

UNIVERSITÁ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI MATEMATICA

Corso di Laurea in Informatica

Realtà aumentata e Apple iOS:

l'informazione tradizionale incontra la terza dimensione

Laureando

Nicolas Casartelli

Relatore

Prof.ssa Silvia Crafa

Co-relatore

Prof. Amir Baldissera

ANNO ACCADEMICO 2012/2013

La realtà non è digitale, è analogica.

Non è una cosa che c'è o non c'è, ma è qualcosa di graduale. In altri termini, la realtà è una qualità delle cose, come il peso. [...]

Mondo Disco è il più irreale possibile, e tuttavia abbastanza reale da esistere. E abbastanza da trovarsi in un mare di guai.

Terry Pratchett, Stelle Cadenti



Indice

1	Introduzione	7
1.1	Presentazione dell'azienda	7
1.2	Descrizione preliminare del progetto	8
1.3	Dominio tecnologico del progetto	9
1.3.1	Apple iOS	9
1.3.2	Realtà aumentata	9
1.3.3	3D Modeling e 3D Rendering	10
1.4	Strumenti di sviluppo	11
1.4.1	IDE: Xcode	11
1.4.2	SDK per la realtà aumentata: Vuforia	12
1.4.3	Dispositivi utilizzati	12
2	Primi passi nella realtà aumentata	15
2.1	Principi base di Vuforia	15
2.1.1	Image target	16
2.1.2	Frame marker	17
2.1.3	Multi-target	17
2.1.4	Virtual buttons	18
2.2	Vuforia samples	18
2.2.1	Struttura generale di Vuforia per iOS: ARCommons	19
2.2.2	Video playback	21

3	Gestione della memoria	23
3.1	MRR e ARC: soluzioni diverse per lo stesso problema	24
3.1.1	Manual retain-release	24
3.1.2	Automatic reference counting	25
3.1.3	La soluzione di Digitalic	25
4	Gestione dell'interfaccia grafica	29
4.1	Due modi diversi per gestire l'interfaccia	30
4.1.1	Gestione programmatica delle view	30
4.1.2	Interface Builder e Nib/Xib files	30
4.1.3	La gestione dell'interfaccia in Digitalic	31
5	3D Modeling e 3D Rendering	33
5.1	Concetti base di modellazione 3D e di rendering 3D	34
5.2	Come si crea un oggetto 3D	35
5.2.1	La modellazione 3D con Blender	35
5.2.2	Rendering su dispositivi mobili: OpenGL ES 2.0	35
5.2.3	Dalla modellazione al rendering	37
6	Deployment su App Store	39
6.1	Il processo di approvazione	40
6.1.1	L'approvazione di Digitalic	41
7	Conclusioni	43
7.1	Statistiche di download	43
7.2	iOS e Android: un breve confronto	44
7.3	Possibili miglioramenti dell'app	44
	Bibliografia	44
	Ringraziamenti	47

```
<html>
  <head>
    <meta name="TITLE" content="www.yourdomain.com" />
    <meta name="KEYWORDS" content="your keywords" />
    <meta name="DESCRIPTION" content="your description" />
    <link rel="stylesheet" type="text/css" href="style.css" />
    <script language="javascript" type="text/javascript">
      document.write("#fffff");
    </script>
  </head>
  <body>
    <h1>Your Page Title</h1>
    <p>Your page content goes here</p>
  </body>
</html>
```

1 — Introduzione

Sommario

- Presentazione dell'azienda**
Descrizione preliminare del progetto
Dominio tecnologico del progetto

Apple iOS
Realtà aumentata
3D Modeling e 3D Rendering

Strumenti di sviluppo

IDE: Xcode
SDK per la realtà aumentata: Vuforia
Dispositivi utilizzati

La seguente tesi intende essere una relazione circa l’attività di stage effettuata da Nicolas Casartelli presso l’azienda Mentis s.r.l. a partire dal 04/07/2012 fino al 04/09/2012. L’attività di stage si è concentrata sullo studio e successiva realizzazione di una piattaforma di realtà aumentata per dispositivi Apple iOS attraverso l’utilizzo del framework Vuforia sviluppato da Qualcomm. In particolare, è stato richiesto un approfondimento circa l’utilizzo di oggetti 3D nell’ambito della realtà aumentata e successivamente di dedicare uno spazio di dimensioni inferiori all’integrazione di video in realtà aumentata.

1.1 Presentazione dell'azienda

L'attività dello stage è stata svolta presso l'azienda **Mentis s.r.l.**, la quale offre servizi di consulenza strategica e realizzazione di progetti innovativi.

Nasce nel 2004 come spin off dell'operatore di telecomunicazioni veneto Trivenet Spa, in quanto è legata ai differenti modelli di business delle proprie aree strategiche. Infatti, mentre la divisione dedicata alle telecomunicazioni - connettività e fonia - si muove alla ricerca di molti clienti intercambiabili tra loro, la divisione dedicata all'erogazione dei servizi (web e consulenza) si muove alla ricerca di pochi clienti con cui instaurare un legame di collaborazione o partnership a lungo termine. Deriva da ciò la scelta di creare Mentis per rafforzare la presenza sul mercato e focalizzare le strategie in modo più centrato sulla ricerca della giusta tipologia di clientela. Dalla nascita ad oggi, la crescita di Mentis è stata costante e i risultati raggiunti sono ben testimoniati dalle aziende clienti, come GE General Electric, Dell Computers, De Agostini Editore e Zucchetti Spa, realtà molto diverse tra loro: dalle multinazionali alle piccolissime aziende, dalla pubblica amministrazione alle piccole e medie, da realtà a forte internazionalizzazione a ditte radicate sul territorio.

Il servizio offerto è di **consulenza strategica**: partendo da un'analisi della realtà aziendale si evidenziano bisogni ed opportunità non sfruttate, da cui la definizione degli obiettivi da raggiungere. Mentis può, su richiesta del committente, progettare la soluzione con la quale perseguire tali obiettivi, tenendo conto sia della fattibilità tecnica della stessa sia dell'impatto organizzativo che può comportare. La fase di progettazione sarà seguita da quella di codifica delle soluzioni concordate con il committente e da quella di verifica dei risultati che la soluzione stessa ha comportato. Ai processi di analisi, progettazione, codifica e verifica è applicato il **sistema di qualità ISO 9001:2008** al fine di garantire un ottimo risultato già a partire dal processo che lo porterà in essere.



Figura 1.1: Logo dell'azienda Mentis s.r.l., azienda presso cui è stato svolto lo stage.

1.2 Descrizione preliminare del progetto



Figura 1.2: Logo di Digitalic, mensile di informazione per il settore IT business.

Il piano di lavoro compilato preventivamente allo stage richiedeva lo studio e la consecutiva realizzazione di una **applicazione mobile** (di seguito: app) per dispositivi iOS di realtà aumentata; in particolare, l'app doveva essere realizzata per aggiungere contenuti multimediali (video, immagini, oggetti 3D) ad una rivista cartacea, **Digitalic** (mensile di informazione per il settore IT business). Si richiedeva quindi che l'app fosse in grado di accedere alla fotocamera del dispositivo Apple su cui

fosse stata fatta funzionare e, mostrando sul display il contenuto stesso della camera, aggiungere ad esso le informazioni multimediali inerenti a ciò che la camera stessa mostra in quel dato momento. Il contenuto multimediale aggiunto doveva quindi essere strettamente dipendente dall'articolo inquadrato con la fotocamera in un dato momento e, per questo motivo, differenziabile e impostabile a priori. Era inoltre richiesto che

i contenuti stessi fossero scaricabili in maniera dinamica senza bisogno di aggiornare l’intera app ad ogni nuova uscita della rivista.

1.3 Dominio tecnologico del progetto

1.3.1 Apple iOS

iOS (precedentemente nominato iPhone OS) é un sistema operativo sviluppato da Apple. Si tratta di una derivazione di UNIX e viene utilizzato da tutti i dispositivi mobili di Apple, quali iPod, iPhone e iPad. Attualmente (maggio 2012), gli smartphone che utilizzano il sistema operativo iOS sono piú di 33 milioni ed é il secondo sistema operativo per dispositivi mobile piú diffuso al mondo. Per lo sviluppo dell’app si é fatto riferimento alla versione di iOS 5.1 che, pur non essendo la piú aggiornata alla data attuale, rappresenta la versione piú ampiamente utilizzata alla data di inizio dello stage. Lo sviluppo dell’app iOS é stato possibile grazie all’utilizzo di un SDK, (Software Development Kit) fornito direttamente da Apple, integrato completamente all’interno di XCode, IDE (Integrated Development Environment) anch’esso fornito direttamente da Apple. La struttura di questo SDK é costituita da strati successivi di framework, con i framework di livello piú alto che forniscono una maggiore astrazione rispetto all’hardware del dispositivo stesso, utilizzando a loro volta i framework degli strati inferiori. Nella figura 1.4 si possono vedere gli strati dell’SDK iOS.



Figura 1.3: Logo di iOS, sistema operativo mobile di Apple.

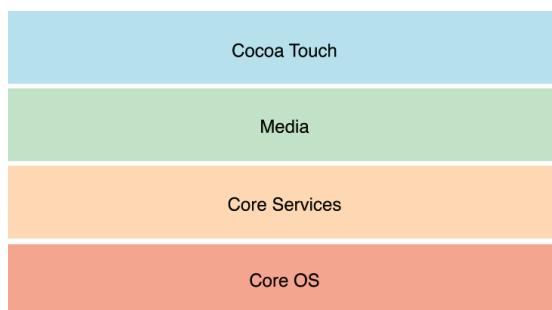


Figura 1.4: Livelli su cui si basa l’intero SDK di Apple iOS. I livelli piú alti sono quelli che forniscono API di livello maggiore.

1.3.2 Realtá aumentata

Con **realtá aumentata** si intende si intende l’arricchimento della percezione sensoriale umana mediante informazioni, in genere manipolate e convogliate elettronicamente, che non sarebbero percepibili con i cinque sensi. Attraverso questo arricchimento, infatti, l’utente che utilizza il sistema é in grado di entrare in contatto con determinate informazioni altrimenti non disponibili nell’ambiente circostante, “sovrapponendole”

alle normali informazioni percepibili. Un software di realtà aumentata necessita di determinati sensori in grado di recepire informazioni dalla realtà circostante l'utente, quali possono essere fotocamera, accelerometro, giroscopio, sensore GPS, microfono e altri; dopo aver elaborato propriamente le informazioni rilevate dai suddetti sensori, quindi, fornisce all'utente ulteriori informazioni (che possono essere presenti nella memoria del dispositivo utilizzato oppure ricavate da internet in real-time), solitamente attraverso un display o un altoparlante.



Figura 1.5: Esempio di realtà aumentata, tecnologia per veicolare informazioni aggiuntive alla realtà circostante.

Il termine realtà aumentata (augmented reality) è stato coniato nel 1990 dal professor Thomas P. Caudell, riferendosi ad un display digitale montato sulla testa (head-mounted display, HMD) in grado di guidare gli operai nell'assemblamento dei fili elettrici dei motori; tuttavia, è con la diffusione degli smartphone degli ultimi anni che il tema della realtà aumentata sta iniziando ad interessare un pubblico sempre più vasto, principalmente data la potenza dei dispositivi stessi e la loro semplicità di utilizzo.

1.3.3 3D Modeling e 3D Rendering

Con 3D modeling, o **modellazione 3D**, si intende il processo di creazione di una rappresentazione matematica di un oggetto tridimensionale attraverso l'uso di software apposito. Il risultato di questa operazione è chiamato modello 3D e può essere utilizzato in svariati modi, dalla riproduzione fisica dell'oggetto attraverso meccanismi di stampa 3D alla simulazione di fenomeni fisici. Particolare importanza nel corso dell'attività di stage ha ricoperto anche il processo di **3D rendering** corrispondente alla trasformazione di un oggetto solido 3D in una sua corrispondente rappresentazione bidimensionale (2D), attraverso anche l'utilizzo di algoritmi per il miglioramento del fotorealismo e simili.

