POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Creazione di hotspot in ambienti VR 360

Relatore: Prof.re Marco Gribaudo

Tesi di Laurea Magistrale di:
Luca Bigoni, matricola 814942

Anno Accademico 2016 - 2017

“You can't depend on your eyes
when your imagination is out of focus”

Mark Twain

RINGRAZIAMENTI

Ringrazio innanzitutto i miei genitori, che mi hanno sostenuto e aiutato durante tutto il percorso universitario, e i miei amici e compagni di corso che mi hanno permesso di superare i momenti di maggior difficoltà.
Un ringraziamento particolare al professor Marco Gribaudo che con le sue conoscenze, la sua disponibilità e i suoi consigli, mi ha aiutato nella realizzazione di questo progetto di tesi.

Indice

Elenco figure

Elenco algoritmi

Sommario

In questo lavoro di tesi si vuole affrontare la problematica della creazione e riconoscimento di zone sensibili (hotspot) in un'immagine, all'interno di un'applicazione di realtà virtuale.
L'obiettivo è quello di fornire un esempio di applicativo che permetta di navigare tra diversi ambienti, eventualmente selezionabili dall'utente, in base alla zona che si sta osservando.
Sono già presenti sviluppi simili, realizzati con il framework VRview, il quale, però, permette di realizzare solo applicazioni online, utilizzando il linguaggio HTML5.
In questo caso, invece, verrà fornita un'alternativa realizzata in ambiente UNITY, quindi utilizzabile offline e su qualsiasi sistema operativo desktop o mobile, sfruttando il framework GoogleVR, ottimizzato per l'uso con il visore Google Cardboard.

Abstract

The presented thesis work is aimed at dealing with the problem of creating and recognizing hotspots in an image, inside a Virtual Reality application.
The goal is to give an example of application that allow to navigate among different environments, possibly chosen by user, depending on the currently observed zone.
You can find similar developments realized with VRview framework, but it permits to program online applications only, using HTML5 language. In this case an alternative will be presented, developed in UNITY environment, that can be used offline and on every operating system, desktop or mobile, using the GoogleVR framework, optimized for use with Google Cardboard headset.

PAROLE CHIAVE: Realtà virtuale, Unity, Google Cardboard, GoogleVR, VRview, hotspot.

Capitolo 1

Introduzione

Nel mondo dell'informatica, la Realtà Virtuale (conosciuta anche come VR o Virtual Reality) è andata incontro, negli ultimi anni, a uno sviluppo senza precedenti, legato soprattutto alla continua evoluzione di smartphone e console per videogiochi.
Infatti, è solo dagli anni 2000 che visori e altre tecnologie VR sono conosciuti e utilizzati da un pubblico molto ampio.
Tuttavia le prime ricerche in quest' ambito ebbero inizio già negli anni '50, quando i primi prototipi dei visori moderni videro la luce. Purtroppo in quel periodo, sia a livello hardware sia software, non era stato raggiunto uno sviluppo tale da permettere un grande impiego di quella tecnologia.
Lo scopo della realtà virtuale è quello di far vivere all'utente l'esperienza di una completa immersione in un ambiente, reale o immaginario, e di farlo interagire con gli elementi presenti nella scena, tramite il solo movimento della testa o con l'ausilio di controller di vario genere.
Si evince che una delle funzionalità principali di qualsiasi applicazione VR è il riconoscimento di *"zone sensibili"* (hotspot) all'interno dell'ambiente, ossia zone che, una volta osservate, modificano la scena o permettono una qualche azione da parte dell'utilizzatore.
Online è possibile trovare un esempio di applicazione, basata sul framework VRview, che permette di navigare all'interno di un acquario, spostandosi tra le diverse zone della sala principale.
Questo è uno dei possibili sviluppi di un'applicazione di questo tipo, ma è usufruibile solo online; inoltre presenta delle zone sensibili sempre in evidenza, anche quando non vengono osservate direttamente.

In questo lavoro di tesi si vuole, invece, fornire un'applicazione alternativa usufruibile offline, possibilmente personalizzabile, e con un meccanismo che permetta di nascondere le zone sensibili se non osservate, per non risultare fastidiose all'utente finale durante la semplice esplorazione dell'ambiente.
La soluzione viene sviluppata in ambiente UNITY, così da non avere problemi di compatibilità con i vari sistemi operativi, e tramite l'utilizzo del framework GoogleVR, per ottimizzare l'applicazione e permetterne l'utilizzo soprattutto tramite il visore Google Cardboard, il più semplice ed economico visore attualmente in commercio.

1.1 Struttura della tesi

La tesi è strutturata come segue:

- Capitolo 1: Vengono introdotti i dettagli principali del lavoro di tesi, e la sua struttura.

- Capitolo 2: Viene presentato lo stato dell'arte, ripercorrendo la storia della realtà virtuale, soprattutto riguardo alla nascita e allo sviluppo dei vari modelli di headset, per poi concentrarsi su uno degli ultimi modelli di visore attualmente in commercio, al quale si riferisce la tesi: il Google Cardboard.

- Capitolo 3: Viene descritto il framework VRview e viene presentata un'applicazione che lo sfrutta. Dopo di che vengono spiegate le motivazioni per cui si è scelto di sviluppare un'applicazione alternativa, in ambiente UNITY, utilizzando il framework GoogleVR.

- Capitolo 4: Viene presentata una descrizione dettagliata dello sviluppo e delle funzionalità dell'applicazione.

- Capitolo 5: Vengono tratte le conclusioni sul lavoro presentato e descritti possibili sviluppi e utilizzi futuri.

Capitolo 2

Stato dell'arte

Con il termine *realtà virtuale* si intendono tutte quelle tecnologie informatiche che utilizzano software per generare immagini, suoni e sensazioni che replicano un ambiente reale, e simulare la presenza di un utente che può interagirvi all'interno[1].
La realtà virtuale è stata definita come "una realistica e immersiva simulazione di un ambiente tridimensionale, creata utilizzando software interattivi e controllata dal movimento del corpo" [2].
In questo capitolo si vuole mostrare l'evoluzione della realtà virtuale, dagli esordi ai giorni nostri, dal punto di vista di hardware e software utilizzati, fino a descrivere i visori e le applicazioni attualmente in commercio.

2.1 Storia della realtà virtuale

Allo stato odierno, l'esperienza della realtà virtuale è fornita principalmente tramite l'utilizzo di un visore, o headset, montato sulla testa (per questo si può chiamare anche head-mounted display, o HMD). Il visore può mostrare un ambiente tramite un display posizionato davanti agli occhi e utilizzando immagini panoramiche a 360° o video stereoscopici sferici, sempre a 360°. In questo caso l'utente vive un'esperienza visiva e sonora, ma può esserci anche interazione con l'ambiente tramite gamepad, game controller particolari, guanti tecnologici o altri strumenti che producono vibrazioni e feedback tattili.
Riguardo al passato, invece, il primo riferimento alla realtà virtuale ci arriva da un'opera di Stanley G. Weinbaum, scritta nel 1935 e

intitolata *"Gli occhiali di Pigmalione",* dove un paio di occhiali magici trasportano il protagonista in una realtà parallela in cui la morte e tutte le sofferenze della vita sembrano non esistere [3].
Negli anni 50, Morton Heilig, un regista, inventore e cameraman statunitense, ebbe l'idea di creare un "cinema del futuro" e iniziò a lavorare a una macchina che potesse permettere non solo di vedere i film o la televisione in 3D, ma anche di coinvolgere ben 4 sensi (vista, udito, olfatto e tatto) in questa esperienza.
Il primo prototipo di questa macchina venne brevettato nel 1962, col nome di *Sensorama*[4], ed era sostanzialmente un cabinato con schermi stereoscopici, altoparlanti stereo e una sedia semovibile.

Figura 2.1: Il Sensorama

L'invenzione era accompagnata da 5 brevi filmati che permettevano di sperimentare diverse situazioni. Uno di questi video faceva provare l'esperienza di un viaggio in moto all'utente, il quale poteva non solo avere una visione tridimensionale del viaggio e sentire i rumori della città, ma anche sentirne gli odori e provare la sensazione del vento sul viso [5]. Il progetto venne poi abbandonato per la mancanza dei fondi necessari a sostenerlo.
Nel 1968 invece, l'informatico americano Ivan Sutherland inventò quello che è considerato il primo visore HMD. Abbastanza arretrato in termini di user interface e realismo (si potevano visualizzare solo immagini wire-frame di stanze), era inoltre talmente grande da poter

essere indossato solo se appeso al soffitto, da cui il meritato nome di "spada di Damocle".
Rilevante anche l'Aspen Movie Map [6], creato al MIT nel 1978 e considerato tra i primi esempi di sistema ipermediale, precursore degli street view moderni, come il Google street view.
Con questo programma l'utente poteva percorrere virtualmente le strade della città di Aspen, in Colorado, in 3 modalità diverse: Inverno, Estate e Poligoni. L'ultima modalità rappresentava semplicemente dei modelli 3D della città, mentre le prime due erano basate su fotografie reali. I ricercatori avevano montato 4 fotocamere ad ogni lato di un'automobile, le quali scattavano una foto ogni 10 piedi di distanza. Le immagini venivano poi salvate su laserdisc e collegate, tramite database, a delle coordinate specifiche sulla mappa 2D della città.
Negli anni 80 il termine "realtà virtuale" venne popolarizzato da Jaron Lanier, un pioniere di questo campo, che nel 1985 fondò la VPL Research.
Tra i prodotti di punta dell'azienda si annoverano:

- DATA GLOVE: inventato da Thomas Zimmerman, questo guanto tecnologico permetteva all'utilizzatore di spostare e muovere oggetti virtuali. Era collegato a un computer da un cavo e inviava i movimenti della mano tramite 6502 microcontrollori integrati, programmati da Mitch Altman.

- EYEPHONE [7]: un primitivo HMD che poteva tenere traccia dei movimenti della testa

- DATA SUIT: una tuta completa con sensori che permettono di misurare i movimenti di braccia, gambe e tronco.

