EsperiaVR

Una simulazione in realtà virtuale dell'ambiente scolastico

Sangalli Matteo

ITIS P. Paleocapa - Esame di Stato 2017

Premessa

A conclusione del mio percorso di studi ho voluto cimentarmi in un ambiente nuovo e molto distaccato da quello che ci viene proposto a scuola. Il seguente documento tratterà delle caratteristiche della realtà virtuale, di una panoramica dell'ambiente di sviluppo Unity, delle motivazioni delle scelte seguite durante l'esecuzione del progetto e delle fasi principali che hanno composto lo svolgimento del lavoro.

Obiettivo

Lo scopo del progetto è sviluppare un'applicazione per dispositivi mobili (Android) che, insieme ad un visore Google Cardboard (o equivalente), possa simulare l'ambiente scolastico.

Sommario

Introduzione

Realtà virtuale

Virtual reality (VR), which can be referred to as immersive multimedia or computer-simulated reality, replicates an environment that simulates a physical presence in places in the real world or an imagined world, allowing the user to interact in that world. – Wikipedia

Per realtà virtuale s'intende una simulazione interamente creata da un computer nella quale ci si può immergere solitamente tramite un apposito visore. L'utente può interagire in modo più o meno realistico a seconda del controller a sua disposizione.

Questa tecnologia ha visto un'enorme crescita a partire dal 2012 con l'inizio dello sviluppo di Oculus Rift che è stato il visore più famoso per molto tempo e che ha visto la nascita di nuovi concorrenti come HTC Vive e Playstation VR o i più economici Google Cardboard e Samsung Gear VR.

La realtà virtuale viene utilizzata soprattutto nel settore dell'intrattenimento, in particolare nei videogiochi e sempre più frequentemente in ambiti culturali tramite simulazioni 3D di luoghi di interesse artistico. Altri settori che stanno investendo in questa tecnologia sono la sanità e l'industria.

Il settore videoludico in cui l'applicazione della realtà virtuale è ottima riguarda i simulatori di guida o di volo.

Realtà aumentata

La realtà aumentata viene spesso confusa con la realtà virtuale ma la divisione tra le due è netta. Nella realtà aumentata, infatti, l'utente vede il mondo reale accompagnato da oggetti o informazioni virtuali sparsi nell'ambiente.

Nella realtà aumentata rientrano spesso due concetti differenti ma ugualmente corretti:

- Il primo consiste nell'utilizzare dispositivi che forniscono all'utente più informazioni riguardo l'ambiente circostante senza immergersi completamente. Un esempio è stato Pokémon Go che ha spopolato nell'estate 2016 e che simula la presenza di Pokémon e di altri punti d'interesse nell'ambiente reale.
- Il secondo consiste invece nell'indossare dispositivi che simulano la presenza di oggetti virtuali nel mondo reale come se fossero degli ologrammi permettendo all'utente ad esempio di visualizzare uno schermo virtuale fissato ad un muro nella propria casa oppure di avere sempre sott'occhio le indicazioni stradali mentre si guida. Nel cinema un esempio di realtà aumentata si ha nella visione che ha Iron Man del mondo attraverso il suo casco. Esistono già dei dispositivi che offrono questo tipo di servizi come Microsoft Hololens o Google Glass (progetto ormai abbandonato da Google).

Dispositivi attualmente in commercio e in sviluppo

Dopo il 2012 si sono sviluppati diversi dispositivi in grado di fornire esperienze in realtà virtuale e aumentata.

La fascia alta comprende tre sistemi principali che sono accomunati dal fatto di avere lo schermo e parte dei sensori incorporati nel visore e tutti hanno bisogno di un sistema esterno che esegua il software:

- Oculus Rift: Nato nel 2012, in parte grazie ad una campagna di raccolta fondi su Kickstarter, è stato il precursore della realtà virtuale nel campo dell'intrattenimento ad alti livelli. Esso necessità di un PC molto potente (i requisiti consigliati parlano di una scheda grafica di potenza pari o superiore a una NVIDIA GTX 970) e viene utilizzato spesso nel campo videoludico. Nel 2014 Oculus è stata acquisita da Facebook e nel 2016 è stata distribuita la prima versione *consumer* di *Rift.*

Figura 1 - La versione *consumer* di Oculus Rift

- HTC Vive: Sviluppato da HTC in collaborazione con Valve e annunciato nel 2015, questo sistema si propone di portare la realtà virtuale ad un altro livello permettendo all'utente di camminare e muovere le braccia nello spazio reale tramite dei sensori che mappano la stanza. Esso viene utilizzato soprattutto nei videogiochi in quanto dispone di una sensoristica molto avanzata. I requisiti consigliati sono pressoché gli stessi di Oculus Rift.
- Playstation VR: Playstation rilascia il suo visore nell'ottobre 2016 con lo scopo di proporre agli utenti un sistema a realtà virtuale più economico da accompagnare ad una Playstation 4. Esso dispone di molta meno potenza rispetto a Rift e Vive ma l'ecosistema Playstation, la semplicità di utilizzo e un parco titoli più mirato ne garantiscono un miglior successo commerciale rispetto ai due diretti concorrenti.

La realtà virtuale di fascia alta non è per tutti in quanto il prezzo dei dispositivi è ancora elevato; per questo sono state ideate soluzioni più economiche che forniscono comunque un'esperienza accettabile. Questi sistemi sono accomunati dal fatto che uno smartphone assolve la funzione di unità di elaborazione e di display. I visori principali sono tre:

- Google Cardboard: Il visore senza dubbio più diffuso e famoso è il Cardboard ideato da Google che tramite del cartone, due lenti apposite e uno smartphone riesce a dare un'esperienza ottima ad un costo irrisorio. Sono supportati gran parte dei dispositivi con una versione di Android superiore alla 4.4 oltre agli iPhone successivi ad iPhone 5. Esso rappresenta la migliore scelta in rapporto qualità/prezzo.

Figura 2 - Google Cardboard

- Google Daydream: Dopo il successo di Cardboard, Google ha deciso di creare un dispositivo di fascia più alta, con materiali migliori e un controller dedicato che ricorda il Wiimote della famosa Nintendo Wii. Il primo smartphone ad essere compatibile con Daydream è stato il Pixel di Google anche se ora il bacino di dispositivi compatibili si sta ampliando.
- Samsung Gear VR: Samsung propose come diretto concorrente di Cardboard (inizialmente) il suo visore Gear VR. Grazie alla collaborazione con Oculus questo visore offre molti più servizi oltre ad una qualità di costruzione e un comfort migliori. Questi benefici però fanno lievitare un po' il

prezzo. Purtroppo gli unici dispositivi compatibili sono gli ultimi smartphone di Samsung, ovvero quelli successivi al Galaxy S6.

