ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UN SISTEMA DI MODELLAZIONE 3D IN REALTÀ AUMENTATA

Elaborata nel corso di: Programmazione di Sistemi Embedded

Tesi di Laurea di:
GIUSEPPE PISANO

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2016–2017
SESSIONE II

PAROLE CHIAVE

Realtà Aumentata

Blender

Leap Motion

Vuforia

Unity

A Palmiro

Indice

Introduzione

Lo studio di alcune delle tecnologie e dei problemi legati allo sviluppo di sistemi in Realtà Aumentata, e lo sviluppo, grazie a questi strumenti, di un prototipo dimostrante le potenzialità di questa nuova tecnica di interfacciamento fra uomo e macchina; questo, in parole povere, è lo scopo che si è cercato di raggiungere con il lavoro presentato in questo elaborato, che, cercando di non dare nulla per scontato, esamina tutte le fasi del lavoro di ricerca e sviluppo svolto al fine di poter comprendere appieno questa tecnologia.
La trattazione parte da un capitolo dedicato all'introduzione del concetto di Realtà Aumentata: la sua storia, i suoi utilizzi, i suoi componenti fondamentali sono tutti punti presenti all'interno di questa prima sezione, mirata a fornire al lettore gli strumenti di base necessari alla comprensione di tale strumento.

Si procede poi alla selezione, nel secondo capitolo, di un dominio applicativo che possa fornire un campo d'applicazione valido per questa tecnologia, e quindi, di conseguenza, alla definizione degli obiettivi e degli strumenti necessari allo scopo. Questo punto risulta essere fondamentale in quanto, essendo comunque la Realtà Aumentata una tecnica di interfacciamento, un campo d'utilizzo inadatto comporterebbe una limitazione dei benefici che tale tecnica può offrire. Come si vedrà tale ambito verrà identificato in quello della modellazione 3D.

A questo punto si entrerà nel dettaglio dell'implementazione del prototipo, concentrandosi sia sull'integrazione degli strumenti puramente atti

alla gestione degli elementi AR, sia su tematiche meramente concernenti la modellazione.

Una riflessione sul lavoro svolto, con pregi e difetti del prototipo realizzato, si potrà trovare nei due capitoli finali.

Capitolo 1

Realtà Aumentata

"Per realtà aumentata (o realtà mediata dall'elaboratore, in inglese augmented reality, abbreviato "AR"), si intende l'arricchimento della percezione sensoriale umana mediante informazioni, in genere manipolate e convogliate elettronicamente, che non sarebbero percepibili con i cinque sensi."[23]

Questa definizione, una fra le tante che la letteratura fino ad ora ha prodotto, è stata scelta per la semplicità e chiarezza con cui descrive il concetto di AR. Una tecnologia capace di migliorare senso, vista, udito e, perché no, anche tatto e odorato. Una tecnologia capace di farci percepire oggetti esistenti solamente da un punto di vista computazionale e aiutarci durante la nostra vita quotidiana, rivoluzionando il concetto di interazione fra uomo e computer. Di questa tecnologia è oggetto questo primo capitolo, partendo dalle sue origini fino ad arrivare alle sue più recenti applicazioni.

1.1 Breve storia

Il termine "realtà aumentata" è attribuito per la prima volta al ricercatore della Boeing, la nota compagnia aereospaziale, Thomas P. Caudell[25]. Questi, nel 1990, introdusse tale concetto descrivendo un sistema, teorizza-

to da lui e dal collega David Mizzell, che doveva permettere agli impiegati dell'azienda di visualizzare gli schemi per l'assemblaggio dei velivoli. Tale sistema, basato sulla sovrapposizione di immagini virtuali a quelle reali, avrebbe permesso di evitare la produzione di costosi modelli in legno raffiguranti le schematiche, che così sarebbero sempre state a disposizione dei lavoratori, e in più modificate efficientemente e velocemente attraverso un sistema computerizzato. I primi esperimenti in questo campo sono però da datare ben prima del progetto di Caudell. Per esempio è del 1966 il lavoro del Professor Ivan Sutherland[28] dell'università di Harvard, che portò alla realizzazione del primo "Head-mounted Display" o HMD, ancora oggi elemento indispensabile di qualsiasi strumento per la Realtà Virtuale o Aumentata. Le dimensioni e il peso di questo modello erano tali però da non poter essere sostenute da una testa umana, motivo per il quale veniva appeso al soffitto del laboratorio per poter essere utilizzato. Proprio da questo deriva lo pseudonimo di questo primo prototipo: "Spada di Damocle" (fig. 1.1).

Figura 1.1: Spada di Damocle

Anche nella cultura di massa si possono trovare degli accenni ad alcuni dei concetti base della Realtà Aumentata in opere antecedenti il lavoro di Caudell. Per esempio, è del 1984 il film "Terminator", che dipinge la vista

del cyborg protagonista come un insieme di immagini reali "aumentate" da un flusso di annotazioni e immagini sintetiche.

Caudell, dunque, nel 1990 dava semplicemente un nome a un concetto già presente negli ambienti di ricerca e nell'immaginario delle persone: il passaggio dalla realtà che ci circonda, accessibile a chiunque attraverso i normali sensi, ad una realtà appunto "aumentata", estesa dalle informazioni prodotte attraverso il calcolatore, e quindi capace di offrire, attraverso gli opportuni strumenti, nuove modalità di interazione con il mondo che ci circonda.

Questa idea rivoluzionaria, nel corso del tempo, è stata approfondita da numerosi altri studiosi, che ne hanno investigato le possibili applicazioni e modalità di attuazione. Importantissimo per esempio è il lavoro di Ronald T. Azuma, che alla fine degli anni '90 rilasciò uno dei principali lavori sull'argomento[22], illustrante le principali modalità di creazione di sistemi in AR e le possibili applicazioni, presenti e future. Solo negli ultimi anni, però, questa tecnologia è uscita dagli ambiti di pura ricerca, in cui era relegata anche a causa dei costi elevati necessari alla creazione di tali sistemi, per entrare anche nella vita dei semplici utenti. Sono molte, infatti, le tecnologie che permettono di fruire di contenuti in realtà aumentata. Si va dagli strumenti creati ad hoc dalle più grandi aziende informatiche del pianeta, come gli HoloLens della Microsoft o i Google Glasses, ormai obsoleti, ai molto più comuni dispositivi mobili, smartphones e tablets, ormai presenti nella vita di qualunque persona. Questi, infatti, possiedono tutti gli elementi basilari per la generazione e la fruizione di contenuti in realtà aumentata: display, sensori, periferiche di input (la videocamera per citarne una), processori e coprocessori grafici. È proprio attraverso questi mezzi che la Realtà Aumentata si è fatta conoscere dal grande pubblico. Sono tantissime le applicazioni nate nell'ultimo periodo: da quelle ludiche, Ingress ne è un esempio, alle più serie, come SkyMap. E sebbene l'esperienza generata sia molto meno realistica rispetto a quella data dagli strumenti creati allo scopo, questi permettono di avere un assaggio di ciò che probabilmente sarà

una tecnologia in costante espansione nei prossimi anni.

1.2 Componenti Fondamentali

Bimber e Raskar nella loro opera[24] hanno definito i sistemi per la Realtà Aumentata come un insieme di alcuni macro blocchi che identificano i principali problemi da tenere in considerazione nella comprensione di questa tecnologia.

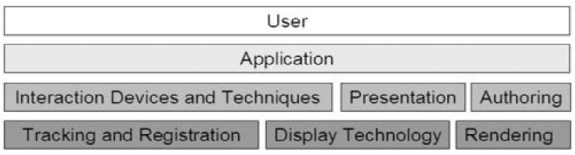

Figura 1.2: AR Building Blocks

Come si può vedere dall'immagine 1.2, questi blocchi sono posti in un ordine ben preciso dato dall'importanza che ognuno ha nel sistema. Partendo dall'utente, che si trova in cima alla pila, si arriva ai componenti più nascosti, che costituiscono gli elementi fondanti di qualsiasi sistema per la Realtà Aumentata: il tracking, le tecnologie per la visualizzazione e il rendering.

Tracking Per quanto riguarda il primo di questi tre punti, la registrazione della posizione dell'osservatore, oltre a quella degli oggetti reali e fittizi, risulta un punto cruciale nella costruzione di qualsiasi sistema AR. Infatti, un tracking preciso e veloce risulta indispensabile per la corretta generazione e visualizzazione delle componenti virtuali della scena. È essenziale quindi conoscere perfettamente i dati sulla posizione dell'osservatore nel mondo reale. Tali informazioni sono ottenute in vari modi: dalle modalità di acquisizione meccaniche ed elettromagnetiche, si va a quelle ottiche, contem-

planti soluzioni più costose e precise, quali l'uso degli infrarossi, e tecniche più economiche, basate sull'uso di videocamere tradizionali. Queste ultime si dividono ulteriormente in soluzioni marker-based, fondate sull'utilizzo di oggetti predefiniti da usare come punti di riferimento nella scena, e marker-less, più difficoltose da implementare, ma allo stesso tempo più promettenti e versatili.

Rendering Il terzo punto, concernente il rendering, riguarda un altro aspetto indispensabile per la fruizione di contenuti in Realtà Aumentata. Infatti, essendo quasi tutti i sistemi di questo tipo basati sulla sovrapposizione di elementi fittizi a quelli reali, il rendering real-time assume una grandissima importanza, soprattutto nell'ottica di una integrazione fra elementi virtuali e reali realistica. Si rendono necessarie, quindi, delle avanzate tecniche di rendering, supportate sicuramente dall'utilizzo delle più innovative schede grafiche, per ottenere delle performance nella generazione delle immagini tali da ricavare dei risultati ottimali dal punto di vista dell'interattività con la scena.

Visualizzazione Rendering e tracking all'avanguardia non bastano però da soli ad ottenere un buon sistema per la Realtà Aumentata. È necessaria, infatti, una modalità di visualizzazione delle informazioni generate. Anche in questo caso sono presenti varie tecniche, con differenze per esempio nel posizionamento e nella modalità di presentazione delle immagini, ma le principali tecnologie possono essere collocate in tre insiemi[29]:

- Video see-through Display, in cui l'ambiente virtuale è ottenuto visualizzando un feed video della realtà e contemporaneamente sovrapponendo all'immagine digitale gli elementi AR.

- Optical see-through Display, approccio utilizzato dallo stesso Sutherland, in cui la percezione tradizionale della realtà viene mantenuta,

e gli elementi AR sono sovrapposti e visualizzati grazie all'utilizzo di speciali display e lenti trasparenti.

- Projective Display, in cui le immagini AR vengono proiettate direttamente su degli oggetti reali. Questa categoria risulta essere la più futuristica, e al momento la meno consueta.

