POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Corso di Laurea Magistrale
in Ingegneria del Cinema e dei Mezzi di Comunicazione

Tesi di Laurea Magistrale

Sviluppo di un'applicazione in realtà virtuale per la performance multimediale VERA (Virtual Experience Real Approach)

Relatore
Prof. Andrea Giuseppe Bottino
Correlatore
Dott. Francesco Strada

Candidato
Gabriele Martin

Anno Accademico 2019/2020

Indice

Abstract

Il presente elaborato illustra il processo di sviluppo e produzione del progetto *VERA*, un'esperienza in realtà virtuale multi-utente e co-localizzata, in cui i membri del pubblico condividono la presenza nel medesimo luogo fisico e virtuale, in un contesto performativo ibrido, in cui si mescolano musica dal vivo e fruizione di contenuti attraverso i visori.

L'obiettivo principale è quello di esplorare le potenzialità espressive offerte dalla VR in un ambiente performativo innovativo, al fine di capire se la cognizione della presenza altrui, così come gli stimoli fisici percepiti dall'utente, possano condizionare positivamente il processo di immersione virtuale, e viceversa.

Per identificare le potenziali criticità e i requisiti operativi associati ad un simile spettacolo, sono state analizzate varie soluzioni nell'ambito della VR multi-utente co-localizzata, con un particolare interesse verso le opere a carattere performativo e narrativo.

Si sono affrontate le principali problematiche associate alla produzione di contenuti grafici 3d per la realtà virtuale, nel caso di visori mobile, così come le tecniche di ottimizzazione adottate per garantire un'esperienza fluida ed esteticamente appagante per il pubblico.

Infine, il lavoro si è concluso con la sperimentazione e la raccolta dei risultati ottenuti dai test di immersione, presenza e comfort, sottoposti al pubblico presente ai primi spettacoli dal vivo di VERA, tenutisi a settembre 2020 presso l'hub culturale *OffTopic* e il *Circolo dei Lettori* di Torino, come evento del festival *Torino Spiritualità*.

1 Introduzione

Nonostante sia ancora spesso descritta in termini avveniristici, la realtà virtuale è una tecnologia di cui si parla ormai da tempo: i primi esperimenti in tal senso, infatti, risalgono ai primi anni sessanta.

Nello specifico, la data che viene solitamente associata alle origini di tale settore è il 1962, anno in cui Morton Leonard Heilig brevetta il *Sensorama*[1], ovvero una macchina cinematografica pensata per far vivere allo spettatore un prodotto audiovisivo in maniera maggiormente immersiva, rispetto alle proiezioni tradizionali.

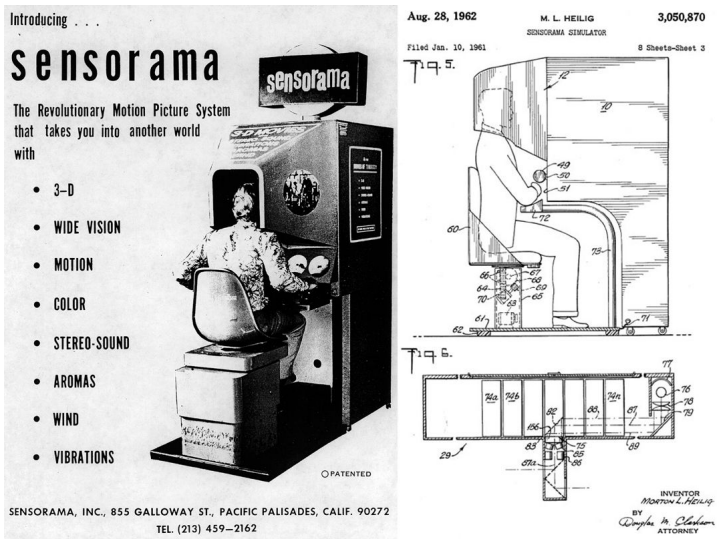

Figura 1: Sensorama

Il dispositivo (Fig.1) mirava a sollecitare tutti e cinque i sensi dello spettatore, includendo nell'esperienza, oltre alle immagini in stereoscopia e al sonoro, anche vibrazioni, soffi di vento e odori.

Dopo questo primo prototipo a grande scala, ebbero inizio gli esperimenti per la realizzazione di apparecchiature più compatte e indossabili dall'utente. Il primo *Head Mounted Display* (HMD), considerato l'antenato originale dei moderni visori, si deve a Ivan Sutherland, che nel 1968 crea *The Sword of Damocles*[2] (letteralmente "Spada di Damocle", Fig.2).

Con tale visore, era possibile vedere ambienti virtuali rudimentali, sfruttando un display stereoscopico e un tracking dei movimenti della testa dell'utente. Tale tecnologia era basata su un'infrastruttura meccanica piuttosto robusta (visto il peso del visore), che era agganciata al soffitto del laboratorio

[1] http://www.treccani.it/enciclopedia/realta-virtuale/
[2] https://en.wikipedia.org/wiki/The_Sword_of_Damocles_(virtual_reality)

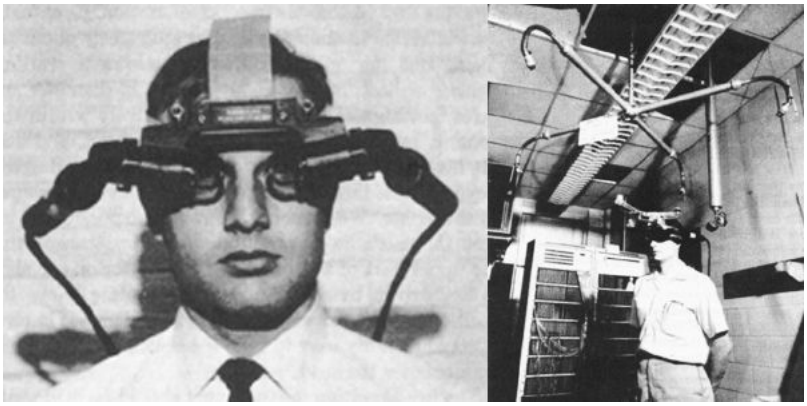

Figura 2: The Sword of Damocles

e sospesa sopra il tester (derivando da questo fatto il suo particolare nome).

Nel corso delle decadi successive, le aspettative per la realtà virtuale iniziarono a crescere con costanza, così come i tentativi di sfruttare una simile tecnologia in modi innovativi.

A metà degli anni 80', ad esempio, viene prodotto dalla NASA il *Virtual Interface Environment Workstastion* (VIEW)[3], un apparecchio costituito da un elmetto con display stereoscopico e accompagnato da guanti dotati di sensori (*data glowes*), che consentivano all'utente di interagire con la scena 3d, usando le proprie mani.

Qualche anno prima, sempre in ambito di ricerca, anche il MIT crea *The Aspen Movie Map*, ovvero un prototipo di tour virtuale in 3d della città di Aspen in Colorado, che può essere considerato un precursore dell'attuale Google Street View.

Con l'arrivo degli anni 90', grazie al crescente interesse all'interno del settore video-ludico, vengono prodotti i primi visori consumer, acquistabili dal pubblico. Tecnologie come il *Nintendo Virtual Boy* (1995), precedute inizialmente da grandissimo entusiasmo, non riescono tuttavia ad affermarsi a livello mainstream, soprattutto per il costo elevato, la povertà di contenuti offerti e dal tutt'altro che eccellente comfort.

La sconfitta commerciale di cui è simbolo Virtual Boy, genera nella stampa e nel grande pubblico un sentimento di diffidenza e disillusione nei confronti dell'intero settore della realtà virtuale, rischiando quasi di mettere la parola fine alle grandi promesse che lo avevano accompagnato per anni.

La situazione rimarrà pressoché immutata fino al 2012, anno in cui viene lanciato sulla piattaforma di crowdfunding *Kickstarter* il prototipo di un

[3] https://www.nasa.gov/ames/spinoff/new_continent_of_ideas/

Figura 3: Il visore VIEW della NASA

Figura 4: Nintendo Virtual Boy

nuovo visore, che sarà destinato a cambiare completamente il mercato della VR, viste le ottime specifiche tecniche, l'ergonomia e il costo sempre più accessibile: *Oculus Rift*.

Dopo soli due anni dall'inizio del suo sviluppo, la startup Oculus che stava sviluppando il visore viene acquistata da Facebook per la cifra di 2

Figura 5: Oculus Rift - versione consumer

miliardi di dollari, riportando attenzione e investimenti ad un settore che da tempo giaceva in silenzio.

Questo, oltre a rinnovare l'interesse del pubblico specializzato, ha convinto anche altri grandi player tecnologici a dedicare fondi sempre più consistenti allo sviluppo delle soluzioni VR, che tornavano piano piano ad essere considerate "the next big thing", ovvero il prossimo segmento di mercato tecnologico destinato a crescere esponenzialmente in tempi brevi.

Infatti, accanto a Oculus Rift, nascono anche i primi prodotti di HTC, Samsung, Microsoft, Sony, Valve e molti altri ancora, ravvivando la competizione all'interno del settore e aumentando l'offerta di prodotti consumer, rendendo complessivamente più accessibili i prezzi d'acquisto per l'utente medio.

Oltre alla proliferazione di soluzioni hardware, negli anni aumentano anche i campi di applicazione degli applicativi VR, basti pensare alla nascita della *social VR* con Facebook sempre in primo piano, ma anche alle applicazioni per il training in sicurezza di personale qualificato in ambienti virtuali (come ad esempio i piloti o gli astronauti). Oppure, si pensi all'industria dell'intrattenimento, con applicazioni che vanno dal gaming in VR (sempre più in voga anche tra le offerte dei produttori di console), all'evoluzione del cinema di animazione e non, che diventa sempre più interattivo e subordinato all'esperienza personale e ai tempi dell'utente.

Da come appare il quadro, dunque, sembra che la realtà virtuale abbia ormai le carte in regola per diventare alla portata di tutti e di conquistare il grande pubblico.

Tuttavia, ci si può dire davvero certi che sarà questo lo scenario?

In questo senso, una delle visioni più pessimistiche tra gli addetti ai lavori è da attribuire a Reginald Fils-Aimé, ex presidente della divisione americana di Nintendo, il quale durante la conferenza *E3* del 2015 aveva dichiarato: "[La VR] non è divertente e non è sociale. È solo *tech*[4]".

Per quanto possa apparire come un punto di vista drastico, esso ha il pregio di focalizzare l'attenzione su un potenziale problema: la VR è un mezzo espressivo e, se non viene tenuto in considerazione anche l'aspetto sociale dell'esperienza, si rischia di fallire, come in passato.

Con aspetto sociale, si intende anche la capacità della tecnologia di poter essere accessibile, nei suoi contenuti, anche ai non appassionati, che costituiscono il segmento di pubblico più complesso da coinvolgere e coltivare, ma anche quello più numeroso, che può realmente determinare una diffusione su larga scala.

Al di là delle migliorie dei produttori hardware, si ritiene che buona responsabilità della crescita del settore sia in mano ai produttori di contenuti, i quali sono chiamati a trovare modalità espressive nuove, per coinvolgere l'utenza e farla familiarizzare con i contesti immersivi.

Si crede che l'industria dell'intrattenimento e gli ambienti performativi siano il contesto ideale per portare avanti una simile sperimentazione, non a caso VERA intende inserirsi ed operare all'interno di questo sotto-settore.

In particolare, con lo spettacolo di VERA si vuole indagare la risposta del pubblico ad un nuovo tipo di applicazione della realtà virtuale, che combini multi-presenza fisica e virtuale dei partecipanti, VR narrativa e il concetto di co-localizzazione (ovvero la presenza di più fruitori simultaneamente nello stesso luogo).

All'interno dell'elaborato, verrà illustrata tutta la realizzazione del progetto, dai suoi obiettivi preliminari, fino alle soluzioni e ai risultati ottenuti. Nel capitolo 2, verrà fornita una descrizione del panorama della realtà virtuale multi-utente, co-localizzata e non, presente sul mercato. Poi, nel capitolo 3 si illustrerà il progetto VERA in una veste ad alto livello, indicandone i requisiti, i limiti e le soluzioni hardware e software adottate. Nel capitolo 4, si entrerà nel vivo della descrizione della struttura narrativa e tecnica del progetto, analizzando le principali sfide di produzione che è stato necessario affrontare. Il capitolo 5 sarà interamente dedicato ai processi di ottimizzazione, che è stato necessario adottare per finalizzare la produzione dei contenuti VR, in un contesto di visori mobile. Il capitolo 6 conterrà l'analisi dei risultati derivanti dai test di immersione effettuati sugli utenti che han-

[4] https://www.polygon.com/2015/6/18/8803127/nintendos-fils-aime-current-state-of-vr-isnt-fun

no sperimentato VERA in un contesto multi-utente e co-localizzato (nella cornice del festival *Torino Spiritualità 2020* e presso l'hub culturale torinese *OffTopic*). Da ultimo, nel capitolo 7 saranno presentate le conclusioni e le potenziali prospettive future per il progetto.

2 Stato dell'Arte

Nella fase di pre-produzione del progetto, al fine di definire l'identità, i limiti e le prospettive di VERA, sono state analizzate alcune delle soluzioni presenti nel panorama della realtà virtuale *multi-utente*.

Si è scelto di catalogare le diverse soluzioni analizzate, distinguendo in:

- esperienze *co-localizzate*
- esperienze *de-localizzate*

La condizione di fruizione dell'esperienza ha perciò costituito il criterio principale per il processo di classificazione.

Accanto a questo, dato che gli approcci nell'uso della VR sono estremamente vari, si è scelto di adottare anche un criterio secondario di classificazione, che ha invece a che vedere con il tipo di applicazione che viene proposta: si va dalla social VR, alla VR videoludica, al design VR collaborativo, alla VR performativa, per citare alcuni esempi.
Proprio quest'ultima modalità, legata al mondo delle live performance e dello storytelling in realtà virtuale, costituisce uno degli elementi focali dell'analisi, in quanto simile a quello che cerca di proporre VERA.

Nel capitolo, vedremo diversi esempi di soluzioni aventi condizioni di fruizione e destinazioni d'uso differenti, prestando una particolare attenzione per le esperienze co-localizzate di tipo performativo.
Tali applicazioni, pur costituendo ancora una nicchia nel mondo della realtà virtuale, consentono agli utenti che coesistono nel medesimo mondo sintetico di avere un livello di interazione ed immersività che, secondo recenti studi [2], supera quello dei paradigmi di fruizione tradizionali.

2.1 Esperienze VR de-localizzate

Per esperienze *de-localizzate* si intendono tutte quelle in cui gli utenti sono normalmente situati in luoghi fisici diversi tra loro. Esiste perciò una simultaneità temporale, ma non necessariamente una condivisione degli spazi tra i partecipanti. La maggior parte delle applicazioni VR sul mercato appartiene a questa categoria.

2.1.1 Facebook Horizon e Oculus Venues

Uno degli attori che, soprattutto negli ultimi anni, ha manifestato la volontà di recitare un ruolo di primo piano nell'universo della realtà virtuale è Facebook.

La compagnia, che peraltro detiene dal 2014 la proprietà di Oculus (azienda produttrice dei visori Oculus Rift, Quest e Go), già da diversi anni afferma di voler puntare sulla cosiddetta *social VR*, ovvero sulla trasposizione delle attività tipiche dei social media, all'interno di mondi virtuali interagibili, personalizzabili e condivisi.

Figura 6: Facebook Horizon

Da quest'idea sta prendendo forma *Facebook Horizon* (ancora in fase beta)[5], che raccoglie l'eredità dell'originario *Facebook Spaces*. Esso si propone come mondo virtuale, dotato di strumenti per reinventare ed arricchire l'interazione sociale, sfruttando al contempo i contenuti generati dagli utenti per espandersi a sua volta.

Ciascun partecipante, dopo aver creato un avatar a partire dal proprio profilo Facebook, può incontrare i propri amici in un nuovo contesto virtuale e interattivo, in cui è possibile chattare, formare delle community, vedere video 360, giocare, disegnare e costruire in un ambiente 3d ed esplorare nuovi mondi, anche in modi non convenzionali (ad esempio a bordo di un aeroplano).

L'uscita dell'applicazione è prevista per il 2020 sulle piattaforme Oculus Rift e Oculus Quest.

Più legata invece al mondo performativo e degli eventi è *Oculus Venues*, l'applicazione ufficiale di Oculus per assistere virtualmente a spettacoli teatrali, eventi sportivi, concerti e proiezioni, per citarne alcuni.

L'utente può consultare un calendario di possibili appuntamenti e prendere parte alla performance, collegandosi all'evento live nel momento opportuno. L'esperienza immersiva può essere fruita in solitaria, oppure in mezzo ad altri fan tra il pubblico virtuale, con cui è possibile interagire, seppur in maniera abbastanza limitata.

[5] https://www.oculus.com/facebookhorizon/

Figura 7: Oculus Venues

2.1.2 AltspaceVR

Così come Facebook, anche Microsoft ha rivolto la propria attenzione al mondo della *social VR*, anche se con un atteggiamento, per così dire, meno proattivo ed innovativo.

L'azienda infatti ha acquistato nel 2017 *AltspaceVR*, salvandola dalla bancarotta[6] e mantenendo così viva una community già esistente e consolidata.

Figura 8: AltspaceVR

Il servizio offerto da *AltspaceVR* è un ibrido tra un modello performativo basato su eventi e un ambiente sociale virtuale.

[6] https://time.com/4967092/microsoft-altspacevr-virtual-reality/

Anche qui, una volta definito il proprio avatar, è possibile partecipare a spettacoli live assieme ad altri utenti, frequentare corsi, sessioni di meditazione, concerti e molto altro. Per il singolo utente, è anche possibile ospitare un proprio show, lezione o semplice incontro, aperto agli altri utenti.

2.1.3 RecRoom, VR Chat e Dreams VR

Un altro segmento di mercato decisamente rilevante e florido per la realtà virtuale multi-utente è quello videoludico.

All'interno di questo contesto, si va dalle soluzioni più *arcade*, ovvero giochi rapidi e immediati, con regole semplici e che si prestano alla ripetibilità; fino ad arrivare a paradigmi di interazione più complessi, in cui ad esempio si dà all'utente la possibilità di fare *world building*, creando a propria volta dei livelli che potranno essere sperimentati dagli altri giocatori.

Un esempio interessante di applicazione che in principio nasce come collezione di mini-giochi, ma che negli anni diventa via via un ecosistema virtuale più complesso è *RecRoom*.

Figura 9: RecRoom

Al tempo del lancio, nel 2016, l'esperienza appare abbastanza statica: gli utenti si possono incontrare in un *hub* centrale, possono comunicare tra loro e partecipare ad un numero limitato di esperienze di gioco (e.g. paintball, dodgeball etc.).

Tuttavia, guidati da una filosofia che accoglie le proposte degli utenti di rendere più sociale e collaborativa l'esperienza di gioco, gli sviluppatori lavorano in una direzione ben precisa: dare più potere di creazione all'utente finale. Questo con l'intento di creare un circolo virtuoso in cui la piattaforma cresce e si diversifica grazie allo sforzo della comunità, la quale contemporaneamente si percepisce come più libera e legittimata.

Una scelta coraggiosa, che però ha portato i suoi benefici all'applicazione, rendendo le *user generated rooms* i luoghi in cui mediamente il giocatore trascorre più tempo all'interno del mondo virtuale[7].

In questo senso, l'abilità nella gestione della crescità di *RecRoom* è data dal fatto che gli sviluppatori sono riusciti a trovare un equilibrio tra il mantenimento e la 'perdita' di controllo sul prodotto finale, evitando quindi abusi o incoerenze dovuti all'azione di agenti esterni, ma contemporaneamente assicurando piena libertà ai giocatori nell'uso degli strumenti forniti.

Un altro importante esempio di applicazione simile dal punto di vista delle possibilità di interazione offerte è *VR Chat*. Essa si propone di replicare il tipo di esperienza offerto da simulazioni come *Second Life* o *Habbo Hotel*, ma in realtà virtuale.

Figura 10: VR Chat

Il gioco offre grandi livelli di personalizzazione: attraverso il *software development kit* fornito, gli utenti possono addirittura caricare i propri modelli 3d e, dipendentemente dalla tecnologia del visore in proprio possesso, sfruttare diverse possibilità di movimento (dalla mappatura dei propri movimenti corporei sull'avatar virtuale, alla sincronizzazione del labiale, all'*eye tracking* e al controllo delle palpebre).

Al contrario di *RecRoom*, qui il controllo e la mediazione in merito a quanto accade nel mondo virtuale sono molto limitati, tanto che negli anni la piattaforma è stata oggetto di controversie di matrice razziale, a causa delle attività di alcuni gruppi di utenti[8].

Infine, dopo il successo ottenuto su console Playstation 4, anche il *sandbox* game *Dreams* approda su Playstation VR.

[7] https://www.wired.com/story/social-vr-worldbuilding/
[8] https://www.polygon.com/2018/1/8/16863932/ugandan-knuckles-meme-vrchat

Figura 11: Dreams

Il gioco, già vincitore di un Gamescom Award nel 2019[9], rappresenta probabilmente una delle massime espressioni del *world building* collaborativo.

Ciò che risalta nel gioco è senz'altro la coerenza artistica unita alla semplicità di utilizzo: attraverso l'uso di *gesture* e pennelli virtuali (con l'utilizzo di Playstation Move) è possibile riarrangiare e personalizzare il mondo di gioco in maniera estremamente intuitiva. Il giocatore potrà scegliere se partire da zero nella costruzione del suo mondo, o se utilizzare e modificare creazioni di altri utenti.

Per quanto *Dreams* possa sembrare a prima vista un'esperienza molto simile agli altri esempi sopracitati, ciò che lo rende davvero unico è la qualità generale della direzione artistica e del design dell'interazione, componenti che permettono anche ad un utente inesperto di creare mini-mondi godibili anche in brevi sessioni di gioco.

2.2 Esperienze VR co-localizzate

Per quanto siano meno comuni all'interno del panorama analizzato, le esperienze co-localizzate, ovvero quelle in cui vi è una fruizione dei contenuti nel medesimo tempo e nel medesimo luogo da parte di un gruppo di utenti, costituiscono il segmento di maggior rilevanza nello studio condotto.

Questo perché VERA nasce come evento performativo live, in cui vengono a mescolarsi esperienza virtuale e percezione fisica dello spazio, degli altri partecipanti e dei suoni circostanti. Le ragioni di tale scelta sono legate al tentativo di superare il senso di isolamento dal mondo e dalle persone, che tipicamente vive il fruitore di un'esperienza VR, anche multi-utente.

[9] https://www.everyeye.it/notizie/gamescom-awards-2019-vincitori-premiati-fiera-tedesca-394931.html

Vediamo ora alcuni esempi di VR co-localizzata che hanno, a loro modo, provato ad affrontare tale criticità.

2.2.1 VR Domes

Un primo esempio di esperienza immersiva condivisa è costituito dai *VR Domes*.

Si tratta essenzialmente di grandi spazi, rivestiti da una cupola dotata di un sistema di *projection mapping* a proiettore singolo o multiplo, che è in grado di renderizzare e proiettare video su tutta la superficie interna. Al contempo, un sistema di diffusione audio spazializzato garantisce un effetto sonoro pervasivo all'interno dell'ambiente.

Figura 12: VR 360 Dome

Gli utenti possono muoversi liberamente all'interno dell'ambiente e non necessitano di indossare un visore. Tale tecnologia ha mosso i primi passi in contesti scientifici, quali quello astronomico, per poi diventare negli anni anche una destinazione per contenuti di intrattenimento.

Per quanto un'installazione simile possa rivelarsi di grande effetto per il pubblico, i limiti nella messa in scena di una performance in un contesto del genere sono logistici e infrastrutturali: servono grandi spazi, tecnologie adeguate ed accorgimenti tecnici, di cui le produzioni a basso budget non sempre possono disporre.

2.2.2 Cave

Presentato come *technical demo* al SIGGRAPH 2018 e presentato l'anno successivo al Tribeca Film Festival, *Cave* costituisce un importante esempio di cortometraggio in VR condivisa [6].

