ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea in Ingegneria Informatica

TESI DI LAUREA

in

Elettronica T-1

METODI DI INTERAZIONE PER ESPERIENZE IN REALTÀ VIRTUALE A CARATTERE EDUCATIVO

Candidato:

Alessandro Péttazzoni

Relatore:

Chiar.mo Prof.

Bruno Riccò

Anno Accademico 2016/17

Sessione III

Ad Alessandro, Federico e Marco

A Matteo, Mattia

Sommario

La diffusione di realtà aumentata e realtà virtuale è sempre più capillare ma al tempo stesso il livello delle esperienze utente è ancora rudimentale e limita il potenziale enorme di queste tecnologie. Questa tesi presenta una panoramica sullo sviluppo in realtà virtuale utilizzando come caso di studio un'applicazione rivolta al pubblico di un moderno museo che punta sull'interattività per coinvolgere i propri visitatori. Si rende quindi fondamentale realizzare l'interazione secondo i più alti canoni di immediatezza e comfort, sfruttando tecnologie allo stato dell'arte come Oculus Rift e l'editor leader di mercato Unity3D, qua descritti e possibilmente paragonati alle alternative parallelamente alle scelte implementative.

Parole chiave: Realtà virtuale, interazione utente, museo, patrimonio artistico, programmazione

Abstract

Virtual and augmented reality are becoming increasingly widespread while the actual user experiences are still rudimentary and unstandardized, limiting the strong potential of these technologies. This thesis presents an overview of virtual reality development using as a case study an application targeted to the public of a modern interactive museum. It becomes therefore crucial to realize the whole experience according to the highest canons of immediacy and comfort, using state of the art technology such as the Oculus Rift ecosystem and the market leader editor Unity3D, here described and compared, if possibile, along with implementation choices.

Keywords: Virtual reality, user interaction, museum, artistic heritage, programming

Indice

Introduzione **7**

1 Realtà alternative **11**

1.1 Storia 12

1.2 Sfide 14

1.3 Stato dell'arte 17

2 Strumenti **19**

2.1 Rift 19

2.2 Touch 20

2.3 Leap Motion 20

2.4 Unity 21

3 Sviluppo **25**

3.1 Punto di partenza 25

3.1.1 Applicazioni concorrenti 25

3.1.2 Navigate-Color e Digi-Painting 26

3.2 Progettazione 28

3.2.1 Sarcofago degli Sposi 28

3.2.2 Santuario di Portonaccio 29

3.2.3 Cappella Sistina 31

3.2.4 Area Introduttiva 32

3.3 Punti salienti 33

3.3.1 Oculus Touch e Leap Motion 33

3.3.2 Pittura 34

3.3.3 Montaggio 39

4 Conclusioni **43**

4.1 Prove al pubblico 44

4.1.1 Configurazione e procedure 44

4.1.2 Risultati 45

4.1.3 Osservazioni 46

4.2 Sviluppi futuri 47

Bibliografia **47**

Elenco delle figure

1.1 The Sword of Damocles[9] 12

2.1 Il kit Rift completo[12] 20
2.2 Il sensore Leap Motion esploso[4] 21
2.3 L'interfaccia dell'editor Unity 22

3.1 Navigate-Color 26
3.2 Digi-Painting 27
3.3 La scena del Sarcofago degli Sposi 29
3.4 La scena del Santuario di Portonaccio 31
3.5 La scena della Cappella Sistina 32
3.6 La scena dell'area introduttiva 33
3.7 La rappresentazione solida dei valori HSV[11] 36

4.1 Età 44
4.2 Esperienza in VR 44
4.3 Facilità d'uso 45
4.4 Interesse nelle interazioni 45
4.5 Comfort fisico 45
4.6 Qualità grafica 46
4.7 Giudizio globale 46

Ringraziamenti

[[Ringraziamenti qua]]

Introduzione

Il 2017 è stato l'anno dell'effettiva esplosione della realtà virtuale (VR). Dopo la rinascita del mercato a metà del decennio e l'uscita dei primi dispositivi di consumo veri e propri all'inizio del 2016, di cui tratterò più approfonditamente nel primo capitolo dedicato alla realtà virtuale nel complesso, la diffusione di questo medium è divenuta sempre più capillare, interessando quasi ogni campo, dal commercio all'intrattenimento all'industria.

I leader del settore dell'editing 3D, precedentemente utilizzato soprattutto per i videogiochi, non a caso usati come cavallo di troia per la realtà virtuale, come per non poche altre tecnologie in passato, si sono velocemente riconvertiti allo scopo, consentendo anche a singoli come me di realizzare esperienze VR con strumenti accessibili. Questi ultimi saranno descritti in termini chiari nel secondo capitolo, per consentire la comprensione dei temi successivi.

Sfortunatamente però, lo sviluppo tecnologico non è andato di pari passo con quello dell'esperienza utente, con alcuni aspetti, come la locomozione artificiale, ovvero il risolvere lo sfasamento tra il movimento nella realtà virtuale e quello, o la di lui mancanza, nel mondo reale, ancora non totalmente compresi. Oggi capita spesso all'occhio esperto di chi conosce il settore di vedere esperienze immersive, anche legate a marchi o istituzioni di altissimo livello, che sbagliano completamente anche le semplici basi che dovrebbero caratterizzare un'applicazione VR, provocando un immediato malessere nell'utente a causa del profondo impatto sensoriale proprio del medium. Per parlare di esperienze

utente immediate e confortevoli non c'è occasione migliore che un'applicazione pensata specificamente per il grande pubblico, che immaginiamo privo di esperienza a priori con simili dispositivi, almeno ancora per qualche anno, e anche della pazienza per sopportare un prodotto non conforme alle aspettative, mentre per la scelta del tema in pochi possono vantare un patrimonio artistico come quello del nostro Paese. Per questo motivo si è deciso di collaborare con il CINECA[1] per realizzare un progetto che possa avvicinare il pubblico a capolavori unici nel loro genere, sfruttando i più moderni principi dell'*edutainment* e le tecnologie a cui accennato precedentemente. Sarà quindi possibile all'utente entrare in un ambiente virtuale modellato sull'opera di riferimento dove potrà osservare e interagire con modelli a grandezza reale e ricostruzioni degli originali attraverso un'interfaccia diegetica, ovvero mimetizzata e concretizzata negli oggetti di scena, uno dei principi dello sviluppo VR di cui tratterò nel terzo capitolo, dedicato allo sviluppo effettivo della nostra applicazione a partire dai prototipi preesistenti.

Infine, esaminerò nella conclusione i punti di forza, le mancanze e gli sviluppi futuri di quanto creato.

[1]Consorzio Interuniversitario del Nord-Est per il Calcolo Automatico, situato a Casalecchio di Reno

1

Realtà alternative

> *"A display connected to a digital computer gives us a chance to gain familiarity with concepts not realizable in the physical world. It is a looking glass into a mathematical wonderland."*
>
> – Ivan Sutherland

Non è facile definire la realtà virtuale; il concetto di un'esistenza alternativa è antichissimo, legato alla produzione creativa come la letteratura o il teatro e in tempi più recenti è stata proprio la letteratura di fantascienza a precedere la tecnologia, come spesso accade. Stanley G.Weinbaum nel 1935 descrive in "Pygmalion's Spectacles" un visore sotto forma di un paio di lenti che permette non solo di vivere un film con la totalità dei sensi ma di interagirvi: *"You are in the story, you speak to the shadows and they reply, and instead of being on a screen, the story is all about you, and you are in it."*[10]. Col senno di poi diremmo che non vi è più nessuna ragione di chiamarlo film, ma quello che lo scrittore aveva già in mente era esattamente un'applicazione informatica *ante litteram*.

Parallelo a questo concetto si sviluppa quello di realtà aumentata, dove le percezioni virtuali sono sovrapposte a quelle reali per aumentare, appunto, le informazioni disponibili all'utente. Le due tecnologie come intuibile condividono gran parte della base tecnica, si è quindi

diffuso il termine di *Mixed Reality*, dalla traduzione italiana ancora incerta "realtà mista" per intendere il loro complesso. Ritengo pertinente, data la novità del settore, offrire una sintetica panoramica dell'avanzamento della tecnica fino allo stato dell'arte attuale.

1.1 Storia

Tralasciando i cabinati degli anni '50 che offrivano esperienze multi sensoriali puramente passive con immagini stereoscopiche, diffusori di aromi e movimenti meccanici, l'invenzione del primo prototipo di VR è attribuita al professor Ivan Sutherland che, nel 1968 nei laboratori del Massachusetts Institute of Technology, montava con i suoi collaboratori un visore talmente pesante da dover essere ancorato al soffitto, da cui il soprannome goliardico The Sword of Damocles[9]. La tecnologia del tempo permetteva solo la visualizzazione delle semplici linee di contorno degli oggetti, il cosiddetto stile *wireframe*, visualizzate proiettando le immagini di due piccoli tubi catodici in sovrimpressione all'ambiente circostante, indicando già uno stretto legame con la realtà aumentata, almeno per convenienza.

