ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

STUDIO E SVILUPPO DI UN FRAMEWORK IN AMBIENTE ANDROID PER LA REALIZZAZIONE DI INTERFACCE UTENTE INNOVATIVE BASATE SU SMART GLASS

Relazione finale in
PROGRAMMAZIONE DI SISTEMI EMBEDDED

Relatore
Prof. ALESSANDRO RICCI

Co-relatore
Dott. ANGELO CROATTI

Presentata da
MANUEL BOTTAZZI

III Sessione di Laurea
Anno Accademico 2014 / 2015

PAROLE CHIAVE

Wearable Computing

Augmented Reality

Interfaccia Utente

Smart Glass

Android

A coloro che si sono fatti spazio nel mio cuore,
che sono la benzina dei miei passi.

Indice

Introduzione

"Ogni tecnologia sufficientemente avanzata è indistinguibile dalla magia" scriveva qualche anno fa Arthur C. Clarke; e tale affermazione diventa tanto più vera tanto ci si addentra nelle ultime tecnologie informatiche.Tali tecnologie hanno subito nel corso degli ultimi decenni numerose e incredibili evoluzioni. Uno degli ultimi e forse più importanti cambiamenti è stata l'enorme diffusione di dispositivi portatili e mobile, i quali hanno ormai raggiunto una capacità di calcolo paragonabile a quella di un computer desktop di media potenza. Questo tipo di tecnologie ha rappresentato un forte cambiamento nelle abitudini e nelle modalità di utilizzo della tecnologia da parte dell'utente medio.

Nell'ultimo periodo stanno emergendo nuove tecnologie, di tipo Embedded, ossia "nascoste" all'interno di oggetti di uso quotidiano, ai quali vengono di conseguenza conferite capacità computazionali. Questa tesi si inserisce nell'ambito di questi dispositivi, ed in particolare nel campo dei dispositivi Wearable, ossia tutti quei dispositivi che possono essere direttamente indossati sul corpo dell'utilizzatore per ottenere gli scopi più svariati. Questo tipo di tecnologie potrebbe rappresentare un nuovo cambiamento nel modo di interfacciarsi alla tecnologia da parte delle persone, analogo a quanto successo con gli smartphone. Queste tecnologie tuttavia hanno ancora bisogno di svilupparsi per mostrare al mondo la loro potenzialità. Lo scopo di questo lavoro è quello di esplorare e testare tramite un semplice prototipo il comportamento di un nuovo tipo di interfaccia utente con cui è possibile interagire con tali sistemi, in particolare con gli Smart Glass, che rappresentano la versione Embedded di un semplice paio di occhiali. Lo scopo di tale interfaccia sarà quello di semplificare l'interazione dell'utente, avvicinando l'esperienza di utilizzo dei dispositivi a quella che è la normale esperienza visiva, per rendere il più possibile piacevole e semplice l'utilizzo di questo tipo di tecnologie.

Capitolo 1

I sistemi Embedded

Iniziamo questo lavoro con una breve panoramica sullo stato dell'arte della tecnologia Embedded e Wearable.

1.1 Cos'è un sistema Embedded ?

Un sistema Embedded è un sistema elettronico configurato per svolgere un compito ben preciso e determinato (*sistema Special-Purpose*) interagendo con il mondo fisico che lo circonda, inserito all'interno di un altro sistema o dispositivo di diverse dimensioni. Il dispositivo embedded è spesso costituito da una parte hardware e da una componente software, ed è costruito per funzionare in maniera continuativa, ricevendo informazioni dal mondo fisico (tramite sensori) e calcolando risposte in output (che possono essere informazioni o effetti veri e propri sul mondo stesso tramite l'uso di attuatori fisici). Spesso sistemi di questo tipo hanno importanza critica, alcune volte addirittura vitale (per esempio in sistemi biomedicali o di controllo di mezzi di trasporto), per cui è necessario che rispondano a particolari criteri di **affidabilità** e **sicurezza**

A tali sistemi è richiesta quindi :

- **Affidabilità**: alta probabilità di continuo e corretto funzionamento.
- **Reattività**: alta capacità di rispondere in tempi brevi a eventi e stimoli provenienti dal mondo esterno, eventualmente anche in *real-time*, senza ritardi.
- **Sicurezza**: non deve recare danno agli utenti e/o all'ambiente circostante.
- **Privacy**: deve garantire riservatezza ai dati dell'utente.

Inoltre tali sistemi hanno il grande problema della scarsità di risorse hardware (capacità computazionali, memoria, batteria), per cui nel momento della loro progettazione è necessario tenere conto di tali limiti e sviluppare una progettazione orientata ad ottenere la massima efficienza possibile (in termini di costo, dimensioni, consumo di energia e performance) con le risorse a disposizione.

1.2 Un caso particolare di sistemi Embedded: Tecnologie Wearable

Un dispositivo Wearable, come suggerisce il nome, è un particolare dispositivo Embedded che l'utente indossa direttamente sul suo corpo per utilizzarlo. Tecnologie di questo tipo rappresentano una novità sul mercato, ma sono un settore in forte crescita. Infatti secondo alcuni studi effettuati dall'International Data Corporation (IDC) il mercato mondiale dei dispositivi Wearable raggiungerà un totale di 111.1 milioni di pezzi venduti nel 2016, (circa il 44.4% percento in più delle previsioni sul 2015), arrivando addirittura ad una stima di 214.6 milioni di unità nel 2019.

Dispositivi di questo tipo sono progettati per essere **costanti**, ossia sempre funzionanti e pronti per poter interagire con essi in qualsiasi momento, e devono **supportare l'utente** nel suo compito: Al contrario di quanto si possa pensare infatti, il compito principale per cui sono stati ideati dispositivi di questo tipo non è quello di eseguire calcoli. Infatti l'utente utilizza il dispositivo mentre sta svolgendo un qualche tipo di attività, e lo scopo fondamentale dei wearable computer è quello di aiutarlo e supportarlo in tale attività, aumentandone in qualche modo i sensi, la percezione e/o la conoscenza. Questo concetto prende il nome di **Augmentation**

Esistono svariate tipologie di questi dispositivi con funzioni molto diverse tra loro, ma tutti presentano caratteristiche che li accomunano e che in qualche modo definiscono tali dispositivi. Essi devono essere:

- **Controllabile**: L'utente deve poter prendere il controllo del dispositivo in qualsiasi momento, a suo piacimento.
- **Osservabile**: L'utente, se vuole, deve poter vedere l'output prodotto dal dispositivo continuamente
- **Attento all'ambiente**: Il dispositivo deve essere in grado di misurare l'ambiente fisico che lo circonda tramite sensori
- **Comunicativo**: Il dispositivo deve essere in grado di comunicare, attraverso la rete o altri tipi di connessioni, con altri dispositivi.

- **Non restrittivo**: Deve essere comodo da portare, non ingombrante, e deve permettere all'utente di fare altro mentre lo si utilizza
- **Non monopolizzante**: Deve essere progettato in modo che permetta all'utente di svolgere altre azioni durante l'utilizzo, in modo da non monopolizzare la sua attenzione o estraniarlo dal mondo reale (come avviene per esempio con la realtà virtuale).

In particolare gli ultimi due punti mettono in luce qual'è la grande novità portata da dispositivi di questo tipo rispetto a tecnologie mobile oggi ampiamente diffuse, e come questo tipo di dispositivi aprano un nuovo modo di interagire ed interfacciarsi con essi. Tali dispositivi diventano infatti veramente **personali**, adattabili in modo che siano una vera estensione del corpo e della mente, che si possono tenere sempre con se e che alla lunga ti fanno dimenticare di averli addosso, permettendo di interfacciarsi con essi mentre si stanno svolgendo altre azioni, lasciando di fatto libere le mani dell'utilizzatore.

Vediamo ora alcuni campi di applicazione e alcuni esempi di tali dispositivi.

1.2.1 Wearable nello sport

L'ambito in cui le tecnologie wearable si sono forse diffuse maggiormente all'interno del mercato è quello legato allo sport e al fitness. Sono infatti presenti sul mercato un gran numero di dispositivi che, a prezzi relativamente bassi, permettono di monitorare il ritmo cardiaco, i passi effettuati, le calorie bruciate, ecc.

Tali dispositivi sono presenti in varie forme, dai braccialetti alle clip da attaccare ai vestiti. Vediamo ora due esempi commericiali di tali dispositivi

FitBit Flex

Figura 1.1: FitBit Flex

Il dispositivo mostrato in figura è un braccialetto contenente un dispositivo che tramite un accelerometro misura i passi e la distanza percorsa, i minuti di

attività e le calorie bruciate dall'utilizzatore. Permette inoltre di monitorare la qualità e i cicli del sonno. Il bracciale si collega in automatico (tramite bluetooth) con lo smartphone dell'utente, comunicando i dati all'apposita App, con cui l'utente potrà visualizzare le statistiche del suo allenamento.

La stessa ditta produce versioni più o meno costose di questi braccialetti, con funzionalità aggiuntive (per esempio il cardiofrequenzimetro o uno schermo per la visualizzazione diretta dei dati) all'aumentare del prezzo.

Nike+ SportBand

Figura 1.2: Nike+ SportBand

Tale dispositivo adotta un diversa modalità per la misurazione dell'attività fisica. Infatti il sensore visibile in figura dovrà essere inserito all'interno della scarpa sinistra dell'utente in uno spazio sotto il plantare (disponibile in modelli di scarpe appositi) ed esso misurerà andatura, distanza, durata e calorie bruciate durante l'allenamento. Il sistema può essere inoltre collegato tramite un ricevitore ad un IPod per ricevere un feedback audio in tempo reale durante l'allenamento.

1.2.2 Gli Smart watch

Gli smartwatch sono orologi che danno la possibilità, oltre alle loro funzionalità di base, di eseguire vere e proprie applicazioni, e di comunicare con il proprio smartphone (collegato tramite bluetooth), visualizzando messaggi e notifiche ed in alcuni casi anche rispondendo/effettuando chiamate direttamente dal proprio polso. Si va dalle versioni più semplici, dove all'interno di un orologio analogico è inserito un sistema digitale che permette di interagire con il proprio smartphone, fino a sistemi più complessi, con schermo touch, che facilità l'interazione con l'orologio stesso, e un proprio sistema operativo dedicato. Molte aziende si stanno interessando in questi ultimi anni a questo tipo di prodotto, con il risultato di avere sul mercato un gran numero di questi dispositivi. Vediamo un paio di esempi

Apple Watch

Figura 1.3: Apple Watch

Apple Watch è uno smartwatch prodotto da Apple, disponibile a partire da Aprile 2015. Esso presenta tutta una serie sensori quali accelerometro, giroscopio, barometro e sensore di battito cardiaco, e permette di eseguire applicazioni provenienti dall'AppStore. Permette inoltre di comunicare con altri dispositivi tramite NFC, Bluetooth o Wi-Fi. Esso è dotato di schermo touch, con il quale è possibile interagirvi, e di un microfono che permette l'utilizzo anche tramite comandi vocale (se esso è collegato ad un IPhone). Il sistema operativo di cui è dotato prende il nome di *watchOS*.

Smartwatch Android

A partire dal 2014 Android ha rilasciato una specifica versione del suo sistema operativo per dispositivi wearable, che prende il nome di *Android Wear*. Molte aziende hanno sviluppato i loro smartwatch appoggiandosi a questo sistema operativo, per coi non ne esiste uno solo, ma tanti modelli provenienti da aziende diverse. Un esempio è lo ZenWatch di Asus, che si vede in figura 1.4, ma potrebbero esserne citati molti altri. Questi dispositivi in generale sono dotati di schermo touch, ed è possibile collegarli ad uno smartphone tramite bluetooth, con il quale interagiscono continuamente, e come nel caso precedente danno la possibiltà di eseguire app.

Figura 1.4: ASUS ZenWatch (WI500Q)

1.2.3 Wearable in campo medico

Un altro settore dove stanno nascendo innumerevoli prodotti legati alle tecnologie wearable è quello sanitario. Con tali dispositivi per esempio e possibile monitorare i parametri vitali di un paziente, anche per periodi prolungati, e inviare tali informazioni a distanza direttamente al medico.

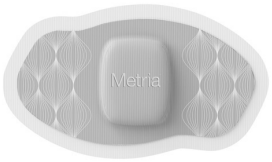

Figura 1.5: Cerotto Smart Metria

Un esempio interessante potrebbe essere il cerotto Metria, prodotto da Avery Dennison Medical Solutions. Esso è pensato per garantire un controllo continuo e sicuro dei parametri fisiologici di un paziente, senza essere più ingombrante di un normale cerotto, e quindi comodo da indossare. Il cerotto funziona grazie ad una serie di sensori in esso contenuti, come per esempio un sensore per l'ECG o un accelerometro che permette di misurare il battito cardiaco, l'attività, il sonno ecc. Il cerotto è inoltre equipaggiato con uno speciale adesivo, che permette di tenerlo addosso anche durante le attività di tutti i giorni

1.2.4 Smart Clothing

Le tecnologie wearable non si sono fermate a semplici accessori come bracciali o occhiali da indossare sul corpo, ma sono entrate anche nel campo della

sartoria stessa. è infatti possibile inserire all'interno degli stessi capi di abbigliamento sensori, microfoni, e antenne per la connessione wireless, dotandoli di capacità simili a quelle viste nei dispositivi citati sopra, come il monitoraggio dei parametri vitali e/o dell'attività fisica.

Figura 1.6: ThermalVision Radiate

Un esempio potrebbe essere la maglia Radiate, visibile in figura. Tale maglia permette di tenere sotto controllo la temperatura del corpo durante lo svolgimento di qualche attività fisica semplicemente osservandone la colorazione. A seconda del livello di calore rilevato essa riesce a capire quali muscoli stanno lavorando e con quale intensità, regolando la propria colorazione di conseguenza.

1.2.5 Gli Smart Glass

Gli Smart Glass sono speciali occhiali progettati per aggiungere informazioni a ciò che l'utilizzatore vede. Tali tipi di sistema offrono nuove possibilità di interazione con l'utente, dai comandi vocali alla *gesture recognition*Essi trovano applicazione in svariati campi della realtà aumentata, in cui una serie di informazioni si lega con la realtà che circonda l'utente. Tale argomento sarà ampiamente trattato ed approfondito nel capitolo 2.