Interazione Sopra questi componenti, che fanno da base dello stack, viene posto il blocco relativo alle tecniche di interazione con relative tecnologie. Infatti, ottenere una rappresentazione unitaria e coerente di elementi reali e fittizi, seppure importantissimo, non basta a creare un'esperienza completa di Realtà Aumentata. Bisogna porsi il problema di come l'interazione con questi ultimi possa avvenire. I dispositivi di input tradizionali, come mouse, tastiera e touch-screen, si dimostrerebbero infatti insufficienti, essendo una delle prerogative principali dei sistemi AR l'immersività dell'esperienza. Proprio a questo scopo, sono varie le tecniche che sono state utilizzate nel tempo [29],dalle interfacce aptiche, che permettono di manovrare dei sistemi reali o virtuali e di riceverne un feedback tattile, fino alle interfacce testuali e ai dispositivi per il tracking del movimento oculare, ma quelle maggiormente impiegate sono due: il riconoscimento vocale e il riconoscimento di gesti. Con il primo si intende l'insieme delle tecniche che mirano al riconoscimento e alla traduzione del linguaggio parlato in linguaggio scritto. Questo permette delle interazioni fra utenti e macchine estremamente spontanee, dovendosi i primi limitare a esprimere comandi e istruzioni semplicemente parlando al calcolatore. Questa tecnica può essere vista come un miglioramento delle interfacce testuali, difficilmente integrabili in dei sistemi AR portatili. Il secondo, invece, mira al riconoscimento dei movimenti del corpo e quindi alla loro associazione a istruzioni da far eseguire alla macchina. Sono state sperimentate varie tecniche di tracciamento, basate su diversi sistemi di raccolta dei dati. Le più comuni si basano sull'utilizzo di telecamere, e quindi su tecniche di visione artificiale, altre più particolari si basano sull'utilizzo di ultrasuoni.

Applicazione L'ultimo strato, che si frappone fra gli utenti e le tecnologie fino ad ora trattate (fig. 1.2), è quello applicativo. Infatti, essendo in fin dei conti la Realtà Aumentata una semplice interfaccia uomo-macchina, si rende necessario trovare degli ambiti in cui questa tecnologia possa rendersi utile e in cui possa portare degli effettivi benefici. Argomento della prossima sezione è proprio la descrizione di alcuni dei domini ove la Realtà Aumentata, allo stato attuale della tecnica, ha già trovato degli ottimi utilizzi.

1.3 Applicazioni

Gli ambiti fino ad ora studiati e in cui sono stati compiuti i maggiori progressi nel campo della Realtà Aumentata risultano essere all'incirca dodici[26]:

- Campo Medico, con lavori mirati alla visualizzazione dei dati dei pazienti;
- Campo Manifatturiero, con studi mirati al miglioramento dei processi di produzione. Anche il lavoro dello stesso Campbell appartiene a quest'area;
- Campo dell'Intrattenimento, sviluppo legato soprattutto alla produzione di videogiochi;
- Campo Militare, per la creazione di sistemi utili all'addestramento e alla visualizzazione di informazioni direttamente sui campi di battaglia;
- Campo della Robotica, utilizzi per il miglioramento dell'interazione fra uomo e sistemi robotici;
- Campo della Visualizzazione, ambito molto generico che raggruppa le applicazioni mirate alla rappresentazione alternativa di dati sennò

difficilmente accessibili, come potrebbero esserlo i concetti più astratti legati al mondo della scienza;

- Campo Educativo, applicazioni per l'integrazione degli strumenti conoscitivi basati sulla visualizzazione di concetti e fenomeni difficilmente sperimentabili nel mondo reale;
- Campo del Marketing, per la creazione di strumenti di vendita interattivi;
- Campo del Turismo, con applicazioni per la visualizzazione di informazioni turistiche e archeologiche, per esempio nei musei;
- Campo dei Trasporti, esperimenti mirati alla visualizzazione dei percorsi e di informazioni utili ai viaggiatori;
- Campo Geo-spaziale, sviluppo legato alla visualizzazione di mappe e dati geografici;
- Campo dell'Ingegneria Civile, per il miglioramento dei processi di costruzione e rinnovamento di strutture architettoniche.

Non si entrerà nel dettaglio di tutti i sistemi creati in questi ambiti, cosa per la quale si rimanda alla bibliografia, ma ci si limiterà a citare un esempio ritenuto emblematico. Si tratta delle prime scarpe in Realtà Aumentata della storia. Nate dal contributo di due grandi marchi Giapponesi e rilasciate a Giugno 2017, queste scarpe, normali ad uno primo sguardo, se osservate attraverso l'applicazione rilasciata gratuitamente, sembrano prender vita grazie ad una semplice animazione che coinvolge il logo del prodotto. Per quanto possa sembrare un prodotto alquanto inutile, risulta invece un ottimo esempio nel mostrare come l'interesse per questa tecnologia stia aumentando incredibilmente all'interno della società.

È in questo contesto che si inserisce il lavoro svolto per questo progetto, che si propone di indagare alcune tecnologie al momento disponibili per la creazione di applicativi basati sulla realtà aumentata, e di realizzare, grazie ad esse, un prototipo dimostrativo delle loro funzionalità.

Capitolo 2

Obiettivi Progettuali

Dopo aver parlato degli elementi principali necessari alla creazione di sistemi per la Realtà Aumentata si possono ora delineare le caratteristiche che il sistema dovrà avere e le sue funzionalità, basandosi anche sulle tecnologie selezionate per l'implementazione.

2.1 Funzionalità Richieste al Software

Come visto nel capitolo precedente, sono svariati i campi in cui la Realtà Aumentata ha dimostrato di avere degli ottimi utilizzi, spaziando, per esempio, da quello medico a quello dell'intrattenimento. Però molti sono anche gli ambiti in cui le prove sono, per ora, state poche, e che sicuramente avrebbero tanto da guadagnare da questa unione. Fra questi è ricaduta la scelta per la realizzazione di questo progetto: la modellazione 3D.

La modellazione 3D, infatti, è una delle attività che beneficerebbe maggiormente dal passaggio da un paradigma di lavoro tradizionale, basato su schermo e tastiera, ad uno completamente nuovo, com'è quello che la Realtà Aumentata può offrire. Questo perché, per quanto gli strumenti utilizzati fino ad ora siano ottimizzati, proprio a causa della natura stessa dell'attività, mirata alla costruzione virtuale di oggetti tridimensionali, i supporti

tradizionali non possono che male adattarvisi. Un esempio banale è la definizione di punti nello spazio 3d, attività alla base di qualsiasi modellazione. Questo gesto, anche se per una persona semplicissimo e del tutto naturale in un ambiente reale, risulta essere alquanto macchinoso se fatto attraverso il classico mouse, ottimo in uno spazio bidimensionale ma del tutto inefficiente in uno tridimensionale. La Realtà Aumentata offre delle possibilità di interazione che molto meglio si adattano alle caratteristiche dell'attività.

Definito l'ambito, non resta che delineare gli obiettivi da raggiungere:

1. Creazione Mesh 3D
    - Possibilità per l'utente di utilizzare delle Mesh predefinite o di creare le proprie a partire dalla definizione di punti nello spazio.
2. Modifica Mesh 3D
    - Possibilità di modificare le Mesh create tramite l'utilizzo delle tre trasformazioni base (traslazione, rotazione, scalatura).
3. Eliminazione Mesh 3D
    - Deve essere possibile eliminare le Mesh realizzate.
4. Multi utenza
    - Possibilità per più utenti di assistere e partecipare contemporaneamente all'attività di modellazione.
5. Interazione fisica con il modello virtuale
    - Grazie all'utilizzo di uno strumento per il riconoscimento di gesti deve essere possibile interagire, e di conseguenza modificare l'oggetto virtuale, solamente grazie all'utilizzo delle proprie mani, rendendo il più naturale possibile l'esperienza di modellazione.

2.2 Tecnologie Prescelte

Prima di definire una possibile architettura del sistema si procede all'introduzione delle due tecnologie selezionate per la creazione del prototipo, necessarie alla gestione dei due aspetti fondamentali del paradigma AR: il tracking e l'interazione.

2.2.1 Vuforia

Una parte fondamentale di qualsiasi sistema per la Realtà Aumentata è l'integrazione fra le immagini reali e le informazioni generate sinteticamente. Questo processo, basato su tecniche di visione artificiale, ha quindi come scopo l'unificazione dei due sistemi di riferimento presi in considerazione: quello virtuale e quello reale.

Sono tanti i development kits nati proprio a questo scopo, e che dunque incorporano al loro interno le funzionalità per il tracking e il rendering dei contenuti AR, come per esempio ARToolKit [1] e Catchoom CraftAR [3], ma per lo sviluppo del software si è scelto di utilizzare l'SDK Vuforia [18].

Questa piattaforma di sviluppo, lanciata nel 2010 dalla Qualcomm e poi acquistata nel 2015 dalla PTC, una multinazionale specializzata in tecnologie per l'Internet of Things e per la realtà aumentata, vanta un ecosistema di più 300.000 sviluppatori in tutto il mondo, ed è stata utilizzata per la creazione di più di 35.000 applicazioni. Questo grande successo, che ha portato questo framework a essere il più conosciuto e usato al mondo, non è però ingiustificato. Infatti Vuforia è capace di offrire, oltre a una grande robustezza e stabilità, una altrettanto grande semplicità di sviluppo. Inoltre, il suo supporto comprende una grande varietà di dispositivi: dai più sofisticati, come gli Epson Moverio e i Microsoft HoloLens, si va ai più comuni, come notebooks, smartphones, utilizzabili anche in modalità Cardboard, e tablets.

Dal punto di vista del funzionamento, questo SDK è basato sul tracking real-time di immagini e oggetti. In relazione a questi target, denominati

"Marker", è possibile posizionare degli elementi virtuali, dei modelli 3D per esempio, che quindi vengono visualizzati a schermo in accordo con la prospettiva dell'osservatore. Uno dei punti di forza principali è la libertà nella scelta di questi Marker. Infatti, il framework permette il riconoscimento di parole, oggetti e immagini, anche più di una contemporaneamente, preventivamente scannerizzati attraverso degli strumenti appositi, messi a disposizione degli sviluppatori.

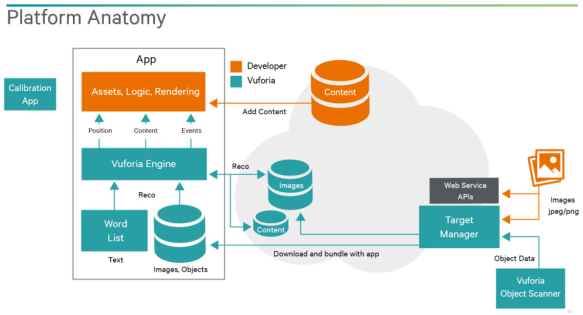

Figura 2.1: Architettura Vuforia

Come si può notare dall'immagine 2.1, il lavoro dello sviluppatore è ridotto al minimo. Infatti, tutti gli aspetti relativi alla generazione dei Target e alla loro gestione sono completamente trasparenti all'utilizzatore del framework, che si deve limitare a decidere quali saranno gli elementi da utilizzare come Marker e a definire la logica dell'applicativo. È presente anche un meccanismo di Cloud Recognition, che permette di mantenere una lista dei Target in un database apposito, senza dover integrare le risorse all'interno dell'applicazione.

