondamentali per capire il lavoro svolto lato client in questa tesi. La spiegazione non vuole essere completamente esaustiva ma solo dare un'idea.

2.2.1 Activity

È il componente alla base di molte applicazioni Android, fornisce una schermata con cui gli utenti possono interagire. Ogni Activity è associata ad una finestra, all'interno della quale disegna la propria interfaccia utente. Solitamente la finestra riempie lo schermo, ma potrebbe anche essere più piccola e risiedere al di sopra di altre finestre.
In genere un'applicazione è formata da più Activity, debolmente collegate tra loro.
Tipicamente, una delle Activity dell'applicazione viene definita principale, questa verrà presentata all'utente all'apertura dell'applicazione.
Ogni Activity può poi lanciare altre Activity per compiere diverse azioni. Ogni volta che una Activity viene lanciata, l'Activity precedente viene fermata e preservata nel back stack di sistema.
Ogni Activity espone una serie di metodi callback, i quali vengono chiamati dal sistema in momenti significativi del ciclo di vita dell'Activity [4].

2.2.2 Fragment

Un Fragment rappresenta un comportamento o una porzione di interfaccia utente in un Activity.
È possibile riutilizzare uno stesso Fragment in più Activity e combinare diversi Fragment in una singola Activity per costruire un User Interface (UI) multi-pannello. Si può pensare ad un Fragment come una sezione modulare di un Activity, con il proprio lifecycle [5], che riceve il proprio input e che può essere aggiunto o rimosso mentre l'Activity è in vita.
Un Fragment deve sempre essere parte di una Activity e il suo lifecycle è in relazione diretta con quello dell'Activity che lo ospita. Ad esempio, quando una Activity è messa in pausa o viene distrutta, lo stesso accade a tutti i suoi Fragment.

2.2.3 Custom View

Il framework Android fornisce una grande varietà di View [6], le quali coprono quasi interamente i bisogni di una tipica applicazione Android.
Queste View, assieme alla moltitudine di opzioni per utilizzare risorse di tipo drawable [7], rendono possibile la realizzazione di interfacce utente complesse.
Il problema di questi componenti è che, dovendo essere usati per una moltitudine di scopi, hanno spesso un set di funzionalità più vasto di quanto venga effettivamente richiesto dai singoli utilizzatori.
Spesso quindi vengono create custom View per ottenere funzionalità simili a quelle offerte dal framework, ma in maniera più performante.
Altre volte invece le View fornite dal framework non offrono le funzionalità richieste, ed è quindi necessario implementarle.

2.3 Sensori in Android

La maggior parte dei dispositivi Android ha a disposizione sensori che misurano il movimento, l'orientazione e varie condizioni ambientali. Questi sensori sono in grado di fornite raw data con alta precisione e accuratezza, e sono utili nel caso si vogliano monitorare movimenti nello spazio o cambiamenti nello spazio che circonda il device.

Le tre grandi categorie di sensori supportati da Android sono [8]:

- sensori di movimento:
  misurano le forze di accelerazione e rotazionali lungo i tre assi. Questa categoria include accelerometri, sensori di gravità, giroscopi e sensori del vettore di rotazione.
- sensori ambientali:
  misurano diversi parametri ambientali, come la temperatura e la pressione dell'ambiente, illuminazione e umidità. Questa categoria include barometri, fotometri e termometri.
- sensori di posizione:
  misurano la posizione fisica del device. Questa categoria include sensori d'orientazione e magnetometri.

Alcuni di questi sensori sono hardware based, mentre altri sono software based.
I sensori hardware based sono componenti fisici presenti all'interno del device. Essi derivano i loro dati da misurazioni dirette.
I sensori software based non sono presenti fisicamente all'interno del device, ma imitano i sensori hardware based. Questi sensori derivano i loro dati da uno o più sensori hardware, e sono spesso chiamati sensori virtuali o "fused sensors" [9] se uniscono i dati provenienti da più sensori.

2.3.1 I giroscopi in Android

Android dispone di due tipi di giroscopio. Quello calibrato e quello non calibrato.
Il giroscopio è sempre un sensore hardware based e viene spesso usato come base per creare altri sensori virtuali.

2.3.1.1 Drift del giroscopio

Come tutti i sensori il giroscopio non è perfetto, ogni misura contiene piccoli errori. Dato che le misurazioni del giroscopio sono integrate nel tempo, questi piccoli errori vengono sommati tra loro, risultando in quello che viene comunemente chiamato drift del giroscopio. Nel tempo quindi i risultati delle misurazioni possono perdere attendibilità e alcune forme di compensazione sono necessarie per risolvere il problema del drift.
La tecnica comunemente adottata per compensare il drift consiste nell'utilizzo di un secondo sensore, che fornisce un'altra misura dell'orientazione del device, la quale viene utilizzata per aggiustare l'integrazione dei dati del giroscopio. Questo secondo sensore è in genere un accelerometro o un magnetometro.

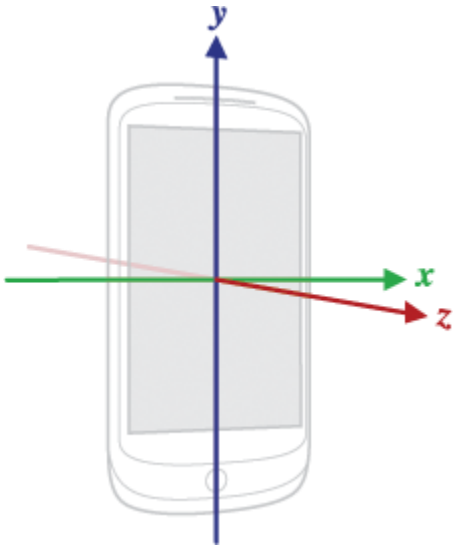
2.3.1.2 Calibrated VS Uncalibrated

A partire da Android 4.3 è stato introdotto il giroscopio non calibrato (Uncalibrated Gyroscope).

La principale differenza tra calibrated e uncalibrated è che utilizzando il giroscopio non calibrato non si gode di alcuna compensazione del drift. Questo dà la possibilità di effettuare la fusione tra diversi sensori, senza doversi preoccupare di fusioni e compensazioni effettuate da Android per compensare il drift.
Utilizzando il giroscopio calibrato invece, Android applica automaticamente delle tecniche di compensazione del drift.

2.3.1.3 Sistema di coordinate

Il giroscopio usa il sistema di coordinate locali al dispositivo.
Per la maggior parte dei dispositivi il sistema di coordinate è definito in relazione allo schermo quando il dispositivo è nella sua orientazione di default (vedi figura).



Sistema di coordinate in Android

Quando il device è in questa orientazione l'asse X è orizzontale e punta verso destra, l'asse Y è verticale e punta verso l'alto e l'asse Z è uscente dello schermo. In questo sistema, coordinate dietro lo schermo hanno Z con valori negativi. È da notare che gli assi non si scambiano quando l'orientazione del device cambia.

2.3.1.4 Integrazione nel tempo

Il metodo di integrazione dei dati del giroscopio è fornito direttamente dalla documentazione di Android e non è stato oggetto di studio di questa tesi. Di seguito viene riportato schematicamente il processo:

Il giroscopio fornisce il cambiamento della posizione angolare nel tempo (velocità angolare) in rad/s. Questo dato corrisponde alla derivata della posizione angolare nel tempo.

$$\dot{\theta} = \frac{d\theta}{dt}$$

Per ottenere la posizione angolare possiamo semplicemente integrare la velocità angolare.
Quindi assumendo che per t=0, theta=0, possiamo trovare la posizione angolare ad ogni momento t con la seguente equazione:

$$\theta(t) = \int_0^t \dot{\theta}(t)dt \approx \sum_0^t \dot{\theta}(t)T_s$$

La rappresentazione finale dell'equazione è l'approssimazione necessaria per l'utilizzo all'interno di sistemi digitali. Dato che non possiamo calcolare perfettamente un integrale continuo, dobbiamo considerare la somma di un finito numero di campioni, presi ad un intervallo costante Ts, detto periodo di campionamento. Questa approssimazione introduce necessariamente degli errori. Se i dati del giroscopio cambiano più velocemente della frequenza di campionamento, l'approssimazione dell'integrale sarà incorretta. Questo è ciò che dà origine al drift.

2.4 Unity

Unity [10] è un ambiente di sviluppo creato appositamente per lo sviluppo di applicazioni 2D e 3D per PC, Mac, Android, iOS, Windows Phone e le principali console in commercio.
Unity è innanzitutto un potente strumento tramite il quale testare e costruire applicazioni, offrendo completo supporto a tutti gli aspetti fondamentali per la realizzazione di applicazioni 2D o 3D: rendering grafico, effetti di luce, creazione di terreni, simulazioni fisiche, implementazione dell'audio, funzionalità di rete, supporto al multiplayer ed anche un sistema di scripting che permette agli sviluppatori di creare componenti custom.
Oltre ad offrire un engine efficiente, viene anche offerta una molteplicità di tools adatti a realizzare specifici task: ad esempio un profiler per monitorare le prestazioni delle applicazioni, un tool per semplificare la generazione di paesaggi ecc. Lo sviluppo è agevolato da alcune features, come la cosiddetta “live preview”, ovvero la preview in tempo reale di quello che si sta creando.
Il tutto è accompagnato da un ottimo sistema di gestione delle risorse utilizzate dall’applicazione.

Con Unity 3D si possono realizzare videogiochi o altri contenuti interattivi, quali visualizzazioni architettoniche e ambientazioni tridimensionali. I linguaggi supportati del sistema di scripting sono due: Javascript e C#.

Il software è disponibile in due versioni: una gratuita e una a pagamento.
Una caratteristica importante è l’asset store [11], un negozio digitale dove è possibile acquistare o scaricare personaggi, oggetti, ambienti e molti altri elementi sviluppati e messi a disposizione dalla community.

Tutte queste caratteristiche semplificano notevolmente il lavoro necessario per sviluppare ambienti virtuali interattivi e il loro rendering in real-time.