La realtà aumentata tramite l'utilizzo di dispositivi *wearable* è di fatto ancora in via sperimentale anche se si sono visti già due dispositivi in questo panorama:

- Google Glass: La divisione Google X del colosso di Mountain View iniziò il progetto nel 2011. Questi occhiali hanno lo scopo di visualizzare delle informazioni in sovrapposizione al mondo reale. L'utente vede le informazioni grazie ad un piccolo display di fronte all'occhio destro e può interagire con il dispositivo tramite un touchpad sugli occhiali oppure con la voce. Il prodotto venne rilasciato in versione beta nel 2013 ma la produzione fu poi interrotta nel 2015.
- Microsoft Hololens: Microsoft annunciò il suo "casco" per la realtà aumentata nel 2016. Esso dispone di uno schermo trasparente di fronte agli occhi che permette all'utente di visualizzare ologrammi nel mondo reale. Dispone di una tecnologia di mappatura dell'ambiente in cui ci si trova in modo da poter affiggere elementi virtuali nel modo reale. Il progetto è ancora in sviluppo, anche se è già possibile acquistare il dispositivo.

Figura 3 – Google Glass

Figura 4 - Microsoft Hololens

Prima di cominciare

Preparazione degli strumenti

Per creare questo progetto ho innanzitutto scelto Google Cardboard come visore perché è il più economico e quindi il più accessibile sia da me sviluppatore, che dagli utenti.

Inoltre sono stati necessari diversi software ognuno dei quali coprirà un ruolo specifico:

- Unity: motore grafico
- Blender: modellazione 3D
- Gimp: editor di immagini
- Microsoft Visual Studio 2015: IDE per scripting in C#
- Android Studio: IDE per sviluppo app per Android
- GoogleVR SDK for Unity

La mia scelta è ricaduta su questi software perché sono tutti gratuiti per uso personale e sono tutti compatibili tra di essi. Per ogni software scelto ci sono diverse alternative sia gratuite che a pagamento.

Questi software sono costantemente in sviluppo (soprattutto Unity, gli SDK di Android e il pacchetto GoogleVR SDK for Unity) perciò una volta scelta una versione, l'ho mantenuta da inizio a fine progetto per evitare problemi di compatibilità tra i diversi software. Nello specifico ho utilizzato le seguenti versioni dei software: Unity 5.5.0f3, Blender 2.78.3, Gimp 2.8.16, Android Studio 2.3.1, l'SDK di Android 7.0.1 e GoogleVR SDK 1.0.3

Unity

Unity è un motore grafico ovvero un software che permette di creare videogiochi ma anche simulazioni 3D di ambienti. Esso fornisce un editor grafico, un motore di rendering (2D e 3D), un motore fisico e altri elementi che permettono di gestire audio, script, animazioni, collisioni tra oggetti, comunicazioni in rete, gestione dei thread ecc.

Unity ha diversi punti di forza che lo rendono un software estremamente flessibile:

- È gratuito per uso personale.
- È relativamente semplice da imparare e permette fin da subito di ottenere dei risultati.
- È disponibile per Windows, macOS e Linux.
- Permette di pubblicare l'applicazione finale per più di 20 piattaforme differenti tra cui PC (Windows, macOS, Linux), console (PS3, PS4, XBOX 360, XBOX ONE, Wii..), dispositivi mobili (Android , iOS, Windows Phone, Fire OS) o altri dispositivi come Smart TV o Smartwatch (Tizen). È richiesta una licenza per pubblicare su piattaforme specifiche come ad esempio Playstation.
- Riesce a dialogare con molti altri software nativamente (ad esempio può importare molti tipi di file differenti che contengono modelli 3D). Inoltre dispone di integrazione con altri software come Visual Studio o gli SDK di Android.
- È attiva una community molto ampia di sviluppatori indipendenti e dispone di uno store (Asset Store) dal quale comprare plugin, modelli 3D o script.

Unity è, di fatto, il cuore del progetto perché tutto ciò che viene creato negli altri software viene importato, assemblato ed infine esportato sotto forma di prodotto utilizzabile dall'utente.

Un'alternativa gratuita a Unity è Unreal Engine che è un motore grafico equivalente ma è più complesso da imparare e non è integrato con i GoogleVR SDK.

Blender

Unity permette di creare dei modelli 3D ma sono solo primitive, ovvero cubi, sfere, piani ecc. Per creare modelli più articolati è necessario un software apposito.

Blender è un software open-source che permette di modellare, animare e renderizzare oggetti tridimensionali.

Per creare un oggetto 3D si parte da delle primitive e tramite semplici operazioni come l'estrusione o la modifica della posizione di vertici, spigoli e facce si può creare ogni tipo di oggetto. Grazie ad un motore di rendering integrato e alla possibilità di impostare la luce e la camera è possibile renderizzare una composizione di oggetti come una singola immagine.

Ci sono molte alternative a Blender gratuite o a pagamento come Maya, Cinema 4D o 3D Studio Max.

Gimp

Gimp è un software open-source di editing di immagini. È l'alternativa per eccellenza a Photoshop anche se è molto più grezzo e meno potente.

Gimp è necessario per creare le texture da apporre sugli oggetti 3D e per creare altri file grafici.

Microsoft Visual Studio 2015

In Unity gli script possono essere creati sia in C# sia in Javascript. La scelta dell'editor di testo è soggettiva. Personalmente ho scelto Visual Studio perché, programmando in C#, l'aiuto fornito dall'ambiente in fase di scrittura è notevole.

Le alternative sono diverse: dall'editor integrato in Unity a Notepad++.

Android Studio

Unity non permette di compilare nativamente per ogni piattaforma ma è richiesta spesso una licenza o degli strumenti aggiuntivi in più. Per quanto riguarda Android, Unity richiede gli SDK di una versione di Android.

Per scaricare gli SDK di Android, Google fornisce un software chiamato "SDK Manager" che permette di scaricare e gestire tutti gli strumenti di sviluppo per creare un'app Android. Purtroppo questo software è disponibile solo insieme all'installazione di tutto l'ambiente Android Studio.

GoogleVR SDK for Unity

Google fornisce degli strumenti creati appositamente per creare app in realtà virtuale per Unity. Questo pacchetto contiene dei componenti già pronti che semplificano il lavoro dello sviluppatore siccome coprono già la realizzazione della camera VR, le interazioni tramite un puntatore virtuale e la gestione dell'audio.