Vuforia mette a disposizione degli sviluppatori delle API in C++, Java e per i linguaggi .NET, attraverso un'estensione per il game engine Unity [17], di cui si parlerà in seguito. In questo modo è possibile realizzare delle applicazioni in linguaggio nativo, oppure crearne una versione Unity, che risulta facilmente portabile in tutte le piattaforme supportate.

Un confronto dettagliato in termini di funzionalità fra questo e gli altri maggiori SDK disponibili sul mercato lo si può trovare nel lavoro di Amin e Govilkar[21].

2.2.2 Leap Motion

Per quanto riguarda l'interazione, che come da specifiche deve basarsi sul riconoscimento di gesti, si è scelto di utilizzare un interessante dispositivo USB, il Leap Motion [9].

Figura 2.2: La periferica Leap

Questa piccola periferica (mostrata nell'immagine 2.2), rilasciata nel 2012 dalla Leap Motion Inc., fondata nel 2010 da David Holtz e Micheal Buckwald, permette di effettuare il tracking del movimento delle mani, e quindi di utilizzarle in sostituzione dei più tradizionali dispositivi di puntamento. Dal punto di vista hardware il Leap è composto da due fotocamere e da tre LED infrarossi. Questi generano una luce con una lunghezza d'onda di 850 nanometri, poco al di sopra di quelle visibili dall'uomo, e proprio tramite l'analisi della luce riflessa, catturata dalle due fotocamere, viene creata una mappa virtuale delle mani. All'uso di luce infrarossa sono però dovuti i limiti dello spazio di acquisizione del dispositivo, limitato a circa un metro intorno a questo, come mostrato in figura 2.3.

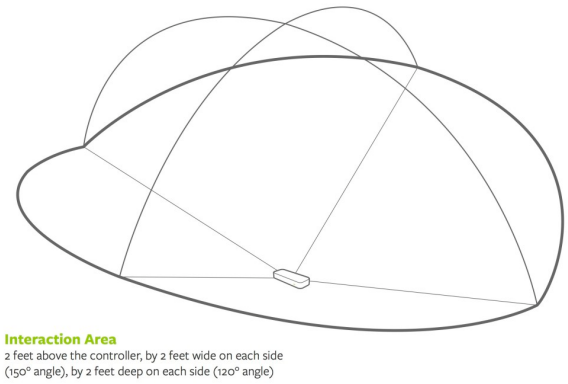

Figura 2.3: Area di interazione

Infatti questo range è condizionato dalla distanza di propagazione della luce data dai Led, a loro volta limitati dall'uso della sola energia che si può ottenere attraverso una connessione USB. Ricavare informazioni da oggetti più distanti del metro, nelle attuali condizioni di funzionamento, avrebbe portato a dei risultati sempre più imprecisi al crescere della distanza.

I dati grezzi acquisiti dalla periferica, nell'ordine di grandezza dei 200 frame per secondo, vengono quindi inviati al Leap Motion Service, installato sul PC a cui il dispositivo è collegato. A questo punto, tramite avanzati algoritmi matematici, i dati vengono filtrati ed elaborati, estrapolandone le informazioni sulla posizione delle mani fin nei minimi dettagli. Queste informazioni sono quindi inviate, su richiesta dei client collegati, utilizzando uno dei due protocolli disponibili, TCP [16] e WebSocket [19], in base alla natura degli stessi.

Il Leap mette a disposizione degli sviluppatori delle API in svariati linguaggi, JavaScript, C#, C++, Java, solo per citarne alcuni, oltre a delle estensioni per due dei maggiori Game Engine disponibili, l'Unreal e Unity. Il plugin per quest'ultimo, oltre a tutte le normali funzioni per la gestione dei frame di dati, e per la visualizzazione delle mani virtuali, offre delle funzionalità aggiuntive, che permettono un'interazione più naturale con gli

oggetti virtuali. Infatti i normali motori fisici, seppure molto avanzati, normalmente non sono pensati per un uso così particolare come quello permesso dal Leap. Grazie a queste funzionalità aggiuntive però, è possibile simulare delle interazioni veramente realistiche, permettendo di afferrare, sfiorare e manipolare gli oggetti virtuali correttamente configurati.

Elencati i punti di forza del dispositivo, va detto che questo al momento presenta un grande limite: è possibile utilizzarlo solamente in ambiente desktop. Una versione dell'SDK per dispositivi Android è al momento in fase di Alpha, e dovrebbe essere rilasciata in tempi brevi, ma, nel periodo di creazione del prototipo discusso in questo elaborato, non è stata resa disponibile. Quindi, per l'utilizzo delle funzionalità del Leap anche in ambiente mobile, si sono dovuti affrontare dei problemi che, in un prossimo futuro, non dovrebbero più essere presenti.

2.3 Architettura

Si possiedono ora tutti gli elementi per poter definire una possibile struttura architetturale di base, derivata dalle caratteristiche che il software dovrà avere e dalle funzionalità offerte dalle tecnologie impiegate per la sua costruzione.

Il software, secondo quanto già detto, dove permettere di assistere e/o partecipare all'attività di modellazione a più persone contemporaneamente. Questo comporterebbe l'avere a disposizione per ogni dispositivo una periferica Leap, cosa non possibile sia per un limite nelle risorse disponibili, sia a causa del dispositivo stesso, che, come si è già detto, al momento non offre un SDK per dispositivi mobili. Di conseguenza il software deve essere composto almeno da due parti distinte: una in esecuzione su un server desktop, necessaria per la gestione di un unico Leap condiviso, e un'altra in esecuzione sui dispositivi mobili, di cui vanno definite le funzionalità.

Ci sono a questo punto due alternative:

- Un server che si limita a recuperare le informazioni sulle mani dalla periferica hardware e ad inviarle a tutti i client disponibili, che così possederebbero tutti una versione del modello dati distinta e indipendente l'uno dall'altro;

- Il server mantiene una versione unificata del modello di dati del dominio e si occupa del loro aggiornamento in base all'input dato dalle mani. In questo caso ai client verrebbero inviate solo le informazioni su ciò che deve essere visualizzato.

Il secondo caso risulta essere il più adatto. Permette infatti di ottenere un modello dati unitario e indipendente dalla presenza o meno di un client collegato. In più riduce al minimo i problemi di sincronizzazione fra questi, andando ad alleggerire contemporaneamente il loro carico di lavoro. La macchina utilizzata come server avrà infatti molte più capacità computazionali rispetto a dei semplici dispositivi mobili, cosa che, in un caso d'uso come quello dato dalla modellazione 3D, può essere necessaria.

Riassumendo, il prototipo sarà costruito in un'ottica client-server, dove il server mantiene le informazioni relative al mondo virtuale in cui si sta operando, rendendole disponibili a tutti i dispositivi client che le richiedono, e che di conseguenza sono deputati solamente al rendering del modello. I dispositivi client, nello specifico, consistono in dispositivi Android utilizzati in modalità AR, grazie all'ausilio per esempio dei Google Cardboard. La scelta di utilizzare i dispositivi client solamente come elementi per il rendering, separandoli completamente dalla logica dell'applicativo, permette di cambiare teoricamente in modo rapido e indolore i dispositivi di output impiegati, che quindi potrebbero consistere in ben più avanzati visori per la realtà aumentata, quali Moverio e Meta2. La parte principale del programma risulta quindi essere quella in esecuzione sul server, che si occupa di reperire le informazioni sulle mani, rese disponibili dai driver di controllo

del Leap Motion, e in base a queste andare a modificare il modello virtuale utilizzando gli algoritmi di modellazione necessari. Proprio a questo scopo, si potrebbe valutare l'uso di strumenti di modellazione esterni al software, andando quindi ad alleggerire il carico di lavoro necessario alla reimplementazione di algoritmi ben noti in letteratura. Il seguente schema concettuale riassume le caratteristiche sopra descritte.

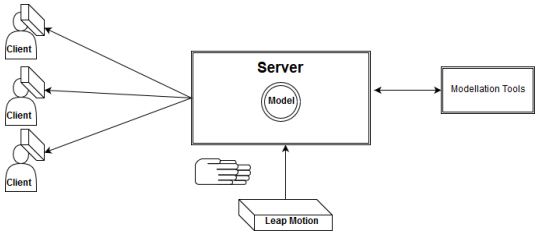

Figura 2.4: Architettura di massima

Scendendo maggiormente nello specifico, la struttura del sistema lato server dovrebbe a grandi linee assumere la forma illustrata in questo schema:

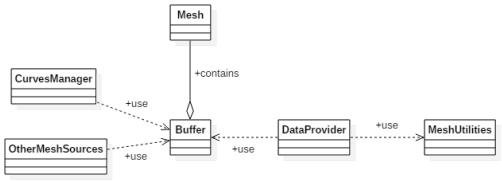

Figura 2.5: Struttura server

Il centro risulta essere la classe Buffer, incaricata di conservare le informazioni sulle Mesh generate internamente al prototipo, o attraverso degli strumenti esterni. Questi dati verrebbero poi utilizzati dall'entità DataProvider, incaricata di distribuire i dati contenuti nel Buffer ai client collegati. Questi, quindi, sarebbero solamente composti da una controparte della classe DataProvider, incaricata della creazione effettiva degli oggetti per il loro

rendering. Questo verrebbe fatto con il supporto della classe MeshUtilities, contenente tutti gli strumenti per la creazione di oggetti virtuali adeguati all'uso in AR. DataProvider è incaricata anche dello stanziamento di questi oggetti all'interno del server, in modo che sia possibile interagirvi attraverso il Leap Motion. Qualsiasi modifica di questi oggetti porterebbe, grazie all'uso di specifici listeners, alla modifica dei dati presenti nel Buffer, e quindi, alla loro rappresentazione su server e clients. Questa descrizione, certamente non esaustiva, verrà affinata nei prossimi capitoli, dove si entrerà nel merito di tutti gli aspetti realizzativi, da quelli concernenti la modellazione a quelli sulla trasmissione dei dati, e si fornirà man mano una descrizione dettagliata degli elementi deputati ai vari compiti all'interno del sistema.

Capitolo 3

Integrazione Leap-Vuforia

A partire da questo, e per i prossimi tre capitoli, si tratteranno nel dettaglio i problemi affrontati nella fase implementativa e le soluzioni che sono state adottate.

Questa prima parte è dedicata alla scelta della piattaforma di sviluppo e alla descrizione dell'integrazione delle tecnologie Leap e Vuforia.

3.1 Unity

Viste le caratteristiche del framework Vuforia e delle API del Leap Motion, si è scelto di implementare il prototipo utilizzando il game engine Unity 3D [17]. Infatti, come si è detto nei relativi capitoli, sia per Vuforia, che per il Leap, sono rese disponibili delle estensioni per questo engine, in modo da poterne sfruttare facilmente le funzionalità, oltre a offrire, nel caso del secondo, delle funzioni aggiuntive.