Lo studio sulla modellazione 3D si è rivelato indispensabile all'interno dell'attività di stage, in quanto è emersa presto la necessità di approfondire determinati aspetti ad essa attinenti; in particolar modo, come verrà evidenziato anche in seguito, è stata data una particolare importanza alla complessità dei modelli 3D utilizzati, i quali, a causa di alcuni limiti imposti dalle tecnologie utilizzate, non potevano superare una determinata quantità di vertici e facce. Sono stati quindi necessari metodi di semplificazione dei



Figura 1.6: Esempio di una teiera di Newell, oggetto utilizzato spesso come esempio basilare nella modellazione 3D a causa della semplicità della sua realizzazione e complessità nel riuscire a rappresentare molti degli aspetti più interessanti del 3D rendering.

modelli 3D e, in alcuni casi, anche di creazione di modelli 3D a partire da zero.

Per quanto riguarda invece ciò che riguarda strettamente i dispositivi mobili, la tecnologia utilizzata è **OpenGL ES 2.0**, sottoinsieme delle librerie grafiche OpenGL rilasciate a marzo 2007 e compatibili con i principali sistemi operativi e device mobile, tra cui anche iOS e Apple iPhone 3GS e successivi. Queste librerie permettono di utilizzare API di alto livello per il rendering 3D a partire da informazioni estratte da modelli 3D, quali vertici, normali e coordinate texture; lo studio al riguardo nella fase di stage si è però concentrato sulle sue funzioni basilari, in quanto le librerie OpenGL ES 2.0 necessitano di una preparazione ad ampio spettro sulla modellazione 3D.

1.4 Strumenti di sviluppo

1.4.1 IDE: Xcode

Per lo sviluppo dell'app è stato utilizzato l'IDE (Integrated Development Environment) **XCode** versione 4.2, scaricabile gratuitamente dal sito Apple Developer, installata su Mac OS X versione 10.6.8 Snow Leopard. Si tratta del principale IDE per Apple, in grado di sviluppare applicazioni per iOS e Mac OS X, e comprende una serie di strumenti atti a facilitare il compito di sviluppo dell'app, i cui principali utilizzati sono **Instruments** e **OpenGL ES Frame Capture**.

Il primo è uno strumento di test e analisi delle performance dinamico utilizzato per tracciare un profilo di un'applicazione durante la sua esecuzione. Permette di tracciare svariati aspetti dell'app in maniera precisa ed esaustiva, a seconda degli aspetti su cui ci si intende focalizzare durante il suo sviluppo; ognuno di questi aspetti che si intende viene chiamato instrument, e il software permette di aggiungere un numero variabile di instrument in modo da eseguire più misurazioni contemporaneamente. Alcuni degli instrument utilizzati sono: *memory leak*, *OpenGL ES driver*, *file I/O*



Figura 1.7: Logo di XCode, ambiente di sviluppo realizzato da Apple per permettere lo sviluppo di applicazioni per dispositivi Apple.

usage.

Il secondo strumento, OpenGL ES Frame Capture, é stato utilizzato come strumento di debug per il rendering 3D sul dispositivo. Si tratta di un’opzione che, nel momento in cui viene attivata, traccia tutte le chiamate eseguite dalla CPU alla GPU nell’azione di rendering di un singolo frame, compresi gli accessi alla memoria della GPU stessa e l’impostazione dei parametri OpenGL ES. Si é rivelato quindi uno strumento indispensabile soprattutto in fase di studio e comprensione del funzionamento delle librerie grafiche 3D di iOS e in fase di debug per la verifica delle operazioni eseguite sulla GPU.

1.4.2 SDK per la realtà aumentata: Vuforia



Figura 1.8: Logo di Vuforia, SDK di realtà aumentata per dispositivi mobili sviluppato da Qualcomm.

Vuforia é una SDK sviluppata da Qualcomm e disponibile per *Eclipse*, *XCode* e *Unity* che permette la creazione di applicazioni mobile con contenuti in realtà aumentata; in particolar modo, Vuforia fornisce API che permettono l’accesso alla fotocamera del dispositivo, l’elaborazione dei frame catturati dalla fotocamera stessa e la rilevazione su di essi di immagini o porzioni di immagini conosciute da cui far apparire contenuti definiti in real-time.

Vuforia utilizza OpenGL ES 1.1 oppure 2.0 per il rendering dei frame catturati dalla fotocamera sul display del dispositivo e per la sua successiva elaborazione; ogni singolo frame infatti viene analizzato da un **tracker** che si occupa di verificare la presenza di parti di immagini note (ovvero presenti nel *database di immagini locale* del dispositivo oppure nel *database di immagini in cloud*, a seconda di quale delle due opzioni sia stata scelta all’atto di inizializzazione della libreria) e di tracciarne i successivi spostamenti, in modo da restituire in ogni momento la posizione precisa dell’immagine e permettere all’app di eseguire il rendering dell’oggetto che si intende mostrare nella posizione esatta. L’immagine 1.9 mostra uno schema riassunto del funzionamento di base dell’SDK. Come si vede, appunto, la maggior parte delle azioni sono svolte dal tracker, mentre é compito dell’app ricevere in ingresso i dati da esso calcolati ed utilizzarli a seconda di ciò che si intende fare.

1.4.3 Dispositivi utilizzati

Per una generica app iOS esistono due diversi metodi per eseguirne il test: utilizzare il **simulatore** fornito da XCode tramite l’iOS SDK, che permette di vedere il funzionamento dell’app stessa direttamente dallo schermo del pc con cui la si sta sviluppando, oppure utilizzare un normale **dispositivo** ed effettuare la compilazione dell’app da XCode, per poi eseguire una vera e propria installazione della stessa sul dispositivo collegato e testarla direttamente. Sfortunatamente, a causa della struttura stessa dell’app in questione, il simulatore non é utilizzabile (per testare questo tipo di app infatti é necessario l’utilizzo di una fotocamera, non presente nel simulatore iOS); a causa di questa limitazione non é

Vuforia SDK

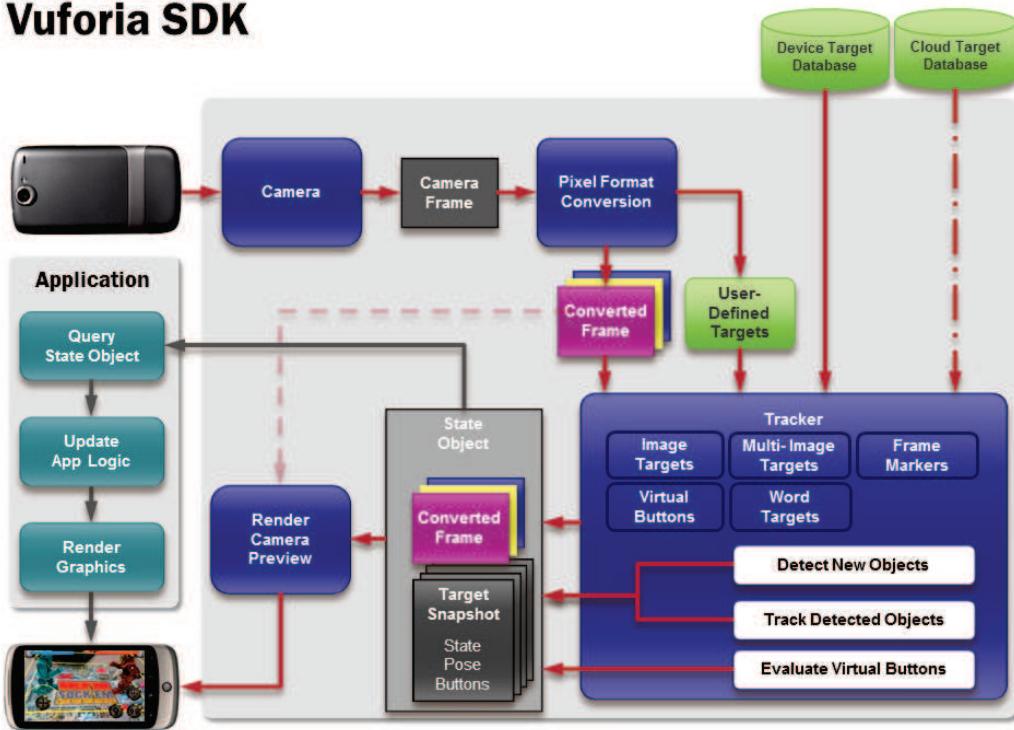


Figura 1.9: Schema riassuntivo rappresentante l’architettura dell’SDK Vuforia e il workflow ad alto livello del funzionamento delle operazioni di tracciamento delle immagini.

stato possibile testare l’app per ogni dispositivo iOS esistente, ma solo su quelli messi a disposizione dall’azienda Mantis. I dispositivi di test sono stati quindi i seguenti:

- iPad terza generazione (**iPad 3**), come dispositivo principale di debug e test;
- iPhone terza generazione (**iPhone 3GS**), come dispositivo di test occasionale;
- iPhone quinta generazione (**iPhone 4S**), come dispositivo di test finale.



(a) iPad 3

(b) iPhone 3GS

(c) iPhone 4S

Figura 1.10: Dispositivi utilizzati per il test dell’applicazione.



2 — Primi passi nella realtà aumentata

Sommario

Principi base di Vuforia

- Image target
- Frame marker
- Multi-target
- Virtual buttons

Vuforia samples

- Struttura generale di Vuforia per iOS: ARCommons
- Video playback

Al fine di comprendere appieno le potenzialità degli strumenti mobile di Apple e della piattaforma Vuforia, una cospicua parte iniziale dello stage si è incentrata sullo studio di queste realtà. In questo capitolo si intende approfondire i passaggi chiave riguardanti soprattutto la parte di realtà aumentata, elemento chiave dell'app. Lo studio riguardante iOS è invece avvenuto attraverso la documentazione fornita da Apple stessa e sui testi consigliati, i cui riferimenti sono presenti nella bibliografia.

2.1 Principi base di Vuforia

Come già accennato precedentemente, la parte principale di Vuforia è costituita dal tracker, una componente fornita dall'SDK stesso che si occupa di analizzare ogni singolo frame catturato dalla fotocamera per verificarne se contengano frammenti di immagini

note e a cui associare determinati comportamenti; queste immagini vengono chiamate **trackable**. Nello specifico, esistono tre tipologie di trackables: **image targets**, **frame markers** e **multi-target**.

2.1.1 Image target

Con image target, o semplicemente target, si intende una qualunque immagine precedentemente analizzata e del quale il software conosca le caratteristiche a priori, quali zone di contrasto, vertici acuti e altre peculiarità che possano essere riconosciute in svariate situazioni dal software.



Figura 2.1: Esempio di image target, nello specifico l'image target *stones* fornito come esempio da Vuforia. Si tratta, come si vede, di una normale fotografia, in grado però di essere riconosciuta dal software per le sue caratteristiche.

Non tutte le immagini però possono essere considerate valide come image target, proprio a causa della sua definizione; vi è la necessità infatti che l'immagine possieda determinate caratteristiche in modo che possa essere riconosciuta facilmente dal software. Il sito developer di Vuforia mette a disposizione uno strumento che permette la valutazione delle immagini scelte previo upload delle stesse; il portale infatti fornisce una *valutazione da 0 a 5 della qualità dell'image target* ottenuto da quell'immagine, con 0 voto più basso e 5 voto più alto. In seguito, è possibile anche effettuare il download delle informazioni estratte dall'immagine stessa, le quali, fornite a sua volta all'SDK di Vuforia sul dispositivo, permettono il riconoscimento dell'immagine a partire dai frame della fotocamera. I dati sono contenuti in due file, un XML e un DAT, del peso solitamente inferiore ad 1 MB.

Nel caso dell'app in questione si è utilizzato soprattutto questo tipo di trackable, principalmente per motivi dettati dalla minore invasività del loro funzionamento, requisito ritenuto determinante per il committente; poiché la rivista stessa contiene già delle immagini relativi agli articoli stessi, infatti, si è ritenuto adatto utilizzare le stesse come trackable per i contenuti aggiuntivi, senza bisogno di aggiungere quindi ulteriori elementi. È stata quindi fondamentale una continua interazione con gli editori della rivista al fine di poter utilizzare immagini adatte al contenuto degli articoli ma che fossero allo

stesso tempo adatte al riconoscimento da parte di Vuforia. Come regola generale, inoltre, si è deciso di rifiutare image target che fornissero una valutazione inferiore a 3, in modo da garantire una qualità maggiore del risultato.