Figura 2.2: Una VPL Data Suit

Ma è solo tra la fine degli anni 80 e l'inizio degli anni 90 che la realtà virtuale inizia ad essere un concetto veramente diffuso, grazie all'attenzione dei media, tanto che alcuni, addirittura, paragonarono l'invenzione della realtà virtuale a quella dell'aeroplano dei fratelli Wright [8].
Nel 1990, Jonathan Waldern, un ricercatore nell'ambito VR, mostrò *Virtuality* ad una esibizione di Computer Grafica tenuta all'Alexandra Palace di Londra.
Questo sistema altro non era che un visore che faceva girare un videogioco arcade ~~e~~,in cui il giocatore interagiva tramite i movimenti della testa e l'uso di controller specifici.
Da questa invenzione nacque poi una vera e propria linea di giochi Virtuality, prodotta dalla Virtuality Group, una startup nata nel 1985, che produsse sistemi e strumenti per la realtà virtuale di ogni tipo.
Questo è anche il periodo in cui videro la luce le prime riviste legate a questo ambito, tra cui CyberEdge e PCVR.
Comunque molte idee riguardanti la VR non vennero mai sviluppate a causa della limitata potenza di calcolo dell'epoca e dei costi troppo

alti di produzione, che non permettevano l'adozione di questa tecnologia da parte del grande pubblico.
Solo con l'avvento di Internet, le aziende e i consumatori iniziarono a focalizzarsi su di essa.
Nel 1991, la multinazionale Sega Games Co. annunciò il Sega VR headset, sia in versione da cabinato sia per la Mega Drive console. Il visore usava sensori inerziali per tracciare i movimenti della testa.
Nello stesso anno, Virtuality divenne il primo sistema di realtà virtuale prodotto a livello mondiale.
Nel frattempo, Antonio Medina, uno scienziato della NASA, creava un sistema VR per guidare i rover su Marte dalla Terra almeno apparentemente in real time, a dispetto del ritardo dato dalla distanza tra i pianeti; inoltre la rivista *Computer Gaming World* preannunciava la realtà virtuale a portata di tutti entro il 1994 [9].
Infatti, proprio nel 1994, Sega rilasciò il *Sega VR-1 motion simulator arcade attraction* in tutte le sale giochi SegaWorld, che permise al pubblico di vivere una delle più complete esperienze di realtà virtuale.
Entrando in una finta navicella e indossando il visore, si poteva vivere in prima persona una battaglia spaziale, sperimentando anche i movimenti stessi della navicella.
A partire dagli anni 2000, i visori di realtà virtuale hanno conosciuto uno sviluppo molto accentuato, soprattutto in ambito video ludico, e ciò a permesso di rendere sempre più accessibile ai consumatori questa tecnologia.
Tra i più famosi e diffusi vi sono *l'Oculus Rift*, prodotto dalla Oculus VR a partire dal 2010, che può essere sfruttato con una vasta gamma di applicazioni dedicate; il *PlayStation VR*, da utilizzare in coppia con la console di ultima generazione PlayStation 4, e *l'HTC Vive*, che ha la peculiarità di utilizzare una tecnologia "room scale" con cui, grazie all'utilizzo di sensori appositi, si può trasformare un'intera stanza in un ambiente di gioco. Questi sensori, infatti, riconoscono il movimento del corpo del giocatore e lo riflettono nella realtà virtuale.
Infine è particolarmente degno di nota il visore più economico al momento in commercio, il *Google Cardboard* [10]. Venduto sia nella versione di cartone, da assemblare a mano, sia in plastica o

nylonABS, questo visore è un semplice supporto in cui si può inserire un qualsiasi smartphone e quindi sfruttare tutte le applicazioni VR del mondo mobile. Inizialmente era fornito di un bottone magnetico che permetteva di interagire con le applicazioni provocando lo stesso effetto che si ha toccando lo schermo. In seguito a problemi di compatibilità con alcuni smartphone, questo bottone è stato sostituito da un altro tasto che va effettivamente a toccare lo schermo [11].

Figura 2.3: Oculus Rift

Figura 2.4: PlayStation VR

Figura 2.5: HTC Vive

2.2 Google Cardboard

Il Google Cardboard, come accennato nella sezione precedente, è il visore per la realtà virtuale più economico esistente sul mercato.

Figura 2.6: Google Cardboard

Creato da David Coz e Damien Henry, due ingegneri di Google, e presentato per la prima volta alla Google I/O developers conference del 2014, questo visore prende il nome dal fatto che la sua prima versione era stata progettata con una semplice base di cartone (*cardboard* in inglese), contenente semplicemente due lenti per la visione stereoscopica e pochi altri elementi.
Esiste anche la possibilità di comprare un kit per l'assemblaggio manuale del visore, contenente un foglio di cartone ondulato, 2 lenti biconvesse con distanza focale di 45mm, una calamita in neodimio e una calamita a disco in ceramica (nella versione con bottone

magnetico), 2 strisce di velcro per la chiusura dello slot per il cellulare, un elastico per impedire che il telefono scivoli, e infine un'etichetta NFC adesiva, programmata per attivare in automatico le applicazioni Google Cardboard (se lo smartphone è provvisto di tecnologia NFC).
L'ideale dietro a questa invenzione è quello di incoraggiare l'interesse e lo sviluppo di applicazioni legate a questa tecnologia [12].
Cardboard, infatti, è anche il nome di una delle due piattaforme per la realtà virtuale fornite da Google, insieme a *Daydream*. Quest'ultima permette un'esperienza più completa e performante, ma necessita di hardware specifici (headset e telecomando venduti appositamente per l'uso con questa piattaforma) e al momento funziona solo con il sistema operativo Android 7.0 Nougat o superiori. Cardboard, al contrario, è più semplice ma anche più diffusa e utilizzabile.
Esistono diversi SDK [13] (Software Development Kit) per lo sviluppo di applicazioni Cardboard compatibili, che semplificano e automatizzano le funzioni fondamentali (controllo del movimento della testa, attivazione di oggetti sensibili…).
Il kit utilizzato per l'applicazione in oggetto della tesi è quello utilizzabile in ambiente Unity, a partire dalla versione 5.2.1.
Questo supporto permette sia di iniziare a programmare una nuova app VR partendo da zero, sia di adattare alla VR un'applicazione unity 3D già esistente. Inoltre permette la transizione della stessa app dalla modalità VR alla modalità standard.
Tra le funzionalità più importanti fornite dall'SDK, vi sono:

- **head tracking**: la classe GvrViewer rileva i movimenti della testa leggendo i dati dal giroscopio del cellulare e permette di correggere eventuali distorsioni dell'immagine

- **rendering di immagini stereoscopiche**: gestito sempre tramite la classe GvrViewer

- **rilevamento di oggetti sensibili**: la classe GvrReticlePointer disegna un piccolo cerchio al centro della scena, che si dilata quando viene posizionato su un oggetto cliccabile

- **interazione utente - sistema**: la classe GvrArmModel interagisce con un qualsiasi controller gestito dall'utente, mentre la classe GvrController legge i comandi inviati dal controller specifico di Daydream

- **gestione dell'audio spaziale**: la classe GvrAudio gestisce l'audio spaziale.

In questo modo lo sviluppatore dovrà solo personalizzare il layout dell'applicazione e i comportamenti dei vari oggetti con cui l'utente potrà interagire, semplificando di molto la scrittura del codice.

Capitolo 3

Da VR View a Unity

In questo capitolo viene presentata una panoramica sulle funzionalità della tecnologia VR View, in generale e nello specifico per la versione web.
Viene poi fornito e descritto un esempio di applicazione di questo framework, per poterlo infine confrontare con Unity, descrivendo i vantaggi nell'utilizzare questa piattaforma.

3.1 VR View

A marzo 2016 Google ha annunciato la nascita di VR View, un framework dedicato a sviluppatori e programmatori web, per rendere più coinvolgenti e immersive le immagini e i video all'interno di siti, applicazioni o piattaforme come YouTube.
VR View permette, infatti, di integrare immagini e video a 360 gradi dentro a siti web su sistemi operativi desktop e mobile, e nelle app native su Android e iOS [14].
Con questa tecnologia si può fare in modo, per esempio, che un'app di viaggi possa fornire un tour virtuale di una località (o di un ambiente marino, per gli utenti abituati a fare immersioni o snorkeling) per permettere una migliore pianificazione della vacanza, o che un architetto possa far visitare le stanze di un appartamento o edificio prima che questo venga costruito.
VR View risolve, inoltre, il problema della poca disponibilità di hardware VR tra il pubblico. Sebbene il framework sia completamente compatibile con Google Cardboard e permetta la visione

stereoscopica, fornisce anche una finestra singola che mostra l'ambiente a 360 gradi e funziona su tutti gli ambienti supportati.
Immagini e video possono essere integrati sia in formato mono che stereo, ma devono essere salvati in un formato panoramico equirettangolare (uno dei formati più comuni in uso). Inoltre, devono essere rispettate delle particolari specifiche:

SPECIFICHE IMMAGINI

- le immagini devono essere salvate come jpeg, png o gif (jpeg fornisce il miglior meccanismo di compressione)

- per fornire la migliore compatibilità e performance, le dimensioni delle immagini devono essere potenze di 2 (es: 2048 o 1024)

- le immagini mono devono avere un rapporto 2:1 (es: 4096 x 2048)

- le immagini stereo devono avere un rapporto 1:1 (es: 4096 x 4096)

Figura 3.1: a sinistra un'immagine mono (single pano); a destra un'immagine stereo (double stacked pano)

SPECIFICHE VIDEO

- I video devono essere salvati come mp4s e codificati con lo standard di compressione H.264 (o MPEG-4)
- I video mono devono avere un rapporto 2:1
- I video stereo devono avere un rapporto 1:1
- In alcuni casi non sono supportati video con una qualità superiore a 1080p

Essendo VR View una tecnologia "client side", è indipendente dagli strumenti utilizzati per catturare o integrare i contenuti VR. Si può quindi utilizzare qualsiasi fotocamera, videocamera o software di grafica che supporti il formato panoramico equirettangolare.