Questo game engine multi piattaforma, rilasciato nel 2005, permette la creazione di grafica 2D e 3D attraverso l'uso di due diversi linguaggi di programmazione, Javascript (non reale JavaScript, ma una versione customizzata, basata su linguaggi .NET) e C#. Ma, soprattutto, permette l'esportazione dell'applicazione risultante su svariate piattaforme, che van-

no da quelle dei più comuni dispositivi mobili, quali Android e iOS, ai più ricercati sistemi per la realtà virtuale, quali GearVR e Oculus Rift.

Dal punto di vista del funzionamento, il lavoro di programmazione all'interno di Unity si basa sull' uso di oggetti: i cosiddetti "Game Objects". A questi elementi, che possono avere o no una rappresentazione grafica, possono essere associati degli script, tutti estensioni della classe base "MonoBehaviour", che permettono di definirne il comportamento grazie all'uso di particolari event handler, chiamate funzioni evento. Queste procedure vengono richiamate automaticamente al sopravvenire di determinate circostanze, per esempio in caso di collisione con un altro oggetto (OnCollisionEnter) o all'inizio del ciclo di vita dell'oggetto stesso(Start), permettendo al programmatore di specificare tutte le funzionalità che si vogliono dare all'elemento.

```
public class UnityObject : MonoBehaviour {

   // Use this for initialization
   void Start () {}

   // Update is called once per frame
   void Update () {}
}
```

Sebbene il linguaggio principale utilizzabile, il C#, possa portare a pensare che Unity si fondi sull'uso del .NET Framework della Microsoft, è interessante notare come invece utilizzi Mono [12], una sua implementazione open source, dotata di un suo compilatore e di un suo Runtime. Ed è proprio l'utilizzo di questo framework che impone i limiti di compatibilità di Unity stesso. Infatti, la versione ufficiale di Mono utilizzata al momento offre una compatibilità con l'ambiente .NET alle versioni 2.0/3.5, non mettendo a disposizione quindi delle funzioni molto utili delle più recenti versioni del framework Microsoft, e limitando pesantemente l'utilizzo di

plugin di terze parti. Va però detto che una nuova versione di Mono, che offre una compatibilità con la versione 4.6 di .NET è attualmente in beta e si pensa che diverrà il Runtime ufficiale a partire dalla versione 6 di Unity.

3.2 Integrazione Plugin

Il primo aspetto da curare nella fase implementativa è l'integrazione e l'uso delle risorse di Vuforia e Leap all'interno di Unity, che risultano essere molto banali. Infatti insieme al codice vengono rilasciati dei Game Object già configurati che incapsulano il funzionamento delle due risorse. Quindi, per accedere alle funzionalità base, è necessaria solamente una breve configurazione di questi.

Nello specifico, per quanto riguarda Vuforia, la configurazione si limita alla sostituzione della telecamera tradizionale di Unity con una sua versione modificata: l'ARCamera. A questo punto, dopo aver impostato il Marker voluto, sarà sufficiente posizionare gli oggetti virtuali che si vogliono mostrare all'utente in una posizione gerarchicamente inferiore rispetto all'oggetto rappresentate il target nella scena. Non è necessario fare altro: durante l'esecuzione Vuforia si occuperà automaticamente del riconoscimento del Marker tramite videocamera, e quindi della visualizzazione degli oggetti virtuali in accordo con la prospettiva dell'utilizzatore.

Anche per il Leap la configurazione è altrettanto semplice: dopo aver posizionato nella scena dei Game Object deputati alla gestione dei dati generati dalla periferica e aver scelto il modello di mani virtuali da visualizzare, saranno questi a occuparsi del reperimento dei dati, della loro trasformazione e, infine, della visualizzazione e dell'aggiornamento delle mani virtuali generate.

Se gestire questi aspetti risulta inaspettatamente semplice, non lo è altrettanto la costruzione della parte necessaria alla gestione dei dati delle Mesh e alla loro condivisione tra client e server, come si vedrà nei capitoli successivi.

Capitolo 4

Comunicazione e Sincronizzazione Client-Server

In questo capitolo verrà illustrato come all'interno del prototipo i dati vengono effettivamente sincronizzati fra server e client, senza però prendere in considerazione la natura degli stessi. Questo aspetto verrà trattato nella sezione riguardante le Mesh e la loro gestione.

4.1 Unity Multiplayer

Un altro vantaggio dato dall'uso di Unity è che, anche per quanto riguarda il collegamento fra i dispositivi client e il server, ci si è avvalsi del protocollo per il Multiplayer messo a disposizione dal game engine. Questo permette la definizione di oggetti di rete, il cui stato viene sincronizzato automaticamente fra i vari dispositivi facenti parte la stessa. Purtroppo però tali oggetti devono essere definiti a priori dal programmatore, cosa non possibile in questo applicativo, in quanto lo scopo dello stesso è proprio quello di poter creare e modificare dinamicamente dei modelli tridimensionali. Però, oltre a questa funzione, è messa a disposizione del programmatore la possibilità di definire delle procedure RPC (Remote Procedure Call). La chiamata di

queste funzioni su oggetti del server, comporta la chiamata delle medesime sugli stessi oggetti stanziati sui client, come mostrato nello schema 4.1.

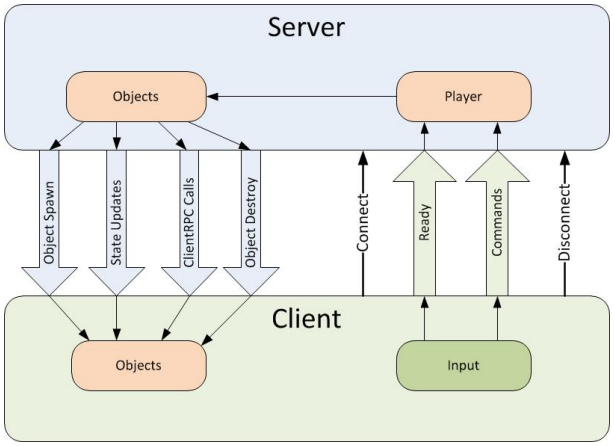

Figura 4.1: Network Unity

È Unity stesso che si prende in carico l'onere della serializzazione dei parametri e del loro invio.

4.2 Sincronizzazione dati client-server

Fatta questa precisazione, si può procedere dicendo che le classi incaricate della gestione di questo aspetto del prototipo sono due: Buffer e DataProvider.

Il Buffer è una classe di cui, grazie all'uso del pattern Singleton, è disponibile una sola istanza all'interno del server e, come il nome suggerisce, ha il compito di mantenere una copia dei dati degli oggetti che devono essere visualizzati dai client. Questa classe mette a disposizione delle procedure per l'aggiunta, l'aggiornamento e la rimozione dei dati, garantendo la consistenza degli stessi grazie al controllo sulla lettura e la scrittura effettuato mediante delle lock sull'oggetto stesso. È la seconda delle classi nominate

precedentemente che si occupa dell'effettivo invio dei dati al client, il Data Provider. Questo script Unity, legato a un Game Object in esecuzione sia sul server che sul client, si incarica di verificare in ogni frame la presenza di nuovi dati all'interno del Buffer e quindi, in base a questi, di aggiornarne la rappresentazione sia lato server che lato client. Questo effetto è raggiunto grazie all'uso delle funzioni RPC sopra introdotte.

```
if (isServer && buffer.isNewDataAvailable()) {
            foreach (var item in buffer.getData())
            {
                if (currentObjects.Contains(item.Key)) {
                    if (item.Value.IsGeometryChanged) {
                        RpcUpdateObjectGeometry(item.Key,
                            item.Value.Vertices, item.Value.Codec);
                        UpdateObjectGeometry(item.Key,
                            item.Value.Vertices, item.Value.Codec);
                    }
                    if (item.Value.IsWorldChanged) {
                        RpcUpdateObjectWorldMatrix(item.Key,
                            item.Value.World);
                        UpdateObjectWorldMatrix(item.Key,
                            item.Value.World);
                    }
                } else {
                    RpcCreateObject(false, item.Key,
                        item.Value.Vertices, item.Value.World,
                        item.Value.Codec);
                    GameObject actualGO = CreateObject(true,
                        item.Key, item.Value.Vertices,
                        item.Value.World, item.Value.Codec);
                }
            }
}
```

Come si può notare dal codice soprastante, in base al contenuto del Buffer viene scelto qual è l'azione da intraprendere sull'oggetto in esame ed eseguita sia sulla versione dell'oggetto mantenuta nel server, sia su quelle mantenute nei client.

4.3 Leap Motion su Mobile

Le chiamate RPC risultano utili anche per la risoluzione di un altro problema, non ancora accennato, e relativo all'interazione fra le mani fisiche e gli oggetti virtuali. Questo problema nasce dalla differenza nei sistemi di riferimento utilizzati dal Leap nella rappresentazione delle mani, e da Unity nel rendering del modello dati. Precisamente, il Leap Motion lavora con delle coordinate definite in millimetri a partire dal centro del controller stesso, mentre in Unity, invece, la grandezza di un oggetto, e quindi le distanze fra gli elementi della scena, sono basati sulle stime che Vuforia esegue a partire dalla grandezza esatta del Marker nel mondo reale, parametro impostabile al momento della sua creazione. Sapendo questo, risulta abbastanza semplice fare in modo che le mani virtuali all'interno della scena abbiano la stessa grandezza e posizione di quelle reali, essendo entrambi i sistemi di riferimento basati su quello metrico. Infatti, se le dimensioni del Marker vengono impostate correttamente, e le risorse vengono collocate facendo in modo che il centro del Marker di Vuforia e quello del controller del Leap Motion coincidano, i dati generati da quest'ultimo possono essere utilizzati, senza bisogno di alcun adattamento, all'interno della scena Unity, ottenendo quindi l'effetto di mostrare le mani virtuali sovrapposte alla controparte reale catturata dalla telecamera (fig 4.2).

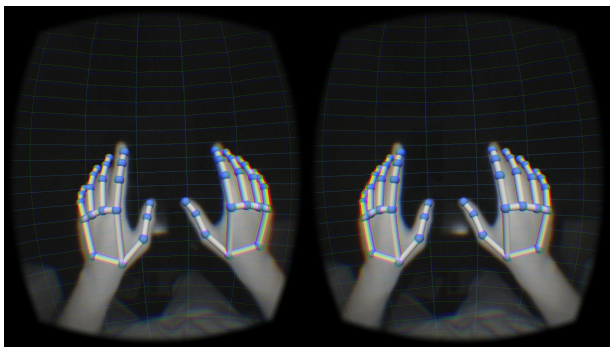

Figura 4.2: Sovrapposizione Mani

Purtroppo questa sovrapposizione non risulta essere abbastanza precisa da permettere di manipolare gli oggetti virtuali senza che l'immagine simulata venga mostrata. Quindi, tornando al problema introdotto a inizio paragrafo, si rende necessario inviare ai client anche le informazioni sulle mani perché queste vengano rappresentate.

