Alma Mater Studiorum - Università di Bologna

Campus di Cesena

Scuola di Ingegneria e Architettura

Corso di Laurea in Ingegneria Elettronica, Informatica e Telecomunicazioni

REALTÀ AUMENTATA PER DISPOSITIVI ANDROID: LO STATO DELL'ARTE

Elaborata nel Corso di:

Sistemi Operativi L-A

Tesi di Laurea di: **COSENZA FABIO**

Relatore: **Prof. RICCI ALESSANDRO**

Anno Accademico 2013/2014

II Sessione

PAROLE CHIAVE

Augmented Reality

Realtà Aumentata

Metaio SDK

Android

Mobile

Java

"But Mousie, thou are no thy-lane,
In proving foresight may be vain:
The best laid schemes o' Mice an' Men,
Gang aft agley,
An' lea'e us nought but grief an' pain,
For promis'd joy!

Still, thou art blest, compar'd wi' me!
The present only toucheth thee:
But Och! I backward cast my e'e,
On prospects drear!
An' forward, tho' I canna see,
I guess an' fear!"

- di **Robert Burns**, tratto da *"To a Mouse"*

Indice

1. Introduzione

Un elemento costante dell'immaginario dell'uomo è sempre stato quello di arricchire la visione fisica del mondo con informazioni o oggetti non realmente presenti, magari con la possibilità di interagirvi: in poche parole, di aumentare la realtà.

Di fantasie di applicazione ne troviamo soprattutto nelle arti dei videogiochi e del cinema, quando ad esempio abbiamo visto Michael J. Fox venire attaccato dall'ologramma di uno squalo in *Ritorno al Futuro – Parte II*, film del 1989, o ancora *RoboCop* rilevare i criminali armati in una stanza e visionare video di prove durante gli arresti. Ma il poliziotto cyborg faceva questo nella pellicola del 1987, mentre nella sua reinvenzione del 2014 lo vediamo anche analizzare i volti delle persone per rilevarne lo stato emotivo e prevederne azioni violente, oppure ricostruire una scena del crimine combinando le registrazioni da diverse angolature di più telecamere, o ancora visualizzare informazioni sull'ambiente circostante.

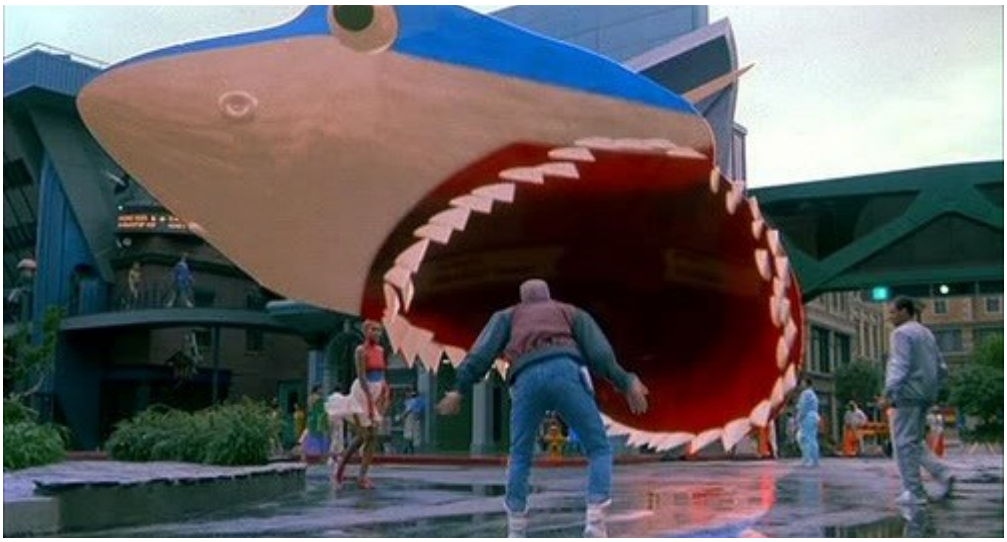


Fig. 1, una scena del film "Ritorno al Futuro – Parte II"

Ma si tratta ancora di fantascienza? La risposta corretta potrebbe essere “no”: con gli attuali calcolatori è possibile creare applicazioni che ci immergano in una realtà aumentata, ed esistono hardware specifici studiati proprio per questo. Quelli che si sente nominare più spesso negli ultimi anni sono gli smartglass, consistenti in occhiali con monitor integrati nelle lenti che si sovrappongono al campo visivo della persona che li indossa. Probabilmente è solo questione di pochi anni prima che tali dispositivi raggiungano un prezzo più accessibile al pubblico e diventino strumenti quasi indispensabili per l'utilizzo quotidiano, come successo recentemente con smartphone e tablet; ma già ora possiamo assaporare esempi di realtà aumentata proprio con questi ultimi strumenti: sfruttando in particolare la loro fotocamera, programmare applicazioni per questi o per occhiali moderni non è diverso, grazie anche al fatto che sono spesso basati sul medesimo sistema operativo Android. Non per scherzo, il colosso americano Google ha presentato all'evento “Google I/O 2014” il progetto *Cardboard*: una semplice app gratuita per cellulari Android accompagnata dalle istruzioni per costruire degli occhiali di cartone fai-da-te, con un alloggio per inserirvi il proprio smartphone.

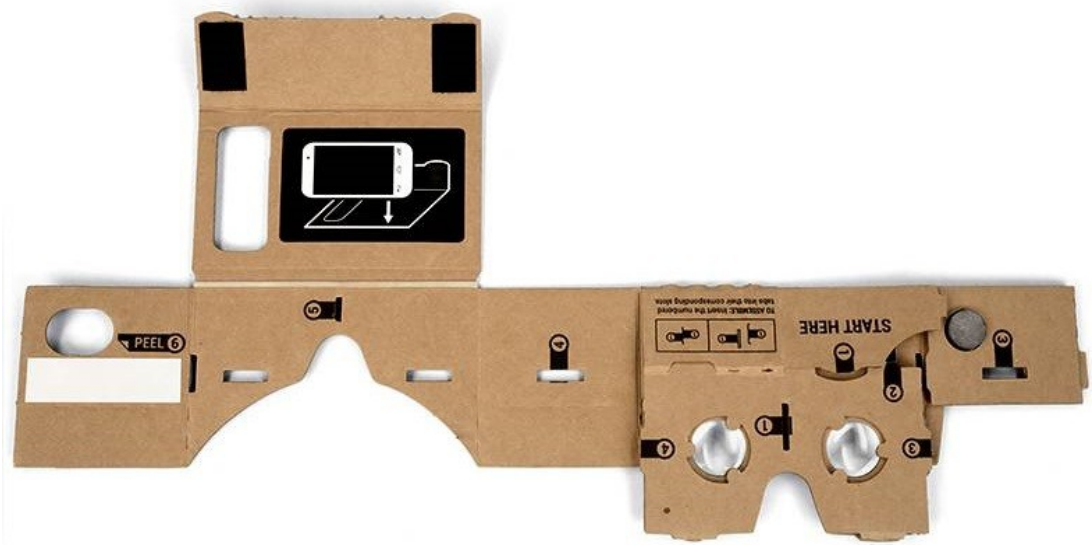


Fig. 2, gli occhiali di cartone di Google

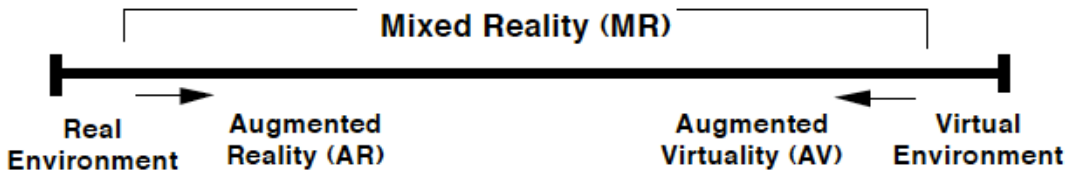
La realtà aumentata è, quindi, già nelle nostre possibilità. Obiettivo di questa tesi è quello di illustrare questo nuovo mondo, descrivendone le ricche caratteristiche ed analizzandone i vari aspetti nella maniera più precisa possibile: si partirà dal darne una definizione e riassumerne i principali fatti storici, all'illustrarne i vari hardware disponibili sul mercato e le tecnologie software per sviluppare progetti. Non verranno tralasciati utilizzi e settori di ricerca, e si presenterà poi il sistema operativo Android. Dopo uno sguardo alla sua architettura e alle sue caratteristiche, nonché al linguaggio di programmazione Java, cardine per lo sviluppo in questo sistema, si presenteranno alcune API dell'SDK nativo che si rivelano utili per lo sviluppo di applicazioni per la realtà aumentata. Infine, verrà presentato un approfondimento sull'SDK Metaio.

2. Realtà Aumentata: panoramica

In questo capitolo verrà fornita una prima e generica panoramica della realtà aumentata. Partendo da una definizione del 1994, che inquadra e separa i concetti di *realtà aumentata* e di *virtualità aumentata*, si riassumerà la storia di questa scienza, a partire dagli anni '50. Infine, saranno proposte una ricca scelta di suoi utilizzi per la vita quotidiana, presenti e futuri, fondamentali per dare un'idea delle possibilità offerte da questa nuova scienza che, col progredire delle tecnologie e delle ricerche, è destinata ad essere limitata dalla sola immaginazione dei programmatori.

2.1 Definizione

Le definizioni assegnate alla realtà aumentata sono molteplici, ma quella che probabilmente è la più intuitiva è stata fornita nel 1994 da Paul Milgram e Fumio Kishino, docenti rispettivamente dell'Università di Toronto e dell'Università di Osaka.