2.1.2 Frame marker

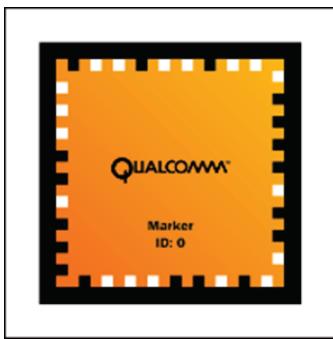


Figura 2.2: Esempio di frame marker, nello specifico il frame marker con ID 0.

Con frame marker, o semplicemente marker, si intende un particolare simbolo univocamente definito e pre-caricato all'interno dell'SDK di Vuforia; come tale, può essere utilizzato direttamente dopo aver definito il suo ID numerico, rappresentato come un intero che può variare da 0 a 511. È in grado di fornire **massime prestazioni di tracking** dato che richiede un algoritmo di riconoscimento più semplice e le sue caratteristiche sono predefinite dalla piattaforma stessa, ma non supporta il riconoscimento parziale dell'immagine (per cui è necessario che il marker sia interamente presente all'interno del frame della fotocamera per permetterne il riconoscimento), oltre ad essere limitato in termini di combinazioni possibili utilizzabili (sono infatti presenti solo 512 possibili marker). In generale è stato deciso di evitare l'utilizzo di questo tipo di trackable, a causa

della loro invasività e si è preferito optare per l'utilizzo di soluzioni come gli image target.

2.1.3 Multi-target

Con multi-target si intende un **insieme di image target** legati tra loro da una **relazione spaziale fissa**; in questo modo, una volta che uno degli image target che costituiscono il multi-target è stato rilevato, la posizione degli altri può semplicemente essere calcolata. I multi-target possono essere utilizzati per eseguire il tracking di oggetti solidi quali scatole o cubi in maniera efficiente, senza bisogno di analizzare ogni singola parte separatamente, con un guadagno in termini di efficienza e utilizzo di memoria. Per l'app in questione, tuttavia, non è stato necessario il loro utilizzo in quanto, trattandosi appunto di una rivista, non è sorta la necessità di eseguire il tracking di immagini in uno spazio 3D. Questa soluzione è stata comunque studiata come approfondimento sulle possibilità della piattaforma.

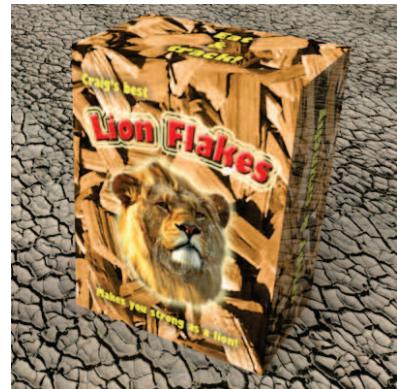


Figura 2.3: Esempio di multi-target: trattasi di un insieme di image target di cui si conosce la posizione l'uno rispetto all'altro.

2.1.4 Virtual buttons

Con virtual button si intende una **regione rettangolare di un image target** definita a priori che, quando **occluse totalmente o parzialmente** dal frame della camera (solitamente attraverso il “tocco” della regione corrispondente sull’immagine stampata) scatenano un particolare **evento**. Gli eventi possono essere molteplici e definiti dal programmatore, quali: cambio di caratteristiche del solido (vertici, texture e altre), link a siti esterni e altro. Per garantire il miglior funzionamento, la valutazione dell’occlusione di un virtual button è eseguita solo se il dispositivo è fermo o se comunque il frame è a fuoco, in modo da evitare di scatenare l’evento corrispondente per errore.

Nel caso dell’app in questione, i virtual button non sono stati momentaneamente previsti, ma il loro studio si è rivelato interessante ritenendo che possano essere introdotti in uno sviluppo futuro.



Figura 2.4: Esempio di virtual button: appoggiando un dito sulla regione dell’immagine corrispondente l’oggetto 3D cambia il proprio colore.

2.2 Vuforia samples

Gli strumenti principali della fase di studio relativa a Vuforia sono state le **Vuforia sample apps**, semplici applicazioni già pronte e fornite da Qualcomm stessa per mostrare le potenzialità della piattaforma Vuforia, comprensive di codice sorgente. Si è ritenuto che questa fosse una strategia adatta in quanto la documentazione online relativa a Vuforia, probabilmente a causa della novità dello strumento stesso, è ancora frammentata e difficilmente utilizzabile; si è optato quindi per l’analisi del codice sorgente di ognuno dei sample forniti utilizzando la documentazione per approfondimenti e chiarimenti, scelta che si è rivelata efficace soprattutto grazie alla chiarezza dei sorgenti forniti. I sample forniti da Vuforia che sono stati analizzati sono i seguenti:

Image target: si tratta della base di partenza per l’utilizzo dei modelli 3D all’interno di Vuforia. Fornisce un esempio basilare di app in grado di eseguire il rendering di un modello 3D (nel caso specifico, una *teiera di Newell*) sopra ad un image target precaricato e la cui immagine è fornita come PDF stampabile.

Frame markers: analogo al precedente, ma utilizza marker invece di image target, e sostituisce la teiera di Newell con alcune lettere in 3D.

Multi-targets: fornisce un esempio per l’utilizzo di un multi-target; il modello 3D orbita attorno al multi-target (che è possibile costruire grazie alla fustella fornita nel PDF stampabile allegato).

Video playback: fornisce un esempio di realtà aumentata con utilizzo di video anziché 3D. Utilizza gli stessi image target forniti negli esempi precedenti renderizzando al di sopra di essi un video precaricato all’interno dell’app. Qualora lo si desideri, il video può anche essere avviato a *schermo intero*, disattivando temporaneamente il tracking della fotocamera.

Virtual buttons: fornisce un esempio di utilizzo di virtual button. L’image target contiene 4 regioni colorate in modo diverso: inquadrandolo, appare una teiera di Newell

che é in grado di cambiare colore toccando (o occludendo) il corrispondente colore sull’image target.

Nonostante siano stati presi in esame tutti i sopra citati sample, solo 2 contengono caratteristiche che sono state effettivamente utilizzate all’interno dell’app, ovvero **Image target** e **Video playback**. Verranno quindi analizzati in modo specifico questi due sample, evidenziando le parti che sono state di particolare utilitá nello sviluppo successivo del progetto.

2.2.1 Struttura generale di Vuforia per iOS: ARCommons

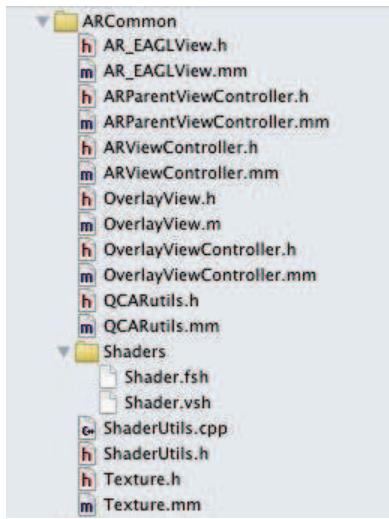


Figura 2.5: Elenco di file del gruppo ARCommons come visualizzati in XCode.

l’oggetto 3D.

Al contrario della versione Android, i sample di Vuforia per iOS sono strutturati in maniera piuttosto modulare: aprendone uno qualsiasi infatti si nota subito che esiste infatti una cartella (o, come viene chiamato in XCode, **gruppo**) dal nome **ARCommons**, contenente tutte le classi utilizzate in comune dai vari sample. Poiché la struttura é stata poi semplificata e replicata in gran parte anche nell’app Digitalic, di seguito si propone una breve descrizione delle classi in esso presenti.

AR_EAGLView: classe contenente una serie di metodi *stub* per il rendering di oggetti 3D con OpenGL ES. Si tratta di una classe pronta da estendere per permettere di implementare facilmente la propria view OpenGL ES. La classe é stata utilizzata all’interno di Digitalic applicando alcune piccole modifiche dovute al metodo di caricamento dei dati del-

QCARUtils: classe che si occupa di fornire una serie di metodi di utilitá per la gestione della libreria Vuforia; contiene il workflow necessario ad avviare tutte le componenti chiave di Vuforia (quali accesso alla fotocamera e rendering dei frame della stessa sullo schermo, caricamento dei trackable interessati, avvio del tracking). La classe é stata utilizzata anche per l’app Digitalic con modifiche minimali dovute soprattutto alla logica di caricamento dei trackable (vedi capitoli successivi).

Shaders: gruppo contenente i due shader OpenGL ES utilizzati dal sample (vedi 5.2.2). I due file sono stati utilizzati come base di partenza per lo sviluppo di shaders in grado di fornire un effetto di realismo maggiore.

ShaderUtils: classe necessaria al corretto funzionamento degli shaders. Si occupa in particolar modo di eseguire la *compilazione a run-time* degli shader e di renderne

possibile l'associazione dei parametri in input. La classe è stata utilizzata quasi interamente nell'app Digitalic, con le dovute correzioni in seguito alla modifica degli shaders.

Texture: classe contenente una parte del modello dell'app e rappresentante i dati relativi ad una singola texture da applicare al modello 3D che si sta renderizzando. Si occupa del suo caricamento in memoria e dell'accesso ai suoi campi dati. La classe è stata interamente utilizzata anche per l'app Digitalic.

ARParentViewController, ARViewController, OverlayView, OverlayViewController:

classi che si occupano di gestire l'interfaccia grafica dell'applicazione. Queste classi non sono state utilizzate ai fini di Digitalic, ma solo studiate al fine di raggiungere un maggior livello di comprensione della progettazione di app per iOS.

Image target

Il sample Image target, come già descritto, rappresenta la base della realtà aumentata per Vuforia: attraverso di esso è possibile visualizzare una teiera di Newell al di sopra di un image target i cui dati sono stati precaricati all'interno dell'app. L'oggetto 3D viene descritto da un file, compilato insieme all'app stessa, contenente i quattro parametri fondamentali per la sua definizione, vale a dire: numero di vertici, array dei vertici, array delle normali, array degli indici.



Figura 2.6: Image target in azione: inquadrando un image target il software mostra un oggetto 3D, nel caso specifico una teiera, appoggiata su di esso.

La struttura di Image target è piuttosto semplice e, tralasciando le classi necessarie ad iOS per la definizione dell'app, è costituita da una sola classe, **EAGLView**, che estende **AR_EAGLView** (già definita in **ARCommons**). La sua funzione è quella di caricare in memoria, all'avvio dell'applicazione, i dati relativi all'oggetto 3D che si intende mostrare e i dati relativi all'image target cui associarlo; possiede poi un metodo chiamato *renderFrame()*, invocato ciclicamente ogni volta che viene eseguito il rendering di un frame da parte del dispositivo (vale a dire, diverse decine di volte al secondo), all'interno del quale viene interrogato il tracker attivo circa la presenza di image target

noti all'interno del frame che si sta renderizzando. Qualora la risposta del tracker sia positiva, provvede quindi a renderizzare il corrispondente oggetto 3D applicando le dovute trasformazioni geometriche (traslazione, ridimensionamento, rotazione e prospettiva) a seconda della posizione dell'image target stesso, in modo da far apparire l'oggetto il più realistico possibile.

Ai fini dell'app Digitalic, lo studio di questo sample specifico è servito soprattutto per stabilire le guidelines circa l'algoritmo da utilizzare per renderizzare un oggetto 3D e le rispettive API di Vuforia necessarie al completamento di ogni operazione; sono state però completamente riviste le operazioni necessarie al **caricamento del modello 3D** e dell'image target associato in quanto il sample fornito utilizza risorse note a tempo di compilazione e quindi caricate direttamente all'interno dell'app in tale fase, mentre l'app Digitalic richiede una dinamizzazione di tutti i contenuti, al fine di evitare di dover eseguire una nuova release dell'app ad ogni nuovo contenuto aumentato da distribuire.