Figura 13: Un frame di *Cave*

L'opera, realizzata dallo studio *Parallux*, con le partnership di Google, Bose, NVIDIA e del Future Reality Lab della New York University, è ambientata in un'epoca preistorica e la narrazione virtuale prende spunto dalle pitture rupestri, in un'atmosfera che ricorda un racconto attorno al falò, creando un ponte ideale tra le prime forme primitive di *storytelling* della tradizione orale e quelle virtuali contemporanee, che il pubblico vive attraverso il visore.

Figura 14: Il pubblico di *Cave*

Ogni sessione della performance dura circa 6 minuti ed è pensata per essere eseguita in una sala, ospitante fino a 30 utenti contemporaneamente, ciascuno con il proprio visore.

Durante l'esperienza, ogni utente vede i propri movimenti mappati sul proprio avatar e lo stesso discorso vale anche per quelli altrui. Così facendo, si acquisisce una maggiore coscienza spaziale all'interno dell'ambiente e viene incentivata la percezione della propria fisicità e di quella degli altri, rendendo l'esperienza più corale e semplificando i processi di sospensione dell'incredu-

lità.

Ovviamente, in contesti come quello descritto, la produzione dei contenuti è caratterizzata da un maggiore coefficiente di difficoltà, in quanto l'ambiente e la narrazione che si vanno a creare, non possono limitarsi a funzionare da un unico punto di vista, ma devono soddisfare tutti quelli possibili dal lato dello spettatore.

2.2.3 Flock

Rimanendo in ambito performativo, degno di nota è *Flock*, il lavoro presentato al SIGGRAPH 2017 e performato al *Future of Storytelling Festival* di New York [8].

Figura 15: Flock

L'opera, diretta da David Lobser, è un altro perfetto esempio di VR colocalizzata e condivisa, ma questa volta più legata alla musica. Si tratta infatti di un video musicale interattivo, ma anche, per dirlo nei termini usati dai suoi creatori, di un *Live Action Role Playing Game* (LARPG), ovvero di una sorta di gioco di ruolo in tempo reale.

I partecipanti, infatti, impersonano dei volatili e sono liberi di muoversi nello spazio fisico e di volare nell'ambiente virtuale, nutrendosi di piccole larve disseminate nella scena.

La performance si svolge in una sala, con 30 utenti per volta, in piedi e liberi di muoversi nello spazio circostante.

2.2.4 Holojam in Wonderland

Holojam in Wonderland costituisce il primo esempio di pièce teatrale in realtà virtuale, in cui gli attori ed il pubblico condividono il medesimo spazio performativo [3].

Figura 16: Utenti immersi nell'esperienza di *Flock*

L'opera, ispirata ad *Alice in Wonderland* di Lewis Carroll, è stata presentata all'edizione 2017 del Future of Storytelling Festival da alcuni ricercatori della New York University in collaborazione con lo studio Holojam.

Figura 17: Holojam in Wonderland

Sia gli utenti, che gli attori sono liberi di muoversi nella stessa stanza e di esplorare il mondo virtuale condiviso, entrando in contatto tra di loro, attraverso i reciproci avatar.

I ricercatori, per rendere possibile l'esperienza hanno dovuto gestire un sistema di tracking a bassa latenza per mappare le posizioni e i movimenti dei partecipanti dal mondo reale a quello virtuale. In più, è stato necessario sincronizzare grafica 3d e audio immersivo tra tutti i dispositivi dei partecipanti.

2.2.5 Segnale d'allarme - La mia battaglia VR

Come ultimo esempio rilevante, sempre in campo performativo, si è scelto di guardare all'interno dei confini nazionali, per capire meglio il livello di avanzamento tecnico e artistico dei produttori di contenuti VR, ma anche per capire il grado di interesse che tali esperienze possono suscitare nel pubblico a noi più prossimo.

A tal proposito, si è scelto di analizzare la trasposizione dello spettacolo teatrale *La mia battaglia* di Elio Germano e Chiara Lagani in realtà virtuale.

L'opera, che per l'occasione muta il suo nome in *Segnale d'allarme* è diretta dallo stesso Elio Germano e da Omar Rashid ed è prodotta in collaborazione con lo studio GoldVR di Firenze[10]. Si tratta di un film in realtà virtuale, della durata di circa un'ora, realizzato attraverso l'uso di camere Insta360 Pro, capaci di registrazioni dell'ambiente a 360 gradi.

Figura 18: Elio Germano durante la performance

Lo spettacolo consiste in una provocazione meta-teatrale volta a far ragionare il pubblico sulle deformità insite nell'odierna società dell'informazione: la discussione parte infatti da posizioni democratiche e di buon senso, per poi degenerare verso ideologie e fanatismi. Il pubblico, attraverso il visore, assiste al monologo di Elio Germano come se fosse seduto esattamente di fronte all'attore e in prima fila.

Il fatto di indossare un visore tutti assieme costituisce un elemento simbolico per una performance che vuole porsi come critica all'alienazione della nostra epoca, portando a conseguenze estreme l'estraniazione comunemente associata alle esperienze virtuali.

[10] http://goldvr.it/produzioni-vr/segnale-dallarme/

Figura 19: Il pubblico di *Segnale d'allarme*

2.3 Best e Worst Practice

Le esperienze analizzate sono state fondamentali per capire le potenzialità espressive della realtà virtuale in diversi contesti, ma anche i limiti tecnici e produttivi ad essa associati.

Dato che VERA si pone come attività di tipo performativo, più simile ad un concerto o ad uno spettacolo dal vivo, che a un social network o ad un videogioco, le esperienze co-localizzate del medesimo tipo hanno costituito l'ispirazione principale.

Questo perché si crede che il mezzo riesca a mostrare le sue più grandi potenzialità in contesti narrativi immersivi, in cui un utente può fruire di un'opera multimediale secondo modalità nuove, che lo coinvolgano in prima persona, senza però richiedergli un'eccessiva dimestichezza tecnica, come può invece accadere per un videogioco.

L'idea poi di sfruttare uno stesso spazio e tempo fisico (cfr. *Cave*, *Holojam in Wonderland* e *Flock*), in un contesto performativo condiviso, invoglia maggiormente anche gli utenti più scettici a provare, magari convinti da qualcun altro che ha già portato a termine l'esperienza.

Questo passaparola costituisce un aspetto positivo per la crescita della realtà virtuale in un'ottica più generale, dato che si tratta di un tipo di tecnologia da sempre accompagnata da grandi aspettative, ma che non ha ancora trovato la sua dimensione in un mercato *mainstream*.

Come sottolineato da Gabe Zetter, COO di Parallux: "Vediamo le applicazioni VR e AR co-localizzate come un'opportunità per portare le persone davanti al contenuto virtuale, senza che abbiano la necessità di dover comprare dei visori da loro [...] e facendo sì che esse associno l'esperienza VR e AR con qualcosa di positivo, speriamo di incoraggiarle a diventare promotori

del mezzo stesso[11]".

Inoltre, la possibilità di ricevere un feedback sulla presenza simultanea degli altri utenti nel mondo virtuale, contribuisce a rendere più coesa e credibile l'esperienza.

Dal punto di vista dell'interazione, bisogna dire che molto dipende dalla tecnologia del visore e dai gradi di libertà che esso offre. Essendo VERA una produzione a basso budget, ci si è concentrati maggiormente sulle soluzioni che adottavano hardware di fascia simile, cercando di trasformare il limite tecnico, in un vantaggio espressivo, votato alla semplificazione e all'intuitività.

A questo proposito, sono state viste con interesse le esperienze che non prevedessero l'uso del controller da parte dell'utente. Questo infatti, pur aumentando le possibilità di input generabili, dall'altro aggiunge ulteriori livelli di complessità nella produzione di contenuti, negli eventi da gestire, ma soprattutto nella fruizione da parte del pubblico generalista, che rischia di subire un'interruzione dell'immersività, se distratto da meccaniche che non comprende. In questo senso, il lavoro fatto da *Cave* è d'esempio.

Per quanto riguarda gli aspetti puramente grafici, gli ambienti creati da *Cave* e *Dreams*, sono sembrati quelli più godibili ed adatti ad una narrazione come quella di VERA, in quanto semplici, ma curati ed espressivi.

Da ultimo, per avere un'idea di quanto la durata possa condizionare la sensazione di comfort nella fruizione dell'esperienza, è stato interessante notare come, a detta dei creatori di *Segnale d'allarme*, il pubblico non abbia sofferto troppo il fatto che lo spettacolo durasse circa un'ora[12].

Ciò è probabilmente dovuto al fatto che lo show avviene in un ambiente virtuale statico, senza grandi cambiamenti nel mondo e senza che gli utenti possano muoversi in esso.

Dovendo però produrre un'esperienza per sua natura più variabile per l'utente, si è considerata una tale durata come un qualcosa di insostenibile per VERA.

[11] https://www.techradar.com/news/are-shared-experiences-the-future-of-virtual-reality

[12] http://www.cinecittalucemagazine.it/2019/08/31/segnale-dallarme-la-mia-battaglia-elio-germano-e-lo-spettacolo-vr/

3 Il progetto VERA

Nonostante la progressiva diffusione di dispositivi con costi sempre più bassi, ormai accessibili al consumatore medio, la realtà virtuale deve ancora trovare una sua dimensione nel mercato mainstream.

Le motivazioni principali sono legate, in primo luogo, alla scarsità di contenuti proposti, se si paragona l'offerta a quella di altre piattaforme e, in secondo luogo, dal fatto che molte delle esperienze proposte costringono l'utente ad isolarsi in un mondo artificiale, inibendo la sua percezione del mondo esterno e degli altri. Allo stesso tempo, gli osservatori esterni, privi di visore, sono automaticamente tagliati fuori dall'esperienza.

In questo senso, il limite, ma anche l'opportunità più grande per questo tipo di tecnologia, è il fattore sociale. Risulta quindi indispensabile, per i produttori di contenuti, sperimentare ed implementare nuove modalità di interazione e fruizione multi-utente, in modo che la realtà virtuale acquisti una dimensione maggiormente sociale e coinvolgente.

Detto ciò, l'industria dell'intrattenimento appare come il contesto più fertile per mettere in atto degli esperimenti in tal senso, ed è all'interno di questa cornice infatti che il progetto VERA intende svilupparsi.

3.1 Che cos'è VERA

Il progetto VERA (Virtual Experience Real Approach) nasce come performance multimediale live, in cui vengono a mescolarsi realtà virtuale, storytelling visivo e performance musicale dal vivo. L'idea originale di combinare un'applicazione fruibile con un visore e un concerto dal vivo è da attribuire ai musicisti e performer torinesi *The Sweet Life Society*, con la collaborazione dell'associazione *Fluxlab*. Il progetto è stato selezionato tra i vincitori dell'edizione 2018 del bando "ORA! Produzioni di cultura contemporanea", indetto dalla Compagnia di San Paolo.

L'obiettivo della performance è quello di portare gli utenti all'interno di un contesto immersivo condiviso, ma di far percepire loro contemporaneamente il mondo esterno, attraverso il suono prodotto dai musicisti in tempo reale, con un sistema di diffusione audio spazializzato.

Ciascun utente ha una percezione degli altri partecipanti nel mondo fisico (in quanto esperienza co-localizzata), ma anche una conferma virtuale della presenza altrui: ad ogni fruitore è infatti associato un avatar virtuale, visibile dagli altri.

Con questo particolare connubio virtuale e fisico, si intende esplorare una nuova modalità di performance artistica, suscitando nel pubblico sensazioni e stimoli nuovi, ancora poco esplorati nei contesti performativi più usuali.

Figura 20: Manifesto di VERA

Figura 21: Render per la pre-visualizzazione dell'esperienza

A livello concettuale e performativo, VERA è da intendersi come un contenitore che ospita diversi moduli o "capitoli" differenti, ciascuno performabile indipendentemente dagli altri, al fine di rendere l'opera più ripetibile, varia ed espandibile in futuro. Nella sua prima versione, sono stati prodotti tre diversi capitoli, legati ai concetti emotivi umani di "quiete", "rabbia" e "tristezza", che verranno affrontati nel dettaglio in una sezione dedicata.

3.2 Contesto, limiti e challenge

Produrre dei contenuti per la realtà virtuale non è semplice, in quanto è necessario far coesistere i requisiti narrativi con delle limitazioni tecniche assai più stringenti rispetto ad altre piattaforme.

In più, fare tutto ciò come team indipendente, alla prima esperienza con una produzione di questo tipo, aggiunge ovviamente diversi gradi di complessità.

Volendo riassumere i principali vincoli che è stato necessario affrontare durante la produzione di VERA, è necessario citare i seguenti punti:

- è una produzione a basso budget;
- il target di riferimento è un pubblico generalista;
- è una performance co-localizzata, multi-utente e live.

Le limitazioni derivanti dal primo punto si riflettono innanzitutto sulla dimensione contenuta del team: non disponendo infatti delle risorse tipiche di uno studio che opera a livello professionale, si è dovuto affrontare l'intera produzione 3D, così come lo sviluppo del codice applicativo, contando soltanto su due figure responsabili del segmento tecnico ed artistico.

In più, il limite economico ha determinato fortemente le scelte hardware e software: si optato infatti per visori *mobile* a basso costo e per una pipeline di produzione che includesse quasi esclusivamente software gratuito ed open source (cfr. Blender[13] e Unity[14]). Tali scelte hanno avuto poi un impatto anche sullo stile artistico che si è scelto di adottare, preferendo una soluzione più "stilizzata", piuttosto che una resa foto-realistica, visti i limiti legati alle performance.

Dal punto di vista del target di riferimento, dato che l'esibizione è pensata per essere fruita da un pubblico generalista, si è scelto di modellare l'esperienza semplificando il più possibile le meccaniche di interazione. Si è eliminato infatti l'uso del controller, per fare affidamento sulla meccanica di *gazing*, in modo che, anche per un utente inesperto, sia semplice interagire con il mondo attraverso un semplice movimento della testa.

Infine, la necessità di performare in condizioni live e con più utenti sincronizzati ha aggiunto un grado ulteriore di complessità, rendendo necessario definire un'architettura di rete appropriata, che garantisse una fruizione simultanea.

3.3 La soluzione hardware

Coniugando i limiti di budget e la necessità di gestire la disposizione di più persone in uno stesso ambiente condiviso, si è scelto di adottare dei visori *mobile standalone*, in modo da evitare soluzioni VR desktop con postazioni

[13] https://www.blender.org/
[14] https://unity.com/

dedicate ed ingombri derivanti dal cablaggio.

A questo proposito, il visore che si è deciso di utilizzare è *Oculus Go* [15]. Si tratta di una soluzione a 3DoF (*Degrees of Freedom*, letteralmente "gradi di libertà"), ovvero con una tecnologia di tracking che capta i movimenti di rotazione della testa sui tre assi (misurando gli angoli di *pitch*, *yaw* e *roll*) e li rimappa nella scena VR in maniera coerente, consentendo all'utente di sperimentare una visione a 360 gradi.

Figura 22: Oculus Go

I visori di fascia più alta (come ad esempio *Oculus Quest* ed *Oculus Rift*), dotati di 6DoF, consentono invece anche il tracking delle traslazioni del corpo nello spazio, oltre che delle rotazioni della testa, facendo sì che l'utente si sposti in maniera concorde nello spazio fisico e in quello 3D. Tale caratteristica aumenta notevolmente la sensazione di immersività e di comfort per l'utente [11], riducendo gli effetti indesiderati di *motion sickness*.

Nei dispositivi a 3DoF, l'inconveniente è che l'utente è costretto a rimanere in una posizione statica, con la sola possibilità di muovere la testa. Questo risulta particolarmente critico quando si vuole variare il suo punto di vista nello spazio virtuale, senza un cambiamento analogo nello spazio fisico. Tale condizione, come si vedrà meglio in seguito, ha costituito una delle sfide principali nel capitolo di Rabbia.

Tornando alle altre caratteristiche di Oculus Go, esso possiede un display LCD dalla risoluzione di 1280 x 1440, con una frequenza di aggiornamento a 60Hz. Il dispositivo possiede poi un sistema di riproduzione audio integrato, con anche la possibilità di collegare degli auricolari.

Il visore è anche dotato di un controller singolo, che tuttavia si è scelto di non utilizzare durante la fruizione della performance, come già accennato.

[15] https://www.oculus.com/go/

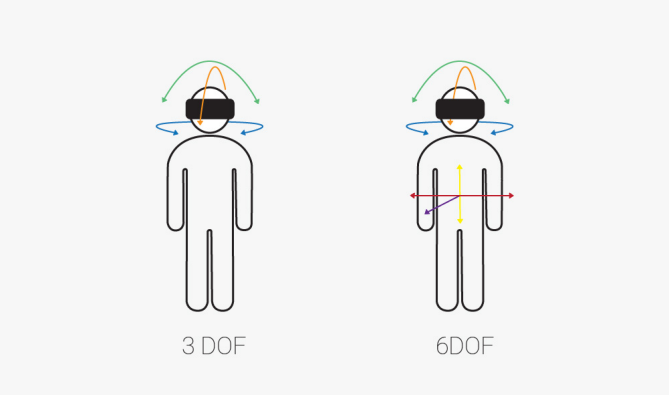

Figura 23: 3DoF vs. 6DoF

Uno degli aspetti positivi dei dispositivi Oculus, lato sviluppatore, è la possibilità di scaricare vari pacchetti, tool e SDK (Software Development Kit), compatibili con i vari *engine* di riferimento (come Unity o Unreal Engine), i quali permettono di semplificare il processo di sviluppo della propria applicazione.

3.4 Framework e soluzioni software

I principali tool di sviluppo software utilizzati sono, come già accennato, Blender (versione 2.83) come suite per la produzione di asset 3D e Unity (versione 2019.4 LTS) come *game engine*.

Tali scelte sono giustificate, innanzitutto, dal fatto che si tratta di programmi gratuiti ed *open source*; ma anche per la presenza di alcune feature interessanti ed innovative, in termini di *shading* e *rendering* del prodotto finale.

Di seguito, verrà fornita una panoramica sulla nuova pipeline di rendering per mobile sviluppata da Unity e sulle potenzialità offerte da *Shader Graph*, un nuovo sistema di shading "a nodi", che permette di ottenere risultati grafici interessanti, flessibili e ottimizzabili.

3.4.1 Universal Render Pipeline

Già da qualche anno, Unity ha iniziato una transizione verso un nuovo paradigma di rendering, che va sotto il nome di SRP (*Scriptable Render Pipeline*) [16]. Si tratta di un tipo di design per cui lo sviluppatore ha la possibilità di accedere al codice sorgente del motore di rendering e di editare un particolare *API layer*, attraverso la scrittura di codice C#, in modo da controllare

[16] https://github.com/UnityTechnologies/ScriptableRenderPipeline

con maggiore libertà il processo di resa grafica nella propria applicazione.

Tuttavia, dato che la riscrittura dei processi di rendering in una versione custom richiede competenze tecniche non necessariamente comuni tra gli sviluppatori, l'azienda ha lavorato con l'intenzione di fornire un framework ugualmente personalizzabile, ma che sia anche gestibile in maniera più intuitiva e senza che sia richiesta la conoscenza dei dettagli di elaborazione grafica di più basso livello.

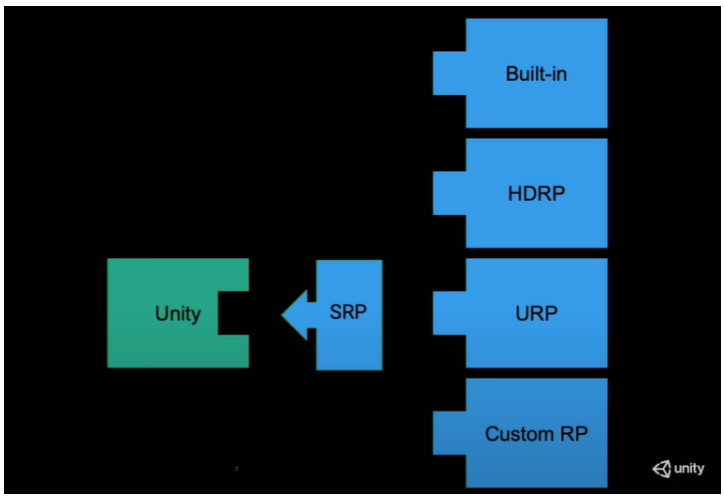

Figura 24: Il paradigma SRP

Con questo scopo sono nate:

- HDRP (*High Definition Render Pipeline*)
- URP (*Universal Render Pipeline* - precedentemente LWRP)

Si tratta essenzialmente di due template di progetto (costruiti sfruttando il paradigma SRP), aventi dei settaggi di rendering preconfigurati, che è possibile modificare secondo le proprie esigenze, attraverso l'interfaccia di Unity. Funzionano in modalità plug-and-play e senza la necessità di scrivere codice complesso.

La HDRP è dedicata ad applicazioni destinate a girare su hardware di fascia alta, come gaming pc o console ed è pensata per un tipo di rendering di stampo foto-realistico, più dispendioso dal punto di vista del calcolo. Come è facile intuire, non risulta essere adeguata per un progetto come VERA.

La URP, invece, è il nuovo standard di rendering per le piattaforme hardware di fascia più bassa, quali ad esempio i dispositivi mobili o quelli VR. Essa nasce per garantire le massime performance, cercando di sacrificare il

meno possibile in termini grafici, ma senza ambire alla qualità raggiungibile con la HDRP. È possibile utilizzare la URP dalla versione di Unity 2019.4 LTS in poi.

Il tipo di rendering path attualmente supportato dalla URP è di tipo *forward* e i calcoli di shading per le luci vengono effettuati in modalità *single-pass*, ovvero calcolando tutte le luci in un unico passaggio e senza aggiungere chiamate aggiuntive alla scheda grafica. In un contesto di sviluppo VR mobile, si ha un limite di:

- 8 luci per singolo oggetto;
- 32 luci per camera.

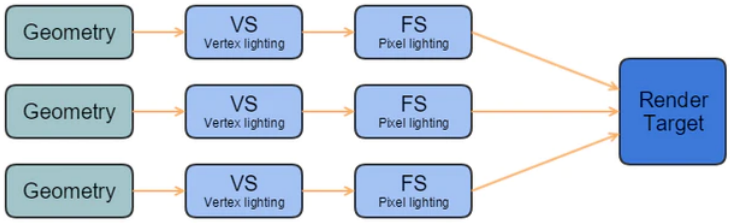

Figura 25: Forward Renderer

La configurazione della URP è abbastanza semplice: è sufficiente creare un nuovo *Universal Render Pipeline Asset* nel proprio progetto Unity. Attraverso tale asset, sarà possibile gestire tutte opzioni di rendering, quali gestione delle luci, delle ombre e degli effetti di post-procesing ad esempio.

Inoltre, è possibile gestire una *Renderer List* in cui si possono specificare diverse configurazioni e preset con qualità differenti, con la possibilità di testare condizioni diverse di resa grafica in semplicità e senza dover cambiare tutti i parametri di volta in volta.

Oltre a quanto già descritto, l'aspetto forse più interessante dell'adozione della nuova pipeline di rendering, è legato alla presenza di nuovi strumenti per la creazione di shader ed effetti visivi, totalmente integrati con il nuovo paradigma: *Shader Graph* e *VFX Graph*.

Durante la produzione di VERA, il primo è stato utilizzato in gran parte, perciò verrà discusso di seguito.