Reality-Virtuality (RV) Continuum

Essi hanno posizionato da una parte l'ambiente reale, dove viviamo e che tutti conosciamo, e posto in completa opposizione la *realtà virtuale*, dove qualsiasi cosa non esiste, in quanto generata da un calcolatore elettronico e totalmente sconnessa dalla nostra realtà. In mezzo ai due estremi è compresa la *realtà mista*, dove mondo reale ed elementi virtuali vengono mischiati, e le cui varie sfumature vengono sostanzialmente divise in *realtà aumentata* e *virtualità aumentata*.

Mentre nella prima la pura realtà fa da sfondo ed elementi non reali vi vengono sovrapposti, nella seconda sono elementi reali a venire mescolati ad un mondo totalmente generato da un computer. Quest'ultima può essere trovata facilmente nella quotidianità, specificatamente nell'intrattenimento videoludico: ad esempio nelle console Nintendo Wii e Microsoft Xbox 360, quando i movimenti del giocatore vengono catturati tramite *Wiimote* o *Kinect* e trasportati all'interno del videogioco. Anche la realtà aumentata, seppur con minor diffusione, ha trovato applicazione nei videogiochi, in particolare per Android e iOs. Per quest'ultimi, ne sono un esempio i *Cupets*: animali giocattolo ciascuno con un marker identificatore, che viene riconosciuto da un'applicazione per smartphone e tablet in modo da dargli vita su schermo.

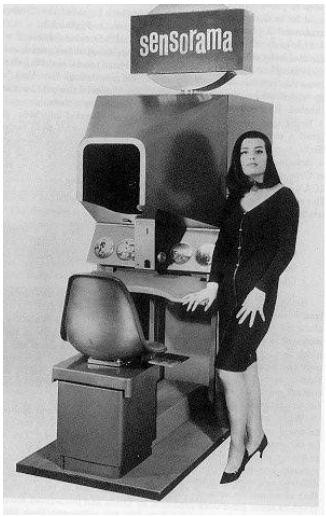
Fig. 4, un esempio di esperienza di virtualità aumentata: una ragazza che gioca a Wii Sports

Una seconda, importante e più selettiva definizione di realtà aumentata è stata pensata da Ronald Azuma nel 1997, pioniere del

settore. Secondo lui, la realtà aumentata è:

- la combinazione di mondo reale e oggetti virtuali,
- con i quali è possibile interagire in tempo reale,
- e che sono integrati in 3D nel campo visivo dell'utente.

2.2 Cenni storici



La storia della realtà aumentata comincia alla fine degli '50, quando Morton Heilig inventò un simulatore denominato *Sensorama*. Nato per estendere il cinema a tutti e cinque i sensi, questo dispositivo permetteva di vedere cinque cortometraggi, accompagnati da suoni e odori che l'utilizzatore poteva percepire, così come il vento generato e le inclinazioni della macchina. Nonostante le potenzialità di *Sensorama*, Heilig non trovò finanziatori per poter continuare il progetto.

Nel 1962 Ivan Sutherland realizzò Sketchpad, la prima applicazione dotata di un'interfaccia grafica. Nello stesso anno, ancora Helig brevettò il primo Head Mounted Display (HDM), consistente in un Sensorama di dimensioni ridotte. Tale dispositivo non entrò mai produzione, e fu Sutherland nel 1966 a creare il primo prototipo di HDM, denominato *Ultimate Display*. Due anni più tardi, insieme al suo studente Bob Sproull, realizzò *The Sword of Damocles*, quello che è considerato il primo HDM della storia, che permetteva di visualizzare un rudimentale ambiente virtuale.

Fig. 6,The Sword of Damocles

Nel 1975, Myron Krueger creò un laboratorio di realtà artificiale denominato *Videoplace*, nel quale veniva creata una nuova realtà attorno all'utente, il quale poteva interagirvi. Nel 1980 Steve Mann inventò invece i Wearable Computer. Appartengono a questa categoria di dispositivi gli smartglass.

Nel 1989 Jaron Lanier coniò il termine *realtà virtuale*, mentre quello di *realtà aumentata* fu coniato da Tom Caudell l'anno seguente. Nel 1992 Louis Rosenberg sviluppò *Virtual Fixtures*, ovvero il primo sistema funzionante di realtà aumentata: con esso, costituito da un visore ed un esoscheletro, l'utente vedeva due braccia robotiche al posto delle proprie, che invece erano occupate ad impartire ordini tramite l'esoscheletro, per ottenere una maggiore precisione nei movimenti. Nel 1998 Ramesh Raskar, Greg Welch e Henry Funchs diedero vita alla *Spatial Augmented Reality*, la quale non utilizza alcun display, ma gli elementi virtuali vengono visualizzati sul mondo fisico tramite proiettori. Nello stesso anno, Jun Rekimoto introdusse il concetto di *marker*.

Nel 1999 Hirokazu sviluppò *ARToolKit*, una libreria di tracking per la realizzazione di applicazioni per la realtà aumentata, che è oggi

opensource. Un anno dopo, Bruce Thomas programmò *ARQuake*: il celebre videogioco Quake giocato nel mondo reale, nel quale il giocatore si muove mentre visualizza i nemici e gli altri elementi di Quake sul proprio visore. Nel 2008 Wikitude rilasciò *AR Travel Guide*, un'app che consente di visualizzare informazioni sui luoghi pubblici, accompagnata da tool per gli sviluppatori. Nel 2011 nacque anche *Wikitude Drive*, niente meno che il primo navigatore satellitare a sfruttare la realtà aumentata.

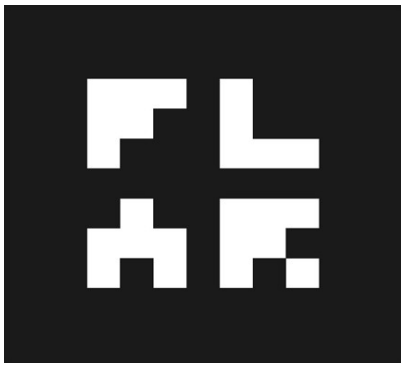


Fig. 7, un marker

Fig. 8, Wikitude Drive su uno smartphone Android

Fig. 9, ARQuake

Negli anni più recenti, con lo sviluppo della tecnologia necessaria e l'enorme diffusione di smartphone e tablet, la ricerca riguardante la

realtà aumentata si è intensificata moltissimo: non solo per quanto riguarda il miglioramento dell'hardware e del software, ma anche per lo studio di possibili utilizzi.

2.3 Esempi di utilizzo

La realtà aumentata è destinata ad entrare nella nostra vita di tutti i giorni, in tanti modi possibili e molto più di come è successo con smartphone e tablet, dato che con dispositivi come gli smartglass ci sarà sufficiente guardare in giro per sfruttarne le potenzialità. Ad esempio, nei negozi potremmo essere accolti all'ingresso da una mascotte virtuale, generata automaticamente incrociando un marker con lo sguardo, oppure avvicinandoci alla nostra automobile potremmo essere informati della carenza di benzina ancor prima di entrare in macchina ed accendere il motore.

Qui di seguito sono elencati alcuni esempi di utilizzi possibili e già sperimentati, divisi per settore di applicazione.

2.3.1 Pubblicità

Il forte impatto visivo della realtà aumentata può rivelarsi un'arma importante per il marketing, che in parte ha già cominciato ad utilizzarla per catturare l'attenzione dei consumatori e migliorarne l'esperienza d'acquisto. Ad esempio, con l'utilizzo di marker nelle riviste, il modello tridimensionale di un prodotto può “saltare” fuori da una pagina pubblicitaria, e l'utente è in grado di ruotarlo per vederlo da ogni angolazione possibile, cosa che non sarebbe in grado di fare con la sola pagina stampata. Il concetto è stato utilizzato anche dal Servizio Postale degli Stati Uniti d'America (USPS): i consumatori possono stampare marker dal sito ufficiale e, con l'utilizzo di un'apposita app, potranno vedersi tra le mani le confezioni da usare per le spedizioni, in modo da trovare la scatola delle dimensioni adatte per gli oggetti che intendono spedire.

Una simile idea è stata sfruttata da AR Door che, in coppia con Topshop (una catena inglese di abbigliamento), ha realizzato specchi virtuali che sostituiscono il camerino: l'utente che vuole provarsi un abito deve semplicemente mettersi di fronte allo schermo, selezionare il vestito e se lo vedrà direttamente addosso. Per realizzare ciò, è stata utilizzata la tecnologia Microsoft Kinect. Invece, Ikea, con un'app per dispositivi mobile, ha realizzato un catalogo virtuale che consente di posizionare i nuovi mobili nella stanza in cui l'acquirente sta puntando la fotocamera.

Fig. 10, l'app di Ikea eseguita su un iPad

2.3.2 Costruzioni e riparazioni

La realtà aumentata si propone anche di far risparmiare tempo e denaro nelle attività di costruzione, riparazione e manutenzione: tramite di essa, infatti, si potranno vedere le operazioni da compiere sovrimpresse a quel che si sta facendo. Ad esempio, in un caso di montaggio, all'utente basterà prendere in mano il pezzo reale e inserirlo dove si trova quello virtuale, evitando di ragionare su istruzioni criptiche. Ciò sarà particolarmente utile per lavori di elevata complessità.

Fig. 11, un meccanico e il suo punto di vista

2.3.3 Intrattenimento

Le applicazioni più creative della realtà aumentate le vediamo e le vedremo, però, nel settore dell'intrattenimento.

Il più grande collegamento tra la realtà fisica e quella virtuale sarà dato dai libri che sfrutteranno i marker, come illustrato precedentemente con la pubblicità nelle riviste: con essi le pagine prenderanno vita davanti ai nostri occhi. Il libro dei Guinness World Records sfrutta già questa tecnologia, mentre un altro esempio è "*The Future is Wild: the Living Book*" del 2011 di Metaio.