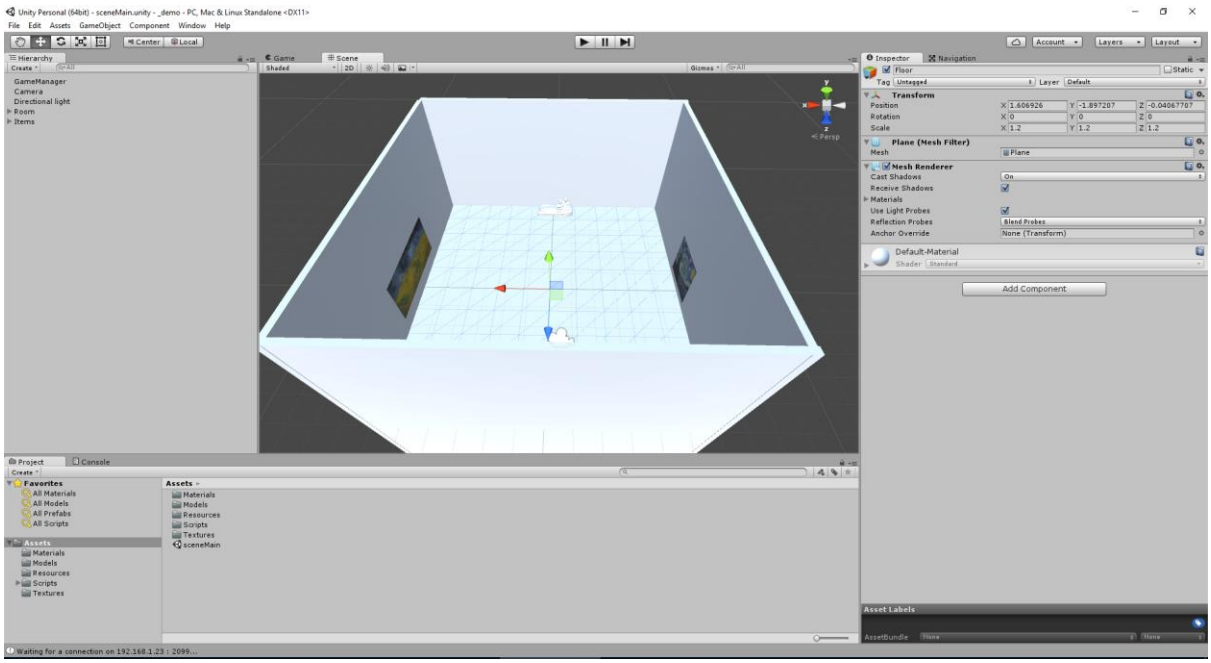
Di seguito vengono introdotti alcuni concetti base fondamentali per capire alcuni aspetti del lavoro svolto.

2.4.1 Editor

L’editor di Unity si suddivide principalmente in sei sezioni:

- **Scene:** dove viene mostrato lo scheletro dell’ambientazione e degli oggetti. All’interno di questa sezione è fornita la possibilità di posizionare tutti i componenti della scena, di scalarli e ruotarli. È possibile inoltre cambiare vista tra quelle predefinite, come vista dall’alto, dal basso, laterale; ruotare, fare lo zoom o spostarsi semplicemente utilizzando il mouse.
- **Game:** in questa sezione è mostrato il render della scena percepito dalla telecamera impostata come principale. Cliccando il tasto play l’applicazione viene eseguita direttamente all’interno dell’editor, il che permette un controllo in tempo reale di quello che sarà l’aspetto e il comportamento finale dell’applicazione.
- **Hierarchy:** in questa sezione sono elencati tutti gli elementi della scena, suddivisi in ordine di parentela tramite una rappresentazione a directory. Ogni componente può avere al suo interno altri componenti figli, i quali sono più o meno vincolati al padre. Grazie al pulsante Create si possono aggiungere nuovi elementi alla scena.

- **Project:** qui è possibile esplorare tutti i file e cartelle che fanno parte del progetto. Si può scegliere di salvare scene diverse e caricarle per la modifica o visualizzazione, facendo doppio click o trascinando il file nella sezione Hierarchy. Tramite il tasto Create possiamo aggiungere cartelle, scene, script, materiali, prefab, ecc. I prefab [12] sono strutture contenenti al loro interno oggetti semplici o complessi, e offrono la possibilità di essere utilizzati come stampi per l'inserimento di oggetti fotocopia (aventi lo stesso comportamento e aspetto).
- **Inspector:** all'interno di questa sezione troviamo tutti i dati relativi alle caratteristiche dell'oggetto della scena correntemente selezionato. La visualizzazione cambia a seconda della selezione mostrando tutti i componenti e le proprietà dell'oggetto.
- **Console:** in fase di testing riporta tutti gli errori, i warning e i log stampati dall'applicazione.



Unity Editor

Ogni oggetto che fa parte del progetto è elencato all'interno del riquadro Project ed ogni oggetto che fa parte della scena si può trovare all'interno di Hierarchy.
Gli oggetti all'interno di Hierarchy vengono chiamati GameObject (GO) e ognuno di questi può essere salvato come componente Prefab, con tutti i suoi figli, e riutilizzato in tutte le scene come oggetto identicamente uguale al GO che lo ha originato.
Questa funzione risulta utile nel caso servano istanze multiple dello stesso GO all'interno di una o più scene.
Quando viene modificato il GO originario, le variazioni si propagano a tutte le sue istanze nel programma.
Ogni GO possiede al suo interno dei Component che ne definiscono caratteristiche e comportamento.

2.4.2 Script

Ad ogni GO è possibile collegare uno o più script, i quali a loro volta, possono essere assegnati a più GO. Attraverso il bottone Create, nella sezione Project, è possibile creare un

file con estensione .cs (nel caso di C#), che può essere trascinato su ogni object all'interno di Hierarchy, per un assegnamento veloce. Ogni script viene editato attraverso l'editor MonoDevelop [13] o, nelle versioni più recenti, tramite VisualStudio [14].
MonoBehaviour [15] è la classe da cui ogni GO deriva. Alla creazione della classe, in automatico vengono ereditate le funzioni callback di MonoBehaviour, utili per eseguire codice in momenti specifici della vita dell'applicazione.
Le funzioni più rilevanti, partendo dalle due principali, sono:

- **Start():** È lanciata all'esecuzione del programma e chiamata una sola volta. È utile principalmente per inizializzare variabili o per compiere altre operazioni di set up.
- **Update():** Questa funzione viene chiamata regolarmente all'update di ogni frame.
- **OnApplicationQuit():** Viene chiamata in fase di chiusura. La sua funzione principale è sfruttata per il reset, per la deallocazione delle strutture dati e per la chiusura di connessioni aperte.

Ovviamente è anche possibile creare classi normali (senza ereditare da MonoBehaviour) in modo da poter distribuire le responsabilità nel migliore dei modi.
Sono disponibili anche buona parte delle librerie del framework .NET [16].

2.5 Cloud Gaming

Il cloud gaming è una tipologia di gaming online. Attualmente ci sono due principali categorie di cloud gaming: una basta su video streaming e l'altra su file streaming.
L'obbiettivo del cloud gaming è quello di fornire all'utente la possibilità di giocare istantaneamente attraverso varie piattaforme, in maniera totalmente indipendente dall'hardware.

La prima dimostrazione di cloud gaming risale all'E3 del 2000. Dove G-cluster dimostrò la fattibilità del cloud gaming su rete Wi-Fi. Successivamente Crytek [17] iniziò a lavorare su un sistema basato sul cloud gaming per Crysis [18], nel 2005, ma poi abbandonò il progetto nel 2007, nell'attesa che le infrastrutture e gli internet provider fossero all'altezza del compito.
La prima vera piattaforma basata sul cloud gaming è stata OnLive [19], lanciata nel 2010. Seguita da Gaikai [20], che permetteva di giocare a giochi AAA direttamente dal browser, con anche la possibilità di integrare il contenuto all'interno di Facebook o sulle IPTV.
Ad oggi sia Gaikai che OnLive sono state acquistate da Sony e integrate all'interno del progetto Playstation Now [21].

La prima categoria, basta sullo streaming video è quella più interessante. Chiamata anche "gaming on demand", è una tipologia di online gaming che permette streaming diretto e on-demand di giochi su computer, console e dispositivi mobili, tramite l'utilizzo di un leggero client.
Il gioco è salvato, eseguito e renderizzato sul server remoto e l'output video è mandato in streaming direttamente al client installato sulla macchina dell'utente. Questa tecnica permette l'accesso ad ogni gioco, senza la necessità di possedere un computer performante, infatti la potenza del computer client è quasi trascurabile, dato che è il server ad occuparsi delle operazioni pesanti.
I comandi impartiti al gioco da parte dell'utente sono trasmessi direttamente al server, il quale li trasmette al gioco.
Il gaming on demand è un servizio di gaming che si appoggia sulle capacità delle connessioni a banda larga, grandi sistemi di server e compressione dei dati per poter distribuire i contenuti al client. Gli utenti possono giocare ad ogni gioco senza doverlo scaricare e installare. Il contenuto del gioco non è memorizzato sull'hard drive dell'utente e l'esecuzione del codice avviene sul server, in modo che l'utente possa accedere al gioco utilizzando un computer molto meno potente di quanto in realtà sarebbe necessario.

La seconda categoria, basta sul file streaming, conosciuta anche come "progressive downloading", utilizza un client dentro il quale il gioco effettivo viene fatto girare sul device dell'utente. Una piccola parte del gioco, di solito meno del 5%, viene scaricata sul computer dell'utente, in modo che esso possa partire molto prima del download completo. Il resto viene scaricato sul dispositivo dell'utente mentre egli gioca. Questo permette accesso quasi istantaneo al contenuto, anche attraverso reti lente. La componente cloud è utilizzata per fornire una modalità scalabile di streaming dei dati d'installazione. Questa tipologia di cloud gaming, basata sul file streaming, richiede che l'utente abbia un computer sufficientemente potente per far giocare il gioco scaricato, è quindi più vicina al gaming tradizionale, dove il gioco gira sulla macchina dell'utente.