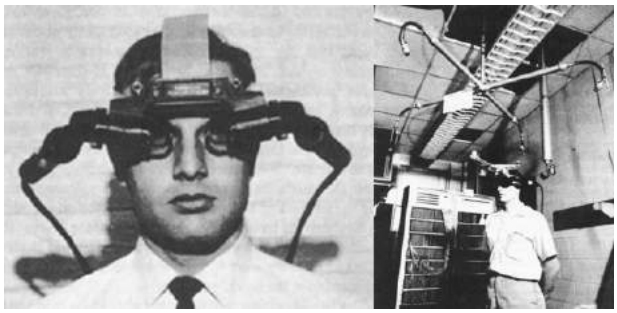

Figura 1.1: The Sword of Damocles[9]

I decenni successivi videro un susseguirsi di visori sul mercato, soprattutto per usi industriali, dato il prezzo inaccessibile per il mercato di massa. Si noti come la parola chiave sia visore, in quanto dopo le prime sperimentazioni per intrattenimento la tecnologia ave-

va velocemente ripiegato sulla simulazione visiva, giudicata probabilmente il senso più materialmente utile nonché facilmente simulabile, ritorneremo sull'argomento nella sezione 1.2.

Gli anni '90 videro invece la prima diffusione nel mercato di consumo della realtà virtuale grazie agli sforzi di Nintendo e Sega, colossi dell'intrattenimento che potevano già contare su una presenza forte nei salotti e nelle sale giochi del tempo dei propri sistemi; nonostante ciò, questi prodotti non furono all'altezza delle aspettative del pubblico dovendo scendere a pesanti compromessi per arrivare ad un prezzo accettabile. Il modello Nintendo, il tristemente famoso Virtual Boy, rinunciava addirittura al tracciamento della testa, mostrando un immagine statica, a bassa risoluzione e monocolore rosso su nero, provocando casi di malessere fisico nei suoi utilizzatori. Non aiutava il fatto che la potenza stessa dell'hardware fosse comunque insufficiente a visualizzare alcunché traesse giovamento da una visualizzazione tridimensionale realistica. Questo fallimento ha ricadute ancor oggi sull'immagine pubblica della realtà virtuale.[6]

La seconda venuta della realtà virtuale di consumo è da imputarsi alla rivoluzione dei dispositivi mobili che seguirono il lancio di iPhone nel 2007. Da un giorno all'altro nasceva un pubblico immenso di clienti per una miniaturizzazione informatica a tappe forzate e prezzi in caduta libera, esattamente quello di cui il nostro settore aveva bisogno per sbocciare. Palmer Luckey ebbe così buon gioco, appena maggiorenne nel garage di casa, ad assemblare con pezzi di ricambio il primo prototipo di quello che sarebbe poi diventato il Rift, visore dalle caratteristiche paragonabili a quelli pensati per usi industriali e icona della realtà virtuale dei nostri tempi.

1.2 Sfide

Dopo il fallimento tutto sommato recente degli anni '90 la realtà virtuale aveva molto da dimostrare. Gusto e olfatto vengono nuovamente scartati, il primo per la sua relativa irrilevanza nelle esperienze quotidiane, il secondo per la difficoltà di riprodurre in maniera convincente e costante nel tempo aromi a partire da pochi componenti base facilmente distribuibili, come accade invece per i colori. E' quindi di fondamentale importanza gestire adeguatamente i restanti sensi, permettendoci di usare vista, udito e tatto per vivere l'esperienza virtuale e al contempo parteciparvi attivamente.

La vista è legata alla fruibilità dell'esperienza nel suo complesso: il cervello umano è abituato a fare affidamento soprattutto su questo senso, con la maggior parte del suo volume dedicatavi, e ogni sbavatura incontra un'immediata reazione di malessere. Nei soggetti più delicati la nausea può perdurare per ore, certamente un effetto che intendiamo evitare, e gli accorgimenti presi in questo senso sono molteplici, di cui mi limiterò ad elencare i più importanti. Innanzitutto il calcolatore deve essere in grado di produrre un'immagine sempre aggiornata e coerente con la posizione spaziale dell'osservatore. Questo significa che l'intera catena dall'input motorio all'output visivo, il cosiddetto *motion-to-photon*, deve occorrere in un tempo stimato inferiore ai 20ms[1].Perché ciò accada è necessario rivoluzionare ogni anello della catena, partendo dall'input: il sistema deve calcolare la posizione dell'utente per fornire un'immagine accurata. Storicamente questo è stato fatto con le cosiddette Unità di Misura Inerziale, composte di accelerometri e giroscopi; si tratta di sistemi collaudati, con latenza ridotta nell'ordine del millisecondo, ma il sistema di calcolo della posizione basato sulla doppia integrazione dell'accelerazione comporta un errore accumulato nel tempo, chiamato deriva, inaccettabile se paragonato alla velocità di rotazione della testa e ai margini

di tolleranza sensoriali. La soluzione è quindi quella di unire questi sensori a una tecnologia più recente, il tracciamento ottico in tempo reale, con tempi di risposta ben maggiori, ma perfetto per correggere la deriva e con il vantaggio non banale di consentire il rilevamento posizionale rispetto all'ambiente circostante. Una volta ottenuta la posizione, il sistema deve renderizzare la visuale dell'osservatore sulla scena, in un numero di volte al secondo e in una risoluzione adeguatamente alti. Per farlo in un tempo accettabile, dobbiamo scendere a pesanti compromessi sulla qualità grafica e degli effetti.

Segue la gestione dei pixel sullo schermo: le tecnologie più diffuse impiegano almeno il tempo di un intero aggiornamento di immagine per illuminare correttamente ogni pixel; per ridurre questo effetto risulta conveniente usare schermi ad aggiornamento più veloce, che tuttavia sono di più difficile reperibilità. C'è inoltre un ultimo componente della latenza percepita, dovuto all'effetto di permanenza dello schermo; a differenza dei tradizionali tubi catodici, la tecnologia a cristalli liquidi lascia accesi tutti i pixel per l'intera durata del fotogramma, pixel che saranno giocoforza ormai non più aggiornati con la posizione corrente. Per risolvere quest'ultimo problema si rendono necessari schermi atipici ideati con la bassa persistenza in mente. La latenza non è l'unica componente del comparto visivo. Come già accennato la risoluzione dovrà essere adeguata: l'occhio umano percepisce circa 120 punti per grado di visione, corrispondenti a circa dieci decimi nel tradizionale sistema di misurazione di acutezza visiva, un livello equivalente alla media o leggermente inferiore. Ipotizzando un angolo di visuale comprendente la visione periferica di 200 per 135 gradi, è chiaro come le tecnologie attuali siano ben lontane dalle capacità umane, e non solo, un campo visivo di quel tipo richiederebbe probabilmente una tecnologia ottica delle lenti al momento inesistente, dovendo già scendere a patti con artefatti e rifrazioni luminose nei modelli attuali. Un aiuto potrebbe venire da tecnologie di tracciamento

oculare per renderizzare solo la parte effettivamente a fuoco della visione; chiaramente questo calcolo introduce ulteriore latenza. Bisogna inoltre tenere conto della struttura stessa degli schermi; per visualizzare un colore infatti ogni pixel è in genere composto di più elementi fisici che accendendosi o spegnendosi danno l'illusione di una moltitudine di colori uniformi. Tecnologie che utilizzano meno *subpixel* per punto daranno un'impressione di minore risoluzione agli ingrandimenti tipici delle lenti VR.

Passando all'udito, mentre l'ingresso viene gestito con un semplice microfono o combinazione di questi, per produrre un audio realistico sono in studio da più anni soluzioni ad hoc basate su algoritmi posizionali e calcoli di rifrazione sonora virtuali, sfruttando le ormai datate, ma mai esplorate troppo a fondo per mancanza di pubblico, tecnologie binaurali, flussi stereo ben distinti per dare la sensazione di tridimensionalità dell'audio, a partire anche dagli studi sulla forma degli organi uditivi e loro risposte in frequenza.

Fondamentale è la riproduzione delle mani, nostro mezzo principale di interazione col mondo che ci circonda. Il sistema deve essere affidabile, veloce e accurato, anche se non necessariamente vincolato agli stringenti criteri per la vista, diventa però più complicata la gestione delle *occlusioni*, dovendo mantenere la linea di visuale per il tracciamento ottico, da cui l'ormai celebre frase di Luckey *"VR input is hard"*. Per quanto riguarda la simulazione del tatto, anche una semplice vibrazione al momento appropriato aumenta l'immersione, mentre l'effettiva simulazione di oggetti crea notevoli interrogativi anche relativi all'intensità accettabile delle suddette interazioni virtuali.