Ma i marker vengono utilizzati anche nei videogiochi, ad esempio con le *AR Cards* di Nintendo, che ci permettono di vedere, attraverso gli schermi 3D del Nintendo 3DS, Super Mario, Pikachu e gli altri personaggi dell'azienda giapponese scorazzare sulla nostra scrivania. Interi giochi sono basati su questa tecnologica, come *Invimals* per PSP.

Nel mondo dello spettacolo, invece, abbiamo visto ologrammi ondeggiare tra un'orchestra mentre suonava i pezzi del *Titanic Requiem* nel 2012, opera classica del celebre Robin Gibb.

Fig. 12, uno squalo bianco emerge dalle pagine del libro di Guinness World Records

2.3.4 Medicina

Un importantissimo utilizzo della realtà aumentata sarà nella medicina: medici e chirurghi potranno vedere ricostruzioni virtuali interne dei pazienti proprio su quest'ultimi. Ciò semplificherà e renderà molto meno invasivi una parte di esami clinici e operazioni chirurgiche.

Fig. 13, la realtà aumentata "come raggi x"

2.3.5 Settore militare

La realtà aumentata avrà anche grande utilizzo in ambito militare. Con gli appositi visori, i soldati potranno vedere mappe del territorio, orientarsi con la bussola, identificazioni per compagni e nemici, la distanza dall'obbiettivo e altro ancora, vantando una interfaccia esattamente uguale a quella dei videogiochi futuristici di guerra. Grande importanza la rivestirà la comunicazione con la base, in grado di arricchire la visuale dei soldati con componenti virtuali aggiuntivi.

Fig. 14, la serie di videogiochi Tom Clancy's Ghost Recon come anteprima di realtà aumentata applicata al settore militare

2.3.6 Turismo

Come ultimo esempio di utilizzo di realtà aumenta è qui proposto l'ambito turistico. Sarà ed è già possibile, ad esempio con *Wikitude*, camminare per strada e vedere attraverso uno schermo, ora quello dello smartphone ed in futuro quello degli smartglass, informazioni sull'ambiente circostante: piazze, monumenti, edifici, negozi e quant'altro. La localizzazione della posizione dell'utente avviene tramite il sistema GPS, mentre le informazioni vengono scaricate da Internet.

Un'applicazione di realtà aumentata è già stata utilizzata anche in alcuni musei, come guida turistica personalizzata per ogni visitatore, che vedrà apparire informazioni e sentire iniziare clip audio in base a ciò che sta osservando.

Fig. 15, Wikitude su iPhone

3. Realtà Aumentata: tecnologie

In questo capitolo, verranno illustrate le tecnologie disponibili per la realtà aumentata. Si partirà con l'hardware, descrivendo i componenti che un dispositivo deve avere e, poi, gli smartglass. Come approfondimento, verranno fornite le caratteristiche di *Google Glass* e degli *Epson Moverio BT-200*, due dei modelli di smartglass attualmente tra i più rilevanti, dei quali il secondo in dotazione alla Scuola.

Subito dopo verranno presentati i principali framework per sviluppare applicazioni dedicate a questi dispositivi e verrà, infine, descritta l'architettura di base di una qualsiasi applicazione di realtà aumentata.

3.1 Dispositivi

Dispositivi utili per la realtà aumentata li abbiamo più o meno tutti sotto mano, e sono i già più volte citati smartphone e tablet. Ciò che rende possibile utilizzarli per questa scienza è la loro fotocamera: tramite di essa viene acquisita una visione del mondo attorno a noi, viene poi elaborata aggiungendoci elementi virtuali e, infine, ci viene mostrata su schermo. Ma ciò che li rende particolarmente appetibili per lo sviluppo di queste nuove applicazioni, oltre alle straordinarie capacità di calcolo impensabili fino a pochissimi anni fa, è anche la presenza di altri sensori hardware. Tra questi vi sono:

- l'*accelerometro*: come si intuisce dal nome, rileva e quantifica l'accelerazione guadagnata dall'oggetto;
- il *giroscopio*: serve per rilevare tutti i movimenti dell'oggetto nello spazio. Insieme all'accelerometro, serve per cogliere tutti i dettagli possibili sugli spostamenti del dispositivo;
- il *magnetometro*: misura il campo magnetico e viene utilizzato come bussola;
- il *sensore di prossimità*: ne esistono di diverse tecnologie. Può

essere basato, ad esempio, sull'utilizzo di onde elettromagnetiche: emette onde e cattura i relativi riflessi, dai quali deduce la distanza dall'ostacolo. Nel caso particolare degli smartphone, serve per disattivare il touch screen quando si avvicina il cellulare all'orecchio;

- il *sensore di luce*: rileva il grado di illuminazione dell'ambiente.

Tuttavia, muoversi per strada tenendo uno smartphone davanti agli occhi non è pratico, quindi servono dispositivi hardware più adatti e con schermi di diversa natura, e nello specifico ricerca e mercato si sono mossi nella direzione degli *smartglass*.

Gli *smartglass* rientrano nella categoria degli Head-Mounted Display, e sovrappongono uno o due schermi (uno per occhio) alla vista degli utenti. Questi sono schermi trasparenti: l'indossatore ha la sua visione dell'ambiente e gli elementi virtuali vi sono quindi messi dinanzi. Ne esistono due tipi: quelli che semplicemente mostrano informazioni sullo schermo e non si preoccupano di integrarli alla realtà, montati ad esempio dai *Google Glass*, e quelli che invece fanno in modo che gli elementi virtuali sembrino immersi nella realtà, come quelli offerti dagli *Epson Moverio BT-200*. Stando alla definizione di Ronald Azuma fornita nel primo capitolo, la prima tipologia di smartglass (e quindi anche i *Google Glass*) non offrono un'esperienza di realtà aumentata.

Esistono anche altri sistemi che si basano su *proiettori*, sia fissi che portatili: non richiedono quindi monitor, in quanto gli elementi virtuali vengono presentati sott oforma di ologrammi.

Qui di seguito vengono presentati i due modelli di smartglass accennati poche righe più sopra.

3.1.1 Google Glass

I Google Glass sono gli smartglass più famosi, sia per la notorietà di Google, sia perché sono considerati i capostipiti della categoria.

Montano un monitor semi-trasparente davanti all'occhio destro,

nel quale un piccolo proiettore proietta un'immagine di risoluzione 640x360 pixel. È l'equivalente di un televisore da 25 pollici guardato da una distanza di due metri.

Fig. 16, i Google Glass

Sono dotati di microfono, accelerometro, giroscopio, magnetometro, GPS e fotocamera da 5Mpx. Hanno una memoria flash da 16GB, mentre montano un processore dual core da 1.2GHz e una RAM di 2GB. Presentano, inoltre, un'antenna Wi-Fi ed una Bluetooth, tramite la quale sono in grado di interfacciarsi con altri dispositivi, in particolar modo gli smartphone. L'audio che generano viene percepito dall'utilizzatore grazie alla conduzione delle ossa del cranio.

Per quanto riguarda l'interfacciamento con l'utente, i Google Glass hanno sull'asta destra della montatura un touchpad per poter navigare tra i contenuti, ed in più i comandi possono essere impartiti anche vocalmente, tramite microfono.

Montano Android come sistema operativo, e ci si aspetta che verranno venduto prossimamente al pubblico con un prezzo attorno ai 600$.

3.1.2 Epson Moverio BT-200

Gli Epson Moverio BT-200, successori dei BT-100, hanno uno schermo per lente, sui quali vengono proiettate immagini di risoluzione 960x540 pixel, equivalenti a circa un televisore di 40 pollici guardato da due metri di distanza.

Fig. 17, gli Epson Moverio BT-200

Presentano accelerometro, giroscopio, magnetometro, GPS, antenna Wi-Fi, antenna Bluetooth e fotocamera in bassa risoluzione, pensata per funzioni di realtà aumentata (come la lettura dei marker) e non per scattare fotografie e girare video. Ha una memoria interna da 8GB, espandibile con micro-SD fino a 32, un processore dual core da 1.2GHz e 1GB di RAM.

Come detto in precedenza, si distinguono dai Google Glass per l'utilizzo: mentre quest'ultimi si limitano a mostrare contenuti 2D all'utente sovrimpressi alla loro visione, i Moverio sono in grado di integrare elementi virtuali nella realtà fisica. Un'altra differenza importante, sta nell'interfacciamento con l'utente: collegato tramite filo, infatti, c'è una scatola di controllo con un touchpad che consente di muovere un puntatore virtuale (come con un notebook) e cliccare le icone, e ha inoltre tasti fisici come quello per tornare all'homepage.

Anch'essi montano Android, e sono già disponibili sul mercato al prezzo di 699$.

3.2 Tool di sviluppo

Sono disponibili molti tool per sviluppare applicazioni di realtà aumentata, sia gratuiti che a pagamento, e per molte piattaforme. Qui di seguito verranno presentati i tre più famosi, tutti disponibili anche per Android.

3.2.1 ARToolKit

ARToolKit è una libreria gratuita di funzioni in C (poi convertite anche in Java su Android). Supporta sia i dispositivi con schermo trasparente, sia quelli senza, e consente di creare applicazioni di realtà aumentata basate solo sull'utilizzo di marker. Per farlo, utilizza tecniche per calcolare la posizione e l'orientamento della fotocamera rispetto ai marker, permettendo quindi di sovrapporgli elementi virtuali.

Per fare ciò, *ARToolKit* riceve ogni frame registrato. Il frame arrivato viene quindi convertito in un'immagine binaria, in modo che sia possibile cercarvi un pattern quadrato, cioè un marker. Se questo viene trovato, viene calcolata la posizione della fotocamera rispetto ad esso, e viene poi confrontato con tutti i marker in memoria. Se viene trovato, l'oggetto 3D associato viene allineato col marker e poi renderizzato.