2.5.1 Protocolli

Nella realizzazione della componente cloud di questa tesi sono stati utilizzati due protocolli: TCP e UDP.

- **TCP:** il transmission control protocol (TCP) è un protocollo di rete a pacchetto del livello di trasporto. All'interno dell'implementazione dello stack di protocolli ISO OSI esso si basa sul protocollo IP. Di conseguenza ci si riferisce alla suite nel suo insieme come TCP/IP.
  TCP fornisce uno stream affidabile, ordinato e con controllo d'errore per la consegna di dati tra applicazioni che girano su degli host che comunicano attraverso una rete IP.
  Le maggiori applicazioni Internet come il World Wide Web, le email, l'amministrazione remota e il trasferimento file fanno uso di TCP.
  TCP fornisce un servizio per la comunicazione ad un livello intermedio, tra l'applicazione e il protocollo internet. Fornisce connessioni host-to-host al livello di trasporto.
  Un'applicazione non deve conoscere il particolare meccanismo necessario per inviare dati, come la dimensione di frammentazione dei packet IP o il mezzo trasmissivo. Al livello di trasporto il protocollo gestisce tutti i dettagli di handshaking e trasmissione, presentando un'astrazione della connessione all'applicazione.
  Ai livelli più bassi dello stack, per via della congestione della rete e altri comportamenti imprevedibili, alcuni packet IP possono essere persi, duplicati o consegnati non in ordine. TCP rileva questi problemi, richiede la ritrasmissione dei dati pesi, riordina quelli in disordine e aiuta a minimizzare i problemi relativi alla congestione della rete. Se i dati continuano a non essere consegnati il mittente viene notificato del fallimento.
  TCP è ottimizzato per garantire una consegna accurata piuttosto che una consegna veloce. Quindi a volte comporta lunghi ritardi mentre risolve errori dovuti alla trasmissione.
  Questo lo rende poco adatto per applicazioni in real-time, per questo tipo di applicazioni sono più adatti altri protocolli, come il real-time transport protocol (RTP) il quale sfrutta UDP.

- **UDP:** User Datagram Protocol (UDP) fa uso di una semplice trasmissione senza connessione. Non prevede meccanismi di handshake tra gli host e quindi espone l'applicazione finale a tutte le problematiche dovute all'inaffidabilità della rete. Non c'è garanzia di consegna, ordinamento o riconoscimento dei duplicati, UDP fornisce solo meccanismi di checksum per garantire l'integrità dei dati.
  Tramite UDP, le applicazioni possono inviare messaggi (datagrammi) ad altri host attraverso reti IP.
  UDP è adatto per scopi che non necessitano di error-checking, dove la correzione non è necessaria o è fornita dall'applicazione stessa. Applicazioni time-sensitive spesso usano UDP, perché, in questi casi, perdere datagrammi è preferibile rispetto all'attesa di pacchetti ritrasmessi, che potrebbe comportare attese non accettabili.

- **RTP:** Real-time Transport Protocol (RTP) è un protocollo di rete utilizzato per trasmettere audio e video tramite reti IP.

RTP è usato estensivamente in sistemi per la comunicazione e l'intrattenimento che comportano lo streaming, come la telefonia, applicazioni per video conferenze, servizi televisivi ecc.
RTP è usato assieme al RTP Control Protocol (RTCP). RTP si occupa dello stream dei dati, mentre RTCP è utilizzato per monitorare le statistiche della trasmissione e la qualità del servizio (QoS).
RTP è pensato per consentire streaming end-to-end in real-time. Il protocollo fornisce compensazione del jitter e riconoscimento della ricezione di dati fuori sequenza, fenomeni comuni in trasmissioni attraverso reti IP.
RTP è considerato come lo standard per il trasporto di audio/video attraverso reti IP.
Le applicazioni per lo streaming in real-time richiedono consegna veloce dell'informazione e a questo scopo, sono accettate perdite di pacchetti. Ad esempio una perdita di un pacchetto all'interno di un'applicazione per lo streaming audio, risulta nella perdita di una frazione di secondo nella riproduzione, che grazie ad opportuni algoritmi di compensazione può passare inosservata all'utilizzatore.
Anche se TCP è standardizzato per essere utilizzato per RTP, non è di solito usato in applicazioni RTP, perché esso favorisce l'affidabilità alla velocità. La maggior parte di implementazioni di RTP sono costruite su UDP.

3 Lavoro svolto

L'applicazione è divisa in due componenti: un server e un client.
Implementando il modello di cloud gaming basato sullo streaming video, è chiaro che il client è leggero ed è l'unico punto d'interazione per l'utente verso l'applicazione 3D.
Il server invece ha il compito di far girare l'applicazione e gestirne interamente il rendering e le sessioni multiple. L'output dell'applicazione 3D viene mandato in streaming al client, il quale allo stesso modo manda l'input per l'applicazione 3D al server.

Il client è scritto in Java ed è stato realizzato in modo da essere abbastanza modulare e indipendente dalla piattaforma, in questo modo è relativamente semplice prendere lo stesso codice ed utilizzarlo in un altro contesto/sistema operativo, semplicemente sostituendo alcuni package e alcune classi.
Ad esempio, senza particolare sforzo, si potrebbe fare un fork e creare un client che funzioni su un PC con un altro visore, come un Oculus Rift o un HTC Vive.

Il server è stato scritto in C# ed esiste sotto forma di insieme di scripts e assets da utilizzare all'interno di un progetto Unity. Il server è stato scritto puntando alla semplicità d'integrazione in progetti Unity, nuovi o già esistenti.

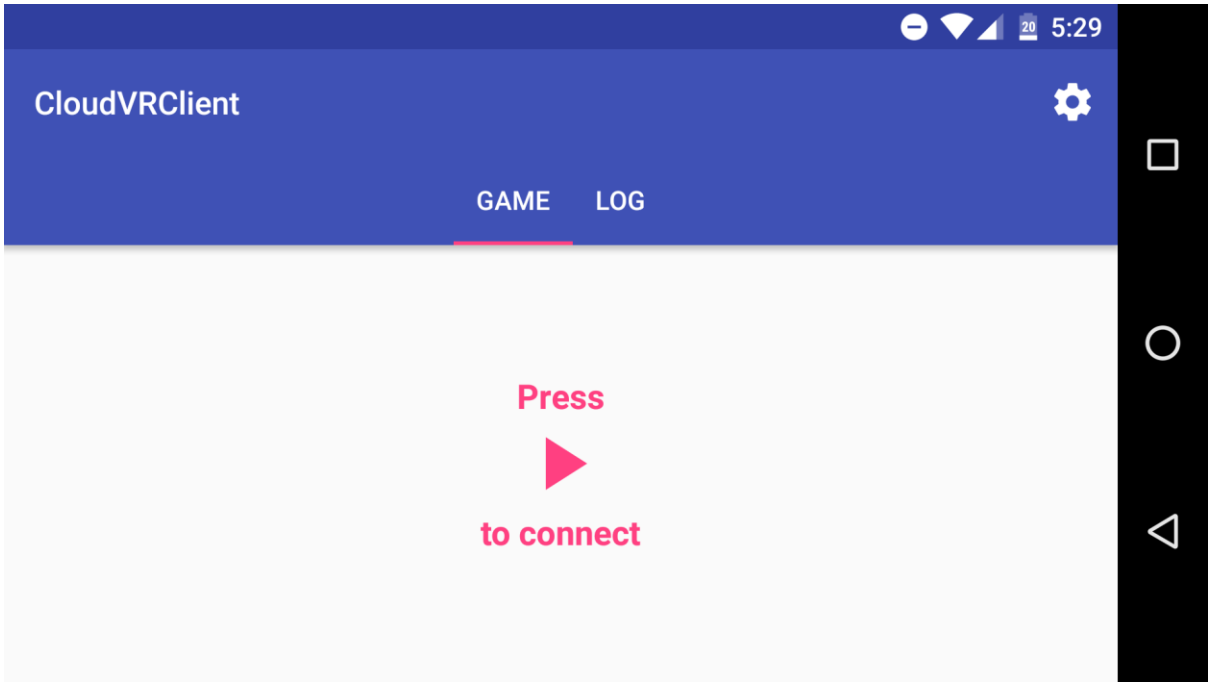
Si è scelto di utilizzare Unity come game engine per semplificare e velocizzare la realizzazione degli ambienti tridimensionali, che non era direttamente oggetto della tesi.

Per la comunicazione tra client e server sono stati implementati due protocolli, uno basato su TCP e uno su UDP, i quali possono essere intercambiati tra loro.

Attualmente l'applicazione è pensata per funzionare all'interno di una rete LAN. Potenzialmente può funzionare anche al di fuori di una LAN, ma per ottenere un'esperienza accettabile sarebbe necessario implementare meccanismi di compressione e utilizzare protocolli più adatti allo streaming in real-time.

3.1 Client Android

Il client Android è stato costruito in modo da essere il più leggero possibile.
Per quanto riguarda l'interazione con l'utente deve svolgere due compiti fondamentali: mostrare l'output dell'applicazione 3D e ricevere l'input dato dall'utente per l'applicazione 3D.
Dietro le quinte ovviamente deve essere in grado di comunicare con il server tramite la rete.



Interfaccia client

3.1.1 Struttura

Il client è costruito come un'applicazione con una singola Activity di base, dentro la quale risiede un ViewPager [22] che ospita due Fragment.

Le classi del client sono raggruppate in packages in base alla loro funzione.
Segue una breve descrizione di alto livello dei packages e del loro contenuto:

- **activities:** contiene le Activity usate all'interno dell'applicazione. In particolare *MainActivity* e *PreferencesActivity*.
  La prima ha il solo compito di contenere due Fragment (all'interno di un ViewPager) e il menù, mentre la seconda contiene le impostazioni dell'applicazione.
- **fragments:** contiene le classi *BaseFragment*, *GameFragment* e *LogFragment*.
  BaseFragment è una classe astratta che implementa le funzionalità comuni a tutti i Fragment dell'applicazione (binding a ButterKnife e sincronizzazione tra EventBus e lifecycle del Fragment).
  GameFragment è la View principale di tutto il client, conviene la View responsabile della visualizzazione delle immagini trasmesse dal server e della ricezione dell'input, inoltre è il punto di avvio e interruzione delle comunicazioni con il server.
  LogFragment ha la responsabilità di mostrare all'utente i vari log lanciati all'interno del client.