3.4.2 Shader Graph

Uno *shader* è essenzialmente un algoritmo, scritto in un linguaggio comprensibile alla GPU, che determina la resa grafica di un oggetto in uno spazio 3D, tenendo in considerazione le proprietà intrinseche della sua superficie e

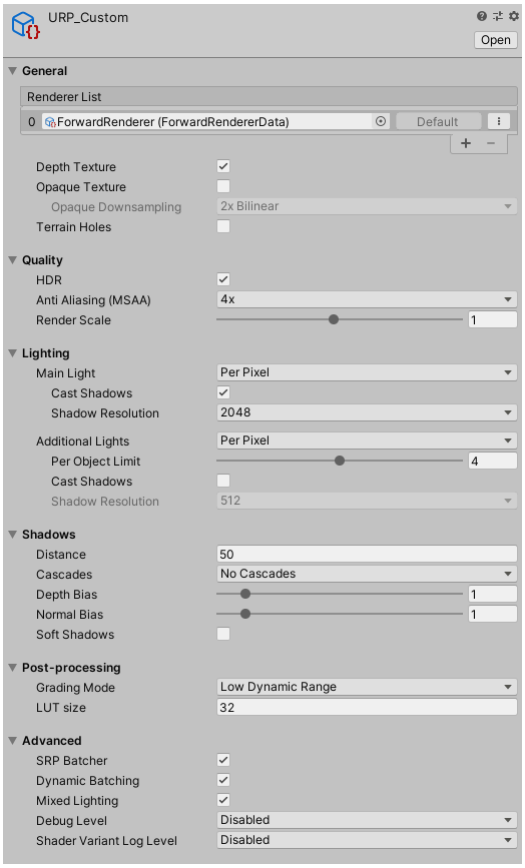

Figura 26: Opzioni dello URP Asset

del modo con cui interagisce con le fonti di illuminazione presenti in scena. I calcoli che determinano tale comportamento sono poi incapsulati in un contenitore dall'interfaccia più facilmente gestibile nell'applicazione 3D: il *materiale* (che può contenere informazioni di colore, texture etc.).

Per passare dalle proprietà della geometria 3D, fatta di vertici e triangoli, al colore finale dei pixel occupati dall'oggetto sullo schermo 2D, è necessario attraversare diversi stadi di elaborazione grafica, ciascuno adibito ad un compito diverso. Tra questi, quelli comunemente più noti e manipolati dai programmatori di API grafiche sono:

- il *Vertex Shader*, in cui vengono gestite le proprietà del singolo vertice, producendo le sue coordinate finali in *screen space*;
- il *Geometry Shader*, in cui si possono effettuare calcoli vettoriali sulla base delle informazioni geometriche del modello (es. calcolo delle normali);

- e il *Fragment Shader*, in cui, successivamente alla rasterizzazione, vengono processate le informazioni finali per donare il colore ai pixel visibili (ovvero non occlusi, confrontando l'informazione sulla *z-depth*).

Nelle schede grafiche più recenti viene adottato un nuovo modello a "shader unificati" (*Unified Shader Model*), che permette di utilizzare lo stesso instruction set di primitive per tutti e tre gli stadi, lasciando decidere alla GPU che cosa vada eseguito a runtime.

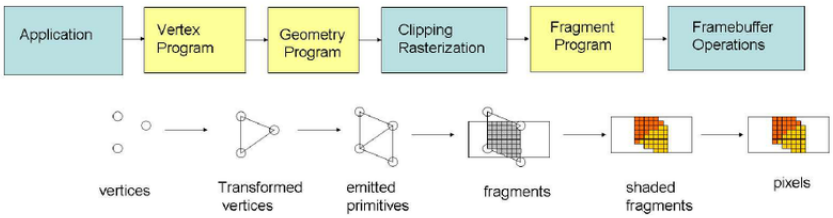

Figura 27: Un esempio di pipeline grafica

Come si può intuire già da questi primi cenni, l'operazione di scrittura degli shader via codice richiede un'elevata competenza tecnica e rischia di diventare un'operazione molto complessa, anche per realizzare materiali semplici.

A questo proposito, la URP mette a disposizione diversi shader di default[17], con cui è possibile iniziare a creare i propri materiali:

- Lit - basato su un modello PBS (*Physically Based Shading*, ovvero basato su calcoli come la conservazione dell'energia e l'effetto Fresnel, che approssimano un comportamento fisicamente corretto della luce). È il più completo e ricco, ma anche il più dispendioso.
- Simple Lit - non è PBS, in quanto semplifica il modello di interazione con le sorgenti luminose, tuttavia è indicato per le situazioni in cui il foto-realismo non è essenziale e in cui contano le performance. A questo proposito risulta adatto per contesti mobile come quello di VERA.
- Baked Lit - è pensato per non essere influenzabile dalla luce realtime, ma solamente dalla luce *baked* (via *lightmaps*) e dai *light probes*. È molto performante, in quanto prevede un calcolo non realtime, ma richiede un processo di configurazione corretto.
- Unlit - utilizzabile per tutti quegli oggetti che non necessitano di interagire con nessun tipo di illuminazione (ad esempio per oggetti dall'aspetto emissivo). Molto efficiente.

[17] https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@9.0/manual/shaders-in-universalrp.html

- Particle Shaders (Lit, Simple Lit, Unlit) - si tratta di vari shader, con proprietà analoghe a quelle descritte sopra, destinati agli elementi dei sistemi particellari.

Tali soluzioni costituiscono un buon punto di partenza, ma non esprimono appieno tutte le possibilità offerte dalla nuova pipeline.

Infatti, potrebbe essere desiderabile voler creare un materiale animato per rappresentare, ad esempio, l'acqua di una cascata, o l'influenza del vento sulla vegetazione, oppure il particolare effetto di scomparsa di un oggetto. O ancora, da un punto di vista più pragmatico, si potrebbero voler sfruttare i vantaggi derivanti dalla creazione procedurale di materiali, la quale aumenta la velocità e la semplicità di iterazione per il raggiungimento dell'estetica finale.

Ciò è soprattutto utile in un contesto in cui vi è la necessità di provare rapidamente diverse soluzioni e di sottoporle agli altri membri di un team: potendo intervenire rapidamente sull'aspetto del materiale modificando dei semplici parametri esposti nell'interfaccia, il processo risulta drammaticamente più efficiente.

Tutto questo, con *Shader Graph*[18] è un processo molto agevole, una volta che si è entrati nella logica di una programmazione visuale "a nodi". Per ottenere il comportamento desiderato per il proprio materiale, sarà necessario collegare coerentemente i diversi blocchi funzionali tra di loro, portando l'output, a valle dell'elaborazione, ad un particolare nodo *Master*. In Shader Graph esistono due tipi di nodi di questo tipo: *Unlit* (più semplice e non sensibile alla luce) e *PBR Master* (contiene opzioni per uno shading fisicamente accurato).

Le possibilità ottenibili con tale paradigma sono pressoché illimitate, a patto ovviamente di conoscere le possibilità offerte dai vari nodi. Si possono usare ad esempio nodi per manipolare le texture, i vertici dell'oggetto, le normali, le proprietà di colore e di risposta alla luce, la trasparenza e via dicendo. Un grande vantaggio è dato dalla finestra di preview, che permette di vedere le modifiche al proprio materiale in tempo reale. In più, è possibile convertire un qualsiasi nodo (fatte alcune eccezioni) in una *property*, che può poi essere esposta e modificata direttamente nell'editor, oppure accedendovi via codice.

3.5 Architettura di rete

Il modello di rete adottato per VERA è una semplice architettura client-server, in cui:

[18] https://unity.com/shader-graph

- un elaboratore centrale opera come server, garantendo la sincronizzazione delle scene 3D e dell'audio tra i vari utenti e gestendo le azioni globali;
- ogni visore VR rappresenta un client, con la propria copia della simulazione, che viene periodicamente sincronizzata con quanto avviene sul server.

Poiché si è ipotizzato fin dall'inizio che il volume del pubblico partecipante all'esperienza potesse essere variabile, uno dei requisiti principali ricercati nella soluzione di networking è stata la *scalabilità*.

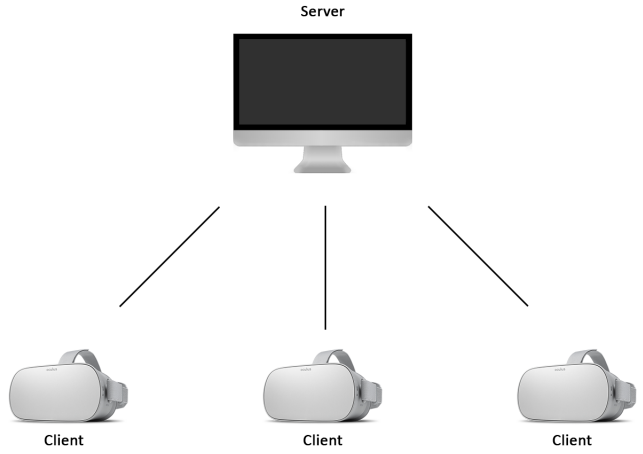

Figura 28: Architettura di rete

Di conseguenza, si è scelto di utilizzare *Mirror*[19], una networking API ad alto livello, costruita a partire dalla liberia *Telepathy* di più basso livello. Tale libreria è costruita e testata per funzionare in contesti come i giochi MMO (*Massive Multiplayer Online*), perciò garantisce stabilità ed affidabilità in situazioni real-time multi-utente.

Il vantaggio di tale soluzione è dato dal fatto che il codice per gestire il client e quello per gestire il server, lato sviluppatore, sono quasi sovrapponibili, con le principali differenze gestite internamente dalla libreria. Questo rende il codice più semplice da scrivere e da manutenere.

[19] https://github.com/vis2k/Mirror

Per ragioni di affidabilità nella sincronizzazione, si è scelto di connettere i client al server con una connessione di trasporto di tipo TCP.

4 Il primo spettacolo di VERA

Il concetto attorno al quale ruota l'impianto narrativo di VERA è l'esplorazione delle emozioni umane.

Tale scelta è motivata, in primo luogo, dal fatto che un concept simile si presta bene ad essere affrontato con un approccio *modulare*, o a capitoli. Questo paradigma seriale rende possibile suddividere VERA in segmenti temporalmente più brevi e indipendenti tra loro, senza sovraccaricare l'utenza con un'unica opera monolitica dalla durata potenzialmente eccessiva per il pubblico più sensibile. Inoltre, ciò aumenta la ripetibilità e la variabilità della performance, in modo che un utente, che sperimenta un determinato capitolo in un determinato contesto, possa essere invogliato a provarne un altro in un momento differente.

In secondo luogo, la narrazione a moduli costituisce un vantaggio dal punto di vista produttivo, in quanto il contenuto dell'opera è potenzialmente scalabile anche in prospettiva futura, qualora si decidesse in seguito di lavorare ad uno o più nuovi capitoli.

Delle ventisette categorie emotive individuate dai ricercatori della University of California [1], sono state scelte le emozioni di Quiete, Rabbia e Tristezza per costituire il nucleo narrativo degli altrettanti capitoli della prima versione di VERA.

Di seguito, ciascun capitolo sarà analizzato nel dettaglio, partendo dal concept, fino ad arrivare agli aspetti tecnici di produzione più interessanti.

4.1 Quiete

Il capitolo di Quiete, è stato concepito come una trasposizione in realtà virtuale di una sessione di *biosonologia* e di ascolto profondo di toni binaurali. Si tratta di una sorta di meditazione guidata dal suono, diretta da Domenico Sciajno, professore del conservatorio di Torino e fondatore dell'Istituto di Biosonologia di Palermo[20].

I battimenti binaurali sono un fenomeno acustico che avviene quando un ascoltatore è esposto contemporaneamente a due suoni simili, ma non coincidenti, con una differenza di frequenza inferiore ai 30 Hz (soglia al di sopra della quale vengono percepiti come due toni distinti) [9]. Quando ciò accade, si è osservato che, nonostante a ciascun orecchio arrivi fisicamente una frequenza diversa, il risultato psico-acustico restituito dal cervello è la percezione di un unico suono modulato, simile ad un battimento acustico.

L'obiettivo dell'esperienza è quello di accompagnare il pubblico nell'ascolto profondo di questi particolari suoni e di farlo abbandonare ad uno

[20] http://www.biosonologia.it/Biosonologia/Biosonologia.html

scenario visivo e acustico che cambia, arrivando infine al raggiungimento della quiete interiore.

Questo viaggio sonoro si articola in tre parti principali: in una prima sezione introduttiva, gli utenti vengono guidati dal prof. Sciajno nell'immedesimazione con una figura virtuale umanoide (che per ogni utente rappresenta l'immagine di sé), situata in uno spazio sospeso ed attraversata da una rappresentazione visiva delle onde sonore, a diversa frequenza, che vengono percepite.

Dopo una prima fase di sincronizzazione guidata del respiro e di visualizzazione sonora, si arriva ad un climax acustico che coincide con la scomparsa della figura umanoide e con l'inizio della seconda parte.

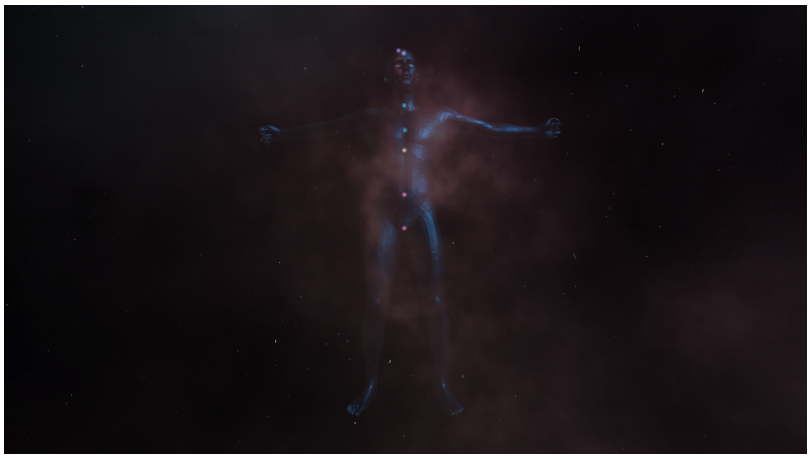

Figura 29: Estratto dalla prima parte di Quiete

La sezione di mezzo è caratterizzata dall'abbandono degli utenti alle sensazioni sonore evocate dal prof. Sciajno, che non interviene più verbalmente. Graficamente, in questa sezione gli utenti hanno l'illusione di trovarsi all'interno di un *wormhole* che cambia dinamicamente trama e colori, in accordo con la traccia sonora. In questo segmento, gli utenti sono sottoposti a stimoli di maggiore intensità, essendo immersi in una specie di caos controllato, volto ad incentivare la costruzione di una tensione emotiva, che sarà propedeutica alla fase di rilassamento finale.

La parte conclusiva del capitolo è infatti dedicata alla decompressione e al rilascio della tensione. Gli utenti infatti arrivano, dopo un secondo climax, alla fine del wormhole e si ritrovano in uno spazio etereo aperto, liberi di osservare e di ascoltare l'ambiente.

Quiete, dal punto di vista dell'utente, costituisce il capitolo più "passivo" dell'intera produzione di VERA, dato che l'aspetto grafico è subordinato alla sessione di ascolto biosonologico. In questo senso, si è pensato che l'aggiunta di meccaniche controllabili da parte dell'utente avrebbe potuto distrarre dal

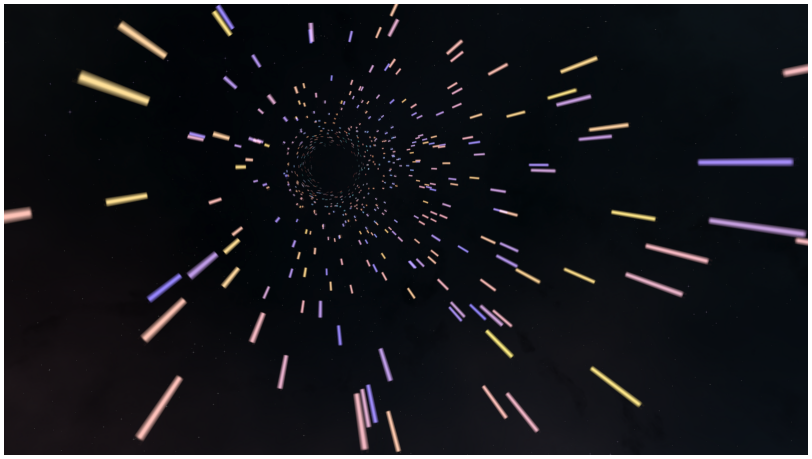

Figura 30: Estratto dalla seconda parte di Quiete

raggiungimento della concentrazione necessaria all'ascolto profondo. Perciò, si è scelto di usare la realtà virtuale con uno scopo semplicemente evocativo e di supporto alla performance audio, destinando le dinamiche più interattive agli altri capitoli.

4.1.1 Aspetti tecnici

A livello produttivo, le principali sfide che è stato necessario affrontare nella produzione di Quiete sono state essenzialmente due:

- la temporizzazione degli eventi;
- la creazione dell'effetto "wormhole".

Come già accennato infatti, il capitolo è costituito da tre sezioni ben distinte, ciascuna caratterizzata da una propria durata, da una propria dinamica e da eventi interni che hanno la necessità di essere schedulati in un determinato modo, in accordo con la traccia sonora.

Si capisce dunque, la necessità di trovare un paradigma per la gestione deterministica e temporizzata degli eventi e delle animazioni.

Perciò, nelle fasi iniziali di produzione, si è deciso di analizzare le potenzialità offerte da un potente strumento interno a Unity: la *Timeline* [21].

Essa si presenta come una normale interfaccia di gestione temporale basata su tracce, praticamente identica a quella di un qualsiasi software di editing video. Una traccia è una "riga" della timeline e può ospitare una o più clip, ovvero elementi temporalmente riproducibili, da un inizio a una fine.

[21] https://docs.unity3d.com/Packages/com.unity.timeline@1.5/manual/tl_about.html

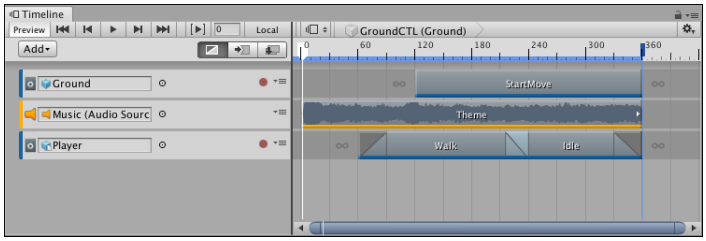

Figura 31: La Timeline di Unity

Per far sì che la propria scena in Unity sia gestibile attraverso una timeline, è necessario creare e configurare opportunamente due componenti:

- un *Timeline Asset*, ovvero un file di progetto che conterrà informazioni statiche su tracce, clip e animazioni per la specifica timeline (come una sorta di contenitore);
- e una *Timeline Instance*, ovvero un Gameobject vuoto con un componente di tipo *Playable Director*, che si occuperà di linkare (*binding*) le varie tracce con gli oggetti di riferimento da animare nella scena.

Grazie a questa separazione logica, è possibile riutilizzare uno stesso Timeline Asset con diverse Timeline Instance.

Rispetto ai tradizionali software di montaggio video, l'unica vera differenza, in Unity, risiede nelle tipologie di tracce che è possibile controllare. I tipi supportati dalla Timeline sono i seguenti:

- Activation Track, servono per abilitare e disabilitare il Gameobject associato;
- Animation Track, utili per gestire animazioni;
- Audio Track, per la regolazione delle tracce audio;
- Control Track, leggermente più complesse, servono per gestire sistemi particellari, per istanziare oggetti e per gestire eventi legati al tempo associati al Gameobject collegato alla traccia;
- Playable Track, interfaccia più generica che serve per gestire delle clip custom.

Da un punto di vista operativo, nonostante tali tipologie risultino sufficienti per gestire gli eventi più semplici e comuni (come, ad esempio, controllare la riproduzione di un'animazione già esistente, o attivare e disattivare un Gameobject), esse risultano abbastanza inadeguate quando si vuole un controllo più fine, ad esempio sull'animazione di un materiale, di una luce

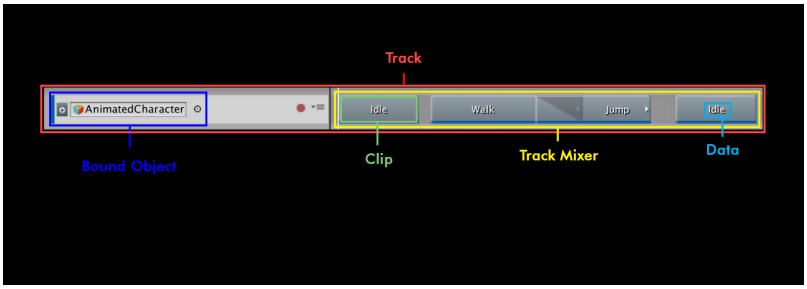

Figura 32: Struttura di una Track

o sull'interpolazione di un particolare parametro di scripting tra due clip diverse.

In questo senso, se si osserva ciò che viene fornito "out-of-the-box", la Timeline di Unity si può dire limitata, ma, al contempo, è anche estremamente flessibile ed estendibile se si ha la volontà di studiarne le API e di provare a modificarne il funzionamento via codice.

Custom Playables

Per estendere le funzionalità della Timeline, si è manipolato il paradigma specificato dalla *Playables API*. Attraverso la scrittura di quattro diversi script, è possibile creare delle tracce e delle clip personalizzate per un qualsiasi tipo di Gameobject o Component. Inoltre, è possibile esporre qualsiasi parametro animabile ed effettuare delle interpolazioni tra più clip sulla stessa traccia.

Con un minimo di configurazione, è possibile gestire il timing della propria scena in maniera estremamente flessibile, rendendo il lavoro di iterazione ed editing con il team molto più rapido ed efficace.

Gli script che è necessario creare sono i seguenti:

- un `TrackAsset` script, che permette di creare tracce di tipo personalizzato nella timeline;
- un `PlayableAsset` script, che consente di creare delle clip riproducibili nella traccia custom;
- un `PlayableBehaviour` script, che serve per comunicare a Unity l'azione da svolgere, ad ogni movimento del cursore, quando viene riprodotta la clip;
- un altro `PlayableBehaviour` script, opzionale, che funge da "mixer" per gestire l'interpolazione di più clip successive sulla stessa traccia.

Verrà preso come caso d'esempio la gestione temporizzata del parametro di comparsa e scomparsa dell'avatar umanoide nella prima parte del capitolo. Per esigenze narrative, infatti, la figura umana con cui gli utenti sono portati ad identificarsi, si forma gradualmente, a partire da un anello luminoso che ne crea prima solo la testa e, in un secondo momento, tutto il resto del corpo. Al termine della prima parte poi, la figura torna a dissolversi dai piedi alla punta del capo.

Per ottenere un simile comportamento, è stato creato uno shader ad hoc, con un parametro di dissolvenza verticale, che controlla l'altezza dell'anello luminoso e dunque della formazione della figura.

Figura 33: Apparizione controllata della figura umana

Per controllare tale parametro in maniera precisa e temporizzata è stato necessario implementare i quattro script sopracitati, in una versione personalizzata.

Dapprima, è stato creato un `ShaderControlTrack` script, come estensione della classe `TrackAsset`, che rende possibile creare una traccia personalizzata, specificando il particolare tipo di oggetto di scena richiesto per effettuare il *binding* (in questo caso, trattandosi di un materiale, è richiesto che l'oggetto possieda un componente di tipo *Renderer*).

In questo modo è possibile creare una traccia sulla timeline, ma non è ancora possibile riprodurre nulla, in quanto non esiste nessuna clip. Per ovviare a ciò, viene creato un `ShaderControlClip` script, che estende la classe `PlayableAsset`. Qui, vengono indicati i parametri che saranno esposti in ciascuna clip, nell'editor di Unity, i quali saranno utili per modificare operativamente le proprietà del materiale. Il parametro più interessante, nel nostro caso, è *Dissolve Amount*, responsabile del controllo verticale dell'effetto di dissolvenza.