Questa complicazione potrebbe sembrare facilmente risolvibile solamente grazie all'utilizzo delle chiamate RPC per l'invio dei dati di posizione sulle mani. Purtroppo però, utilizzando le tecniche di serializzazione native a disposizione in Unity, i tempi dedicati al processo, uniti alla grande mole di dati gestita, portano a un degrado evidente nelle prestazioni dell'applicativo con conseguenti lag. Per evitare questo si è fatto uso di un protocollo di serializzazione più performante, il MessagePack, congiuntamente ad uno dei più veloci algoritmi di compressione disponibili, l'LZ4 [10].

MessagePack [11] è un protocollo di serializzazione binario disponibile al momento in più di 50 linguaggi di programmazione differenti. La sua velocità ed efficienza derivano, oltre all'uso del binario in sé, formato molto più leggero rispetto per esempio al JSON [8], basato su una codifica testuale delle informazioni, all'utilizzo di contratti per la definizione delle strutture dati serializzabili. Definendo a priori, per esempio, quali siano i campi di una classe serializzabili e il loro ordine di serializzazione, questa risulterà

estremamente più efficiente rispetto a quella offerta dai procedimenti più tradizionali, come per esempio quello basato sul BinaryFormatter, disponibile nativamente in C#, in cui deve essere compiuto un lavoro di inferenza per determinare quale sia il formato dei dati da codificare. Un confronto fra le performance di questi due ed altri protocolli di serializzazione più famosi è mostrato nel seguente grafico (fig. 4.3).

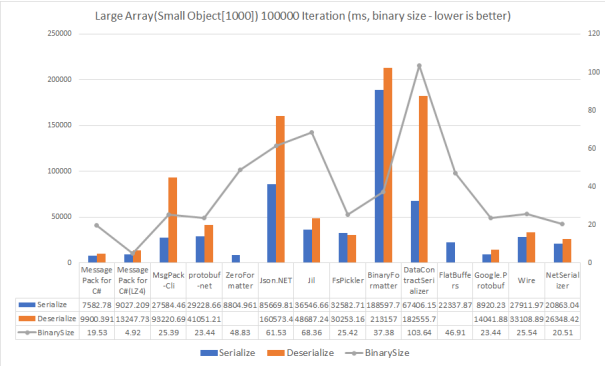

Figura 4.3: Confronto serializzazione

La velocità di serializzazione offerta da questo protocollo, unitamente alla compattezza dei dati raggiunta grazie a LZ4, un algoritmo di compressione capace di raggiungere i 600 MB/s in fase di compressione e i 3000 MB/s in decompressione (una trattazione più accurata di questa e di altre tecniche verrà fatta nella sezione relativa alle Mesh), permette di poter inviare i dati sulle mani dal server ai client senza alcuna ripercussione negativa sulle performance.

Nello specifico questo processo è gestito tramite due script Unity: il NetworkHandsController e il NetworkHandsUpdater. Il primo dei due, si occupa del reperimento dei frame dati e della loro codifica.

```
private void Update()
    {
        if (newFrameAvailable)
```

```
        {
            var serializedFrame =
                LZ4MessagePackSerializer.Serialize(new
                FrameSerial(frameCopy));
            handsUpdater.setFrame(serializedFrame);
            newFrameAvailable = false;
        }
    }
```

Il secondo invece è incaricato della trasmissione dei dati fra server e client e dell'aggiornamento delle rappresentazioni delle mani.

```
void Update () {
        if (isClient && updateAvailable)
        {
            UpdateHandRepresentations(graphicsHandReps,
                ModelType.Graphics, newFrame);
            updateAvailable = false;
        }
}

public void setFrame(byte[] frame) {
        int i = 0;
        byte[][] result = frame.GroupBy(s => i++ / 4096).Select(g
            => g.ToArray()).ToArray();
        int code = (int)DateTimeOffset.Now.UtcTicks;
        i = 0;
        foreach (var x in result)
        {
            RpcSetFrame(code, result.Length, i++, x);
        }
}
```

È importante notare come queste misure per la serializzazione e la tra-

smissione dei dati siano state rese necessarie da una mancanza nell'SDK del Leap. Infatti, secondo la documentazione, teoricamente questo dovrebbe offrire delle procedure native per la serializzazione dei frame di dati acquisiti, ma che, in pratica, risultano non implementate. Questo risulta essere un grave difetto, in quanto la serializzazione e l'invio sono l'unica modalità di accesso ai dati da dispositivi mobili.

Oltretutto, per rendere possibile la definizione dei contratti di serializzazione, che normalmente avviene tramite annotazioni sui campi di classe, si è dovuto procedere a una reimplementazione delle classi deputate alla rappresentazione nel paradigma a oggetti delle informazioni sulle mani. Questo perché l'SDK del Leap per Unity non mette a disposizione il loro codice sorgente, ma solamente una versione precompilata.

Capitolo 5

Codifica e Ottimizzazione di Mesh Poligonali

È arrivato il momento di trattare un argomento fino ad ora ignorato, ma che costituisce una parte essenziale del software in esame: la gestione delle Mesh poligonali. Con questo termine si definiscono degli oggetti tridimensionali, dati dal contributo di vertici e facce, che costituiscono l'elemento essenziale di qualsiasi attività di modellazione 3D. L'infrastruttura delineata nei capitoli precedenti nasce proprio allo scopo di conservare e trasportare i dati relativi a questi oggetti, e proprio la loro struttura è uno dei fattori principali di cui si è dovuto tenere conto nella realizzazione dei protocolli di trasporto.

5.1 Struttura Mesh

La rappresentazione più comune di una Mesh, adottata anche all'interno di Unity, ha il difetto di generare una mole di dati non indifferente, cosa che in un applicativo distribuito e con dei vincoli temporali abbastanza stringenti, dati dal dovere mostrare nel minor tempo possibile il risultato di

una interazione all'utente, potrebbe portare a ritardi e malfunzionamenti se non gestita in modo adeguato.

La grande mole di memoria necessaria alla memorizzazione è data dalla natura stessa delle Mesh, che, come si è detto, sono strutturate in un reticolo di vertici e facce. La loro rappresentazione, quindi, consiste in due insiemi per ognuna: il primo contenente la lista dei vertici; il secondo la lista delle facce, nel nostro caso dei triangoli, espressa come lista dei riferimenti ai suoi spigoli.

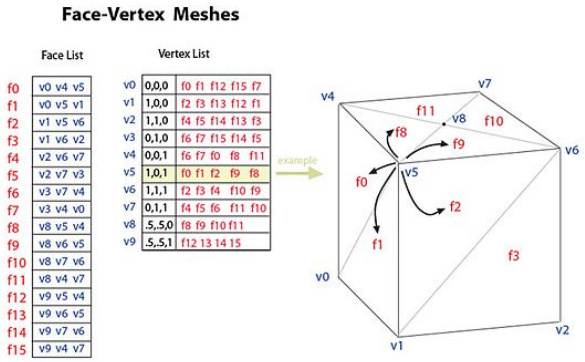

Figura 5.1: Face-Vertex Mesh

Per le dimensioni basti pensare che una Mesh molto basilare di 98 vertici ha una dimensione in memoria di circa 3,4 Kbyte. Preso di per sé questo valore potrebbe sembrare trascurabile, ma unito al frame rate del software, che potrebbe comportare l'aggiornamento di questi dati oltre le 60 volte al secondo, e alla presenza di più client contemporaneamente, si trasforma in un volume considerevole anche per gli oggetti più semplici.

Gli accorgimenti che sono stati presi per ottimizzare questo aspetto si possono riassumere in due categorie: la compressione e il filtraggio dei dati da spedire.

5.2 Filtraggio Dati

La risposta più frequente a chi lamenta problemi nel trasporto dei dati, e quindi cerca dei modi per poter migliorare questo aspetto, è molto semplice: il miglior modo per ottimizzare l'invio dati è non inviarne. Sebbene a prima vista possa sembrare un consiglio paradossale e non molto utile, si è dimostrato invece il primo passo per l'ottimizzazione del protocollo di trasporto creato. Infatti, la maggior parte delle operazioni sulle Mesh, nella gran parte dei casi, comportano una modifica solamente parziale dei dati, rendendo possibile, utilizzando i corretti accorgimenti, inviare solamente le informazioni strettamente necessarie al loro aggiornamento.

È il caso, per esempio, delle modifiche che comportano la modifica globale di una Mesh in termini di posizione, scalatura o rotazione. In un caso del genere trasmettere le informazioni su tutti i vertici sarebbe inutile, essendo la modifica la stessa da applicare a tutti. Perciò si rende necessario solamente inviare una matrice rappresentante la posizione della Mesh nel mondo virtuale per poterla aggiornare.

Anche nel caso la modifica non sia globale, ma localizzata a un numero finito di vertici e facce, si possono limitare i dati trasmessi. Infatti, se l'alterazione non comporta il cambiamento della struttura generale della Mesh, non sarà necessario trasmettere tutte le informazioni di collegamento fra i vertici per la composizione delle facce, che risultano essere la parte più gravosa, ma solamente il nuovo valore dei punti modificati.

Si può trovare un riscontro dell'utilizzo di questi metodi nella struttura stessa del Buffer, che distingue i dati delle Mesh conservate in due tipologie: quelle geometriche, relative alla struttura stessa dell'oggetto, e quelle di trasformazione, necessarie alla modifica globale delle Mesh.

```
public void addData(string name, UnityEngine.Vector3[] world)
{
   ...
}
```

```
public void addData(string name, byte[] vertices)
{
    ...
}
```

Come si può notare da questa porzione di codice, mentre i vettori di scalatura, rotazione e posizione, sono codificati usando un array multidimensionale, le informazioni geometriche sono trasmesse come array di byte. Questo, oltre a permettere una maggiore flessibilità nella struttura dei dati trasmessi, essendo possibile adattare il contenuto del messaggio alle più svariate esigenze senza dover modificare la struttura del Buffer stesso o del protocollo descritto nel capitolo precedente, permette di poter applicare sui dati delle tecniche di compressione per poter diminuire ulteriormente la loro dimensione.

La rappresentazione dei dati geometrici nell'implementazione scelta è vincolata al contratto fornito dall'interfaccia GeometryMessageWrapper:

```
interface GeometryMessageWrapper
{
   bool isOptimized();
    Vector3[] getVertices();
    int[] getFaces();
    int[] getIndexes();
}
```

Come si può vedere, oltre alle informazioni su vertici e facce, questa presenta un campo booleano "isOptimized". In caso questo valore risulti positivo nel messaggio non saranno presenti tutte le normali informazioni sulla struttura, ma solamente quelle sui vertici modificati, necessari all'aggiornamento della Mesh.

5.3 Compressione

Ci sono delle occasioni però in cui non si può fare a meno di inviare le informazioni in modo completo, per esempio al momento della connessione di un client o in caso di modifiche radicali nella composizione di un oggetto. In questi casi la compressione risulta l'unico modo per migliorare le performance del software.