Vediamo qualche esempio di tali sistemi.

Google Glass

Figura 1.7: Google Glass

I Google Glass sono sono il risultato di un progetto di Google, e rappresentano degli occhiali per la realtà aumentata basati su sistema operativo Android. Il display ad alta definizione è montato su una lente e proietta le immagini direttamente sugli occhi dell'utente. le informazioni sono visualizzate come se ci si trovasse di fronte a uno schermo da 25 pollici ad una distanza di due metri.

Per interagire con essi è presente sul lato destro della montatura un touchpad utile a scorrere tra i menu e tra i contenuti, ed è possibile comandarli anche tramite comandi vocali. Presentano inoltre connettività Wi-Fi e Bluetooth che permette anche il collegamento con uno smartphone Android.

Tali Glass offrono tantissime funzionalità, tra le quali effettuare ricerche su Google, leggere notizie online, controllare i social network e condividere contenuti tramite essi, avviare videoconferenze mostrando ciò che si sta guardando (tramite l'app Hangout), telefonare, visualizzare e inviare sms, tradurre un testo da una lingua di origine a una di destinazione, scattare fotografie, registrare video e utilizzare Google Maps per ottenere indicazioni stradali.

Epson Moverio

Figura 1.8: Epson Moverio BT-200

Anche questi occhiali come i precedenti si basano su tecnologia Android (in particolare montano la versione 4.0.4), ma lo schermo su cui vengono proiettate le immagini è di tipo binoculare e interessa quindi entrambi gli occhi, per una

dimensione risultante di 40 pollici a 2,5 m, fino a 320 pollici a 20 metri. Grazie a questa particolarità è possibile utilizzarli anche per funzioni 3D. Sono inoltre dotati di un trackpad con cui è possibile scorrere sullo schermo mediante un cursore per facilitare l'interazione con il dispositivo. Questo modello presenta una serie di sensori (GPS, bussola, giroscopio e accelerometro), alcuni dei quali presenti sia nel trackpad sia negli occhiali stessi, con i quali è possibile ad esempio controllare i movimenti della testa. È disponibile un apposito store (Moverio Apps Market) con il quale è possibile accedere ad una serie di App appositamente sviluppate per funzionare su tali dispositivi.

Capitolo 2

Nuove forme di interazione interazione con le macchine

L'introduzione sul mercato di un gran numero di dispositivi Embedded e Wearable provoca inevitabilmente un cambiamento nel modo che l'utente ha di interfacciarsi con il dispositivo elettronico, rendendo possibile nuove forme di interazione uomo macchina che fino a pochi anni fa sarebbero state impensabili.

2.1 Human-Computer Interaction

Nei primi anni 2000 il cambiamento introdotto dagli smartphone è stato enorme, permettendo di unire in un unico dispositivo le funzionalità che prima erano svolte da un gran numero di dispositivi diversi (fare telefonate, foto, video, ascoltare musica, ...). Il grande limite, ancora oggi presente, di tali dispositivi è la lentezza, o meglio il tempo che intercorre tra il pensiero di effettuare una determinata azione e il risultato di tale azione. Infatti in uno smartphone tale lasso di tempo è dell'ordine di una ventina di secondi. Se tale tempo è troppo lungo scoraggia l'utente dall'utilizzo, e incide sull'effettivo utilizzo di un certo dispositivo. Un'ulteriore problema presentato da tali dispositivi è la monopolizzazione dell'attenzione dell'utente e l'ingombro delle mani, rendendo di fatto quasi impossibile fare altro mentre si fa un utilizzo intensivo dello smartphone. Le tecnologie Wearable offrono una soluzione a tale problema, rendendo possibile un'interazione estremamente veloce, abbastanza veloce da rendere in alcuni casi il dispositivo un'estensione stessa della persona. Ed inoltre offrono modalità di interazione nuove e ottimali, permettendo interazioni veloci e senza la necessità di ingombrare le mani, come ad esempio attraverso comandi vocali o gesti delle mani. Questo sviluppo *hands-free* dell'interazione con i dispositivi assume un'importanza ancora maggiore

se la si pensa applicata in determinati settori del mondo del lavoro e della vita privata.

L'obiettivo finale del Wearable Computing è il potenziamento della persona stessa (*Human Augmentation*), estendendo le potenzialità del corpo e della mente dell'utente tramite l'interazione e la sinergia tra l'uomo e i dispositivi che indossa, arrivando infine ad un miglioramento della qualità complessiva della vita. Affinché questo divenga possibile è necessario che tali dispositivi diventino una vera e propria estensione della volontà della persona, e l'utente quasi non si accorga di avere tali dispositivi addosso. Tuttavia secondo una recente indagine di Accenture [16] la maggior difficoltà incontrata dagli utenti nell'utilizzo di queste tecnologie di nuova generazione è quella della difficoltà e complessità di utilizzo, subito seguita dalle difficoltà di installazione. Questo problema rappresenta una grande sfida per il design di questi nuovi dispositivi, per cui è necessario ripensare alle modalità di realizzazione e progettazione. La Human–computer interaction (HCI) è l'ambito di studio che si occupa di questo tipo di problemi, ed in particolare di come avviene l'interazione tra l'utente e la macchina e di come esso vi si interfaccia.

2.1.1 Cenni sul design delle interfacce

Affinché questi dispositivi diventino effettivamente parte integrante della vita delle persone è necessario che essi siano desiderabili da parte dell'utente finale. Per soddisfare tale condizione è necessario pensare e progettare interfacce comprensibili ed ergonomiche per l'utente, facili da utilizzare. Solo in questo modo l'esperienza utente sarà positiva e spingerà lo sviluppo nella direzione di queste nuove tecnologie. A questo riguardo è stato prodotto diverso materiale che mette in luce in vari modi come il miglior modo per compiere una progettazione di questo tipo sia applicare quello che viene definito *Human-Centered design*, ossia *"l'approccio allo sviluppo di sistemi interattivi che punta a rendere tale sistema usabile e utile, concentrandosi sull'utente, sui sui bisogni e le sue necessità. (...) Questo approccio aumenta l'efficacia e l'efficienza, aumentando il benessere dell'uomo, la soddisfazione dell'utente nell'utilizzo, l'accessibilità e la sostenibilità"* [17]. Quello che si deve ricercare è quindi una progettazione che parta dalle necessità dell'utente stesso e dalle capacità dell'essere umano per poi sviluppare in base ad essi un prodotto in accordo con le necessità tecniche e di marketing. Quello che avviene normalmente nel processo di progettazione è l'esatto opposto, e questo produce oggetti spesso ricchi di funzioni, belli ed accattivanti, ma estremamente difficili da utilizzare. Per fare una buona progettazione di un dispositivo è necessario conoscere quindi oltre che la parte tecnica anche la parte umana, focalizzandosi sul modo di interazione e sulla comunicazione stessa che avviene tra la macchina e l'utente,

affinché essa risulti piacevole ed efficace.Perché ciò sia possibile è necessario che la macchina comunichi in maniera chiara e semplice con l'utente, in modo che esso riesca a capire in maniera immediata quali sono le operazioni possibili in quel momento e qual'è il comportamento che deve avere per ottenere un certo risultato. Queste sono le sfide che deve superare la progettazione di un dispositivo perché esso venga accolto positivamente da parte del mercato degli utenti.

2.2 Augmented Reality

Una delle tecnologie capaci di trarre maggiore vantaggio dai dispositivi Wearable ed Embedded è senza dubbio la Realtà Aumentata. In questo ampio ambito si pone il lavoro di esplorazione che verrà affrontato in questa tesi. Come prima cosa è necessario capire bene qual'è l'obiettivo di tale tecnologia e come essa differisca dalla più conosciuta Realtà Virtuale.

La Realtà Virtuale crea un mondo completamente digitale attorno all'utente, con il risultato di estraniarlo dalla realtà fisica che lo circonda. Lo scopo per cui è concepita la Realtà Aumentata (dall'inglese *Augmented Reality*) è profondamente diverso da questo; la Realtà Aumentata infatti si basa su un sistema che produce una serie di informazioni che vengono aggiunte a quella che è la normale percezione del mondo circostante da parte dell'utente attraverso i suoi sensi. L'utente continua quindi a condividere la realtà con altre persone, ma grazie ai dispositivi di cui è equipaggiato ha a disposizione informazioni aggiuntive rispetto a chi è sprovvisto di tali dispositivi. Essa ha trovato grande diffusione con l'avvento della tecnologia degli smartphone, uscendo dai laboratori e arrivando anche al grande pubblico, ma le prime applicazioni e documenti sull'argomento risalgono a ben prima che la tecnologia supportasse tale tipo di tecnologia. Il primo esempio di dispositivo per la realtà aumentata risale al 1968, ed è stato realizzato da Ivan Sutherland e Bob Sproull. Tale dispositivo aveva dimensioni tali da dover essere sostenuto da un braccio meccanico e permetteva la visualizzazione di una semplice stanza in wireframe. Da quel momento sono stati fatti moltissimi passi avanti e la tecnologia si è evoluta permettendo di applicare la Realtà Aumentata in moltissimi ambiti della società.

2.2.1 Realtà aumentata in campo commerciale

La Realtà aumentata rappresenta un grandissimo strumento per il marketing commerciale, e allo stesso tempo permette di fornire all'utente informazioni aggiuntive sul prodotto che sta per acquistare. Un esempio potrebbero essere

quelle applicazioni che mostrano il contenuto della confezione senza che essa venga effettivamente aperta. Un'azienda che applica questo tipo di approccio con successo sin dal 2009 è la Lego, che offre tramite apposite installazioni nei negozi la possibilità, semplicemente inquadrando la confezione, di visualizzare il risultato montato del contenuto della confezione.

2.2.2 Realtà aumentata in architettura

Le possibilità offerte dalla realtà aumentata in campo architetturale sono svariate, e vanno dalla visualizzazione di edifici e monumenti all'interno di uno spazio prima che essi vengano effettivamente costruiti, fino all'arredamento stesso, per cui è possibile visualizzare mobili prima di procedere all'acquisto. Un esempio di quest'ultima funzionalità è fornito da una nota azienda di arredamento svedese, che ha reso disponibile un'applicazione per smartphone che permette appunto di inserire proiezioni degli articoli dei propri cataloghi all'interno dell'abitazione dell'utente per visualizzarne l'effetto. É possibile vedere un esempio di utilizzo di questa applicazione nel video *"Place IKEA furniture in your home with augmented reality"*[21].

2.2.3 Realtà aumentata in campo medico

É in campo medico che la Realtà Aumentata mostra tutto il suo potenziale. I benefici portati da tale tipo di tecnologia sono innegabili, e permettono ad esempio di visualizzare direttamente sul corpo del paziente modelli tridimensionali di strutture anatomiche costruiti a partire da immagini radiologiche del paziente, oppure la presentazione immediata di qualsiasi informazione utile sullo stato del paziente.

2.2.4 Realtà aumentata nel tempo libero

Sono già disponibili applicazioni che semplicemente inquadrando una parte della città che abbiamo di fronte sono in grado di fornire una serie di indicazioni su ristoranti, monumenti e luoghi di interesse dell'ambiente che ci circonda. Un esempio di tali applicazioni potrebbe essere Etips, che si può vedere in azione nella città di Firenze nel video *""* [22].

Quelli visti finora sono soltanto alcuni degli innumerevoli campi di applicazione in cui trova spazio la Realtà Aumentata, che può veramente entrare nella vita di tutti i giorni degli utenti, così come è stato per le App per smartphone negli ultimi anni.

2.3 Head Mounted Display

La maggior parte delle applicazioni viste nel esempi sopra si basa su smartphone, ma come già detto questo rappresenta un grosso limite all'utilizzo di tale tecnologia. Immaginando tuttavia l'utilizzo con dispositivi diversi, che rendono l'interazione con il sistema veloce e non invasivo per le attività dell'utente lo scenario cambia notevolmente. A questo proposito è già da molti anni che vengono studiati e realizzati dispositivi montati direttamente sulla testa dell'utente. Tali dispositivi rientrano nella categoria degli HMD, Head-Mounted Display.

2.3.1 La visione umana

Questi dispositivi funzionano aggiungendo informazioni alla visione umana. La visione rappresenta il più affidabile e il più complesso dei nostri sensi, e fornisce più del 70% delle informazioni sensoriali raccolte dal nostro corpo. Le informazioni visuali sono raccolte dagli occhi sotto forma di onde luminose che entrano attraverso la pupilla all'interno del bulbo oculare, e vengono convertiti dalle cellule della retina, che si trova sul fondo dell'occhio, in impulsi nervosi, trasportati dal nervo ottico al cervello.

2.3.2 Caratteristiche e requisiti hardware degli HMD

Esistono molteplici modi per realizzare un HMD, e l'effetto ottenuto può variare moltissimo in base alle scelte costruttive del dispositivo stesso. Innanzitutto è possibile ottenere l'effetto desiderato, ossia quello di avere un oggetto virtuale visualizzato all'interno della realtà, in due maniere diverse:

- **Con approccio Video:** In questo caso la realtà viene catturata da una fotocamera e l'immagine viene combinata elettronicamente con l'immagine virtuale. Soltanto il risultato viene presentato all'utente. Questo metodo permette l'applicazione di tecniche di Image-Processing prima di visualizzare il risultato stesso.

- **Con approccio Ottico:** In questo modo l'immagine virtuale viene calcolata e sovrapposta all'immagine direttamente proveniente dal mondo reale tramite metodi ottici. Questo secondo metodo è di gran lunga quello più utilizzato e diffuso.

Al di la del metodo di visione scelto tutti gli HMD devono ingrandire un'immagine di dimensioni limitate per permettere di avere uno schermo virtuale di grandi dimensioni visualizzato ad una certa distanza, in modo da riempire

il più possibile il campo visivo dell'utente. Per ottenere un effetto di questo genere è necessario che la distanza occhio-lente sia la minore possibile. Non è tuttavia possibile ridurre tale distanza sotto una certa soglia senza perdere parte del campo visivo. Di solito si utilizza una distanza compresa tra i 20 e i 40 mm. Questa distanza prende il nome di *eye-relief*.