Tuttavia, è necessario ancora un passaggio per rendere effettive le modifiche indicate sulla singola clip: essa infatti non è che un contenitore di

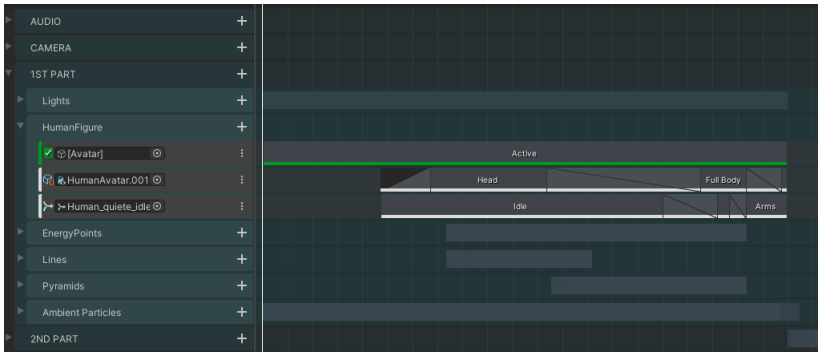

Figura 34: Track per il controllo dello shader della figura umana

parametri, i cui valori vanno poi passati ad un altro script, responsabile dell'attuazione della logica di esecuzione in fase di riproduzione.

Nello specifico, ciò è svolto dallo `ShaderControlBehaviour` script (che estende `PlayableBehaviour`). Esso ha la funzione di ospitare il dato indicato nella clip e di elaborarlo nel metodo `ProcessFrame`, chiamato, appunto, ogni frame. Perché tutto funzioni correttamente, è necessario referenziare tale script all'interno del metodo `CreatePlayable` della clip, e aggiornare adeguatamente le variabili interne del `PlayableBehaviour` con i valori specificati nella clip, tramite l'interfaccia grafica.

A questo punto, è possibile aggiornare lo script responsabile della track, in modo che accetti questo nuovo tipo di clip personalizzate (utilizzando l'indicazione `TrackClipType(typeof(ShaderControlClip))`).

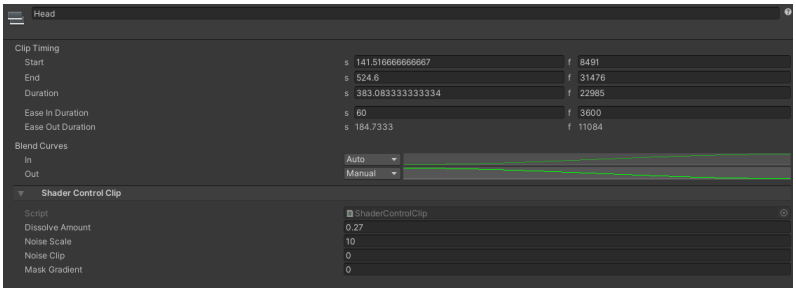

Figura 35: Clip custom per il controllo della dissolvenza con il parametro *Dissolve Amount*

In questo modo, è possibile creare una traccia, aggiungere delle clip del tipo desiderato e riprodurle, vedendo cambiare il loro comportamento sulla base di quanto indicato nell'editor.

Però, nel caso in cui si volesse effettuare un'interpolazione più o meno graduale tra i valori di una clip e la successiva, in modo da ottenere un effetto di *blending* dell'animazione, è richiesto un ultimo passaggio.

Va infatti messo a punto un track mixer, che permetta di controllare la sovrapposizione di due diverse clip, decidendo in che misura l'una e l'altra influenzano il valore finale dei paramatri, in un dato istante temporale. In sostanza, viene trasferito il controllo del comportamento in fase di riproduzione dalle singole clip, all'intera track che le ospita.

Viene dunque creato un `ShaderControlTrackMixer` script (sempre derivante da `PlayableBehaviour`) e si trasferisce la logica presente nel metodo `ProcessFrame` in questo script, anziché in `ShaderControlBehaviour`. Inoltre, bisogna comunicare al proprio `ShaderControlTrack` script che si intende usare il mixer appena creato (sovrascrittura del metodo `CreateTrackMixer`).

Va detto che questi passaggi sono opzionali, poiché non è detto che sia sempre richiesta un'interpolazione tra le clip: nel caso in cui non sia un requisito necessario, si può anche evitare di aggiornare gli script.

Il mixer, ciclando tra tutte le varie clip, consente l'interpolazione dei valori, pesandoli attraverso il parametro `inputWeight`.

Al termine di tutti questi passaggi, è finalmente possibile gestire la comparsa e la scomparsa dell'avatar semplicemente creando delle clip in timeline con determinati valori nei parametri e aggiustando le durate e le transizioni a piacere.

Tale aspetto ha costituito un vantaggio enorme per il lavoro con gli altri membri del team, grazie al feedback visivo immediato e all'intuitività dell'approccio di iterazione, durante le diverse prove per la sincronizzazione.

Effetto wormhole

Un secondo aspetto degno di menzione durante lo sviluppo di Quiete, è quello legato al tunnel spazio-temporale, presente nella seconda sezione del capitolo.

I principali requisiti iniziali erano:

- *variabilità*, infatti, trattandosi di una parte abbastanza lunga del capitolo, era necessario che non risultasse monotona, ma in costante evoluzione;
- *controllo della dinamica*, in modo da garantire accelerazioni, decelerazioni e cambi di ritmo, in accordo con la traccia audio;
- *controllo fluido delle transizioni*, per percepire un'esperienza *seamless*, senza discontinuità.

L'approccio che si è scelto di utilizzare per la creazione di un simile effetto animato ha coinvolto la scrittura di un *Wormhole Shader* ad hoc con ShaderGraph e l'estensione della Timeline per la gestione temporizzata di tale shader, in modo analogo a quanto già citato per la figura umanoide nel paragrafo precedente.

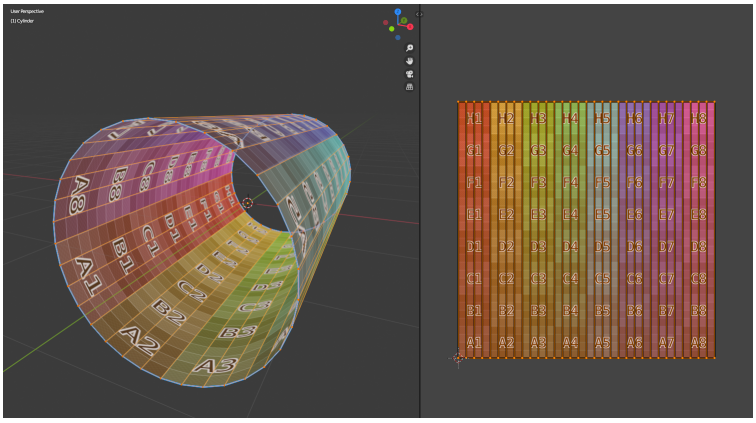

Figura 36: Modellazione e UV mapping del tunnel

Per prima cosa, è stato necessario creare un modello specifico per il tunnel, che rendesse possibile sfruttare la tecnica di *UV Scrolling*, per animare lo shader. Si tratta essenzialmente di creare un cilindro senza le faccie di base, con le normali rivolte verso l'interno e con delle coordinate UV proiettate in maniera longitudinale in accordo con la direzione delle facce laterali (Fig.36).

In questo modo, facendo scorrere una texture in direzione della coordinata v, ovvero variando l'offset in verticale, si ha un effetto di scorrimento, con una conseguente illusione di movimento, se ci si trova all'interno del cilindro.

Cambiando invece il *tiling*, ovvero la ripetizione della texture, si possono variare sia la rarefazione longitudinale, sia la ripetizione radiale, ottenendo pattern molto diversi anche con una singola texture.

In tutto, nello shader finale sono state utilizzate quattro texture differenti, con la possibilità di passare gradualmente dall'una all'altra attraverso dei parametri di blending e si sono utilizzati i valori di tiling e offset per variare la velocità di scorrimento e la ripetizione dei pattern.

Inoltre, è stata anche messo a punto un controllo sul gradiente iniziale e finale del tunnel, in modo da variarne percettivamente la profondità. Per ottenere un controllo simile è stata sfruttato il nodo *Position*, che fornisce la posizione nello spazio di ciascun vertice. Se si separa (con un nodo *Split*) la singola componente sull'asse z locale, si ottiene un gradiente naturale che va dall'origine dell'oggetto a valori di z crescenti. Controllando l'esponente di un nodo *Power* (elevamento a potenza), è possibile rendere il gradiente più o meno ampio.

Infine, per variare il colore delle texture, si è scelto di far variare nel tempo l'ingresso di un nodo *Hue*, che cambia tonalità ciclicamente, percorrendo la ruota dei colori.

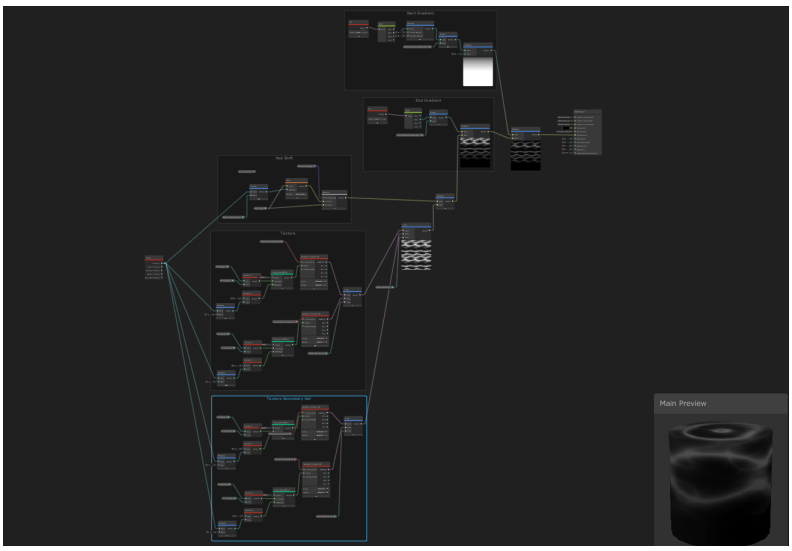

Figura 37: La struttura del Wormhole Shader in ShaderGraph

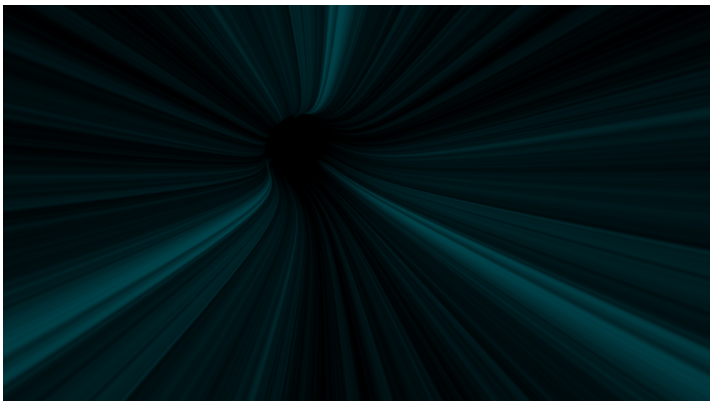

Figura 38: Esempio di effetto per il wormhole

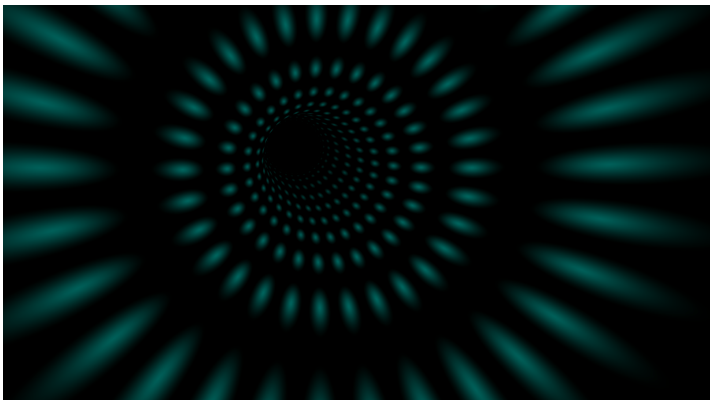

Figura 39: Altro effetto per il wormhole

4.2 Rabbia

Il secondo capitolo invece ruota attorno all'emozione della Rabbia. Il contesto narrativo in cui ha luogo l'azione è quello di un ambiente urbano sci-fi, dai tratti distopici. Gli utenti, accompagnati da un robot-guida, sono invitati sinistramente ad addentrarsi sempre più nei meandri della città di *V-Rage*, come in una moderna discesa negli inferi.

Il viaggio di Rabbia ha inizio in un tunnel di metropolitana, che viene percorso da ciascun utente a bordo di una capsula fluttuante, dotata di una plancia di comando con due pulsanti e di uno schermo olografico, sul quale appariranno delle domande, a cui ciascuno sarà libero di rispondere.

Una volta giunti alla fine del tunnel, ci si troverà in uno spazio periferico aperto, da cui è possibile scorgere per la prima volta la città di *V-Rage* in lontananza.

Proseguendo la propria corsa, si rientra nel tunnel di metropolitana e si incontra una stazione, in cui si iniziano a vedere alcuni avatar, abitanti del luogo, in condizioni di degrado. Oltre a questo, sulle pareti della stazione è possibile scorgere vari cartelloni e *billboard* con messaggi di propaganda.

Figura 40: Il tunnel che apre il capitolo di Rabbia

Dopo la stazione, si sbuca nel cuore di *V-Rage*, il luogo più caotico e centrale del capitolo. Si procede per le vie della città, fino a raggiungere una piazza monumentale. In questa sezione, agli utenti sono forniti stimoli visivi, in accordo con i temi sociali delle domande a loro sottoposte nella prima parte del capitolo. Vi è perciò una sorta di restituzione grafica delle problematiche sociali indagate a livello narrativo.

Arrivati idealmente al climax di rappresentazione della rabbia nella piazza, si passa ad un'atmosfera più distesa ed eterea, giungendo nel *Limbo*. In questo ambiente, si chiuderà il capitolo svelando agli utenti la dina-

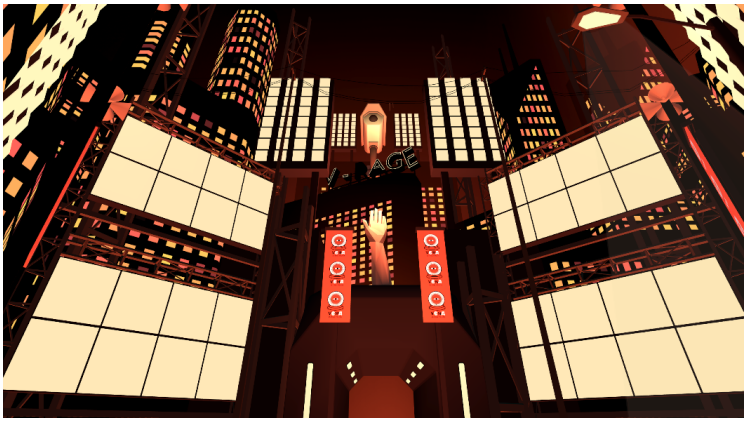

Figura 41: La città di *V-Rage* in Rabbia

mica di generazione della rabbia alla base dell'esperienza, fornendone una giustificazione psicologica.

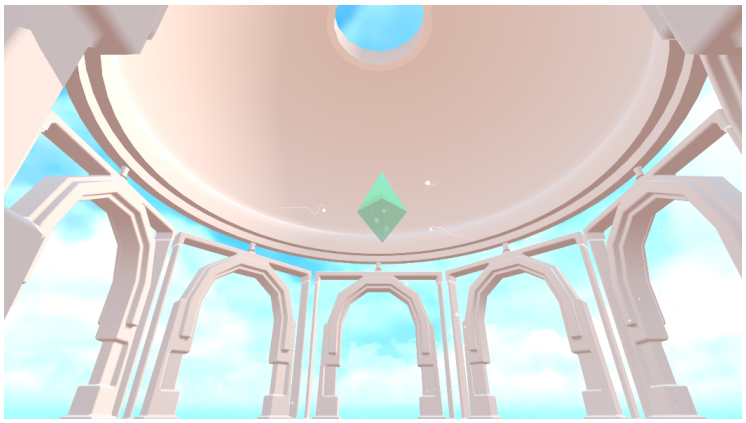

Figura 42: Il Limbo in Rabbia

Per la realizzazione degli ambienti e dell'atmosfera generale del capitolo, ci si è ispirati a titoli come *Inside* dello studio Playdead[22], a *Virtual Virtual Reality* dello studio Tenderclaws[23] e a *Overland* di Finji[24].

4.2.1 Aspetti tecnici

Il capitolo di Rabbia si è rivelato senz'altro il più complesso, sia per la sua struttura intrinseca, costituita da ben cinque scene da raccordare tra loro, sia per la quantità di contenuti che ognuna di queste scene si è trovata ad ospitare.

[22] https://playdead.com/games/inside/
[23] https://tenderclaws.com/vvr
[24] https://overland-game.com/

Sarebbero molti gli aspetti da discutere per tale capitolo, che oltretutto è stato quello che ha richiesto il maggiore tempo di produzione. Volendo però selezionare alcuni punti salienti, si possono citare i seguenti:

- gestione e prevenzione della motion sickness;
- dinamica di interazione;
- shader per l'animazione degli edifici nella città;
- shader e script per la gestione dei billboard.

Di seguito, i precedenti punti verranno affrontati nel dettaglio.

Controllo della motion sickness

Rabbia, come già accennato, nasce come capitolo in movimento, in cui ogni utente compie un viaggio a bordo di una capsula flottante.

Tuttavia, è noto che in VR, un movimento del punto di vista dell'utente nel mondo virtuale, non associato ad un movimento fisico nel mondo reale [4], genera nell'utente una sensazione di nausea, poiché il cervello processa un'informazione di movimento che contrasta con la staticità di cui fa esperienza il corpo.

Si è visto però, che tale effetto può essere mitigato nei seguenti modi:

- utilizzando una tecnologia a maggiori gradi di libertà (non fattibile in VERA);
- utilizzando meccaniche di movimento "attivo", come ad esempio il *teleporting*, in cui è l'utente a triggerare l'azione che porta a spostarsi nello spazio (è richiesto però l'uso di un controller);
- dare all'utente un punto fisso con cui sentirsi solidale, ad esempio ponendolo su di un sedile o su un mezzo di trasporto. Si è visto che in questa maniera, poiché viene sfruttato un paradigma di movimento "credibile" per l'esperienza dell'utente, si genera un fastidio inferiore durante il movimento.

È proprio per questo motivo che si è scelto di creare delle capsule per ospitare i vari avatar (come visibile in figura 40).

Però, nel determinare la sensazione di maggiore o minore comfort per l'utente, entrano in gioco anche altri fattori, come ad esempio la qualità del percorso (ad es. presenza di curvature, saliscendi) e la velocità di percorrenza.

A questo proposito, le prime versioni delle varie scene di Rabbia, erano pensate per essere curve, circolari o variabili in velocità in alcuni punti, in modo da rendere maggiormente vario ed interessante il viaggio virtuale.

Ciò che però è emerso dai primi test, è che molti segmenti risultavano fastidiosi o addirittura insostenibili per alcuni utenti, perciò si è deciso, in ultima analisi, di ripensare tutta l'esperienza e la costruzione del mondo di Rabbia come un segmento pressoché rettilineo, perdendo qualcosa in termini di varietà, ma guadagnando un'esperienza complessivamente più fluida nella fruizione.

Dinamica di interazione

Nel progettare la meccanica di interazione per la risposta alle varie domande all'interno del capitolo, si è da subito tenuto conto il fatto di doversi interfacciare con un pubblico generalista.

L'idea dunque di dare in mano all'utente medio un controller, che rischia di non saper usare da subito, è stata scartata, poiché rischiava di rompere l'immersione nel mondo virtuale e di generare frustrazione.

Perciò, si è scelto di optare per un approccio più semplice ed intuitivo, in cui ogni domanda è a risposta binaria e l'utente può rispondere con una meccanica di *gazing*, ovvero mantenendo puntato il proprio sguardo (a cui è associato un cursore visibile) sul pulsante interagibile desiderato.

Per realizzare ciò, è stato necessario creare i seguenti script:

- `QuestionManager` e `RabbiaConfigurationFileLoader`, responsabili del caricamento dei testi delle domande, salvate in un file .json di configurazione;
- `CartQuestionHandler`, responsabile della gestione dell'interfaccia grafica della domanda, attraverso la manipolazione di un oggetto *Canvas* in world space. Questo oggetto gestisce anche i *callback* derivanti dalla pressione dei bottoni;
- `CartButtonInteractable`, il cui compito è avvertire l'interazione con i pulsanti fisici sulla plancia di comando, riconoscendo le eventuali collisioni con i *raycast* emessi dall'`InputManager` della camera e comunicando le risposte dell'utente al `CartQuestionHandler`.

Inoltre, per rendere operativa una simile architettura, è necessario che in scena sia presente un oggetto di tipo *Event System*.

Animazione degli edifici

Il cuore del capitolo di Rabbia, è la città distopica di *V-Rage*. Al fine di rendere l'ambiente urbano vibrante e ricco di stimoli, era necessario trovare una soluzione, non troppo dispendiosa dal punto di vista computazionale, ma che provocasse nello spettatore una sensazione di oppressione e di smarrimento nel caos luminoso della metropoli.

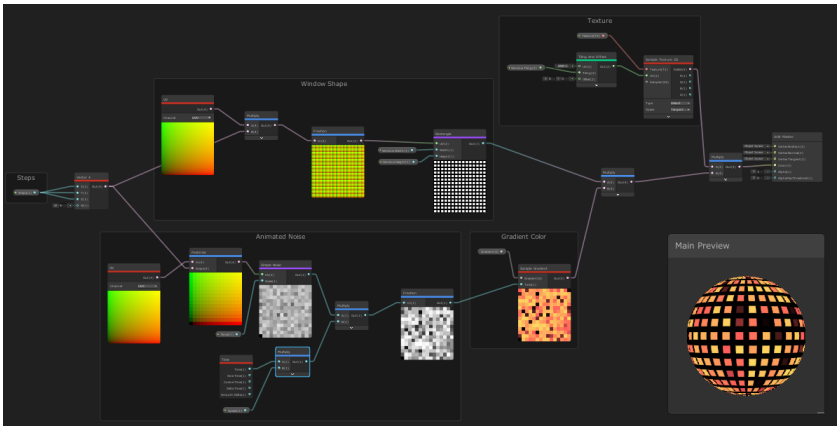

Figura 43: Shader dinamico per gli edifici di V-Rage

Si è scelto perciò di lavorare con mesh semplici per gli edifici (perlopiù parallelepipedi o cilindri), per risparmiare nel *vertex count* (visto il loro numero in scena) e di puntare su uno shader dinamico creato ad hoc, per generare le variazioni di luminosità nell'ambiente.

Lo shader realizzato (Fig. 43) è stato costruito partendo dalle coordinate UV di ciascun modello, che sono state suddivise in piccoli quadranti attraverso il nodo *Posterize*, che sostanzialmente opera una quantizzazione del gradiente lineare del nodo *UV*, generando diversi *steps*.

Queste nuove coordinate sono state utilizzate in ingresso ad un nodo *Simple Noise*, animato nel tempo e fatto passare attraverso un nodo *Fraction*, il quale, dato un valore in ingresso, mantiene solo la parte frazionaria. Così facendo, si ottiene una griglia di quadratini animati in scala di grigio, che è stata moltiplicata con un gradiente a tinte calde per determinare il colore finale.

Per creare poi un minimo di bordo, al fine di separare i quadratini tra loro, come se fossero delle finestre, si è messo a punto un procedimento simile per le UV in ingresso ad un nodo *Rectangle*, che è stato poi combinato (con un nodo *Multiply*), al rumore colorato descritto in precedenza.

L'effetto finale complessivo è quello di una griglia di finestre colorate che si accendono, si spengono e cambiano colore individualmente, con una velocità di animazione regolabile.