Per ultimi vi sono i limiti fisici dei visori, che devono essere leggeri, bilanciati, ergonomici e possibilmente consentire una connessione wireless al chip di calcolo, se non auto contenuto, per evitare gli impedimenti presentati da uno o più cavi di collegamento. Tutto questo naturalmente offerto a un prezzo competitivo.

Analizziamo ora le caratteristiche tecniche di questa prima generazione di realtà virtuale di consumo.

1.3 Stato dell'arte

Senza entrare troppo nel dettaglio dei singoli visori, i principali attori sul mercato a partire dall'anno 2016 hanno caratteristiche più o meno assimilabili, escludendo l'ambito, numericamente rilevante, dei visori mobili progettati per utilizzare gli schermi preesistenti degli smartphone con risultati prevedibilmente mediocri. Le soluzioni sono basate su un singolo schermo progettato su misura e un sistema di lenti tale da assicurare un campo visivo di circa 100 gradi, sufficiente a coprire la cosiddetta visione periferica vicina e media, con una risoluzione di circa 15 pixel per grado. Sono specifiche ben lontane da quelle ideali, ma considerate il minimo per raggiungere la *presenza*, l'inganno della mente che ci fa sentire fisicamente nella simulazione, congiuntamente ad una frequenza di aggiornamento di almeno 90Hz.

Le soluzioni di tracciamento usano le tecnologie di *sensor fusion* precedentemente introdotte, con sensori ottici appaiati ad emettitori luminosi o, negli ultimi mesi, anche con soluzioni basate sul riconoscimento dell'ambiente circostante per localizzare l'utente.Tutti i visori sono venduti insieme a *controller* costruiti su misura, dovendo sposare i meccanismi di tracciamento ad un'ergonomia e un'immediatezza dei comandi necessariamente maggiore di quella dei tradizionali equivalenti da videogioco dei decenni passati. Questi visori non contengono chip di rendering al loro interno, dovendo quindi obbligatoriamente rimanere collegati a unità di calcolo da centinaia di watt di potenza, difficili da rendere portatili. Si sono quindi diffuse soluzioni commercializzate da terze parti per la trasmissione WiFi basate su tecnologie innovative a frequenze nell'ordine delle decine di giga hertz.

I produttori forniscono agli sviluppatori librerie software, o kit di

sviluppo (SDK), che coprono ogni aspetto di basso livello, consentendo loro di focalizzare il proprio lavoro sull'esperienza utente e sui contenuti. Fondamentali si sono resi gli studi sulla *riproiezione*, insieme di algoritmi molto complessi ma dal calcolo relativamente leggero con cui il calcolatore è in grado di riprocessare l'immagine renderizzata distorcendola secondo gli ultimi dati ricevuti dal sistema di tracciamento, con il risultato di una sorta di interpolazione intelligente per migliorare la frequenza di aggiornamento con un minimo impatto prestazionale.

Veniamo ora ad analizzare gli strumenti specifici da noi utilizzati.

https://www.youtube.com/watch?v=f3vKCbi4UAw

2

Strumenti

L'applicazione oggetto di questa tesi è stata sviluppata per il visore Rift commercializzato da Oculus, azienda del gruppo Facebook. Le librerie di sviluppo sono fornite da Oculus stessa mentre la creazione tridimensionale delle scene è realizzata grazie all'editor Unity3D. Il beneficio di programmare attraverso lo strumento Unity è che, oltre a non dovermi occupare di tutte le librerie grafiche di basso livello, la scena così prodotta è nella sua quasi totalità scollegata da ogni dipendenza e potrà quindi in futuro essere velocemente riadattata e riproposta quando giocoforza la tecnologia disponibile andrà a mutare.

2.1 Rift

Rift è un visore rilasciato nel marzo 2016, costruito su un singolo schermo OLED[1] a bassa persistenza dalla risoluzione di 2160x1200 pixel e frequenza di aggiornamento di 90Hz. Il sistema di tracciamento ottico chiamato Constellation è basato su led infrarossi distribuiti su casco e *controller* e una serie di telecamere per la triangolazione posizionale. Una telecamera sarebbe da sola in grado di triangolare la posizione sfruttando i numerosi led, ma per una tipica installazione sono consigliati almeno due o tre sensori per minimizzare il rischio di occlusione,

[1] *Organic Light-Emitting Diode*

ovvero la mancanza di visibilità dell'oggetto tracciato, coperto da un altro. Per quanto riguarda l'audio Rift è invece equipaggiato con due cuffie supra-aurali ad alta fedeltà integrate nel casco.[12]

2.2 Touch

I *controller* ufficiali Oculus, venduti prima singolarmente da dicembre 2016 col nome di Touch e ora insieme ad ogni visore, sono stati sviluppati in collaborazione e poi con la diretta acquisizione dell'azienda di design Carbon, precedentemente al lavoro sulla console Xbox 360, e hanno incontrato un grande favore nel pubblico, sposando metodi di controllo tradizionali e innovativi a un'ergonomia perfetta. Sfruttando lo stesso sistema Constellation, sono tracciati completamente nell'ambiente in maniera indipendente dal casco, a cui si appoggiano per la trasmissione dati per evitare ulteriori cavi.[12]

Figura 2.1: Il kit Rift completo[12]

2.3 Leap Motion

Il Leap Motion è un piccolo sensore formato da telecamere ed emettitori a infrarossi, tecnologie ormai familiari al grande pubblico e agli sviluppatori software dopo il successo di Kinect e iPhone X. Sviluppato nei primi anni 2010 trovò rapidamente la sua principale applicazione come metodo di input VR, approfittando anche della mancanza di concorrenza nel settore che utilizzava ancora normali *controller* da console

di videogiochi. La sua corta portata e quindi alta risoluzione infatti sono state calcolate con l'obiettivo del tracciamento delle mani, grazie a una ricostruzione tridimensionale in tempo reale dalle immagini stereo delle telecamere con una precisione superiore al millimetro.[4] Come per i Touch, l'azienda produttrice provvede agli sviluppatori

Figura 2.2: Il sensore Leap Motion esploso[4]

con un SDK che gestisce autonomamente l'hardware e la riproduzione delle dita e offre anche basilari metodi di interazione, come ad esempio la possibilità di afferrare un oggetto chiudendo la mano.

2.4 Unity

Unity è uno degli editor 3D più utilizzati del mondo, ovvero un programma dedicato alla realizzazione di contenuto tridimensionale più o meno interattivo, come videogiochi o animazioni. Presentato alla WWDC[2] del 2005 con supporto per la compilazione solo per MacOS, è arrivato rapidamente a supportare quasi ogni piattaforma disponibile sul mercato, dai computer alle console agli smartphone, vero punto di svolta per Unity, primo motore grafico a compilare nativamente per iPhone alla fine del 2008 assicurandosi popolarità.[3] L'editor consente la programmazione in linguaggio C#, incoraggiando un'organizzazione

[2] *Worldwide Developers Conference*, la conferenza annuale per sviluppatori Apple

fortemente orientata agli oggetti[3]. Ogni script che scriviamo estende la classe[4] fondamentale MonoBehaviour, che gli consente di invocare le funzioni fondamentali di Unity e di ricevere gli *eventi*, chiamate di funzione legate ad avvenimenti rilevanti nel programma, spesso derivanti dall'interazione utente, ad esempio, gli eventi fondamentali Start e Update, gestiti rispettivamente al lancio della scena corrente di simulazione e una volta ogni fotogramma calcolato, già espressi in ogni nuovo script per comodità.

Ogni istanza di oggetto di scena è in esecuzione un oggetto C# chiamato *GameObject*; attraverso una funzione possiamo avere accesso ad ogni suo componente di scena, tra cui eventuali script a lui collegati, i *Rigidbody* che gestiscono la fisica, i *Collider* per la collisione con altri oggetti, i *Transform* con le informazioni sulla posizione e i *Renderer* che si occupano appunto di renderizzare luci e materiali. Si possono osservare nella figura 2.3 a sinistra la gerarchia degli oggetti di scena, in basso il gestore di file del progetto e a destra i componenti dell'oggetto selezionato. Come già accennato Unity prevede già nativamente il

Figura 2.3: L'interfaccia dell'editor Unity

supporto per i visori di realtà virtuale e aumentata, successivamente espanso da *package* resi disponibili dalle varie aziende, contenenti oltre agli SDK necessari al funzionamento delle applicazioni anche degli

[3]Paradigma di programmazione che raggruppa strutture dati e funzioni, istruzioni per compiere determinate azioni

[4]Codice che costituisce una struttura sulla base del quale sono istanziati gli oggetti, generati anche in molteplici copie

oggetti di scena preimpostati e delle scene di esempio per capirne il funzionamento. L'oggetto di scena legato al visore Rift, per esempio, contiene una serie di oggetti figli con i componenti che si occupano della telecamera stereoscopica, ovvero della visuale di renderizzazione sulla scena relativa a ciascuno degli occhi, dei modelli delle mani, e dei *collider* che gestiscono le interazioni con gli oggetti.