3.2 VR View sul web

L'integrazione di immagini o video VR in un sito web può essere attuata tramite l'utilizzo di un'API JavaScript che crea e controlla i contenuti di un iframe (un elemento HTML che permette di integrare vari tipi di contenuti in una pagina web), oppure dichiarando esplicitamente l'iframe stesso [15].
Innanzitutto, per utilizzare la classe VR View, si deve includere il seguente script, fornito da Google, nella pagina HTML:

Algoritmo 3.1 script contenente l'API VR View

```
<script src="//storage.googleapis.com/vrview/2.0/build/vrview.min.js">
</script>
```

Per indicare dove inserire l'iframe, si utilizza il seguente tag:

Algoritmo 3.2 tag di inserimento dell'iframe

```
<div id="vrview"></div>
```

Quando la pagina web viene caricata, si deve registrare l'evento di caricamento e chiamare una funzione che crea una nuova istanza della classe VR View:

Algoritmo 3.3 creazione istanza VR View al caricamento

```
window.addEventListener('load', onVrViewLoad)
function onVrViewLoad() {
  var vrView = new VRView.Player('#vrview', {
    video: 'link/to/video.mp4',
    is_stereo: true
  });
}
```

In alternativa si può dichiarare direttamente l'iframe:

Algoritmo 3.4 dichiarazione diretta dell'iframe

```
<iframe src="//storage.googleapis.com/vrview/2.0/embed?video=link/to/
video.mp4&is_stereo=true">
</iframe>
```

Si noti, però, che nella funzione si possono dichiarare ulteriori parametri non disponibili nella dichiarazione dell'iframe.

VRView.Player è il costruttore della classe VR View e necessita di un "selettore" ('#vrview' in questo caso), ossia l'id (indicato precedentemente tra i tag) della posizione dove comparirà l'iframe, e di altri parametri particolari.
Gli eventi possibili sono di 4 tipi, e vengono registrati dalla funzione **VRView.on**, che riceve 2 parametri in ingresso, il nome dell'evento e la funzione che lo gestisce. Per esempio:

Algoritmo 3.5 gestione evento "click" del mouse

```
VRView.on('click', function(event) {
  if (event.id == myHotspotId) {
    // Handle hotspot click.
  }
});
```

Gli eventi possibili sono:

- **Ready**: gestito subito dopo che l'elemento VR viene caricato

- **Error**: gestito se l'elemento VR non viene caricato correttamente
- **Click**: gestito quando avviene un click del mouse (o un tap su uno schermo touch, o viene osservato un hotspot in modalità VR). Può essere usato per passare da un ambiente a un altro, o per attuare comportamenti diversi in base a quale hotspot viene cliccato

- **Modechange**: gestito quando cambia la modalità di rappresentazione (da VR a Fullscreen, o viceversa)

Riguardo ai video VR, esistono 3 funzioni specifiche per la loro gestione:

- **VRView.play**(): attiva il video, al primo caricamento o dopo una pausa
- **VRView.pause**(): mette in pausa il video
- **VRView.setVolume**(double fractionVol): imposta il volume del video a una frazione del massimo valore; fractionVol deve avere un valore compreso tra 0 e 1 (0 = muto, 1 = massimo)

La funzione VRView.setContentInfo() permette di cambiare il contenuto di un iframe, senza doverlo ricaricare. Riceve in ingresso l'URL di un elemento VR e altri parametri:

Algoritmo 3.6 caricamento nuovo contenuto nell'iframe

```
vrView.setContentInfo({
  image: '/url/to/amazing-4096.jpg',
  preview:'/url/to/amazing-512.jpg',
  is_stereo: true
});
```

Una delle funzionalità più importanti messe a disposizione dall'API è la creazione e gestione degli hotspot.
Gli hotspot sono zone dell'elemento VR con cui l'utente può interagire, e possono essere visualizzati su qualsiasi piattaforma, ma le tipologie di interazione sono differenti:

- Ambiente desktop: posizionarsi col mouse su un hotspot ne cambia l'aspetto, mentre il click attiva un evento
- Ambiente mobile: il tap sulla zona sensibile attiva un evento

- Modalità VR: solitamente viene visualizzato un puntatore al centro dello schermo; spostando, tramite il movimento della testa, il puntatore su un hotspot si può attivare un evento a seguito di un tap su un qualsiasi punto dello schermo.

Ogni hotspot è visualizzato come una regione circolare ed è descritto da un ID (per identificarlo univocamente), le coordinate del centro, il raggio del cerchio e la distanza dalla camera (ossia il punto di vista dell'osservatore).
La funzione vrView.addHotspot () aggiunge un hotspot sull'immagine o video:

Algoritmo 3.7 Definizione di un hotspot

```
vrView.addHotspot('ID', {
  pitch: 30,      // In degrees. Up is positive.
  yaw: 20,        // In degrees. To the right is positive.
  radius: 0.05,   // Radius of the circular target in meters.
  distance: 2     // Distance of target from camera in meters.
});
```

I valori delle variabili pitch e yaw sono indicati in gradi e rispetto a coordinate sferiche. Il centro è posizionato di default alle coordinate (0,0). Il range di valori della variabile pitch è [-90, 90] con valori positivi verso l'alto, mentre il range di valori della variabile yaw è [-180, 180] con valori positivi verso destra.
Riguardo alla gestione degli eventi causati dall'interazione dell'utente con un hotspot, si rimanda all'algoritmo 3.5.

3.3 Milstein Hall of Ocean Life

Dalla metà del 2015, l'american museum of natural history ha dato vita, in collaborazione con Google, a una nuova iniziativa per favorire la visita, se pur virtuale, delle proprie sale ai ragazzi delle scuole di tutto il mondo.
Si tratta delle *"Expeditions"* [16] che, secondo Google, "permettono agli insegnanti di dare vita alle loro lezioni portando gli studenti in gita scolastica ovunque possano immaginare" [17].
Tra le sale del museo che è possibile visitare in questo modo, vi sono la "Hall of North American Mammals", la "Hall of Saurischian Dinosaurs" e la "Milstein Hall of Ocean Life".
Di quest'ultima si può trovare online [18] la rappresentazione a 360 gradi, realizzata tramite VR View.

Figura 3.2: Milstein Hall of Ocean Life - prima prospettiva

Questa rappresentazione permette di visitare la sala da più angolazioni, e, per ogni angolazione, si può avere una visione panoramica del luogo.
Utilizzando VR View, si può notare che gli hotspot sono rappresentati da cerchi di colore grigio e sono sempre presenti sulla scena,

indipendentemente dal punto che l'utente sta osservando in quel momento.
Cliccando sul hotspot al centro, in figura 3.2, la scena cambia e si può osservare la sala usando quel hotspot come punto di vista (figura 3.3).

Figura 3.3: Milstein Hall of Ocean Life - seconda prospettiva

Trascinando la figura si possono osservare altre zone della sala (figura 3.4).

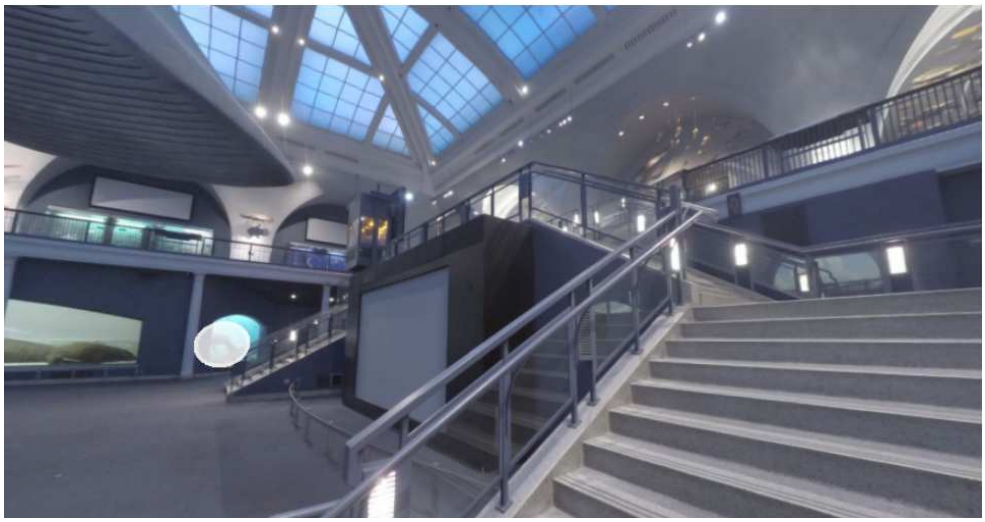


Figura 3.4: Milstein Hall of Ocean Life - seconda prospettiva dopo il trascinamento

3.4 I vantaggi di UNITY

UNITY è un motore grafico multi-piattaforma sviluppato dalla "Unity Technologies" e utilizzato principalmente per la creazione di videogiochi [19].
Annunciato per la prima volta nel 2005 alla Worldwide Developers Conference di Apple, a San Francisco, inizialmente era stato progettato per funzionare solo sul sistema operativo OS X. Nel tempo è stato esteso a ben 27 piattaforme diverse [20].
È giunto attualmente alla sua quinta versione (o major), chiamata appunto Unity 5.
Con l'avvento della realtà virtuale, Unity si è dotato di un supporto VR nativo che permette di realizzare applicazioni VR senza l'ausilio di plug-in esterni.