2.2.2 Video playback

Il sample Video playback presenta il funzionamento basilare di un'app di realtà aumentata Vuforia che intende renderizzare un video in 2D anziché un modello 3D. Come il sample precedente, utilizza un image target come trackable, ma carica preventivamente un video anziché i parametri di un oggetto 3D; questo video viene poi renderizzato in real-time sull'immagine una volta inquadrata, mentre l'audio viene gestito a parte in maniera indipendente da Vuforia e utilizzando librerie proprie di iOS. Viene inoltre fornita la possibilità di passare da una visualizzazione in realtà aumentata del video ad una a schermo pieno che utilizza il normale lettore multimediale di iOS, dando quindi la possibilità di continuare a visualizzare il video senza mantenere il trackable inquadrato. Al contrario del sample precedentemente analizzato, questo utilizza due classi principali: **VideoBackgroundHelper**, una classe che funge da controller per la gestione del video e delle sue funzioni, quali ad esempio caricamento, play e pausa, e **EAGLView**, che similmente all'omonima classe di Image target, esegue i metodi relativi all'associazione dell'oggetto da renderizzare con il suo rispettivo image target; possiede tuttavia alcune caratteristiche aggiuntive rispetto al suo corrispettivo 3D, tra cui la ricezione di eventi “*tap*” (in quanto un utente può avviare o mettere in pausa un video semplicemente eseguendo il tap, ovvero tocco, su di esso sul display del dispositivo) e il rendering multiplo all'interno del metodo di `renderFrame()` (poiché il sample rende un'icona al di sopra del video rappresentante lo stato, play o pausa, del video in questione).

Per quanto riguarda i contributi dello studio di questo sample ai fini di Digitalic, anche in questo caso si è fatto riferimento alla logica riguardante il caricamento del video e il suo rendering, ignorando le dinamiche di caricamento dalla memoria per gli stessi motivi enunciati in precedenza. Maggiore importanza è stata invece data a **VideoBackground-**



Figura 2.7: Esempio di Video playback fornito da Vuforia: un breve video viene visualizzato sul target scelto.

dHelper, le cui funzioni sono state quasi interamente importate nel progetto, eccezion fatta per quelle relative all'avvio di un video a schermo pieno.



3 — Gestione della memoria

Sommario

MRR e ARC: soluzioni diverse per lo stesso problema

- Manual retain-release
- Automatic reference counting
- La soluzione di Digitalic

Superata la fase di studio di Vuforia e compreso sufficientemente a fondo il funzionamento delle caratteristiche fondamentali dell'SDK, l'attività di stage si è incentrata sulla progettazione e sullo sviluppo dell'app richiesta. Una delle problematiche principali con cui ci si è dovuti confrontare fin dalle fasi iniziali riguarda la **gestione della memoria** utilizzata dall'applicazione nel corso del suo utilizzo. Il motivo per cui essa rappresenti un problema per un qualunque sviluppatore di app mobile è presto detto: in un dispositivo di questo tipo, la memoria RAM è estremamente limitata se paragonata a quella disponibile nei computer attualmente in commercio, anche di fascia non elevata. Basti pensare che i dispositivi iPhone di quarta e quinta generazione e i dispositivi iPad di seconda generazione, tra i più recenti e diffusi al momento di inizio di sviluppo dell'app, possiedono solamente 512 MB di RAM, una quantità affatto elevata se si considera che alcuni modelli 3D possono essere parecchio complessi e che l'utente normalmente avrà più di un'app aperta contemporaneamente ognuna con la propria grafica e le proprie

animazioni (iOS supporta il multitasking dalla versione 4.0), e anche nel caso dei dispositivi migliori in commercio (iPhone di sesta generazione e iPad di terza generazione) la quantità di RAM sale a 1 GB, rappresentando un miglioramento notevole ma costituendo ancora una sensibile limitazione. A questo scenario si aggiunge l'ulteriore difficoltà dell'**assenza di qualunque tecnologia di garbage collecting**, prevista dal linguaggio Objective-C ma non resa utilizzabile da Apple nei dispositivi iOS per motivi prestazionali. Per tutti questi motivi, la gestione della memoria rappresenta uno degli scogli più ardui da superare nella progettazione di un'app mobile come Digitalic, specialmente perché utilizza caratteristiche con forte impatto sulla memoria come fotocamera, video di lunghezza e peso variabile e modelli 3D.

3.1 MRR e ARC: soluzioni diverse per lo stesso problema

Fortunatamente, iOS fornisce due diversi metodi per la gestione della memoria, richiedenti competenze di livello diverso ma allo stesso tempo adatti a progetti di complessità differenti.

3.1.1 Manual retain-release

Il primo metodo viene chiamato “**Manual retain-release**” (abbreviato in *MRR*) o anche “**Manual reference counting**” e corrisponde ad una gestione manuale della memoria. Utilizzando questo approccio, il programmatore stesso si fa carico di mantenere traccia del proprio utilizzo della memoria, dichiarando costantemente quali oggetti sta utilizzando e quali invece intende rilasciare; ciò è possibile utilizzando alcuni metodi della classe NSObject (da cui vengono estese tutte le altre classi), ovvero il metodo retain e il metodo release.

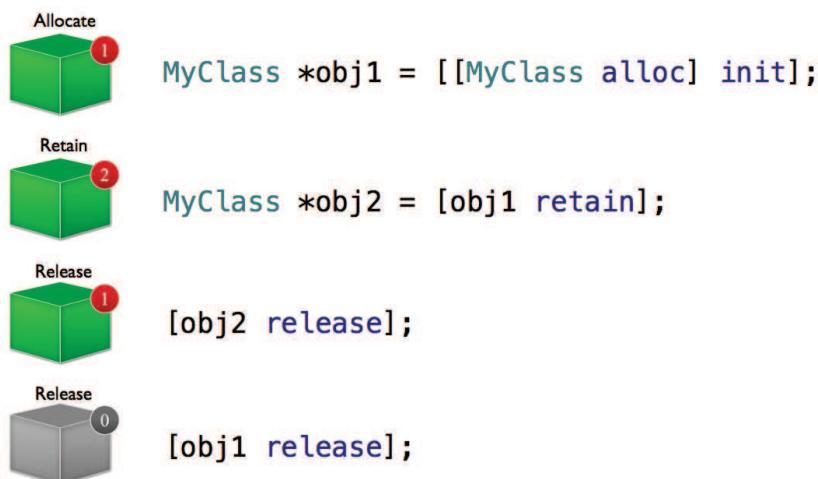


Figura 3.1: Schema di funzionamento del Manual retain-release; il programmatore è responsabile dell'intera gestione degli oggetti in memoria.

Internamente, il meccanismo utilizza un modello chiamato “**Reference counting**”, il

quale consiste semplicemente nel tenere traccia di quante volte un oggetto venga dichiarato come utilizzato da un'altra classe e quante volte venga invece dichiarato come non più necessario, il tutto attraverso l'utilizzo di un semplice contatore; quando il contatore raggiunge lo zero, l'oggetto viene rilasciato e la memoria corrispondente viene liberata. La chiamata al metodo retain serve quindi per dichiarare che la classe chiamante sta attualmente utilizzando quell'oggetto, mentre la chiamata al metodo release serve per dichiarare che l'oggetto in questione non è più utilizzato dalla classe chiamante. Esistono poi ulteriori metodi necessari a rendere questo tipo di gestione più completa e adatta ad un progetto complesso, ma non verranno analizzati qui. Per approfondimenti si veda la bibliografia.

3.1.2 Automatic reference counting

Il secondo metodo per la gestione della memoria invece si basa sul metodo MRR automatizzandone gran parte del comportamento; viene chiamato “**Automatic reference counting**” (abbreviato con *ARC*) e consiste in una gestione virtualmente simile a quella di un garbage collector. Abilitando questa opzione, infatti, le chiamate ai metodi retain e release vengono inserite automaticamente a tempo di compilazione.

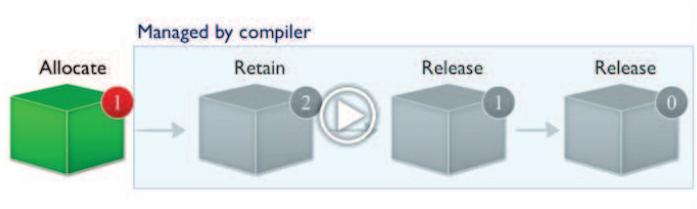


Figura 3.2: Schema di funzionamento dell’Automatic reference counting; dopo aver istanziato un oggetto, il programmatore perde il controllo sul suo ciclo di vita, ma tutta la gestione è esclusivamente a carico del compilatore.

Per stabilire quando i metodi debbano essere chiamati, l’SDK di iOS analizza il codice esaminando l’utilizzo di ogni oggetto all’interno del codice sorgente. Sebbene si tratti di un ottimo metodo per avere una gestione della memoria automatica, il suo utilizzo pone due vincoli: il divieto di utilizzo di qualunque chiamata relativa ai metodi di MRR (utilizzare una chiamata ad un metodo retain, release o altri provoca un errore a tempo di compilazione) e il divieto di utilizzare puntatori alla C all’interno di strutture dati (in quanto il loro utilizzo non può essere tracciato dal compilatore).

3.1.3 La soluzione di Digitalic

La soluzione applicata nel progetto consiste nell’utilizzo di metodi diversi per scenari diversi. Entrambi i metodi infatti possedevano notevoli difetti per il tipo di app che si andava a sviluppare: il metodo ARC non poteva essere applicato in assoluto poiché Vuforia utilizza al suo interno alcune strutture dati con puntatori alla C per la descrizione degli attributi di un solido 3D e, come abbiamo sottolineato, esse non sono compatibili con la gestione automatizzata da parte del compilatore. D’altra parte, una gestione manuale

richiedeva un livello di esperienza con il linguaggio Objective-C e di conoscenza circa il meccanismo di funzionamento di un'app e il suo ciclo di vita all'interno del dispositivo che non si riteneva di avere: oggetti non rilasciati (**leaking objects**) infatti possono causare memoria occupata e non recuperabile che, alla lunga, può portare iOS a generare un segnale chiamato “*Memory warning*” il quale porta iOS a rilasciare casualmente oggetti presenti in memoria per tentare di mantenere l'app in vita, specialmente oggetti appartenenti all'interfaccia grafica; ciò può però portare l'app a subire di problemi di inconsistenza interna e, alla fine, alla sua terminazione inaspettata.

Sotto queste premesse, si è deciso quindi di applicare ognuna delle due soluzioni nelle parti dell'app a loro più adatte, ovvero di utilizzare una gestione manuale della memoria per la parte riguardante Vuforia e la realtà aumentata, mentre demandare la gestione al compilatore relativamente alla parte di interfaccia grafica dell'app. Una delle caratteristiche di ARC infatti consiste nella possibilità di escludere particolari classi dall'inserimento automatico di chiamate retain/release, permettendo quindi in esse il loro utilizzo esplicito; alla luce di questo, la soluzione più adatta è stata quella di abilitare ARC globalmente, escludendo però tutte le classi dell'SDK di Vuforia e le classi a loro strettamente correlate, come quelle di inizializzazione e di rendering dei modelli 3D.

Al fine di limitare il più possibile il pericolo di leaking objects, inoltre, è stato utilizzata l'applicazione **Instruments** per l'analisi e il tracciamento della memoria durante un normale utilizzo dell'applicazione fin dalle prime fasi di debug; in particolare, l'instrument “Memory leak” si è rivelato particolarmente utile nella ricerca di oggetti inizializzati ma non deallocati e il suo utilizzo ha permesso di correggere tutti i bug riscontrati al riguardo.