3

Sviluppo

3.1 Punto di partenza

Esaminiamo brevemente le condizioni di partenza della mia applicazione, ovvero il progetto antecedente sul quale si basa e alcuni similari che compongono lo stato dell'arte applicativo al momento della scrittura.

3.1.1 Applicazioni concorrenti

Come già descritto approfonditamente, il 2016 è stato definito l'anno della realtà virtuale e i musei sono stati veloci ad approfittare di questa nuova tecnologia.

In un primo momento si è dato ai visitatori un luogo dove provare la realtà virtuale con esperienze già disponibili, come il Natural History Museum of Los Angeles County, che ha approntato una postazione con l'applicazione theBlu[7], una simulazione sottomarina, quindi già vagamente legata alla natura del museo. E' chiaro come questo *modus operandi* sia valido soltanto in un primo tempo, poiché tali esperienze puntano tutto sulla novità della tecnologia e non offrono contenuti realmente educativi o particolarmente legati al luogo in questione.

Successivamente, mediante collaborazioni si è iniziato a creare applicazioni su misura, la stragrande maggioranza delle quali composte

da foto 360 o modelli tridimensionali di stanze dei musei o delle loro opere. Un importante esempio è quello del British Museum[2], creato in collaborazione con Oculus, che impiega vere foto ma modellate secondo fotogrammetria per consentire il massimo realismo in VR, conservando le distanze percepite. Un altro esempio è quanto sviluppato dal Museo dell'Ara Pacis a Roma, che sfrutta il visore GearVR unito a un Samsung S7 per realizzare un'esperienza in realtà aumentata dell'altare coi colori originali.[5] A differenza delle precedenti, queste applicazioni offrono contenuti interessanti per i visitatori, reali o virtuali, ma non sfruttano a fondo la tecnologia della realtà virtuale, che come sappiamo non è solo passiva ma anche attiva. Il nostro progetto vuole andare oltre, ed offrire una vera interazione all'utente per catturarne l'attenzione e lasciare un ricordo indelebile dell'esperienza.

3.1.2 Navigate-Color e Digi-Painting

Nella nostra Università questo progetto parte nel 2015 con l'applicazione per sistemi iOS Navigate-Color che, sempre in collaborazione col CINECA[8], permette all'utente di visualizzare tridimensionalmente il Sarcofago degli Sposi e colorarlo applicando alle sue parti dei colori predefiniti usando una semplice interfaccia tattile.

Figura 3.1: Navigate-Color

Il programma è funzionale, intuitivo, espandibile a piacere con altre opere e conta su una base installata di dispositivi iOS enorme. Tuttavia l'interazione è basica, ormai datata e difficilmente potrà cat-

turare l'attenzione del pubblico. Si passa così ad una sperimentazione ibrida sfruttando il visore OSVR e il sistema di controllo Leap Motion. L'applicazione risultante, chiamata Digi-Painting, prevede due fasi, alternate con un battito di mani: nella prima si indica la zona da colorare, nella seconda si fanno apparire i colori rivolgendo il palmo della mano verso l'alto e si trascinano sulla zona in questione con la tipica *gesture pinch*. In una tesi successiva nel 2017 questo sistema è stato semplificato prevedendo dei pulsanti per la transizione da una fase all'altra, un menù introduttivo e un elemento di aiuto per spiegare all'utente i comandi. Appare palese al lettore, anche col confronto delle

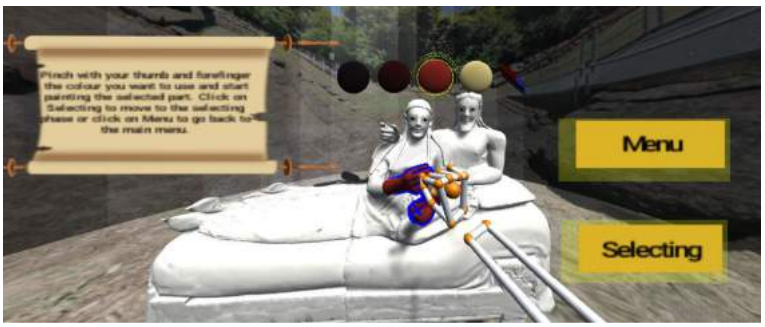

Figura 3.2: Digi-Painting

immagini, come questa interfaccia nasca da una semplice trasposizione tridimensionale di quella originaria dell'app per smartphone, con un evidente risparmio di tempo ma pesanti conseguenze sull'esperienza finale. I pulsanti flottanti rompono l'immersione nella realtà virtuale, non per la mancanza di realismo di un oggetto volante quanto per la realizzazione bidimensionale in un mondo tridimensionale. Inoltre, gli stessi comandi applicati, come il suddetto *pinch*, utilizzati a distanza indicando un determinato oggetto, sono parte fondante dell'interazione iOS ma non certo di quella della realtà virtuale, risultando faticosi, imprecisi e per nulla intuitivi.

La mia versione si occuperà quindi di risolvere questi problemi, trasportando correttamente l'idea originaria dell'applicazione nella sua manifestazione VR.

3.2 Progettazione

La mia applicazione si divide in *scene*, secondo la nomenclatura di Unity. In un primo momento vi era un'unica scena, quella del Sarcofago degli Sposi, corrispondente approssimativamente a quanto contenuto nella versione precedente. Successivamente mi è stato proposto dal CINECA un altro modello che ho ritenuto fondamentale per dare un'idea delle capacità del software di essere espanso e applicato a qualsiasi opera. Avendo finito anche questa scena in anticipo sui tempi ho deciso di implementarne una terza per proporre un terzo modello di interazione e variare un po' il ventaglio dei tipi di opera ed epoche storiche, sfruttando il modello della Cappella Sistina di un grafico polacco. Si è quindi reso successivamente necessario progettare una scena centrale per permettere l'accesso alle altre.

3.2.1 Sarcofago degli Sposi

La rivisitazione di Digi-Painting parte dall'interfaccia, di cui vedremo il codice nella sezione 3.3.2. Il progetto prevede un'interfaccia completamente diegetica, composta da un pennello e una tavolozza per renderla il più naturale possibile all'utente, salvo il fatto che questi oggetti possono essere lasciati ovunque senza quell'effetto di gravità che ci obbligherebbe a recuperarli da terra piegandoci a cercarli. La tavolozza presenta una serie di colori preimpostati, esattamente come nelle applicazioni precedenti, corrispondenti a colori ritenuti validi nel contesto artistico del tempo, per mantenere il massimo realismo. Il pennello campiona questi colori al contatto, nel naturale gesto dell'intingere la punta nella tavolozza, per poi applicarlo direttamente sulle zone in cui è preventivamente stato diviso il modello del sarcofago per facilitare l'operazione, ad esempio capelli, abiti, carnagione.

Per quanto riguarda l'ambientazione invece, al solido ottagonale che fungeva da sfondo per Digi-Painting è stato sostituito l'intero mo-

dello tridimensionale della Tomba dei Rilievi, realizzato sempre dal CINECA con la tecnologia laser, e illuminato da qualche sorgente di luce puntiforme collocata in modo da creare un'atmosfera suggestiva. Intorno all'utente sono anche disposte alcune indicazioni visuali sotto forma di scritti alle pareti, sempre nell'ottica di un'interfaccia trasparente e immersiva, per illustrare la posizione dell'unico tasto sul *controller* Touch responsabile della presa sugli oggetti e dare alcune informazioni sull'opera: una foto dell'originale, il periodo storico di appartenenza e qualche curiosità sul soggetto. Anche la luce contribuisce all'interfaccia: grazie ad uno script all'inizio dell'esperienza un cono di luce si stringe su tavolozza e pennello per poi illuminare completamente la stanza una volta raccolti; accorgimenti di questo tipo guidano l'utente senza compromettere la presenza. Completano l'esperienza due effetti distinti per il cambio di colore e la pittura, il secondo di una certa durata per sottolineare l'effetto continuato della permanenza del pennello sulla superficie del sarcofago, altrimenti non sufficientemente immediato, e una traccia monotona di un generatore di corrente o aerazione per confermare la sensazione di trovarsi in profondità nel terreno.