- **adapters:** contiene la classe *ViewPagerAdapter*, una semplice implementazione di FragmentPagerAdapter, necessaria per mostrare e interagire con i Fragment dell'applicazione.
- **headtracking:** questo package è suddiviso in due sotto package: *providers* e *representation*.
  Le classi contenute in *providers* hanno il compito di interfacciarsi con i sensori del sistema per ottenere i dati necessari al tracciamento dei movimenti dell'utente.
  Le classi in *representation* sono utilizzate per ottenere una rappresentazione ad alto livello dei dati ottenuti dai sensori.
  La classe *OrientationProvider* è una classe astratta e definisce una interfaccia verso un generico sensore. Estendendola è estremamente facile fornire interfacce verso sensori differenti. La classe *CalibratedGyroscopeProvider* è appunto un'implementazione di questa classe e permette di interfacciarsi con il giroscopio calibrato del dispositivo.
  I dati raccolti da queste classi vengono visti all'esterno sotto forma di quaternioni.
- **io:** questo package è diviso in due sotto packages: *connections* e *data*.
  Il package *connections* contiene l'interfaccia *ServerIO* che definisce il contratto per la comunicazione con il server. Quest'interfaccia ha due implementazioni, una sfrutta TCP e l'altra UDP.
  Il package *data* contiene le classi rappresentati i dati da trasmettere al server, quindi l'input dell'applicazione 3D.
- **logging:** contiene le classi che implementano la funzionalità di logging all'interno dell'applicazione.
- **utils:** contiene classi di utilità per la gestione dei Fragment, per la gesture detection ecc.

3.2 RemoteVRView

RemoteVRView è una custom View realizzata appositamente per questa applicazione.
La classe ha il compito di mostrare le immagini che le vengono passate e di ricevere l'input dell'utente.

3.2.1 Output

L'output del client (verso l'utente) è costituito dalle immagini dell'applicazione 3D inviate dal server.
Le immagini vengono ricevute sotto forma di array di byte, questo array di byte viene decodificato per creare oggetti Bitmap.
Ogni volta che un Bitmap è pronto viene passato a RemoteVRView che si occupa di renderizzarlo sullo schermo, avendo cura di ridimensionare l'immagine in modo che non subisca deformazioni.

```
private double drawBitmap(Bitmap bitmap, Canvas canvas) {
   double viewWidth = canvas.getWidth();
   double viewHeight = canvas.getHeight();
   double imageWidth = bitmap.getWidth();
   double imageHeight = bitmap.getHeight();
   double scale = Math.min(viewWidth / imageWidth, viewHeight / imageHeight);
```

```
    Rect destBounds = new Rect(0, 0, (int) (imageWidth*scale), (int) (imageHeight*scale));
    canvas.drawBitmap(bitmap, null, destBounds, null);
    return scale;
}
```

3.2.2 Input

L'applicazione è pensata per funzionare con Google Cardboard quindi le tipologie di input supportate sono due: giroscopio e touch sullo schermo.
RemoteVRView, ovviamente, non si occupa della gestione del giroscopio, ma è sensibile al touch dello schermo. Dato che quando la comunicazione con il server è in corso essa è l'unica View visibile sullo schermo.
Le gesture riconosciute dalla View sono: tocco, rilascio e swipe dall'alto verso il basso.
Tocco e rilascio sono i comandi da impartire all'applicazione 3D, mentre lo swipe dall'alto verso il basso viene intercettato da GameFragment, che lo gestisce terminando la connessione con il server.

```
public boolean onTouchEvent(MotionEvent event) {
    publishSubject.onNext(event);

    swipeDetector.onTouchEvent(event);

    // return true for ACTION_DOWN, so we can detect ACTION_UP
    if(event.getAction() == MotionEvent.ACTION_DOWN)
        return true;

    return super.onTouchEvent(event);
}
```

La View quindi si occupa solo di rilevare questi eventi e rilanciarli verso l'esterno. Non contiene alcuna logica per la gestione.

3.2.3 Librerie utilizzate

- **RxJava [23]:** RxJava è una implementazione di ReactiveX (Reactive Extensions) per la Java Virtual Machine.
  ReactiveX è una libreria per comporre programmi asincroni ed event-based tramite l'uso di sequenze "osservabili".
  Estende il pattern observer al supporto di sequenze di dati e/o eventi e aggiunge operatori che permettono di comporre sequenze diverse mantenendo un'astrazione da concetti come: threading di basso livello, sincronizzazione, thread safety, strutture dati concorrenti e I/O non bloccante.
  Il modello ad Observable di ReactiveX permette di trattare stream di eventi asincroni con lo stesso tipo di semplici operazioni con cui vengono trattate normali collezioni di dati, come gli array. Questo modello libera lo sviluppatore dal rischio del cosiddetto "callback hell", di conseguenza rende il codice più leggibile e meno soggetto a bug.
  Gli Observable di ReactiveX supportano non solo l'emissione di singoli valori, ma anche sequenze di valori o addirittura stream infiniti. Observable è un'astrazione che può essere utilizzata per ognuno di questi casi d'uso.

ReactiveX offre un insieme di operatori con cui si può filtrare, selezionare, trasformare, combinare e comporre Observables. Questo permette efficienza nell'esecuzione e composizione.
Si può pensare alla classe Observable come ad un equivalente di tipo "push" della classe Iterable, che è "pull". Con un Iterable, l'utilizzatore prende (pull) i valori dal produttore e il thread blocca finché i valori non arrivano. In contrasto, con un Observable il produttore pubblica (push) i valori al consumatore ogni volta che sono disponibili. Questo approccio è più flessibile, perché i valori possono arrivare in modo sincrono o asincrono.

RxJava è alla base del funzionamento della logica di rete dell'applicazione client.
Ad esempio questo metodo della classe *ServerTCP* ritorna un Observable<Bitmap> il quale pubblica un Bitmap ogni volta che la socket connessa al server riceve un'immagine.

```
public Observable<Bitmap> getServerOutput() {
   Observable.OnSubscribe<Bitmap> onSubscribe = subscriber -> {
      try {
         // receive images
         int dim;
         while ((dim = inSocket.readInt()) > 0) {
            byte[] img = new byte[dim];
            inSocket.readFully(img, 0, dim);
            // notify the subscribers that a new image is ready
            subscriber.onNext(BitmapFactory.decodeByteArray(img, 0, img.length));
         }
      } catch (Exception e) {
       LoggerBus.getInstance().post(new LoggerBus.Log("Exception: " + e.getClass(),
LOG_TAG, LoggerBus.Log.ERROR));
        subscriber.onError(e);
      } finally {
         subscriber.onCompleted();
         disconnect();
      }
   };

   return Observable.create(onSubscribe);
}
```

Questo Observable viene utilizzato dal GameFragment, tramite un'interfaccia estremamente leggibile e molto semplice da gestire grazie ad un approccio funzionale.

```
serverConnection.getServerOutput()

   // performance monitor
   .doOnSubscribe(mPerformanceMonitor::start)
   .doOnNext(bitmap -> mPerformanceMonitor.newFrameReceived())
   .doOnUnsubscribe(mPerformanceMonitor::stop)

   .subscribeOn(Schedulers.io())
   .doOnSubscribe(() -> EventBus.getInstance().post(new Events.ServerConnected()))
```

```
   .doOnSubscribe(() -> LoggerBus.getInstance().post(new LoggerBus.Log("Started
connection with server.", LOG_TAG)))

   .observeOn(AndroidSchedulers.mainThread())
   .doOnError((error) -> SnackbarFactory.snackbarRequest(getView(),
R.string.error_receiving_images, -1, Snackbar.LENGTH_LONG))
   .subscribe(bitmap -> remoteVRView.updateImage(bitmap),
Throwable::printStackTrace);
```

- **Otto [24]:** questa libreria implementa il concetto di EventBus.
  Quindi permette comunicazione in stile publish-subscribe tra componenti senza richiedere che i componenti si registrino tra di loro esplicitamente (e quindi senza che si conoscano).
  La libreria è utilizzata principalmente per implementare la funzionalità di logging internamente all'applicazione.

- **ButterKnife [25]:** è una libreria estremamente popolare nel mondo Android e ha lo scopo principale di semplificare alcune operazioni di binding delle View, riducendo le righe di boilerplate che gli sviluppatori devono scrivere.

3.3 Struttura Unity package

Il pacchetto Unity è composto da un insieme di scripts, prefabs e risorse. È possibile importare il pacchetto in qualsiasi progetto, nuovo o già esistente ed usarlo semplicemente aggiungendo uno script base ad un qualsiasi oggetto della scena.
Il package è suddiviso in cartelle.
Segue una breve illustrazione delle cartelle e del loro contenuto.

3.3.1 Resources

All'interno di questa cartella sono contenuti:

- Un modello 3D che viene assegnato agli utenti quando si connettono, in modo che ognuno abbia una presenza fisica all'interno della scena.
- Un prefab, *VRCamera*, composto da un oggetto a cui sono associati gli script *PlayerController* e *VRCamera* (vedi dopo). Questo oggetto ha due figli: la camera per l'occhio di sinistra e quella per l'occhio di destra.
- Un altro prefab, *VRCharacter*, che è formato esattamente come *VRCamera* ma in più ha associato il modello 3D che rappresenta l'utente.

Quando un client si connette è possibile istanziare uno qualsiasi tra i due prefab, la differenza è che istanziando *VRCamera* l'utente non è visibile dagli altri, dato che non è associato ad alcun modello 3D. Invece istanziando *VRCharacter* l'utente è visibile.

3.3.2 Scripts

Gli script sono divisi in due moduli: *CloudVRScripts* e *MultiThreading*.