Billboard

I billboard hanno una funzione narrativa importante in rabbia, poiché veicolano informazioni sul mondo e sul regime di V-Rage. Anche qui, uno dei requisiti principali era la varietà di contenuto, perciò si è deciso di non limi-

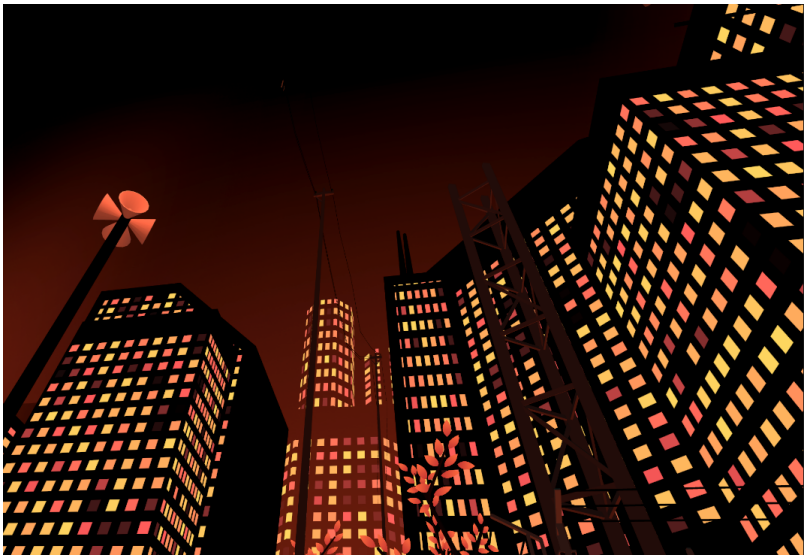

Figura 44: Gli edifici di V-Rage

tarsi ad un'unico formato, ma di considerare 3 dimensioni d'aspetto diverse per i billboard: 1:1, 2:1 e Wide.

Per non dover gestire ogni singolo billboard separatamente con la propria particolare texture, si è scelto di creare 3 diversi shader, uno per formato, ciscuno con un unico *texture atlas* in ingresso, che ospita da solo tutte le texture singole per quel particolare rapporto di immagine.

Tale tecnica sarà discussa più nel dettaglio nel capitolo 5.

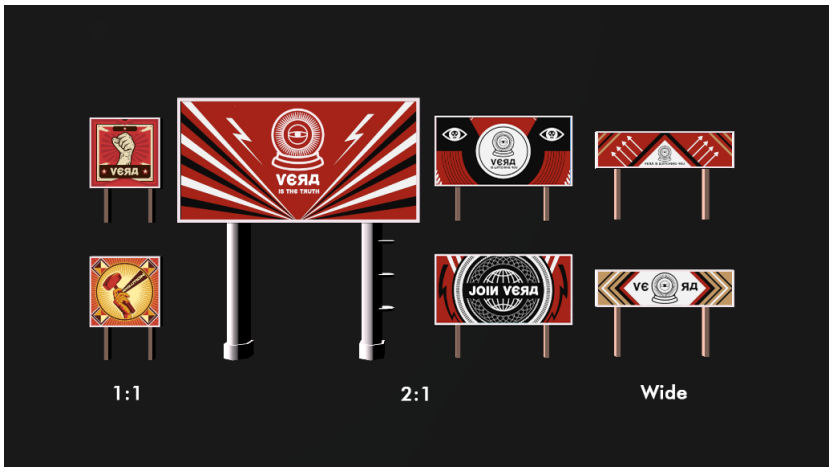

Figura 45: I vari tipi di billboard in Rabbia

In sostanza, sono state create 3 differenti macro texture 2048 x 2048 pixel

(Fig. 46), suddivise rispettivamente in:

- 16 quadrati 512 x 512 px, per il formato 1:1;
- 8 rettangoli 1024 x 512 px, per il formato 2:1;
- 16 rettangoli 1024 x 256 px, per il formato wide.

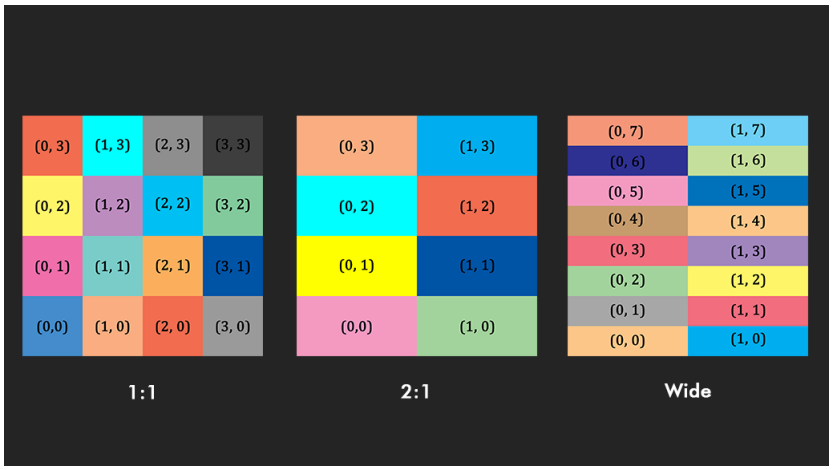

Figura 46: Texture atlas per i tre tipi di billboard

Infine, per garantire una maggiore variabilità nell'ambiente, ad ogni billboard è stato associato uno script, responsabile di assegnare a runtime un indice di offset casuale al materiale, in modo da modificare la porzione di texture atlas a cui la singola istanza punta.

In questo modo, ad ogni esecuzione, ciascun billboard potrà mostrare una porzione di texture diversa, modificando l'aspetto dell'ambiente.

4.3 Tristezza

Il terzo ed ultimo capitolo della performance indaga l'emozione della Tristezza. Lo scopo narrativo del capitolo è quello di far rievocare nella mente dello spettatore un proprio ricordo triste o malinconico e di condividerlo con gli altri.

L'idea è quella di trasformare la tristezza da una dimensione individuale ad una collettiva, in modo da esorcizzare i suoi effetti negativi, attraverso una condivisione catartica.

Al contrario del caos artificiale di Rabbia, il capitolo è ambientato in un paesaggio naturale primordiale, in cui sono presenti i quattro elementi di acqua, terra, fuoco e vento e non vi sono segni espliciti di civiltà.

Figura 47: Gli avatar di Tristezza in cerchio

In una prima fase, gli utenti vedranno, in un ambiente spoglio, delle rappresentazioni video di persone che diventano progressivamente sempre più tristi. Lo scopo di questa parte è di concentrarsi ed empatizzare con figure umane in cui l'espressione della tristezza è manifesta.

In una seconda fase, più interattiva, il mondo si forma attorno agli utenti, i quali sono chiamati a decidere se condividere o meno il proprio *ricordo*, rappresentato da una sfera luminosa. Qualora decidano di farlo, esso andrà ad alimentare il falò posto al centro della scena, liberando idealmente l'utente dal peso della propria tristezza.

Così come per il capitolo di Rabbia, la colonna sonora di Tristezza è stata prodotta dalla collaborazione di vari musicisti e produttori, durante una residenza artistica, tenutasi dal 7 all'11 ottobre 2019 alla Fondazione Pistoletto di Biella[25].

[25] http://www.cittadellarte.it/

Figura 48: Rappresentazione del ricordo in Tristezza

4.3.1 Aspetti tecnici

Dal punto di vista tecnico, il capitolo di Tristezza si è rivelato forse il più semplice dei tre da affrontare, poiché l'azione avviene in un unico luogo e poiché la produzione è iniziata quando si era già a buon punto con i capitoli di Quiete e Rabbia, dunque gran parte delle criticità erano già state viste.

L'unica vera nuova problematica, è stata la necessità di gestire dei video all'interno della scena 3D. Gli altri aspetti interessanti, hanno più che altro riguardato la scrittura di shader per ottenere determinati effetti grafici, come la dissoluzione degli avatar, l'acqua del laghetto e della cascata e l'effetto del vento sulla vegetazione.

Nei successivi paragrafi, vedremo tali aspetti nel dettaglio.

Integrazione dei video player nella scena 3D

Nelle idee iniziali di produzione, il processo di empatizzazione con una manifestazione di tristezza, doveva avvenire sfruttando una mesh 3D di un volto, dotato di rig facciale e animato attraverso delle *blendshapes*, ovvero dei controlli di deformazione della mesh, che simulano l'azione dei vari muscoli di espressione del viso.

Tuttavia, ci si è resi subito conto che, senza disporre nel team di una figura di *rigger* esperto, il processo di animazione rischiava di richiedere un tempo eccessivo e dei risultati di qualità incerta.

Perciò, la prima alternativa che si è tentato di considerare, è stata provare ad utilizzare un software per la *facial motion capture* come Avatar Ma-

ker[26]. Esso consente di registrare un video, che poi viene analizzato con algoritmi di computer vision, i quali consentono idealmente di mappare i movimenti del soggetto sulla mesh 3D, andando a creare automaticamente una deformazione coerente con quanto ripreso dalla camera.

Sfortunatamente, i risultati ottenuti non si sono rivelati incoraggianti. La fedeltà di riproduzione dei movimenti è risultata troppo imprecisa e rischiava di non rendere credibile il volto finale, mettendo a repentaglio l'immedesimazione dell'utente.

Perciò, alla fine si è deciso di percorrere una strada più sperimentale, provando ad integrare del girato video all'interno della scena 3D. Inserire un video all'interno della scena di Unity è semplice: è sufficiente creare un oggetto dotato di un componente *VideoPlayer* e un materiale che referenzi una determinata *Render Texture* contenente il video.

Il problema è che, specialmente in ambiente mobile, è sconsigliabile avere troppi video in scena, per ragioni di performance e memoria. Si è deciso, perciò, di preparare le registrazioni dei vari volti, includendole tutte in un unico grande video, una dopo l'altra, lasciando poi ad uno script (`VideoPlayerDesyncManager`) la gestione dei vari offset, per far cominciare ogni oggetto video dal frame corretto, simulando la presenza di sorgenti differenti.

Per ragioni di compatibilità con il sistema mobile (Android), i video sono stati convertiti in codifica .webm con un codec VP8[27].

Shader per il laghetto e la cascata

Creare uno shader per simulare l'effetto dell'acqua è stato uno degli aspetti più interessanti per il capitolo di Tristezza. Per ottenere l'effetto complessivo che si può osservare in Fig.49, sono stati in realtà combinati:

- un *water shader* per il laghetto;
- un *waterfall shader* per il volume principale della cascata;
- un *ripple shader* per la generazione dell'effetto vorticoso alla base della cascata;
- un effetto particellare che simula gli schizzi dell'acqua, dove la cascata incontra il laghetto.

Poiché gli shader sopracitati sono logicamente molto simili (al massimo cambia la modalità con cui effettuare il *vertex displacement*), discuteremo

[26] https://assetstore.unity.com/packages/tools/modeling/avatar-maker-pro-3d-avatar-from-a-single-selfie-134800

[27] https://docs.unity3d.com/Manual/VideoSources-FileCompatibility.html

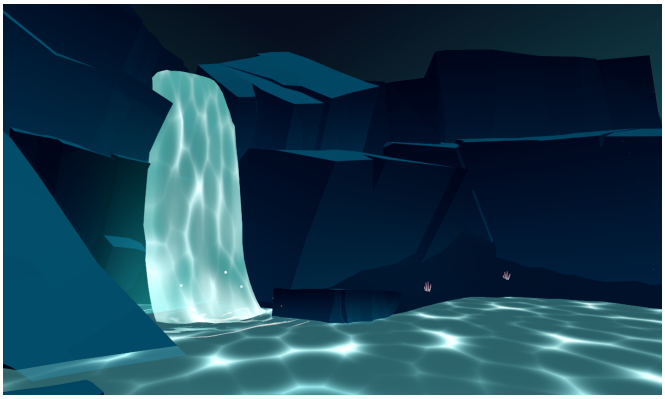

Figura 49: La cascata e il laghetto in Tristezza

come esempio di riferimento lo shader per lo specchio d'acqua (Fig.50).

Il *water shader* è composto essenzialmente da tre parti. Una prima parte serve per simulare l'effetto dei fronti d'onda che increspano la superficie del piano, attraverso lo spostamento dei vertici (nodo *Position*) lungo la direzione normale, moltiplicata per un parametro di rumore pseudocasuale. La quantità ottenuta è sommata alla posizione originale di ciascun vertice, per ottenere la posizione finale.

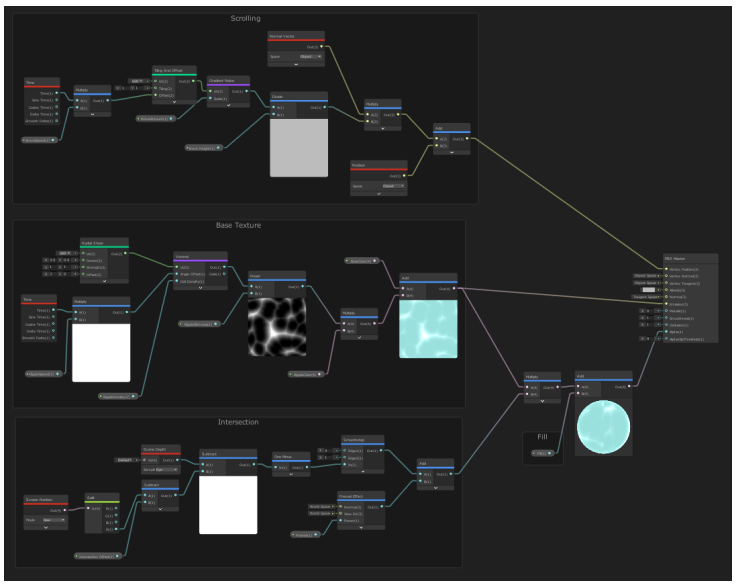

Figura 50: Shader per il piano acquatico del laghetto

Una seconda parte si occupa di definire il pattern acquatico di base ed è realizzata attraverso un nodo di *Voronoi*, fatto passare attraverso una fun-

zione di elevamento a potenza che fa variare il gradiente di distanza tra le varie celle, creando un effetto simile ad una ragnatela. Tale pattern è poi animato con una leggera distorsione radiale delle coordinate UV, attraverso l'influenza di un nodo *Radial Shear*.

Infine, l'ultima parte dello shader serve per gestire le intersezioni con altri oggetti, creando un gradiente che va schiarendosi in prossimità dei bordi. Tale comportamento è realizzato sottraendo alla z-depth, un valore relativo alla posizione in coordinate di screen space. Invertendo questo risultato e rimappandolo nell'intervallo (0, 1), si può controllare l'opacità del piano, in corrispondenza delle intersezioni con altri oggetti.

Shader per la simulazione del vento sulla vegetazione

Un'altra applicazione interessante dello ShaderGraph è stata la creazione di un materiale che simulasse l'effetto del vento sull'erba e sulla vegetazione.

La parte più interessante dello shader è legata allo scostamento dei vertici dalla loro posizione originale.

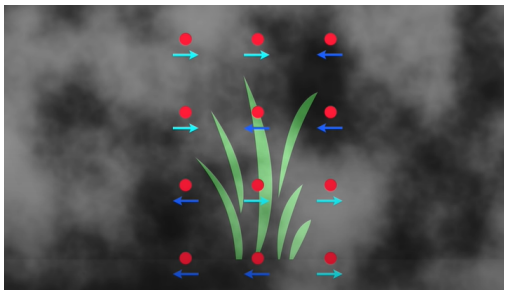

Figura 51: Rumore che viene sommato o sottratto sull'asse x

Per ottenere il movimento dei vertici, si genera un pattern di rumore animato in world space con uno scorrimento orizzontale. Ciò che si ottiene è un insieme di valori in scala di grigio che variano nell'intervallo (0, 1) e che vengono sommati o sottratti alla coordinata x del vertice, causandone lo spostamento a destra o a sinistra (51).

Per far sì che la parte bassa dell'oggetto risenta meno dell'effetto del vento, rispetto alla parte alta, si utilizza un gradiente verticale (canale *green* del nodo UV) come maschera per l'interpolazione lineare tra la vecchia posizione del vertice e quella nuova.

Per ultima cosa, è necessario trasformare queste operazioni sui vertici dal world space al sistema di coordinate locali dell'oggetto.

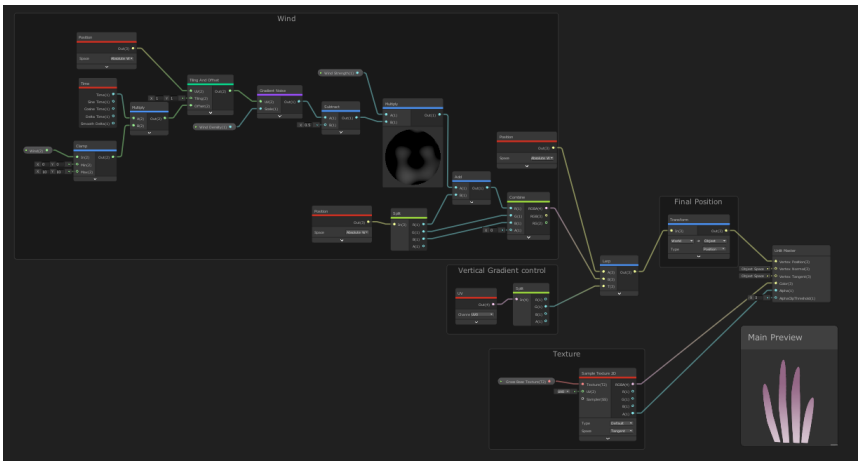

Figura 52: Shader che simula l'azione del vento

5 Ottimizzazioni e Performance

Uno degli aspetti più decisivi, al fine di determinare il potenziale successo o insuccesso di un'applicazione in realtà virtuale, è la capacità di garantire una fruizione fluida ed un *frame rate* consistente.

Infatti, se un utente venisse messo davanti ad un prodotto anche estremamente curato e visivamente coinvolgente, ma con delle evidenti problematiche come cadute nel frame rate, *jittering*, perdita di fotogrammi o surriscaldamento del dispositivo, l'esperienza complessiva risulterebbe certamente compromessa.

In contesto come VERA, in cui la piattaforma target è oltretutto un dispositivo VR *mobile*, con:

- uno schermo a risoluzione abbastanza elevata (2560 x 1440 pixel);
- un'unità di elaborazione che si avvicina più alle specifiche di uno smartphone (chipset Snapdragon 821, simile al processore di un Samsung Galaxy S7), che di una workstation;
- un'immagine stereoscopica da renderizzare due volte (una per occhio);

è evidente che, per raggiungere una soglia come i 60 frame per secondo[28] (per evitare motion sickness e perdita di comfort), c'è bisogno di qualche accogimento.

Nel capitolo, vedremo alcune regole e concetti generali per massimizzare le performance in un dispositivo come Oculus Go, per poi soffermarci nel concreto su quanto adottato nella fase di ottimizzazione di VERA. Va detto, che non sempre tutti i principi sono applicabili per tutti i progetti, perciò, a seconda delle particolari esigenze dell'applicazione, si dovrà trovare una soluzione su misura, accettando alcuni compromessi.

5.1 Concetti generali

Di seguito, saranno presentati alcuni tra i più importanti concetti per l'ottimizzazione di applicazioni VR *mobile*. Verranno indagati i principali problemi e "colli di bottiglia" possibili per un'applicazione, così come le principali soluzioni per fronteggiarli.

5.1.1 Una situazione ideale

Ogni progetto ha i suoi requisiti grafici specifici, che possono essere giustificati da necessità narrative o semplicemente estetiche. Non è raro che, la

[28] https://developer.oculus.com/documentation/unity/unity-perf/?device=GO

maggior parte degli elementi visivamente più interessanti, sia anche la più intensiva dal punto di vista del costo computazionale.

D'altro canto, l'ottimizzazione e la ricerca delle migliori performance, se spinta eccessivamente oltre la sua vera utilità (ovvero garantire la migliore resa possibile, senza snaturare l'aspetto estetico del prodotto), rischia di prendere il sopravvento sulla creatività, impoverendo l'aspetto finale del prodotto.

Per questo, un importante punto da tenere a mente è il seguente: ottimizzazione e resa artistica sono due aspetti correlati ed è necessario operare nella direzione di un equilibrio reciproco, senza voler pretendere l'impossibile dal proprio hardware con effetti eccessivamente audaci, ma anche senza semplificare fino all'osso la propria applicazione, solo per ottenere un frame rate più alto di quanto effettivamente necessario.

Fatte queste premesse, esistono alcune linee guida fornite da Oculus[29], che forniscono una buona base ideale di confronto, per capire se (e quanto) il proprio prodotto sia potenzialmente adatto ad eseguire su Oculus Go senza problemi. È consigliabile rimanere all'interno dei seguenti termini quantitativi:

- 60 frame al secondo (16,6 millisecondi per frame);
- meno di 150 *draw call* per frame;
- meno di 100K vertici o triangoli per frame.
- meno di 1024 megabyte di memoria grafica dedicata alle texture.

5.1.2 Applicazioni CPU e GPU-bound

Generalmente, gli esperti nel campo dell'ottimizzazione, al fine di isolare e classificare i potenziali problemi, sono soliti dividere le applicazioni in due macro-categorie distinte, a seconda dell'unità di elaborazione che è causa del maggiore carico di lavoro per il dispositivo:

- applicazioni CPU-bound;
- applicazioni GPU-bound.

La prima cosa che si fa per capire in che contesto ci si trova, è: non renderizzare nulla (ovvero disabilitare la camera). Se così facendo il frame rate migliora, allora vuol dire che si è GPU-bound; mentre se il frame rate rimane invariato, si è CPU-bound.

[29] https://developer.oculus.com/documentation/unity/unity-perf/?device=GO

Se si è CPU-bound, è possibile ottenere maggiori informazioni sugli elementi che impattano maggiormente sul budget temporale del frame, sfruttando strumenti come il *profiler* di Unity.

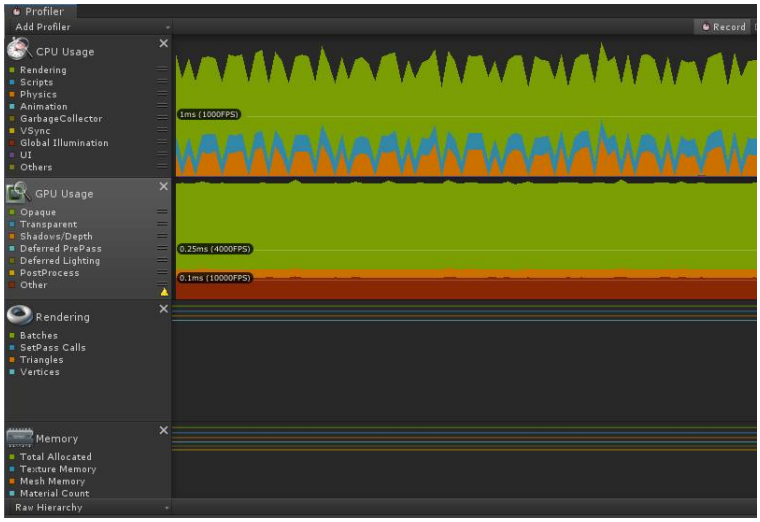

Figura 53: Profiler di Unity

Nel caso GPU-bound, bisogna capire ulteriormente se si è:

- vertex-bound;
- fill-bound (o fragment-bound);

ovvero se il peso principale è dato dalle geometrie, oppure dalla resa dei pixel finali via shader. Anche qui, si tratta di una verifica semplice: basta scalare il proprio *render target* ad un valore molto piccolo (es. 0.01), in modo da renderizzare meno pixel e fragment a schermo. Se le performance migliorano, vuol dire che si è fill-bound; altrimenti, si è vertex-bound.

Se sono i fragment a determinare il costo maggiore, è consigliabile usare un profiler specifico per la scheda grafica di cui si dispone. Mentre, se il problema sono le geometrie, si può provare a semplificare la quantità di vertici dei propri modelli in una suite di modellazione 3d, combinare mesh tra loro, oppure usare tecniche come il LOD (*Level Of Detail*) che prevede di sostituire un oggetto 3d con varie sue versioni, via via più semplificate, all'aumentare della distanza dalla camera.

Per riassumere, in figura 54 è riportato un flow chart, che rappresenta il ciclo di analisi che è utile mettere in pratica per individuare le problematiche del proprio progetto, a livello di performance.