Un'ulteriore aspetto hardware che può modificare la visualizzazione risultante dello schermo virtuale è il tip e la forma dell'ottica del dispositivo. Molti dei moderni HMD funzionano tramite un canale riflettente che trasmette l'immagine virtuale generata da un sistema sulla montatura fino ad una superficie semitrasparente che si trova davanti all'occhio. In questa superficie si ottiene in questo modo la sovrapposizione dell'immagine virtuale con la realtà che l'utente ha davanti agli occhi.

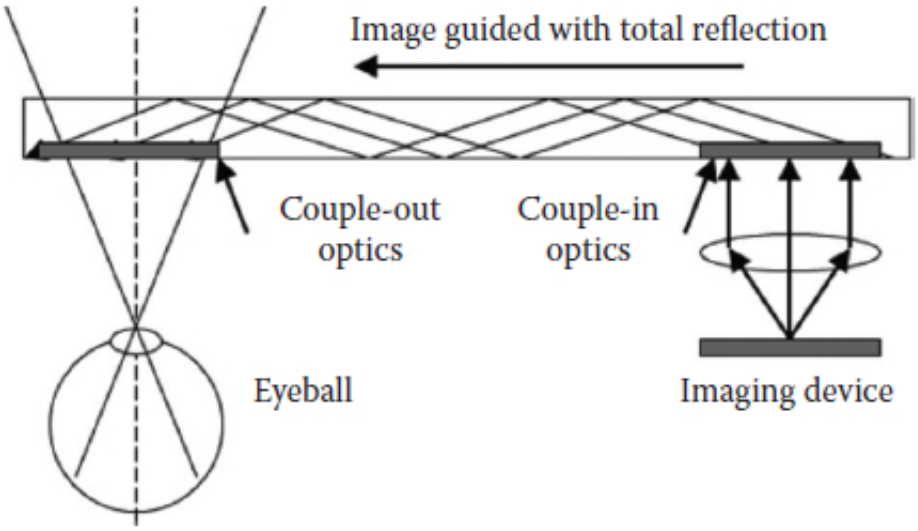

Figura 2.1: Funzionamento dell'ottica

I dispositivi che utilizzano un design ottico di questo tipo risultano compatti, leggeri e con un ampio campo visivo risultante. Sia i Google Glass che gli Epson Moverio funzionano in questo modo.

2.3.3 Caratteristiche interne degli HMD

Ogni HMD presenta una serie di caratteristiche interne che differenzia un modello dall'altro. Vediamo quali sono i parametri di valutazione più importanti e il motivo per cui essi sono parametri importanti da valutare:

- **Risoluzione**
- **Campo Visivo**
- **Occlusione**
- **Profondità**

La Risoluzione: indica il grado di fedeltà di un'immagine. Essa dipende sia dagli elementi ottici sia dai dispositivi che producono l'immagine virtuale. Un HMD ideale dovrebbe avere una risoluzione pari a quella dell'occhio umano, che è circa 12000 x 7200 pixel, un grado di definizione che non è ancora possibile ottenere con la tecnologia a disposizione.

Il campo visivo: rappresenta la superficie visualizzata. Per avere la visione stereo tipica di un ambiente tridimensionale è necessario che le immagini monoculari dei singoli occhi si sovrappongano in una certa regione per una percentuale che deve essere superiore al 50%. É importante controllare tale effetto poiché in un HMD soltanto una parte della regione del campo visivo presenta l'immagine virtuale sovrapposta a quella reale.

L'occlusione: rappresenta un parametro fondamentale per la percezione della profondità. La profondità di un oggetto reale viene valutata in base a quale oggetto sta davanti e copre quelli dietro. É fondamentale in un HMD che gli oggetti virtuali abbiano lo stesso comportamento degli oggetti reali per quanto riguarda la gestione della profondità. Se questo comportamento non è rispettato il risultato sarà una confusione dell'utente stesso.

La gestione della profondità: Gli oggetti virtuali vengono disegnati dal sistema a distanze fisse, mentre la distanza dell'utente dagli gli oggetti reali varia nel tempo. A causa di questo fenomeno è impossibile mettere a fuoco contemporaneamente un oggetto virtuale ed uno reale con un semplice HMD, ed è necessario utilizzare strutture ottiche complesse per risolvere questo inconveniente.

2.3.4 Requisiti necessari per l'utente

Oltre a questi parametri è necessario che gli HMD rispondano a determinati requisiti per poter essere utilizzati con successo da un utente. Infatti l'uso improprio di questi dispositivi può causare disturbi fisici più o meno gravi, ed è assolutamente necessario che tali dispositivi vengano progettati per ridurre

al minimo questi effetti collaterali. Per poter essere utilizzati per lunghi tempi i dispositivi devono essere leggeri, piccoli e non ingombranti, e il più possibile comodi da indossare. Il centro di massa del dispositivo deve essere il più possibile vicino alla testa dell'utente, e deve essere il più possibile bilanciato. Inoltre il dispositivo dovrà essere bello da vedere e da indossare per garantire una usabilità duratura e un'accettazione da parte della società.

Inoltre sarà necessario creare applicazioni che presentano un numero minimo di informazioni sullo schermo e solo per il tempo necessario, perché tali applicazioni distraggono l'utente dalla realtà stessa a causa delle immagini virtuali che vi sono proiettate sopra, causando effetti potenzialmente pericolosi causati dalla perdita di attenzione.

2.3.5 Gli Smart Glass come dispositivi per la Realtà Aumentata

Torniamo a parlare degli Smart Glass: Essi rappresentano l'ultima generazione di HMD, e ne mantengono le caratteristiche, e tra tutti i dispositivi Wearable disponibili sul mercato questi sono specificatamente progettati per supportare applicazioni di Realtà Virtuale e Realtà Aumentata. Dispositivi di questo tipo permettono forme di interazione nuove, quali possono essere i comandi vocali o il riconoscimento di gesti delle mani, liberando in questo modo l'utente dalla necessità dell'utilizzo delle mani in maniera esclusiva per interagirvi e permettendogli così di svolgere altre attività in contemporanea con l'utilizzo del dispositivo. Tuttavia in alcune occasioni il numero di informazioni presentate davanti agli occhi dell'utente può limitare questa libertà di svolgere altre attività ostruendo il campo visivo in contesti in cui si vorrebbe avere la visuale completamente libera. In questo caso l'unica soluzione è togliere i Glass o chiudere l'applicazione che era in esecuzione, perché anche muovendo la testa le informazioni rimangono solidali al dispositivo stesso e seguono la testa dell'utente. Da questa semplice considerazione nasce l'idea che viene sviluppata in questa tesi, ossia la possibilità di realizzare un'interfaccia utente diversa che risolva questo problema e renda l'utente veramente libero di svolgere altre attività durante l'utilizzo dei Glass.

Capitolo 3

Un'interfaccia utente diversa

In questo capitolo andremo a spiegare nel dettaglio l'idea che è stata sviluppata in questa tesi, presentandone i dettagli tecnici ma soprattutto il significato che abbiamo voluto dare a tale lavoro.

3.1 L'idea

L'idea di base che ha guidato questo lavoro è stata l'esplorazione sia in linea teorica che pratica di un possibile scenario di utilizzo di Smart Glass come strumento per ottenere un'interfaccia in cui l'utente vede oggetti virtuali contenenti informazioni di vario tipo, generati da un applicazione che gira sull'occhiale stesso, in trasparenza, ossia all'interno della realtà che lo circonda, il più possibile come facenti parte della realtà fisica. In particolare il lavoro si è focalizzato sulla progettazione e lo sviluppo di un prototipo di interfaccia che permettesse all'utente di muovere la testa liberamente (e di conseguenza di modificare la finestra di visione rispetto all'ambiente circostante) senza che gli elementi virtuali della realtà aumentata seguissero il suo movimento. In questo modo si ottengono elementi virtuali che non appartengono al sistema di riferimento degli occhiali, e che rimangono fermi nel punto dell'ambiente dove sono stati posizionati, quasi come se fossero veri oggetti fisici. Tale interfaccia permette quindi all'utente una maggiore libertà di movimento, e la possibilità di avere il campo visivo più libero, mantenendo allo stesso tempo una rapida accessibilità alle informazioni portate da questi oggetti, raggiungibili semplicemente ruotando la testa fino a portare il campo visivo nella loro direzione.

È da notare come non si tratti di una vera e propria applicazione di realtà aumentata, in quanto manca completamente il fattore di interazione con il mondo fisico che gli oggetti virtuali devono avere in tale tipo di applicazione. Ma allo stesso tempo va oltre la normale interfaccia che presentano di

default tali dispositivi, in cui l'utente ha sempre sotto gli occhi la schermata proposta dall'applicazione in quel momento con tutte le informazioni in essa contenute. Il lavoro che si è voluto sviluppare in questa tesi si pone in mezzo a queste differenti modalità di visualizzazione, andando in direzione della realtà aumentata, ma senza entrarvi nel merito.

3.1.1 Possibili Campi di Applicazione

Un'interfaccia di questo tipo apre l'utilizzo degli Smart Glass ad un numero molto grande di possibili applicazioni reali, sia in ambito lavorativo che nella vita di tutti i giorni. Essa permette infatti di aggiungere una componente aumentata alla realtà circostante senza che essa entri in modo prepotente nell'esperienza utente, che potrà a suo piacimento visualizzare o meno le informazioni aggiuntive semplicemente con una piccola rotazione della testa.

3.2 Caratteristiche e Requisiti dell'interfaccia

3.2.1 Caratteristiche

Il motivo per cui si è voluta esplorare la realtà di un'interfaccia di questo genere è che essa si avvicina maggiormente all'esperienza naturale che l'utilizzatore ha con gli oggetti del mondo fisico nelle sue normali interazioni di tutti i giorni. Quando ruotiamo la testa quello che ci aspettiamo infatti è che alcuni oggetti escano dal nostro campo visivo, mentre altri vi entrino. Questo invece non avviene nelle normali applicazioni di default che possono essere eseguite su Smart Glass, in cui il campo visivo è sempre e comunque riempito dalla schermata di tali applicazioni e dagli elementi in esso disegnati. Per ottenere questo è necessario progettare un sistema in cui il sistema di riferimento per gli oggetti virtuali non è più legato al dispositivo che l'utente indossa, ma all'utente stesso.

L'interfaccia che si vuole realizzare deve essere quindi :

- *Naturale, semplice ed immediata da utilizzare*
- *Adattabile alle varie necessità*
- *Hands-Free*: Deve lasciare all'utente le mani libere per svolgere altre attività durante l'utilizzo del sistema, prevedendo altri metodi di interazione con esso che non comportino l'uso delle mani.

3.2.2 Requisiti

Vediamo ora quali requisiti presenta la realizzazione di un sistema di tale genere.

Un primo requisito chiave che un sistema di questo genere deve avere è la **stabilità degli oggetti virtuali**. Essi infatti dovranno risultare fermi allo sguardo dell'utente, anche per facilitare l'accesso ai contenuto che l'oggetto stesso presenta.

Inoltre il sistema dovrà sicuramente deve presentare **un alto grado di reattività al movimento**. Questo perché il punto chiave del funzionamento di questa interfaccia è la simulazione dell'immobilità degli oggetti virtuali. Perché essi risultino immobili all'interno dell'ambiente il sistema deve calcolare la loro posizione in relazione al movimento della testa dell'utente in maniera veloce e quasi immediata in risposta all'evento di rotazione. Questo infatti risulta necessario per evitare che l'effetto ottenuto sia quello di un oggetto che si muove lungo la direzione dello spostamento, seguendo lo sguardo dell'utente stesso, per poi spostarsi nuovamente (ed eventualmente uscire dal campo visivo) per ritornare alla posizione corretta una volta terminata l'elaborazione della sua posizione.

Un ultimo requisito importante che deve essere presente è la **libertà del campo visivo**. In questo caso infatti l'interfaccia stessa basa il suo funzionamento sul fatto di lasciare il campo visivo dell'utente libero da elementi virtuali durante l'utilizzo, a meno che egli non decida volontariamente di visualizzarli ruotando la testa fino a farli entrare nel suo campo visivo.

3.3 Il Framework

Il risultato finale dell'esplorazione di questa tipologia di sistema dovrà essere lo sviluppo di una prima versione di un framework che offra ad uno sviluppatore una serie di primitive per lo sviluppo di applicazioni che utilizzino l'interfaccia fin qui descritta per il loro funzionamento ed interfacciamento con l'utente, e l'utilizzo di tale framework per la realizzazione di un semplice prototipo sperimentale che abbia come scopo quello di testare direttamente su dispositivo se esso risponde ai requisiti di cui sopra.

Nella sua versione finale, che va oltre questo lavoro, tale framework dovrebbe offrire la possibilità di realizzare facilmente superfici e spazi virtuali modellabili a seconda delle esigenze dello sviluppatore stesso, permettendo di stabilirne forma e dimensioni, di posizionare facilmente contenuti di varia natura all'interno di essi, in modo che l'utente stesso possa interagirvi nel modo più immediato possibile secondo la filosofia espressa sopra.

Per questa prima versione si è deciso di implementare le seguenti funzionalità:

- Posizionamento di un oggetto virtuale all'interno del sistema di riferimento definito dal Faramework. Ciò significa poter:
  - Definire le coordinate.
  - Definire la dimensione
- Definizione delle informazioni portate da tale oggetto.

Capitolo 4

Progettazione del Framework

In questo capitolo verrà mostrato il modo in cui è stato possibile definire un'interfaccia con questo genere di caratteristiche, entrando nel dettaglio delle necessità che si sono dovute affrontare e di quale modo si è scelto per risolvere tali punti critici.

4.1 Studio preliminare del problema

Per realizzare un framework che metta in pratica l'interfaccia descritta nel capitolo precedente è necessario ideare un sistema in grado di funzionare su Smart Glass che permetta di generare all'interno del dispositivo uno spazio virtuale, con sistema di riferimento slegato dal dispositivo stesso, e una serie di oggetti virtuali di varie forme e dimensioni. É quindi necessario definire un sistema che fornisca una serie di primitive che permettano di generare un mondo virtuale e degli oggetti in esso contenuti.

É inoltre necessario che l'utente sia in grado di muovere liberamente il suo campo visivo all'interno di tale sistema per visualizzare a piacimento le informazioni contenute portate dagli oggetti virtuali. Per rendere possibile questo sarà necessario utilizzare una serie di sensori che attraverso i loro valori permettano di calcolare la rotazione della testa dell'utente nel mondo reale, in modo da permettere all'applicazione di posizionare gli elementi in relazione ad essa, per poter dare la possibilità al sistema di valutare se visualizzare o meno gli oggetti all'interno del campo visivo dell'utente (che per forza di cose rappresenta solo una parte limitata dell'intero spazio virtuale).