Questo problema non è nuovo in campo informatico. Infatti, nel corso degli anni, sono vari gli algoritmi e le tecnologie nate in questo ambito: partendo dal protocollo OpenCTM [13], pensato appositamente per la codifica compatta di mesh triangolari, fino ad arrivare al nuovissimo e molto più performante Draco [5]. Questa libreria, rilasciata dalla Google a inizio 2017 e nata pensando soprattutto alla velocità e all'efficienza, permette la compressione di Mesh triangolari e di nuvole di punti. Una ulteriore alternativa, anche questa recentissima e nata dal lavoro di due ricercatori italiani, Federico Ponchio e Matteo Dellepiane, è Corto [4], libreria anch'essa dedicata alla compressione di Mesh e nuvole di punti.

Queste librerie, tutte disponibili in C++ o Javascript, si basano principalmente sull'utilizzo di due tecniche: una precisione ridotta nella codifica dei valori e l'uso di rappresentazioni sulla connettività delle Mesh, che come si è già detto risultano essere le più gravose, altamente ottimizzate. Le differenze nella codifica di queste informazioni sono rispecchiate dalle differenze nelle performance, che, come si può vedere dal loro confronto , vedono OpenCTM leggermente svantaggiato.

Questo infatti utilizza per la codifica semplicemente delle tecniche di compressione entropica, cosa che risulta essere meno performante rispetto per esempio a Draco, che, tra le altre cose, fa uso dell'algoritmo "Edgebreaker" [27]. Sviluppato dal professor Jarek Rossignac del Georgia Institute of Technology, questo algoritmo, descritto in modo molto semplicistico, si basa sull'utilizzo di una macchina a stati finiti che, nell'attraversare la Mesh di triangolo in triangolo, codifica le informazioni sulla connettività utilizzando

dei simboli prestabiliti (nella variante originale questi sono C, L, E, R ed S). Questi simboli, come si può vedere dall'immagine 5.2, permettono di ricostruire le informazioni sul collegamento di ogni singolo triangolo, e quindi la topologia della Mesh intera.

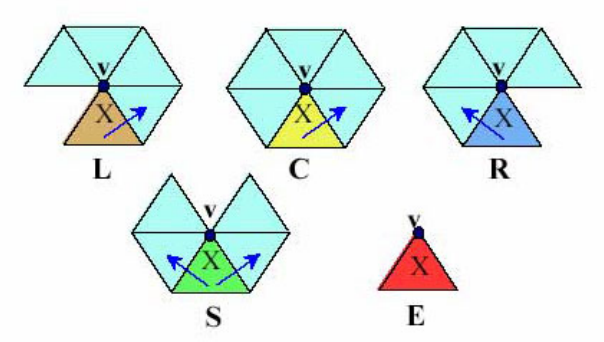

Figura 5.2: Codifica EdgeBreaker

Utilizzando questa tecnica sono necessari circa 2 bit per ogni faccia, portando il consumo di memoria a 2n bit o meno per le Mesh più semplici. L'algoritmo utilizzato da Corto si basa sul medesimo principio.

Dato il miglioramento di performance promesso da queste librerie si è provveduto alla realizzazione di wrapper che ne permettessero l'uso da C# (come richiesto da Unity), essendo come si è già detto disponibili solamente in C++ e Javascript, in modo da verificarne le caratteristiche e valutarne l'uso all'interno del prototipo. Questo si è reso necessario in quanto, essendo tecnologie molto recenti, non è presente al momento una documentazione tale da riuscire a valutarne le caratteristiche senza il loro uso diretto, e quindi l'unico modo per scoprire se si adattassero effettivamente al funzionamento del resto del prototipo è stato verificarne in prima persona il funzionamento.

I test, per quanto abbiano portato concretamente dei buonissimi risultati dal punto di vista delle performance, sia in termini di tempo che di spazio, hanno dimostrato una incompatibilità fondamentale di queste librerie con l'architettura precedentemente descritta. Gli algoritmi come Edgebreaker

hanno il difetto, se così può essere definito, di creare una topologia certamente identica all'originale nel risultato finale, ma differente nella sua struttura. Infatti, per ottenere una minore occupazione di memoria, l'ordine di vertici e facce vengono modificati, rendendo impossibile applicare le tecniche di aggiornamento localizzato illustrate nel paragrafo precedente. Per essere più chiari, l'utilizzo di tali tecniche avrebbe comportato differenze fra la struttura posseduta lato server e quella lato client, a seguito della codifica e della decodifica attuate per mezzo della libreria. Si sono fatti dei tentativi per ovviare a questo problema consistenti nella trasmissione di informazioni aggiuntive atte alla ricostruzione della topologia originale, ma questi hanno portato a un degrado di prestazioni tale da essere paragonabili ai risultati dati da tecniche molto più semplici e meno laboriose, che quindi sono state preferite nell'implementazione finale. Precisamente, si sono utilizzati degli algoritmi per la compressione generici, coadiuvati dall'utilizzo di filtri sui dati che ne migliorassero la compressibilità.

Per quanto riguarda gli algoritmi, la scelta è ricaduta su Snappy [15], realizzato dalla Google e largamente utilizzato nei loro protocolli RPC, e su LZ4 [10], già utilizzato per la compressione dei dati generati dal Leap. Fra i più efficienti e veloci in circolazione, entrambi appartengono alla classe degli algoritmi lossless (senza perdita di informazione) di codificazione LZ77. Questo schema di codifica si basa sulla sostituzione delle occorrenze ripetute di una determinata porzione dei dati con un riferimento alla posizione dei dati già incontrati. In pratica durante la codifica, quando si trova un insieme di dati che già si era incontrato in precedenza, si procede alla sua sostituzione con un puntatore alla posizione di queste informazioni nello stream precedentemente processato. Logicamente questo metodo comporta un rate di compressione maggiore all'aumentare della ridondanza dei dati, e proprio a questo scopo si sono utilizzati dei filtri mirati alla semplificazione delle informazioni attraverso l'uso dei principi della codifica delta.

Il primo è stato applicato sui vertici, rappresentati come triplette di valori float a 32 bit. Questi numeri, codificati secondo lo standard IEEE754

[7], normalmente risultano difficilmente comprimibili e, anche le tecniche di delta encoding, basate sulla trasmissione non dei valori stessi ma delle loro differenze, in modo da trattare dei numeri teoricamente più piccoli e quindi con un maggiore numero di zeri nella loro rappresentazione, risultano inefficaci data la struttura del formato (fig. 5.3).

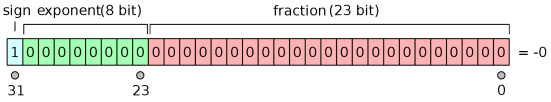

Figura 5.3: Codifica IEEE754

Per riuscire ad aumentare il numero di zeri nella rappresentazione si è ricorso a una codifica XOR. In pratica, ogni vertice viene processato eseguendo l'operazione XOR, operazione booleana che restituisce vero solo in caso di ingressi con valore differente, con il vertice precedente. L'utilizzo di questa operazione risulta vantaggiosa proprio a causa della conformazione teorica di una Mesh, dove risulta che i punti sono sparsi tutti in uno spazio delimitato, e quindi vertici consecutivi hanno spesso un valore molto simile fra loro. Questo si traduce nella codifica IEEE754 in valori dei primi 9 bit, quelli relativi ad esponente e segno, molto simili, e quindi grazie alla codifica XOR teoricamente si riescono ad ottenere dei valori molto più omogenei.

Si potrebbero aggiungere a questo dei procedimenti di quantizzazione, per diminuire ulteriormente l'occupazione di memoria, ma per le esigenze di performance richieste non si è rivelato necessario, e quindi si è preferito conservare la precisione originale.

Il secondo filtro è stato posto sui valori degli indici contenuti nella lista delle facce. Dalla loro osservazione si è notato come nella maggior parte dei casi la loro sequenza fosse composta da valori positivi (si fa notare come questo sia sempre vero essendo questi degli indici riferiti al vettore dei vertici) in ordine crescente. Data questa caratteristica è stato quasi forzato

l'utilizzo di una codifica delta incrementale, in base alla quale piuttosto che il valore vero e proprio viene memorizzata la differenza fra valori consecutivi. Per esempio, in vece della lista di valori [1, 2, 3], secondo questo metodo andrebbe considerata la lista [1, 1, 1]. Se la distribuzione dei valori in termini di differenze risulta essere abbastanza lineare, questo tipo di codifica può portare a enormi miglioramenti negli algoritmi LZ77, fondati proprio sulla ripetizione.

Va detto però che le liste delle facce, sebbene per la maggioranza della loro lunghezza presentino queste caratteristiche, tendono in alcune loro parti ad avere dei valori discontinui molto differenti fra loro. La discrepanza fra questi valori può portare alla generazione di differenze negative, cosa che potrebbe vanificare il risparmio di spazio ottenuto. Per evitare questo è stata effettuata una ricodifica delle differenze attraverso la tecnica di ZigZag encoding. Grazie a questo algoritmo molto semplice si effettua un'operazione di mappatura dei numeri negativi in numeri positivi. In pratica il numero -1 viene mappato nel numero 1, che a sua volta viene mappato come 2 e via di seguito (proprio da questa alternanza fra valori negativi e positivi l'algoritmo prende il nome). La sua implementazione risulta essere estremamente semplice e veloce, coinvolgendo solamente operazioni solitamente svolte in un solo ciclo macchina. Qui di seguito è riportato il codice per la codifica e la decodifica.

```
int ZigZagEncode(int i)
{
   return (i >> 31) ^ (i << 1);
}

int ZigZagDecode(int i)
{
    return (i >> 1) ^ -(i & 1);
}
```

Grazie a queste tecniche si è ottenuta una diminuzione media di circa il 50% nella grandezza dei dati geometrici, con picchi sul 75-80% (il valore varia in base alla conformazione della Mesh). Di sicuro questo non risulta essere il miglior risultato raggiungibile, anche solo se paragonato ai risultati dati dalle librerie sopra descritte, ma risulta abbastanza per dimostrare come la trasmissione di dati geometrici, anche in un contesto con dei tempi abbastanza stringenti, possa essere possibile. Infatti, approfondendo tecniche e algoritmi più avanzati, si potrebbero migliorare questi risultati raggiunti con dei metodi, sì sufficienti al funzionamento del prototipo in esame, ma oltremodo rudimentali.

Capitolo 6

Tecniche di Modellazione

Nei capitoli precedenti è stata delineata un'architettura capace di trattare dei dati geometrici, permettere la loro visualizzazione attraverso dei dispositivi client, e l'interazione utilizzando solamente le mani. Dei dati trattati si sono approfonditi gli aspetti legati alla trasmissione e alla codifica, senza mai fare cenno alle modalità con cui questi dati sono creati o modificati. Proprio questo è l'argomento di quest'ultimo capitolo, incentrato sulle tecniche utilizzate per raggiungere lo scopo principale per cui questo prototipo è stato creato: la modellazione.