Rendersi conto dei principali "colli di bottiglia" nella propria applicazione è il primo passo per poter capire quali siano i punti su cui intervenire. Volendo semplificare un minimo la trattazione, si potrebbe dire che i due costi

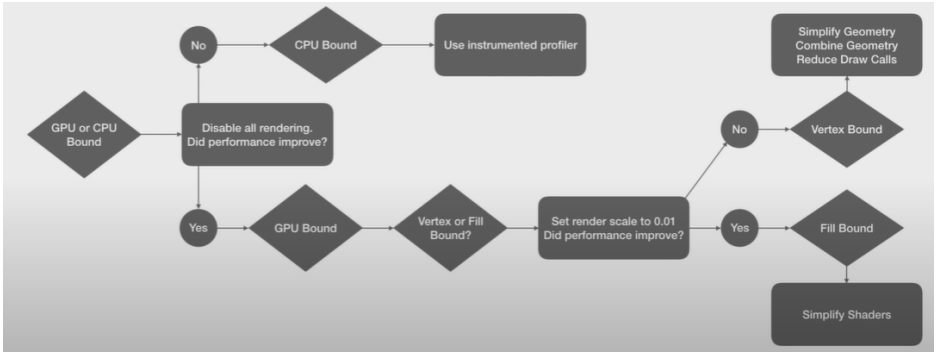

Figura 54: Ciclo di analisi delle prestazioni

principali, che occupano normalmente la maggior parte del tempo di CPU e GPU, sono rispettivamente le *draw call* e il *fill* dei fragment e dei pixel.

Tabella 1: Principali problemi e soluzioni per CPU e GPU

	CPU	GPU
Problema	Draw Calls	Pixel Fill
Soluzioni	Single-pass stereo rendering Batching Light baking Occlusion culling	Fragment shader ottimizzati No post-effects No multi-pass effects

In tabella 1, si possono vedere alcune delle principali soluzioni che fanno fronte a tali problemi. Nei seguenti paragrafi, tali aspetti verranno analizzati nel dettaglio.

5.1.3 Draw call, SetPass call e Batching

Una *draw call* è un particolare comando che la CPU manda alla GPU per ordinarle di disegnare una mesh. Per portare a termine un compito simile, è necessario che alla scheda grafica vengano fornite diverse informazioni di contesto, quali: le posizioni dei vertici, la presenza di eventuali texture con le loro coordinate, lo shader con i suoi argomenti di input ed altro ancora.

Una volta raccolte tutte queste informazioni (o "stato"), il driver del processore deve compiere del lavoro (ad esemio effettuando delle conversioni di formato) per integrarle nei vari *vertex streams* e per renderle dunque disponibili alla GPU, che effettuerà poi il rendering finale dell'elemento 3d.

In questo senso, si può dire che una draw call è assimilabile ad un costo derivante dal processore, poiché si dedica del tempo alla formattazione di

determinate informazioni, che poi verranno passate alla scheda grafica.

Perciò, quando si parla di ottimizzare le draw call, che sono comunque essenziali per disegnare qualsiasi cosa a schermo, si può operare essenzialmente con due approcci:

- cercare di ridurne il numero complessivo
- cercare di settare lo "stato" un numero minimo di volte, facendo sì che si possano disegnare più oggetti, sfruttando la condivisione delle stesse informazioni di contesto.

Questo secondo punto è estremamente importante da tenere a mente, poiché empiricamente si può vedere che, una volta definito uno stato condivisibile da più mesh, le draw call successive sono molto semplici e poco costose in fase di esecuzione. Questa tecnica è nota col nome di *static batching* e, se sfruttata correttamente, può essere responsabile di una massiva riduzione del numero di draw call.

Per godere dei vantaggi del batching statico e disegnare più mesh in una sola volta, è necessario configurare la propria scena adeguatamente, seguendo determinate regole che consentono di non "spezzare" il processo, in modo da non cambiare lo stato della GPU mentre si disegna il frame:

- avere un solo materiale assegnato per ogni mesh. Per definizione infatti, nel caso in cui vi siano due materiali, è necessario cambiare stato per disegnare prima uno e poi l'altro.
- Marcare la mesh come *static* (Fig. 55). Questa condizione è necessaria perché Unity proverà a combinare solo le mesh che appartengono a questa categoria. Va detto che, gli oggetti che sono etichettati in questo modo non possono cambiare i loro attributi di posizionamento a runtime. Devono infatti rimanere fissi nello spazio, per poter essere combinati con altri oggetti in mesh cumulative più grandi.
- Far sì che più mesh statiche condividano lo stesso materiale e dunque lo stesso shader e le stesse texture eventuali. Si parla di *shared material*.

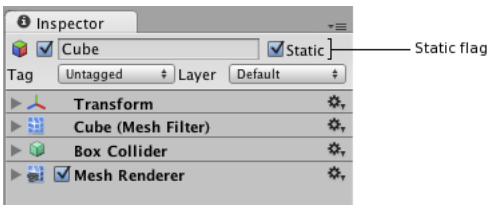

Figura 55: Un oggetto marcato *static* in Unity

Tabella 2: Esempio con differenti condizioni di batching

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Batching	No	Sì	No	Sì
Material	Mat_A Mat_B Mat_C	Mat_A Mat_B Mat_C	Mat_shared	Mat_shared
Draw events	1. SetPass (Mat_A) 2. Draw call (obj 1) 3. SetPass (Mat_B) 4. Draw call (obj 2) 5. SetPass (Mat_C) 6. Draw call (obj 3)	1. SetPass (Mat_A) 2. Draw call (obj 1) 3. SetPass (Mat_B) 4. Draw call (obj 2) 5. SetPass (Mat_C) 6. Draw call (obj 3)	1. SetPass (Mat_shared) 2. Draw call (obj 1) 3. Draw call (obj 2) 4. Draw call (obj 3)	1. SetPass (Mat_shared) 2. Draw call (obj 1 + obj2 + obj 3)
SetPasses	3	3	1	1
Batches	3	3	3	1
Performance	Worst	Worst	Good	Best

Lo stato del materiale, dunque, rappresenta un elemento fondamentale perché il batching funzioni correttamente. Per comprendere però correttamente questa sfumatura, è necessario discutere un nuovo concetto, sempre vicino alle draw call, ma con delle sottili ed importanti differenze: le *setPass call*.

Se una draw call (o batch) rappresenta un comando di disegno per l'unità grafica, una setPass call rappresenta un'operazione in realtà più complessa, con cui si determina un cambio di stato per un materiale. Questo, come si è visto, implica anche un cambio del *render state* per la GPU e quindi una rottura del batching.

Ogni materiale presente in scena porta con sé almeno una setPass call e il numero aumenta ogniqualvolta si ha un cambio di shader, della configurazione di rendering, delle opzioni di alpha blending, dello z-buffer e via dicendo. Per questo motivo, si cerca di mantenere il minor numero possibile di materiali in scena, cercando, dove possibile, di condividerli tra mesh diverse.

Per chiarire tale aspetto, proviamo ad analizzare un esempio in cui vi sono 3 oggetti da renderizzare, che condividono la stessa mesh e che sono posti in differenti condizioni di batching (Tab. 2).

Si può vedere (scenari 1 e 2) come l'utilizzo di materiali diversi, impatti negativamente sul numero degli eventi di rendering. Nel caso 3, poiché gli oggetti condividono lo stesso materiale, le draw call saranno sempre 3, ma meno costose, poiché lo stato è invariato. Nello scenario 4, il batching è sfruttato alle sue massime possibilità, determinando un grande incremento nelle performance.

Oltre al batching di tipo statico, è possibile anche renderizzare in batch alcuni oggetti dinamici (non marcati *static*), a patto che:

- condividano lo stesso materiale;
- abbiano meno di 300 vertici e meno di 900 *vertex attributes*.

Questo processo, meno importante nella pratica rispetto alla sua controparte statica, funziona processando i vertici degli oggetti in movimento nella CPU e si può capire che acquista un senso solo quando il costo è inferiore a quello che avrebbe la singola draw call.

Per finire, un tool di analisi estremamente utile per monitorare il costo della propria propria scena è il *Frame Debugger* di Unity. Esso consente di visualizzare, chiamata per chiamata, le operazioni svolte dall'engine durante la resa di un frame, in modo da poter individuare eventuali problematiche e correggerle.

5.1.4 Aspetti di configurazione del progetto

Esistono delle tecniche di ottimizzazione "gratuite" che è possibile implementare semplicemente, attraverso un'adeguata configurazione del proprio progetto in Unity. È consigliabile dunque dedicare una minima attenzione alla configurazione di alcuni parametri, che cumulativamente possono portare ad un incremento anche discreto nelle performance.

Innanzitutto, è bene limitarsi ad utilizzare esclusivamente un tipo di compressione delle texture di tipo ASTC (*Adaptive Scalable Texture Compression*). Essa garantisce infatti il miglior equilibrio tra qualità della compressione e dimensione finale del file.

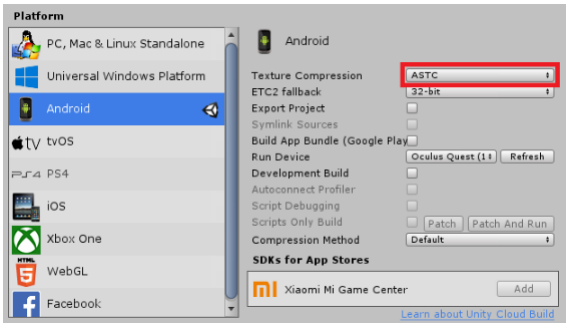

Figura 56: Opzione per ASTC nelle impostazioni di Build

Questo tipo di compressione permette di trasferire meno dati dalla memoria principale alla GPU, che dunque può spendere più tempo per il rendering vero e proprio.

Figura 57: Esempio di compressione ASTC

Sebbene possa causare qualche artefatto (ad esempio nelle texture per la UI), su un modello 3d, la differenza è praticamente impercettibile ed il risparmio di memoria notevole.

Un parametro cruciale da settare correttamente, soprattutto se si sviluppa in VR, è la modalità di rendering in *single pass stereo*[30].

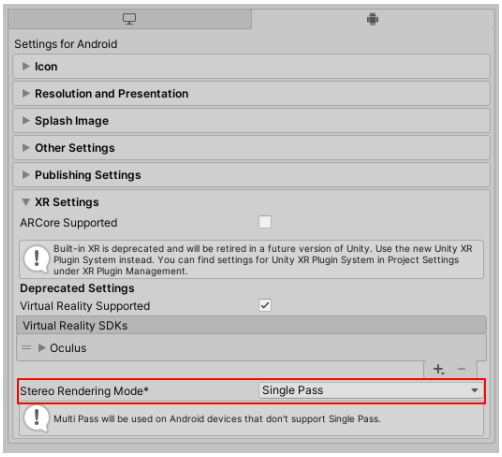

Figura 58: Single pass stereo rendering

Questa feature, nota anche come *multi-view rendering*[31], è un'estensione di OpenGL ES costruita specificatamente per la VR mobile, che permette di disegnare con le stesse informazioni l'immagine per l'occhio destro e l'occhio sinistro, senza duplicare le draw call necessarie.

[30] https://docs.unity3d.com/Manual/SinglePassStereoRendering.html
[31] https://arm-software.github.io/opengl-es-sdk-for-android/multiview.html

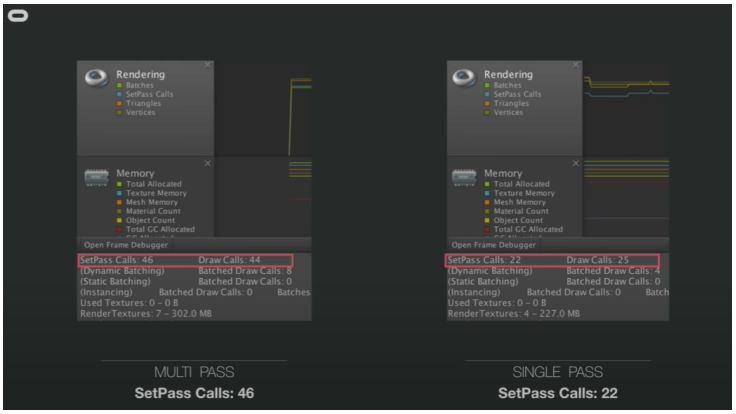

Figura 59: Single pass stereo vs. multi pass stereo

Si ha dunque un risparmio del 50% nelle chiamate necessarie, lato CPU.

Per evitare di vedere pixelati i bordi degli oggetti nella propria scena 3d, è opportuno usare dei filtri di MSAA (*Multi Sample Anti-Aliasing*). Un valore di 4x è considerato il miglior compromesso in VR.

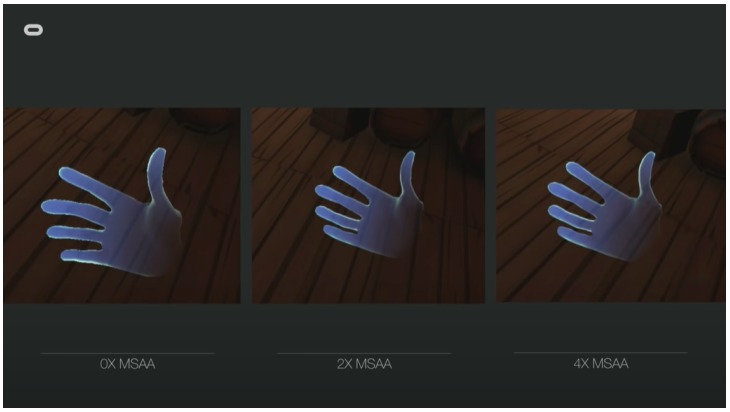

Figura 60: Effetto di smoothing del 4x MSAA

Altre opzioni che è conveniente abilitare, sono le seguenti:

- *multi-threaded rendering*, che permette di spalmare il costo di rendering su più thread paralleli;
- *static batching*, di cui abbiamo già parlato nel paragrafo precedente;
- *compute skinning*, per performare i calcoli di deformazione delle mesh in GPU.

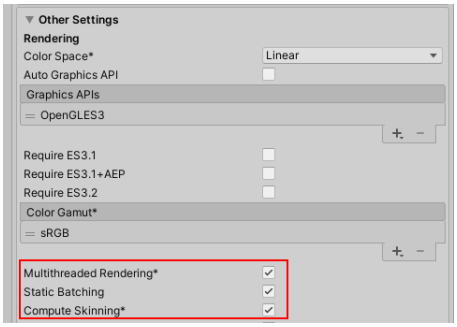

Figura 61: Altre opzioni interessanti

Un'ultimo aspetto degno di nota è legato alla possibilità di sfruttare IL2CPP (*Intermediate Language To C++*)[32] per effettuare la propria build. Questo può portare ad un leggero aumento dei tempi di build, poiché Unity deve tradurre tutti gli script in C++, però permette generalmente all'applicazione di eseguire in maniera più rapida nel device e di occupare meno spazio.

5.1.5 Lightmapping e Baking

Nella VR mobile, si vuole evitare il più possibile la presenza di luci e ombre calcolate in real-time, dato il loro costo computazionale.

Per questo, si cerca di pre-calcolare l'informazione luminosa presente in scena in modalità "offline", per evitare di pagare il costo in fase di esecuzione. Tale processo è noto come *light baking* e il risultato che si ottiene in output è costituito da texture di grandi dimensioni, in cui vengono registrati i valori di illuminazione per gli oggetti presenti nei vari punti della scena.

Queste texture (chiamate anche *lightmap*) sono poi "applicate" alla superficie degli oggetti (in overlay al materiale), in modo da conferire il look finale, in maniera statica e senza calcoli costosi.

L'aspetto positivo è che, in questo modo, la resa di un oggetto con luci ed ombre pre-calcolate è molto performante a runtime, ma il prezzo da pagare è dato dal tempo necessario al baking (operazione di norma abbastanza complessa) e dal fatto che l'oggetto dev'essere necessariamente statico nel mondo di gioco (luci e ombre sono calcolate in quella configurazione precisa).

Il baking è un argomento molto vasto e complesso, che meriterebbe un approfondimento a sé, perciò, di seguito, ne verrà fornita una descrizione ad alto livello, dedicata soltanto ad alcuni aspetti principali, senza una pretesa di esaustività.

[32] https://docs.unity3d.com/Manual/IL2CPP-HowItWorks.html

Luce diretta e indiretta

In computer grafica, per esigenze di calcolo, si tende a semplificare il comportamento della luce, classificandola in:

- luce diretta, ovvero la luce parte dalla sorgente e colpisce un oggetto con 0-1 riflessione;
- luce indiretta, nota anche come GI (*Global Illumination*), che invece costituisce la componente diffusa che arriva all'oggetto dopo 2 o più riflessioni.

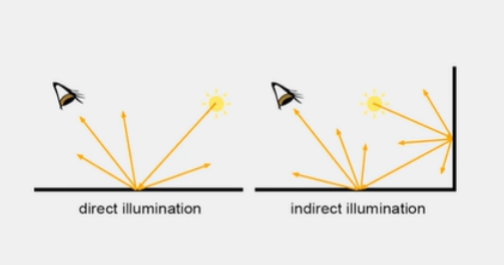

Figura 62: Luce diretta e indiretta

Nella figura 63, si può vedere da un lato l'effetto della sola luce diretta (che causa principalmente le riflessioni speculari e le ombre portate) e dall'altro l'aggiunta della GI (che va a colorare e ad illuminare le zone d'ombra).

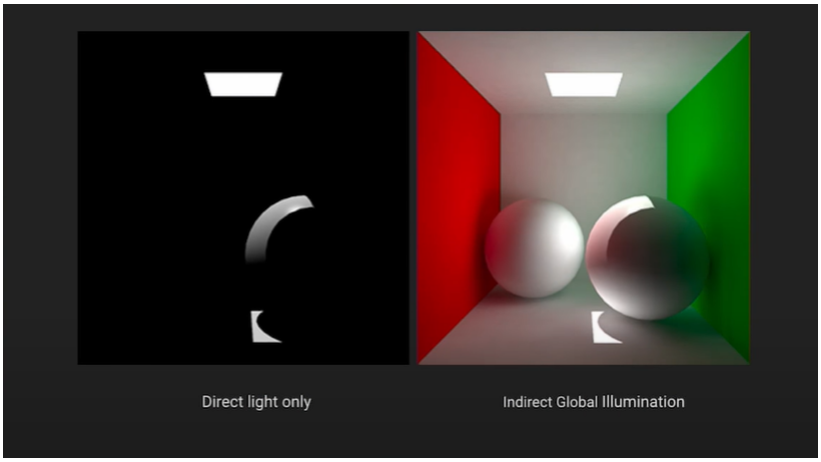

Figura 63: Luce diretta e Global Illumination

L'illuminazione indiretta richiede molti calcoli, perciò non è possibile simularla in real-time, ma solo via baking, mentre per l'illuminazione diretta

è possibile, anche se è consigliato limitare tale approccio, soprattutto nel caso mobile.

Se è proprio necessario disporre di illuminazione real-time sugli oggetti in movimento, Unity mette a disposizione, per le luci direzionali, una modalità *mixed*, che ha effetto sia sul baking, sia sugli oggetti dinamici.

Come già accennato, un oggetto per poter essere incluso nei calcoli di GI, deve avere un mesh renderer contrassegnato come *static* e ciò significa che non potrà muoversi nel mondo virtuale. È possibile specificare se l'oggetto dovrà contribuire alla GI, ricevere la GI o entrambe le opzioni.

Allo stesso modo, anche le luci dovranno essere esplicitamente contrassegnate come *baked* (o *mixed*).

Progressive lightmapper

Invece di usare tecniche come il path tracing o il ray tracing in cui i raggi luminosi virtuali partono dalla camera e vanno a collidere con gli oggetti 3d, il *Progressive Lightmapper* di Unity calcola l'illuminazione facendo partire i raggi da vari *texel*. Questo permette di muovere la camera in giro per la scena, anche mentre il baking è in corso.

Un texel è l'unità minima di risoluzione presente nelle lightmap ed è mappato spazialmente sulla geometria 3d (su ogni mesh renderer è possibile specificare la risoluzione necessaria all'oggetto).

Ogni texel rappresenta un punto nello spazio 3d in cui viene salvata una certa informazione luminosa aggregata, sulla base delle fonti dirette e indirette che lo circondano.

Per effettuare questo calcolo, ogni quadratino emette un certo numero di raggi verso le sorgenti lumino, con un certo numero di *bounces* disponibili (riflessioni). Quando il raggio giunge al numero massimo di riflessioni, si ferma e riporta il valore di luminosità, che viene poi registrato dal texel nella sua porzione di lightmap.

Lighting modes

Essenzialmente, esistono 3 configurazioni principali di illuminazione adottabili nel proprio progetto:

- solo luce real-time;
- solo luce baked;
- mixed lighting.

Il primo caso non è praticamente mai utilizzato, visto il costo eccessivo dei calcoli di illuminazione in tempo reale.

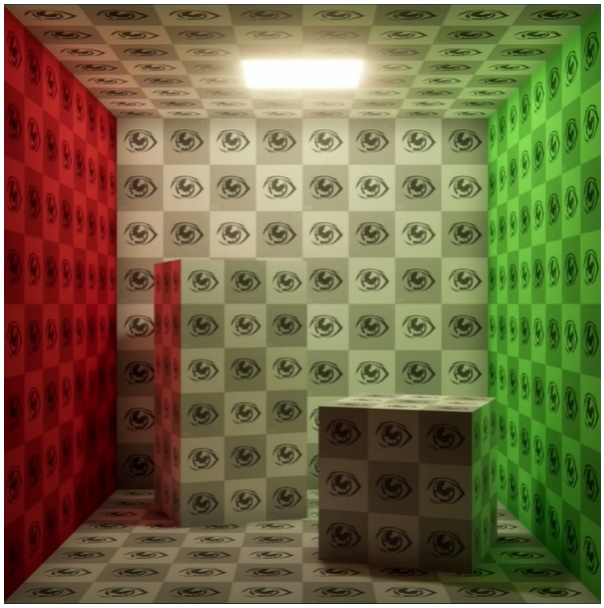

Figura 64: Rappresentazione dei texel

Nel caso in cui si usi esclusivamente un tipo di illuminazione baked, gli oggetti statici sono resi correttamente, mentre ci sono delle limitazioni abbastanza evidenti per gli oggetti dinamici. Come si può vedere in figura 65, la sfera blu sulla destra, che non è marcata *static*, non riceve correttamente le informazioni di luce e di ombra, in assenza di una sorgente realtime.

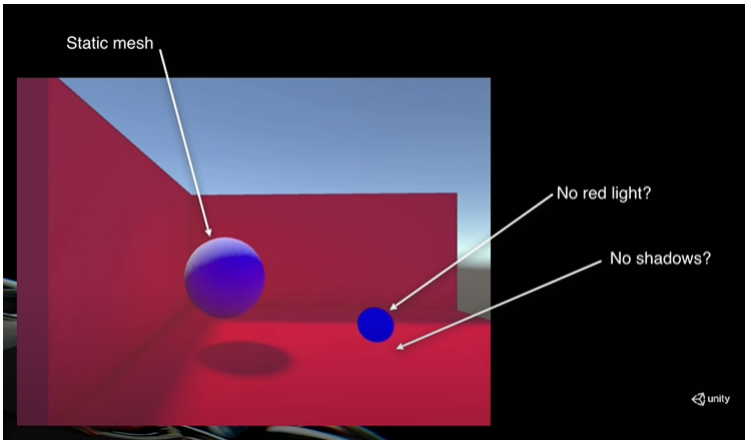

Figura 65: Oggetto dinamico con sola luce baked

Per questo motivo, è stata introdotta la modalità *mixed*, che consente di far coesistere sia l'approccio real-time, che quello baked.

Quello che manca ancora, in questo contesto, è l'influenza delle luci indirette sugli oggetti in movimento. È proprio a questo scopo che servono i *light probes*, ovvero dei reticoli volumetrici in grado di catturare l'informazione di GI in un determinato punto dello spazio e di "colorare" l'oggetto dinamico che passa in quell'area.