Come funziona Unity

L'Interfaccia

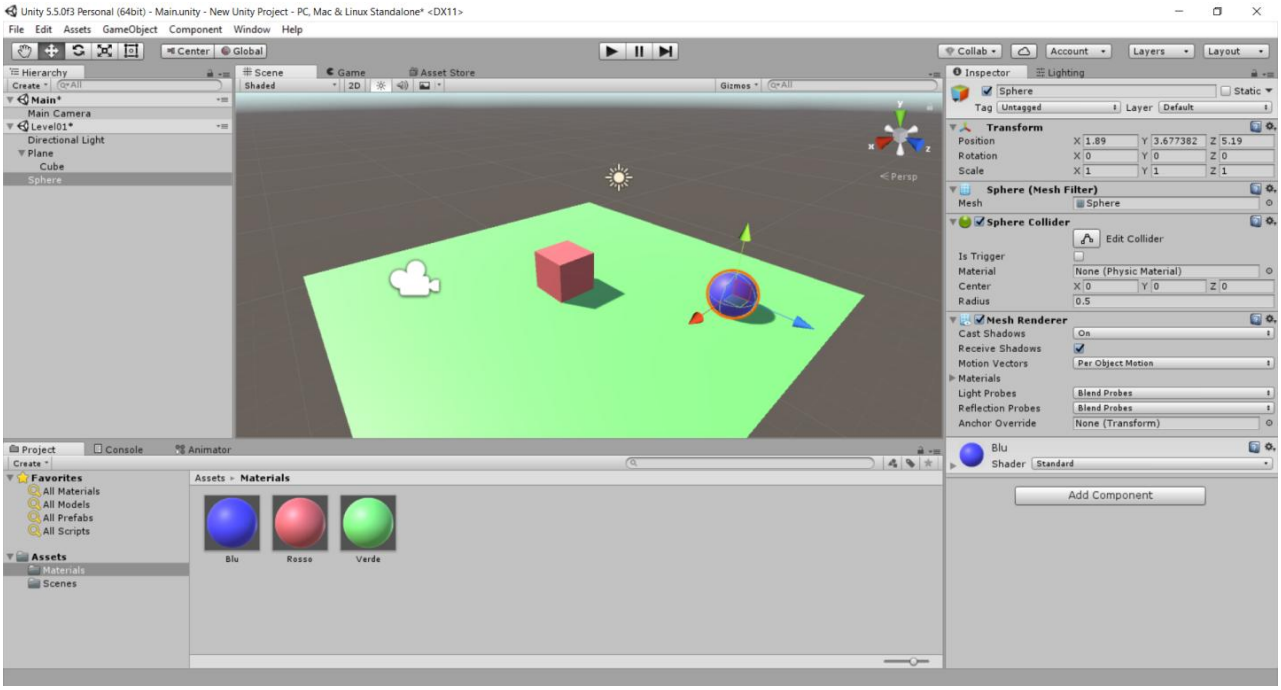


Figura 5 - La GUI di Unity

Unity si presenta con un'interfaccia abbastanza intuitiva ed estremamente personalizzabile infatti ogni scheda può essere agganciata in molte posizioni differenti nella finestra principale.

Nella schermata si possono identificare diverse schede fondamentali:

Hierarchy

Sulla sinistra la scheda *Hierarchy* mostra tutti gli oggetti presenti attualmente nella scena generale. Gli oggetti sono racchiusi in scene. Una singola scena si può identificare ad esempio come un singolo livello di un videogioco. Nella scena gli oggetti sono organizzati in una struttura ad albero ovvero ognuno di essi può dipendere da un altro (come si vede in figura, *Cube* dipende da *Plane*).

Scene

Nel centro, la scheda *Scene* mostra in modo visuale gli oggetti presenti nella Hierarchy. In questa scheda ci si può muovere nel mondo tridimensionale principalmente per posizionare, ruotare e ridimensionare gli oggetti oltre che per vedere un primo risultato generale della composizione. La camera può essere ruotata liberamente o con il mouse o, in modo più preciso, cliccando sul componente nell'angolo superiore destro della scheda. Esso permette di ruotare la telecamera lungo uno degli assi x, y, o z oppure di attivare e disattivare la visuale ortogonale premendo al centro di esso. La visuale ortogonale è fondamentale quando si vogliono disporre gli oggetti.

Game

Sempre nel centro (nella figura non è visibile perché coperta dalla scheda Scene), la scheda *Game* mostra la scena dal punto di vista della camera principale presente nella scena. Di fatto mostra come appare il gioco quando viene eseguito. Infatti, quando viene premuto il pulsante *Play,* Unity passerà immediatamente dalla scheda Scene alla scheda Game. In questo modo si può testare l'ambiente senza

doverlo compilare per la piattaforma di destinazione. Nella parte superiore destra della scheda è presente il pulsante *Stats* che durante l'esecuzione del gioco permette di visualizzare diversi parametri utili per l'ottimizzazione.

Inspector

L'*Inspector* è una delle schede più importanti infatti essa mostra tutte le proprietà e i componenti appartenenti ad un oggetto presente nella scena. Ogni tipo di oggetto ha dei componenti specifici che gli possono essere attribuiti.

Project

La scheda Project non è altro che un gestore dei file del progetto. Esso visualizza tutti i file riconosciuti da Unity come "Esplora risorse" lo fa per Windows. I file possono essere importati nel progetto tramite un "drag and drop" in questa scheda. È molto importante suddividere coerentemente i file in cartelle in base alla tipologia in modo da trovare rapidamente i file anche durante l'avanzamento del progetto. Tutti i file visualizzati in questa schermata sono presenti nella cartella Assets del progetto.

Console

La scheda Console mostra gli errori avvenuti durante l'esecuzione degli script oppure può essere utilizzata come strumento di debugging. Essa assume lo stesso ruolo delle console presenti in un qualsiasi altro ambiente di sviluppo.

Altre schede

Quelle sopra elencate sono solo le principali schede offerte da Unity ma accedendo al menù *Windows* nella barra superiore sono presenti altre schede utili come *Lighting* che permette di modificare le proprietà dell'illuminazione o e schede *Animator* e *Animation* che permettono di gestire e creare le animazioni per gli oggetti.

GameObjects, componenti e tipi di file

GameObjects

Come spiegato in precedenza, nella *Hierarchy* sono presenti tutti gli oggetti presenti nella scena. Unity li identifica come *GameObjects* e si possono trovare nel menu *GameObject* della barra superiore. Esistono 3 categorie principali di oggetti: *3D Object*, *Light* e *Camera*.

Un *3D Object* è un qualsiasi modello 3D, dalle primitive fornite da Unity come *Plane*, *Cube*, *Sphere* ecc. ai modelli 3D personalizzati importati dall'utente nel progetto e creati tramite un programma di modellazione 3D.

Sotto la categoria *Light*, Unity identifica 4 tipi diversi di illuminazione che simulano una fonte di luce differente:

- *Directional Light* simula la luce del sole o della luna.
- *Point Light* è una fonte di luce che illumina in tutte le direzioni intorno a se e può essere utilizzata per creare lampioni o lampadine ma anche esplosioni.
- *Spot Light* emette la luce solo in una direzione con uno specifico grado di ampiezza. Può essere utilizzata per creare torce o fari delle auto.
- *Area Light* può essere accomunata ad una *Point Light* ma la fonte di luce non è un singolo punto ma un rettangolo ai cui lati viene emessa luce.

L'ultimo componente fondamentale è la *Camera*. Essa rappresenta la visuale che l'utente ha sul mondo virtuale. In caso di visuale in prima persona, spostando la *Camera* si da l'impressione dello spostamento effettivo dell'utente.

Oltre a questi 3 tipi principali esistono anche altri oggetti che permettono di gestire l'audio, l'interfaccia utente (UI), terreni, alberi, vento, o gli sprite per la grafica 2D.