Figura 3.3: La scena del Sarcofago degli Sposi

3.2.2 Santuario di Portonaccio

Una volta visto il modello del Santuario, ho immediatamente pensato al senso di scala, una delle componenti vincenti della realtà virtuale.

Non è facile immaginarlo senza averlo provato ma, una volta dentro la simulazione, gli oggetti appaiono come reali nella loro imponenza, in maniera completamente diversa dalla stessa esatta applicazione vista su uno schermo regolare. Per questo ho progettato di includere il modello nelle sue dimensioni reali, e stupire l'utente con questa interazione unica nel suo genere. Diveniva però difficile concepire un'interazione con un oggetto tanto grande, da qui l'idea di sdoppiare il tempio, includendone anche una rappresentazione in scala. Si potrà quindi montare il modellino, grazie a una sagoma trasparente che indica le posizioni finali dei pezzi, e osservare il tempio reale comporsi in completa sincronia. Anche questa interazione è progettata per essere il più accessibile e semplice possibile, con un'interfaccia ridotta alle sole mani dell'utente: i *controller* tracciano il movimento e il pulsante che registra lo stringersi delle mani è responsabile della presa sugli oggetti che vengono poi spostati. Gli stessi oggetti non facenti parte dell'interfaccia sono stati dotati di fisica per rendere l'interazione più piacevole e realistica. Uno script apposito è stato deputato al contenimento di tali oggetti nell'area di interazione data l'imprevedibilità dell'interazione con la fisica: se fossero lanciati al di fuori di propria volontà o per effetto di collisioni impreviste prevede ad azzerarne la velocità e ripristinarne la posizione originaria.

Per quanto riguarda l'ambientazione ho ripreso l'immagine 360 contenuta nell'applicazione Digi-Painting, che rappresentava proprio il sito del tempio dell'antica Veio. Invece di applicarla ad un solido con la resa mediocre che abbiamo constatato ho provato ad applicarla direttamente allo *skybox*[1], con risultati non troppo entusiasmanti: l'immagine 360 grezza mostra una distanza virtualmente infinita da ogni lato, compreso quello del suolo, dando la sensazione nauseante di essere sospesi nel vuoto. Per risolverlo ho quindi abbozzato un rudi-

[1]La sfera a distanza infinita che usa Unity proprio per permettere l'applicazione di sfondi

mentale modello di suolo fangoso e rovine di pietra per nascondere la parte più bassa del campo visivo, lasciando quindi solo quello dove la distanza infinita è realistica, con un'ottima resa. E' stata posizionata una sola sorgente di luce in modo congruo alla posizione del sole nella foto, più quella diffusa della *skybox* stessa per gestire il riverbero. Ho inoltre provveduto a nascondere dalla foto quanto resta del tempio originario, visto che quello spazio è già occupato dalla ricostruzione. Il comparto audio è formato da una registrazione molto tranquilla di un'ambiente boschivo e un leggero effetto che si attiva al piazzamento di un oggetto nella sua posizione corretta. Anche qua sono presenti indicazioni alle pareti in aspetto e contenuti del tutto simili a quelle dell'altra scena, per garantire la coerenza delle interfacce e garantirne la riconoscibilità pur rimanendo perfettamente integrate nell'ambiente circostante.

Figura 3.4: La scena del Santuario di Portonaccio

3.2.3 Cappella Sistina

Il fatto che l'opera d'arte qui rappresentata componga l'interezza della scena ci porta a progettare un'interazione che spinga l'utente a trascorrere del tempo al suo interno e lo induca a guardarsi intorno. Nasce così l'idea di un grande puzzle basato sul Giudizio Universale, per completare il quale ci si può aiutare con l'affresco che ci si trova davanti. L'interazione recupera per intero i metodi e anche gli script già utilizzati per la costruzione del tempio, con l'effetto di semplificare

contemporaneamente l'interfaccia presentata all'utente, dai comandi ai cartelli, e il codice su cui si basa. L'oggetto di riferimento per il posizionamento questa volta sarà chiaramente invisibile per non rovinare il senso del puzzle.

Anche la parte grafica è notevolmente semplificata, trattando solo l'illuminazione, prodotta da una serie di sorgenti puntiformi per cercare di rendere l'effetto naturale di luce proveniente dalle finestre, ma al contempo cercando di mantenere visibile la maggior parte degli affreschi. Come sottofondo ho invece scelto una registrazione molto silenziosa della Cattedrale di Colonia che mi sembrava adatta a questa visita privata negli ambienti vaticani.

Figura 3.5: La scena della Cappella Sistina

3.2.4 Area Introduttiva

L'area introduttiva si è resa necessaria per consentire all'utente di scegliere e accedere alle diverse scene presenti nel programma, ma è stata sfruttata anche per semplificare altri aspetti. Per esempio, i modellini in scala collocati dentro alcune teche disposte in perfetta simmetria intorno al centro dell'area intrigano già il fruitore dell'esperienza e la introducono semplicemente con una forma e una scritta. Inoltre, la posizione naturale che si assumerà per premere il bottone che si occupa del lancio effettivo della scena sarà sfruttata per la collocazione in quella stessa, evitando tutti i possibili problemi derivanti da una po-

sizione iniziale supposta casuale, ad esempio la compenetrazione negli oggetti di scena o la mancata visibilità di altri.

Ho disegnato questa scena perché fosse quanto più possibile astratta e intangibile, una realtà virtuale nella realtà virtuale, per non rubare la scena alle opere protagoniste dell'esperienza, optando quindi per un nero avvolgente, rotto solo dalle teche e da un cerchio luminoso che guida l'utente verso il centro dell'area. Completano la scena una luce soffusa, anch'essa radiante dal centro, e una sonda per i riflessi, che come da nome consiste in una telecamera fittizia che renderizza la scena per creare riflessi su appositi materiali, dando una piacevole sensazione di realismo ai vetri delle teche. Nessun suono è udibile se non l'effetto audio del click dei bottoni durante i cambi di scena.

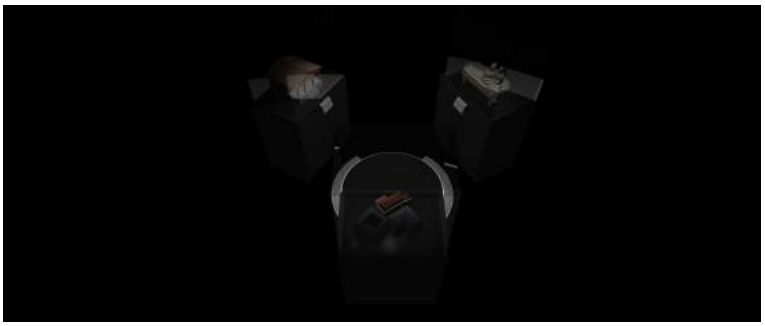

Figura 3.6: La scena dell'area introduttiva

3.3 Punti salienti

3.3.1 Oculus Touch e Leap Motion

Il programma Digi-Painting si basava unicamente sulla periferica Leap Motion quindi logicamente questa è stata il punto di riferimento anche per la prima versione di questo progetto. Dalle prove pratiche però, questa tecnologia si è rivelata limitante in un vero ambiente di realtà virtuale a 360 gradi, essendo nata per controllare una superficie come uno schermo, pensate a quelli di *Minority Report*, più o meno quello

che accadeva in Digi-Painting. Ad esempio, non potendo contare su nessuna rilevazione al di fuori del campo visivo del sensore, qualsiasi oggetto tenuto in mano viene irrimediabilmente ad essere lasciato una volta che lo spostiamo appena al di fuori. Anche la precisione del sensore non ci è sembrata adeguata, facendo fatica a distinguere la posizione rotazionale delle mani una volta che le dita si fossero chiuse, e la mancanza di risposta fisica, dovuta alle mani che si chiudono sul niente, si è rivelata estraniante per gli utenti. Abbiamo quindi giudicato sfavorevole il compromesso fra l'immediatezza del controllo con le sole mani e tutti questi svantaggi, conservando il supporto ad entrambi nel programma ma scegliendo di mostrarlo con il solo controllo con Touch.

3.3.2 Pittura

La funzione delegata alla pittura ha visto una serie di iterazioni durante lo sviluppo del progetto. La prima versione reagiva alla collisione, ovvero l'evento lanciato dalla compenetrazione di un oggetto, in questo caso il pennello, con un altro, l'oggetto da colorare, sostituendo immediatamente al colore di questo quello selezionato precedentemente dalla tavolozza, in maniera analoga a Navigate-Color e Digi-Painting. A seguito di alcune prove pratiche è stata giudicata funzionale, ma non abbastanza realistica per il livello simulativo raggiunto dal resto dell'esperienza, ed è stato implementato un sistema che, ad ogni collisione, produceva gradazioni sempre più vicine al colore scelto, così da dare una sensazione di progressione. Ulteriori prove pratiche l'hanno valutato poco intuitivo, dato che alcuni utenti tenevano il pennello dentro l'oggetto e si domandavano perché non vi fosse alcun cambiamento. Si è giunti così al terzo adattamento che sfuma il colore dell'oggetto in maniera continua nel tempo finché si protrae la permanenza del pennello al suo interno. Esaminiamo la classe[2] nel dettaglio per poter

[2]Vedi sezione 2.4

procedere poi con più scioltezza con le successive.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Brush : MonoBehaviour
6 {
```

Dopo gli import[3] canonici definiamo la classe come estensione di MonoBehaviour, la base definita da Unity che, oltre a consentirci di gestire lo script direttamente dall'editor, implementa gli eventi[4] fondamentali.