Attivando questo supporto, si ottiene:

- una versione stabile dell'applicazione per ogni dispositivo VR
- una singola interfaccia API che interagisce con i diversi dispositivi
- un'unica cartella di progetto senza plug-in esterni
- la possibilità di funzionare su più dispositivi senza dover apportare modifiche
- performance aumentate per ambienti VR

Inoltre, tramite l'utilizzo di vari SDK (come l'SDK di Google Cardboard, utilizzato in questo progetto) si possono importare librerie

specifiche per la programmazione VR, che semplificano il codice e forniscono funzionalità standard.
Di conseguenza, sviluppare un progetto utilizzando Unity permette di creare applicazioni che funzionano su tutti i sistemi operativi supportati dal motore grafico, e non solo online (a differenza di VR view per web).
Infine, essendo Unity un vero e proprio ambiente di sviluppo, permette di trovare diverse soluzioni per la creazione di una certa funzionalità, come il riconoscimento di hotspot, senza essere vincolati a metodi predefiniti
Nel prossimo capitolo verrà spiegato come, in questo progetto, si è deciso di affrontare il problema del riconoscimento di zone sensibili, avendo come principale obbiettivo quello di non renderle costantemente visibili, come accade nell'esempio fornito al paragrafo 3.3, ma di attivarle solo quando osservate dall'utente.

Capitolo 4

L'applicazione

In questo capitolo verranno mostrate nel dettaglio le diverse fasi della programmazione dell'applicazione oggetto della tesi, denominata "HotspotNavigator", dalla preparazione delle immagini da utilizzare fino alla scrittura del codice.

4.1 Creazione degli ambienti

Per creare le immagini da utilizzare nell'applicazione, è stato usato *PHOTOSHOP* [21], un software di elaborazione immagini proprietario, prodotto dalla Adobe System Incorporated.
Il motivo per cui è stato scelto questo software è la possibilità di creare e gestire facilmente nuovi livelli sopra un'immagine di base, per potervi aggiungere altre immagini o elementi grafici, come forme geometriche o testi.

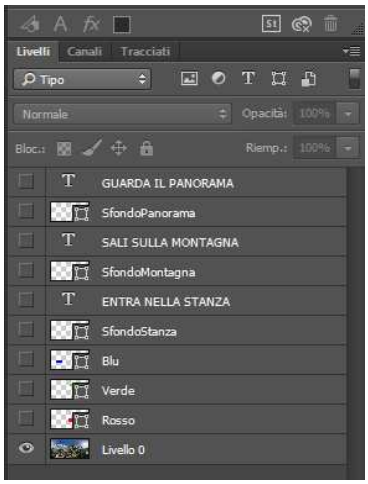
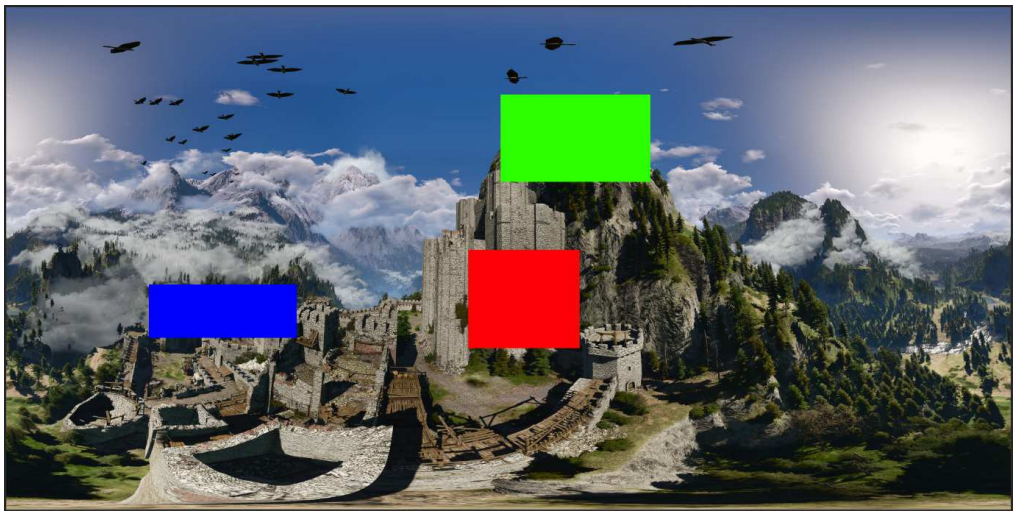


Figura 4.1: Livelli creati sull'immagine base (Livello 0)

Innanzitutto si sceglie un'immagine base che sarà utilizzata come ambiente principale.

Figura 4.2: Ambiente iniziale

Sopra questa si creano dei livelli su cui disegnare le macchie di colore (possono essere forme geometriche o a piacere) che serviranno a individuare gli hotspot dell'immagine.



13 Figura 4.3: Creazione hotspot

Riducendo a zero l'opacità dell'ambiente base, si potrà salvare l'immagine comprendente solo le macchie colorate.

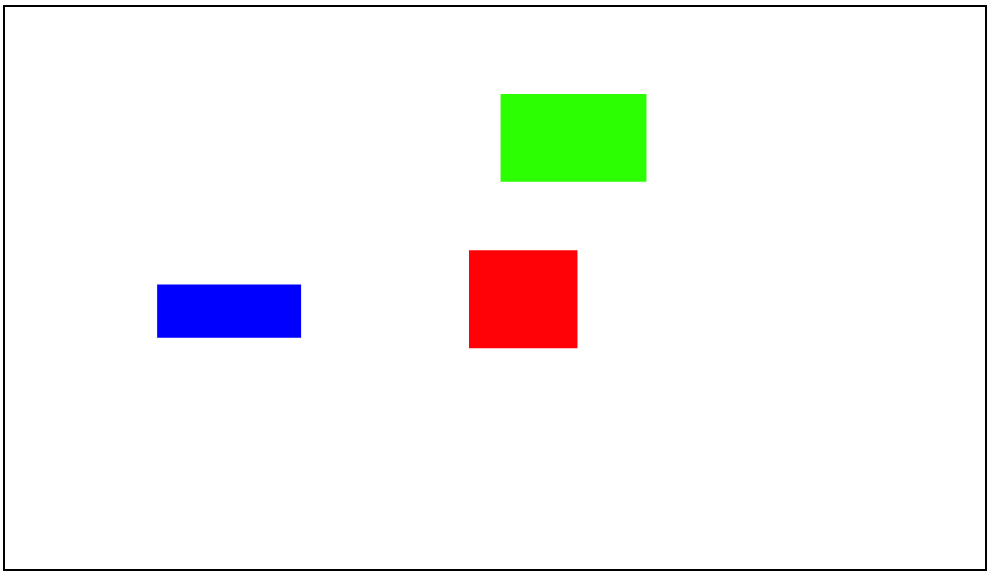


Figura 4.4: Macchie di colore

Successivamente si creano le immagini che compariranno a seguito dell'osservazione di uno degli hotspot. In questo caso si è deciso di evidenziare le zone sensibili con un messaggio di testo su sfondo bianco, che indica il tipo di ambiente dove ci si può spostare.

Figura 4.5: Messaggio "entra nella stanza" - hotspot rosso

Figura 4.6: Messaggio "guarda il panorama" - hotspot blu

Figura 4.7: Messaggio "Sali sulla montagna" - hotspot verde

Quando l'utente eseguirà un tap sullo schermo tramite il bottone del Google Cardboard (o comando simile) mentre sta osservando una zona sensibile, l'ambiente osservato cambierà, a seconda di quale hotspot è stato selezionato.

In questo caso gli ambienti scelti sono i seguenti:

Figura 4.8: Ambiente "stanza" - hotspot rosso

Figura 4.9: Ambiente "panorama" - hotspot blu

Figura 4.10: Ambiente "picco della montagna" - hotspot verde

4.2 Sviluppo dell'applicazione

4.2.1 Creazione e preparazione degli oggetti della scena

Come già esposto precedentemente, per lo sviluppo dell'applicazione è stato usato il motore grafico UNITY.
Innanzitutto è stato importato nell'ambiente di sviluppo il Google VR SDK di Google Cardboard, scaricato dal sito ufficiale [22], ed è stato importato nella scena principale *(MainScene)* l'oggetto "GvrMain" contenente tutta la logica che gestisce la visione stereoscopica a 360 gradi (oggetti figli e script).

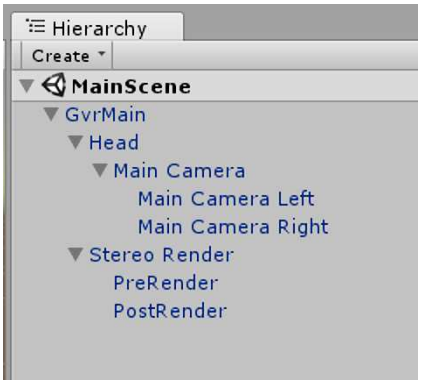


Figura 4.11: struttura dell'oggetto GvrMain

A questo è stato aggiunto un oggetto Camera, chiamato *"Camera2",* come figlio dell'oggetto Head.
Camera2 è l'oggetto che sta alla base del meccanismo implementato in questa applicazione, infatti è grazie a questa camera che si possono riconoscere gli hotspot.

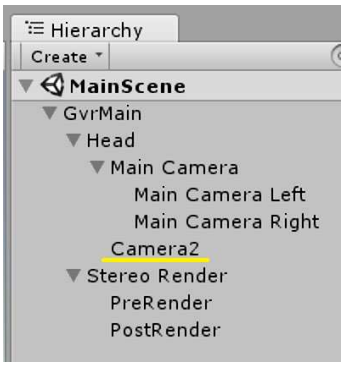


Figura 4.12: Camera2

È stato deciso di introdurre la possibilità di far decidere all'utente quali immagini caricare nell'applicazione, per poterla rendere personalizzabile. Quindi, oltre a GvrMain, nella scena sono presenti degli oggetti atti a questo scopo.
In particolare, gli oggetti sono *"Gallery", "ColorsVal"* e *"Camera".*

- Gallery: è un oggetto vuoto a cui è legato uno script che gestisce tutta la logica di scelta e inserimento delle immagini e dei valori RGB dei colori utilizzati per gli hotspot.

- ColorsVal: è un altro oggetto vuoto in cui vengono salvati i valori RGB dei colori.

- Camera: è un oggetto di tipo camera che serve a renderizzare lo sfondo della scena iniziale; senza questo si vedrebbe uno sfondo nero.