3.2.2 Metaio

Metaio è probabilmente il framework più completo per la realizzazione di applicazioni di realtà aumentata. Disponibile sia in versione gratuita che in versioni a pagamento, mette a disposizione dello sviluppatore diversi strumenti.

Mentre ARToolKit permette il tracking solo tramite marker,

Metaio SDK permette la realizzazione applicazioni che utilizzino tecniche di tracking basate su immagini 2D, modelli 3D e sulla posizione dell'utente. È anche possibile utilizzare un apposito linguaggio di scripting, chiamato *AREL*, che consente di collegare le interfacce grafiche in XML, HTML5 e JavaScript alle API dell'SDK.

Con *Metaio Creator*, invece, è possibile creare esperienze di realtà aumentata anche senza abilità di programmazione, semplicemente con pochi click e *drag and drop*. L'applicazione creata può essere salvata come app stand alone, oppure da utilizzare attraverso *Junaio*, l'*augmented reality browser* di Metaio, ovvero un visualizzatore di contenuto di realtà aumentata.

3.2.3 Wikitude

Wikitude offre framework e SDK, gratuiti e non, per molteplici piattaforme, anche versioni specifiche per dispositivi come i Google Glass o gli Epson Moverio BT-200. I tool sono fondati su tecnologie web (HTML, CSS e JavaScript) e sfruttano sostanzialmente il tracking basato sulla posizione dell'utilizzatore.

3.3 Architettura di un'applicazione

In seguito allo studio di applicazioni di realtà aumentata, Thomas Reicher, Asa MacWilliams e Bernd Brügge nel 2004 hanno identificato un'architettura comune a tutte le applicazioni di questo genere.

Tale architettura si divide in sei sottosistemi:

- *Application Subsystem*: è un contenitore del codice dell'intera applicazione;
- *Interaction Subsystem*: raccoglie e processa gli input forniti dall'utente;
- *Presentation Subsystem*: si occupa dell'output, ovvero della

visualizzazione dei contenuti;

- *Tracking Subsystem*: si occupa del tracciamento della posizione dell'utente, e condivide il risultato con gli altri sottosistemi;
- *Context Subsystem*: raccoglie differenti dati, non di tracking, e li condivide con gli altri sottosistemi;
- *World Model Subsystem*: l'utente si muove nel mondo reale e, da questo, l'applicazione trae informazioni, che vengono salvate proprio in tale sottosistema.

I sottosistemi, a loro volta, sono formati da diversi componenti, come illustrato nella figura seguente.

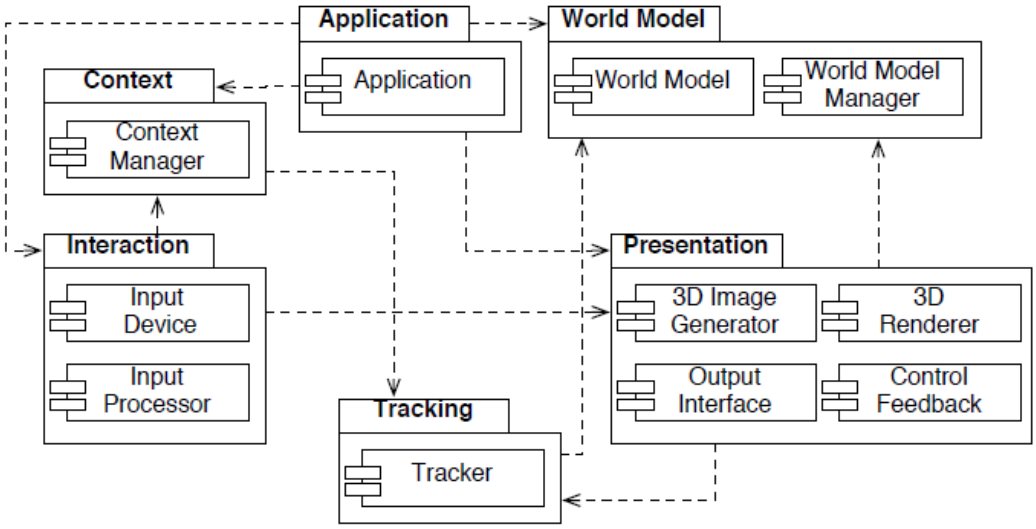


Fig. 18, un grafico UML rappresentante l'architettura di un'applicazione di realtà aumentata

4. Realtà Aumentata: settori di ricerca

Lo sviluppo della realtà aumentata va avanti con lo studio di nuovi dispositivi ed il miglioramento di quelli già esistenti, oltre che con la programmazione di nuovo software per migliori soluzioni e l'invenzione di nuovi utilizzi. Tuttavia, esistono due settori di ricerca di particolare interesse: quello di *tracking* e *registration*, ovvero come identificare gli oggetti reali e la posizione dell'utente e come integrare gli oggetti virtuali nella realtà fisica, e quello dell'*interazione uomo-macchina*, ovvero come si possa dare comandi di input ai dispositivi nella maniera più comoda ed intuitiva possibile.

In questo capitolo verranno descritti ed approfonditi questi importanti aspetti.

4.1 Tracking e Registration

Il *tracking* è il problema di dover rilevare la posizione dell'utente e il suo orientamento rispetto all'ambiente circostante, mentre per *registration* si intende la problematica di dover allineare gli oggetti virtuali alla visione dell'utente, in modo che sembrino realmente presenti. Ad esempio, immaginando una caccia al tesoro in realtà aumentata, è un problema di *registration* quello di mostrare i tesori in modo che le loro dimensioni varino in base alla distanza dalla quale il giocatore li osserva, mentre calcolare la posizione dell'utente, e quindi questa distanza, è un problema di *tracking*. Come si vede, le due cose vanno a braccetto, e bisogna fare in modo che il risultato dei calcoli sia affidabile, preciso e in tempo reale, quindi col minor ritardo possibile.

Il tracking può essere effettuato attraverso i sensori descritti in un paragrafo precedente, di cui i dispositivi mobile per la realtà aumentata sono forniti. In particolare, per rilevare la posizione dell'utente può essere sfruttato il *GPS*, che rileva i segnali inviati da appositi satelliti in orbita intorno alla Terra e, conoscendo la posizione di questi, riesce a fornire al sistema l'esatta posizione del dispositivo misurando il tempo

che hanno impiegato i segnali per arrivare dai rispettivi satelliti al sensore. L'affidabilità di questo metodo, tuttavia, può venire compromessa se l'utente si trova in un luogo chiuso o nelle vicinanze di un ostacolo imponente.

Per rilevare l'orientamento dell'utilizzatore, ovvero la direzione verso cui sta guardando, può essere utilizzato il *sensore magnetico*, che rileva il campo magnetico terrestre e ne trae il proprio risultato. Tuttavia, questo potrebbe essere influenzato dalla presenza di altri campi magnetici, ad esempio generati da oggetti elettronici. I piccoli spostamenti e le inclinazioni del dispositivo di realtà aumentata in uso possono venire rilevati da *accelerometri*, che misurano l'accelerazione lungo un singolo asse, e da *giroscopi*, in grado di percepire le rotazioni. Per verificare invece la presenza di ostacoli nei dintorni dell'utente, invece, possono essere utilizzati *sensori di prossimità*, che si basano su diverse tecnologie e sono in grado di rilevare gli oggetti entro la loro portata, ricevendo il riflesso del segnale da loro inviato e facendo considerazioni sul tempo intercorso tra l'invio e l'eventuale ricezione. I risultati, tuttavia, possono venire influenzati dalle condizioni climatiche.

Altrimenti, senza utilizzare i sensori, il tracking può essere basato sull'analisi di quanto rilevato dalle videocamere, e può sfruttare o meno la presenza di *marker*. Nel caso avvenga tramite marker, ogni frame catturato viene analizzato per cercarne la presenza. Quando ne viene identificato uno, attraverso tecniche grafiche per la rilevazione e la rimozione dei margini, dal marker viene isolato il bordo e letto il contenuto, che è il solo a contenere informazione e codice per la rilevazione di errore. Una volta fatto, sul marker è possibile posizionare un oggetto virtuale che ne segua gli spostamenti. Tuttavia, possono sorgere problemi di lettura dovuti alla lontananza dall'identificatore e a quanto si è defilati rispetto ad esso. Inoltre, questa tecnica di tracking non è affatto dinamica, in quanto dipende interamente da marker già presenti e noti: per questo, esistono tecniche di tracking visivo che analizzano le immagini identificando gli oggetti, deducono le caratteristiche di questi (come i bordi e gli spigoli) e le confrontano con quelle in memoria. Alcune tecnologie, come Metaio, permettono

addirittura di creare una mappa 3D con i particolari dell'ambiente o di un oggetto rilevati, da usare poi come riferimento per il tracking. Ovviamente, queste tecniche richiedono algoritmi molto più complessi rispetto a quelle basate sui marker, e sono più esposte al rischio di falsi positivi.

Naturalmente, una tecnica non esclude l'altra e quindi se ne possono utilizzare più contemporaneamente, sfruttando i vantaggi di ognuna per colmare le mancanze delle altre. Si parla in questo caso di *tracking ibrido*, ed è in questa direzione che guardano progettisti e sviluppatori.

Inoltre, gli algoritmi di tracking possono richiedere un elevato uso delle risorse. Ciò può compromettere il risultato, che dovrebbe essere offerto in un tempo minimo in modo da garantire un'esperienza di realtà aumentata in tempo reale. Quindi, si può creare un sistema distribuito, nel quale i dati rilevati vengono analizzati da una o più altre macchine, che possono occuparsi anche della problematica di registration, sfruttando librerie grafiche e motori di rendering.