6.1 Creazione di Mesh in Unity

La realizzazione di Mesh a partire da punti definiti dall'utente è un processo articolato composto da più fasi. Il primo aspetto di cui ci si deve preoccupare è la raccolta dei punti base, gestita attraverso l'uso del Leap Motion. Attraverso il movimento della mano, più precisamente dell'indice, vengono definiti i punti di controllo per la creazione della curva di regressione, una curva che si avvicini il più possibile alla posizione dei dati forniti. Il metodo utilizzato per la sua costruzione è quello dei minimi quadrati: viene ricercata la funzione che minimizza la somma dei quadrati degli scarti fra i dati

forniti e la curva cercata. Una volta ottenuta la curva si procede alla sua rotazione attorno ad uno degli assi e quindi alla generazione di un oggetto tridimensionale. Dopo aver applicato un algoritmo di triangolarizzazione, in modo da definire le facce dell'oggetto, si saranno ottenuti tutti i dati necessari alla creazione di una Mesh.

Il procedimento appena descritto riassume efficacemente l'implementazione realizzata internamente al prototipo. Nello specifico le classi coinvolte sono quelle appartenenti al package Curves. Questi script, di cui il principale risulta essere CurvesManager, si occupano di tutte le fasi sopra descritte, dalla raccolta dei punti alla triangolarizzazione della Mesh, per poi affidare il risultato finale al Buffer descritto nei capitoli precedenti per la sua distribuzione.

6.2 Modifica Mesh

La modifica delle Mesh create, che come da specifiche deve permettere di gestire posizione, dimensioni e rotazione, è gestita tramite l'utilizzo delle funzionalità aggiuntive messe a disposizione dall'SDK del Leap per Unity, già introdotte nella parte introduttiva dell'elaborato. Questo set di funzionalità, riunite sotto il nome di Interaction Engine, permettono la definizione di particolari oggetti aventi le stesse caratteristiche di entità fisiche. Come tali sarà quindi possibile trattarle, interagendovi attraverso la rappresentazione delle mani creata all'interno dell'engine. La definizione di questi oggetti risulta semplice per lo sviluppatore, essendo limitata all'aggiunta di uno script, l'"Interaction Behaviour" al Game Object interessato. Da questo punto in poi sarà Unity stesso a prendere in carico la gestione dell'interazione fra mani e oggetto, registrando qualsiasi attività relativa a questa sfera, e dando la possibilità di definire dei comportamenti personalizzati. Un esempio è dato dal codice utilizzato per la gestione del cambio di dimensioni delle Mesh.

```
if (interactionController.isGrasped) {
   if (interactionController.graspingHands.Count == 1) {
      //Debug.Log("One Hand is Grasping");
    } else if (interactionController.graspingHands.Count == 2) {
        //Debug.Log("Two Hands are Grasping");
        ...
        Updating Mesh dimension according to hands distance
        ...
    }
}
```

Come si può vedere, lo script InteractionBehaviour, permette di monitorare lo stato dell'oggetto in relazione alle mani. In questo caso queste informazioni vengono sfruttate per identificare i momenti in cui la Mesh è agguantata da entrambe le mani, e, in base alla loro distanza, determinare una nuova scala da applicare. Traslazione e rotazione vengono gestiti in modo analogo.

6.3 Integrazione Strumenti Esterni

Il procedimento adottato per la costruzione di una Mesh, per quanto non sia nemmeno fra i più complicati, mostra chiaramente come l'implementazione di algoritmi per la modellazione sia un procedimento alquanto lungo e difficoltoso, e che in più non garantisce lo sfruttamento ottimale delle risorse offerte dal calcolatore. Infatti, normalmente, gli strumenti creati proprio a questo scopo offrono tutta una serie di accorgimenti, a partire da un'implementazione in linguaggi di bassissimo livello, per riuscire a raggiungere il massimo delle performance ottenibili in termini di velocità d'esecuzione e occupazione di memoria. Proprio per questo motivo, piuttosto che offrire ulteriori funzionalità interne al prototipo, si è deciso di indagare la possibilità di una possibile integrazione con un software esterno, che, occupandosi

della parte algoritmica, riesca ad offrire una sorta di ottimalità dal punto di vista delle performance, e contemporaneamente, comporti un'enorme diminuzione nei tempi di sviluppo relativi alla parte di modellazione. Di fatto, anche se questo processo renderebbe alto il costo iniziale, dovuto alla ricerca e alla realizzazione di un protocollo di trasporto adeguato, porterebbe a dei costi minimi nella fase di aggiunta di qualunque delle funzionalità offerte dal software esterno. Per quanto riguarda i problemi che questa soluzione può comportare, il principale è la latenza nelle comunicazioni che verrebbe generata dal dialogo fra l'applicativo realizzato e il software esterno. Per questo aspetto però valgono tutti i ragionamenti fatti per la parte di comunicazione fra server e client. Precisamente, dai test sull'ottimizzazione e la compressione delle Mesh, si è potuto notare come le tecniche in questo ambito permettano di ottenere ottimi risultati capaci di rendere possibile una trasmissione efficiente, anche con tempistiche abbastanza stringenti, di dati di tipo geometrico.

Fatte questa premessa, si può parlare del prodotto scelto per l'integrazione: il software di modellazione Blender [2].

6.3.1 Blender

Questo programma libero e multipiattaforma, rilasciato nel gennaio del 1995, sebbene in maggioranza scritto in C++ e C, fra le molte features legate alla produzione di immagini e video artificiali presenta una interfaccia di scripting in Python liberamente accessibile, grazie alla quale è possibile adattare il funzionamento del software alle più disparate esigenze, rendendo oltretutto disponibili per via programmatica tutte le funzioni di modellazione. Questa estrema adattabilità, unita alla solidità di un prodotto che con gli anni ha conquistato un bacino di utenze sempre maggiore grazie anche alle sue funzionalità all'avanguardia, ha inciso profondamente sulla scelta di questo software come strumento di supporto alla modellazione. Di fatto, una volta completata l'integrazione, si renderebbero accessibili le più dispa-

rate funzionalità in questo campo, semplificando non di poco il processo di creazione e modifica delle Mesh.

La creazione di plug-in Python è la modalità con cui Blender permette l'aggiunta di funzionalità. Sebbene questa interfaccia sia soprattutto mirata alla creazione di strumenti di supporto alla modellazione, non è presente nessuna limitazione sulle funzionalità implementabili in essi, tranne quelle date dal linguaggio utilizzato. Fortunatamente, il Python risulta essere uno dei linguaggi più diffusi fra gli sviluppatori, e, proprio grazie a questa sua fama, si è creata un'enorme comunità che ha reso disponibili una grande quantità di strumenti in questo linguaggio. Non è stato difficile così trovare delle ottime implementazioni di protocolli e funzioni necessarie alla comunicazione fra Blender e il software in esame, e il primo passo in questa direzione è stato proprio la definizione di un protocollo di trasporto e la sua integrazione in Blender e Unity.

6.3.2 Protocollo di Trasporto

Il bisogno principale nella ricerca del protocollo adatto si dimostra certamente la velocità. Infatti, qualsiasi ottimizzazione nei dati si dimostrerebbe vana in assenza di un mezzo di comunicazione altamente efficiente e che quindi non imponga dei limiti troppo onerosi nella comunicazione. Altro bisogno è la bidirezionalità del mezzo, che deve permettere il trasporto dei comandi da Unity e dei dati da Blender. Entrambe queste caratteristiche sono presenti nel protocollo prescelto: il WebSocket [19]. Basato su TCP, esso permette comunicazioni full-duplex fra client e server a basso overhead, facilitando così i trasferimenti di dati di tipo real-time. Questo è possibile grazie alla sua particolare implementazione che permette l'invio di messaggi da server a client senza nessuna sollecitazione da parte di quest'ultimo e lo scambio di dati su un'unica connessione sempre aperta. Queste caratteristiche, oltre a rendere il protocollo molto adatto all'uso nei videogiochi, hanno portato alla sua scelta come mezzo di comunicazione fra Blender e Unity.

Per la sua integrazione all'interno del primo, il punto di partenza è stato un lavoro precedente, svolto dall'utente GitHub Jonathan Giroux, che ha realizzato un add-on [20] capace di eseguire l'esportazione e l'invio dei dati presenti nella scena utilizzando questo mezzo. Il formato dei dati prescelto per la comunicazione in questa implementazione è il JSON (JavaScript Object Notation) [8]. Questo standard pensato per lo scambio di dati in ambito web, è basato sull'utilizzo di coppie nome/valore per la codifica e la trasmissione delle informazioni. Da questo derivano i suoi punti di forza, che risultano essere la facilità di gestione da parte delle macchine e la possibilità per le persone di poter leggere e scrivere i messaggi in modo molto intuitivo, essendo completamente codificati in forma testuale, ma anche le sue debolezze. Infatti la codifica testuale comporta un elevato overhead sia nelle dimensioni del messaggio, sia nel suo tempo di elaborazione, se confrontato all'utilizzo del binario. Benché questi aspetti negativi in alcuni casi risultino di secondaria importanza rispetto ai vantaggi che questo formato può offrire, in questo gli aspetti relativi alle performance risultano essere essenziali per il corretto funzionamento di tutto il prototipo. Di conseguenza al JSON è stata preferita una codifica binaria dei dati, che a costo di una maggiore difficoltà nella realizzazione, permettesse delle comunicazioni altamente efficienti.

Fra i tanti strumenti mirati alla serializzazione disponibili, fra cui si ricorda il formato MessagePack [11], utilizzato per la codifica dei dati relativi alle mani, si è scelto di utilizzare FlatBuffers [6]. Questa tecnologia, realizzata dalla Google e basata sul più noto progetto Protocol Buffer [14], di cui mantiene alcuni degli aspetti principali, promette un'elevata velocità di serializzazione unita a delle dimensioni nei dati abbastanza compatte. Cosa altrettanto importante, vanta il supporto per tutti i principali linguaggi di programmazione (C#, Python, Java solo per citarne alcuni), ottenendo così un formato altamente ottimizzato e indipendente dalla piattaforma. Questa caratteristica risulta essenziale, in quanto Blender e Unity utilizzano due linguaggi differenti: C# e Python. Nel principio di funzionamento di

questa tecnologia sta anche la spiegazione di questa grande portabilità. Il formato dei messaggi infatti viene definito in una sorta di pseudo linguaggio, vagamente rassomigliante al C, realizzato appositamente. I file così creati, denominati Schemi, vengono poi interpretati da un apposito compilatore, il FlatCompiler, in base ai linguaggi definiti come target. Il risultato saranno delle rappresentazioni di questi schemi nei linguaggi scelti, utilizzabili per la lettura e la scrittura dei messaggi.

Qui viene fornito un esempio della definizione di una classe nel formato FlatBuffers.

```
enum Command : byte { Rotate, Scale, Translate, Update, Create,
    Delete, New, Save}
enum MeshType : byte { Cube, Cone, Torus, Monkey }

table CommandMessage {
  command:Command;
  data:Vec3;
  info:string;
  type:MeshType;
}
```

È un'altra però la caratteristica che contraddistingue questo strumento dalla maggior parte dei suoi rivali, e cioè il particolare procedimento di serializzazione e deserializzazione adottati. Infatti, piuttosto che attuare i processi di codifica e di decodifica in un unico momento, FlatBuffers gestisce unitariamente e distintamente ogni attributo del messaggio analizzato. Per essere più chiari, ogni dato inserito in un ipotetico messaggio viene ricodificato al momento esatto dell'inserimento. In questo modo la serializzazione avviene proceduralmente, man mano che il messaggio viene creato, distribuendo il lavoro di codifica in più iterazioni. Analogo è il discorso relativo alla decodifica: i singoli dati di cui un messaggio è composto vengono deserializzati solamente quando richiesti, permettendo quindi una decodifica

parziale e mirata dei dati.