Componenti

Per ogni oggetto aggiunto nella scena si possono vedere e modificare le sue proprietà tramite l'*Inspector.* Oltre alle proprietà base come il nome dell'oggetto, il suo stato (attivo/non attivo) e la sua posizione nello spazio, nell'*Inspector* si possono aggiungere diversi componenti che andranno a caratterizzare sempre più un certo oggetto. Alcuni componenti sono già parte di tipi particolari di oggetti, come la camera o le luci, ma possono comunque essere applicati ad altri tipi di oggetto. Quelli degni di nota sono:

Colliders

Un *Collider* può essere pensato come un rivestimento che copre un oggetto. Il *Collider* ha un ruolo fondamentale ovvero quello di rilevare le collisioni tra gli oggetti. Sarà poi lo sviluppatore a specificare il comportamento che dovranno avere i due oggetti che hanno colliso. Se il *Collider* ha il ruolo di *Trigger*, esso scatenerà un evento che potrà essere utilizzato ad esempio per eseguire uno script. Esistono diversi tipi di collider a seconda del tipo di modello che devono "rivestire". I più comuni sono:

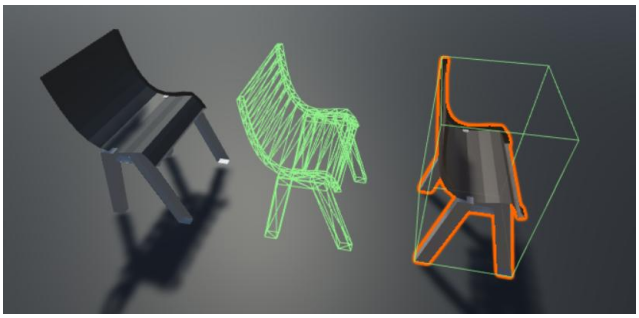


Figura 6 – La differenza del numero di poligoni tra un *Mesh Collider* (al centro) e un *Box Collider* (a destra) è notevole

- Il *Box Collider* ha la forma di un parallelepipedo ed è quello che richiede una minore capacità di elaborazione siccome ha un ridotto numero di poligoni e vertici (vedi paragrafo Ottimizzazione) ed è utile se applicato ad oggetti interattivi o ad oggetti complessi che non richiedono una precisione elevata nelle collisioni.
- Il *Mesh Collider* invece è completamente l'opposto perché segue perfettamente la forma dell'oggetto e richiede quindi una maggiore capacità di elaborazione.
- Il *Capsule Collider*, come dice il nome, ha la forma di una capsula e viene attribuito normalmente ad un personaggio o al giocatore stesso.

Renderer

Nella maggior parte dei casi un oggetto deve essere visualizzato e non deve essere invisibile per cui è necessario un componente che attribuisca alla geometria dell'oggetto un colore. Il *Renderer* ha proprio questo scopo: dati dei *Materials* (vedi paragrafo Tipi di file) esso calcola le luci e le ombre. Tramite due proprietà *Cast Shadows* e *Receive Shadows* è possibile decidere se l'oggetto debba rispettivamente fornire e ricevere ombre. Il *Renderer* più utilizzato è il *Mesh Renderer.*

Event Trigger

L'*Event Trigger* è un componente che sta in ascolto di eventuali eventi e quando essi si scatenano esegue lo script specificato.

Animator

Se un oggetto vuole essere animato, esso deve possedere obbligatoriamente un *Animator*. In questo componente deve essere specificato un *Animator Controller* che conterrà a sua volta i riferimenti ai vari stati in cui l'oggetto si può trovare e le relative animazioni.

Script

Oltre ai componenti già previsti da Unity, possono essere definiti degli *Script* che potranno essere scatenati da un *Event Trigger* oppure al momento della creazione dell'oggetto. Negli script possono essere definite delle variabili che, se rese pubbliche, potranno essere modificate direttamente dall'*Inspector* facendo diventare di fatto questo componente molto flessibile.

Tipi di file e organizzazione del progetto

Durante la creazione di un progetto saranno necessari diversi tipi di file la cui organizzazione è fondamentale. Personalmente ho suddiviso tutti i file in 7 cartelle:

Animations

Nella cartella *Animations* ho racchiuso tutto ciò che riguarda le animazioni dei modelli. In particolare si possono distinguere due file specifici:

- *Animation*: racchiude un'animazione di un oggetto. In particolare salva il cambiamento delle proprietà dei componenti dell'oggetto (o dei suoi oggetti "figli") durante un arco di tempo specificato opportunamente scandito in N intervalli regolari (*Samples*). Ad esempio se l'animazione dovesse durare un secondo con *Samples*=60, sarà possibile specificare una variazione delle proprietà ogni 1/60 secondi.
- *Animation Controller*: questo file specifica il comportamento dell'oggetto durante i diversi stati che attraversa. Per ogni stato è possibile specificare un'animazione. Ad esempio, pensando al movimento di un personaggio esso può passare da tre stati principali: camminata, corsa o attesa; per ognuno di essi può essere specificata un'animazione differente.

Oltre alle *Animations* e agli *Animation Controller* è utile specificare nella stessa cartella gli script che riguardano le animazioni.

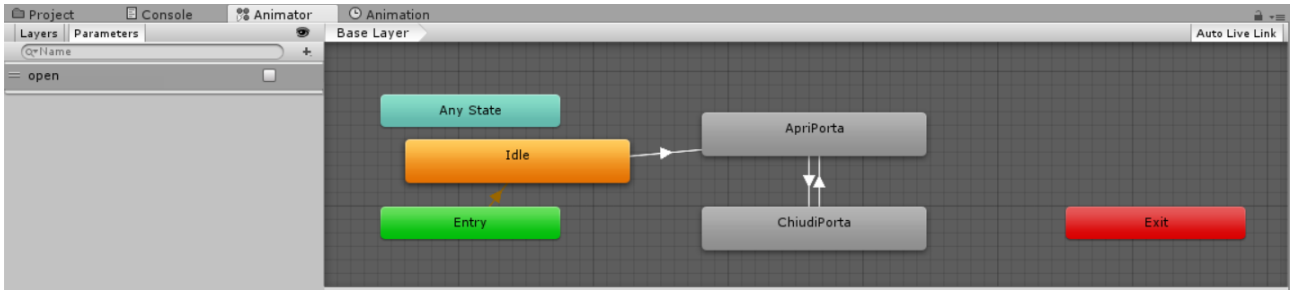


Figura 7 - La visualizzazione degli stati in un Animation Controller

Materials

I *Materials* sono dei file che specificano come deve essere renderizzato un oggetto. Essi possono contenere oltre ad un colore unico o ad una texture, le proprietà di riflessione della luce di un oggetto o le volte che una data texture dovrà essere ripetuta sulla superficie dell'oggetto. Definire molti *Materials* diversi influirà sulle prestazioni del prodotto finale (vedi paragrafo Ottimizzazione).

Textures

Le *Textures* sono delle immagini che racchiudono una trama che solitamente viene ripetuta durante tutta la superficie dell'oggetto. Vengono spesso utilizzate per superfici vaste come prati o rocce. Altri tipi di texture sono invece applicate su oggetti piccoli e complessi che richiederebbero troppi poligoni per essere definiti correttamente. Ad esempio nel caso si voglia modellare una tastiera risulterebbe troppo complesso creare ogni singolo tasto; in questo caso quindi è molto più semplice creare un parallelepipedo ed applicare una texture sulla parte superiore della tastiera che mostrerà così una foto dei tasti.

Models

Nella cartella *Models* sono contenuti tutti i modelli custom creati da un software esterno.

Prefabs

I *Prefabs* sono oggetti che si possono definire astratti ovvero sono un insieme di oggetti e componenti che sono stati definiti allo scopo di poterli riutilizzare rapidamente più volte senza doverli ridefinire di nuovo. Inoltre, le modifiche apportate ad una istanza di *Prefab* possono essere applicate al modello generale in modo da essere trasferite a loro volta a tutte le altre istanze.