4.2 Interazione uomo-macchina

Gran parte dell'utilità di un calcolatore elettronico deriva da quanto gli utenti possano interagirvi e, soprattutto, da come possano farlo. Per quanto riguarda la realtà aumentata, abbiamo già visto come per monitor e dispositivi l'attenzione si sia rivolta verso gli smartglass, ma, per quanto riguardo il mezzo di interazione, la discussione è totalmente aperta.

Nel paragrafo sugli Epson Moverio BT-200, abbiamo accennato a come i progettisti abbiano deciso di dotare gli occhiali di una *scatola di controllo*, dotata di un touchpad e collegata tramite filo, che consente di interagire con l'interfaccia grafica di Android, come si fa normalmente con un notebook. Seppure sia un metodo efficace, che consente di utilizzare appieno il software, si basa su un concetto di interfaccia e

sistema di controllo nato decenni fa per usare macchine differenti e, inoltre, in questo particolare ambito si rivela poco immediato e pratico. Oltre a ciò, tenere costantemente in mano un secondo dispositivo, per di più collegato tramite filo, è piuttosto ingombrante.

Lo stesso tipo di interazione può essere offerta, in maniera più intuitiva e meno invasiva, dagli *smartwatch*. Considerando come Apple abbia reso un fenomeno di massa lettori MP3 prima, e smartphone e tablet poi, con iPod, iPhone e iPad, non è escluso che il lancio di *Apple Watch* nel 2015 valorizzi questa nuova tipologia di wearable device, con possibili conseguenze anche sulla realtà aumentata.

Per liberare comunque gli smartglass dai comandi fisici, Google con i suoi Google Glass ha puntato sui *comandi vocali*, che comunque non possono essere impartiti in qualsiasi situazione e possono soffrire del rumore dell'ambiente. Gli occhiali vantano anche un touchpad montato sull'asta destra.

La ricerca, in ogni caso, punta a trovare la soluzione migliore per liberare i consumatori da dispositivi di controllo da tenere saldi in mano, magari usando le mani stesse come mezzo di input. Sfruttando l'intuitività della gesticolazione, potrebbero essere utilizzati sistemi di rilevazione dei movimenti come il *Kinect* di Microsoft. Oppure, sono molteplici i casi di device realizzati come guanti: ad esempio, il *Zerkin Glove* si estende fino alla spalla e rileva in maniera fedelissima i movimenti dell'intero braccio, con il suo costo di produzione che si aggira sui 300$. Come hanno già fatto altre aziende, ad esempio Fujitsu, anche Google si sta muovendo nella direzione dei guanti.

Inoltre, c'è anche chi dedica la propria ricerca al controllo tramite gli occhi, in maniera analogo a come col Samsung Galaxy S4 è possibile scorrere i menù semplicemente spostando lo *sguardo*.

Tuttavia, nonostante tutte queste tecniche, risulta difficile l'immissione di testo: solo tramite comandi vocali può essere facile, ma poco adatto alla privacy. Quindi, potrebbe essere in ogni caso necessario un collegamento con un altro dispositivo, magari uno smartphone. Altrimenti, il progetto *SixthSense* utilizza un piccolo proiettore per

mostrare un'interfaccia grafica sulle superfici fisiche, navigabili tramite i movimenti delle dita, che vengono tracciate mettendo dei marker sulle unghie. Per scrivere qualcosa, una tastiera virtuale viene proiettata su una mano, e con la mano libera si "cliccano" le lettere. Un progetto simile a questo è *MARISIL* dell'Università di Oulu, ma in questo caso non ci sono proiettori e marker e la tastiera viene vista sulle mani attraverso lo schermo del dispositivo di realtà aumentata. Tuttavia, questo progetto sembra essere stato accantonato.

Fig.19, Apple Watch

Fig. 20, Zerkin Glove e il suo ideatore

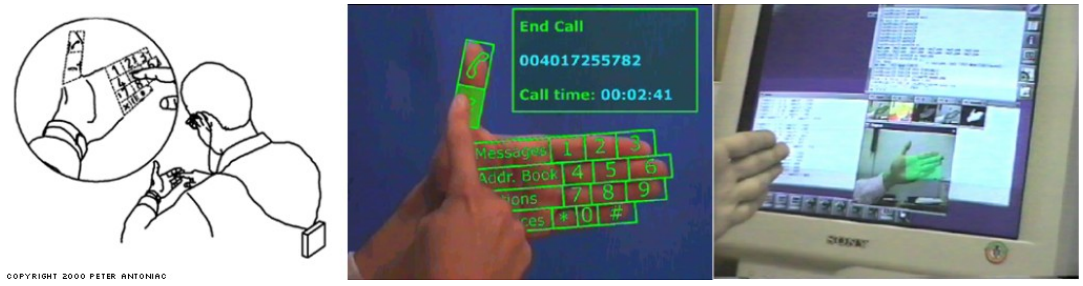
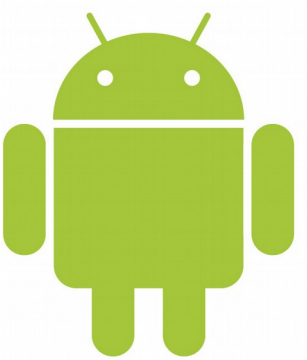


Fig. 21, il progetto MARISIL

5. Android: panoramica

In questa parte, verrà fornita una panoramica sul famoso sistema operativo Android. Dopo una sua descrizione generale con cenni storici, si entrerà più nel dettaglio illustrandone l'architettura, strato per strato. Verrà poi dato uno sguardo a com'è fatta una classica applicazione sviluppata per Android e a come vengono gestiti i processi, mentre, infine, saranno presentate API native che possono rivelarsi utili per creare applicazioni di realtà aumentata.

5.1 Che cos'è



Android è un sistema operativo ideato per essere utilizzato su sistemi mobile: è stato progettato in particolare per smartphone, ma col tempo si è diffuso anche su tablet, smartglass, netbook, smartwatch, smart-TV e addirittura automobili. Ad oggi, più dell'84% degli smartphone in commercio montano Android, e tale successo è dovuto al fatto che si tratta di un software libero, ovvero che i suoi sorgenti sono distribuibili e modificabili a piacimento. Ma a decretare il successo del sistema operativo è stato anche il supporto degli sviluppatori, complice il fatto che le applicazioni vengano realizzate per lo più in *Java*, un linguaggio di programmazione ad alto livello orientato agi oggetti, anch'esso tra i più diffusi nel suo genere.

Attualmente, la versione più recente di Android è la 5.0, denominata *Lollipop*, mentre i primi sviluppi sono cominciati nel 2003 dalle mani dell'azienda Android Inc., in California. Due anni più tardi tale ditta è stata acquistata da Google, desiderosa di entrare nel settore mobile, ma solo nel novembre del 2007 è stata mostrata la prima versione del sistema operativo.

2.2 Architettura

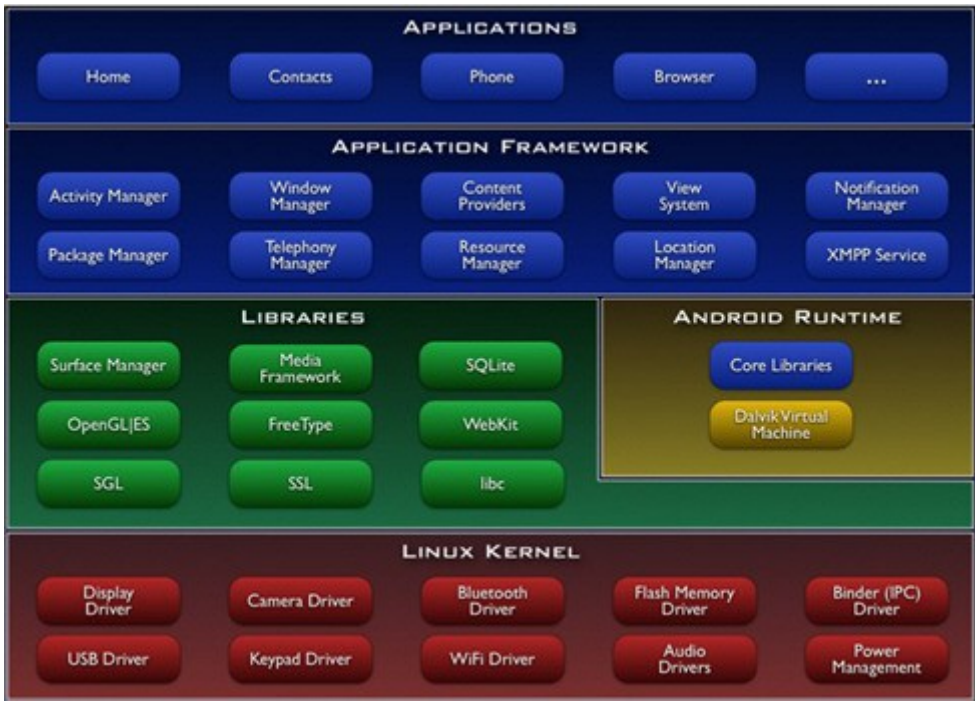


Fig. 23, l'architettura a strati di Android

Al livello più basso dell'architettura di Android risiede il kernel di *Linux* in versione 2.6, al quale Google ha apportato leggere modifiche, per lo più legate alla comunicazione tra processi e alle gestione della batteria del dispositivo. Tale scelta deriva non solo dalla necessità di disporre di uno strato attraverso il quale comunicare con l'hardware, che avviene tramite i numerosi driver, ma anche grazie alla gestione dei processi, della memoria e dei permessi che Linux è in grado di garantire.