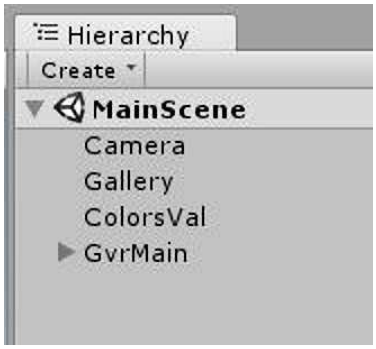


Figura 4.13: struttura completa della scena

Fatto questo, è stato inserito un componente Skybox sia a MainCamera che a Camera2.
Le Skybox servono a fare in modo che un ambiente, o un'immagine in generale, circondi completamente la scena e ne faccia da sfondo. Tutte le immagini visualizzate nell'app saranno assegnate (tramite dei Material) alla Skybox di MainCamera, mentre l'immagine delle macchie di colore sarà assegnata alla Skybox di Camera2.
Infine sono stati creati i Material, di tipo Cubemap e inizialmente vuoti, che, appunto, conterranno le immagini (sottoforma di Texture) selezionate dalla galleria.

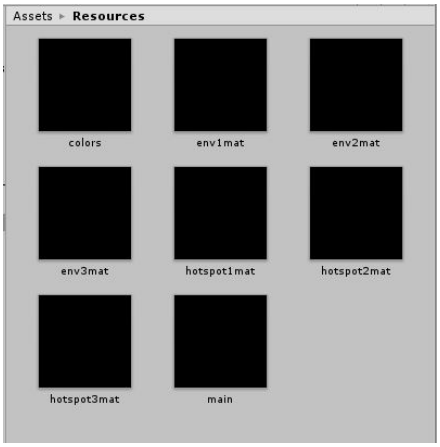


Figura 4.14: creazione dei Material

Con questo si completa la fase di preparazione degli oggetti della scena.

4.2.2 Scrittura degli script

Accedendo all'applicazione, compare la schermata di selezione delle immagini e inserimento dei valori RGB dei colori. Questa è creata e gestita dallo script GalleryUse

Algoritmo 4.1 GalleryUse - gestione schermata iniziale

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GalleryUse : MonoBehaviour {

    public Texture2D mainImage, HS1Image, HS2Image, HS3Image;
    public Texture2D ColorsImage, ENV1Image, ENV2Image, ENV3Image;

    bool isMainImageLoaded, isHS1Loaded, isHS2Loaded, isHS3Loaded;
    bool isColorsLoaded, isENV1Loaded, isENV2Loaded, isENV3Loaded;
    bool buttonMain, buttonHS1, buttonHS2, buttonHS3;
    bool buttonColors, buttonENV1, buttonENV2, buttonENV3;
    bool buttonOK;
    public string hexCol1R, hexCol1G, hexCol1B;
    public string hexCol2R, hexCol2G, hexCol2B;
    public string hexCol3R, hexCol3G, hexCol3B;
    WWW www;
    int wUnit = Screen.width/20;
    int hUnit = Screen.height/10;
    int i,j;
    string text;
    private GameObject gvrmain, gallery, colorList;
    private Material mainMat,hs1Mat,hs2Mat,hs3Mat;
    private Material colMat,env1Mat,env2Mat,env3Mat;
    private Color newColor = new Color();
    private float m_direction = 0.0f;
```

/ Inizializzazione oggetti della scena e disabilitazione di GvrMain (per non far visualizzare la schermata stereoscopica) */*

```
void Start () {
        gvrmain = GameObject.FindGameObjectWithTag ("gvrmain");
        gallery = GameObject.FindGameObjectWithTag ("gallery");
        colorList = GameObject.FindGameObjectWithTag ("colorsval");
        gvrmain.SetActive (false);
    }
```

```
/*creazione elementi grafici*/
void OnGUI () {
        GUIStyle style = new GUIStyle("button");
        style.fontSize = 25;
        GUIStyle styletxt = new GUIStyle ("textfield");
        styletxt.fontSize = 25;

        buttonMain = GUI.Button (new Rect(wUnit,hUnit,wUnit*2,hUnit),
                                 "MAINiMG",style);
        buttonHS1 = GUI.Button (new Rect(wUnit,hUnit*3,wUnit*2,hUnit),
                                "HOTSPOT1",style);
        buttonHS2 = GUI.Button (new Rect(wUnit,hUnit*5,wUnit*2,hUnit),
                                "HOTSPOT2",style);
        buttonHS3 = GUI.Button (new Rect(wUnit,hUnit*7,wUnit*2,hUnit),
                                "HOTSPOT3",style);
        buttonColors = GUI.Button (new Rect(wUnit*9,hUnit,wUnit*2,hUnit),
                                   "COLORS",style);
        buttonENV1 = GUI.Button (new Rect(wUnit*9,hUnit*3,wUnit*2,hUnit),
                                 "ENV1",style);
        buttonENV2 = GUI.Button (new Rect(wUnit*9,hUnit*5,wUnit*2,hUnit),
                                 "ENV2",style);
        buttonENV3 = GUI.Button (new Rect(wUnit*9,hUnit*7,wUnit*2,hUnit),
                                 "ENV3",style);
        buttonOK = GUI.Button (new Rect(wUnit*17,hUnit*8,wUnit*2,hUnit),
                               "OK",style);

        hexCol1R = GUI.TextField (new Rect(wUnit*17,hUnit,wUnit*2 +
                                  wUnit/2,hUnit/2),hexCol1R,3,styletxt);
        hexCol1G = GUI.TextField (new Rect(wUnit*17,hUnit + hUnit/2,
                                  wUnit*2 + wUnit/2,hUnit/2),
                                  hexCol1G,3,styletxt);
        hexCol1B = GUI.TextField (new Rect(wUnit*17,hUnit*2,wUnit*2 +
                                  wUnit/2,hUnit/2) hexCol1B,3,styletxt);

        hexCol2R = GUI.TextField (new Rect(wUnit*17,hUnit*3,wUnit*2 +
                                  wUnit/2,hUnit/2),hexCol2R,3,styletxt);
        hexCol2G = GUI.TextField (new Rect(wUnit*17,hUnit*3 + hUnit/2,
                                  wUnit*2 + wUnit/2,hUnit/2),
                                  hexCol2G,3,styletxt);
        hexCol2B = GUI.TextField (new Rect(wUnit*17,hUnit*4,wUnit*2 +
                                  wUnit/2,hUnit/2), hexCol2B,3,styletxt);

        hexCol3R = GUI.TextField (new Rect(wUnit*17,hUnit*5,wUnit*2 +
                                  wUnit/2,hUnit/2),hexCol3R,3,styletxt);
        hexCol3G = GUI.TextField (new Rect(wUnit*17,hUnit*5 + hUnit/2,
                                  wUnit*2 + wUnit/2,hUnit/2),
                                  hexCol3G,3,styletxt);

        hexCol3B = GUI.TextField (new Rect(wUnit*17,hUnit*6,wUnit*2 +
                                   wUnit/2,hUnit/2),hexCol3B,3,styletxt);
```

*/*Quando viene premuto un bottone, viene richiamato un Plugin java che apre la galleria e salva l'immagine selezionata*/*

```
if(buttonMain){
            text = "MAINiMG";
            isMainImageLoaded = false;
            AndroidJavaClass ajc = new AndroidJavaClass ("com.unity3d.player.UnityPlayer");
            AndroidJavaObject ajo = new AndroidJavaObject ("com.bigoni.gallery.UnityBinder");
            ajo.CallStatic("OpenGallery",ajc.GetStatic<AndroidJavaObject>("currentActivity"));
        }

        if(buttonHS1){
            text = "HOTSPOT1";
            isHS1Loaded = false;
            AndroidJavaClass ajc = new AndroidJavaClass ("com.unity3d.player.UnityPlayer");
            AndroidJavaObject ajo = new AndroidJavaObject ("com.bigoni.gallery.UnityBinder");
            ajo.CallStatic("OpenGallery",ajc.GetStatic<AndroidJavaObject>("currentActivity"));
        }

        if(buttonHS2){
            text = "HOTSPOT2";
            isHS2Loaded = false;
            AndroidJavaClass ajc = new AndroidJavaClass ("com.unity3d.player.UnityPlayer");
            AndroidJavaObject ajo = new AndroidJavaObject ("com.bigoni.gallery.UnityBinder");
            ajo.CallStatic("OpenGallery",ajc.GetStatic<AndroidJavaObject>("currentActivity"));
        }

        if(buttonHS3){
            text = "HOTSPOT3";
            isHS3Loaded = false;
            AndroidJavaClass ajc = new AndroidJavaClass ("com.unity3d.player.UnityPlayer");
            AndroidJavaObject ajo = new AndroidJavaObject ("com.bigoni.gallery.UnityBinder");
            ajo.CallStatic("OpenGallery",ajc.GetStatic<AndroidJavaObject>("currentActivity"));
        }

        if(buttonColors){
            text = "COLORS";
            isColorsLoaded = false;
            AndroidJavaClass ajc = new AndroidJavaClass ("com.unity3d.player.UnityPlayer");
            AndroidJavaObject ajo = new AndroidJavaObject ("com.bigoni.gallery.UnityBinder");
            ajo.CallStatic("OpenGallery",ajc.GetStatic<AndroidJavaObject>("currentActivity"));
        }

        if(buttonENV1){
            text = "ENV1";
            isENV1Loaded = false;
            AndroidJavaClass ajc = new AndroidJavaClass ("com.unity3d.player.UnityPlayer");
            AndroidJavaObject ajo = new AndroidJavaObject ("com.bigoni.gallery.UnityBinder");
            ajo.CallStatic("OpenGallery",ajc.GetStatic<AndroidJavaObject>("currentActivity"));
        }

        if(buttonENV2){
            text = "ENV2";
            isENV2Loaded = false;
            AndroidJavaClass ajc = new AndroidJavaClass ("com.unity3d.player.UnityPlayer");
            AndroidJavaObject ajo = new AndroidJavaObject ("com.bigoni.gallery.UnityBinder");
            ajo.CallStatic("OpenGallery",ajc.GetStatic<AndroidJavaObject>("currentActivity"));
        }

        if(buttonENV3){
            text = "ENV3";
            isENV3Loaded = false;
            AndroidJavaClass ajc = new AndroidJavaClass ("com.unity3d.player.UnityPlayer");
            AndroidJavaObject ajo = new AndroidJavaObject ("com.bigoni.gallery.UnityBinder");
            ajo.CallStatic("OpenGallery",ajc.GetStatic<AndroidJavaObject>("currentActivity"));
```

/ Viene creata una Texture per ogni immagine selezionata. Avviene una copia pixel per pixel dell'immagine nella Texture corrispondente*/*