4.2 Scelte progettuali

4.2.1 Gestione del movimento della testa

La parte del mondo virtuale mostrata sullo schermo presente nei Glass in un dato momento rappresenta il campo visivo dell'utente stesso. L'interfaccia che vogliamo creare richiede che tale campo visivo si comporti in maniera il più possibile coerente con l'esperienza che l'utente ha quando non indossa tali dispositivi. Per ottenere un comportamento di questo genere è necessario che il sistema che implementa tale interfaccia risponda al movimento della testa dell'utente modificando il riquadro relativo al campo visivo mostrato, e non modificando la posizione degli oggetti stessi all'interno del loro mondo virtuale.

Nel framework che verrà realizzato sarà quindi necessario implementare un sistema in grado di modificare la posizione della camera virtuale che definisce la sezione dello spazio virtuale mostrata in un dato istante nello schermo dei dispositivi sui quali è in esecuzione l'applicazione in risposta ad una rotazione della testa.

Inoltre è interessante notare che non è necessario che sia gestito lo spostamento dell'utente all'interno del mondo fisico. Infatti il risultato che si vuole ottenere è quello di uno spazio virtuale che l'utente può portare con se e utilizzare a piacimento. Per cui l'unico movimento che il framework andrà a gestire sarà la rotazione della testa per il controllo del campo visivo.

La camera

La Camera rappresenta una astrazione che permette di vedere tutte le coordinate dello spazio virtuale in prospettiva come se fossero viste attraverso l'occhio di una videocamera. Il risultato dipende quindi dalla direzione in cui essa punta, dall'orientamento e dalla posizione della camera stessa. Per definire una camera sarà quindi sufficiente conoscere la posizione all'interno del sistema di riferimento dello spazio, un vettore che indica la direzione a cui guarda, un vettore che punta in alto e uno che punta a destra dalla camera, tali da definire l'orientamento. Quello che si crea in questo modo è un nuovo sistema di riferimento con tre assi perpendicolari (i vettori) centrato nel punto in cui è posizionata la camera. Tale sistema di riferimento andrà a coincidere con quello dei Glass, in modo da rendere coerenti gli spostamenti.

Come è possibile ottenere questo effetto ?

Esistono sensori in grado di fornirci il valore della rotazione di un dispositivo rispetto ad ognuno dei tre assi che definiscono il sistema di riferimento per tale dispositivo. Tali angoli prendono il nome di angoli di Eulero, e permettono di

descrivere l'orientamento di un corpo rigido all'interno di uno spazio euclideo a tre dimensioni. Per la precisione permettono di descrivere la posizione di un sistema di riferimento XYZ solidale con un corpo rigido attraverso una serie di rotazioni a partire da un sistema di riferimento fisso xyz. Questo sistema trova ampio utilizzo nel campo dell'aeronautica e della robotica. in questi contesti tali angoli prendono rispettivamente il nome di angoli di imbardata (Yaw), rollio (Roll) e beccheggio (Pitch).

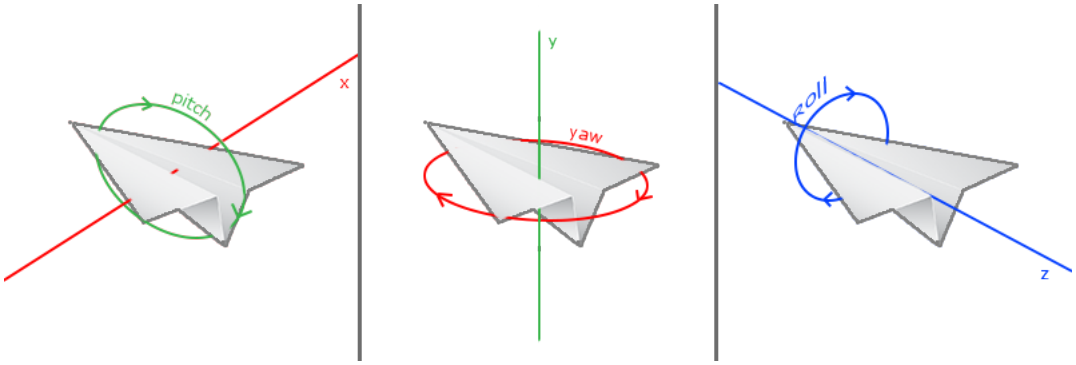

Figura 4.1: Angoli di Eulero

Conoscendo il valore di questi angoli è possibile ricavare facilmente tramite un procedimento matematico il valore delle componenti di un vettore che rappresenta la nuova direzione in cui la camera sta guardando. Per ottenere le componenti della rotazione effettuata rispetto ad ognuno degli assi è sufficiente ragionare nel seguente modo:

Nel mondo fisico la rotazione della testa dell'utente genera uno spostamento del suo campo visivo di una certa componente lungo le tre direzioni di riferimento. Per poter ottenere lo stesso effetto nel mondo virtuale bisognerà, conoscendo il valore degli angoli di rotazione rispetto agli assi, calcolare le relative componenti lungo il sistema di riferimento utilizzando tecniche di trigonometria elementare (sono necessarie solamente le funzioni seno e coseno). Per ogni angolo di rotazione, e quindi per ogni asse è possibile ricavare due componenti del vettore finale. Infatti risulta evidente che la componente relativa all'asse attorno alla quale avviene la rotazione è sempre 0, poiché in tale eventualità le coordinate dei punti non subiscono alcuna variazione rispetto a tale asse. Il prodotto di tali componenti permette di ottenere il vettore che indica il punto verso il quale sta guardando la camera dopo la rotazione.

4.2.2 Gli oggetti virtuali

Gli oggetti che saranno inseriti nello spazio virtuale dovranno essere il più possibile gestibili e personalizzabili da coloro che andranno ad utilizzare il framework che si sta sviluppando per l'implementazione di applicazioni che seguono il comportamento descritto dall'interfaccia utente che il framework presenta. Per poter ottenere oggetti di questo genere è stato deciso di ricorrere ad oggetti generati virtualmente tramite Computer-Grafica. Nonostante questo necessiti che lo sviluppatore abbia qualche base in questa materia per poter creare nuovi oggetti virtuali tale scelta è stata fatta per garantire una assoluta libertà nel tipo di oggetti gestibili con questo framework. Infatti esso si occuperà della gestione dello spazio virtuale e dei movimenti dell'utente, in maniera assolutamente scollegata dagli oggetti stessi che andrà a contenere, per i quali sarà sufficiente definire la forma e la posizione, oltre ovviamente alle informazioni che dovranno contenere.

4.3 Diagramma dei casi d'uso del framework

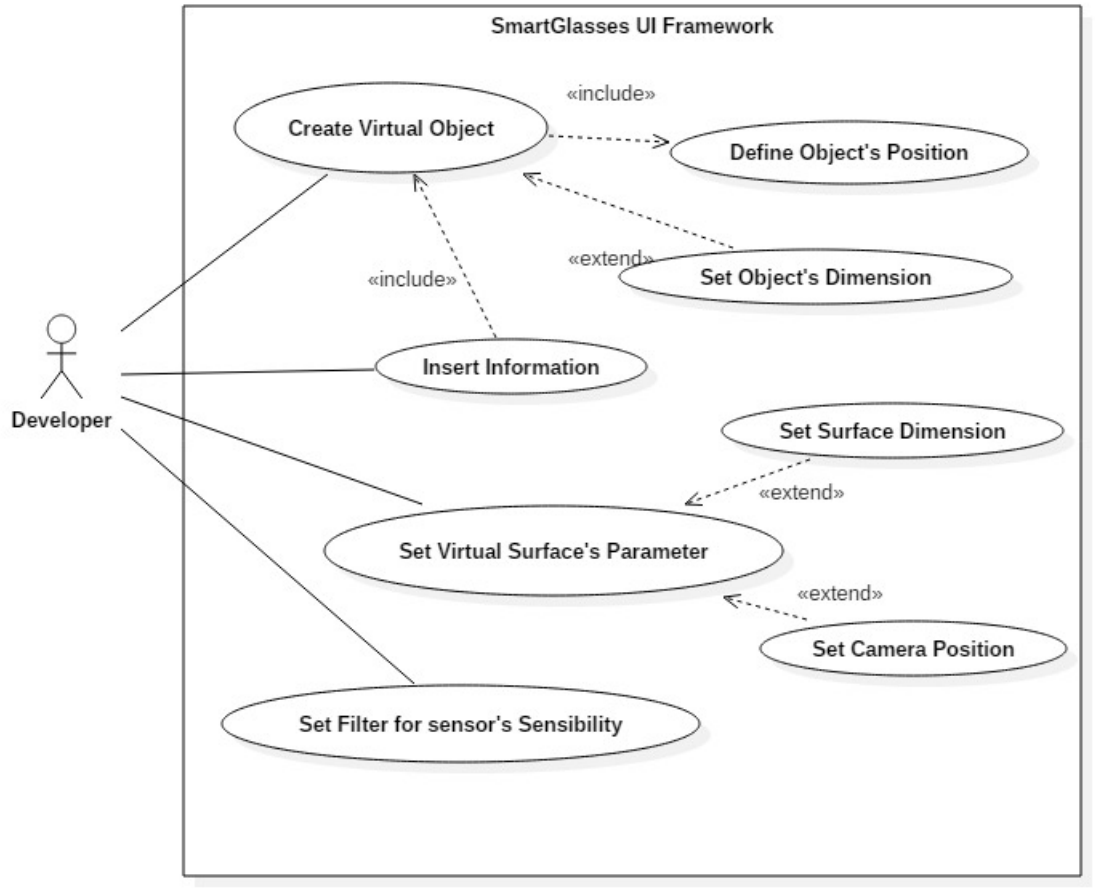

Figura 4.2: Diagramma dei casi d'uso

4.4 Scelte Implementative e Tecnologie abilitanti

Al momento della scrittura di questo elaborato erano a mia disposizione un paio di Epson Moverio BT-200 a rappresentanza dei dispositivi per cui il framework è pensato, e quindi è stato immediato scegliere tali dispositivi come base su cui testare e sviluppare il framework stesso. La scelta di tali dispositivi (già presentati nel capitolo 1) si è dimostrata ottimale poiché un gran numero di Smart Glass oggi esistenti montano, come quelli da noi scelti, un sistema operativo Android, per cui non solo la progettazione teorica rimarrà

valida per differenti tipi di Smart Glass, ma anche l'implementazione pratica del framework sarà compatibile con un grande numero di dispositivi. A seguito di questa scelta è stato possibile decidere in maniera semplice quali strumenti utilizzare per l'implementazione del framework.

4.4.1 Android

Come già detto i Moverio BT-200 montano un sistema operativo Android, per cui il framework sarà sviluppato per tale sistema, in modo da fornire la possibilità di testare il suo funzionamento direttamente sul dispositivo.

Il Sistema Android

Il sistema Android è un sistema operativo basato su Linux Kernel specificatamente pensato per dispositivi mobili, che fornisce agli sviluppatori un vero e proprio stack di strumenti e librerie per la realizzazione di applicazioni mobile. A partire dai primi test su dispositivi reali nel 2008, è oggi ampiamente diffuso in un grandissimo numero di dispositivi (84,7% del mercato dei dispositivi mobile nel terzo trimestre del 2015 secondo uno studio presentato da Gartner).

Tale sistema permette l'esecuzione di applicazioni scritte principalmente in linguaggio Java e basate sul pattern MVC. Per ogni applicazione si hanno infatti diverse View gestite ognuna da un Controller, che prende il nome di *Activity*, che permette anche di gestire gli eventi provenienti dall'utente e dal mondo fisico che circonda il dispositivo. L'utente può visualizzare ed interagire con una singola View per volta, per cui in ogni momento avrò una sola Activity in esecuzione, e durante l'utilizzo dell'applicazione ci sarà l'alternanza di varie Activity (e di conseguenza di varie View). Il ciclo di vita di questi oggetti è gestito in maniera completamente trasparente dal sistema stesso.

I sensori nei dispositivi Android

Quasi tutti i dispositivi che montano il sistema Android sono dotati di una serie di sensori per la misurazione di parametri provenienti dal mondo fisico, il cui valore può essere utilizzato all'interno delle applicazioni in svariati modi. All'interno dell'SDK di Android è presente una classe che aiuta gli sviluppatori nell'utilizzo di questi sensori, la classe *SensorManager*. Essa è una classe che fornisce un gran numero di metodi per accedere ai sensori ed elaborare i dati da essi ritornati.

Nel caso del framework in questione sarà necessario utilizzare due differenti sensori per ottenere informazioni il più possibile accurate riguardo il movimento della testa dell'utente. Saranno utilizzati un **accelerometro** e un **sensore di campo magnetico**.

L'accelerometro a tre assi è un sensore che permette di ottenere il valore dell'accelerazione applicata sul dispositivo. Questa informazione viene ricavata misurando la forza applicata al sensore stesso lungo ognuno dei tre assi(F_s), e ricavando l'accelerazione tramite la relazione:

$$A_d = -\frac{\sum_{k=1}^{3} F_s}{m}$$

dove m indica la massa del dispositivo.

In questo modo è possibile ricavare per ognuno dei tre assi ortogonali l'accelerazione (intesa come variazione di velocità rispetto al tempo) applicata al dispositivo in quella direzione, e stabilire quindi se il dispositivo stesso si sta muovendo o è fermo.

Il sensore di campo magnetico É un sensore che permette di misurare il valore del campo magnetico che circonda il dispositivo. Anche in questo caso i valori misurati sono in relazione ai tre assi che compongono il sistema di riferimento spaziale, e vengono misurati in microTesla. Si è dimostrato necessario utilizzare anche questo sensore perché con il solo accelerometro non sarebbe possibile stabilire se il dispositivo è effettivamente fermo o procede con un moto rettilineo uniforme lungo uno dei tre assi di riferimento. Infatti in questa ipotesi di spostamento la variazione di velocità è nulla, e l'accelerometro non rileverebbe alcun movimento, nonostante il dispositivo si stia effettivamente spostando.