Subito sopra al kernel si trovano le *librerie native*, scritte in C e C++, che forniscono agli strati superiori le funzionalità fondamentali. Nello stesso piano abbiamo anche la *Dalvik Virtual Machine*, ovvero una versione modificata della Java Virtual Machine, pensata per girare su hardware meno performanti (nonostante l'incredibile sviluppo tecnologico avvenuto recentemente in ambito mobile) e sfruttare al meglio quanto offerto dal livello inferiore, con particolare attenzione rivolta all'offrire il minor ingombro di memoria possibile. A suo

supporto vi sono le librerie che ricreano ed estendono opportunamente le API fondamentali di Java.

Al penultimo livello risiedono invece le applicazioni di sistema, mentre nello strato più alto troviamo le applicazioni utente.

5.3 Applicazioni e processi

Sebbene Android offra agli sviluppatori la possibilità di creare applicativi in C e C++ sfruttando le librerie native (secondo livello dell'architettura, partendo dal basso), garantendo migliori prestazioni a fronte di un minor utilizzo di risorse, il linguaggio di programmazione su cui Google ha puntato è il *Java*, ed è con questo che viene realizzata gran parte delle applicazioni.

Mentre, quindi, le funzionalità di un programma vengono scritte in Java, Android offre la possibilità di realizzarne l'interfaccia utilizzando il linguaggio *XML*. Ogni applicazione deve anche essere accompagnata da un file *manifest*, sempre in XML, che contiene informazioni sull'applicazione stessa, come, ad esempio, l'elenco delle librerie aggiuntive che sfrutta (oltre alla core library) o i permessi che richiede. Altre informazioni che contiene il manifesto riguardano i componenti dell'applicazione e quali processi le ospiteranno: infatti, un'app per Android non ha un metodo *main()*, ma è formata da diverse componenti che vengono eseguite quando necessario e vengono poi terminate. Tali componenti si dividono in:

- *Activities*: l'interfaccia grafica che permette all'utente di impartire comandi e visualizzare gli output;
- *Services*: i componenti che vengono eseguiti costantemente in background;
- *Broadcast Receivers*: componenti che attendono specifici segnali dal sistema operativo o da altre parti dell'applicazione, e che si attivano nel momento in cui questi arrivano;

- *Content Providers:* componenti che gestiscono la condivisione di dati tra applicazioni diverse.

Tutto il codice, le risorse e i dati di un'applicazione sono contenuti in un file *apk*. Ognuna viene eseguita in un proprio processo Linux, che viene creato quando avviene la richiesta di esecuzione dell'app. Inoltre, ogni processo, identificato da un ID univoco, ha una propria istanza della Dalvik Virtual Machine. Questo è stato realizzato in maniera che il codice di ogni applicazione venga eseguito in maniera isolata da quello degli altri, ed in modo che ogni processo non possa accedere direttamente alle risorse altrui.

5.4 API core utili per la Realtà Aumentata

Seppure la library core di Android non offra API pensate per la realtà aumentata, nella libreria sono comunque presenti primitive per la gestione della fotocamera, del GPS e degli altri sensori, che possono rivelarsi utili per creare applicazioni di realtà aumentata.

Tuttavia, in questo caso, è necessario che il programmatore si occupi personalmente della gestione dei sensori in ogni dettaglio e di problematiche come tracking e registration: è per questo, quindi, che per sviluppare vere e proprie applicazioni di realtà aumentata è fortemente consigliato utilizzare un SDK specifico per questa finalità, come, ad esempio, Metaio (al quale è dedicato il prossimo capitolo).

5.4.1 Fotocamera

Per controllare la fotocamera del dispositivo, Android mette a disposizione la classe *Camera*. Basterà chiamare il metodo statico *open()* al suo interno, passandogli, come parametro, l'identificatore della fotocamera di cui vogliamo prendere il controllo. Questo metodo lancia un'eccezione nel caso in cui la fotocamera richiesta sia già utilizzata da un'altra applicazione.

Per vedere sullo schermo l'anteprima della fotocamera, bisogna invocare il metodo *setPreviewDisplay()*, fornendogli come parametro la *SurfaceHolder* dove vogliamo vedere l'anteprima. Infine, è sufficiente richiamare *startPreview()*.

La classe *Camera* offre anche vari metodi per configurare la nostra istanza della fotocamera. Ad esempio, per impostare la dimensione dell'anteprima, dobbiamo ottenere i settaggi dalla classe *Camera* richiamando il metodo *getParameters()*, che restituisce un oggetto *Parameters*. Impostandone larghezza e altezza con *setPreviewSize()* passandogli i due valori numerici, sarà poi sufficiente trasferire l'istanza modificata della classe *Parameters* a *Camera*, tramite il metodo *setParameters()*.

È anche possibile impostare l'orientamento del display invocando il metodo *setCameraDisplayOrientation()* sul nostro oggetto di tipo *Camera*.

È bene precisare, in ogni caso, che la classe *Camera* è divenuta deprecata nell'ultima e recente revisione delle API (API Level 21). Al momento, Google consiglia di usare la nuova *Camera2*, affiancata dal package *Camera2.params*.

5.4.2 GPS

Per recuperare la posizione del dispositivo attraverso il GPS, è necessario prima di tutto istanziare un oggetto *LocationManager* richiamando il metodo *Context.getSystemService()*, passandogli la costante *Context.LOCATION_SERVICE* come parametro.

Bisognerà poi creare un oggetto *Criteria* per decidere come selezionare il provider che ci fornirà la posizione, impostando i parametri attraverso diversi metodi (come, ad esempio, *setAccuracy()* o *setSpeedRequired()*). Invocando poi il metodo *getBestProvider()* del nostro oggetto *LocationManager*, fornendogli i criteri definiti in precedenza, otterremo come valore di ritorno una *String* da trasferire al metodo *getLastKnownLocation()*, sempre appartenente all'istanza di

LocationManager. Questo ci restituirà un oggetto *Location*.

In seguito, sarà necessario dare un corpo al metodo *onLocationChanged()* dell'interfaccia *LocationListener*. Qui dovremo richiamare i metodi *getLatitude()* e *getLongitude()* sul precedente oggetto di tipo *Location*, passato al metodo come parametro, per ottenere i dati desiderati.

5.4.3 Altri sensori

L'SDK di Android fornisce le API per gestire un numero molto elevato di sensori. Questi sono descritti da un valore costante, univoco per ognuno, tutti presenti nella classe *Sensor*. Tra i più interessanti per il caso di studio della realtà aumentata, vi sono:

- *TYPE_ACCELEROMETER*;
- *TYPE_GYROSCOPE*;
- *TYPE_LIGHT*;
- *TYPE_PROXIMITY*;
- *TYPE_MAGNETIC_FIELD*.

Per riconoscere i sensori presenti nel dispositivo, è necessario creare un oggetto di tipo *SensorManager* richiamando il metodo *Context.getSystemService()* e passandogli *Context.SENSOR_SERVICE* come parametro. Invocando sulla nuova variabile il metodo *getSensorList()* con *Sensor.TYPE_ALL* come argomento, otterremo una lista *List<Sensor>* contenente tutti i sensori montati nel device. Sostituendo a *TYPE_ALL* una specifica costante, ad esempio una di quelle elencate poco più sopra, otterremmo, invece, la lista di sensori di quel determinato tipo.

La classe *Sensor* offre anche diversi metodi per ottenere informazioni sullo specifico sensore, come *getMaximumRange()* e *getMinDelay()*.

Sarà poi necessario implementare il metodo *onSensorChanged()* dell'interfaccia *SensorEventListener*, che viene invocato quando è cambiato il valore del sensore. Le informazioni saranno contenute nell'oggetto *SensorEvent,* che il metodo accetta come parametro, e saranno recapitate a tutti gli oggetti di tipo *SensorManager* registrati all'ascoltatore.

6. Approfondimento su Metaio SDK per Android

Metaio SDK è tra i più interessanti e completi SDK per la realtà aumentata attualmente disponibili. Data l'esistenza di una versione specifica per Android, presente anche in versione gratuita, è stato scelto come approfondimento per questa tesi.

6.1 Introduzione

Metaio SDK si propone di nascondere allo sviluppatore i dettagli dell'implementazione dei metodi che fornisce, occupandosi quindi lui stesso dei problemi di tracking, registration, rendering e gestione dei sensori, lasciando al programmatore il solo pensiero di dedicarsi a progettare la propria applicazione. Per fare questo, è composto di quattro componenti: *capturing component*, *sensor interface component*, *rendering component* e *tracking component*. L'interazione tra l'applicazione e i componenti è affidata alla *Metaio SDK interface*.

IMetaioSDK è l'interfaccia principale per la gestione delle attività di realtà aumentata, mentre *IGeometry* è l'interfaccia principale per la realizzazione visiva degli elementi virtuali mostrati su schermo.

Nel caso specifico di Android, ogni applicazione realizzata con Metaio è una classe che eredita dalla classe base *ARViewActivity*, contenuta nello stesso progetto dell'app. Quando quest'ultima viene messa in esecuzione, le più importanti chiamate che articolano il tempo di vita del processo sono:

- *onCreate()*: avviene al momento del lancio dell'applicazione, quindi è qui che dovrebbe essere gestita dal programmatore l'inizializzazione dei componenti. Anche l'istanza di Metaio SDK e i sensori vengono preparati in questa fase;

- *onStart()*: viene invocata dopo *onCreate()* e quando l'applicazione è pronta per mostrare l'output all'utente. È qui che la fotocamera viene inizializzata e l'interfaccia grafica viene preparata;
- *onPause()*: l'istanza di Metaio SDK e l'interfaccia grafica vengono messe in pausa;
- *onResume()*: l'istanza di Metaio SDK e l'interfaccia grafica vengono rimesse in esecuzione;
- *onStop()*: l'interfaccia grafica viene rimossa, e non è quindi più visibile dall'utente;
- *onDestroy()*: l'istanza di Metaio SDK e i sensori "fittizi" vengono eliminati;
- *onDrawFrame()*: è il metodo chiamato per disegnare il frame del momento, dove Metaio SDK renderizza quanto richiesto.