```
if (www != null && www.isDone && text == "MAINiMG") {
            mainImage = new Texture2D (www.texture.width, www.texture.height);
            mainImage.SetPixels32 (www.texture.GetPixels32());
            mainImage.Apply ();
            www = null;
            isMainImageLoaded = true;
            text = " ";
        }
        if (www != null && www.isDone && text == "HOTSPOT1") {
            HS1Image = new Texture2D (www.texture.width, www.texture.height);
            HS1Image.SetPixels32 (www.texture.GetPixels32());
            HS1Image.Apply ();
            www = null;
            isHS1Loaded = true;
            text = " ";
        }
        if (www != null && www.isDone && text == "HOTSPOT2") {
            HS2Image = new Texture2D (www.texture.width, www.texture.height);
            HS2Image.SetPixels32 (www.texture.GetPixels32());
            HS2Image.Apply ();
            www = null;
            isHS2Loaded = true;
            text = " ";
        }
        if (www != null && www.isDone && text == "HOTSPOT3") {
            HS3Image = new Texture2D (www.texture.width, www.texture.height);
            HS3Image.SetPixels32 (www.texture.GetPixels32());
            HS3Image.Apply ();
            www = null;
            isHS3Loaded = true;
            text = " ";
        }
        if (www != null && www.isDone && text == "COLORS") {
            ColorsImage = new Texture2D (www.texture.width, www.texture.height);
            ColorsImage.SetPixels32 (www.texture.GetPixels32());
            ColorsImage.Apply ();
            www = null;
            isColorsLoaded = true;
            text = " ";
        }
        if (www != null && www.isDone && text == "ENV1") {
            ENV1Image = new Texture2D (www.texture.width, www.texture.height);
            ENV1Image.SetPixels32 (www.texture.GetPixels32());
            ENV1Image.Apply ();
            www = null;
            isENV1Loaded = true;
            text = " ";
        }
        if (www != null && www.isDone && text == "ENV2") {
            ENV2Image = new Texture2D (www.texture.width, www.texture.height);
            ENV2Image.SetPixels32 (www.texture.GetPixels32());
            ENV2Image.Apply ();
            www = null;
            isENV2Loaded = true;
            text = " ";
        }
```

```
        if (www != null && www.isDone && text == "ENV3") {
            ENV3Image = new Texture2D (www.texture.width, www.texture.height);
            ENV3Image.SetPixels32 (www.texture.GetPixels32());
            ENV3Image.Apply ();
            www = null;
            isENV3Loaded = true;
            text = " ";
        }
```

*/*Viene creata un'anteprima per ogni immagine selezionata*/*

```
if (isMainImageLoaded) {
            GUI.DrawTexture (new Rect(wUnit*4,hUnit,wUnit*4,hUnit*2), mainImage);
        }
        if (isHS1Loaded) {
            GUI.DrawTexture (new Rect(wUnit*4,hUnit*3,wUnit*4,hUnit*2), HS1Image);
        }
        if (isHS2Loaded) {
            GUI.DrawTexture (new Rect(wUnit*4,hUnit*5,wUnit*4,hUnit*2), HS2Image);
        }
        if (isHS3Loaded) {
            GUI.DrawTexture (new Rect(wUnit*4,hUnit*7,wUnit*4,hUnit*2), HS3Image);
        }
        if (isColorsLoaded) {
          GUI.DrawTexture (new Rect(wUnit*12,hUnit,wUnit*4,hUnit*2), ColorsImage);
        }
        if (isENV1Loaded) {
          GUI.DrawTexture (new Rect(wUnit*12,hUnit*3,wUnit*4,hUnit*2), ENV1Image);
        }
        if (isENV2Loaded) {
          GUI.DrawTexture (new Rect(wUnit*12,hUnit*5,wUnit*4,hUnit*2), ENV2Image);
        }
        if (isENV3Loaded) {
          GUI.DrawTexture (new Rect(wUnit*12,hUnit*7,wUnit*4,hUnit*2), ENV3Image);
        }
```

*/*Alla pressione del bottone "OK" i colori inseriti vengono salvati in un altro script (ColorsList), richiamati i Material, inserite le Texture in essi, disattivato l'oggetto Gallery e riattivato l'oggetto GvrMain */*

```
if(buttonOK){
            colorList.GetComponent<ColorsList> ().hexCol1R = hexCol1R;
            colorList.GetComponent<ColorsList> ().hexCol1G = hexCol1G;
            colorList.GetComponent<ColorsList> ().hexCol1B = hexCol1B;
            colorList.GetComponent<ColorsList> ().hexCol2R = hexCol2R;
            colorList.GetComponent<ColorsList> ().hexCol2G = hexCol2G;
            colorList.GetComponent<ColorsList> ().hexCol2B = hexCol2B;
            colorList.GetComponent<ColorsList> ().hexCol3R = hexCol3R;
            colorList.GetComponent<ColorsList> ().hexCol3G = hexCol3G;
            colorList.GetComponent<ColorsList> ().hexCol3B = hexCol3B;
```

```
            mainMat = Resources.Load("main") as Material;
            hs1Mat = Resources.Load("hotspot1mat") as Material;
            hs2Mat = Resources.Load("hotspot2mat") as Material;
            hs3Mat = Resources.Load("hotspot3mat") as Material;
            colMat = Resources.Load("colors") as Material;
            env1Mat = Resources.Load("env1mat") as Material;
            env2Mat = Resources.Load("env2mat") as Material;
            env3Mat = Resources.Load("env3mat") as Material;

            SetTextureToCubemap(mainMat,mainImage);
            SetTextureToCubemap(hs1Mat,HS1Image);
            SetTextureToCubemap(hs2Mat,HS2Image);
            SetTextureToCubemap(hs3Mat,HS3Image);
            SetTextureToCubemap(colMat,ColorsImage);
            SetTextureToCubemap(env1Mat,ENV1Image);
            SetTextureToCubemap(env2Mat,ENV2Image);
            SetTextureToCubemap(env3Mat,ENV3Image);

            gvrmain.SetActive (true);
            gallery.SetActive (false);
        }

    }
```

*/*Quando viene selezionata un'immagine da galleria, viene creata una variabile di tipo WWW che ne contiene tutte le informazioni*/*

```
public void OnPhotoPick(string filePath){
        www = new WWW ("file://" + filePath);
    }
```

/ "SetTextureToCubemap" riceve in ingresso una Texture e un Material, che deve essere settato come Cubemap, e suddivide la Texture su ognuna delle 6 facce del Material*/*

```
private void SetTextureToCubemap (Material mat, Texture2D tex){

        Texture2D tex0 = CreateCubemapTexture (512, 0,tex);
        mat.SetTexture("_FrontTex", tex0);
        Texture2D tex1 = CreateCubemapTexture (512, 1,tex);
        mat.SetTexture("_BackTex", tex1);
        Texture2D tex2 = CreateCubemapTexture (512, 2,tex);
        mat.SetTexture("_LeftTex", tex2);
        Texture2D tex3 = CreateCubemapTexture (512, 3,tex);
        mat.SetTexture("_RightTex", tex3);
        Texture2D tex4 = CreateCubemapTexture (512, 4,tex);
        mat.SetTexture("_UpTex", tex4);
        Texture2D tex5 = CreateCubemapTexture (512, 5,tex);
```

```
        mat.SetTexture("_DownTex", tex5);
    }
```

/ *"CreateCubemapTexture" riceve in ingresso la dimensione e l'indice di una faccia del Material e una Texture. In base all'indice, cattura solo la parte di Texture che deve essere copiata su quella faccia, e restituisce una nuova Texture così costruita*/*

```
private Texture2D CreateCubemapTexture (int texSize, int faceIndex,
Texture2D srcTex) {

     Texture2D tex = new Texture2D(texSize, texSize, TextureFormat.RGB24, false);

        Vector3 [] vDirA = new Vector3[4];

        if (faceIndex == 0) {
            vDirA[0] = new Vector3(-1.0f, -1.0f, -1.0f);
            vDirA[1] = new Vector3( 1.0f, -1.0f, -1.0f);
            vDirA[2] = new Vector3(-1.0f,  1.0f, -1.0f);
            vDirA[3] = new Vector3( 1.0f,  1.0f, -1.0f);
        }
        if (faceIndex == 1) {
            vDirA[0] = new Vector3( 1.0f, -1.0f, 1.0f);
            vDirA[1] = new Vector3(-1.0f, -1.0f, 1.0f);
            vDirA[2] = new Vector3( 1.0f,  1.0f, 1.0f);
            vDirA[3] = new Vector3(-1.0f,  1.0f, 1.0f);
        }
        if (faceIndex == 2) {
            vDirA[0] = new Vector3( 1.0f, -1.0f, -1.0f);
            vDirA[1] = new Vector3( 1.0f, -1.0f,  1.0f);
            vDirA[2] = new Vector3( 1.0f,  1.0f, -1.0f);
            vDirA[3] = new Vector3( 1.0f,  1.0f,  1.0f);
        }
        if (faceIndex == 3) {
            vDirA[0] = new Vector3(-1.0f, -1.0f,  1.0f);
            vDirA[1] = new Vector3(-1.0f, -1.0f, -1.0f);
            vDirA[2] = new Vector3(-1.0f,  1.0f,  1.0f);
            vDirA[3] = new Vector3(-1.0f,  1.0f, -1.0f);
        }
        if (faceIndex == 4) {
            vDirA[0] = new Vector3(-1.0f,  1.0f, -1.0f);
            vDirA[1] = new Vector3( 1.0f,  1.0f, -1.0f);
            vDirA[2] = new Vector3(-1.0f,  1.0f,  1.0f);
            vDirA[3] = new Vector3( 1.0f,  1.0f,  1.0f);
        }
        if (faceIndex == 5) {
            vDirA[0] = new Vector3(-1.0f, -1.0f,  1.0f);
            vDirA[1] = new Vector3( 1.0f, -1.0f,  1.0f);
            vDirA[2] = new Vector3(-1.0f, -1.0f, -1.0f);
            vDirA[3] = new Vector3( 1.0f, -1.0f, -1.0f);
        }