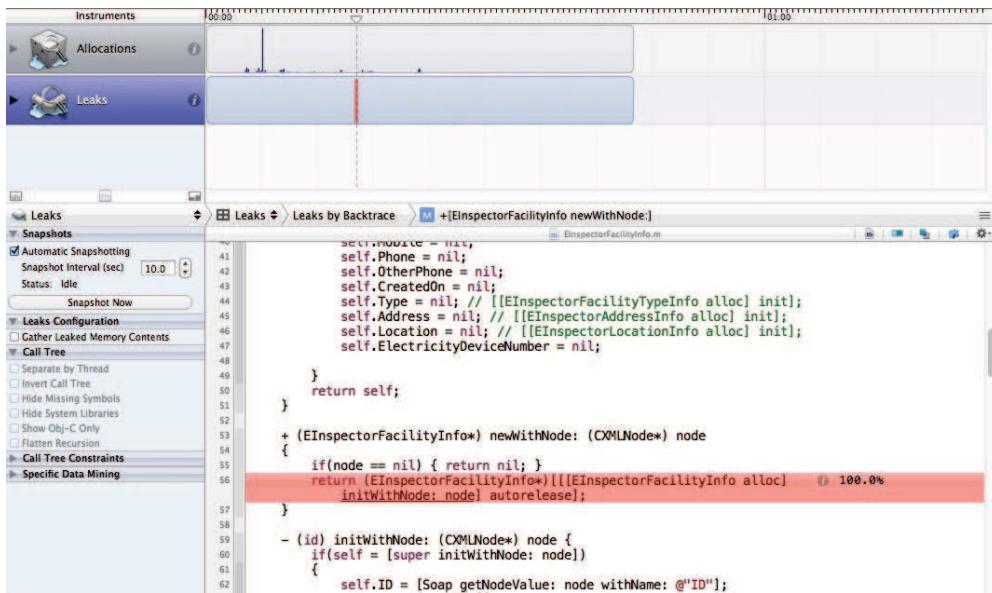
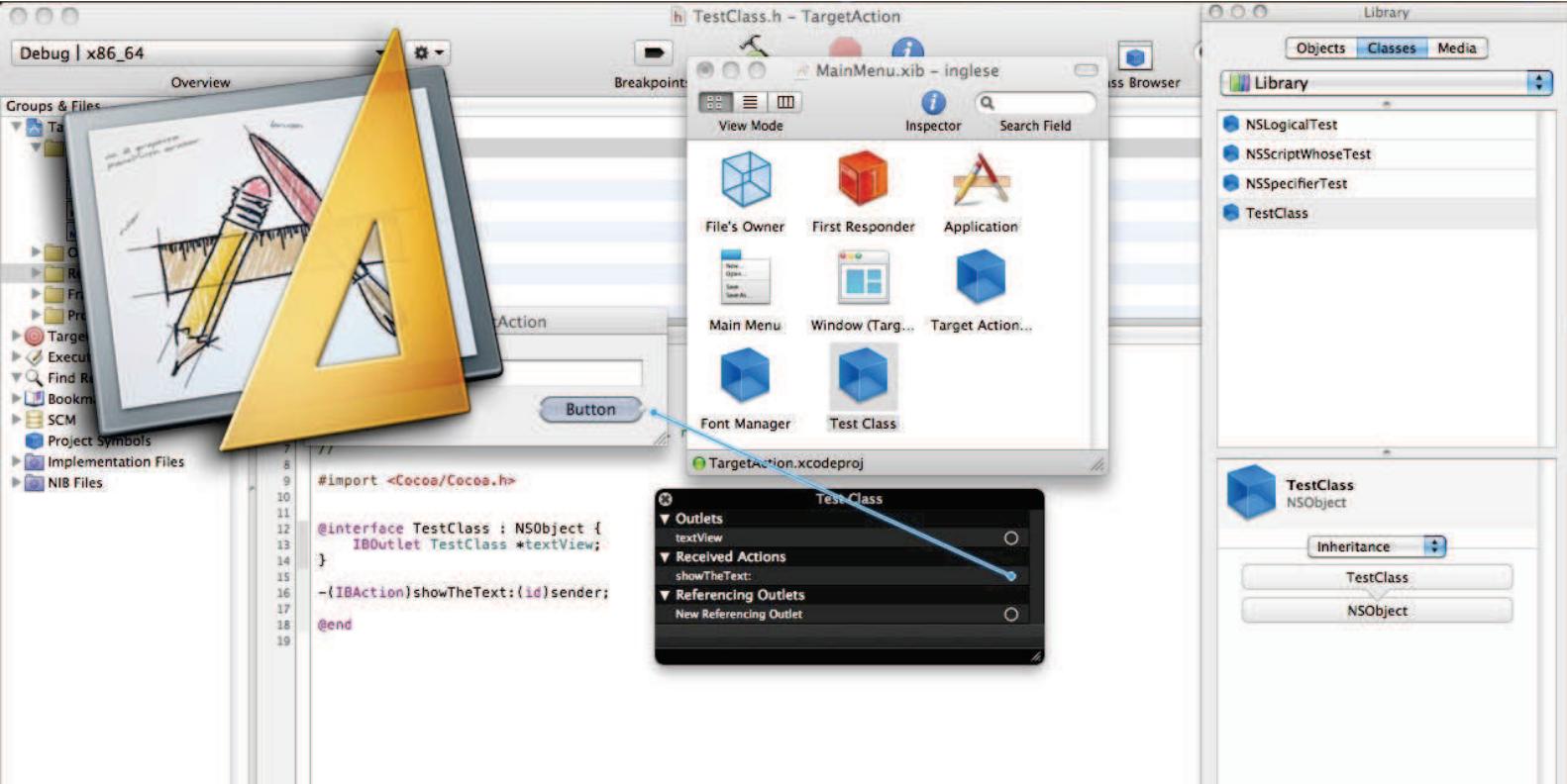


Figura 3.3: Esempio di schermata dell'applicazione Instruments che utilizza l'instrument Memory leak; l'applicazione è in grado di evidenziare in maniera automatica le istruzioni che generano spreco di memoria.

In generale, dunque, il risultato finale dell'applicazione di questa soluzione si può considerare soddisfacente, in quanto l'app mantiene i requisiti richiesti da Apple circa la

gestione della memoria (ovvero la mancanza pressoché totale di memory leaks). Il tempo dedicato all'utilizzo di questo metodo però è stato parecchio dispendioso, soprattutto per quanto riguarda le parti di codice di passaggio da un sistema di gestione della memoria ad un altro; la maggior parte dei memory leaks, infatti, si veniva a verificare in situazioni in cui classi che utilizzavano ARC eseguivano chiamate a metodi che non lo utilizzavano, e viceversa. In definitiva, probabilmente l'utilizzo di Instruments avrebbe permesso di utilizzare immediatamente il metodo MRR senza grosse differenze in termini di tempistiche ma con un arricchimento maggiore in termini di esperienza finale.



4 — Gestione dell'interfaccia grafica

Sommario

Due modi diversi per gestire l'interfaccia

- Gestione programmatica delle view
- Interface Builder e Nib/Xib files
- La gestione dell'interfaccia in Digitalic

Un'altra delle problematiche interessanti contro cui ci si è dovuti confrontare riguarda l'interfaccia grafica dell'applicazione. Per un'app iOS l'interfaccia grafica, o **GUI**, ricopre un ruolo fondamentale, essendo uno dei punti di forza dell'intero sistema operativo; Apple mira infatti a fornire una GUI semplice ed efficace, che pone in primo piano l'usabilità. Esiste perciò un'intera lista di raccomandazioni, o guidelines, molto specifiche e fornite da Apple stessa al fine di incoraggiare gli sviluppatori ad utilizzare un design che si adatti quanto più possibile al sistema operativo stesso e al suo grado di semplicità d'utilizzo, senza per questo motivo sacrificare la qualità grafica, anzi accentuandola qualora possibile.

4.1 Due modi diversi per gestire l'interfaccia

Dal punto di vista strettamente tecnico, esistono diversi modi per creare l'interfaccia grafica di un'app e si è rivelato necessario studiare in maniera almeno introduttiva i due principali metodi forniti da iOS SDK: la **gestione programmatica** e la gestione tramite **file Nib**.

4.1.1 Gestione programmatica delle view

Con gestione programmatica delle view si intende una gestione degli elementi grafici dell'app decisa interamente attraverso codice sorgente scritto dallo sviluppatore, senza alcun intervento di facilitazione da parte dell'IDE. È la tipologia di gestione che permette la maggiore flessibilità possibile, permettendo di utilizzare qualunque elemento grafico presente nel framework **CocoaTouch**, framework esplicitamente disegnato da Apple per la costruzione e gestione dell'interfaccia utente di un'applicazione iOS. CocoaTouch contiene a sua volta un insieme di framework necessari per gestire le svariate caratteristiche ad alto livello che possono essere tipicamente utilizzate per il design di una GUI su touch screen; il principale framework di CocoaTouch per la gestione della grafica è **UIKit**.

La gestione programmatica, come accennato, permette di inserire nelle proprie GUI tutti gli elementi previsti da Apple nel proprio framework, al costo però di una maggiore complessità del codice sorgente nella componente di Controller (in particolare, la componente ViewController, una componente prevista nella documentazione iOS che svolge la funzione di coordinare esclusivamente una o più view ad essa associate) e di un maggiore carico di lavoro da parte del programmatore; un approccio come questo infatti rende più complessa l'operazione di design dell'interfaccia grafica, mancando totalmente di automatizzazione delle operazioni anche più fondamentali, oltre a far aumentare in maniera notevole la quantità di errori che possono venire introdotti nel codice.

Utilizzare la gestione programmatica permette tuttavia di avere accesso ad alcune caratteristiche altrimenti non utilizzabili in altri modi, come le animazioni o l'utilizzo delle view OpenGL ES, fondamentali per il funzionamento dell'app in questione, e permette la modifica dei contenuti delle view, delle loro relazioni e dei loro comportamenti a runtime. In generale, quindi, viene raccomandato questa tipologia di gestione grafica per applicazioni particolarmente complesse dal punto di vista grafico, grazie all'elevato grado di libertà da essa permessa.

4.1.2 Interface Builder e Nib/Xib files

L'SDK di iOS mette a disposizione anche un ulteriore metodo per il design dell'interfaccia grafica, ovvero l'**Interface Builder**. Si tratta di un componente integrato in XCode che permette di creare una o più view attraverso l'uso di un editor *WYSIWYG* (*What You See Is What You Get*, ovvero un editor che permetta di vedere immediatamente il risultato sullo schermo senza bisogno di compilazione). L'Interface Builder crea un file con estensione **.xib**, il quale è fondamentalmente un XML contenente tutti gli elementi grafici e le relazioni tra di essi. Una volta che un file .xib viene compilato, questo si

trasforma in un file **.nib**, pronto per essere utilizzato da iOS stesso all'interno dell'app. L'utilizzo di un editor visuale per il design di un'interfaccia grafica presenta alcuni ovvii benefici, dovuti al fatto di poter visualizzare in real-time il proprio lavoro; inoltre, al contrario della gestione programmatica, la possibilità di introdurre errori è ridotta al minimo, dal momento che si utilizza un linguaggio di markup invece di un linguaggio di programmazione a tutti gli effetti.

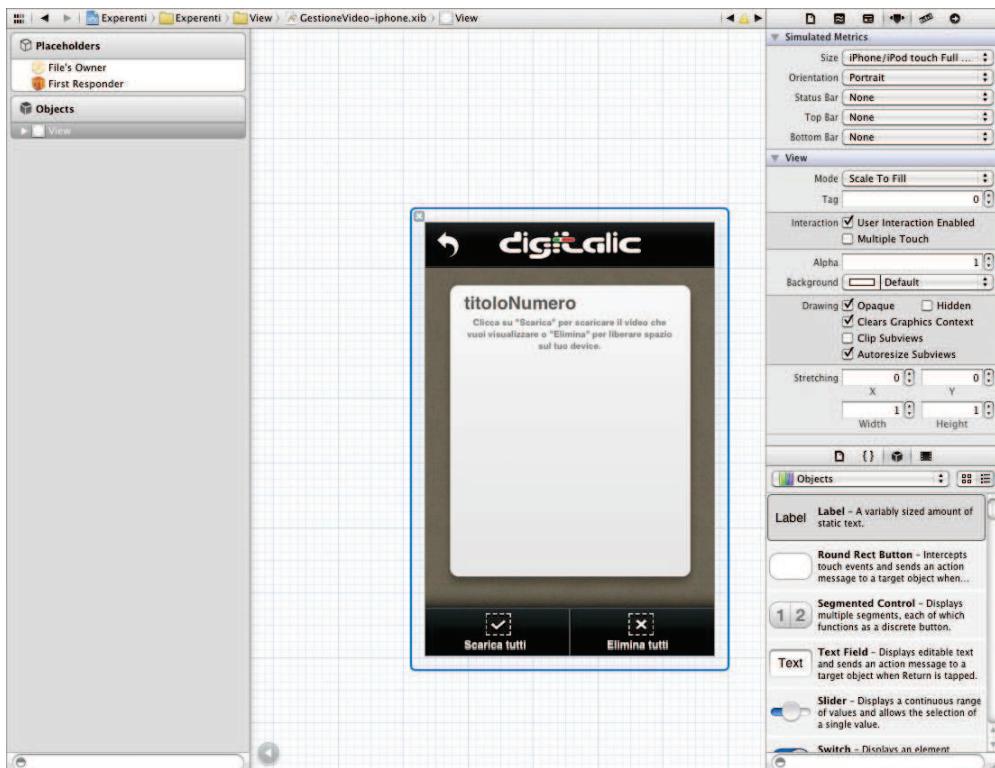


Figura 4.1: Esempio di Interface Builder in XCode, per la costruzione del file **.xib** contenente il layout per la gestione dei contenuti video in Digitalic.