MultiThreading contiene delle classi di utility, utilizzate per passare eventi da un worker thread al main thread. L'utilizzo di questo modulo è necessario perché Unity consente di interagire con gli oggetti di sistema (che estendono MonoBehaviour) solo dal main thread. Quindi, vista la tipologia di applicazione, è fondamentale svolgere il lavoro di IO in dei worker thread, limitandosi a notificare il main thread per certi eventi significativi.

Il modulo *CloudVRScripts* invece, contiene l'effettiva logica dell'applicazione.
Segue una breve descrizione di alto livello del contenuto del modulo:

- La classe base del sistema, presente nella root directory, è *CloudVR*. Per utilizzare le funzionalità di cloud gaming è sufficiente aggiungere questa classe ad un qualsiasi oggetto della scena, preferibilmente un oggetto vuoto apposito, per mantenere ordine nel progetto.
  Questa classe ha la responsabilità di creare il server (TCP o UDP a scelta) e mantenere la lista dei *Player* collegati all'applicazione.
- Nella directory *IO* sono contenute tutte le classi che implementano le funzioni di IO con il client.
  Due interfacce *IServer* e *IClient*. *IServer* è una classe astratta e rappresenta la base comune alle due classi *ServerTCP* e *ServerUPD*. *IClient* invece è un'interfaccia e definisce il contratto che ogni classe deve soddisfare per comunicare con il client.
  Sono presenti inoltre delle classi che definiscono le tipologie di input accettate, in modo speculare a quanto definito lato client.

- Nella directory *Game* sono contenute le classi che modellano elementi della scena.
  La classe *Player* è l'elemento base e viene istanziata ogni volta che un client si connette. Questa classe aggiunge alla scena un'istanza dell'oggetto prefab rappresentante il player (*VRCamera* o *VRCharacter*).
  Le classi *RemoteInputManager* e *RemoteOutputManager* si occupano rispettivamente di ricevere l'input dell'applicazione 3D (comandi) dal client e inviare l'output dell'applicazione 3D (immagini) al client.
  In questo modulo queste classi sono le uniche che fanno uso di classi del package di *IO*.
  La classe *PlayerController* usa *RemoteInputManager* per aggiornare la posizione e la rotazione del player all'interno della scena 3D.
  La classe *VRCamera* è uno script che crea una camera appositamente realizzata per ottenere l'effetto stereoscopico necessario per la VR.

3.3.3 VRCamera

VRCamera è un'implementazione custom di una camera stereoscopica. Non è stata utilizzata una camera preesistente per eliminare ogni tipo di limitazione imposta da progetti esterni e per avere una classe semplice e leggera.
VRCamera estende MonoBehaviour, e deve essere assegnata ad un oggetto che possiede due figli di tipo camera, queste saranno usate per renderizzare l'immagine per l'occhio di sinistra e quello di destra. L'output delle due camere viene poi affiancato in modo da ottenere una visione stereoscopia della scena.
Tramite appositi campi interni è possibile settare la risoluzione della camera, in modo da poterla adattare alla dimensione del display del client e alla capacità della rete.
La classe inoltre espone un metodo GetImage() che restituisce il rendering della camera come immagine jpg, trasformata in un array di byte.

```
byte[] GetImage()
{
   RenderTexture.active = renderTexture;
   texture.ReadPixels(new Rect(0, 0, renderTexture.width, renderTexture.height), 0, 0);
   texture.Apply();

   byte[] bytes = texture.EncodeToJPG();

   return bytes;
}
```

3.4 Head tracking

La parte di raccolta dati per il movimento dell'utente viene fatta interamente client side, utilizzando i sensori interni al dispositivo.
Per lo sviluppo di questa parte della tesi ho provato a collegarmi al lavoro di ricerca svolto da uno studente tedesco: Alexander Pacha [26].
La sua attività di ricerca riguardava lo studio e sviluppo di sensori virtuali più precisi e veloci di quelli offerti dal framework Android.
Dopo un periodo di test i suoi sensori si sono dimostrati poco accurati, decisamente non abbastanza per il tipo di utilizzo fatto in questa tesi. Si è deciso quindi di utilizzare un altro sensore virtuale: il giroscopio calibrato offerto dal framework Android. Con questo sensore il tracciamento è estremamente preciso e il drift quasi inesistente.

I dati ottenuti dal giroscopio vengono trasformati in quaternioni [27]. Rappresentare le rotazioni sotto forma di quaternioni porta diversi vantaggi. Primo fra tutti una notazione comune a quella utilizzata internamente da Unity. In secondo luogo, rispetto agli angoli di Eulero i quaternioni sono più facili da comporre e non hanno il problema del gimbal lock [28].

La parte di tracking è una componente fondamentale per garantire un'esperienza priva di motion sickness.

In particolare le informazioni sulla rotazione vengono conservate e composte all'interno del client, il quale si occupa di aggiornare continuamente la rotazione corrente, che viene semplicemente inviata al server, il cui unico compito è di applicarla all'oggetto della scena 3D.
Tenendo il calcolo e l'aggiornamento della rotazione sul client si evitano eventuali problemi dovuti alla latenza.

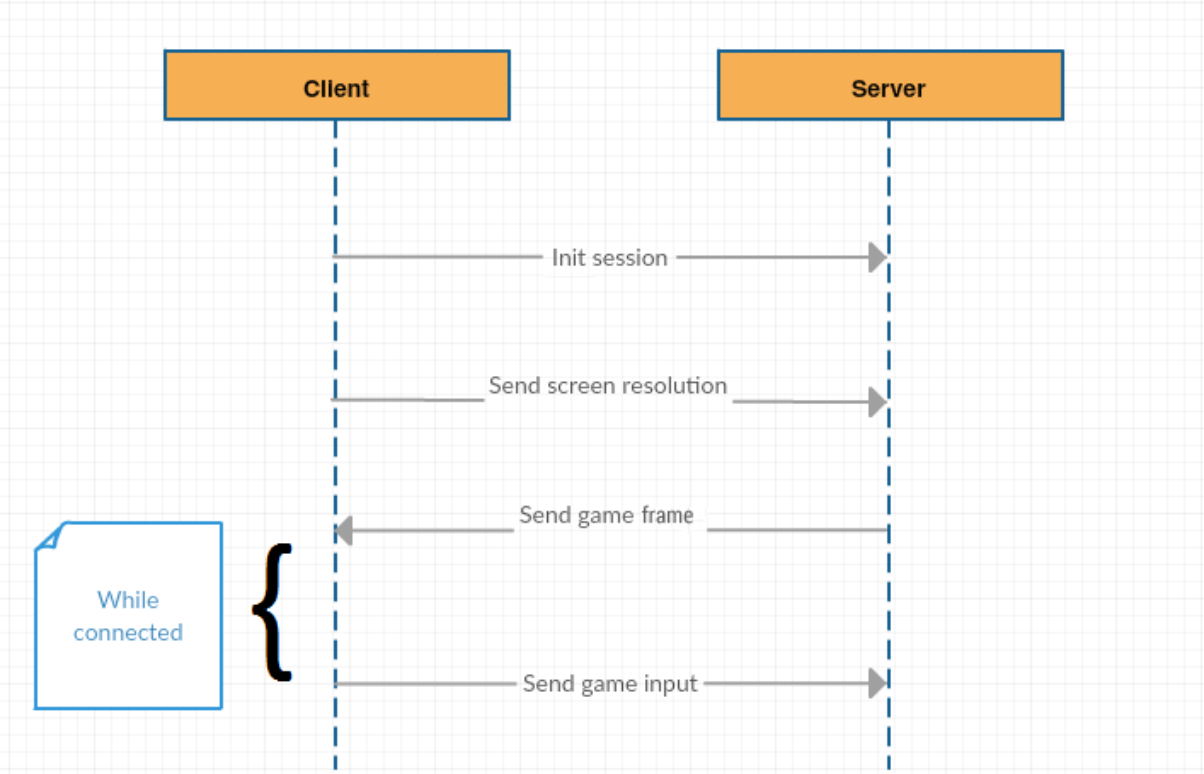
3.4.1 Formato dei dati

I dati trasmessi tra client e server sono tutti sotto forma di array di byte.
Ogni pacchetto rappresentante un dato di input (giroscopio e touch sullo schermo del client) è costituito da 1 + 4*4 byte.
Il primo byte serve per identificare il tipo di dato che si sta inviando. Gli ultimi 4*4 byte rappresentano il payload del pacchetto, ovvero l'informazione effettiva.
È stata scelta una dimensione di 4*4 byte perché il dato di dimensione maggiore che viene inviato è il quaternione, formato da 4 interi.
Per quanto riguarda l'input di tipo gesture, il payload è costituito da solo un intero, che indica il tipo di gesture rilevato.
È stata fatta la scelta di inviare dati della stessa lunghezza in modo da velocizzare la lettura lato server nel caso TCP. In questo modo è possibile fare sempre letture di 1 + 4*4 byte e poi cambiare il flusso d'esecuzione in base al dato ricevuto. Inviando array di dimensioni diverse sarebbe stato necessario fare sempre due letture. La prima per determinare il tipo di dato ricevuto e la seconda per leggere il dato effettivo.
Vista la frequenza con cui queste letture devono essere effettuate, il metodo scelto è vantaggioso.

Anche le immagini vengono inviate come array di byte, ovviamente di lunghezza variabile. In questo caso, usando TCP, in ricezione è inevitabile fare due letture. Infatti il server invia prima la dimensione dell'immagine e poi l'immagine effettiva. Quindi in client legge la dimensione e poi legge tanti byte quanto indicato dalla prima lettura.

3.4.2 IO

Il protocollo utilizzato per la comunicazione tra client e server è il seguente:



Ovviamente questa è una rappresentazione ad alto livello del protocollo di comunicazione, essendo realizzato utilizzando sia TCP che UDP ogni scambio ha delle caratteristiche specifiche dovute all'implementazione.