4.4.2 Librerie OpenGL

Va al di la delle possibilità di questo lavoro creare un nuovo sistema per la gestione di elementi di computer grafica da utilizzare all'interno del framework stesso, ed inoltre tale operazione risulta superflua, in quanto esistono una serie di librerie che offrono le funzionalità necessarie per questo lavoro disponibili gratuitamente in rete, e già completamente integrate all'interno del sistema Android. Per cui verrà di seguito presentata una di queste librerie, che verrà utilizzata per tutte le applicazioni di computer grafica necessarie all'interno del framework.

Lo standard OpenGL

OpenGL (Open Graphics Library) è una libreria cross-platform che definisce delle specifiche standard per la realizzazione di applicazioni di grafica digitale sia in 2D che in 3D in modo real-time. Esso rappresenta uno degli

standard più diffusi ed utilizzati nell'ambito della grafica, e fornisce un gran numero di API che permettono di semplificare lo sviluppo di applicazioni di questo tipo semplificando la gestione di molteplici aspetti della visualizzazione, tra cui il rendering, il mapping delle texture e la gestione del movimento degli oggetti.

Pipeline del rendering grafico OpenGL

Ogni oggetto che viene disegnato tramite le funzioni di questa libreria per poter essere disegnato subisce una serie di operazioni che lo portano da una matrice di coordinate a diventare un oggetto virtuale mostrato nell'applicazione. Questo procedimento prende il nome di *Pipeline del Rendering*, e segue una serie di passi fissi per ogni oggetto. Vediamo ora a grandi linee come avviene il rendering di un oggetto:

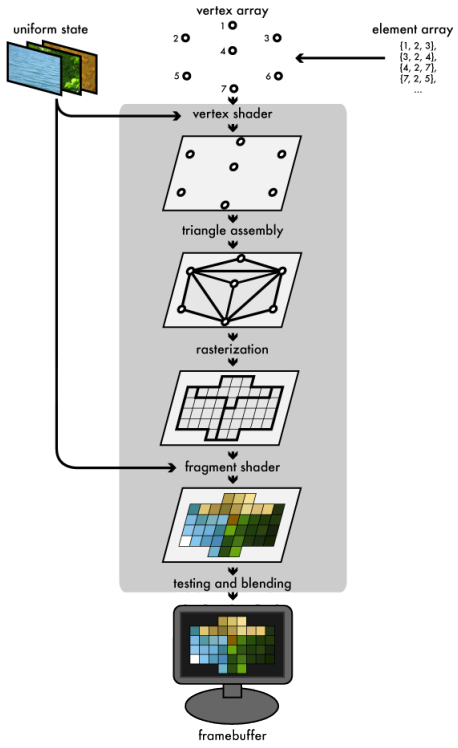

Figura 4.3: Pipeline del Rendering

Fase 1 - Definizione dei vertici:
Per ogni oggetto che deve essere disegnato l'applicazione genera:

- Un *array di Vertex*, ognuno dei quali porta una serie di informazioni quali posizione nello spazio 3D, texture e colore.

- Un *Element Array*, che è un array di indici che indica in quale ordine andranno uniti i vari Vertex.

Fase 2 - Elaborazione geometrica (Vertex Shader):
Ogni Vertex definito nella fase precedente subisce una trasformazione che converte i valori dei suoi attributi e modifica le coordinate dello spazio in 3 dimensioni in coordinate relative allo schermo di visualizzazione. Una volta effettuata questa procedura i vertici vengono uniti 3 a 3 in modo da formare una serie di triangoli che compongono la figura stessa. Durante questa procedura viene anche applicata la proiezione prospettica e il dimensionamento rispetto allo schermo, eliminando i punti esterni ai margini della finestra (*clipping*).

Fase 3 - Rasterizzazione:
Per ogni pixel viene effettuato un *sampling*, che genera un *Fragment* contenente informazioni su posizione, colore e profondità che quel pixel dovrà avere nella figura finale. In seguito a questo per ogni Fragment viene applicata la texture (se presente) e calcolato il colore definitivo che tale pixel dovrà avere tramite un processo di interpolazione. Prima di restituire il risultato vicino vengono effettuati dei test sui pixel vicini e un *test di profondità*, che serve a stabile se un pixel è visibile o coperto da un altro e quindi non visibile.

Una volta terminata questa operazione viene restituito il risultato finale.

OpenGL ES

OpenGL ES è un sottogruppo delle librerie OpenGL specificatamente definito per funzionare su dispositivi Embedded e mobile (per esempio su console, smartphone, e veicoli). Essa rappresenta un'interfaccia potente e flessibile tra lo strato software e l'accelerazione grafica.

In particolare il framework è stato realizzato usando la versione 1.0 di queste librerie. La scelta di utilizzare questa versione è stata effettuata per rendere più lineare e trasparente anche ad un utilizzatore meno esperto la gestione della parte di grafica necessaria all'interno del framework, per garantire una migliore comprensione del funzionamento. Sarebbestato possibile effettuare anche versioni successive, ma il procedimento risulta meno comprensibile ad un utilizzatore inesperto, ed inoltre richiede allo sviluppatore che vuole creare nuovi oggetti virtuali la conoscenza di alcuni elementi del linguaggio GLSL (OpenGL Shading Language) per la definizione degli Shader utilizzati nella fase di elaborazione geometrica.

La Camera in OpenGL

OpenGL non presenta la Camera come concetto, e permette solamente di ruotare e traslare gli oggetti virtuali contenuti nello spazio e di definire la superficie su cui avviene la proiezione della scena stessa; Questa superficie è l'approssimazione più simile al concetto di camera che viene offerto da OpenGL. Per ottenere l'effetto desiderato sarà necessario simularlo, definendo il sistema di riferimento relativo alla camera come spiegato precedentemente, e sarà inoltre necessario trovare la giusta impostazione per la finestra su cui avviene la proiezione della scena perché essa coincida con questo sistema di assi. In questo modo uno spostamento della camera coinciderà sempre con una variazione della posizione della finestra di visualizzazione stessa, che produrrà l'effetto desiderato.

4.5 Diagramma delle classi del Framework

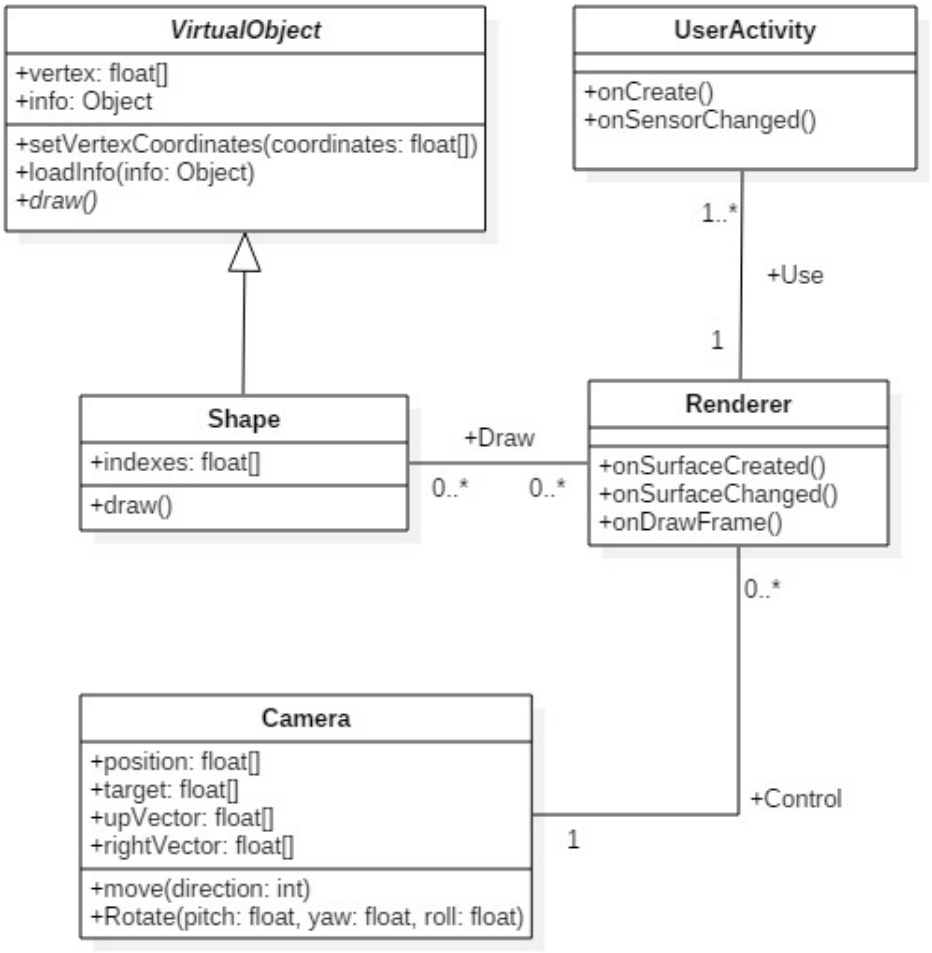

Figura 4.4: Diagramma delle classi del framework

Capitolo 5

Utilizzo del Framework

In questo ultimo capitolo andremo a vedere le varie fasi dell'implementazione del framework e dello sviluppo di una semplice applicazione di esempio che lo utilizza per funzionare

5.1 Implementazione delle classi del framework

Per prima cosa vediamo come sono state implementate le classi che formeranno la struttura generale del framework.

5.1.1 Gli oggetti Virtuali: VirtualObject e Shapes

La prime classi che andremo ad analizzare sono quello che offrono il supporto alla generazione di oggetti virtuali da inserire nello spazio virtuale generato dall'applicazione.

All'interno del framework è definita la classe *VirtualObject*, che è una classe astratta che incapsula al suo interno la maggior parte delle criticità legate al disegno di figure utilizzando le librerie OpenGL. In questo modo lo sviluppatore può realizzare figure personalizzate in maniera semplificata. Si è scelto di realizzare una classe astratta in modo da fornire supporto al maggior numero possibile di implementazioni, lasciando all'utente solo la necessità di definire il metodo astratto *draw()*, che indica in quale modo la figura dovrà essere disegnata.

Tale classe presenta inoltre il metodo *setVertexCoordinates()* che permette di impostare facilmente le coordinate che l'oggetto dovrà avere, semplicemente passando come parametro un array che le contiene. Questo metodo potrebbe sembrare banale, ma invece, come è possibile vedere dall'estratto di codice che mostra il metodo, esso automatizza la gestione dei buffer necessari ad OpenGL, che andrebbero altrimenti gestiti manualmente ogni volta.

```
public void setVertexCoordinates(float[] vertices) {
    ByteBuffer byteBuf =
        ByteBuffer.allocateDirect(vertices.length * FLOAT_SIZE);
    byteBuf.order(ByteOrder.nativeOrder());
    this.vertexBuffer = byteBuf.asFloatBuffer();
    this.vertexBuffer.put(vertices);
    this.vertexBuffer.position(0);
}
```

Listato 5.1: metodo *setVertexCoordinates* della classe *VirtualObject*

Per quanto riguarda il caricamento delle informazioni sull'oggetto si è deciso di utilizzare una delle feature offerte delle librerie OpenGL, l'uso delle *texture*. In questo modo è infatti possibile disegnare direttamente sulla figura qualunque immagine, ed è perciò sufficiente convertire in un'immagine bitmap qualunque informazione si voglia presentare all'utente finale e caricarla come texture. Anche in questo caso la classe *VirtualObject* viene in aiuto dello sviluppatore con due metodi:

1. Il metodo *setTextureCoordinates*, che analogamente al precedente prepara il buffer con le coordinate relative al posizionamento della texture sulla figura.

2. Il metodo *loadTexture*, che permette di caricare l'immagine passata come parametro e preparare la texture OpenGL. Un overload di questo metodo permette di caricare più immagini per avere texture differenti su facce diverse di una figura tridimensionale.

É importante che questi metodi vengano richiamati **sempre entrambi ed in questo ordine** per garantire il corretto funzionamento.

```
public void loadTexture(Bitmap textureBitmap, GL10 gl) {

    this.bitmap = textureBitmap;
    gl.glGenTextures(1, texturesID, 0);
    gl.glBindTexture(GL10.GL_TEXTURE_2D, texturesID[0]);
        // -- Define filter for different zoom level
    gl.glTexParameterf(GL10.GL_TEXTURE_2D,
        GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR);
    gl.glTexParameterf(GL10.GL_TEXTURE_2D,
        GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_NEAREST);
        // -- Define the behaviour when the dimension of the
            bitmap is smaller than the texture' surface
```

```
        gl.glTexParameterf(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_WRAP_S, GL10.GL_CLAMP_TO_EDGE);
        gl.glTexParameterf(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_WRAP_T, GL10.GL_CLAMP_TO_EDGE);
            // -- Load the bitmap
        if ( this.bitmap != null)
            GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
            //Clean up
        gl.glBindTexture(GL10.GL_TEXTURE_2D, 0);
    }

    public void loadTexture(Bitmap[] textureBitmap, GL10 gl) {

       int elem = textureBitmap.length;
       this.texturesID = new int[elem];
         gl.glGenTextures(elem, texturesID, 0);
         for (int i = 0; i < elem; i++) {
            this.bitmap = textureBitmap[i];
            gl.glBindTexture(GL10.GL_TEXTURE_2D, texturesID[i]);
            gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR);
            gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_NEAREST);
            gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                GL10.GL_TEXTURE_WRAP_S, GL10.GL_CLAMP_TO_EDGE);
            gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                GL10.GL_TEXTURE_WRAP_T, GL10.GL_CLAMP_TO_EDGE);
          if ( this.bitmap != null){
            GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
          }
          gl.glBindTexture(GL10.GL_TEXTURE_2D, 0);
            bitmap.recycle();
        }
    }
```

Listato 5.2: le due versioni del metodo *loadTexture*

Qualunque forma lo sviluppatore voglia implementare è sufficiente che:

1. Dichiari una classe che estende dalla classe astratta *VirtualObject*.
2. Inserisca un buffer contenente una lista di indici che indicano in quale ordine devono essere collegati i vertici
3. Implementi in tale classe il metodo *draw()*.

4. Richiami i metodi *setVertexCoordinates()*, *setTextureCoordinates* e *loadTexture* per completare l'inizializzazione.