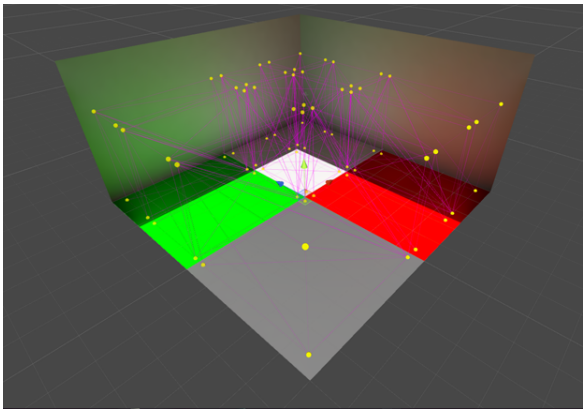

Figura 66: Light Probes

Combinando queste tecniche, si possono ottenere effetti di illuminazione efficienti e piacevoli da vedere. In tabella 3 sono riassunte le varie configurazioni possibili in materia di illuminazione.

Tabella 3: Modalità di illuminazione

	Object Type	Lighting Mode	
Light Type	-	Baked	Mixed
Direct Light	Dynamic	None	Realtime / Baked
	Static	Baked	Realtime / Baked
Indirect Light	Dynamic	None / Probes	None / Probes
	Static	Baked	Baked

5.1.6 Occlusion Culling

Durante ogni frame, la camera di Unity analizza la lista dei *Renderer* presenti nella scena, ovvero i componenti necessari per disegnare un oggetto a schermo, e performa delle operazioni di *culling* (letteralmente "abbattimento, eliminazione selettiva"), con cui evita di renderizzare quegli oggetti che

non si trovano nel campo di vista.

Tale volume immaginario, meglio noto come *frustum*, è un tronco di piramide, con un'apertura pari all'angolo di vista (*field of view*) della camera e delimitato da due piani, noti come *near* e *far clipping plane*.

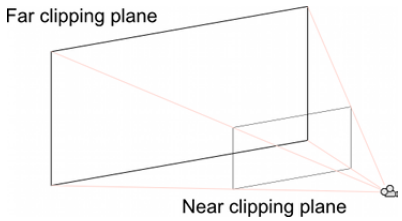

Figura 67: Frustum di vista della camera

Di default, viene adottata in Unity la tecnica di frustum culling, per cui vengono esclusi dall'immagine finale tutti i vertici appartenenti ad un renderer che non rientra nel volume di vista.

Tale tecnica costituisce già un'ottimizzazione importante, che alleggerisce il carico della GPU. Tuttavia, essa non tiene conto del fatto che, all'interno del campo di vista, si potrebbero migliorare ulteriormente le prestazioni, evitando di considerare quegli oggetti che non sono visibili, perché occlusi da altre mesh.

Per questo motivo, si utilizza spesso anche la tecnica di *occlusion culling*[33], che permette di evitare sprechi di tempo di CPU e GPU, altrimenti dedicato alla resa di oggetti non visibili dalla camera, ma comunque presenti nel frustum. La solozione adottata in Unity si basa sulla libreria *Umbra* [34].

Il processo con cui funziona questo tipo di tecnica è noto come *occlusion data baking* e funziona nel seguente modo: la scena 3d viene divisa in celle (voxelizzazione) e, in ciascuna di queste, vengono registrate informazioni sulla geometria ivi presente e sulla visibilità delle celle adiacenti (*portals*), in modo da poter avere un'informazione relativa a ciascun punto di vista possibile. L'informazione è combinata con quella derivante dal *depth buffer*.

Poi, in fase di runtime, i dati di occlusione vengono caricati in memoria e vengono gestiti dalla CPU, che informa la camera (attraverso delle *visibilty queries* alle strutture dati predisposte da Umbra) su ciò che è possibile vedere dal determinato punto di vista in cui si trova, evitando di disegnare informazioni inutili.

Anche per questa tecnica, è necessario che gli oggetti che occludono, o che sono occlusi, siano statici. Gli oggetti dinamici possono solo essere oc-

[33] https://docs.unity3d.com/Manual/OcclusionCulling.html
[34] https://umbra3d.com/

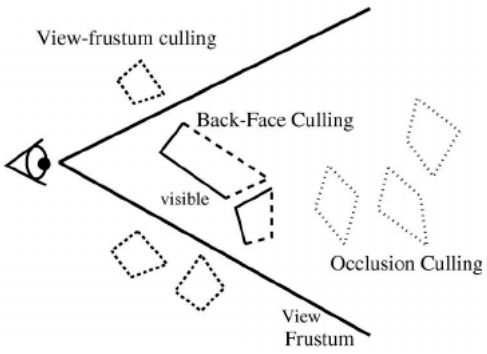

Figura 68: Occlusion culling

clusi, ma non occludere altri oggetti.

Per ottenere un vantaggio dalle tecniche di occlusion culling, bisogna ricercare un compromesso tra un'analisi fine (con una risoluzione di celle molto elevata), che dà risultati precisi, ma richiede molto tempo per il baking e che può causare un overhead per la CPU in tempo di esecuzione; e un approccio a tolleranza maggiore, che è senz'altro più veloce da processare, ma che rischia di includere troppi oggetti non indispensabili nella vista.

Tale trade-off è sempre dipendente dalla scena che si sta considerando e dagli oggetti che contiene, perciò non esiste una soluzione migliore a priori.

In generale, è bene tenere a mente i seguenti aspetti se si vuole procedere con questo tipo di ottimizzazione:

- la tecnica è più efficace se si è GPU-bound, visto che si vanno a ridurre in numero le geometrie e che i calcoli per la gestione dell'occlusione sono a carico della CPU;
- è bene disporre di memoria disponibile a runtime, in quanto i dati vengono caricati e sfruttai in fase di esecuzione;
- funziona meglio in scene in cui piccole aree ben definite vengono occluse completamente da altri oggetti solidi che occupano una grande porzione di schermo (cfr. i muri di un corridoio che occludono il contenuto di una stanza);
- tale tecnica può entrare in contrasto con approcci come il *mesh baking*, visto che è necessario disegnare tutto il contenuto di una mesh, non appena si incontra anche un solo vertice non occluso.

5.1.7 GPU Instancing

La tecnica di *GPU instancing*[35] è utile per renderizzate copie multiple della stessa mesh (con lo stesso materiale), in posizioni, rotazioni e scale diverse, riducendo di molto il numero di draw call necessarie.

È dunque utile per riprodurre efficientemente oggetti ripetuti come palazzi o elementi di vegetazione.

Per abilitare il GPU instancing, è sufficiente attivare l'opzione presente sul materiale (se il tipo di shader utilizzato lo supporta).

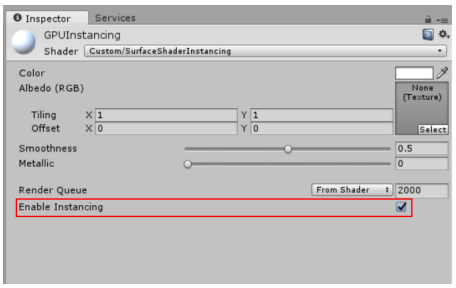

Figura 69: Opzione di GPU instancing presente sul materiale

Tale pratica può entrare in conflitto con lo static batching, e Unity, nel caso in cui siano attivi entrambi, tende a preferire il secondo approccio, disabilitando automaticamente il GPU instancing.

5.1.8 Mesh Baking e Texture Atlasing

Nei paragrafi precedenti, è stato possibile vedere quali vantaggi in termini di performance può portare il batching. Ma, a partire da una scena complessa qualsiasi, com'è possibile ridursi ad un numero contenuto di mesh (draw call) e di materiali (setPass call) per sfruttare tale tecnica?

Un primo semplice approccio prende il nome di *mesh baking*. Poiché è più semplice per la GPU disegnare molti vertici in un'unica draw call, piuttosto che disegnare molti oggetti con pochi vertici in draw call diverse, si cerca può cercare di combinare più mesh diverse in un unico oggetto.

Ogni mesh non è nient'altro che una collezione di vertici (ovvero vettori di coordinate (x, y, z) nello spazio), perciò nulla vieta di prendere alcune di queste liste di punti e di combinarle in un array cumulativo che includerà le informazioni per più oggetti, ma che sarà visto dalla scheda grafica come un elemento unico.

Così facendo, invece di avere draw call multiple, una per singolo oggetto, se ne avrà una unica.

[35] https://docs.unity3d.com/Manual/GPUInstancing.html

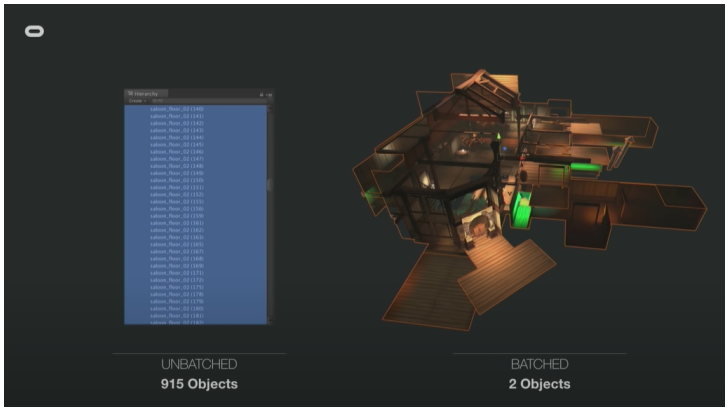

Figura 70: Mesh baking

Esistono tool come *Mesh Baker*[36] che permettono in pochi passaggi di fare un'operazione simile.

Tuttavia, si è visto che le draw call sono solo un aspetto del costo necessario per finalizzare la scena 3d: vanno gestite infatti anche le setPass call, in modo da non dovere cambiare lo stato di rendering ogni volta.

In particolare, se le mesh combinate hanno comunque materiali o shader diversi, il batching risulterà compromesso.

Perciò, si cerca di avere un unico materiale per la mesh combinata, che però deve tener conto delle eventuali texture singole applicate agli oggetti e delle loro coordinate UV.

Con un approccio simile a quanto visto per le mesh, si possono combinare più materiali e rimappare correttamente le coordinate UV degli oggetti con la tecnica del *texture atlasing*.

Un tool come *Mesh Baker* consente di effettuare anche questa operazione. In sintesi, a partire dalle singole texture degli oggetti da combinare, si crea un *atlas*, ovvero una texture di grandi dimensioni (ad esempio 4096 x 4096), costruita affiancando tutte le texture originarie.

Questo atlas viene poi associato ad un singolo materiale condiviso e vengono rimappate le coordinate UV di ciascun oggetto, in modo da puntare alla porzione di atlas corretta.

Questo permette di gestire intere scene, anche complesse, condividendo un unico grande materiale, mantenendo lo stato della GPU invariato.

[36] https://assetstore.unity.com/packages/tools/modeling/mesh-baker-5017

Figura 71: Texture Atlasing

5.1.9 Ottimizzazione del pixel fill

Si è dunque visto come ottimizzare le draw call lato CPU, ma non è ancora stato affrontato un secondo problema, altrettanto importante: il costo derivante dal *pixel fill*.

In particolare, ciascun pixel p dell'immagine finale (anche il più anonimo dei pixel di sfondo) ha un costo (in ms) che è dato dalla seguente espressione:

$$Costo(p) = N_w \times t_{FragmentShader(p)}$$

dove N_w è il numero di riscritture successive che p riceve all'interno del frame e $t_{FragmentShader(p)}$ è il tempo necessario all'esecuzione del fragment shader per ogni scrittura.

A seconda della presenza di oggetti semi-trasparenti in foreground, di specular highlights, di una luce real-time, o in generale di uno shader che richieda passi multipli, potrebbe essere necessario riscrivere il colore del pixel più volte, prima di ottenere quello finale.

Figura 72: Risoluzione degli eye-buffer in Oculus Go

Ciò può ovviamente causare un notevole impatto nei tempi di GPU, soprattutto in dispositivi come Oculus Go, in cui la risoluzione dei due schermi è abbastanza elevata. Si hanno infatti oltre 3 milioni di pixel da colorare, per ogni frame.

Le tecniche principali con cui questo problema può essere affrontato sono le seguenti:

- utilizzare fragment shader semplici e rapidi da calcolare;
- evitare qualsiasi tipo di effetto di post-processing, poiché essi sono implementati facendo una copia dell'eye buffer corrente, applicando l'effetto ai pixel intermedi e poi ricopiando ulteriormente tutto il buffer finale per il rendering, raddoppiando essenzialmente il costo;
- evitare altri tipi di effetti multi-pass, come real-time shadows, riflessioni speculari e per-pixel-lighting (che prevede un passo per ogni luce);
- evitare il fragment discard, ovvero interrompere il calcolo di uno shader e scartare parte di quanto ottenuto prima di arrivare al risultato finale (ad esempio con alpha clipping). Questa pratica è svantaggiosa, soprattutto nel caso delle GPU mobile, perché il rendering viene eseguito dividendo la scena in *tile*, ovvero sottoinsiemi quadrati dei pixel dello schermo, all'interno delle quali gli oggetti opachi vengono ordinati per determinare ciò che è visibile. Nel caso in cui un pixel particolare venga scartato, il *sorting* fatto sulle geometrie rischia di non essere più valido e di dover essere ricalcolato da capo.

5.1.10 Fixed Foveated Rendering

Generalmente, quando una persona è immersa nella fruizione di un contenuto VR, tende a focalizzare la propria attenzione al centro dell'immagine, soffermandosi meno sulle parti di contenuto presenti ai bordi dello schermo.

Tale assunto è alla base di una tecnica di ottimizzazione nota come *Fixed Foveated Rendering*. Si tratta essenzialmente di gestire l'eye buffer in modo da destinare la massima risoluzione al centro dell'immagine e di risparmiare nei pixel presenti nelle zone di visione periferica, dove l'occhio non riesce comunque a distinguere troppo i dettagli.

Abbassando dunque la risoluzione ai bordi, si riesce a risparmiare una buona quantità di tempo nelle operazioni di pixel filling, da parte della scheda grafica. Stando alle stime di Oculus, si riesce a risparmiare anche il 20-30% di tempo di GPU, nel caso di un'applicazione GPU-fill-bound.

Questo tipo di particolari di rendering si dice *fixed*, perché non sfrutta le informazioni derivanti dall'eye tracking, ma applica una matrice fissa al

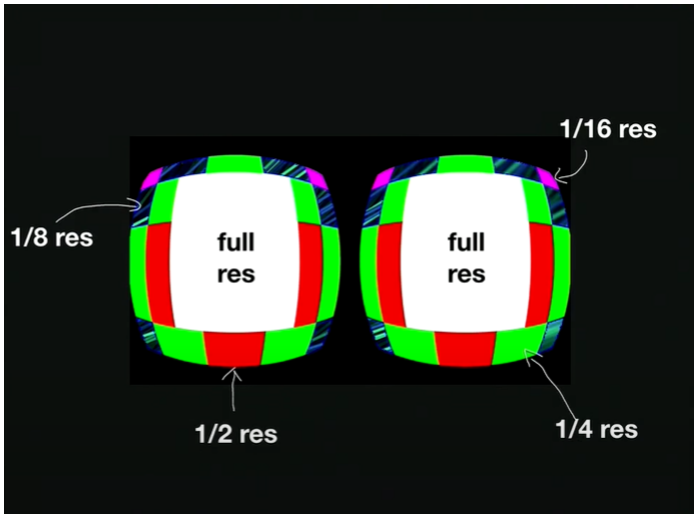

Figura 73: Fixed Foveated Rendering

quadro dell'immagine che va a definire le zone a diversa densità di risoluzione. La funzione di FFR è eseguita a livello hardware, non è una particolare funzione di Unity.

In alcuni casi, soprattutto in presenza di texture molto dettagliate o di testo a contrasto elevato, si può incorrere in qualche artefatto grafico, ma nelle normali situazioni la differenza con l'immagine originale non è poi così evidente.

5.2 Analisi delle performance in VERA

Dopo aver visto una panoramica generale delle principali tecniche di ottimizzazione nel caso mobile, in questa sezione ci si concentrerà su quanto adottato durante la produzione di VERA.

Per ottenere il massimo nelle performance, è stata messa a punto una particolare pipeline di produzione degli asset, che permettesse di sfruttare al massimo (ove possibile) la tecnica dello *static batching*, per la riduzione delle draw call e del tempo di CPU. A questa, sono state affiancate tecniche come il *light baking*, il *mesh baking*, una modalità di rendering di tipo *single-pass stereo* e sono stati evitati completamente gli effetti di *post-processing* dell'immagine.

Altre tecniche potenzialmente interessanti, come ad esempio l'*occlusion culling*, sono state accantonate in ultima analisi, perché mal si adattavano alla particolare conformazione delle scene in Unity. Come già detto, nel processo di ottimizzazione non esiste una ricetta universale, applicabile in tutti

i casi. È, piuttosto, un processo che va studiato caso per caso, trovando una soluzione su misura.

Nei successivi paragrafi, prenderemo come caso d'esempio l'ottimizzazione degli asset e delle scene del capitolo di Rabbia, in quanto sufficientemente rappresentativo delle tecniche adottate globalmente all'interno del progetto.

5.2.1 Produzione ottimizzata degli asset

Un primo importante punto di partenza, da non sottovalutare nella definizione della strategia di ottimizzazione, è relativo allo stile grafico che si intende adottare, all'interno della propria applicazione.

Si è scelto, infatti, di optare per uno stile non-fotorealistico e "stilizzato", cosa che pone sicuramente delle sfide in termini di design grafico e direzione artistica, in quanto è normalmente più semplice aggiungere, che semplificare. Ma, allo stesso tempo, si ha da subito un guadagno dal punto di vista del carico computazionale, poiché gli asset saranno costituiti da un numero inferiore di vertici, in più materiali e texture saranno necessariamente semplificati.

Perciò, si vedrà che tutti gli asset di VERA sarannò perlopiù *low poly* (ovvero con un basso numero di poligoni) e con uno shading *flat* (ovvero "piatto" e semplice).

La produzione degli asset inizia in un software di modellazione 3D come Blender, dove vengono creati i modelli, con le loro coordinate UV, attraverso il processo di *UV unwrapping*. Qui, vengono anche associate le varie mesh ai relativi materiali, che poi però andranno rimappati all'interno di Unity.

Come si è visto, per sfruttare al massimo il batching statico, è necessario che venga preservato il contesto di rendering per la GPU, il che vuol dire che è necessario mantenere il più possibile gli stessi materiali in scena, se possibile condividendoli tra più mesh.

Per questo, si è cercato di creare un unico grande materiale condiviso tra gli asset di Rabbia, che sfruttasse la tecnica del *texture atlasing*. Il risultato è il *Rabbia_Atlas_mat*, un materiale universale, che sfrutta nel suo canale di *albedo* (per semplicità, il colore del materiale) una *palette texture* che ospita tutte le variazioni cromatiche del capitolo di Rabbia.

In questo modo, con un unico materiale e con un'adeguata preparazione delle coordinate UV di ciscun asset, si può dare colore a tutti gli oggetti del capitolo (fatta eccezione per quelli che necessitano di uno shader particolare, ad es. gli edifici), come mostrato in figura 74.

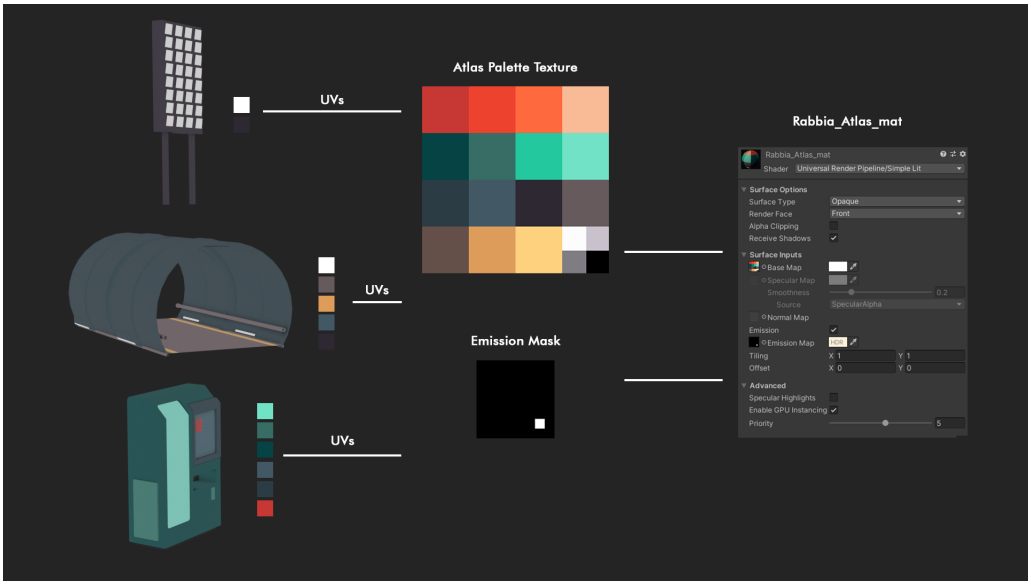

Figura 74: Il materiale di Atlas per gli asset di Rabbia

Tale approccio porta enormi vantaggi, poiché consente di ottenere varietà, coerenza stilistica e performance, a partire da una semplice texture di piccole dimensioni, che dunque non pesa troppo nel budget della memoria.

Inoltre, si creano delle condizioni ideali per il batching statico e per il mesh baking: si ha un contesto di rendering che non cambia e vi è la possibilità di combinare anche mesh diverse tra loro, poiché condividono lo stesso materiale.

Infine, poiché lo shader utilizzato all'interno del materiale è il *Simple Lit* fornito dalla Universal Render Pipeline, anche il calcoli intrinseci per la resa del materiale sono semplici ed efficienti.

5.2.2 Un esempio di confronto numerico

Per dimostrare l'efficacia delle tecniche di ottimizzazione adottate, verrà preso come esempio un frame da una delle scene più complesse dell'intera produzione: la città di Rabbia.

Si effettuerà un confronto chiamate di rendering necessarie per renderizzare la scena prima e dopo il processo di ottimizzazione. Le statistiche sono ottenute utilizzando il frame debugger di Unity.

Va tenuto a mente che, poiché tali test sono stati effettuati all'interno dell'editor di Unity, i risultati ottenuti potrebbero essere leggermente differenti rispetto a quelli interni prodotti in fase di esecuzione su Oculus GO. Ad ogni modo, quanto indicato dal *frame debugger* dovrebbe essere sufficientemente indicativo di quanto possa avvenire anche sul device.

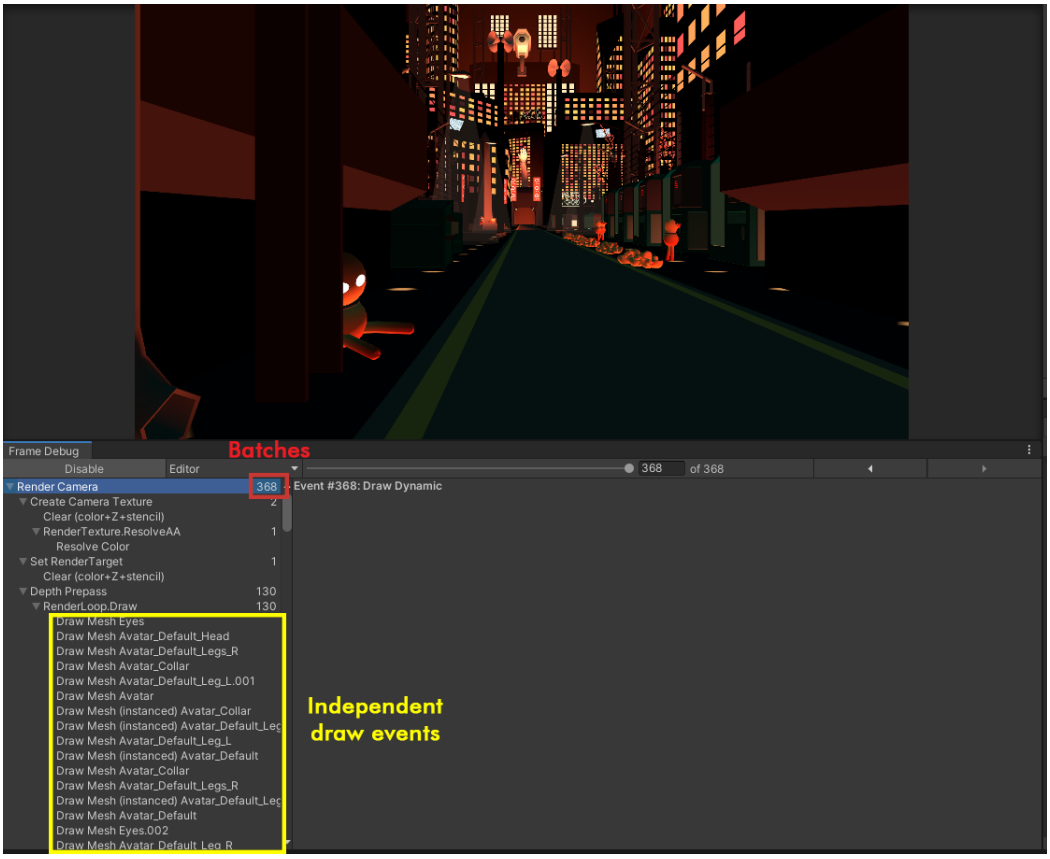

Figura 75: La città di Rabbia prima dell'ottimizzazione

Inoltre, all'interno di Unity si è notata una discrepanza tra il numero di eventi di rendering indicati dal frame debugger e quelli mostrati nella tabella delle statistiche in game view.