Di contro, però, utilizzare Interface Builder mette lo sviluppatore di fronte a tutta una serie di vincoli e limitazioni non banali, prima tra tutti la difficoltà nell'automaticizzazione di comportamenti quali animazioni o view in grado di cambiare relazioni tra di esse (come ad esempio il cambio di view padre o simili). Inoltre, Interface Builder non permette di gestire in alcun modo le view di tipo OpenGL ES, a causa della complessità delle stesse. Risulta quindi ovvio il motivo per cui l'utilizzo di tale sistema in maniera assoluta non fosse tra le opzioni percorribili per l'app in questione.

4.1.3 La gestione dell'interfaccia in Digitalic

Per ovviare ai problemi di entrambe le soluzioni, anche in questo caso si è deciso di applicare ognuna di esse al fine di esaltare i rispettivi punti di forza. Si è deciso perciò di utilizzare la costruzione dell'interfaccia grafica tramite Interface Builder per stabilire gli elementi base delle view iniziali all'applicazione, per poi aggiungere ad esse eventuali animazioni o subview OpenGL ES attraverso la gestione programmatica.



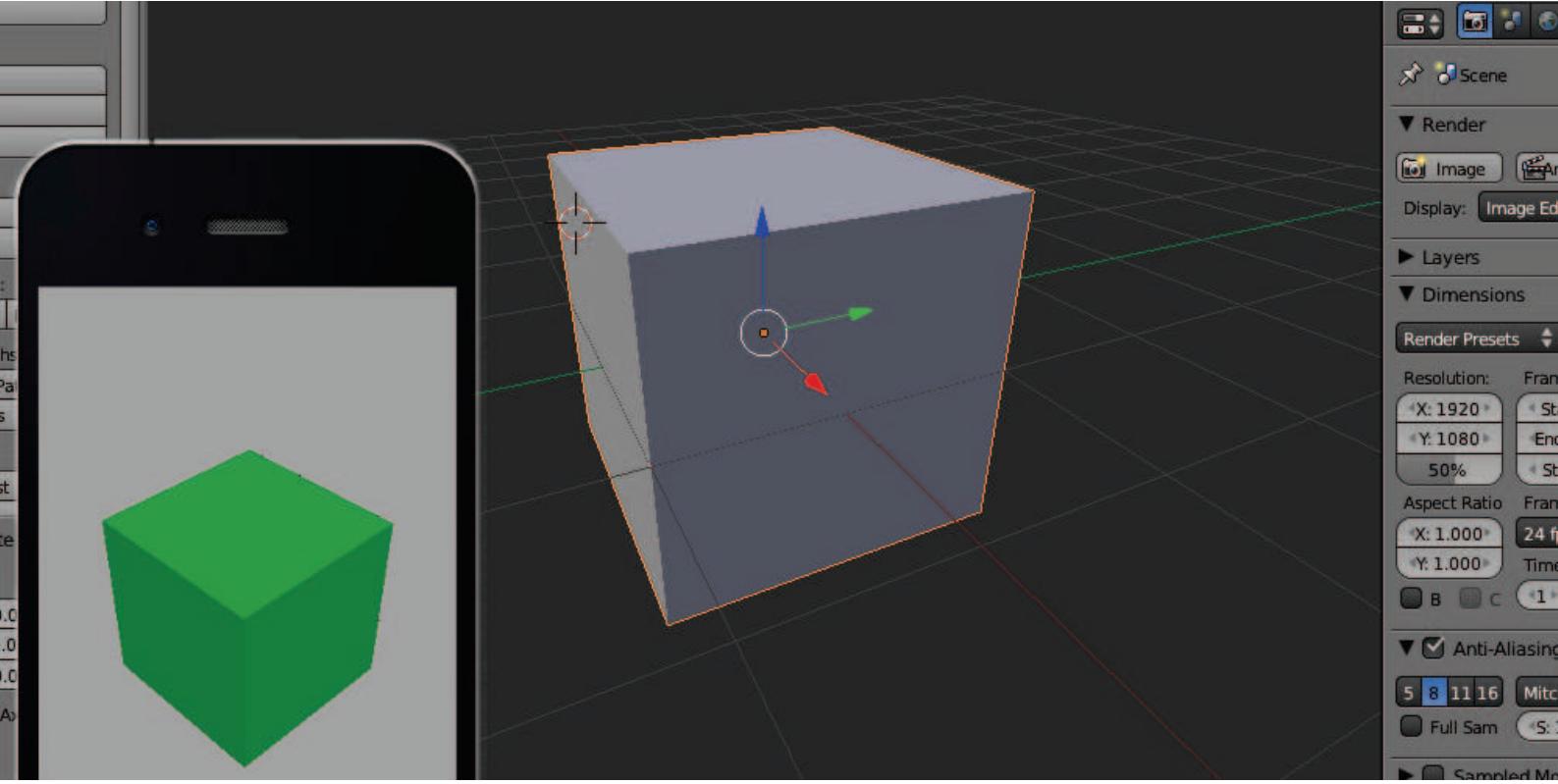
(a) Dashboard dell'applicazione: il layout è definito da un file .Nib, mentre l'immagine della copertina del numero è caricata dinamicamente da codice.

(b) Scelta dell'articolo da aumentare: oltre all'immagine e alla descrizione del contenuto, anche l'animazione di sfoglio delle pagine è realizzata programmaticamente.

Figura 4.2: Esempi di interfaccia grafica ottenuta dalla combinazione delle due tecniche.

Al fine di permettere un risultato quanto più esteticamente gradevole possibile, in accordo con i principi di usabilità e di esperienza utente su cui si basa iOS, si è pensato di costruire le singole view utilizzate grazie all'uso dell'Interface Builder, per poi stabilire il loro comportamento, le loro animazioni e le modalità del loro caricamento attraverso una gestione programmatica che permettesse una più grande flessibilità. In questo modo, la definizione delle view attraverso l'Interface Builder assume la funzione di costruzione di un layout grafico base, su cui poi costruire funzionalità grafiche per arricchire l'esperienza utente in un secondo momento, mentre le view di tipo OpenGL ES vengono gestite direttamente da codice.

A posteriori, si è notato che una tale scelta si è risolta in una grande semplicità al momento realizzativo, ma in una maggiore difficoltà nella successiva fase di manutenzione, principalmente dovuta all'eccessivo accoppiamento che si è venuto a formare tra le view e i rispettivi ViewController.



5 — 3D Modeling e 3D Rendering

Sommario

Concetti base di modellazione 3D e di rendering 3D

Come si crea un oggetto 3D

La modellazione 3D con Blender

Rendering su dispositivi mobili: OpenGL ES 2.0

Dalla modellazione al rendering

Come anticipato, una delle parti piú impegnative dell'attività di stage, principalmente perché non si trattava di un prerequisito in possesso né di un argomento affrontato durante la carriera universitaria, riguarda lo studio svolto sulla computer grafica 3D. Si tratta infatti di un tema estremamente vasto e difficilmente assimilabile in pochi mesi, per cui si è cercato di assimilare i concetti teorici base il prima possibile per poi cercare di colmare le lacune che si sono eventualmente presentate in corso d'opera.

La fase di studio riguardante la modellazione si è quindi incentrata sui suoi concetti basilari da un punto di vista piú generale, quali la costruzione di una mesh (collezione di vertici, spigoli e facce che costituiscono la forma di un oggetto poliedrico), la sua texturizzazione e i concetti base del rendering 3D. Il focus del problema si è poi successivamente spostato verso i dispositivi mobili e il dover replicare tutte le caratteristiche appena elencate anche su di essi, avendo quindi a che fare con capacità computazionali e memoria ovviamente inferiori rispetto a quelle di un computer.

5.1 Concetti base di modellazione 3D e di rendering 3D

C'è una grossa differenza tra la modellazione e il rendering 3D, pur essendo due concetti che spesso vengono confusi tra loro. La **modellazione 3D** consiste nella descrizione in linguaggio matematico di tutte le caratteristiche di un solido; il **rendering 3D** consiste invece nella rappresentazione in due dimensioni del suddetto modello 3D rispettando le corrette caratteristiche di colore, illuminazione e ombreggiatura di ogni singolo componente tridimensionale (chiamato **voxel**, abbreviazione di *volumetric pixel*). Si pensi quindi di voler rappresentare un semplice cubo; in fase di modellazione 3D si definiscono tutti i suoi vertici descrivendoli in uno spazio a tre dimensioni cartesiano (ovvero specificandone i valori x, y e z); questi vertici vengono poi combinati tra loro per formare spigoli e facce triangolari che compongono il cubo. La fase di rendering, successiva a quella di modellazione, riceve in input le caratteristiche del solido in questione e le analizza, fornendo in output una rappresentazione su immagine bidimensionale del cubo stesso, dando ad ogni specifico voxel un proprio colore risultante da calcoli eseguiti dalla GPU del dispositivo e comprendenti il colore naturale del voxel stesso (derivante da un preciso colore specificato in RGB/RGBA oppure in seguito all'applicazione di una texture), la normale della superficie cui appartiene il voxel e le caratteristiche di illuminazione presenti nella scena. Il risultato è quindi un'immagine 2D che può essere realistica o meno a seconda dello specifico algoritmo matematico utilizzato nel calcolo del colore di ogni singolo voxel.

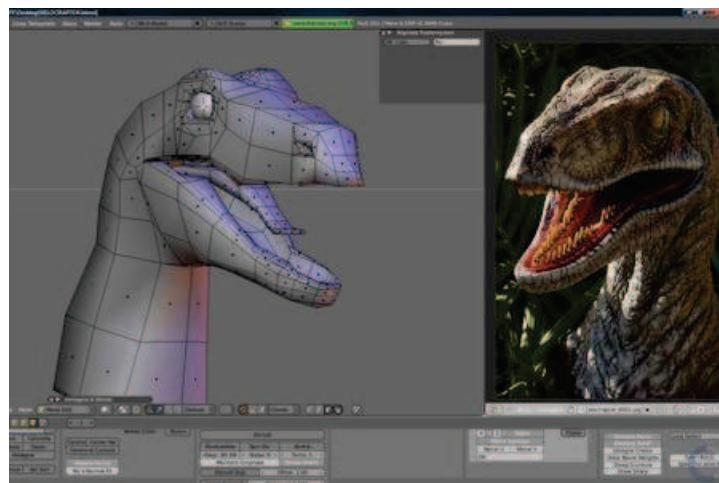


Figura 5.1: Esempio di modellazione 3D (sulla sinistra) e di rendering 3D (sulla destra) con Blender.

Nel caso specifico dell'app che si doveva andare a sviluppare, la modellazione 3D doveva essere un'attività eseguita a “back office” al fine di preparare una descrizione dell’oggetto che il dispositivo doveva essere in grado successivamente di elaborare e renderizzare sul proprio display. Lo studio di modeling e rendering quindi sono stati eseguiti su due tecnologie diversificate appartenenti a due dispositivi differenti: il primo su personal computer, il secondo su dispositivi mobili.

5.2 Come si crea un oggetto 3D

Di seguito viene descritto il processo con cui si è giunti alla soluzione finale implementata in Digitalic per ottenere il corretto rendering di un oggetto 3D.