Ad esempio la richiesta di inizializzazione di una sessione, in TCP è realizzata semplicemente aprendo una connessione dal client al server, mentre in UDP è realizzata tramite l'invio di uno specifico messaggio dal client, a cui il server deve rispondere dando conferma, dato che UDP non fornisce garanzia di avvenuta ricezione.

Quando il server riceve la richiesta di iniziare una sessione crea una nuova istanza di *Player* e la aggiunge alla scena 3D, poi aspetta che gli venga comunicata la risoluzione dello schermo del client. A questo punto inizia un loop durante il quale il server manda un frame ogni volta che un nuovo frame è disponibile e il client manda periodicamente l'input fornito dall'utente. Il loop prosegue fino a che il client è connesso.

Visto il supporto alla multiutenza questo protocollo può essere eseguito più volte contemporaneamente per ogni istanza di client collegata al server.

3.5 Problematiche

Di seguito sono elencate alcune delle problematiche o incompletezze del progetto che non lo rendono pronto per essere usato in produzione.

3.5.1 Overhead introdotto da Unity

Il problema principale di tutto il progetto riguarda l'estrema difficoltà riscontrata nell'esecuzione di rendering ad alta risoluzione.
Il rendering server side si adatta alla risoluzione dello schermo del client. Al giorno d'oggi è abbastanza comune per gli schermi dei dispositivi mobile avere risoluzione elevate. Collegando un client con risoluzione di 1920x1080 l'immagine prodotta dal rendering è quindi pesante.
Il problema con queste immagini, controintuitivamente, è risultato essere server side, e non relativo alla trasmissione via rete.
Il rendering viene eseguito utilizzando la memoria della GPU, ma per essere trasformato in jpg e trasmesso via rete, i dati prodotti devono essere copiati dalla memoria della GPU a quella della CPU e questo manda in stallo la pipeline del rendering. Ciò si è dimostrato estremamente problematico dato che l'astrazione di alto livello fornita da Unity copre questi dettagli di basso livello e quindi non è facile trovare una soluzione.

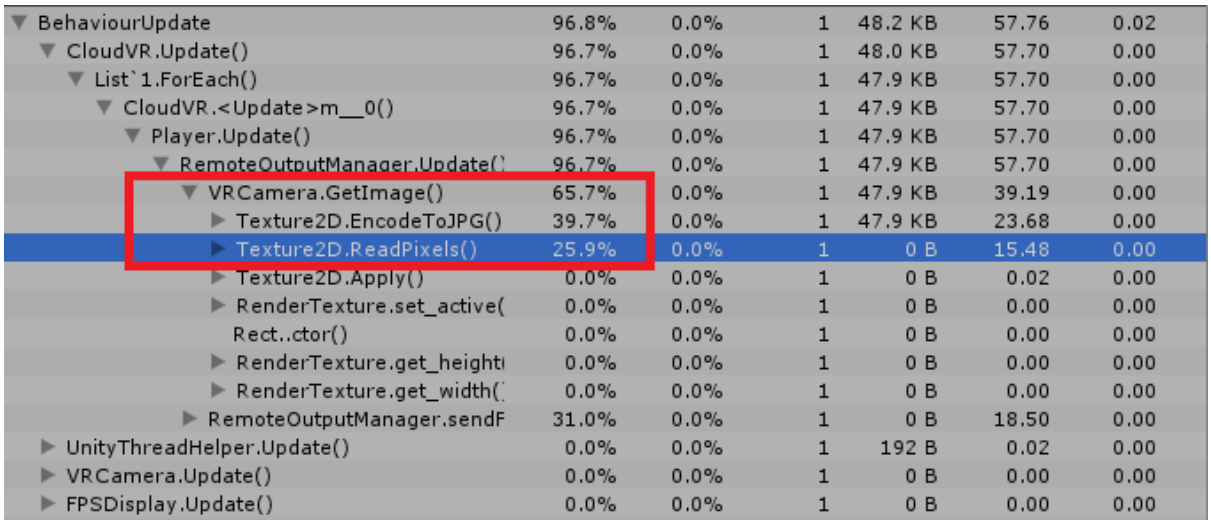
Analizzando l'esecuzione dell'applicazione utilizzando il profiler di Unity e renderizzando a 1920x1080 si può osservare che:
Ad ogni ciclo di update il 97% del tempo viene speso all'interno di metodi degli script relativi all'update.

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▶ BehaviourUpdate	96.8%	0.0%	1	48.2 KB	57.76	0.02
▶ GUI.Repaint	1.3%	0.0%	1	3.4 KB	0.79	0.03
▶ Camera.Render	0.8%	0.0%	3	0 B	0.48	0.05
▶ Physics.Processing	0.2%	0.1%	2	0 B	0.16	0.11
Overhead	0.1%	0.1%	1	0 B	0.10	0.10
Profiler.FinalizeAndSendFrame	0.1%	0.1%	1	0 B	0.08	0.08
▶ PlayerController.FixedUpdate()	0.1%	0.0%	2	0 B	0.06	0.00
▶ Monobehaviour.OnMouse_	0.1%	0.0%	1	0 B	0.06	0.00
▶ GameView.GetMainGameViewRenderRect()	0.1%	0.0%	1	0 B	0.06	0.00
AudioManager.Update	0.0%	0.0%	1	0 B	0.01	0.01
TextRendering.Cleanup	0.0%	0.0%	1	0 B	0.00	0.00
Physics.FetchResults	0.0%	0.0%	2	0 B	0.00	0.00
Physics.Simulate	0.0%	0.0%	2	0 B	0.00	0.00
▶ GUIUtility.SetSkin()	0.0%	0.0%	1	0 B	0.00	0.00

In particolare il 66% del tempo viene speso per ottenere l'immagine e il 31% per inviarla.

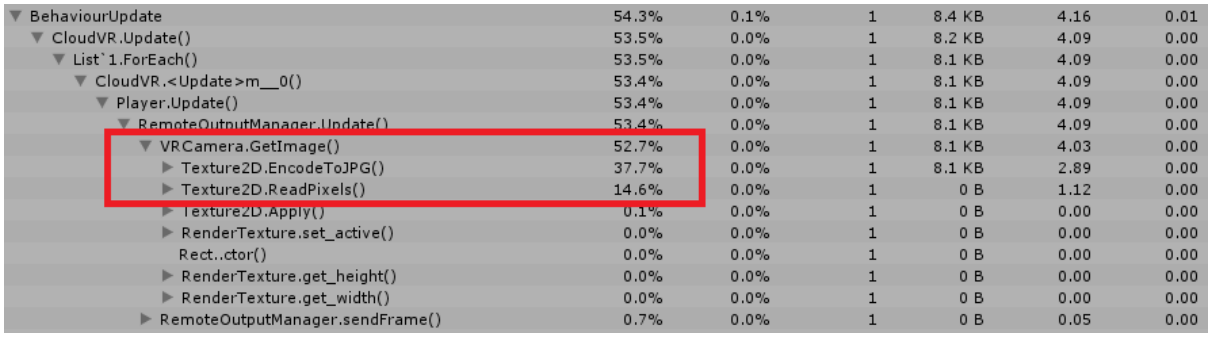
▼ BehaviourUpdate	96.8%	0.0%	1	48.2 KB	57.76	0.02
▼ CloudVR.Update()	96.7%	0.0%	1	48.0 KB	57.70	0.00
▼ List`1.ForEach()	96.7%	0.0%	1	47.9 KB	57.70	0.00
▼ CloudVR.<Update>m__0()	96.7%	0.0%	1	47.9 KB	57.70	0.00
▼ Player.Update()	96.7%	0.0%	1	47.9 KB	57.70	0.00
▼ RemoteOutputManager.Update()	96.7%	0.0%	1	47.9 KB	57.70	0.00
▶ VRCamera.GetImage()	65.7%	0.0%	1	47.9 KB	39.19	0.00
▶ RemoteOutputManager.sendF	31.0%	0.0%	1	0 B	18.50	0.00
▶ UnityThreadHelper.Update()	0.0%	0.0%	1	192 B	0.02	0.00
▶ VRCamera.Update()	0.0%	0.0%	1	0 B	0.00	0.00
▶ FPSDisplay.Update()	0.0%	0.0%	1	0 B	0.00	0.00

Nella lettura dell'immagine 40% del tempo viene speso per codificarla in jpg e il 26% del tempo viene speso per salvare l'immagine all'interno di una texture (dalla quale verrà estratta come jpg).



BehaviourUpdate	96.8%	0.0%	1	48.2 KB	57.76	0.02
CloudVR.Update()	96.7%	0.0%	1	48.0 KB	57.70	0.00
List`1.ForEach()	96.7%	0.0%	1	47.9 KB	57.70	0.00
CloudVR.<Update>m__0()	96.7%	0.0%	1	47.9 KB	57.70	0.00
Player.Update()	96.7%	0.0%	1	47.9 KB	57.70	0.00
RemoteOutputManager.Update()	96.7%	0.0%	1	47.9 KB	57.70	0.00
VRCamera.GetImage()	65.7%	0.0%	1	47.9 KB	39.19	0.00
Texture2D.EncodeToJPG()	39.7%	0.0%	1	47.9 KB	23.68	0.00
Texture2D.ReadPixels()	25.9%	0.0%	1	0 B	15.48	0.00
Texture2D.Apply()	0.0%	0.0%	1	0 B	0.02	0.00
RenderTexture.set_active(	0.0%	0.0%	1	0 B	0.00	0.00
Rect..ctor()	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.get_height	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.get_width(	0.0%	0.0%	1	0 B	0.00	0.00
RemoteOutputManager.sendF	31.0%	0.0%	1	0 B	18.50	0.00
UnityThreadHelper.Update()	0.0%	0.0%	1	192 B	0.02	0.00
VRCamera.Update()	0.0%	0.0%	1	0 B	0.00	0.00
FPSDisplay.Update()	0.0%	0.0%	1	0 B	0.00	0.00

Calando la risoluzione dell'immagine il tempo speso per inviarla tramite la rete si riduce praticamente a 0. Per ottenere l'immagine invece si spende circa il 53% del tempo, il cui 38% viene speso per codificarla in jpg.