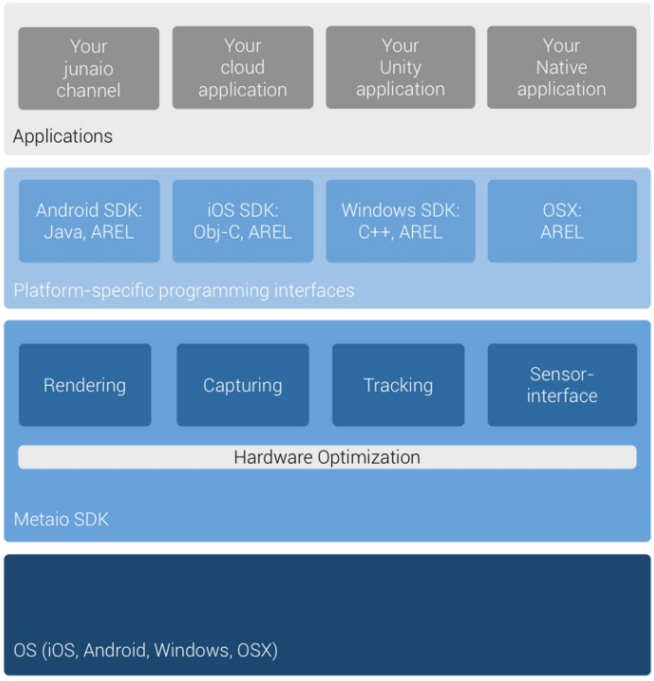


Fig. 24, il framework di Metaio SDK

6.2 Tracking

Metaio SDK offre tutto il necessario per gestire sia il tracking ottico, sia quello basato sui sensori. Nel primo caso, è supportata non solo la tecnica basata sui marker, ma anche quella che ne fa a meno.

Le strategie di tracking utilizzate da un'applicazione sono memorizzate in file XML denominati *configuration file*. Per scegliere un modo di tracking, basta invocare il metodo *setTrackingConfiguration()* dell'istanza di Metaio SDK, indicando il configuration file desiderato come parametro. Per avviare il tracking viene utilizzato il metodo *startInstantTracking()*. Passandogli la stringa *ISTANT_2D* come parametro, verrà creato al volo un file di configurazione che usa come marker il frame catturato in quell'istante.

Sono presenti poi diversi altri metodi che servono per il tracking. I principali di questi metodi sono:

- *pauseTracking()*;
- *resumeTracking()*;
- *getTrackingDuration()*;
- *getTrackingFrameRate()*;
- *getTrackingValues()*;

i cui nomi indicano chiaramente le loro specifiche utilità.

Un punto di forza di Metaio per la realizzazione del tracking è, inoltre, la tecnologia *SLAM*, acronimo di *Simultaneous Localization and Mapping*, che consente di localizzare la posizione della camera e i suoi movimenti mentre crea una rappresentazione 3D dell'ambiente circostante. Ne sono offerte due varianti:

- *3D SLAM Tracking*: permette di creare la mappa 3D senza l'utilizzo di alcun marker, ma non è possibile controllare la posizione e le dimensioni del contenuto virtuale da piazzare;
- *3D SLAM Extended Tracking*: è lo stesso algoritmo precedente, ma consente di posizionare e dimensionare il contenuto virtuale

grazie all'utilizo di un marker.

Sono caratterizzate da tre parametri da configurare nel configuration file:

- *MinTriangulationAngle*: è il numero minimo di angoli utili per triangolarizzare un punto;
- *SimilarityThreshold*: è usata per determinare se un punto è stato correttamente tracciato oppure no;
- *MinNumberOfObservation*: indica per quanto tempo un punto deve essere tracciato prima di poterlo triangolarizzare e inserire nella mappa 3D.

L'*Extended Tracking* si differenzia dal primo per due parametri aggiuntivi:

- *Initialization Type*: deve essere impostato a Markerless2D per identificarlo rispetto al tipo precedente;
- *ReferenceImage*: indica l'immagine che deve essere usata come marker e le sue dimensioni.

Per utilizzare la tecnologia SLAM, è sufficiente invocare il metodo *startIstantTracking()* descritto prima, con la stringa *INSTANT_3D* come parametro di ingresso.

6.2.1 Optical Tracking

Nel caso di tracking visivo basato sui marker, esistono due modalità differenti per realizzarlo, denominate *TrackingQuality*: il metodo *Fast* utilizza una soglia (*ThresholdOffset*) prefissata per binarizzare l'immagine ed è consigliato per situazioni con illuminazione stabile, mentre il metodo *Robust* adatta la soglia un certo numero di volte (*NumberOfSearchIterations*) ed è quindi utilizzato per situazioni dove l'illuminazione è variabile. Per i marker, invece, possono essere impostati due parametri, ovvero le dimensione in millimetri (*Size*) e l'identificatore univoco (*MatrixID*). Tutti questi parametri vanno registrati nel configuration file XML descritto precedentemente.

Anziché utilizzare un classico marker, è possibile usare un'immagine. Anche in questo caso esistono due *TrackingQuality*, sempre chiamati *Fast* e *Robust*, con il secondo che è pensato per immagini con texture così dettagliate da non venir tracciate dal primo metodo. I parametri da impostare nel file di configurazione sono, in questo caso:

- *FeautureDescriptorAlignment*: che serve per determinare l'allineamento del dispositivo ed il cui valore predefinito è *regular*, che è altamente consigliato di non modificare;
- *MaxObjectsToDetectPerFrame*: il numero massimo di oggetti planari da localizzare in un unico frame;
- *MaxObjectsToTrackInParallel*: il numero massimo di oggetti da tracciare contemporaneamente;
- *SimilarityThreshold*: una soglia per determinare se il tracking è avvenuto con successo o meno, che si consiglia di non cambiare dallo 0.7 di default;
- *ReferenceImage*: il percorso del file immagine da usare per il tracking (è possibile, ma sconsigliato, impostarne anche le dimensioni in millimetri con *WidthMM* e *HeightMM*);

L'immagine che si intende utilizzare come marker è necessario che abbia colori differenti e di diversa luminosità, alti contrasti e spigoli ben evidenti. Deve essere di risoluzione molto alta, e la sua stampa su materiale non riflettente deve essere il più fedele possibile alla controparte digitale.

Metaio SDK riconosce anche il volto umano, e permette di tracciarne la posizione nello spazio per al massimo un volto per frame. Per usare questa tipologia di tracking, basta passare la stringa *FACE* come parametro al metodo *setTrackingConfiguration()* dell'istanza di Metaio SDK. Il suo file di configurazione sarà simile a quello di un tracking tramite immagine.

Per quanto riguarda, invece, l'optical tracking senza marker, Metaio consente di utilizzare una mappa 3D di punti di un oggetto reale

scelto come riferimento per il tracking. Tale mappa può essere creata dall'SDK, ma anche tramite l'applicazione Metaio Toolbox. In questo caso, il file di configurazione conterrà i parametri:

- *MinMatches*: un valore numerico che specifica l'accuratezza del tracking;
- *NumExtensibleFeatures*: il numero massimo di nuovi punti che si vogliono aggiungere alla mappa 3D già generata;
- *MinTriangulationAngle*: è il numero minimo di angoli utili per triangolarizzare un nuovo punto, nel caso se ne vogliano aggiungere di nuovi alla mappa.

È anche possibile realizzare il tracking di un ambiente senza marker, partendo da un oggetto fisico presente di cui si ha un modello CAD tridimensionale. Si tratta della funzionalità più complessa dell'intero SDK.

Per la calibrazione della fotocamera, invece, Metaio ha integrato la funzionalità in Toolbox, e il risultato viene salvato in un file XML. Il percorso di suddetto file dovrà essere passato al metodo *setCameraParameters()* dell'istanza di Metaio SDK, insieme ad una costante appartenente all'enum *ECAMERA_TYPE* per indicare se la camera verrà utilizzata per il tracking, il rendering o entrambi (rispettivamente, *ECT_TRACKING*, *ECT_RENDERING* e *ECT_ALL*). Gli altri metodi dell'interfaccia *IMetaioSDK* per la gestione della fotocamera sono:

- *getCameraParameters()*;
- *startCamera()*;
- *stopCamera()*;
- *getCameraImage()*;
- *getCameraFrameRate()*;
- *setStereoRenderingLeftCameraTransformation()*, s*etStereoRenderingRightCameraTransformation()*: impostano rispettivamente la trasformazione di rendering per la camera

sinistra e quella destra del dispositivo capace di mostrare immagini tridimensionali.

Tali metodi richiedono un'istanza della classe *Camera*, che rappresenta il dispositivo fisico. Per ottenere una lista di tutte le camere del dispositivo, c'è il metodo *getCameraList()*.

6.2.2 Non-Optical Tracking

Metaio SDK fornisce allo sviluppatore anche i mezzi necessari per implementare il tracking non ottico. Al momento, i sensori supportati dal framework per la gestione di questo tracking sono il GPS, l'accelerometro, il giroscopio e il sensore magnetico.

Per attivare questa configurazione, sarà sufficiente invocare il solito metodo *setTrackingConfiguration()* passandogli, stavolta, la stringa *GPS*. Per utilizzare solo accelerometri e giroscopi, invece, la stringa da passare è *ORIENTATION*.