All'interno del framework sono presenti anche alcune classi di esempio che implementano alcune forme base (classi *Triangle*, *Square*, *Cube* e *TextureCube*). Come esempio vediamo l'implementazione del metodo *draw()* della classe *TextureCube*, che rappresenta un esempio completo di come procedere all'implementazione di tale metodo:

```
public void draw(GL10 gl) {

    gl.glFrontFace(GL10.GL_CCW);
    gl.glEnable(GL10.GL_CULL_FACE); // Enable cull face
    gl.glCullFace(GL10.GL_BACK); // don't display the back face

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
    gl.glEnable(GL10.GL_TEXTURE_2D);

    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
    gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, textureBuffer);
    gl.glBindTexture(GL10.GL_TEXTURE_2D, texturesID[0]);

    for (int i=0; i<FACE; i++){
     indexBuffer.position(FACE * i);
       // -- Draw the vertices as triangles
       gl.glDrawElements(GL10.GL_TRIANGLES, 6,
           GL10.GL_UNSIGNED_BYTE, indexBuffer);
    }

    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
    gl.glDisable(GL10.GL_CULL_FACE);
}
```

Listato 5.3: metodo *draw()* della classe *TextureCube*

Dato che per disegnare anche un semplice cubo servono 24 vertici con tre coordinate per vertice il framework fornisce una classe di utilities, *CubeCoordsUtilities*, che presenta due metodi per ricevere in ritorno le coordinate delle texture e le coordinate dei vertici di un cubo. In particolare è possibile specificare la posizione e la dimensione del cubo desiderato come parametri del metodo *getVertices()* di tale classe per ottenere le coordinate adeguatamente scalate e traslate.

```
public class CubeCoordsUtilities {

    // The vertex for a cube with an edge of 2 unit, with the center
        in the origin (0,0,0).
    // This is the default cube.
    private float baseVertices[] = {
            // face 0
            -1.0f, -1.0f, 1.0f, //Vertex 0
            1.0f, -1.0f, 1.0f, //v1
            -1.0f, 1.0f, 1.0f, //v2
            1.0f, 1.0f, 1.0f, //v3
          ...
    };

    // The texture coordinates are relative to the dimension of the
        surface.
    private float texture[] = {
            //Mapping coordinates for the vertices
            0.0f, 1.0f,
            1.0f, 1.0f,
            0.0f, 0.0f,
            1.0f, 0.0f,
          ...
   };

    public float[] getBaseVertices() {
        return baseVertices;
    }

    // Return the coordinates for a cube, translating and scaling
        the default cube.
    public float[] getVertices(float transX, float transY, float
        transZ, float scaleFactor) {

        float[] vertices = baseVertices.clone();

        if ( scaleFactor != 0)
            for (int i = 0; i < vertices.length; i++) {
                vertices[i] *= scaleFactor;
            }
        for (int i = 0; i < baseVertices.length; i++){
            // X component
            if (i % 3 == 0)
```

```
                vertices[i] -= transX;
            // Y component
            else if (i % 3 == 1)
                vertices[i] += transY;
            // Z component
            else if (i % 3 == 2)
                vertices[i] += transZ;
        }
        return vertices;
    }

    public float[] getTextureCoords() {
        return texture;
    }
}
```

Listato 5.4: classe *CubeCoordsUtilities*

5.1.2 Camera

La camera come già detto implementa un sistema di assi ortogonali centrati in una posizione relativa da cui viene generato il punto di vista che viene mostrato all'utente. Tale classe è fondamentale per simulare lo spostamento dell'utente all'interno dello spazio virtuale. Il metodo fondamentale di tale classe è il metodo *setRotation()*, che permette di trasformare gli angoli di rotazione rispetto agli assi in uno spostamento del campo visivo della camera. É infatti interessante notare come non venga modificata la posizione della camera, ma soltanto la direzione in cui essa, e di conseguenza l'utente, guarda. Questa direzione è rappresentata dal vettore *cameraFront*.

La rotazione laterale della testa, che nel sistema di riferimento del sistema coincide con una rotazione rispetto all'asse z viene elaborato diversamente, e non va a comporre il vettore *cameraFront*. Il risultato di tale rotazione è infatti un inclinazione della camera a destra o a sinistra, con conseguente rotazione del campo visivo. Questa particolare posizione della camera è descritta esattamente dal vettore *Up*, che sarà poi richiesto come parametro per impostare la finestra dell'applicazione. Per questo motivo la rotazione rispetto all'asse z viene mappata direttamente in questo vettore.

5.1.3 CameraRenderer

La classe *CameraRenderer* è l'implementazione dell'interfaccia *GLSurfaceView.Renderer*, richiesta da OpenGL per definire lo spazio virtuale e gli oggetti

da disegnare. Tale interfaccia richiede di implementare tre metodi che verranno richiamati automaticamente dal thread che esegue le operazioni OpenGL. Tali metodi sono:

- *onSurfaceCreated()*
- *onSurfaceChanged()*
- *onDrawFrame()*

il metodo *onSurfaceCreated()*

Il metodo *onSurfaceCreated()* viene richiamato una sola volta al momento della creazione della superficie virtuale e viene quindi utilizzato per impostare alcuni parametri dell'ambiente che non saranno più modificati e per inizializzare alcuni campi della classe.

```
@Override
 public void onSurfaceCreated(GL10 gl, EGLConfig config) {
   // -- set the background color as black
     gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

   // -- Enable the depth test for drawing only the part of the
       shapes that are visible.
     gl.glClearDepthf(1.0f);
     gl.glEnable(GL10.GL_DEPTH_TEST);
     gl.glDepthFunc(GL10.GL_LEQUAL);

     this.cam = new Camera();
 }
```

Listato 5.5: il metodo *onSurfaceCreated()*

il metodo *onSurfaceChanged()*

Il metodo *onSurfaceChanged()* viene richiamato ogni volta che la superficie virtuale cambia in qualche modo. In questa funzione viene impostata la prospettiva che dovrà essere applicata agli oggetti e viene definito il volume di spazio che verrà visualizzato. Questo è possibile tramite la funzione *glFrustumf()* di OpenGL. Essa prende come parametri, in ordine, le coordinate sinistra, destra, sotto e sopra del nearPlane. Gli ultimi due parametri definiscono la distanza dalla camera rispettivamente del nearPlane e del farPlane. Il volume di spazio contenuto nel parallelepipedo ottenuto unendo i vertici di

questi due piani è tutto ciò che viene mostrato in un dato momento dalla camera. Se un oggetto si trova al di fuori di questo volume non viene mostrato. Il nearPlane rappresenta la superficie su cui viene proiettato il contenuto di tale volume.

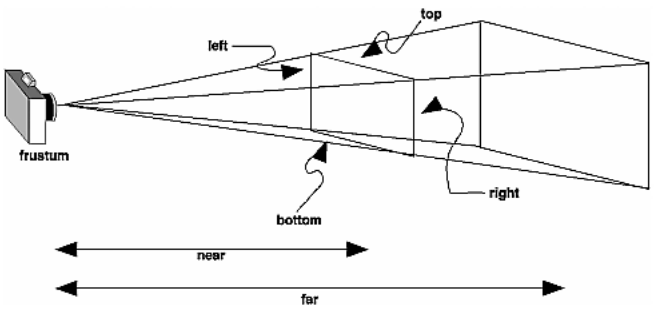

Figura 5.1: il View Frustum di OpenGL

Per ottenere una finestra di visualizzazione delle stesse dimensioni dello schermo del dispositivo di output si utilizza la funzione *glViewport()* passando come parametro le dimensioni dello schermo.

```
@Override
public void onSurfaceChanged(GL10 gl, int width, int height) {
    gl.glViewport(0, 0, width, height);
    float ratio = (float) width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glFrustumf(-ratio, ratio, -1, 1, this.nearPlanePos, 100);
}
```

Listato 5.6: il metodo *onSurfaceCreated()*

il metodo *onDrawFrame()*

Il metodo *onDrawFrame()* viene richiamato ogni volta che il sistema deve disegnare un frame. É in questo metodo che viene impostata ogni volta la posizione della camera relativamente alla posizione del dispositivo nel mondo fisico. Per fare questo è necessario richiamare la funzione *gluLookAt()* di OpenGL, che calcola in automatico tramite rotazioni e traslazioni la matrice da applicare allo spazio virtuale per descrivere la corretta posizione degli oggetti rispetto al punto di visualizzazione dell'utente. Tale funzione richiede come parametri tre vettori: *posizione*, *direzione* e *UpVector*, che descrivono rispettivamente la posizione nello spazio della camera, la direzione verso la

quale punta e l'inclinazione laterale della camera stessa. É inoltre interessante vedere come sia necessario calcolare il vettore *direzione* in relazione alla posizione stessa della camera per ottenere una visualizzazione che sia sempre perpendicolare agli oggetti stessi.

```
@Override
public void onDrawFrame(GL10 gl) {

   gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT);
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();

    float[] pos = cam.getPosition();
    float[] up = cam.getUpVector();
    float[] front = cam.getCameraFront();

    float posX = pos[0];
    float posY = pos[1];
    float posZ = pos[2];
    float tarX = pos[0] + front[0];
    float tarY = pos[1] + front[1];
    float tarZ = pos[2] + front[2];
    float upX = up[0];
    float upY = up[1];
    float upZ = up[2];

    GLU.gluLookAt(gl, posX, posY, posZ, tarX, tarY, tarZ, upX,
        upY, upZ);
}
```

Listato 5.7: il metodo *onSurfaceCreated()*

In questa classe questo metodo è in realtà implementato solo parzialmente, in quanto non specifica quali oggetti devono essere disegnati. Questo perché è stato scelto di mantenere questa classe il più possibile generale. Per poter ottenere degli oggetti virtuali è necessario che lo sviluppatore estenda questa classe nel suo progetto e finisca di implementare questo metodo facendone un override e aggiungendo il codice per il disegno degli oggetti. Questo rappresenta il metodo più veloce per l'applicazione del framework all'interno delle applicazioni. In questo modo questa classe risolve la maggior parte delle problematiche legate al funzionamento dell'interfaccia, mentre uno sviluppatore si deve preoccupare solamente della definizione degli oggetti virtuali che intende inserire all'interno dello spazio virtuale per ottenere l'effetto desiderato.

5.1.4 UserActivity

Essendo quella che si vuole realizzare un'applicazione Android è necessario definire all'interno del progetto una classe che estende da Activity, che funziona da Controller per la parte di View che verrà visualizzata a schermo. in tale classe verrà anche gestita la lettura dei valori dai sensori.

Nel metodo *OnCreate()*, richiamato in automatico dal sistema al momento della creazione della Activity sono settati tutti i parametri necessari all'utilizzo di OpenGL e viene inoltre specificato quale Renderer utilizzare per disegnare lo spazio virtuale. Lo sviluppatore dovrà semplicemente modificare questo assegnamento per poter utilizzare una propria classe per il rendering. In questa fase vengono inoltre inizializzati i sensori che verranno successivamente utilizzati per la lettura dei valori di posizione. É importante selezionare manualmente quali sensori utilizzare sui Moverio BT-200 poiché tali dispositivi presentano sensori sia sugli occhiali che nel dispositivo tascabile collegato. Questo è possibile usando un'istanza della classe *SensorControl* offerta dal SDK dedicato di tali dispositivi.

```
protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);

       // -- set the fullscreen mode on Moverio
       Window win = getWindow();
       WindowManager.LayoutParams winParams = win.getAttributes();
       winParams.flags |= 0x80000000;
       win.setAttributes(winParams);

       setRequestedOrientation
        (ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

       // ---- OpenGl setup
     setContentView(R.layout.glsurface_layout);
       surface = (GLSurfaceView) findViewById(R.id.surfaceView);
       surface.setEGLContextClientVersion(1);
       renderer = new MyRenderer(this);
       surface.setRenderer(renderer);

       // ---- Manage the sensors
     // -- tell to use the sensor on Moverio headset
       SensorControl sens = new SensorControl(this);
       sens.setMode(SensorControl.SENSOR_MODE_HEADSET);
     // -- initialize Android sensor
       this.sensManager = (SensorManager)
```

```
            getSystemService(SENSOR_SERVICE);
        sensManager.registerListener(this,
            sensManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
            SensorManager.SENSOR_DELAY_GAME);
        sensManager.registerListener(this,
            sensManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
            SensorManager.SENSOR_DELAY_GAME);

      accDiff = 0.00f;
        currentNormAcc = SensorManager.GRAVITY_EARTH;
        lastNormAcc = SensorManager.GRAVITY_EARTH;
    }
```

Listato 5.8: il metodo *onCreate()*

In questa classe assume particolare importanza anche il metodo *onSensorChanged()*, che viene richiamato dal sistema quando avviene una variazione nel valore letto dai sensori. In tale metodo avviene l'elaborazione dei dati presi dai sensori per ottenere i relativi valori di rotazione rispetto agli assi.

```
  public void onSensorChanged(SensorEvent event) {
       switch (event.sensor.getType()) {
           case Sensor.TYPE_MAGNETIC_FIELD:
               geomag[0] =
           geomag[0] * alpha + event.values[0] * (1 - alpha);
               geomag[1] =
           geomag[1] * alpha + event.values[1] * (1 - alpha);
               geomag[2] =
           geomag[2] * alpha + event.values[2] * (1 - alpha);
               break;
           case Sensor.TYPE_ACCELEROMETER:
               gravity[0] =
           gravity[0] * alpha + event.values[0] * (1 - alpha);
               gravity[1] =
           gravity[1] * alpha + event.values[1] * (1 - alpha);
               gravity[2] =
           gravity[2] * alpha + event.values[2] * (1 - alpha);
               linear_acceleration[0] = event.values[0] - gravity[0];
               linear_acceleration[1] = event.values[1] - gravity[1];
               linear_acceleration[2] = event.values[2] - gravity[2];
               break;
       }
      ...
```

Listato 5.9: il filtro applicato sui valori letti dai sensori

I sensori dei dispositivi Android hanno una sensibilità molto elevata, per cui per ottenere una lettura più stabile e affidabile è stato applicato un filtro ai valori letti dai sensori, come può essere visto nell'estratto di codice sopra. Inoltre per poter conoscere l'effettiva accelerazione subita dal dispositivo è stato necessario togliere dai valori letti dall'accelerometro il contributo dato dall'accelerazione di gravità. In questo modo vengono registrati solo accelerazioni dovute ad interazioni dell'utente con il sistema.