```
7     public float speed = 0.05f;
8
9     private Material brushColor;
10    private float h, s, v;
11    private float sMax, vMin;
12
13    private GameObject target;
14    private Material[] mats;
```

La velocità con cui le superfici assumono il colore è una variabile che dovrà essere provata dal vivo più e più volte prima di essere definita con certezza, e probabilmente cambiata in futuro, quindi scegliamo di definirla con una variabile pubblica liberamente modificabile da editor. Seguono le variabili private: le prime tre dedicate al colore del pennello, immagazzinato come un Material[5] di Unity e a quello corrente dell'oggetto dipinto, salvato per comodità sotto forma di tre float[6] secondo la convenzione HSV, liberamente tradotta con Tonalità, Saturazione e Luminosità: il primo valore si occupa della tonalità di colore, il secondo della sua intensità e il terzo definisce una scala dal nero assoluto al colore effettivo. Sono espressi da Unity con un valore da 0 a 1. Questo sistema consente una rappresentazione molto naturale e umana, rendendo immediate le modifiche che mi interes-

[3] Riferimenti a codice di librerie esterne che utilizziamo implicitamente

[4] Vedi sezione 2.4

[5] Insieme delle proprietà che definiscono la renderizzazione di una superficie, include texture, tassellazione, tinte

[6] Numeri in virgola mobile

sano. Infine salviamo in due variabili l'oggetto corrente che stiamo colorando, ovvero il suo riferimento interno a Unity e i suoi Material, visto che un oggetto può averne più d'uno, che andremo a modificare e successivamente sostituire ai correnti.

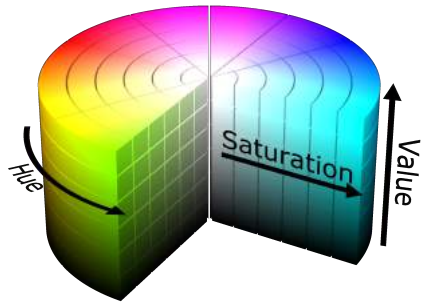

Figura 3.7: La rappresentazione solida dei valori HSV[11]

```
16    void Start()
17     {
18         brushColor = this.gameObject.GetComponent<Renderer>().material;
19     }
```

Inizializziamo la variabile con il materiale corrente del pennello, settato in editor al bianco ma facilmente modificabile all'occorrenza

```
21     void Update()
22     {
23         if (target != null)
24         {
25             if (brushColor.name.Substring(0, 4) == "BIAN")
26             {
27                 s -= Time.deltaTime * speed;
28                 v += Time.deltaTime * speed;
29             }
30
31             else
32             {
33                 s += Time.deltaTime * speed;
34                 v -= Time.deltaTime * speed;
35             }
36
37             if (s > sMax)
38                 s = sMax;
39             if (s < 0)
40                 s = 0;
41             if (v < vMin)
```

```
42                v = vMin;
43            if (v > 1)
44                v = 1;
45
46            foreach (Material m in mats)
47                m.color = UnityEngine.Color.HSVToRGB(h, s, v);
48
49            target.GetComponent<Renderer>().materials = mats;
50        }
51    }
```

Nella funzione che gestisce l'evento Update, chiamata automaticamente da Unity una volta per ogni frame renderizzato solo e soltanto se si ha un oggetto corrente da dipingere, lo si colora gradualmente dal bianco, aumentando la saturazione e diminuendo la luminosità, in proporzione al tempo passato dal calcolo dell'ultimo frame, deltaTime, che non è mai costante, giungendo ai valori limite sMax e vMin, che sono quelli propri del colore scelto, si consulti la figura 3.7 per chiarezza. Infine, si riconverte il colore in RGB, la tradizionale scala parametrizzata attraverso i tre colori principali, e si applicano i materiali così ottenuti all'oggetto corrente.

```
53   void OnTriggerEnter(Collider other)
54    {
55        if (other.gameObject.CompareTag("Paint"))
56        {
57            float temp;
58            brushColor = new Material(other.gameObject.GetComponent<Renderer>().
              ↪material);
59            this.gameObject.GetComponent<Renderer>().material = brushColor;
60            if (other.gameObject.GetComponent<Renderer>().material.name.
              ↪Substring(0, 4) != "BIAN")
61                UnityEngine.Color.RGBToHSV(other.gameObject.GetComponent<Renderer>(
                  ↪).material.color, out temp, out sMax, out vMin);
62        }
```

Gestiamo ora le collisioni con gli altri oggetti di scena. Se questi ultimi sono tra le vernici della tavolozza, marchiate con il *tag* Paint, si crea una copia di quel materiale e la si assegna sia al pennello vero e proprio che alla variabile brushColor per accedervi più velocemente. A meno che il materiale non sia la pittura bianca, che come abbiamo visto

precedentemente fa regola a sé, fungendo come da gomma sugli altri colori, si impostano anche i valori di sMax e vMin.

```
64          else if (other.gameObject.CompareTag("Canvas"))
65          {
66              if (brushColor != null)
67              {
68                  if (other.gameObject.GetComponent<Renderer>().material.name.
                ↪Substring(0, 4) == brushColor.name.Substring(0, 4))
69                      UnityEngine.Color.RGBToHSV(other.gameObject.
                ↪GetComponent<Renderer>().material.color, out h, out s, out v);
70                  else if (brushColor.name.Substring(0, 4) == "BIAN")
71                      UnityEngine.Color.RGBToHSV(other.gameObject.
                ↪GetComponent<Renderer>().material.color, out h, out s, out v);
72                  else
73                  {
74                      UnityEngine.Color.RGBToHSV(brushColor.color, out h, out s, out
                ↪v);
75                      mats = other.gameObject.GetComponent<Renderer>().materials;
76                      s = 0;
77                      v = 1;
78                      for (int i = 0; i < other.gameObject.GetComponent<Renderer>().
                ↪materials.Length; i++)
79                      {
80                          mats[i] = new Material(brushColor);
81                          mats[i].color = UnityEngine.Color.HSVToRGB(h, s, v);
82
83                      }
84                      other.gameObject.GetComponent<Renderer>().materials = mats;
85                  }
86                  mats = other.gameObject.GetComponent<Renderer>().materials;
87                  target = other.gameObject;
88              }
89          }
90      }
```

Quando invece la collisione è con un oggetto marcato come colorabile dal tag Canvas, in caso il colore applicatovi sia già quello immagazzinato nel pennello, oppure quest'ultimo contenga il bianco, ci limitiamo a salvare il colore corrente dell'oggetto nei campi h, s e v e l'oggetto e i suoi materiali nei campi mats e target visti precedentemente. Se invece sull'oggetto è stato applicato un altro colore lo ripristiniamo al bianco originale ponendo s a 0 e v a 1 ma soprattutto sovrascriviamo i

suoi materiali con delle copie di quello corrente per assicurarci la continuazione della colorazione in una prossima collisione se dovessimo smettere prima di raggiungere la saturazione massima.

```
92      private void OnTriggerExit(Collider other)
93      {
94          target = null;
95      }
96  }
```

Per ultimo, vediamo il gestore di fine collisione che semplicemente elimina il riferimento all'oggetto corrente per interrompere il ciclo di colorazione in atto.

3.3.3 Montaggio

Sia il livello del santuario che quello della Cappella Sistina usano una famiglia di script comuni, con l'obiettivo di consentire all'utente di spostare a piacere i pezzi e bloccarli sul posto una volta raggiunta la loro collocazione. Tralascio l'analisi del codice che visualizza le mani e gestisce la sincronizzazione dei movimenti di esse e dell'oggetto afferrato in quanto parte dell'SDK Oculus e mi focalizzo su ciò che è da me prodotto.