6.3.3 Realizzazione

Definito il protocollo di trasporto e trovata una modalità efficiente per la definizione di struttura e semantica dei messaggi, il lavoro di implementazione risulta semplice.

Lato Blender, la gran parte del codice si divide in quattro moduli principali:

- BlenderSocket, modulo per la gestione del server WebSocket;
- FlatMessageBuilder, deputato alla creazione dei messaggi nel formato FlatBuffers;
- MeshEncoder, per l'estrapolazione dei dati geometrici da Blender;
- MeshOperator, libreria contenente le operazioni attuabili sulle Mesh;

Lato Unity, invece, i principali artefici della comunicazione sono due:

- BlenderSocket, incaricata della gestione del client WebSocket;
- FlatMessageBuilder, classe utilizzata per la costruzione dinamica dei messaggi di comando per Blender (Esempio nel codice sottostante).

```
\\Comando per la creazione di una Mesh
BlenderMessageBuilder.NewMessage()
                    .addCommand(Command.Create)
                    .addType(MeshType.Monkey)
                    .build());
```

Si entrerà nel merito del funzionamento di questa integrazione, discutendo dei suoi limiti e vantaggi, nel prossimo capitolo, dedicato alla validazione del prototipo.

Capitolo 7

Validazione Prototipo

È venuto il momento di parlare dei risultati raggiunti con il lavoro svolto, e quindi delle funzionalità e dei limiti del prototipo realizzato. Tutti i test sono stati eseguiti utilizzando una macchina Windows 10 lato server, e dei dispositivi Android 6.0.1 in modalità Cardboard lato client.

Per quanto riguarda le funzionalità base richieste, a partire dalla creazione delle Mesh, alla loro modifica, basata su un meccanismo di riconoscimento di gesti, fino ad arrivare alla loro gestione in modalità condivisa da più utenti, sono stati efficacemente realizzate. Nelle figure seguenti si possono trovare alcune immagini raccolte durante i test che mostrano alcune fasi dell'interazione.

Dal punto di vista della circolazione di dati fra client e server, si è riusciti a gestire delle Mesh di bassa e media complessità grazie agli accorgimenti di cui si è parlato nel capitolo dedicato alla loro codifica. Come già detto, ulteriori miglioramenti sono certamente possibili grazie a delle tecniche di

Figura 7.1: Interfaccia utente

Figura 7.2: Mani virtuali

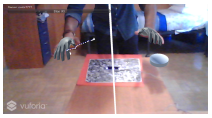

Figura 7.3: Definizione punti

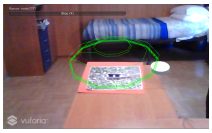

Figura 7.4: Poligono di controllo

Figura 7.5: Creazione Mesh

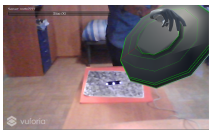

Figura 7.6: Rotazione e traslazione

compressione più avanzate, di cui sono stati dati anche alcuni esempi. Tali accorgimenti si sono rivelati fondamentali anche nella realizzazione dell'integrazione con il software esterno Blender. Purtroppo, però, allo stato attuale di sviluppo non si è raggiunto un livello tale del prototipo da sfruttare appieno questo strumento, né si ritiene che con gli strumenti attualmente utilizzati si possa raggiungere facilmente questo risultato. Infatti Blender permette l'accesso a tecniche di modellazione anche molto avanzate, ma per il cui utilizzo è richiesta una flessibilità e una precisione che attraverso gli strumenti per il riconoscimento di gesti non si è stati in grado di raggiungere. In pratica, il livello di precisione raggiunto nella manipolazione degli oggetti virtuali, si è dimostrato abbastanza buono per quanto riguarda le azioni che coinvolgono l'elemento intero, ma il riconoscimento di gesti da solo, almeno per quanto riguarda le prove fatte, non è stato sufficiente a garantire delle interazioni più articolate.

Il protocollo messo a punto per la comunicazione con Blender, tuttavia,

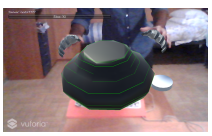

Figura 7.7: Scalatura

Figura 7.8: Traslazione e scalatura

ha dimostrato di poter gestire i dati generati dallo stesso. Sono state svolte delle prove, infatti, in cui lavorando lato desktop su Blender, si utilizzava il prototipo realizzato solamente come componente per il rendering degli oggetti in realtà aumentata, mantenendo i mezzi di interazione tradizionali (mouse e tastiera). In questo modo si è riusciti ad utilizzare tecniche di deformazione e trasformazione della Mesh i cui risultati venivano gestiti correttamente dalla parte Unity e visualizzati in AR dai client.

Per quanto riguarda i client Android è stato riscontrato, nonostante l'intera architettura sia stata pensata per alleggerire il più possibile le loro responsabilità, un utilizzo molto intenso delle risorse computazionali da parte dell'applicativo. Questo è dovuto in larga parte al processo di registrazione video e rendering real-time portato avanti da Vuforia, e che, nonostante i dispositivi utilizzati possiedano un hardware ritenuto più che performante (Qualcomm Snapdragon 801 e Qualcomm Snapdragon 820), porta spesso a delle situazioni di surriscaldamento dei dispositivi. Questo problema può essere limitato diminuendo la qualità di acquisizione video, ma questo inficia enormemente il prototipo in termini di usabilità.

Conclusioni

Scopo dichiarato dell'elaborato è stato lo studio di alcune tecnologie per la costruzione di sistemi in Realtà Aumentata e il loro utilizzo nella costruzione di un prototipo dimostrativo. Proprio dall'AR in sé si è voluto partire, esplorando le sue modalità di funzionamento e chiarendo in questo modo i requisiti necessari alla costruzione del sistema. In base a questi requisiti basilari si sono selezionate le tecnologie necessarie e si è potuto appurare come allo stato attuale della tecnica siano presenti sul mercato degli strumenti che permettono facilmente, e con degli investimenti molto limitati, di poter creare tale tipologia di applicativi. È il caso del framework Vuforia, development kit che ha permesso di gestire senza sforzi le parti più critiche nella creazione di sistemi AR: il tracking dell'utente e il rendering real-time del mondo "aumentato". È anche il caso della periferica Leap, che offre delle ottime capacità sul fronte del riconoscimento di gesti, una delle alternative più affermate nel campo dell'interazione. Sulla base dei punti di forza di questa rivoluzionaria modalità di interfacciamento si è anche proceduto a selezionare un dominio applicativo che si pensava potesse valorizzare al meglio le caratteristiche di questa tecnologia: la modellazione 3D. Si è cercato di tracciare un percorso implementativo chiaro, che si soffermasse soprattutto sui maggiori problemi che si sono dovuti affrontare nell'integrazione delle varie tecnologie in gioco, e che documentasse il processo di creazione del prototipo che è stato presentato nel capitolo procedente, con i suoi limiti e i suoi punti di forza. Non si possono dare giudizi generali su questa tecnologia, non avendo lavorato con strumenti che risultano essere

allo stato dell'arte e avendo solo scalfito la superficie di un sistema molto articolato, che riunisce in sé dei campi dell'informatica molto avanzati, ma il lavoro svolto permette comunque di poter trarre alcune conclusioni circoscritte a questo prototipo. Le tecnologie utilizzate hanno permesso di creare un'esperienza di modellazione veramente immersiva, che, nel piccolo delle funzionalità offerte, rende piacevole il lavoro di creazione e modifica degli elementi virtuali. L'interazione, seppure importantissima nell'immersività dell'esperienza, risulta essere al contempo un punto negativo, non riuscendo, il solo riconoscimento di gesti, a garantire una grande libertà di controllo all'utilizzatore. L'espansione del prototipo con delle tecnologie per il controllo vocale potrebbe essere uno dei punti su cui varrebbe la pena di indagare ulteriormente. Va detto però come il riuscire a toccare con mano ciò su cui si sta lavorando riesca a creare un'esperienza d'uso molto particolare, e che ben riesce a far presagire quali possano essere le grandissime possibilità di utilizzo della Realtà Aumentata in un prossimo futuro.

Bibliografia

[1] Artoolkit. https://artoolkit.org/.

[2] Blender. https://www.blender.org/.

[3] Catchoom craftar. https://catchoom.com/product/craftar/augmented-reality-and-image-recognition/.

[4] Corto. https://github.com/cnr-isti-vclab/corto.

[5] Draco. https://github.com/google/draco.

[6] Flatbuffers. https://google.github.io/flatbuffers/.

[7] Ieee 754. https://www.csee.umbc.edu/ tsimo1/CMSC455/IEEE-754-2008.pdf.

[8] Json. http://www.json.org/.

[9] Leap motion. https://www.leapmotion.com/.

[10] Lz4. http://lz4.github.io/lz4/.

[11] Messagepack. http://msgpack.org/.

[12] Mono. http://www.mono-project.com/.

[13] Openctm. http://openctm.sourceforge.net/.

[14] Protocol buffers. https://developers.google.com/protocol-buffers/.

[15] Snappy. https://google.github.io/snappy/.

[16] Tcp rfc 793. https://tools.ietf.org/html/rfc793.

[17] Unity. https://unity3d.com/.

[18] Vuforia. https://www.vuforia.com/.

[19] Websocket rfc 6455. https://tools.ietf.org/html/rfc6455.

[20] Websocket server for blender. https://github.com/KoltesDigital/websocket-server-for-blender.

[21] D. Amin and S. Govilkar. Comparative study of augmented reality sdk's. In *International Journal on Computational Sciences And Applications Vol.5 No.1*. IJCSA, 2015.

[22] R. T. Azuma. Making augmented reality a reality. 1997.

[23] V. D. Bari and P. Magrassi. In *2015 weekend nel futuro*. Il Sole 24 Ore, 2005.

[24] O. Bimber and R. Raskar. In *Spatial augmented reality: merging real and virtual worlds*. A K Peters, 2005.

[25] T. P. Caudell and D. W. Mizell. Augmented reality: An application of heads-up display technology to manual manufacturing processes. In *Proceedings of the Twenty-Fifth Hawaii International Conference on, vol. 2. IEEE, 1992, pp. 659–669*, 1992.

[26] M. Mekni and A. Lemieux. Augmented reality: Applications, challenges and future trends. 2014.

[27] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. 1999.

[28] I. E. Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I. ACM, 1968, pp. 757–764*, 1968.

[29] D. van Krevelen and R. Poelman. A survey of augmented reality technologies, applications and limitations. In *The International Journal of Virtual Reality 9(2):1-20*, 2010.