Una volta ottenuti i valori dai sensori è possibile ricavare da essi prima la matrice di rotazione relativa, dalla quale si può facilmente ottenere il vettore contenente gli angoli di rotazione relativi agli assi. Questo procedimento geometrico di operazioni sulle matrici è offerto in maniera trasparente dalla classe *SensorManager* di Android, che offre diverse funzionalità oltre ad occuparsi della gestione dei sensori. In particolare:

- la funzione *getRotationMatrix()* prende i valori di accelerazione e campo magnetico letti dai sensori e restituisce una matrice che rappresenta la matrice di rotazione del dispositivo. Con tale matrice è possibile ottenere le coordinate relative al mondo reale moltiplicando per tale matrice il vettore contenente le coordinate relative al dispositivo.

- la funzione *remapCoordinateSystem()* permette di convertire la matrice di rotazione ottenuta dalla funzione precedente ad una relativa ad un diverso sistema di riferimento. Questa operazione risulta necessaria poiché i Moverio funzionano in modalità Landscape e i valori relativi allo spostamento risultano su assi differenti. Tramite questa funzione è possibile tornare a riferirsi al sistema di riferimento standard di Android, visibile in figura, all'interno del codice. Il sistema convertirà infatti i valori letti al nuovo sistema di riferimento in automatico, mascherando il cambiamento.

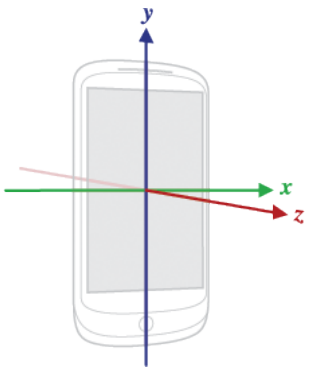

Figura 5.2: il sistema di riferimento di Android

- la funzione *getOrientation()* infine ricava dalla matrice di rotazione un vettore contenente gli angoli di orientamento rispetto agli assi. L'applicazione della funzione precedente rimappa l'intera matrice di rotazione, e quindi anche i valori degli angoli di Eulero restituiti da questa funzione risultano riferiti ad assi diversi da quelli che sono quelli di default del sistema Android. Gli angoli a cui si fa riferimento nel codice sono quelli relativi alla rotazione rispetto agli assi come in figura 5.3:

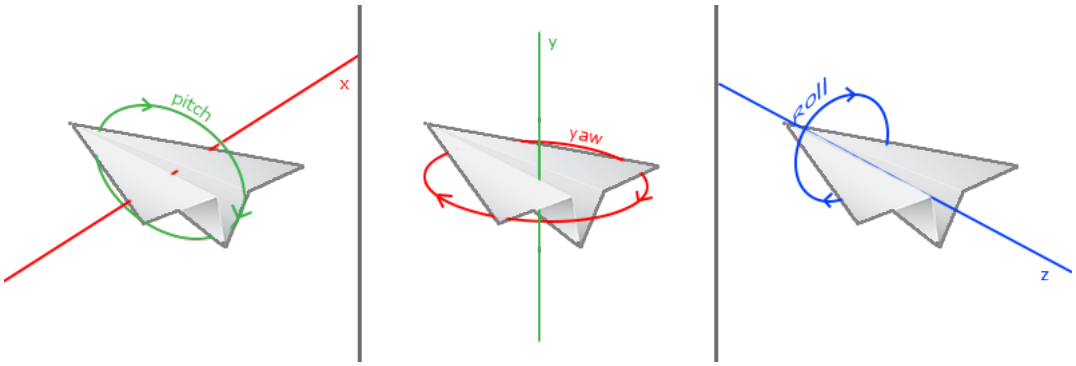

Figura 5.3: Angoli di Eulero

```
...

if (gravity != null && geomag != null) {
        // ---- detecting motion ----
        lastNormAcc = currentNormAcc;
        currentNormAcc = (float) Math.sqrt(linear_acceleration[0]
            * linear_acceleration[0] + linear_acceleration[1] *
            linear_acceleration[1] + linear_acceleration[2] *
            linear_acceleration[2]);
        float delta = currentNormAcc - lastNormAcc;
        accDiff = Math.abs(delta);
     // -- if the device is moved by a significant amount ( >
         threshold ) we need to calculate the traslation for the
         camera
        if (accDiff >= movementThreshold) {
            // -- calculate rotation and get the device
                orientation
            if (SensorManager.getRotationMatrix(rotMatrix, I,
                gravity, geomag)) {
```

```
                SensorManager.remapCoordinateSystem(rotMatrix,
                    SensorManager.AXIS_X, SensorManager.AXIS_Z,
                    remapMatrix);
                SensorManager.getOrientation(remapMatrix,
                    orientation);
                renderer.setOrientationValues(orientation);
            }
        }
        surface.requestRender();
    }
}
```

Listato 5.10: Il calcolo degli angoli di orientamento

5.2 Realizzazione del prototipo usando il framework

Le classi viste finora rappresentano il framework che si è sviluppato. Come ultimo passaggio verrà ora mostrato come è possibile applicare tale framework per la realizzazione di una applicazione che sfrutta l'interfaccia descritta in questo lavoro realizzando un esempio di applicazione.

Per realizzare una applicazione personalizzata è sufficiente seguire i seguenti passi:

- Creazione dei propri oggetti virtuali
- Creazione di una propria implementazione del Renderer
- Impostazione del Renderer nell'Activity.

5.2.1 Definizione degli oggetti virtuali

Per definire un oggetto virtuale personalizzato è sufficiente creare una classe che estende la classe astratta *VirtualObject* presente nel framework e ne implementa il metodo *draw()*. All'interno del package *shapes* del framework stesso sono presenti alcuni oggetti di esempio da cui è possibile prendere spunto. Come esempio vediamo la classe *TextureCube*, che permette di disegnare nel sistema un cubo delle texture sulle sue facce.

```
public class TextureCube extends VirtualObject{
    private static final int FACE = 6;
    private byte indices[] = {
            //Faces definition
            0,1,3, 0,3,2,         //Face front
            4,5,7, 4,7,6,         //Face right
            8,9,11, 8,11,10,      //...
            12,13,15, 12,15,14,
            16,17,19, 16,19,18,
            20,21,23, 20,23,22,
    };

    public TextureCube(){
        this.setIndices(indices);
    }

    public void draw(GL10 gl) {
        gl.glFrontFace(GL10.GL_CCW);
        gl.glEnable(GL10.GL_CULL_FACE);
        gl.glCullFace(GL10.GL_BACK);

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
        gl.glEnable(GL10.GL_TEXTURE_2D);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, this.vertexBuffer);
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, this.textureBuffer);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, this.texturesID[0]);

        for (int i=0;i<FACE;i++){
            indexBuffer.position(FACE*i);
            //Draw the vertices as triangles
            gl.glDrawElements(GL10.GL_TRIANGLES, 6,
                GL10.GL_UNSIGNED_BYTE, indexBuffer);
        }

        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
        gl.glDisable(GL10.GL_CULL_FACE);
    }
}
```

Listato 5.11: la classe *TextureCube*

Le immagini usate per le texture sono delle semplici immagini Bitmap caricate dalle resources dell'applicazione. Tuttavia su alcuni dispositivi, compresi i Moverio, **è necessario che le dimensioni di tali immagini siano esattamente delle potenze di 2** affinchè esse possano essere utilizzate con successo. Se questa condizione non fosse verificata si ottiene un oggetto completamente bianco.

5.2.2 Implementazione del Renderer

Il passo successivo è quello di creare una classe che estenda dalla classe *CameraRenderer* del framework. in questo modo è possibile ottenere un renderer grafico per far disegnare all'ambiente OpenGL gli oggetti personalizzati che sono stati definiti in precedenza senza entrare nel dettaglio della gestione dell'ambiente. É infatti sufficiente inizializzare gli oggetti virtuali all'interno di un override del metodo *onSurfaceCreated()* e richiamare il relativo metodo *draw()* all'interno di un override del metodo *onDrawFrame()*. É estremamente importante ricordare di chiamare il relativo metodo della superclasse quando viene effettuato l'override, poichè in tali metodi è gestito l'ambiente grafico e il comportamento che deve avere la camera.

```
public class MyRenderer extends CameraRenderer {

    private TextureCube cube;
    private TextureCube cube2;
    private TextureCube cube3;
    private TextureCube cube4;
    private Bitmap textureBitmap;
    private CubeCoordsUtilities coord = new CubeCoordsUtilities();

    public MyRenderer(Activity a) {
        super(a);
    }

    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        super.onSurfaceCreated(gl, config);

        this.textureBitmap = BitmapFactory.decodeResource
      (this.currentActivity.getResources(), R.drawable.texture_3);
        this.cube = new TextureCube();
        this.cube.setVertexCoordinates(this.coord.getBaseVertices());
        this.cube.
      setTextureCoordinates(this.coord.getTextureCoords());
```

```
        this.cube.loadTexture(this.textureBitmap, gl);

        this.textureBitmap = BitmapFactory.decodeResource
      (this.currentActivity.getResources(), R.drawable.texture_1);
        this.cube2 = new TextureCube();
        this.cube2.setVertexCoordinates(this.coord.getVertices(5, 0,
            0, 2));
        this.cube2.
      setTextureCoordinates(this.coord.getTextureCoords());
        this.cube2.loadTexture(this.textureBitmap, gl);

        this.textureBitmap = BitmapFactory.decodeResource
      (this.currentActivity.getResources(), R.drawable.texture_2);
        this.cube3 = new TextureCube();
        this.cube3.setVertexCoordinates(this.coord.getVertices(0, 5,
            0, 0.5f));
        this.cube3.
      setTextureCoordinates(this.coord.getTextureCoords());
        this.cube3.loadTexture(this.textureBitmap, gl);

        this.textureBitmap = BitmapFactory.decodeResource
      (this.currentActivity.getResources(), R.drawable.texture_4);
        this.cube4 = new TextureCube();
        this.cube4.setVertexCoordinates(this.coord.getVertices(0, 0,
            -15, 0));
        this.cube4.
      setTextureCoordinates(this.coord.getTextureCoords());
        this.cube4.loadTexture(this.textureBitmap, gl);
    }

    @Override
    public void onDrawFrame(GL10 gl) {
        super.onDrawFrame(gl);
        this.cube.draw(gl);
        this.cube2.draw(gl);
        this.cube3.draw(gl);
        this.cube4.draw(gl);
    }
}
```

Listato 5.12: esempio di estensione della classe *CameraRenderer*

Tale esempio nello specifico definisce quattro cubi di dimensioni differenti con differenti texture posizionate lungo i tre assi e nell'origine in punti diffe-

renti. É fondamentale che gli oggetti virtuali vengano inizializzati all'interno di una delle funzioni di cui si fa l'override, perché tali funzioni sono gestite da particolari thread in grado di eseguire funzioni OpenGL, che altrimenti non potrebbero essere eseguite, e causerebbero un crash dell'applicazione stessa.

Impostazione del Renderer nell'Activity

Per completare l'implementazione dell'applicazione è sufficiente instanziare un oggetto della classe relativa al renderer all'interno della classe *SensorMovingActivity* e associarlo alla variabile *renderer* presente in tale classe.

5.3 Test dell'interfaccia

Il risultato ottenuto è una semplice applicazione funzionante sugli Epson Moverio e che mostra alcune delle funzionalità chiave offerte dal framework. Essa presenta semplicemente uno spazio in cui sono inseriti 4 cubi virtuali di diverse dimensioni e in posizioni diverse all'interno dell'ambiente, ognuno dei quali presenta una diversa texture sulla sua superficie. Essi presentano in questo modo informazioni differenti, di natura sia visuale che testuale, in modo che fosse possibile testare il comportamento del sistema in un caso di applicazione reale. Con questo primo e semplice prototipo sono stati effettuati alcuni test per impostare al meglio alcuni parametri del framework stesso e rendere l'utilizzo da parte di uno sviluppatore ancora più immediato.

Verranno ora presentati i test effettuati sul prototipo e il risultato ai quali essi hanno condotto.

5.3.1 Test della distanza

Il primo test effettuato è stato quello che ha permesso di stabilire una distanza adeguata della camera rispetto all'origine degli assi del sistema OpenGL per ottenere una visualizzazioni ottimale delle delle informazioni testuali e/o visuali presenti all'interno dell'applicazione. Ovviamente tale distanza dipende moltissimo dalla posizione degli oggetti all'interno del sistema, per cui il test è stato effettuato basandosi su oggetti posizionati nell'origine. Impostando per la camera una posizione troppo lontana o troppo vicina non sarebbe possibile visualizzare correttamente le informazioni presentate dagli oggetti, rendendo di fatto inutile l'interfaccia e l'applicazione stessa. Inoltre è possibile impostare anche la distanza relativa della camera rispetto al piano di proiezione degli oggetti (nearPlan). Questo produce una visualizzazione differente della prospettiva degli oggetti stessi.

In seguito ai test si è visto come per l'applicazione di esempio una distanza di tre unità dal piano di proiezione e una posizione della camera a -8 unità lungo l'asse Z del sistema di riferimento assoluto dello spazio permetta di ottenere una visualizzazione ottimale. Questi valori sono stati ottenuti tramite test successivi e si ritengono validi soltanto per l'applicazione corrente, in quanto può essere necessario posizionare la camera in punti diversi a seconda della necessità dell'applicazione. É lasciata quindi allo sviluppatore la possibilità e l'incombenza di impostare questi parametri a seconda del posizionamento degli oggetti nella propria applicazione, assegnando un valore adeguato al campo *nearPlanePos* della classe *CameraRenderer* per posizionare il piano di proiezione e impostando il parametro *distance* nella classe *Camera* per modificare la posizione della camera. Si noti che tale parametro dovrebbe avere un valore minore di zero, a causa dell'orientamento dell'asse Z, che è entrante al dispositivo.