5.2.1 La modellazione 3D con Blender

Per modellare i solidi è stato usato prevalentemente **Blender**, software open source di grafica 3D fornito tramite *GPL* (GNU General Public License). L'attività eseguita con questo software è iniziata con la modellazione più semplice e banale, ovvero la modellazione poligonale, che corrisponde all'utilizzare **primitive**, ovvero poligoni e poliedri di forme geometriche base (quadrato, cerchio, cubo, sfera, cono, ecc...), combinandole tra loro per creare forme più complesse.

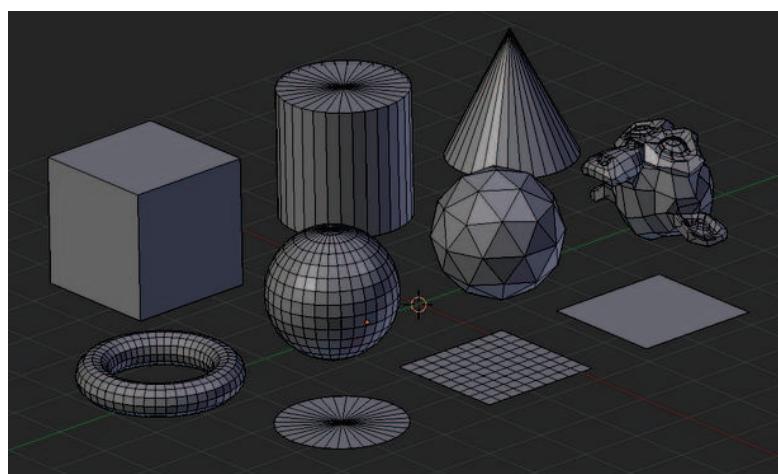


Figura 5.2: Le primitive in Blender. Ognuno di questi solidi basilari può essere utilizzato in combinazione con gli altri per creare forme geometriche tridimensionali complesse.

Successivamente, dopo aver acquisito una padronanza basilare dello strumento, ci si è spostati all'utilizzo di solidi già pronti e scaricabili gratuitamente dal web, in modo da minimizzare il lavoro e allo stesso tempo di utilizzare modelli di qualità più alta.

5.2.2 Rendering su dispositivi mobili: OpenGL ES 2.0

La parte di studio più dispendiosa riguardante l'ambito 3D è stata sicuramente rappresentata dall'acquisizione dei principi necessari di OpenGL ES, standard per il rendering 3D su dispositivi mobili. In particolare, è stata utilizzata la versione più recente disponibile di questo motore di rendering, ovvero la versione 2.0, profondamente diversa dalla precedente versione 1.1 in termini sia di qualità del risultato sia di funzionalità offerte oltre che di complessità di apprendimento e utilizzo.

Il principio fondamentale che regola OpenGL ES sono gli **shaders**. Trattasi di brevi programmi in C,



Figura 5.3: Logo di OpenGL ES, standard di riferimento per il 3D su dispositivi mobili quali iOS e Android.

soltamente composti da poche decine di righe di codice, che vengono eseguiti dalla GPU del dispositivo centinaia di migliaia di volte per ogni singolo frame. Per i dispositivi mobili esistono due tipi di shaders: i *vertex shaders*, che vengono eseguiti una volta per ogni vertice presente nella scena da renderizzare ad ogni frame, e i *fragment shader*, che vengono eseguiti una volta per ogni pixel contenuto nello schermo del dispositivo da renderizzare ad ogni frame. Per ovvii motivi, quindi, gli shaders devono essere più semplici e veloci possibili, dato che devono eseguire milioni di operazioni di calcolo al secondo. Basta infatti pensare che una GPU che deve renderizzare un quadrato a schermo pieno su un iPhone 3GS (con risoluzione 320x480 e impostazione di default di 30 frame al secondo) deve eseguire il fragment shader 4608000 volte al secondo; questo numero sale a più di 94 milioni di volte al secondo se consideriamo un iPad 3 con risoluzione molto maggiore (2048x1536).

Nonostante la difficoltà del loro utilizzo, comunque, gli shaders permettono di avere un'estrema libertà per quanto riguarda gli effetti applicabili. All'interno dell'app, i fragment shaders sono stati utilizzati principalmente per dare realismo all'oggetto 3D renderizzato attraverso l'applicazione di una fonte di luce puntuale, posizionata esattamente a fianco della fotocamera del dispositivo, proprio come se fosse il flash della fotocamera stessa. Il fragment shader riceve quindi in input la posizione della fonte di luce e calcola per ogni singolo pixel se e quanto la luce influisca sul suo colore, moltiplicando l'apporto dell'illuminazione per il normale colore della texture in quel punto. Il risultato è un maggiore realismo e una migliore interazione da parte dell'utente che è portato effettivamente a spostare il dispositivo attorno all'oggetto stesso per vederne tutte le caratteristiche.

Codice 5.1: Fragment shader utilizzato in Digitalic per il rendering degli oggetti 3D.

```

1 precision mediump float;
2 uniform mat4 modelViewProjection;
3 uniform mat4 modelViewMatrix;
4 varying vec3 vTransformedNormal;
5 varying vec4 vPosition;
6 varying vec2 texCoord;
7 uniform vec3 uPointLightingLocation;
8 uniform sampler2D texSampler2D;
9 void main(void) {
10     vec3 lightWeighting;
11     vec3 uAmbientColor = vec3(0.2,0.2,0.2);
12     vec3 uPointLightingColor = vec3(0.8,0.8,0.8);
13     vec3 lightDirection = normalize(uPointLightingLocation - vPosition.
14         xyz);
14     float directionalLightWeighting = max(dot(vTransformedNormal,
15         lightDirection), 0.0);
15     lightWeighting = uAmbientColor + uPointLightingColor *
16         directionalLightWeighting;
16     vec4 fragmentColor;
17     fragmentColor = texture2D(texSampler2D,texCoord);
18     gl_FragColor = vec4(fragmentColor.rgb * lightWeighting,
19                         fragmentColor.a);
19 }
```

5.2.3 Dalla modellazione al rendering

Un altro ostacolo posto dalle tecnologie utilizzate riguarda la necessità di far sì che i dati ottenuti in fase di modellazione fossero in qualche modo importabili dai dispositivi mobili per eseguirne il rendering. OpenGL ES infatti non prevede un meccanismo di importazione dei modelli 3D; al contrario, richiede che tutti i dati relativi al modello siano precaricati dall'applicazione stessa in strutture dati (principalmente array) e forniti già pronti per essere renderizzati. D'altra parte, nemmeno l'iOS SDK fornisce metodi per il caricamento dei formati di modelli 3D esistenti (quali OBJ, 3DS, Collada e altri); inoltre, l'importazione di simili file direttamente da parte del dispositivo presentava il problema non banale di ottimizzare il caricamento dei dati dal punto di vista della memoria su dispositivi che, come già sottolineato, presentano una quantità di memoria molto limitata: se consideriamo che i file sopra elencati presentano dati non effettivamente utilizzati dal rendering OpenGL ES e che quindi non avrebbe avuto senso caricare in memoria al fine di effettuarne il parsing, si capisce che la soluzione migliore era quella di effettuare il parsing preventivamente in modo da escludere tutte quelle caratteristiche non riproducibili dal dispositivo e inutili da inserire all'interno del file. La soluzione adottata è stata quella di utilizzare un parser Perl open source (disponibile al sito <http://heikobehrens.net/2009/08/27/obj2opengl/>) in grado di ricevere in input un file in formato OBJ e di produrre un file di testo che contenesse tutti e soli i dati utilizzabili da OpenGL ES per il rendering. L'adozione di questo script non è stata immediata, ed è stato ovviamente necessario adattarlo al fine di fornire un output che fosse quanto più facilmente utilizzabile dai dispositivi; il risultato è stato quindi uno script in grado di ricevere in input direttamente il file esportato dal software Blender e in grado di fornire un file di output contenente il minimo necessario delle informazioni e di cui il dispositivo fosse in grado di eseguire il parsing in maniera semplice e quanto più privo di problemi di memoria possibili. Alla data di redazione della presente tesi, il parser modificato viene ancora utilizzato, e si è rivelato quindi un investimento di tempo efficace e produttivo a lungo termine.



6 — Deployment su App Store

Sommario

Il processo di approvazione

L'approvazione di Digitalic

Terminata l'attività di sviluppo, ci si è preoccupati dell'attività di distribuzione dell'app. La scelta della modalità di distribuzione, quando si tratta di un'app iOS, è quasi una scelta obbligata: l'App Store di Apple. Rilasciata inizialmente nel 2008, l'App Store di iOS è un'applicazione di distribuzione digitale di prodotti mantenuta direttamente da Apple stessa. Il suo obiettivo è quello di fornire una fonte unica da cui ottenere tutte le app esistenti per iOS, occupandosi anche del loro successivo mantenimento (come gli aggiornamenti). Attualmente, il numero di app presenti nell'App Store arriva quasi al milione, e dalla sua data di rilascio è stato utilizzato per scaricare più di 50 miliardi di app.

6.1 Il processo di approvazione

Ogni app che intende essere distribuita attraverso l’App Store deve necessariamente passare attraverso un processo di verifica e approvazione da parte di Apple stessa. Questo processo viene effettuato al fine di garantire la migliore esperienza utente possibile; in questo modo infatti Apple può evitare all’origine che vengano distribuite app non funzionanti o, peggio, in grado di danneggiare il sistema. Processo e motivazioni analoghe anche per la pubblicazione di eventuali aggiornamenti di app già esistente all’interno dello store; solitamente, il processo di approvazione dura circa una settimana, ma possono volerci anche fino a 14 giorni.

Il processo di approvazione comprende alcuni passaggi che sono stati seguiti direttamente dallo stagista, in stretta collaborazione con il tutor aziendale, per garantire una comprensione completa dei passaggi necessari. Si inizia “firmando” l’intera applicazione con un certificato digitale generato e fornito direttamente dal sito di Apple Developer; dopodiché si richiede di compilare un form di descrizione dell’app che verrà inviato ad Apple assieme all’app da approvare. Questo form comprende svariate voci, che vanno dal nome e dall’icona dell’app fino al suo prezzo e al rating dei suoi contenuti (necessario per garantire la protezione degli utenti più giovani). Una volta compilato il form, si può procedere al caricamento dell’app direttamente da XCode; al caricamento segue una fase di verifica automatizzata che verifica la presenza di tutte le parti richieste internamente all’app oltre che controllare che esse corrispondano a quanto dichiarato nel form precedente.

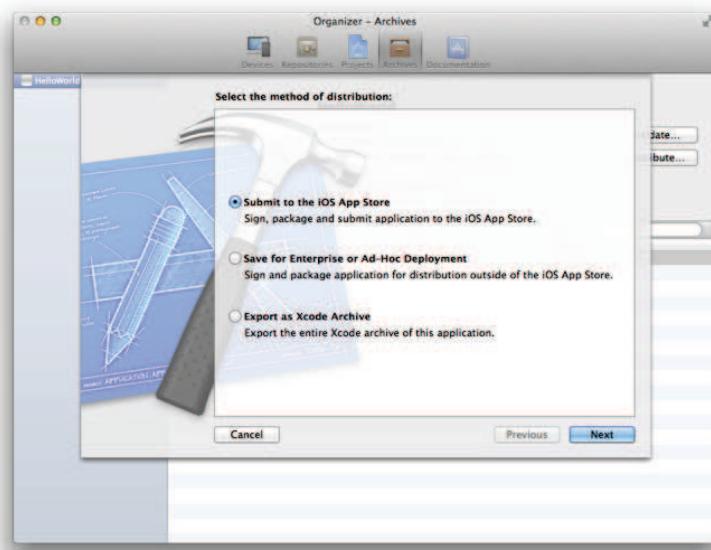


Figura 6.1: Schermata di XCode per la selezione del metodo di distribuzione dell’app; l’App Store è la prima opzione, selezionata di default.

Infine, l’app viene inviata all’App Review Team per la verifica finale. Essi controllano che l’app funzioni effettivamente senza provocare crash improvvisi durante il normale utilizzo, che non danneggi in maniera temporanea o permanente il sistema e che si integri correttamente coi servizi forniti da iOS. Esiste infatti una lista di criteri secondo cui le app vengono valutate da Apple, chiamate App Store Review Guidelines; l’osservanza di

tale lista di consigli garantisce l'approvazione senza problemi e il successivo inserimento all'interno dell'App Store.