Scenes

Una scena può essere pensata come un livello di un videogioco o ad una stanza. Essa, di fatto, è un insieme di *GameObjects* e l'insieme delle scene crea l'ambiente virtuale. La gestione delle scene in modo appropriato è fondamentale per una buona ottimizzazione del gioco, infatti, esse possono essere caricate e scaricate tramite degli script durante l'esecuzione dell'applicazione (vedi paragrafo Ottimizzazione).

Scripts

Nella cartella *Scripts* sono presenti tutti gli scripts generici associati a diversi oggetti o utilizzati nell'editor.

Rendering

Per comprendere bene come ottimizzare al meglio l'applicazione è necessario capire in che modo lavora Unity al fine di renderizzare l'immagine finale.

Innanzitutto l'immagine è ridisegnata interamente una volta per frame quindi se l'applicazione deve funzionare a 60 FPS (Frame Per Second) il dispositivo (specialmente la GPU) dovrà ridisegnare gli oggetti 60 volte in un secondo, sottoponendola ad uno stress non indifferente.

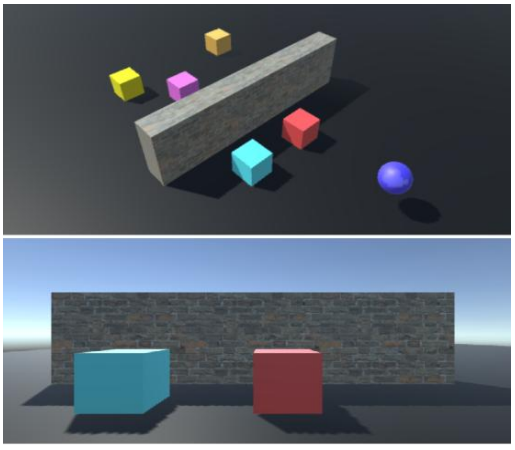


Figura 8 - Campo visivo e rendering

L'ambiente però non è ridisegnato completamente, infatti, Unity si preoccupa di renderizzare solo quello che è nel campo visivo della camera. Il campo visivo della camera però non si ferma solo agli oggetti visibili dall'utente ma prosegue individuando anche quelli nascosti da altri oggetti. Come si vede dalla figura 8, Unity si dovrà preoccupare di visualizzare sia gli elementi prima del muro sia quelli dietro il muro, che l'utente in realtà non vede.

Ogni oggetto ha una complessità diversa e di conseguenza un carico differente sulla GPU. Come già anticipato precedentemente, alcuni tipi di oggetto o componenti (come i modelli 3D o i *Colliders*) sono composti da poligoni che a loro volta sono identificati da vertici uniti da spigoli. Più poligoni compongono un oggetto, più un oggetto è definito. Il motore di rendering subirà invece un carico maggiore al crescere del numero di poligoni. Si può quindi dire che:

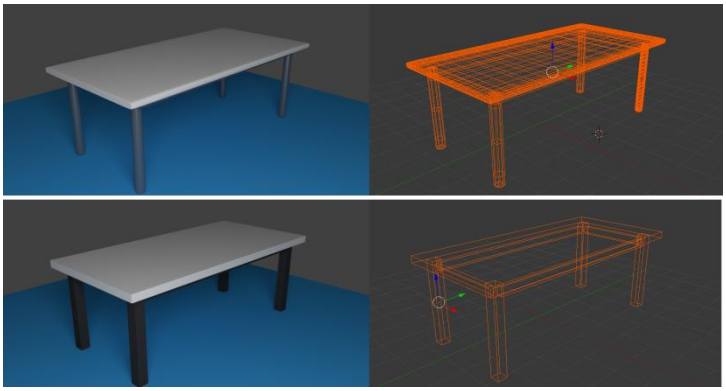


Figura 9 - Un modello HighPoly (sopra) e un modello LowPoly (sotto)

La definizione di un oggetto è direttamente proporzionale al numero dei poligoni

mentre

Le prestazioni sono inversamente proporzionali al numero di poligoni

Non solo i poligoni influiscono sulle prestazioni ma anche i *Materials.* Ogni *Material* differente influisce sulle operazioni che il motore di rendering deve eseguire. Ad esempio renderizzare un cubo interamente rosso, quindi con solo un *Material*, è diverso da renderizzare lo stesso cubo con 3 facce rosse e 3 facce gialle, quindi con due *Materials*: uno rosso e uno giallo.

Uno dei parametri che indica quanto è pesante la scena che deve essere renderizzata viene nominato *Draw Calls* (o *SetPass Calls*). Questo parametro è un numero che indica quante volte per frame viene chiamata la funzione che disegna uno o più oggetti. Più questo numero è basso, migliori saranno le prestazioni. Tornando all'esempio precedente, per disegnare un cubo con un solo *Material* è necessaria una sola chiamata alla funzione mentre per disegnare un cubo con due *Materials* sono necessarie due chiamate alla funzione, una per ogni *Material*.

Un ultimo fattore da considerare è l'illuminazione della scena. L'impostazione predefinita dell'illuminazione della scena su Unity è *Realtime* ovvero le luci e le ombre vengono ricalcolate ad ogni frame. Questo garantisce un effetto realistico siccome ad ogni spostamento di un oggetto l'ombra e i riflessi cambiano in tempo reale. Questo però è computazionalmente oneroso e non sempre è utilizzabile.

Al fine di ridurre al minimo il carico sulla GPU è doveroso attuare dei sistemi di ottimizzazione.

Ottimizzazione

Fin da subito, ancora prima di iniziare a modellare qualsiasi oggetto, è stato fondamentale concentrarsi sull'ottimizzazione. Questo è particolarmente importante in un progetto del genere per diversi motivi:

- La realtà virtuale è computazionalmente onerosa. Questo perché l'immagine deve essere renderizzata due volte, una volta per l'occhio destro e una volta per l'occhio sinistro.
- In un'applicazione in realtà virtuale è necessario mantenere un framerate elevato, il più possibile vicino ai 60 FPS. Un framerate inferiore potrebbe causare nausea più velocemente o comunque rovinare molto l'esperienza.
- Uno smartphone non dispone di una elevata capacità di elaborazione come un PC o una console.
- L'ambiente è formato da molte stanze ognuna delle quali ha molti oggetti all'interno e quindi la quantità di poligoni da gestire è molto elevata.

Al fine di ottenere un'esperienza più fluida possibile è stato necessario adottare diverse tecniche di ottimizzazione scendendo a compromessi tra fluidità e grafica:

Grafica LowPoly

Adottare una grafica realistica ricca di riflessi e con dei modelli ultra definiti formati da migliaia di poligoni sarebbe stato impraticabile per via dei motivi sopra enunciati, per cui è stato meglio adottare una grafica definita LowPoly ovvero con un basso numero di poligoni. Questo stile si basa sulla rappresentazione di oggetti molto squadrati ma basati comunque su modelli realistici. I colori pastello sono dominanti e sono accompagnati da una tonalità vivace, vicino a uno stile cartoonesco. Oltre a ridurre il carico sulla GPU questo stile permette una più rapida creazione dei modelli e scelta dei colori. L'uso delle texture è limitato alla semplificazione di modelli troppo complessi.