```
5  public class Builder : MonoBehaviour {
6      public GameObject[] targets;
7      public float accuracy = 0.002f;
8      public AudioClip clip;
9      public GameObject manager;
10
11     void Start () {
12         AudioSource source = this.gameObject.AddComponent<AudioSource>();
13         source.clip = clip;
14     }
```

L'inizializzazione, intesa sia come quella dell'oggetto C# che quella dell'oggetto di scena in Start(), vede la creazione di variabili dedicate ad esprimere rispettivamente: la posizione finale del pezzo di costruzione, o più d'uno per gestire il caso in cui ci siano più pezzi identici, la precisione richiesta nel posizionare il prezzo prima che si attivi il

blocco nella posizione esatta prevista, il suono eventuale da riprodurre quando questo accade e, sempre facoltativo, un oggetto manager che gestisca il completamento del montaggio. Viene quindi creato un componente AudioSource che si occupa della riproduzione del suono scelto.

```
16    void Update () {
17         foreach (GameObject target in targets)
18         {
19          if (Vector3.SqrMagnitude(this.gameObject.transform.position - target.
                ↪transform.position) < accuracy)
20          {
21             this.gameObject.transform.position = target.transform.position;
22             this.gameObject.transform.rotation = target.transform.rotation;
23             this.gameObject.GetComponent<AudioSource>().Play();
24             if (manager != null)
25             {
26                manager.GetComponent<BuilderManager>().Built();
27                 }
28             this.gameObject.GetComponent<Rigidbody>().constraints =
                ↪RigidbodyConstraints.FreezeAll;
29             Destroy(this.gameObject.GetComponent<OVRGrabbable>());
30             Destroy(target);
31             Destroy(this);
32             }
33         }
34    }
35 }
```

Ad ogni aggiornamento di frame, andiamo a calcolare la distanza, per esattezza la distanza quadrata, per risparmiare la dispendiosa operazione di radice, con la posizione finale più vicina. In caso questa fosse minore della soglia precedentemente definita, attiviamo la procedura di bloccaggio impostando posizione e rotazione finali pari all'oggetto di riferimento. Se presenti, riproduciamo il suono e avvisiamo il gestore dell'avvenuto piazzamento. Infine, impediamo ulteriori spostamenti del pezzo impostando tutti i vincoli posizionali, rimuovendo lo script dell'SDK Oculus che gli permetteva di essere afferrato e distruggiamo l'oggetto di comodo non più necessario e questo script stesso per evitare ulteriori calcoli inutili.

```
public class BigScale : MonoBehaviour {
    public GameObject equivalent;

   void Update () {
      this.gameObject.transform.localPosition = equivalent.transform.localPosition;
   }
}
```

Questo semplice script gestisce il santuario a scala naturale sfruttando la proprietà localPosition che rappresenta la posizione con una terna cartesiana locale, ignara della scala e identica, quindi, per entrambi i templi. Basterà perciò uguagliarle per far muovere i pezzi in sincronia.

```
public interface BuilderManager
{
    void Built();
}
```

Definiamo ora un'interfaccia, una parte comune di codice tra più script, che consente ad altri di operare con uno qualsiasi di loro come se fossero uguali, nascondendo le parti dell'implementazione specifica. In questo caso, Builder, come abbiamo già visto, saprà che l'oggetto assegnatogli contiene un BuilderManager e che quest'ultimo implementa un metodo chiamato Built() per segnalare il posizionamento di un pezzo.

```
public class Sanctuary : MonoBehaviour, BuilderManager
{
   private int pieces = 8;
   private int placed = 0;

   public void Built()
   {
      placed++;
   }

   void Update()
    {
      if (placed == pieces)
      {
         this.gameObject.GetComponent<AudioSource>().Play();
         Destroy(this);
      }
```

```
27    }
28 }
```

L'effettiva implementazione Sanctuary utilizzata nella seconda esperienza, invece, tiene traccia di quanti pezzi sono già stati piazzati e, quando questo conteggio raggiunge il totale dei pezzi, precedentemente immesso, riproduce un suono per dare all'utente un senso di realizzazione e informarlo che ha collocato tutti i pezzi disponibili.