BehaviourUpdate	54.3%	0.1%	1	8.4 KB	4.16	0.01
CloudVR.Update()	53.5%	0.0%	1	8.2 KB	4.09	0.00
List`1.ForEach()	53.5%	0.0%	1	8.1 KB	4.09	0.00
CloudVR.<Update>m__0()	53.4%	0.0%	1	8.1 KB	4.09	0.00
Player.Update()	53.4%	0.0%	1	8.1 KB	4.09	0.00
RemoteOutputManager.Update()	53.4%	0.0%	1	8.1 KB	4.09	0.00
VRCamera.GetImage()	52.7%	0.0%	1	8.1 KB	4.03	0.00
Texture2D.EncodeToJPG()	37.7%	0.0%	1	8.1 KB	2.89	0.00
Texture2D.ReadPixels()	14.6%	0.0%	1	0 B	1.12	0.00
Texture2D.Apply()	0.1%	0.0%	1	0 B	0.00	0.00
RenderTexture.set_active()	0.0%	0.0%	1	0 B	0.00	0.00
Rect..ctor()	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.get_height()	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.get_width()	0.0%	0.0%	1	0 B	0.00	0.00
RemoteOutputManager.sendFrame()	0.7%	0.0%	1	0 B	0.05	0.00

Chiaramente sarebbe ottimale velocizzare il processo di decodifica dell'immagine per velocizzare l'applicazione. A tale proposito sono stati fatti tentativi utilizzando chiamate OpenGL di basso livello ma, come detto prima, Unity limita fortemente la libertà d'azione.

Ovviamente più utenti sono collegati contemporaneamente e più il problema è visibile, causando un calo sensibile al frame rate dell'applicazione.

Il problema è stato temporaneamente aggirato riducendo la risoluzione del rendering di un certo fattore. Tenendo questo fattore abbastanza basso è possibile ridurre la qualità dell'immagine quanto basta per alleggerire l'esecuzione del server (in mondo da mantenere un'esecuzione a 60 fps) e allo stesso tempo mantenere una buona qualità visiva client side.

3.5.2 Possibili estensioni e miglioramenti

Per migliorare le performance dell'applicazione server side sarebbe interessante studiare metodi alternativi per ottenere l'immagine dell'applicazione 3D.
Una via potrebbe essere quella di lasciare la funzionalità interna all'applicazione 3D però scendendo più a basso livello, ottimizzando il più possibile il passaggio dei dati.
L'alternativa sarebbe di trovare un modo per catturare l'output dell'applicazione 3D esternamente all'applicazione 3D stessa. Questa modalità sarebbe interessante perché (assieme ad un'opportuna interfaccia verso l'input dell'applicazione 3D) permetterebbe di interagire con applicazioni di terze parti, non pensate per funzionare in modalità remota. Un lavoro simile viene portato avanti da Riftcat con VRidge [29].

Un'altra area del progetto su cui lavorare potrebbe essere l'ottimizzazione dello streaming delle immagini. Come detto precedentemente con una risoluzione di 1920x1080 il tempo perso per inviare l'immagine corrisponde al 31% del tempo di ogni update.

Sarebbe anche utile implementare un algoritmo per il controllo della qualità. In modo da aggiustare la qualità del rendering in base al numero di sessioni attive, alla velocità della rete e alla potenza della macchina host. Per questa tesi è stato realizzato un semplice algoritmo per il controllo della qualità, ma al momento non prende in considerazione molti scenari, quindi non è stato incluso nel progetto finale.

Queste migliorie, assieme all'utilizzo di protocolli più adatti allo streaming (RTP), aprirebbero la strada ad un utilizzo anche esterno alle reti LAN, sbloccando la vera potenzialità del cloud gaming.

Un'altra componente importante e non ancora implementata è la funzionalità di streaming dell'audio della scena 3D.

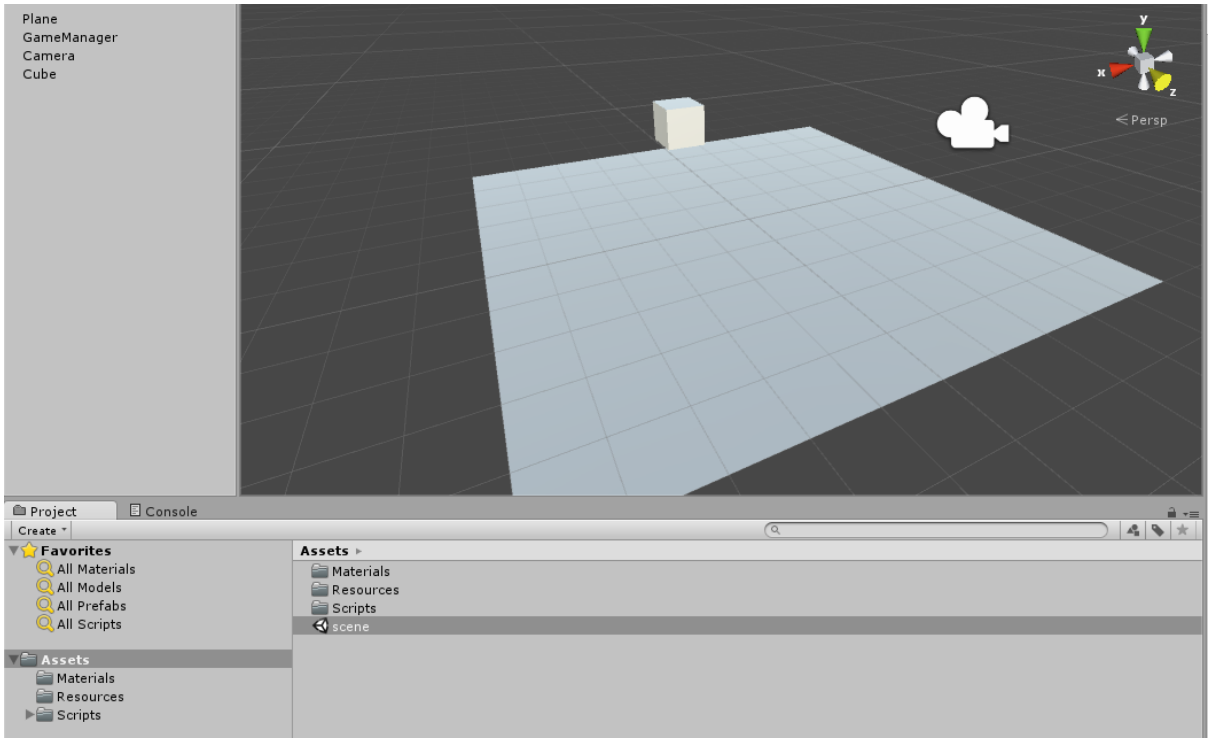
Sarebbe inoltre interessante sfruttare maggiormente il sistema di sensori presenti all'interno dell'ecosistema Android per provare modalità d'interazione alternative. Ad esempio utilizzando il GPS o il conta passi di uno smartwatch per collegare gli spostamenti nel mondo reale a quelli nel mondo virtuale. O il microfono del telefono per impartire comandi vocali all'applicazione.

4 Esempi di utilizzo

4.1 Prototipazione

Essendo estremamente semplice da integrare in qualsiasi progetto, allo stato attuale CloudVR potrebbe essere utilizzato da degli sviluppatori per prototipare rapidamente e validare le idee prima di iniziare lo sviluppo effettivo e prima di sostenere la spesa relativa all'acquisto di un headset di qualità.
Questo approccio è vantaggioso rispetto all'utilizzo dell'SDK di Cardboard appunto per la sua semplicità d'utilizzo. Non è necessario imparare ad usare alcun SDK o avere conoscenze particolari, oltre saper usare Unity.

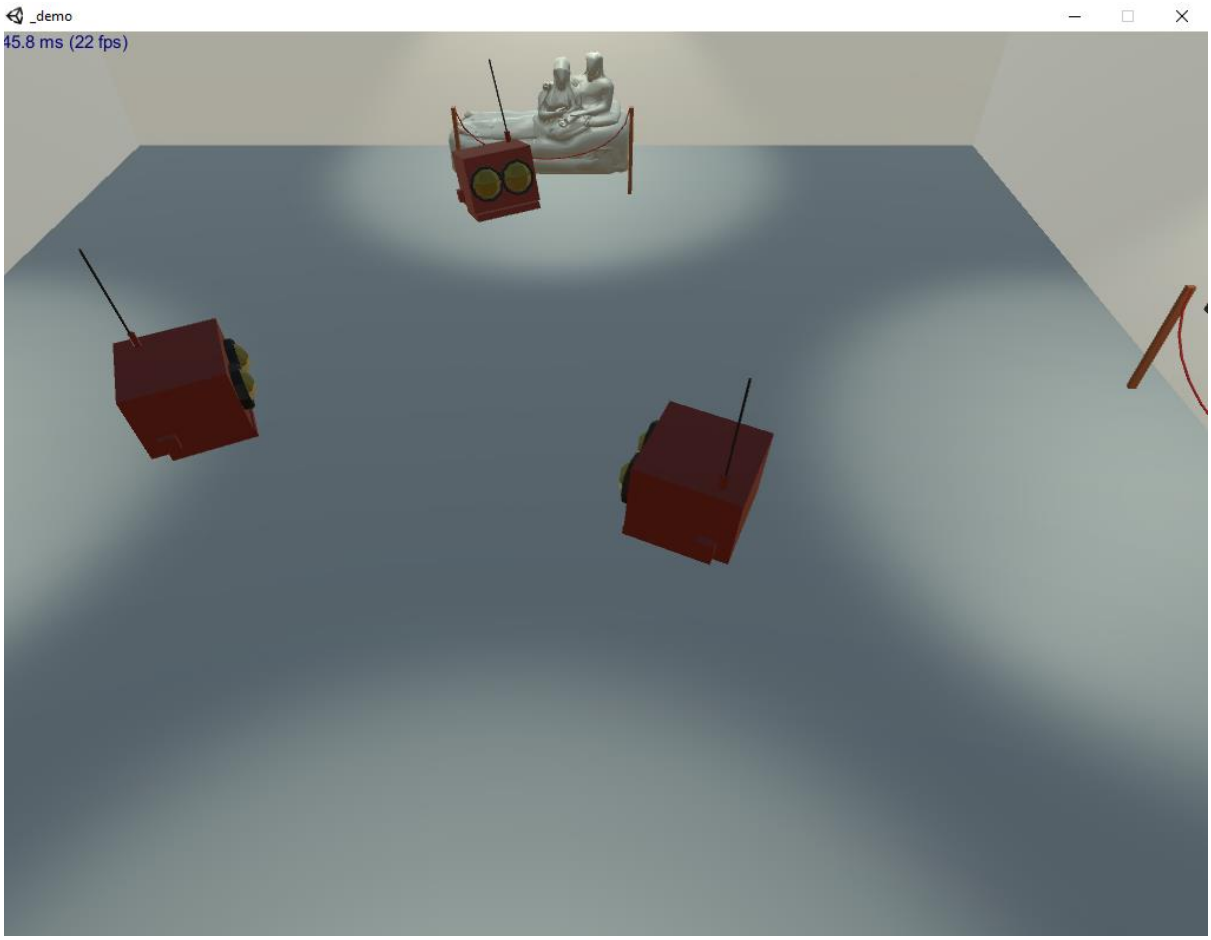
Per importare gli script e le risorse del server all'interno di un progetto Unity è sufficiente aprire il file (o importarlo tramite Unity) "RemoteVR.unitypackage". Il package contiene anche una semplice scena già pronta per iniziare lo sviluppo.