Figura 10 - Illustrazione in stile LowPoly

Baked Lighting

L'illuminazione *Realtime*, come detto in precedenza, sarebbe troppo onerosa in certe situazioni per cui Unity fornisce un'altra metodologia di creazione dell'illuminazione definita *Baked.* Essa permette di elaborare luci ed ombre prima della compilazione, direttamente dall'editor. Il risultato dell'elaborazione viene memorizzato nelle *lightmaps* che, a seconda del livello di dettaglio impostato, peseranno da pochi MB a decine di MB. Le *lightmaps* saranno poi caricate al momento dell'esecuzione dell'applicazione e l'illuminazione non sarà più calcolata.

L'illuminazione *Baked* però non è dinamica per cui si avrà come l'effetto che le ombre siano direttamente "dipinte" sugli oggetti e non varieranno mai. Questa impostazione non è adatta per oggetti dinamici ma nel caso di ambienti statici è perfetta.

Static Batching

Unity permette di ottimizzare il rendering di più oggetti simili ovvero che condividono lo stesso modello e gli stessi *Materials*. Se più oggetti simili sono impostati come *Static* (ovvero statici, non dinamici) essi verranno renderizzati tutti tramite un'unica *Draw Call*. Questo può portare notevoli miglioramenti perché si avrà pressoché lo stesso carico sulla GPU ad esempio sia disegnando un solo albero che disegnandone mille uguali.

Additive Scenes (caricamento dinamico delle scene)

In Unity è possibile caricare più scene contemporaneamente. Questa tecnica di caricamento viene definita *Additive*. Tramite la somma di più scene è possibile identificare diversi "blocchi" ad esempio una scena per il giocatore e la sua camera, un'altra scena per la stanza e un'altra scena per il cielo. In questo modo in caso si debba cambiare solo la stanza basterà scaricare la stanza corrente e caricarne un'altra lasciando inalterate le scene del giocatore e del cielo.

Questa tecnica è stata utile nel progettare l'ambiente di gioco infatti ogni stanza è identificata da una scena. Appena il giocatore si avvicina ad una porta, esso interagisce con un collider che scatena un evento che permette all'applicazione di scaricare le scene inutili e di caricare invece la stanza nella quale il giocatore presumibilmente entrerà. Continuando a caricare e scaricare scene è possibile ingrandire l'ambiente a piacere aggiungendo sempre più stanze senza intaccare le prestazioni siccome l'applicazione ha in memoria solo le poche scene necessarie. Tramite questo metodo si risolve il problema di Unity che renderizza anche gli oggetti che l'utente non vede.

Antialiasing e vSync

Per migliorare la qualità grafica, Unity implementa *Antialiasing* e *vSync*:

- L'*Antialiasing* permette di ridurre l'effetto seghettato che si forma su degli spigoli che dovrebbero essere dritti.
- Il *vSync* (anche detto "Sincronizzazione verticale") permette di sincronizzare la frequenza di immagini renderizzate con la frequenza di aggiornamento del monitor. La mancata sincronizzazione causa una sovrapposizione di frame che danneggia la qualità dell'immagine creando delle bande orizzontali.

L'utilizzo di queste tecniche è utile ma richiedono una maggiore capacità di elaborazione. In dispositivi poco potenti è bene disattivarle.

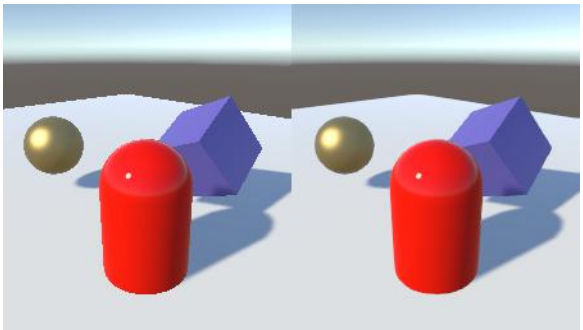


Figura 11 - Esempio di Antialiasing

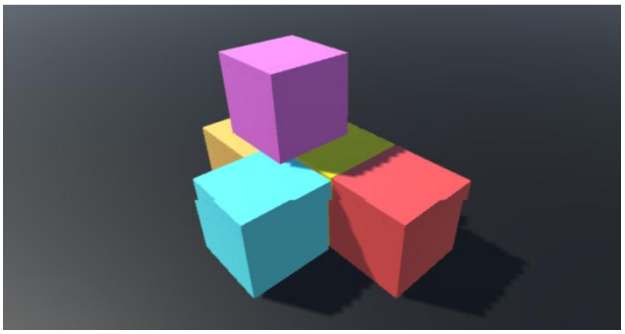


Figura 12 - Distorsione dell'immagine causata dalla mancata sincronizzazione dei frame

Inizio del progetto

Analisi

L'inizio vero e proprio del progetto è stato preceduto da una lunga fase di analisi e formazione allo scopo di conoscere, in modo abbastanza approfondito, il nuovo ambiente di lavoro. In questa fase ho sperimentato tutto ciò che avevo in programma di fare su una singola stanza, al fine di procedere in modo spedito durante l'ampliamento dell'ambiente ed evitare di dover tornare indietro a causa di errori alla base del progetto. Questa strategia si è rivelata un successo.

Dopo la prima fase di analisi si sono delineati i punti fondamentali del progetto che sono stati organizzati nello specifico su Trello. Essi possono essere schematizzati nei seguenti punti:

- Creazione della palazzina di informatica con diversi oggetti interagibili.
- La grafica sarà in stile *LowPoly* ma comunque fedele all'ambiente reale.
- L'ottimizzazione è fondamentale per garantire un frame rate il più possibile vino ai 60 FPS.

Impostazione del progetto

Una volta creato un progetto generico su Unity è necessario impostare fin da subito dei parametri e importare il necessario in modo da renderlo specifico per il mio contesto particolare.

Compilare per Android

L'applicazione deve essere eseguita su un dispositivo Android per cui è necessario specificarlo dalla finestra *Build Settings* accedibile dal menù *File* della barra superiore. In questa finestra basta premere su *Android* e successivamente su *Switch Platform.*

Premendo invece sul pulsante *Player Settings…* si possono modificare diversi parametri di compilazione specifici per la piattaforma scelta. In questa finestra oltre ad impostazioni specifiche si può selezionare l'icona, il nome dell'app, la versione e il nome del pacchetto che verrà compilato. Una cosa importante da impostare è la versione minima del sistema operativo richiesto dall'app. Esso deve essere *Android 4.4 'Kitkat' (API level 19)* perché le applicazioni VR non sono supportate dalle versioni precedenti di Android.

Qualità e grafica

Nel menu *Edit* della barra superiore è possibile trovare *Project Settings* da cui è possibile accedere alle finestre *QualitySettings* e *GraphicsSettings*.

Unity permette di definire dei livelli di qualità per l'applicazione. Quando si crea un'applicazione per pc ad esempio all'avvio si può decidere il livello di qualità da un minimo (*Fastest*) a un massimo (*Fantastic*).

Per questa applicazione ho deciso di definire un unico livello partendo dal livello preimpostato *Simple*. Le modifiche apportate si limitano a disattivare *Anisotropic Textures*, *Antialiasing* e *vSync* (vedi paragrafo Antialiasing e vSync).

Anche per le impostazioni grafiche Unity permette di definire dei livelli. In questo caso non è possibile disabilitarli per cui ho definito la qualità grafica minore per tutti e tre i livelli possibili.