        Vector3 rotDX1 = (vDirA[1] - vDirA[0]) / (float)texSize;
        Vector3 rotDX2 = (vDirA[3] - vDirA[2]) / (float)texSize;
```

```
        float dy = 1.0f / (float)texSize;
        float fy = 0.0f;

        Color [] cols = new Color[texSize];
        for (int y = 0; y < texSize; y++) {
            Vector3 xv1 = vDirA[0];
            Vector3 xv2 = vDirA[2];
            for (int x = 0; x < texSize; x++) {
                Vector3 v = ((xv2 - xv1) * fy) + xv1;
                v.Normalize();
                cols[x] = CalcProjectionSpherical(v, srcTex);
                xv1 += rotDX1;
                xv2 += rotDX2;
            }
            tex.SetPixels(0, y, texSize, 1, cols);
            fy += dy;
        }
        tex.wrapMode = TextureWrapMode.Clamp;
        tex.Apply();

        return tex;
    }
```

/ *"CalcProjectionSpherical" riceve in ingresso una Texture e un Vettore a 3 dimensioni, che indicano delle coordinate x, y, z; da queste calcola delle coordinate sferiche e restituisce il colore del pixel della Texture che si trova a quelle coordinate* */*

```
private Color CalcProjectionSpherical(Vector3 vDir, Texture2D srcTex) {
        float theta = Mathf.Atan2(vDir.z, vDir.x);
        float phi   = Mathf.Acos(vDir.y);

        theta += m_direction * Mathf.PI / 180.0f;
        while (theta < -Mathf.PI) theta += Mathf.PI + Mathf.PI;
        while (theta > Mathf.PI) theta -= Mathf.PI + Mathf.PI;

        float dx = theta / Mathf.PI;
        float dy = phi / Mathf.PI;

        dx = dx * 0.5f + 0.5f;
        int px = (int)(dx * (float)srcTex.width);
        if (px < 0) px = 0;
        if (px >= srcTex.width) px = srcTex.width - 1;
        int py = (int)(dy * (float)srcTex.height);
        if (py < 0) py = 0;
        if (py >= srcTex.height) py = srcTex.height - 1;

        Color col = srcTex.GetPixel(px, srcTex.height - py - 1);
        return col;
    }
}
```

Lo script (*ColorsList*) dove vengono salvati i valori RGB dei colori scelti è rappresentato nell'algoritmo 4.2 ed è formato solamente da variabili pubbliche che vengono lette in fase di riconoscimento del colore. Grazie a questo script si possono leggere i valori inseriti anche dopo che l'oggetto *gallery* viene disattivato.

Algoritmo 4.2 ColorsList - salvataggio colori inseriti

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ColorsList : MonoBehaviour {

    public string hexCol1R;
    public string hexCol1G;
    public string hexCol1B;
    public string hexCol2R;
    public string hexCol2G;
    public string hexCol2B;
    public string hexCol3R;
    public string hexCol3G;
    public string hexCol3B;

    void Start () {

    }

    void Update () {

    }
}
```

I plugin java utilizzati per la comunicazione tra l'ambiente Unity e il sistema operativo Android sono riportati negli algoritmi 4.3 e 4.4.
UnityBinder viene richiamato dallo script GalleryUse e permette di attivare il secondo plugin, Gallery, il quale, a seguito della scelta di un'immagine dalla galleria Android, invia il dato all'applicazione.

Algoritmo 4.3 Plugin UnityBinder - collegamento tra Unity e Android

```
import android.app.Activity;
import android.content.Intent;

public class UnityBinder {

public static void OpenGallery(Activity activity){

        Intent intent = new Intent(activity,Gallery.class);
        activity.startActivity(intent);
  }
}
```

Algoritmo 4.4 Plugin Gallery - invio immagine scelta all'applicazione

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.net.Uri;
import android.provider.MediaStore;
import android.database.*;
import com.unity3d.player.*;


public class Gallery extends Activity {
        int PHOTO_GALLERY=1;

/*onCreate apre la galleria Android*/

        @Override
        protected void onCreate(Bundle bundle){

super.onCreate(bundle);
Intent intent=new Intent(Intent.ACTION_PICK, android.provider.
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
startActivityForResult(intent, PHOTO_GALLERY);
        }

/*onActivityResult invia l'immagine all'applicazione */

        @Override
        protected void onActivityResult(int requestCode,int
resultCode, Intent data){
```

```
if(resultCode==RESULT_OK &&requestCode==PHOTO_GALLERY &&
data!=null){

Uri uri=data.getData();
String[] fileColumn={MediaStore.Images.Media.DATA};
Cursor cursor=getContentResolver().query(uri, fileColumn, null, null,
null);
cursor.moveToFirst();
int columnIndex=cursor.getColumnIndex(fileColumn[0]);
String photoPath=cursor.getString(columnIndex);

                        // Gameobject name   method name    argument
UnityPlayer.UnitySendMessage("Gallery",    "OnPhotoPick", photoPath);
Gallery.this.finish();

              }
       }
}
```

Inizialmente la schermata di selezione presenta i bottoni e le aree dove indicare i valori RGB dei colori scelti.

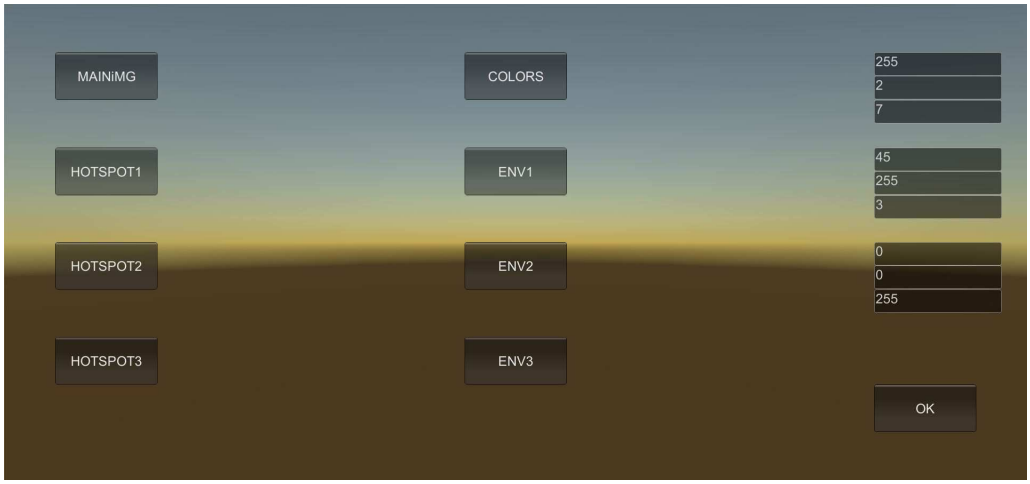


Figura 4.15: Schermata iniziale

Dopo la selezione da galleria, compariranno di fianco a ogni bottone le anteprime delle immagini scelte.

Figura 4.16: schermata iniziale con anteprime

Premendo il bottone "OK" si passa alla visualizzazione stereoscopica dell'immagine scelta come ambiente principale.

Figura 4.17: Visione stereoscopica dell'ambiente principale

Da questo momento si possono osservare le ambientazioni scelte a 360 gradi, tramite l'ausilio di un visore VR.
La visione stereoscopica e il tracciamento del movimento della testa sono gestiti da script già inseriti nell'impalcatura dell'oggetto GvrMain.
In particolare l'oggetto Head si occupa della gestione del movimento della testa. Di conseguenza qualsiasi oggetto di tipo Camera, inserito come "figlio" di Head, seguirà lo stesso movimento, e questo è fondamentale per il meccanismo implementato.
Infatti, se la Main Camera sta osservando un pixel dell'immagine principale posizionato a certe coordinate, la Camera 2 osserva nello stesso momento un pixel dell'immagine dei colori (che definisce gli hotspot) alle stesse coordinate, e questo permette di mantenere la corrispondenza tra ambiente e hotspot.
Riguardo al meccanismo, innanzitutto lo script inserito nell'oggetto Camera2, chiamato CheckColCam2 e descritto dall'algoritmo 4.5, si occupa di catturare il colore del pixel osservato dalla camera.

Algoritmo 4.5 CheckColCam2 - lettura del colore associato al hotspot

```
using UnityEngine;
using System.Collections;

public class CheckColCam2 : MonoBehaviour {
    private Texture2D tex;
    public Color center;
    public static bool hex;
    private RenderTexture rt;
    public float r;
    public float g;
    public float b;

    void Start () {
        tex = new Texture2D(1, 1, TextureFormat.RGB24, false);
        rt = this.GetComponent<Camera> ().targetTexture;
    }

    void Update () {
        this.GetComponent<Camera> ().Render ();
        RenderTexture.active = rt;
```

```
        tex.ReadPixels(new Rect(Screen.width / 2f, Screen.height / 2f, 1,
 1), 0, 0);
        tex.Apply();
        center = tex.GetPixel(0,0);
    }
}
```

Quando Camera 2 legge uno dei colori scelti come hotspot, lo script ChangeTexture, inserito in Main Camera, si occupa di modificare l'immagine della Skybox di questa camera, selezionando il Material (quindi la Texture contenuta al suo interno) associato al hotspot osservato.
Per sapere quale hotspot è osservato, viene effettuato un controllo di uguaglianza tra i valori RGB inseriti nella schermata iniziale e quelli del colore appena catturato.

Algoritmo 4.6 ChangeTexture - modifica Skybox in base al colore letto