Scena di base all'interno di RemoteVR.unitypackage

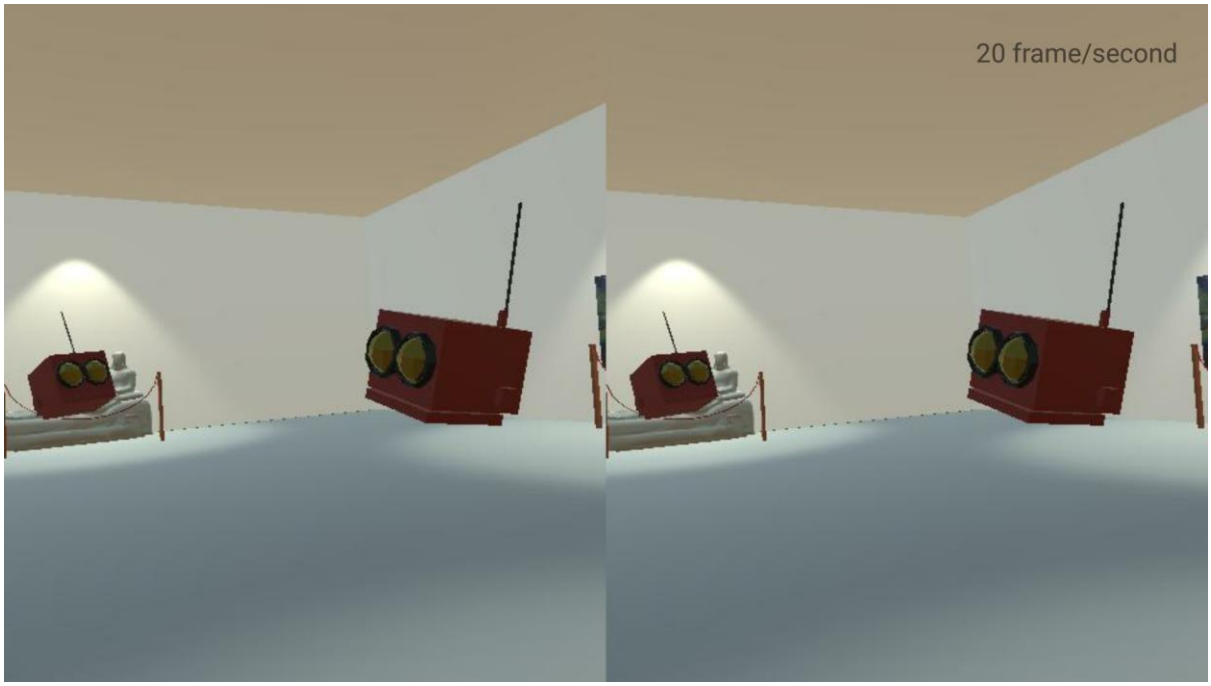
4.2 Demo museo

La demo del museo è stata realizzata per fornire un semplice esempio di utilizzo del lavoro realizzato in questa tesi.
La scena 3D rappresenta una semplice stanza di un museo, con dei quadri e il modello 3D di un sarcofago egizio. Più persone possono collegarsi ed essere contemporaneamente all'interno della stessa stanza del museo.
Per poter usare efficacemente il lavoro di questa tesi in applicazioni di questo tipo (in produzione), sarebbe necessario risolvere alcune delle problematiche elencate precedentemente.



Stanza museo, con tre utenti collegati
(l'immagine è ottenuta tramite una camera che renderizza unicamente lato server, aggiunta alla scena apposta per questo screenshot)

Prospettiva utente n.1



Prospettiva utente n.2

5 Conclusioni

Il prototipo è funzionante ed è servito a validare l'idea originale, mostrando le potenzialità e le problematiche di questo tipo di approccio.

È stato dimostrato che è relativamente facile ottenere applicazioni di realtà virtuale nel cloud e che, una volta soddisfatto questo requisito, la multiutenza e l'interazione tra sistemi operativi e visori diversi non è un problema.

L'applicazione è pensata per funzionare all'interno dell'ambiente ottimale fornito da una rete LAN, quindi usare TCP o UDP è stato quasi indifferente.
UDP si è dimostrato leggermente più snello nelle rarissime situazioni critiche verificatesi; visto il tipo di applicazione non è un problema perdere sporadicamente qualche frame. Nelle medesime situazioni invece TCP ha comportato un leggero ritardo.
Per supportare comunicazioni al di fuori di una LAN sarebbe necessario usare protocolli più adatti allo streaming in real-time (RTP).

Quindi seppure funzionante il prototipo ottenuto non è completo e necessita ulteriore lavoro per trovare soluzioni migliori ai problemi di performance rilevati lato server e a livello di trasmissione.

Bibliografia

[1] "Cardboard." *Google – Google VR*. Web. 27 June 2016. https://vr.google.com/cardboard/index.html
[2] "Daydream - Google VR." *Daydream - Google VR*. Web. 27 June 2016. https://vr.google.com/daydream/
[3] "HTC Vive." *Vive*. Web. 27 June 2016. https://www.htcvive.com/eu/
[4] "Managing the Activity Lifecycle." *Android Developers*. Web. 27 June 2016. https://developer.android.com/training/basics/activity-lifecycle/index.html
[5] "Fragment lifecycle." *Android Developers*. Web. 27 June 2016. https://developer.android.com/guide/components/fragments.html#Creating
[6] Android View https://developer.android.com/reference/android/view/View.html
[7] "Drawable Resources." *Android Developers*. Web. 27 June 2016. https://developer.android.com/guide/topics/resources/drawable-resource.html
[8] "Sensors Overview." *Android Developers*. Web. 27 June 2016. https://developer.android.com/guide/topics/sensors/sensors_overview.html
[9] "Sensor Fusion." *Wikipedia*. Wikimedia Foundation. Web. 27 June 2016. https://en.wikipedia.org/wiki/Sensor_fusion
[10] "Unity." *Unity*. Web. 27 June 2016. https://unity3d.com/
[11] "Asset Store." *Asset Store Unity*. Web. 27 June 2016. https://www.assetstore.unity3d.com/
[12] "Prefabs." *Unity*. Web. 27 June 2016. http://docs.unity3d.com/Manual/Prefabs.html
[13] "Cross Platform IDE for C#, F# and More." *MonoDevelop*. Web. 27 June 2016. http://www.monodevelop.com/
[14] "Visual Studio - Microsoft Developer Tools." *Visual Studio - Microsoft Developer Tools*. Web. 27 June 2016. https://www.visualstudio.com/
[15] "MonoBehaviour." *Unity*. Web. 27 June 2016. https://docs.unity3d.com/ScriptReference/MonoBehaviour.html
[16] ".NET Framework." - *Wikipedia*. Web. 27 June 2016. https://it.wikipedia.org/wiki/.NET_Framework
[17] “Crytek” - Crytek. Web. 27 June 2016. http://www.crytek.com/
[18] "Crysis." - *Wikipedia*. Web. 27 June 2016. https://it.wikipedia.org/wiki/Crysis
[19] "OnLive" *OnLive*. Web. 27 June 2016. http://onlive.com/
[20] "Gaikai Inc. A Sony Computer Entertainment Company." *Gaikai.com*. Web. 27 June 2016. https://www.gaikai.com/
[21] "PlayStation Now." *Playstation*. Web. 27 June 2016. https://www.playstation.com/en-us/explore/playstationnow/
[22] "ViewPager." *Android Developers*. Web. 27 June 2016. https://developer.android.com/reference/android/support/v4/view/ViewPager.html
[23] "ReactiveX/RxJava." *GitHub*. Web. 27 June 2016. https://github.com/ReactiveX/RxJava
[24] "Otto." *Otto*. Web. 27 June 2016. http://square.github.io/otto/
[25] "JakeWharton/butterknife." *GitHub*. Web. 27 June 2016. https://github.com/JakeWharton/butterknife
[26] "Bitbucket." *Apacha / Sensor Fusion Demo* —. Web. 27 June 2016. https://bitbucket.org/apacha/sensor-fusion-demo

[27] "Quaternion." *Wikipedia*. Wikimedia Foundation. Web. 27 June 2016. https://en.wikipedia.org/wiki/Quaternion
[28] "Gimbal Lock." *Wikipedia*. Wikimedia Foundation. Web. 27 June 2016. https://en.wikipedia.org/wiki/Gimbal_lock
[29] "*RiftCat*." *RiftCat*. Web. 27 June 2016. https://riftcat.com/vridge