La struttura delle scene

Per applicare il principio delle *Additive Scenes,* il progetto è stato strutturato con una sola scena sempre presente denominata *VrMain* e altre scene che vengono caricate e scaricate in base alle azioni dell'utente.

La scena VrMain

La scena *VrMain* rappresenta il giocatore. Essa è composta da un GameObject vuoto da cui dipendono diversi oggetti tra cui la camera, il contatore di FPS (*GvrFPSCanvas*) e il puntatore (*GvrReticle*). Questi ultimi due oggetti sono dei *Prefabs* che si trovano nei GoogleVR SDK. Nella scena è presente anche l'oggetto *GvrViewerMain* che fornisce la camera VR completa di movimento della testa.

Siccome la scena non verrà mai scaricata essa contiene un oggetto custom definito GameController. Esso è un GameObject vuoto con uno script all'interno che gestisce il caricamento dinamico delle scene (vedi paragrafo Ottimizzazione).

Le altre scene

Oltre alla scena principale *VrMain* ci sono solo scene che rappresentano l'ambiente di gioco. Ogni scena rappresenta una stanza dell'ambiente virtuale. Nella stanza sono presenti il pavimento, il soffitto, i muri laterali, gli oggetti statici e interattivi e i punti luce.

Al lavoro!

Il progetto è stato suddiviso in 3 fasi di lavoro che si sono susseguite nell'ordine per ogni oggetto da inserire nell'ambiente:

- Modellazione
- Assemblaggio
- Scripting e animazione

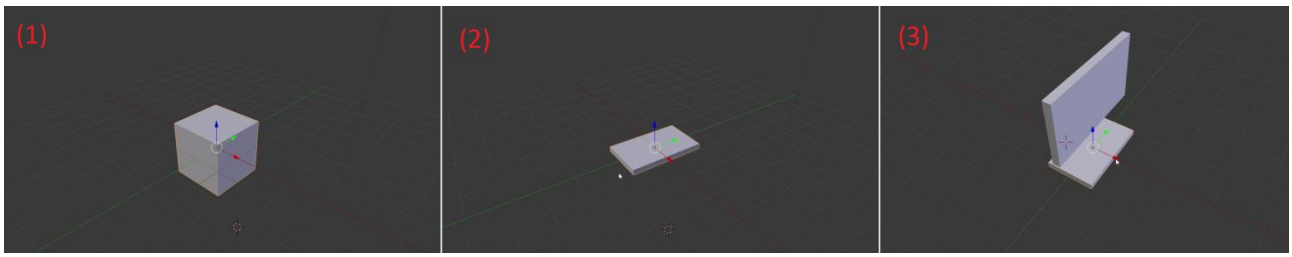
Modellazione

Siccome ho deciso di modellare tutti gli oggetti personalmente e non di procurarmene già pronti dall'Asset Store, questa fase è stata fondamentale al fine di creare i modelli 3D da inserire nelle scene.

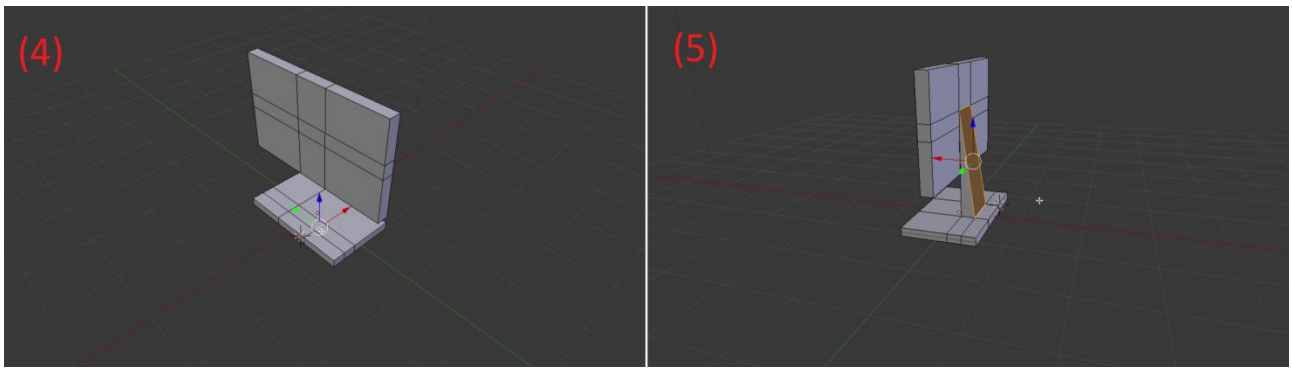
Per ogni oggetto sono partito dall'aspetto reale da cui estrapolare i tratti principali. Questi sono sufficienti per rendere l'oggetto riconoscibile nonostante le forme squadrate e approssimate della grafica LowPoly.

Una volta definita un'idea generale, ho potuto iniziare a modellare su Blender. Solitamente si parte da un cubo e tramite l'estrusione di varie facce, lo spostamento dei singoli vertici o l'unione di questi tramite degli spigoli si può definire una forma sempre più complessa.

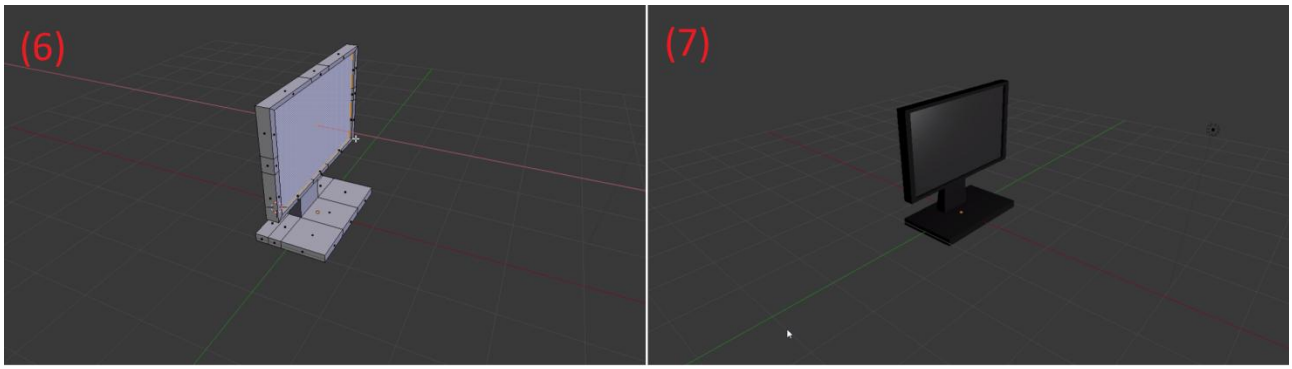
Per esempio, analizziamo la creazione di un monitor. Si parte da un cubo (1) che opportunamente scalato diventerà la base del monitor (2). La stessa forma può essere utilizzata anche per il monitor per cui con una duplicazione e qualche aggiustamento alle proporzioni si hanno già i due componenti principali (3).



Ora le due parti devono essere unite. Si procede sezionando i due oggetti (*Loop cut and slide*) in modo da avere due rettangoli da congiungere come nel modello reale (4). I due rettangoli creati si devono congiungere selezionando i due spigoli corrispondenti e creando una faccia (premendo F) tra di essi (5).