I parametri che caratterizzano il nuovo file di configurazione sono:

- *LocalProvider*: impostando *none*, verrà disabilitato il GPS;
- *IgnoreCompass*: se impostato con *true*, non verrà utilizzato il sensore magnetico;
- *HandEyeCalibration*: permette di specificare la posizione e l'orientamento di un sensore rispetto ad un altro;
- *TranslationOffset*: imposta il vettore 3D per la traslazione;
- *RotationOffset*: imposta l'offset per la rotazione;
- *COSOffset*: imposta il sistema di coordinate per poter muovere il contenuto virtuale in output.

Un sensore è rappresentato dall'interfaccia *ISensorsComponent*, che offre i metodi necessari per attivarli, fermarli e leggerne i risultati. I valori dei sensori sono descritti dalla classe *SensorValues*. Metodi per controllare i sensori sono forniti anche dall'interfaccia *IMetaioSDK,* in

particolare *sensorCommand()* che invia al sensore desiderato il comando indicato come primo parametro.

6.3 Content Creation

Metaio SDK consente di rendere immagini, video e modelli 3D dei contenuti virtuali in un'applicazione di realtà aumentata.

I formati immagine supportati sono jpg, png e bmp, e non ci sono particolari vincoli sulla sua risoluzione. Invece, gli unici filmati supportati sono quelli con codifica MPEG4 dentro ad un contenitore 3G2, e devono essere rispettate anche particolari condizioni sulla risoluzione del video e su compressione e risoluzione dell'audio. Per quanto riguarda il modelli tridimensionali, sono supportati i formati OBJ, MD2 e FBX e, per garantire ottime prestazioni, il numero di poligoni non deve essere troppo elevato e le texture non devono superare la risoluzione di 2048x2048.

Per creare un'immagine, bisogna prendere la classica istanza di Metaio SDK e invocare il suo metodo *createGeometryFromImage()*, passandogli il percorso dell'immagine. Essa verrà inserita in un piano tridimensionale, e potrà essere rimossa con *unloadGeometry()*. Per i video vale lo stesso discorso, con la differenza che il metodo per crearli è *createGeometryFromMovie()*, mentre per i modelli 3D il metodo è semplicemente *createGeometry()*. In tutti i casi, viene restituito un oggetto di tipo *IGeometry*. Questa è l'interfaccia principale per la modellazione di contenuto di realtà aumentata. Essa presenta numerosissimi metodi, tra cui *setTranslation()*, *setScale()*, *setRotation()* e *setVisible()*.

Nel caso particolare di un modello 3D, alcuni metodi importanti sono:

- *setTexture()*;
- *setMovieTexture()*: differisce dal precedente in quanto rende un

video, e non un'immagine, una texture;

- *startAnimation()*: fa partire l'animazione indicata come parametro di tipo *String*;
- *onAnimationEnd()*: fa partire un'altra animazione dopo che un'altra, indicata, è conclusa.

Inoltre, dato un modello 3D renderizzato (del rendering se ne occupa direttamente l'SDK), per farlo sembrare appartenente alla realtà fisica può essere necessario anche applicargli effetti di luce e ombre. Il metodo *createLight()* dell'interfaccia *IMetaioSDK* crea un oggetto di tipo *ILight*,. Quest'ultima interfaccia offre svariati metodi, ad esempio per impostarne colore ed attenuazione, e col metodo specifico *setType()* si può scegliere il tipo di luce che si desidera tra i tre possibili:

- *ELIGHT_TYPE_DIRECTIONAL*: va lungo una sola direzione ed è infinita;
- *ELIGHT_TYPE_POINT*: nasce in un punto e va in qualsiasi direzione;
- *ELIGHT_TYPE_SPOT*: ha una direzione ed è a forma di cono.

Una volta impostata una luce, le ombre saranno generate di conseguenza.

Per quanto riguarda gli effetti di una superficie, invece, tramite alcuni tool deve essere creato uno speciale file di configurazione XML, da caricare poi con il metodo *loadShaderMaterials()* dell'interfaccia *IMetaioSDK*.

7. Conclusioni

Le potenzialità della realtà aumentata sono elevatissime, e le interfacce grafiche che abbiamo visto e vediamo in film e videogiochi fantascientifici sembrano destinate a diventare realtà. Tuttavia, la tecnologia attuale non consente di realizzare quanto voluto nella maniera desiderata e, soprattutto, i device mobile come gli smartglass hanno un costo ancora troppo elevato per il pubblico, che incrementerebbe ulteriormente se gli venissero affiancati dispositivi di controllo come i guanti attualmente in studio. Quindi, per il momento, siamo solo all'alba di quella che si rivelerà una rivoluzione tecnologia.

Nel frattempo, in tale scenario, il sistema operativo Android ha imboccato la giusta strada per diventare un punto di riferimento in tale contesto, come già gli è riuscito nel mercato degli smartphone e dei tablet. Questo anche perché si propone come una valida base per l'utilizzo di SDK appositi per la realtà aumentata, che permettono agli sviluppatori di programmare le proprie applicazioni ad un più alto livello.

Tra quest'ultimi ne spiccano diversi, come ARToolKit e Wikitude, accennati nel corso della tesi. Tuttavia, quello che al momento offre il pacchetto più completo ed efficace sembra essere Metaio, in ambiente Android e non solo.

8. Ringraziamenti

Ringrazio tutti gli autori e i proprietari dei testi che ho consultato per poter scrivere questa opera di ricerca, nonché gli autori e i proprietari delle immagini che ho inserito nel testo. Senza di loro, il mio lavoro non sarebbe stato possibile.

Ringrazio, inoltre, chiunque abbia avuto l'interesse, la voglia e la pazienza di leggere, anche solo in parte, questa mia tesi di laurea triennale.

9. Bibliografia

[1] P. Milgram, A. F. Kishino, (1994). *"Taxonomy of Mixed Reality Visual Displays"*.

[2] R. T. Azuma, (1997). *"A Survey of Augmented Reality"*.

[3] F. Biocca, M. R. Levy, (1995). *"Communication in the age of virtual reality"*.

[4] World of Computer Science Biography, (2009). *"Ivan Sutherland Biography"*.

[5] T. H. Höllerer, S. K. Feiner, (2004). Mobile Augmented Reality. *"Telegeoinformatics: Location-Based Computing and Services"*.

[6] L. B.Rosenberg, (1993). *"The use of virtual fixtures to enhance operator performance in time delayed teleoperation"*.

[7] R. Raskar, G. Welch, H. Fuchs, (1998). *"First International Workshop on Augmented Reality"*.

[8] J. Rekimoto, (1998). *"Augmented Reality using 2D matrix code"*.

[9] H. Kato, M. Billinghurst, (1999). *"The 2nd IEEE and ACM International Workshop on Augmented reality"*.

[10] B. Thomas, B. Close, J. Donoghue, J. Squires, P. D. Bondi, W. Piekarski, (2001). *"Personal and Ubiquitous Computing"*.

[11] T. Reicher, A. MacWilliams, B. Brügge, (2004). *"Towards a System of Patterns for Augmented Reality Systems"*.

10. Sitografia

[1] https://it.wikipedia.org/

[2] https://en.wikipedia.org/

[3] http://www.epson.com/cgi-bin/Store/jsp/Landing/moverio-bt-200-smart-glasses.do

[4] https://www.google.com/glass/start/what-it-does/

[5] http://www.hitl.washington.edu/artoolkit/

[6] http://www.metaio.com

[7] http://www.wikitude.com

[8] http://zerkinglove.wordpress.com

[9] http://www.pranavmistry.com/projects/sixthsense/

[10] http://marisil.org

[11] http://developer.android.com/index.html

[12] https://dev.metaio.com/sdk/

11. Elenco delle figure

[1] http://nerdbastards.com/wp-content/uploads/2014/01/bttfjaws.jpg

[2] http://www.total-ltd.co.uk/blog/wp-content/uploads/2014/10/CardboardOpen.jpg

[3] http://sweetar.com/blogg/wp-content/uploads/VirtualityCont.png

[4] http://settore.myblog.it/media/01/02/3783390443.jpg

[5] http://intelligentheritage.files.wordpress.com/2011/09/sensorama.jpg

[6] https://triviahappy.com/images/articles/03252014sword.jpg

[7] http://teamsiems.com/wp-content/uploads/2009/10/PennStateAR.gif

[8] http://www.extremetech.com/wp-content/uploads/2011/08/wikit5.jpg

[9] http://wearables.unisa.edu.au/wp-content/uploads/2010/05/quake4.jpg

[10] http://www.buzzstate.com/media/uploads/Augmented-Reality.png

[11] http://www.aidmo.org/beta/images/stories/augmented_reality.jpg

[12] http://i.i.cbsi.com/cnwk.1d/i/tim/2012/10/28/fmimg7308166078529320734.jpg

[13] https://www.in.tum.de/typo3temp/pics/08048f35c1.jpg

[14] http://static.trustedreviews.com/94/368fd5/4219/4241-graw2b.jpg

[15] http://www.skyscanner.net/sites/default/files/image_import/wikitude.jpg

[16] http://ecx.images-amazon.com/images/I/61yzLGO6ROL._SL1500_.jpg

[17] http://www.epson.jp/products/moverio/bt200/images/index/ph_visual.jpg

[18] http://ceur-ws.org/Vol-91/paperE4.pdf (pag. 2)

[19] https://www.apple.com/v/watch/a/images/og_apple_watch.jpg?201411240829

[20] http://zerkinglove.files.wordpress.com/2009/09/noah-and-glove-on-terrace.jpg

[21] http://marisil.org/etc/pics/marisil/about-big.png

[22] http://www.android.com/media/android_vector.jpg

[23] http://i.zdnet.com/blogs/android-architecture-485b.jpg

[24] https://dev.metaio.com/typo3temp/pics/b6a33362d1.png