```
6  public class Puzzle : MonoBehaviour, BuilderinoManager
7  {
8     public GameObject finalLight;
9
10    private int pieces = 17;
11    private int placed = 0;
12
13    public void Built()
14    {
15       placed++;
16    }
17
18    void Update ()
19    {
20       if (placed == pieces)
21       {
22          finalLight.SetActive(true);
23          this.gameObject.GetComponent<AudioSource>().Play();
24          Destroy(this);
25       }
26    }
27 }
```

Allo stesso modo l'implementazione Puzzle, usata nell'ultima esperienza, riproduce un suono diverso e accende la luce responsabile dell'illuminazione intensa del Giudizio Universale. Questo principio di programmazione rende facile estendere l'utilizzo del codice in esame per usi futuri anche decisamente differenti in caso ve ne fosse bisogno, senza doverlo duplicare e, in caso di modifiche future, doverle ripetere anch'esse più volte per ogni versione presente.

4

Conclusioni

Sono soddisfatto del prodotto finale. Quello che doveva essere una semplice rielaborazione è divenuto un progetto articolato frutto di settimane di programmazione, collaudo, discussione. Usualmente, questo prodotto è quello che si definirebbe una demo, ma in un mercato come quello VR dove la durata delle esperienze è effettivamente definita in minuti, anche per l'estraniazione totale richiesta all'utenza, non siamo lontani da quelli realizzati da nomi ben più blasonati. In particolare, ritengo le interazioni coinvolgenti e non banali e anche la realizzazione grafica, mio tallone d'Achille in quanto proveniente da un contesto di sola programmazione, si è rivelata adeguata. Ciononostante, se dovessi venderlo, individuerei il principale problema da risolvere proprio nella rifinitezza dell'opera nel suo complesso, cercando modelli grafici più definiti e un modo migliore per comunicare con l'utente, come un *avatar* tridimensionale che illustra le opere e le azioni da compiere. Per rimanere oggettivi tuttavia, servono più pareri, ed entrano quindi in gioco le prove al pubblico.

4.1 Prove al pubblico

4.1.1 Configurazione e procedure

I test si sono svolti in più luoghi e in giornate diverse ma hanno seguito un metodo comune. I partecipanti si sono riuniti in gruppo e uno alla volta hanno indossato il casco per provare l'esperienza in realtà virtuale mentre gli altri assistevano. Una volta finito, ho somministrato ad ognuno di loro un questionario verbale o scritto con le domande riportate in queste sezioni e registrato le risposte.

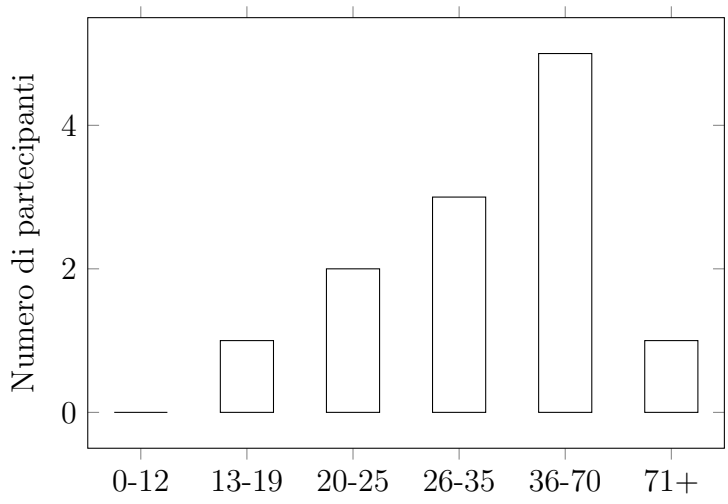

Figura 4.1: Età

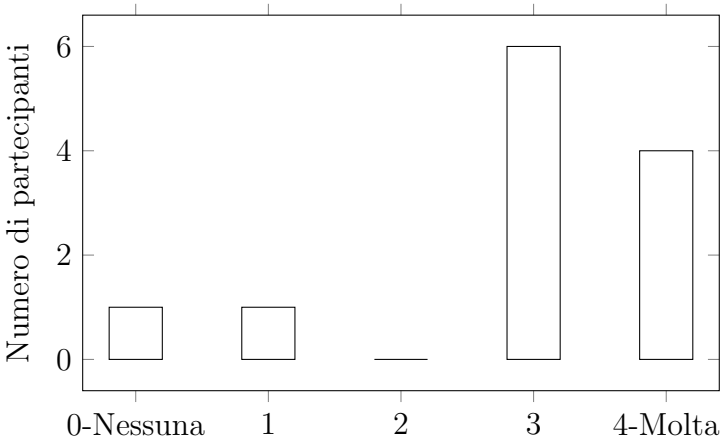

Figura 4.2: Esperienza in VR

4.1.2 Risultati

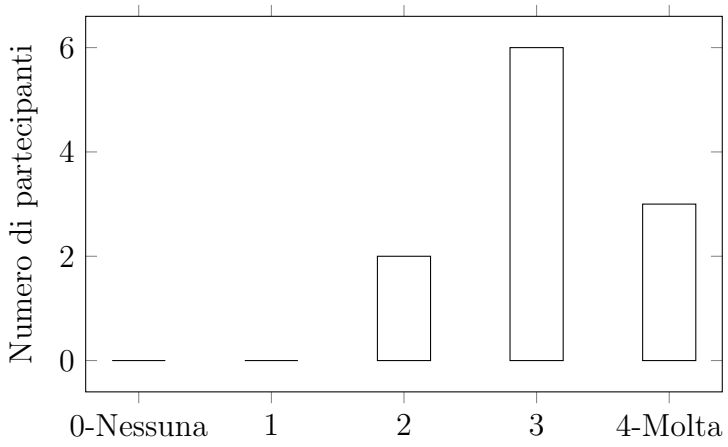

Figura 4.3: Facilità d'uso

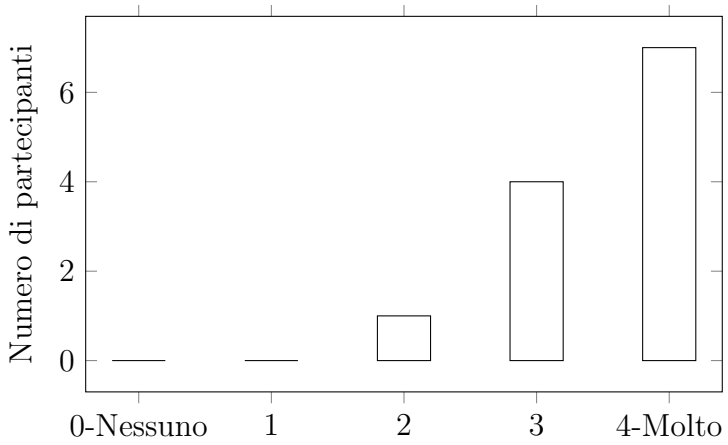

Figura 4.4: Interesse nelle interazioni

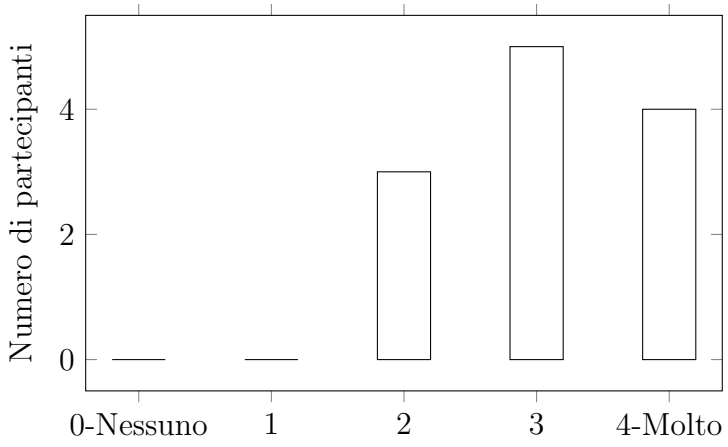

Figura 4.5: Comfort fisico

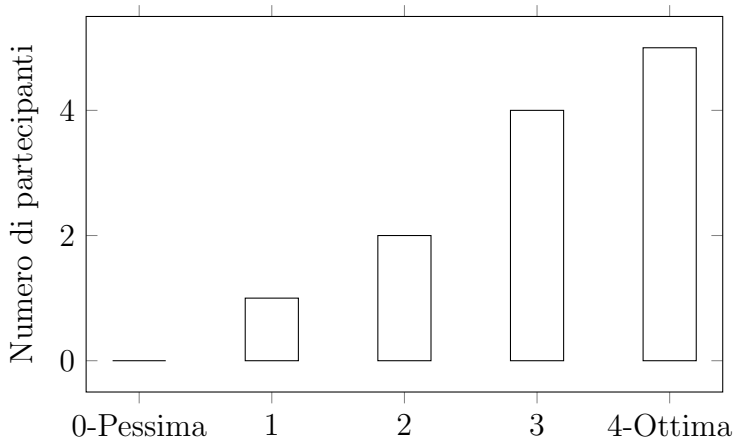

Figura 4.6: Qualità grafica

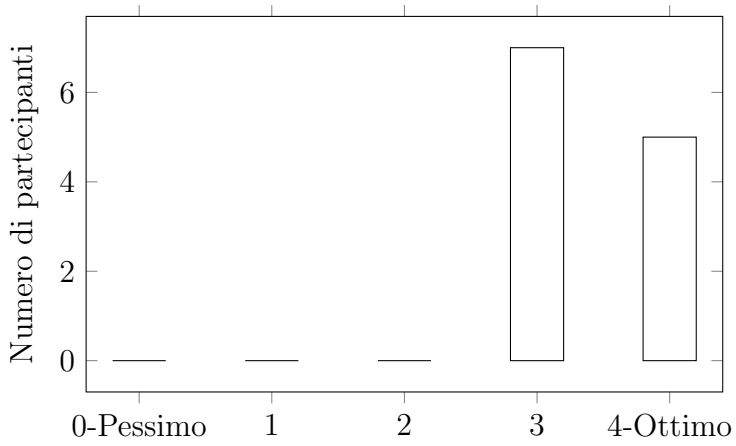

Figura 4.7: Giudizio globale

4.1.3 Osservazioni

Le valutazioni sono andate ben oltre le aspettative. I grafici evidenziano come il pubblico non abbia rilevato una debolezza in nessuno dei parametri valutati, riflettendosi in maniera estremamente positiva sul voto globale. Ho raccolto anche una varietà di consigli per eventuali future iterazioni, ad esempio visualizzare un paio di scarpe nella posizione dell'utente per ridurre lo spaesamento derivante dal non vedere il proprio corpo o introdurre interfacce flottanti a scomparsa per evidenziare alcuni particolari di interesse in ambientazioni reali come la Tomba dei Rilievi o la Cappella Sistina.

4.2 Sviluppi futuri

Il CINECA si è detto interessato al progetto nell'ambito dell'iniziativa già avviata da tempo Experience Etruria del Ministero dei Beni e delle Attività Culturali e del Turismo, ma anche come demo a sé stante per dimostrare le potenzialità della realtà virtuale nella loro sezione recentemente aperta dedicata alla stessa. Sono fiducioso nelle capacità di espansione del progetto fondato su una solida base di codice, facilmente riadattabile ad usi futuri anche relativamente diversi.

Bibliografia

[1] Michael Abrash. *Latency – the sine qua non of AR and VR*. 29 Dic. 2012. URL: http://blogs.valvesoftware.com/abrash/latency-the-sine-qua-non-of-ar-and-vr/.

[2] Eric Brackett. *Oculus React VR takes you behind the scenes of 'Jumanji' and the British Museum*. 19 Nov. 2017. URL: http://www.digitaltrends.com/virtual-reality/oculus-react-vr-offerings/.

[3] Jon Brodkin. *How Unity3D Became a Game-Development Beast*. 3 Giu. 2013. URL: http://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/.

[4] Alex Colgan. *How Does the Leap Motion Controller Work?* 9 Ago. 2014. URL: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/.

[5] Anna Dichiarante. *Roma, realtà aumentata e colori: è la nuova vita dell'Ara Pacis*. 10 Ott. 2016. URL: http://roma.repubblica.it/cronaca/2016/10/10/news/roma_l_ara_pacis_rivive_in_realta_aumentata-149486071/.

[6] Benj Edwards. *Unraveling The Enigma Of Nintendo's Virtual Boy, 20 Years Later*. 21 Ago. 2015. URL: http://www.fastcompany.com/3050016/unraveling-the-enigma-of-nintendos-virtual-boy-20-years-later.

[7] Hoyt Hilsman. *VR Experience at LA Museum of Natural History*. 22 Mar. 2017. URL: http://www.huffingtonpost.com/entry/vr-experience-at-la-museum-of-natural-history_us_58d2c1f0e4b062043ad4af44.

[8] Daniele Pipitone. *Il Sarcofago degli Sposi ora in 3D*. 16 Giu. 2014. URL: http://www.archeomatica.it/documentazione/il-sarcofago-degli-sposi-ora-in-3d.

[9] Leroy Spence. *VR tech in manufacturing: One-time "Sword of Damocles" has prosperous future*. 14 Dic. 2016. URL: http://www.smart2zero.com/news/vr-tech-manufacturing-one-time-sword-damocles-has-prosperous-future.

[10] Stanley Grauman Weinbaum. *Pygmalion's Spectacles*. 1935.

[11] Wikipedia. *HSL and HSV*. 2009. URL: http://en.wikipedia.org/wiki/HSL_and_HSV.

[12] Wikipedia. *Oculus Rift*. 2016. URL: http://en.wikipedia.org/wiki/Oculus_Rift.