```
using UnityEngine;
using System.Collections;

public class ChangeTexture : MonoBehaviour {

    private GameObject cam, colorsVal;
    private Color checkedCol;
    public float r,g,b,h1R,h1G,h1B,h2R,h2G,h2B,h3R,h3G,h3B;
    public float int1R,int1G,int1B,int2R,int2G,int2B,int3R,int3G,int3B;
    private Material main, hotspot1, hotspot2, hotspot3;

    void Start () {
        cam = GameObject.FindGameObjectWithTag ("Camera2");
        colorsVal = GameObject.FindGameObjectWithTag ("colorsval");
       h1R = float.Parse(colorsVal.GetComponent<ColorsList> ().hexCol1R);
        int1R = Mathf.Round (h1R / 255);
       h1G = float.Parse(colorsVal.GetComponent<ColorsList> ().hexCol1G);
        int1G = Mathf.Round (h1G / 255);
       h1B = float.Parse(colorsVal.GetComponent<ColorsList> ().hexCol1B);
        int1B = Mathf.Round (h1B / 255);
       h2R = float.Parse(colorsVal.GetComponent<ColorsList> ().hexCol2R);
        int2R = Mathf.Round (h2R / 255);
       h2G = float.Parse(colorsVal.GetComponent<ColorsList> ().hexCol2G);
```

```
        int2G = Mathf.Round (h2G / 255);
       h2B = float.Parse(colorsVal.GetComponent<ColorsList> ().hexCol2B);
        int2B = Mathf.Round (h2B / 255);
       h3R = float.Parse(colorsVal.GetComponent<ColorsList> ().hexCol3R);
        int3R = Mathf.Round (h3R / 255);
       h3G = float.Parse(colorsVal.GetComponent<ColorsList> ().hexCol3G);
        int3G = Mathf.Round (h3G / 255);
       h3B = float.Parse(colorsVal.GetComponent<ColorsList> ().hexCol3B);
        int3B = Mathf.Round (h3B / 255);
        main = Resources.Load("main") as Material;
        hotspot1 = Resources.Load("hotspot1mat") as Material;
        hotspot2 = Resources.Load("hotspot2mat") as Material;
        hotspot3 = Resources.Load("hotspot3mat") as Material; }

    void Update () {
        checkedCol = cam.GetComponent<CheckColCam2> ().center;
        r = Mathf.Round (checkedCol.r);
        g = Mathf.Round (checkedCol.g);
        b = Mathf.Round (checkedCol.b);

        if(    Camera.main.GetComponent<Skybox> ().material == main
            || Camera.main.GetComponent<Skybox> ().material == hotspot1
            || Camera.main.GetComponent<Skybox> ().material == hotspot2
            || Camera.main.GetComponent<Skybox> ().material == hotspot3)
        {
            if (r == int1R && g == int1G && b == int1B)
            {
                Camera.main.GetComponent<Skybox> ().material = hotspot1;
            }
            else if(r == int2R && g == int2G && b == int2B)
            {
                Camera.main.GetComponent<Skybox> ().material = hotspot2;
            }
            else if(r == int3R && g == int3G && b == int3B)
            {
                Camera.main.GetComponent<Skybox> ().material = hotspot3;
            }
            else
            {
                Camera.main.GetComponent<Skybox> ().material = main;
            }
        }
    }
}
```

Per esempio, quando Camera2 legge il colore rosso, nella Skybox di Main Camera compare l'immagine legata a quel hotspot.

Figura 4.18: Visione stereoscopica dell'hotspot

Main Camera Left e Main Camera Right sono le due camere, figlie di Main Camera, che gestiscono la parte sinistra e la parte destra della visione stereoscopica.
In questi due oggetti viene creato in automatico un componente Skybox, quando Main Camera è attiva, e lo script TexEyeChange, inserito in essa, assegna alle Skybox delle camere figlie, in ogni momento, il suo stesso Material, e questo permette l'effettiva visione dell'ambiente da parte dell'utente.

Algoritmo 4.7 TexEyeChange - modifica Skybox di Main Camera Left e Main Camera Right

```
using UnityEngine;
using System.Collections;

public class TexEyeChange : MonoBehaviour {

    private GameObject[] cam;
    private int i;
    private Material mat;
    public int l;

    void Start () {
        cam = GameObject.FindGameObjectsWithTag("Eye");
        l = cam.Length;
```

```
    }

    void Update () {
        for(i = 0; i<cam.Length ; i++){
            if(Camera.main.GetComponent<Skybox> ().material != cam[i].Get
Component<Skybox> ().material){
                cam [i].GetComponent<Skybox> ().material = Camera.main.Ge
tComponent<Skybox> ().material;
            }
        }
    }
}
```

Infine, quando viene osservato un hotspot, se l'utente preme il bottone del Google Cardboard (o "tocca" lo schermo in qualsiasi modo), visualizzerà l'immagine dell'ambiente legato a quel hotspot.

Figura 4.19: Visione stereoscopica di un ambiente

A quel punto, a seguito di un altro tocco, si tornerà all'ambiente principale.
Il passaggio da un ambiente all'altro è gestito dallo script GoTo, inserito anch'esso in Main Camera.

Algoritmo 4.8 GoTo - passaggio tra gli ambienti

```
using UnityEngine;
using System.Collections;

public class GoTo: MonoBehaviour {

    private Material hotspot1, hotspot2, hotspot3, main;
    private Material env1, env2, env3, mat;
    private Camera cam;
    private GameObject head;
    private float zero = 0;
    private Vector3 vec = new Vector3(0f , 0f , 0f);

    void Start () {
        cam = Camera.main;
        head = GameObject.FindGameObjectWithTag ("Head");
        hotspot1 = Resources.Load("hotspot1mat") as Material;
        hotspot2 = Resources.Load("hotspot2mat") as Material;
        hotspot3 = Resources.Load("hotspot3mat") as Material;
        main = Resources.Load("main") as Material;
        env1 = Resources.Load("env1mat") as Material;
        env2 = Resources.Load("env2mat") as Material;
        env3 = Resources.Load("env3mat") as Material;
    }

    void Update () {
        if(Input.GetMouseButtonDown(0)){

            mat = cam.GetComponent<Skybox> ().material;

            if (mat == hotspot1) {
                cam.GetComponent<Skybox> ().material = env1;
            } else if (mat == hotspot2) {
                cam.GetComponent<Skybox> ().material = env2;
            } else if (mat == hotspot3) {
                cam.GetComponent<Skybox> ().material = env3;
            } else if (mat == env1 || mat == env2 || mat == env3) {
                cam.GetComponent<Skybox> ().material = main;
            }
        }
    }
}
```

Capitolo 5

Conclusioni e sviluppi futuri

L'algoritmo presentato in questo lavoro di tesi permette il riconoscimento di zone sensibili in applicazioni di realtà virtuale, senza dover evidenziarne la posizione in ogni momento.
Questo permette la programmazione di applicazioni di vario tipo, ad esempio un catalogo VR di un negozio dove osservando gli oggetti venduti possono comparire le caratteristiche e il prezzo, o un'applicazione che permetta di muoversi tra le stanze di un edificio per un tour guidato.
Inoltre l'applicazione mostrata in questa tesi può essere riutilizzata per qualsiasi scopo, una volta create le immagini per gli hotspot e gli ambienti.
Tutto questo è, comunque, un punto di partenza da cui diversi sviluppi possono essere fatti.
I miglioramenti possono riguardare, per esempio, la possibilità di aggiungere un numero a piacere di hotspot nell'applicazione (al momento si possono avere al massimo 3 zone sensibili), o l'utilizzo di video e ambienti dinamici, oltre alle semplici immagini, per poter creare applicazioni sempre più interattive o veri e propri videogiochi.

Bibliografia

[1] https://en.wikipedia.org/wiki/Virtual_reality

[2] http://www.dictionary.com/browse/virtual--reality

[3] http://www.bibliotecagalattica.com/racconti/occhiali_di_pigmalione.html

[4] https://en.wikipedia.org/wiki/Sensorama

[5] Kock, N. (2008). *"E-collaboration and e-commerce in virtual worlds: The potential of Second Life and World of Warcraft"*. International Journal of e-Collaboration. 4 (3):1–13.

[6] https://en.wikipedia.org/wiki/Aspen_Movie_Map

[7] Delaney, Ben. *Sex, Drugs and Tessellation: The Truth About Virtual Reality, as Revealed in the Pages of CyberEdge Journal*. p. 274

[8] Barlow, John Perry (1990). *"Being in Nothingness: Virtual Reality and the Pioneers of Cyberspace"*. Electronic Frontiers Foundation.

[9] Engler, Craig E. (November 1992). *"Affordable VR by 1994"*. Computer Gaming World. p. 80.

[10] https://vr.google.com/cardboard/

[11] http://www.videogiochi.com/news/2015/05/google-cardboard-il-nuovo-modello-funziona-anche-con-iphone/

[12] Pierce, David (May 28, 2015). *"Google Cardboard is VR's Gateway Drug"*. Wired. Retrieved June 17, 2015

[13] https://developers.google.com/vr/cardboard/overview

[14] https://developers.google.com/vr/concepts/vrview

[15] https://developers.google.com/vr/concepts/vrview-web

[16] https://support.google.com/edu/expeditions/answer/6335093?hl=en&ref_topic=6334250

[17] http://www.amnh.org/explore/news-blogs/news-posts/museum-joins-with-google-to-launch-virtual-reality-visits/

[18] http://googlevr.github.io/vrview/examples/hotspots/index.html

[19] https://en.wikipedia.org/wiki/Unity_(game_engine)#cite_note-7

[20] https://unity3d.com/unity/multiplatform/

[21] http://www.adobe.com/it/products/photoshop.html

[22] https://developers.google.com/vr/unity/download