Una volta completata la base si possono rifinire dei dettagli come la cornice attorno allo schermo (6). Infine si possono assegnare i colori in modo da avere una prima bozza del risultato finale (7).



Prima di esportare il modello è fondamentale applicare (tramite il menù *Apply*) posizione, rotazione e proporzioni dell'oggetto per evitare problemi in Unity. Inoltre è necessario creare anche una *UV Map* in modo che Unity possa correttamente calcolare luci ed ombre. Senza di questa, l'oggetto non creerebbe nessuna ombra, come se fosse invisibile.

Dopo avere ultimato la creazione del modello su Blender, esso deve essere esportato in uno dei formati riconosciuti da Unity, ad esempio FBX. In questa operazione bisogna fare attenzione ad esportare solo il modello che ci interessa e non altri componenti presenti del progetto di Blender come le luci o la camera.

Assemblaggio

Il modello può essere importato in Unity semplicemente trascinandolo nella scheda *Project*. Cliccando sull'oggetto importato, nell'*Inspector* si possono impostare le proporzioni e automatizzare la creazione dei *Colliders.*

Ora che il modello è importato basterà trascinarlo nella scena e disporlo dove è necessario.

Questa operazione potrebbe sembrare banale, ma non è semplice impostare correttamente le proporzioni e far combaciare coerentemente gli oggetti

Scripting e animazione

Se l'oggetto deve essere interattivo è necessario assegnargli diversi componenti e script.

Nel mio caso per interagire con un oggetto ho previsto due script e un *Event Trigger*.

Lo script *Interactive Object* si occupa di notificare l'evento di avvenuta pressione del pulsante di interazione. Per rendere più realistica l'interazione esso interpreta come click anche quando l'utente guarda per due secondi un oggetto interattivo.

Una volta lanciato l'evento, l'*Event Trigger* lo intercetta e, se è definita un'azione per quell'evento, la esegue. Nell'esempio mostrato dall'immagine, in caso di un evento *PointerDown*, l'*Event Trigger* eseguirà la funzione toggleStatus() presente nello script *Apri Chiudi.*

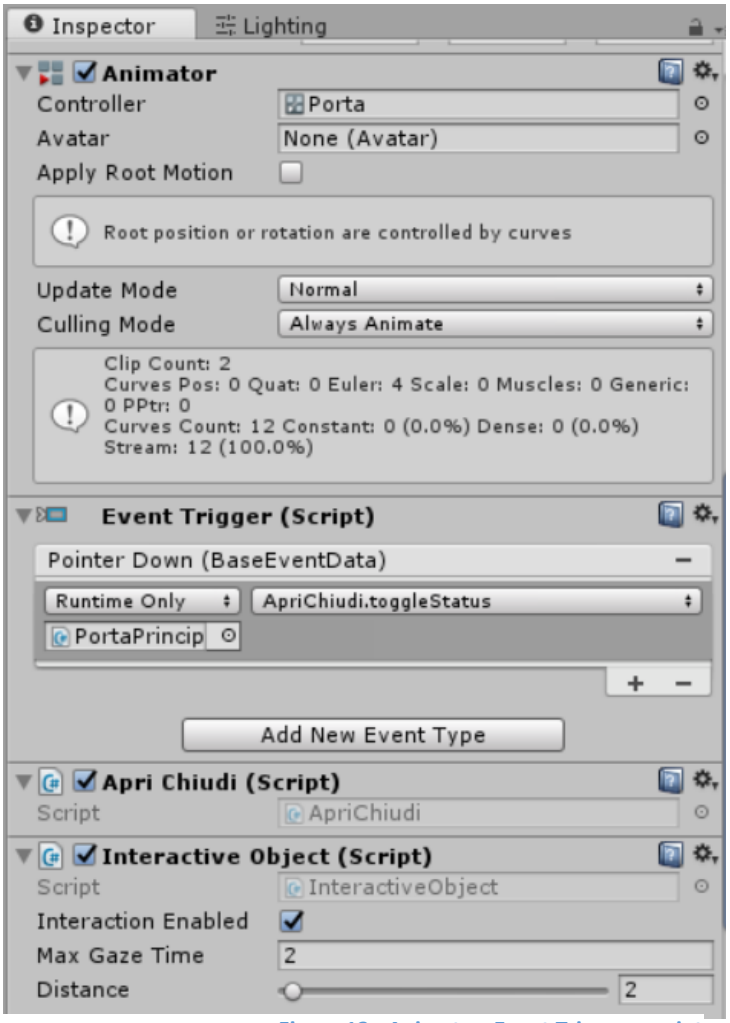


Figura 13 - Animator, Event Trigger e script

Nel caso particolare mostrato in figura, l'interazione prevede l'esecuzione di un'animazione. All'oggetto in questione (una porta) è assegnato un *Animator*. Esso ha lo scopo di referenziare un *Animation Controller.* Lo script *Apri Chiudi* ha lo scopo di modificare un parametro booleano "open" nel *Controller* in modo da cambiare lo stato dell'oggetto ed eseguire l'animazione.

Conclusione e compilazione

Una volta che tutti gli oggetti sono al posto giusto e le interazioni funzionano si può procedere con un *Baking* dell'illuminazione tramite la scheda *Lighting* così verranno create tutte le lightmap necessarie. In presenza di molte scene questa operazione potrà richiedere diversi minuti quindi è bene essere sicuri che tutto sia corretto per non sprecare del tempo.

Ora che tutto funziona non resta che compilare il progetto in modo da creare un file apk che si possa installare su un dispositivo Android. Dalla scheda *Build Settings…* (già vista durante l'impostazione del progetto) basterà aggiungere, nella parte superiore della finestra, tutte le scene che verranno utilizzate dal progetto mettendo al primo posto la scena *VrMain.* Premendo su *Build,* verrà generato il file apk.

Se l'applicazione riesce ad essere eseguita intorno ai 60 FPS su diversi dispositivi, il lavoro può considerarsi concluso.

Risultati

Come prima esperienza in un ambiente così diverso da quello cui si è abituati durante le lezioni, la ritengo senza dubbio un successo.

L'applicazione in linea generale è buona, la fluidità è ottima, la giocabilità è buona ma migliorabile, la grafica è semplice ma allo stesso tempo fedele all'ambiente reale.

Conoscere un nuovo ambiente di sviluppo, come lavora e i problemi che possono sorgere, mi ha permesso di analizzare e capire il funzionamento di realtà che utilizzo normalmente oltre che ad aprirmi nuovi orizzonti e possibilità di miglioramento delle abilità apprese.

Si può fare di meglio?

Certamente. Il progetto vuole essere una rappresentazione basilare ma abbastanza completa della sola palazzina di informatica dell'istituto. Grazie ai pochi script versatili e al caricamento dinamico delle scene su cui si basa l'applicazione, si possono effettuare diversi miglioramenti o introdurre nuovi contenuti.

L'ambiente può essere espanso a piacere senza che le prestazioni degradino perché la tecnica di caricamento dinamico delle scene manterrà in memoria solo le poche scene necessarie evitando di sovraccaricare inutilmente la GPU.

Possono essere introdotti nuovi oggetti con nuove interazioni in modo da rendere l'ambiente sempre più realistico.

Oltre a rimanere una semplice simulazione fine a se stessa può essere tramutata in videogioco inserendo delle missioni da completare tramite diverse interazioni con gli oggetti.