5.3.2 Test di stabilità

Il successivo test effettuato è stato quello relativo alla stabilità degli oggetti disegnati. A causa della elevata sensibilità dei sensori anche appoggiando i Glass ad una superficie ferma come un tavolo gli oggetti non risultano perfettamente fermi, ma anzi continuano a vibrare leggermente, spostandosi a causa di piccolissime variazioni nei valori letti. La soluzione a questo problema è stata l'applicazione di un filtro che escludesse queste letture accidentali, provocando la generazione di un nuovo frame soltanto quando la variazione dei valori superasse una certa soglia minima. É possibile impostare questo valore tramite il parametro *movementThreshold* all'interno della classe *SensorMovingActivity*. Tramite una serie di prove successive si è visto che per limitare l'effetto di queste letture accidentali e far sembrare gli oggetti effettivamente immobili il valore di tale parametro deve essere maggiore di 0,1. Tale limite tuttavia provoca una sensibile perdita di reattività, ed inoltre provoca una assoluta mancanza di fluidità nella visualizzazione degli oggetti, che si spostano ora a scatti. Visto che tale soluzione produce un effetto visivo di gran lunga peggiore del problema che ha portato al suo inserimento, nella versione di esempio la soglia è stata impostata a 0, eliminando di fatto il filtro, che si è tuttavia deciso di mantenere per permettere ad uno sviluppatore che ne avesse la necessità di utilizzarlo facilmente.

5.3.3 Test di reattività

Un ulteriore test effettuato riguarda la reattività dei sensori al movimento dell'utente. Questo test è quello che ha rilevato i maggiori problemi nell'appli-

cazione. Infatti il risultato finale non riesce a mantenere gli oggetti perfettamente fermi all'interno del campo visivo in seguito ad una rotazione della testa, e gli oggetti tendono a seguire leggermente il movimento dell'utente stesso.

Inoltre Android offre la possibilità di impostare diverse frequenze di campionamento, e questo test è stato effettuato per valutare quale di queste frequenze fosse ottimale per ottenere la maggior fluidità possibile nel movimento della visuale rispetto agli oggetti. In seguito a questo test è stato impostato il delay dei sensori al valore *SENSOR DELAY GAME*, che corrisponde ad un ritardo di circa 20000 microsecondi tra una lettura e la successiva. Questo valore può essere aumentato o diminuito nella classe dell'Activity. Più tale frequenza sarà elevata più sarà alto il consumo di risorse e di batteria, per cui è meglio non andare oltre la reale necessità dell'applicazione che si sta sviluppando.

5.3.4 Test delle letture inaffidabili

Un ultimo test effettuato è quello relativo alle letture inaffidabile da parte dei sensori. Infatti è stato possibile notare che in alcune occasioni quando l'applicazione è eseguita sui Glass i sensori restituiscono dei valori che rientrano nella categoria *SENSOR STATE UNRELIABLE*. Per controllare tale fenomeno sono stati inseriti due contatori, il primo dei quali tieni il conto delle letture totali effettuate dai sensori, mentre il secondo soltanto di quelle che rientrano in questa categoria. A seguito di numerosi test si è visto come il numero di queste letture vari moltissimo (da 0 a 100%) a seconda dell'esecuzione, mentre durante una singola esecuzione la percentuale rimane più o meno costante per tutta la sua durata, anche prolungata. La causa di questo fenomeno è sconosciuta, ma non provoca disturbi significatici all'esecuzione dell'applicazione.

5.4 Risultati

Il risultato di tale lavoro ad è una applicazione che risponde in maniera accettabile ai requisiti dell'interfaccia descritta in questo lavoro, pur con alcuni limiti e problemi, realizzata sfruttando le classi che definiscono un framework per l'implementazione di tale interfaccia in maniera semplice. L'applicazione disegna su sfondo nero una serie di cubi virtuali con differenti texture, inseriti in diverse posizioni intorno all'utente. Egli può navigare dall'uno all'altro semplicemente ruotando la testa e spostando il campo visivo di conseguenza. Il comportamento del sistema in risposta al movimento non è ancora perfetto, e gli oggetti tendono a seguire leggermente il movimento della testa durante la rotazione. Tuttavia questo comportamento non desiderato è di lieve entità

e il risultato ottenuto si può definire pienamente accettabile come esempio di ciò che si vuole ottenere con tale lavoro. Per arrivare ad avere risultati ottimali sarà necessario studiare ulteriormente il problema. In questo esempio è possibile visualizzare alcune informazioni testuali e alcune informazioni visuali, sia semplici (una texture grafica) che più complesse (una rappresentazione del quadro *"Orologio fluido alla prima esplosione"* di S. Dalì).

Ovviamente tale lavoro si pone nell'ottica di una esplorazione iniziale di interfacce di questo genere, e tutti i test effettuati sono stati pensati per controllare l'effettivo funzionamento. Ulteriori test sull'usabilità saranno necessari per poter arrivare ad avere un prodotto finito ed usabile per gli utenti.

L'applicazione quindi offre la possibilità di visualizzare all'interno di uno spazio virtuale delle informazioni personalizzate, che accompagnano l'utente nelle sue attività senza occupare il suo campo visivo in alcun modo se egli non desidera visualizzarle. Questo spazio personale può essere la base di partenza per lo sviluppo di applicazioni con questa tecnologia. Questa prima versione presenta soltanto alcune prime funzionalità, ma rappresenta una buona base di partenza per poter espandere tale interfaccia. Inoltre un'applicazione di questo genere rappresenta un buon esempio di come potrebbero evolvere le applicazioni di realtà aumentata applicando l'interfaccia qui descritta. Perciò il lavoro svolto è un buon punto di partenza per valutare i punti a favore e contro ad uno sviluppo della tecnologia in tale direzione.

Conclusioni

Al termine di questo lavoro di tesi sono stati approfonditi diversi argomenti legati a dispositivi Embedded Wearable e a scenari di applicazione della Realtà Aumentata. Il lavoro svolto ha portato alla realizzazione di un'applicazione in grado di mostrare oggetti virtuali all'interno di uno spazio generato solidalmente all'utente stesso da Smart Glass che egli indossa, mettendo in luce una possibile via di sviluppo di tali tecnologie. L'importanza dell'esplorazione di interfacce innovative per questi dispositivi, per cui la ricerca e lo sviluppo sono argomenti attualissimi, è cruciale, in quanto può decidere il successo o meno di una tecnologia. Lo sviluppo di un framework di supporto per realizzare applicazioni che applicano tale interfaccia non è che il primo passo di una più lunga esplorazione che potrà portare ad innovazioni ancora maggiori. Il risultato ottenuto mostra che tale interfaccia ha tutte le potenzialità per essere ulteriormente studiata ed applicata in diversi ambiti anche della vita reale, permettendo all'utente finale di interfacciarsi facilmente ed in maniera naturale con queste nuove tecnologie, aiutandone la diffusione e lo sviluppo, ottenendo infine un'applicazione che permetta all'utente di essere veramente libero di scegliere se visualizzare informazioni aggiuntive o tenere il campo visivo libero per svolgere altre azioni senza dover togliere i dispositivi.

Sviluppi Futuri

Essendo il lavoro ancora alle prime fasi le possibili evoluzioni future di tale lavoro sono immense, anche soltanto a livello di funzionalità implementative il lavoro da svolgere è immenso, iniziando dallo studio di un sistema che permetta di muovere la camera all'interno del mondo virtuale, spostandosi liberamente nell'ambiente, e permettendo così di creare spazi di qualsiasi dimensione e forma. Lo sviluppo di una tecnologia di questo tipo potrebbe portare ad una più stretta interazione tra gli oggetti virtuali e la realtà fisica circostante, data la natura degli oggetti virtuali che li vuol fare assomigliare come comportamento ad oggetti fisici reali, e data la semplicità di interfacciamento, per cui potrebbe essere un modo di favorire la diffusione della realtà aumentata a

numerosi aspetti della vita di tutti i giorni, fino a farla entrare nella vita quotidiana delle persone. Tra i possibili sviluppi si potrebbero avere superfici a più livelli, contenenti oggetti ed informazioni differenziate a seconda del livello selezionato, oppure, tramite l'implementazione di tecniche di comunicazione con altri dispositivi, arrivare ad avere spazi condivisi tra più utenti in una stessa stanza, o addirittura condivisi attraverso la rete, in modo da vedere la stessa informazione su dispositivi differenti distanti tra loro migliaia di chilometri.

Ringraziamenti

E anche questa lunga avventura universitaria è arrivata al termine. E come ogni avventura che si rispetti non può concludersi senza ringraziare le persone che vi hanno preso parte:
Ringrazio innanzitutto il Professor Ricci e il Dottor Croatti per il supporto che mi hanno dato durante la stesura di questo lavoro: senza la loro guida e la disponibilità del loro tempo questa tesi non esisterebbe. Ed un ringraziamento va anche alla prof. Mirri per la disponibilità ed il materiale che mi ha fornito.

Ringrazio Mattia, Piero, Luca, Nicola, Marco, Lorenzo e Jacopo, che più di ogni altro hanno condiviso con me il peso degli esami e le lunghe ore di lezione in questi tre anni, rendendo questo percorso molto più leggero.

Ringrazio la grande famiglia dell'Azione Cattolica, che come una seconda casa e una seconda famiglia mi ha aiutato a crescere come persona in tutti questi anni e che mi ha permesso di conoscere le persone più importanti della mia vita.

E proprio a voi, Amici, è rivolto il mio prossimo grazie. Potrei citarvi uno ad uno, ma non penso che ce ne sia bisogno. Grazie innanzitutto per il supporto che mi avete dato per questo lavoro, rileggendo, discutendo o semplicemente ascoltando i miei ragionamenti. E poi ancore grazie perché con la vostra presenza avete riempito le mie giornate, impedendomi di conoscere la noia. Grazie perché con la vostra follia mi avete spinto ben oltre i miei limiti, portandomi a fare cose che non avrei mai pensato di fare. Grazie per avermi sopportato anche se "Devo scrivere la tesi" ormai era diventato il mio tormentone. Grazie per avermi sopportato e voluto bene per tutti questi anni, nonostante tutto.

Infine ringrazio amici e conoscenti, e tutti coloro che con me hanno condiviso o stanno condividendo un pezzo di vita, e ultima, ma non ultima, ringrazio la mia famiglia, che in tutti questi anni mi ha sostenuto ed aiutato non solo economicamente, non facendomi mai mancare il suo affetto.

Grazie.

Bibliografia

[1] Leslie Pack Kaelbling, *Learning in Embedded Systems*, MIT Press, 1993.

[2] Y. Narasimha Murthy, *Unit I - Introduction to Embedded systems*, in *"http://www.slideshare.net/yayavaram/unit-1-embedded-systems-and-applications"*

[3] *Worldwide Quarterly Wearable Device Tracker*, International Data Corporation, December 17, 2015.

[4] Samuel Mann, *Wearable computing as a means for personal empowerment* in *"Keynote speech of 1998 International Conference on Wearable Computing"*

[5] Thad Starner, *The challenges of Wearable Computing: part 1*, IEEE Micro, 2001.

[6] FitBit, *Fai del fitness il tuo stile di vita con Flex*, in *"https://www.fitbit.com/it/flex"*

[7] Maxi Sport, *Cos'è e come funziona Nike+?*, in *"http://www.maxisport.com/ap/cos-e-come-funziona-nike-2-a.htm"*

[8] Apple News Italia, *Apple Watch: scheda tecnica, uscita e prezzo in Italia*, in *"http://applenewsitalia.com/apple-watch-scheda-tecnica-uscita-e-prezzo-italia"*

[9] Asus, *ASUS ZenWatch WI500Q*, in *"https://www.asus.com/it/ZenWatch/ASUS-ZenWatch-WI500Q/"*

[10] WearableTecnologies.com *Avery Dennison Metria – Wearable Sensor Patch*, in *"https://www.wearable-technologies.com/gadgets-of-the-month/nov-11-metria-wearable-sensor-technology"*

[11] ThermalVision *What is Thermal Vision ?*, in *"http://www.radiateathletics.com/pages/about-us"*

[12] WebNews, *Google Glass, tutto sugli occhiali Android*, in *"http://www.webnews.it/speciale/google-glass/"*

[13] Microsoft, *Microsoft HoloLens*, in *"https://www.microsoft.com/microsoft-hololens/en-us"*

[14] Epson *Moverio BT-200, Visore con lenti trasparenti*, in *"http://www.epson.it/moverio"*

[15] Thad Starner, *Project Glass: An Extension of the Self*, in *"www.computer.org/pervasive"*, April–June 2013

[16] Accenture, *Engaging the Digital Consumer in the New Connected World*, in *"https://www.accenture.com/"*

[17] UNI, *Ergonomics of human-system interaction, part 210*, in *UNI EN ISO 9241-210*, Novembre 2010

[18] Donald A. Norman, *La caffettiera del masochista. Il design degli oggetti quotidiani*, Ed. Giunti, 2015

[19] Andrea Zingoni, *Un tuffo nell'universo reale-virtuale della realtà aumentata*, in *"http://www.wbt.it/igel/un-tuffo-nell%E2%80%99universo-reale-virtuale-della-realt%C3%A0-aumentata"*,

[20] Craig Smith, *Augmented reality in Lego stores*, in *"http://retail-innovation.com/augmented-reality-in-lego-stores/"*

[21] IKEA, *Place IKEA furniture in your home with augmented reality*, in *"https://www.youtube.com/watch?v=vDNzTasuYEw"*

[22] World Tour, Etips, *Florence Travel Guide - Italy - Augmented Reality*, in *"https://www.youtube.com/watch?v=a90DDQZmGj4"*

[23] Ivan E. Sutherland *A head-mounted three dimensional display*, in *"http://design.osu.edu/carlson/history/PDFs/p757-sutherland.pdf"*

[24] Kiyoshi Kiyokawa *Head-Mounted Display Technologies for Augmented Reality*, in *Fundamentals of Wearable Computers and Augmented Reality*, Second edition, CRC Press, 2015

[25] Bernard Kress *Optics for Smart Glasses, Smart Eyewear, Augmented Reality, and Virtual Reality Headsets*, in *Fundamentals of Wearable Computers and Augmented Reality*, Second edition, CRC Press, 2015

[26] Android *Sensors Overview*, in *"http://developer.android.com/guide/topics/sensors/sensors-overview.html"*

[27] Android *SensorEvent*, in *"http://developer.android.com/reference/android/hardware/SensorE*

[28] OpenGL.org *Rendering Pipeline Overview*, in *"https://www.opengl.org/wiki/Rendering-Pipeline-Overview"*

[29] Khronos Group *OpenGL ES: The Standard for Embedded Accelerated 3D Graphics*, in *"https://www.khronos.org/opengles/"*

[30] Joey de Vrie *OpenGL Camera*, in *"http://learnopengl.com/#!Getting-started/Camera"*