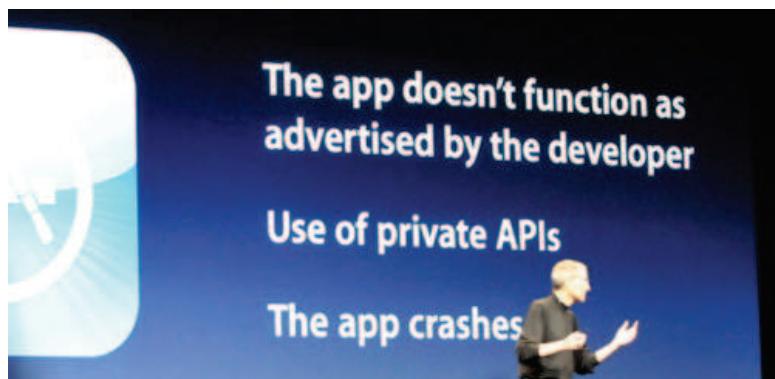


Figura 6.2: Steve Jobs, durante il keynote dell'Apple World Wide Developer Conference (WWDC) di giugno 2010 spiega le motivazioni principali di rifiuto delle app iOS. Nell'ordine: l'app non funziona come dichiarato dallo sviluppatore; l'app utilizza API private; l'app è soggetta a crash.

6.1.1 L'approvazione di Digitalic

Ovviamente, puntando ad una distribuzione sull'App Store di Apple, anche Digitalic è stata sottoposta al relativo processo di approvazione. Il primo tentativo, tuttavia, non è andato a buon fine a causa di una incomprensione delle iOS Data Storage Guidelines; di seguito la parte del documento di interesse.

Only documents and other data that is user-generated, or that cannot otherwise be recreated by your application, should be stored in the <Application_Home>/Documents directory and will be automatically backed up by iCloud.

Data that can be downloaded again or regenerated should be stored in the <Application_Home>/Library/Caches directory. Examples of files you should put in the Caches directory include database cache files and downloadable content, such as that used by magazine, newspaper, and map applications.

L'app inizialmente rifiutata eseguiva infatti il download dei contenuti aumentati all'interno della cartella <Application_Home>/Documents, la quale è però riservata ai contenuti creati dall'utente stesso e di cui iOS esegue in automatico il backup sul servizio iCloud nel caso in cui la memoria sul dispositivo scarseggi. Nel caso in cui tali contenuti fossero mantenuti all'interno di tale cartella, quindi, iOS potrebbe decidere di effettuare il backup dei contenuti occupando memoria riservata all'utente, quando in realtà si tratta di contenuti che possono essere ricreati semplicemente effettuando nuovamente il download degli stessi. Per questo motivo, si è rivelato necessario rivedere l'app in seguito al rifiuto da parte dell'iOS Review Team, in modo tale che salvasse i contenuti aumentati scaricati all'interno della cartella <Application_Home>/Library/Cache, cartella adibita a

contenere file che possono essere replicati o comunque riscaricati.

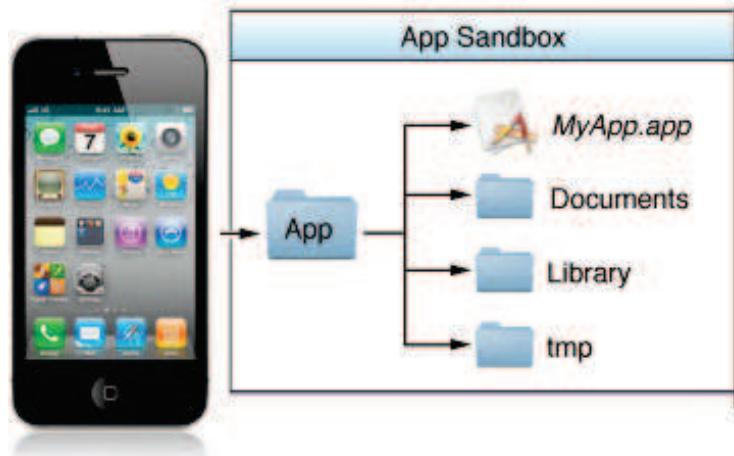


Figura 6.3: Schema ad alto livello della struttura delle directory di un'app iOS. La cartella Documents è riservata ai contenuti specifici dell'utente, mentre la cartella Library ai contenuti necessari all'applicazione stessa.

L'intero processo di primo invio, comprensione e correzione del problema e secondo invio ha occupato circa due settimane, successivamente alle quali l'app è stata valutata positivamente e inserita poi all'interno dell'App Store, da cui è attualmente disponibile per il pubblico.

MIGLIOR STAGE PER L'INNOVAZIONE

PREMIO stage IT 2012

7 — Conclusioni

Sommario

Statistiche di download
iOS e Android: un breve confronto
Possibili miglioramenti dell'app

L'esperienza di stage si è dimostrata arricchente sotto diversi punti di vista, in quanto oltre ad aver ampliato il bagaglio di conoscenze si è risolta anche in un alto grado di soddisfazione personale. Lo stage infatti è stato premiato, in occasione di Stage-IT 2013, come *Miglior Stage per l'Innovazione*; l'app prodotta, inoltre, è stata presentata in occasione di SMAU Milano 2012, in occasione della quale ha conseguito un discreto grado di successo.

7.1 Statistiche di download

Attraverso il portale iTunesConnect fornito da Apple è possibile verificare le statistiche di download dell'app e il suo andamento durante i mesi. I risultati che emergono sono piuttosto positivi: nel primo mese di pubblicazione l'app è stata scaricata più di 200 volte, arrivando a circa 500 download nell'arco di 6 mesi. Il numero di feedback positivi

inoltre si è rivelato soddisfacente anche da parte dell'azienda, che si è mobilitata molto presto per promuovere l'app il più possibile.

7.2 iOS e Android: un breve confronto

Digitalic è stata sviluppata contemporaneamente anche per la piattaforma Android da un altro stagista con cui si è collaborato in maniera piuttosto stretta. Il confronto tra le due piattaforme, le loro possibilità, le loro facilitazioni e le loro limitazioni è stato ovvio; è emerso infatti, rispetto allo sviluppo della stessa app per iOS, una maggiore difficoltà da parte di Android relativamente a problematiche di interfaccia grafica e di gestione della memoria, principalmente dovute alla molto più grande quantità di modelli di dispositivi con cui si deve far conto e ai loro adattamenti, problema inesistente per iOS dato l'esiguo numero di modelli disponibili.

Di contro, però, si è riscontrata una maggiore difficoltà relativamente all'SDK di sviluppo dell'app; a causa della sua struttura a livelli, infatti, in iOS esistono molteplici modalità per eseguire le stesse azioni, ognuna con un diverso grado di vincoli e di possibilità, ed è necessario quindi studiare in maniera approfondita ognuna di esse prima di poter scegliere in maniera adeguata e ottimale la strada da percorrere. Inoltre, esiste anche il problema del linguaggio utilizzato da iOS , Objective-C, totalmente diverso e svincolato da qualunque altro linguaggio precedentemente studiato, al contrario di Android che utilizza Java.

In definitiva, quindi, si può dire che Android si rivela più developer-friendly di iOS nei momenti iniziali di avvicinamento con la piattaforma, ma iOS prevede una serie di automazioni e meccanismi che, una volta appresi completamente, rendono più agevole lo sviluppo.

7.3 Possibili miglioramenti dell'app

Per quanto complessa e funzionale, l'app Digitalic possiede ancora un certo grado di miglioramento sotto parecchi punti di vista. Tra questi, la maggior parte riguardano la resa dell'oggetto 3D renderizzato, in particolar modo l'introduzione di caratteristiche non banali della modellazione 3D quali i materiali e le trasparenze, che contribuiscono parecchio al miglioramento del realismo dell'oggetto mostrato in realtà aumentata. Altre caratteristiche importanti che sono state richieste dall'azienda sono la possibilità di tracciare in maniera migliorata le statistiche di utilizzo dell'app e, soprattutto, la possibilità di mostrare più oggetti 3D e video contemporaneamente, migliorando di molto l'esperienza utente. Tutte queste caratteristiche sono, alla data di redazione della tesi, in fase di implementazione e potranno successivamente venire rilasciate con un aggiornamento dell'app.



Bibliografia

- [1] Buck, Erik M., *Learning OpenGL ES for iOS, A Hands-on Guide to Modern 3D Graphics Programming*, Addison-Wesley, 2012.
- [2] Flavell, Lance, *Beginning Blender, Open Source 3D Modeling, Animation, and Game Design*, Apress, 2010.
- [3] *iOS Technology Overview*, iOS Developer Library, Settembre 2012, <https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf>.
- [4] *Advanced Memory Management Programming Guide*, iOS Developer Library, Luglio 2012, <https://developer.apple.com/library/mac/documentation/cocoa/conceptual/MemoryMgmt/MemoryMgmt.pdf>.
- [5] *Transitioning to ARC Release Notes*, iOS Developer Library, Luglio 2012, <http://developer.apple.com/library/ios/releasenotes/ObjectiveC/RN-TransitioningToARC/RN-TransitioningToARC.pdf>.
- [6] *iOS Human Interface Guidelines*, iOS Developer Library, Maggio 2013, <http://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/MobileHIG.pdf>.
- [7] *UIKit Framework*, iOS Developer Library, Settembre 2012, http://developer.apple.com/library/ios/documentation/uikit/reference/UIKit_Framework/UIKit_Framework.pdf.

- [8] *Resource Programming Guide*, iOS Developer Library, Giugno 2012, <http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual>LoadingResources>LoadingResources.pdf>.
- [9] *App Store Submission Tutorial*, iOS Developer Library, Aprile 2013, <http://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/YourFirstAppStoreSubmission/YourFirstAppStoreSubmission.pdf>.
- [10] *App Store Review Guidelines*, iOS Developer Library, 2013, <https://developer.apple.com/appstore/resources/approval/guidelines.html>.
- [11] Rivera, J., Van der Meulen, R., *Gartner Says Asia/Pacific Led Worldwide Mobile Phone Sales to Growth in First Quarter of 2013*, Maggio 2013, <http://www.gartner.com/newsroom/id/2482816>.
- [12] Kronos Group, *OpenGL 2.0 Specification*, 2013, <http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf>.
- [13] Qualcomm Austria Research Center, *Vuforia Dev Guide*, 2013, <http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf>.

Ringraziamenti

Desidero ringraziare prima di tutti i miei genitori Mauro e Susy. Spero di essere un giorno la metà dei genitori che voi siete stati per me. Grazie per aver compreso e supportato in ogni modo che vi era possibile ogni mia scelta, e di continuare a farlo tuttora. Ringrazio mia sorella Desirée, perché so quanto bene vuole al suo fratellino. Ringrazio tutti i miei parenti, perché sento il loro orgoglio nei miei confronti ogni volta che mi parlano, e questo a sua volta riempie me di orgoglio.

Ringrazio Cristina, per avermi insegnato che a volte basta anche la cosa più semplice, come il sorriso di una persona a cui vuoi bene, a raddrizzare un'intera giornata storta.

Ringrazio tutto il gruppo dei miei amici: Adriana, Elena, Gianluca, Luca, Nicolas M., Paola, Petra, Sara, Sofia, Stefania. Avere un posto in cui tornare ogni weekend e da poter chiamare casa non ti fa sentire davvero mai solo. Ringrazio i miei compagni di università: Daniel, Margherita, Martina, Massimiliano. Mi hanno insegnato una cosa importante a cui non avevo mai dato molto peso: il viaggio conta ben più della destinazione.

Ringrazio tutto il team di Mentis: Amir, Barbara, Diego, Elisa, Lorna, Marco, Stefano, per aver trasformato la mia prima esperienza lavorativa in un'occasione di crescita come poche altre me ne siano capitate.

Ringrazio la mia relatrice, la prof.ssa Crafa, per essere stata sempre gentile e disponibile nei momenti di necessità.

Ringrazio tutti quelli che hanno fatto in modo che arrivassi a questo piccolo grande traguardo; sappiate che non intendo renderlo l'ultimo, e che spero tanto di potervi ancora di nuovo ringraziare tutti quando arriverà il prossimo, qualunque esso sia.