Si ipotizza che ciò sia dovuto ad una non ancora perfetta integrazione tra tale pannello di analisi e le ottimizzazioni offerte dalla nuova pipeline di rendering. Infatti, si è visto che l'attivazione dello *SRP Batcher*, ovvero la nuova funzionalità per il batching statico, ottimizzato per la Scriptable Render Pipeline, non determina una variazione numerica coerente anche nelle draw call indicate nelle *stats*.

Si è deciso dunque, di far fede a quanto esplicitato dal frame debugger per il confronto.

Il confronto è stato effettuato utilizzando un frame della medesima scena della città di V-Rage, con le seguenti differenze di configurazione:

- in una versione non ottimizzata (Fig.75), l'opzione per lo static batching è disattivata, ogni oggetto ha un proprio materiale individua-

le, le luci sono in real-time e non esistono cluster di mesh combinate con Mesh Baker. In questo caso, tutti gli eventi di rendering sono indipendenti tra loro e lo stato della GPU cambia in continuazione;

- nella versione ottimizzata (Fig.76), è stato attivato lo SRP Batcher, gli oggetti fermi sono marcati come statici, viene utilizzato il materiale condiviso *Rabbia_Atlas_mat* per tutte le mesh che non richiedono uno shading particolare, l'illuminazione è ottenuta attraverso il light baking e più mesh sono state combinate tra loro attraverso il processo di mesh baking. Qui, più oggetti con lo stesso materiale possono essere raggruppati in un *group batch* e disegnati a schermo in una volta sola.

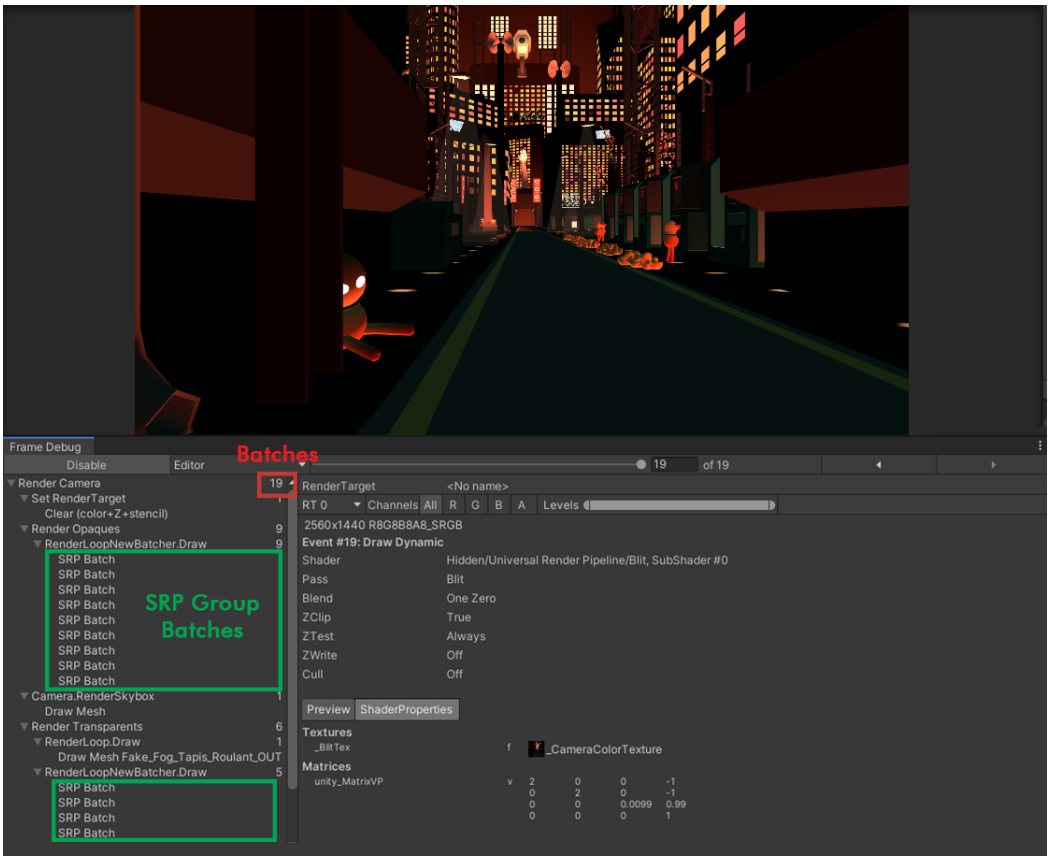

Figura 76: La città di Rabbia dopo il processo di ottimizzazione

Come si può facilmente notare, la differenza tra gli eventi di draw prima e dopo l'ottimizzazione è piuttosto sensibile: si passa infatti da oltre 350 draw call a meno di 20.

Attraverso tale esempio concreto, si può vedere la potenza, soprattutto in un contesto mobile e CPU bound, delle tecniche di atlasing, baking e batching discusse.

6 Sperimentazione e Test

Il progetto VERA, fin dai suoi albori, includeva nel piano una fase di sperimentazione, necessaria a testare sul campo i diversi tipi di risposta da parte del pubblico, in modo da capire se un'esperienza simile (multi-utente e co-locata) potesse stimolare la curiosità e costituire un valore aggiunto, rispetto alle tradizionali applicazioni VR.

Oltre a tale principale motivazione, era necessario anche capire, in un contesto concreto, se le soluzioni tecniche ideate ed implementate fossero affidabili e scalabili, anche per i potenziali eventi futuri.

Tuttavia, a causa dell'emergenza sanitaria di inizio 2020, la gran parte degli appuntamenti di sperimentazione in gruppo schedulati per il primo semestre dell'anno, hanno dovuto subire delle cancellazioni.

Perciò, è stato necessario riadattare, in corso d'opera, la quantità e la modalità di esecuzione dei test necessari a VERA. Si è scelto, in ultima analisi, di utilizzare esclusivamente il capitolo di Quiete in questa prima fase di sperimentazione, in quanto si trattava del capitolo più avanzato nello sviluppo, al momento della decisione. Inoltre, esso costituiva un test significativo, in termini di lunghezza dell'esperienza, delle possibili problematiche di comfort e di percezione e immersione.

6.0.1 Le prime sperimentazioni col pubblico

Il capitolo di Quiete è stato sottoposto per la prima volta al pubblico verso la fine di Settembre 2020, con un totale di 7 repliche da 30 partecipanti ciascuna, in due contesti performativi, tenutisi rispettivamente presso:

- l'hub culturale torinese *OffTopic*, il 23 settembre (3 repliche);
- il *Circolo dei Lettori di Torino*[37], nella cornice del festival *Torino Spiritualità*, il 26 e il 27 settembre (4 repliche).

In figura 77, 78 e 79 sono raffigurate alcune delle immagini della performance.

Al termine delle stesse, è stato inviato al pubblico partecipante un particolare questionario, il cui scopo e contenuto verrà discusso in seguito.

6.1 Questionario di test e risultati

Il questionario di test sottoposto al pubblico è costituito da un totale di 30 domande, con risposte da dare in *scala Likert* da 1 a 5 ("completo disaccordo" - "completo accordo"). Esso mira ad analizzare la risposta del pubblico, per capire:

[37] https://torinospiritualita.org/vera-virtual-experience-real-approach-quiete/

Figura 77: VERA ad OffTopic, il 23 Settembre

Figura 78: VERA al Circolo dei Lettori, il 26 e 27 Settembre

- se la durata dell'esperienza è adeguata;
- quale livello di immersione e presenza ha raggiunto l'utente;
- con che effetto viene percepita la co-presenza altrui;
- qual è la soddisfazione generale dell'utenza e la propensione futura a ripetere un'esperienza VR simile.

Per facilitare l'analisi, il questionario è stato diviso in quattro sezioni principali:

1. *Background dell'utente* - in cui si chiedono informazioni anagrafiche e il grado di esperienza con le tecnologie VR;

Figura 79: VERA al Circolo dei Lettori, il 26 e 27 Settembre

2. *Durata e comfort* - in cui si indaga la risposta dell'utente alla lunghezza del capitolo, al comfort dell'applicazione e del visore, l'eventuale motion sickness etc.;

3. *Immersione e presenza* - in cui viene valutato il grado di successo dell'esperienza immersiva, indagando la risposta agli stimoli audio-visivi e all'influenza del contesto di fruizione;

4. *Percezione del capitolo e valutazione* - in cui si testa la percezione della chiarezza degli elementi narrativi specifici del capitolo e in cui si valuta globalmente il grado di soddisfazione dell'utente.

Per delineare soprattutto le domande delle sezioni 2 e 3, si è preso spunto da questionari standard per la valutazione della percezione delle esperienze VR, come ad esempio il *Presence Questionnaire* di Witmer & Singer [12].

In totale, si sono raccolti dati per 98 utenti di test. Il pubblico risulta diviso in 3 principali fasce d'età: 18-35 (54,1%), 36-50 (26,5%) e over 50 (19,4%). Generalmente, l'utenza si dimostra poco esperta di tecnologie VR (83.7% non usa regolarmente strumenti simili).

Per quanto riguarda il feedback in termini di durata e comfort, ci si può dire soddisfatti del riscontro ottenuto, specialmente se si considera il fatto che Quiete rappresenta il capitolo più lungo di VERA: l'83.7% infatti dichiara di approvare la durata dell'esperienza, il 55.1% di aver trovato il visore confortevole e solo il 10.2% ha lamentato una qualche forma di motion sickness.

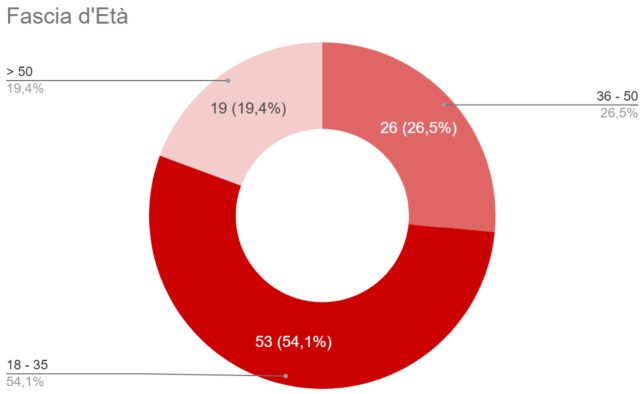

Figura 80: Fasce d'età

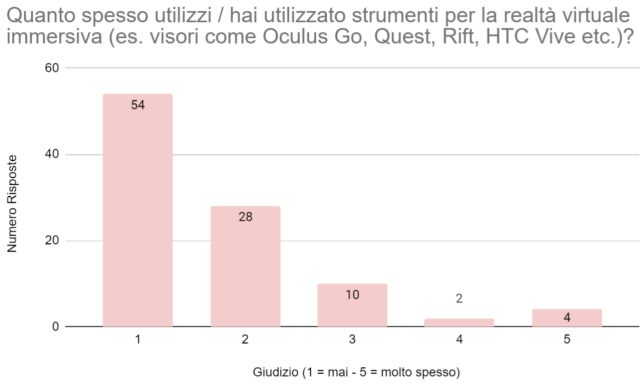

Figura 81: Esperienza con tecnologie VR

Figura 82: Feedback sulla durata dell'esperienza

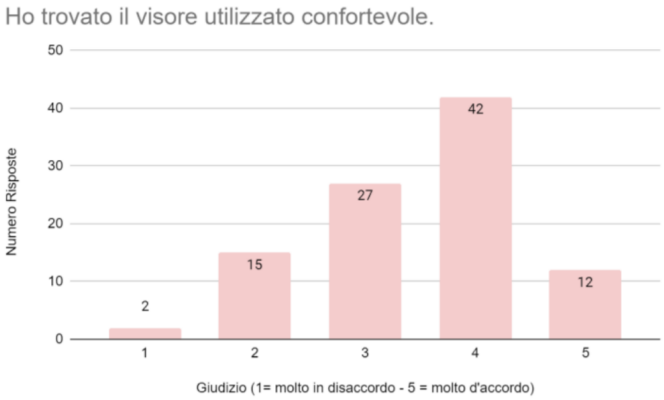

Figura 83: Feedback sul comfort del visore

Figura 84: Motion sickness

Anche per quanto riguarda le sensazioni di immersione e presenza, si sono ottenuti dei risultati incoraggianti: il 67,3% si è infatti sentito "immerso" nel mondo virtuale, il 76,5% degli intervistati si è detto "soddisfatto" del livello di immersione raggiunto, con un 54% che dichiara addirittura di aver perso la cognizione del tempo.

L'effetto della presenza altrui si è rivelato benefico in termini di immersione per il 56,1% dei partecipanti e il contesto performativo multi-utente è stato giudicato preferibile, rispetto ad una fruizione individuale, dal 77,5% del pubblico.

Infine, ben il 77,5% degli utenti ha espresso un giudizio complessivo positivo sull'esperienza svolta e una percentuale ancora maggiore (81,6%) si dichiara disposta a ripetere un'esperienza simile in futuro.

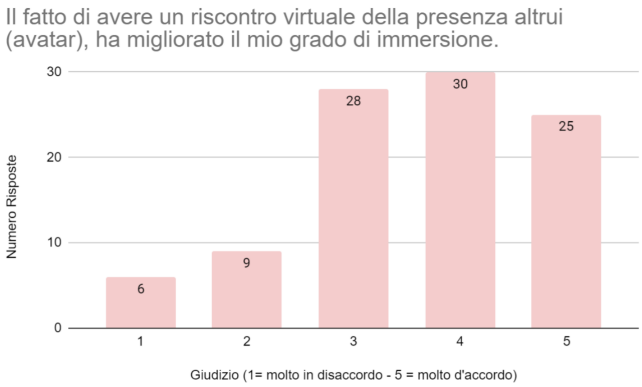

Figura 85: Percezione altrui

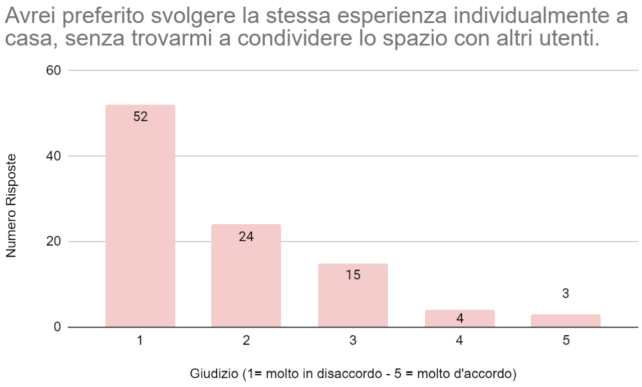

Figura 86: Attitudine all'esperienza di gruppo

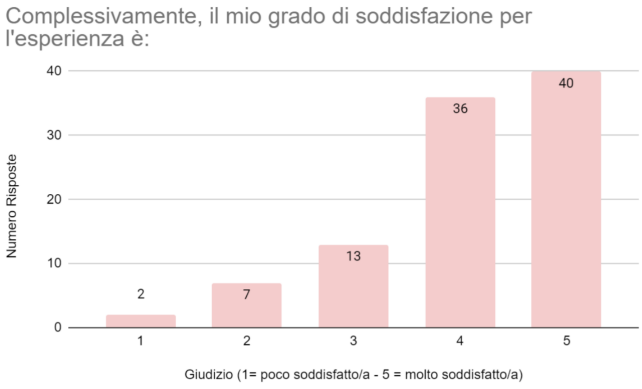

Figura 87: Soddisfazione generale

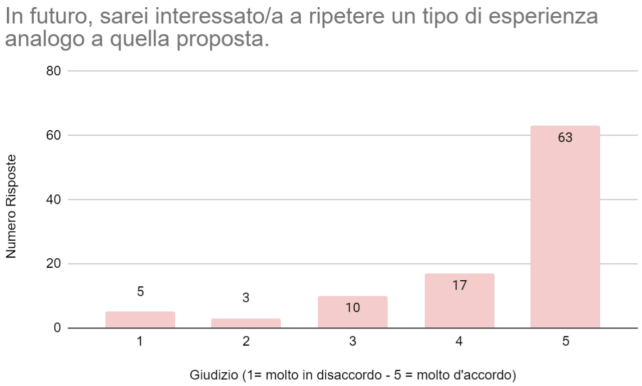

Figura 88: Predisposizione futura

7 Conclusioni

La scelta di iniziare a collaborare ad un progetto come VERA, oltre che mossa da una motivazione accademica, è stata portata avanti anche (e soprattutto) per un sentimento di sana curiosità e voglia di cimentarsi con un'opera ambiziosa, sperimentale e concretamente realizzabile.

L'idea di lavorare a dei contenuti che puntassero a sovvertire la percezione comune della realtà virtuale, spesso intesa come un qualcosa di esclusivo per pochi appassionati e di implicitamente associato all'isolamento dell'utente in un mondo virtuale surrogato, ha costituito una delle motivazioni principali.

Si crede, infatti, che tale tecnologia nasconda ancora molte potenzialità espressive inesplorate, soprattutto nelle sue applicazioni più orientate al mondo performativo, narrativo e dell'intrattenimento. In tali contesti, è possibile creare dei contenuti multimediali ibridi (VERA ne è un esempio), che permettono, se non di superare le modalità di fruizione tradizionali, quantomeno di fornire una percezione totalmente nuova e diversa per il pubblico che li sperimenta.

Inoltre, la creazione di contenuti narrativi (e non ludici, ad esempio), i quali non richiedono necessariamente una dimestichezza eccessiva per essere goduti con soddisfazione da un utente medio, si ritiene che possa favorire l'abbattimento delle barriere di ingresso per il grande pubblico e contribuire alla diffusione e "democratizzazione" della realtà virtuale su più larga scala.

Dal punto di vista dell'esperienza di produzione, è stato particolarmente interessante e formativo confrontarsi con gli onnipresenti limiti tecnici, i quali sono una costante quando si lavora in VR, ancor più quando si utilizza una tecnologia mobile. Questi hanno spesso costretto il team a trovare delle soluzioni creative ottimizzate e a costruire una pipeline di lavoro, secondo un approccio *bottom-up*, attraverso prove costanti, sperimentazioni ed errori.

Oltre alle questioni meramente tecniche, un altro aspetto fondamentale è stato quello umano della mutua interazione e comunicazione in un team multi-disciplinare, oltretutto alla prima esperienza assoluta con un progetto simile.

Il fatto di provenire da background molto diversi, ha portato un grande valore aggiunto ai processi creativi e di scambio di idee; tuttavia ha anche richiesto un periodo di rodaggio e di adattamento reciproco, trovando un "linguaggio" comune, che fosse comprensibile da tutti i membri della produzione. Una volta però superate le prime difficoltà e trovato l'equilibrio comunicativo adatto, la lavorazione del progetto e i processi di confronto interni al team, ne hanno beneficiato enormemente.

Volendo isolare una singola lacuna in merito al team, si segnala la mancanza di una figura di *concept artist*, che avrebbe potuto portare ad una lavorazione sicuramente più fluida, creando un maggior raccordo tra il com-

parto tecnico e quello creativo.

Le prime performance pubbliche presso OffTopic e Circolo dei Lettori sono state molto soddisfacenti, sia per il team di produzione, che ha visto funzionare nel concreto il prodotto, senza intoppi critici, sia per il pubblico presente agli spettacoli, il cui giudizio si è rivelato sorprendentemente positivo nella risposta ai test sottoposti.

In un'ottica di potenziali miglioramenti e arricchimenti futuri per la performance, va senza dubbio citata la possibilità di sviluppare nuovi capitoli per espandere il "corpus" di VERA. Il fatto di aver già sperimentato molte delle difficoltà associate alla produzione di contenuti per la realtà virtuale, costituisce un punto di partenza importante, che permetterebbe di affrontare un nuovo lavoro con uno sguardo e un'esperienza differenti.

Dal punto di vista hardware, considerate anche le ultime indiscrezioni secondo le quali Oculus starebbe pensando di puntare fortemente sui nuovi visori *Oculus Quest*, l'idea di un upgrade dei visori utilizzati durante la performance, sarebbe senz'altro desiderabile. I visori *Quest*, infatti, sono dotati di 6 gradi di libertà, di una tecnologia di tracking per le mani, nonché di minor peso e di maggiore ergonomia per l'utente, rispetto ai più modesti Oculus Go.

Ciò, potrebbe inoltre consentire la sperimentazione di nuove tecniche di interazione, poiché l'utente potrebbe utilizzare naturalmente le proprie mani per interagire col mondo virtuale, senza necessità di maneggiare un controller.

In conclusione, per quanto ci siano ancora potenzialmente diversi dettagli da migliorare, aspetto oltretutto abbastanza normale per un team alla prima esperienza del genere, ci si può dire soddisfatti del viaggio che ci ha portati alla conclusione della prima versione di VERA, soprattutto perché si tratta di un qualcosa di sperimentale e, a suo modo, d'avanguardia per il panorama italiano, nella sua fusione di musica, realtà virtuale e storytelling.

Riferimenti bibliografici

[1] Alan S. Cowen and Dacher Keltner. Self-report captures 27 distinct categories of emotion bridged by continuous gradients. *Proceedings of the National Academy of Sciences*, 2017.

[2] Connor DeFanti, Davi Geiger, and Daniele Panozzo. Co-located augmented and virtual reality systems. 2019.

[3] David Gochfeld, Corinne Brenner, Kris Layng, Sebastian Herscher, Connor DeFanti, Marta Olko, David Shinn, Stephanie Riggs, Clara Fernández-Vara, and Ken Perlin. Holojam in wonderland: Immersive mixed reality theater. In *ACM SIGGRAPH 2018 Art Gallery*, SIGGRAPH '18, page 362–367, New York, NY, USA, 2018. Association for Computing Machinery.

[4] Lawrence J. Hettinger and Gary E. Riccio. Visually induced motion sickness in virtual environments. *Presence: Teleoperators and Virtual Environments*, 1(3):306–310, 1992.

[5] Michael Lankes, Jüergen Hagler, Georgi Kostov, and Jeremiah Diephuis. Invisible walls: Co-presence in a co-located augmented virtuality installation. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '17, page 553–560, New York, NY, USA, 2017. Association for Computing Machinery.

[6] Kris Layng, Ken Perlin, Sebastian Herscher, Corinne Brenner, and Thomas Meduri. Cave: Making collective virtual narrative. In *ACM SIGGRAPH 2019 Art Gallery*, SIGGRAPH '19, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Stefan Liszio and Maic Masuch. Designing shared virtual reality gaming experiences in local multi-platform games. In Günter Wallner, Simone Kriglstein, Helmut Hlavacs, Rainer Malaka, Artur Lugmayr, and Hyun-Seung Yang, editors, *Entertainment Computing - ICEC 2016*, pages 235–240, Cham, 2016. Springer International Publishing.

[8] David Lobser, Ken Perlin, Lily Fang, and Christopher Romero. Flock: A location-based, multi-user vr experience. In *ACM SIGGRAPH 2017 VR Village*, SIGGRAPH '17, New York, NY, USA, 2017. Association for Computing Machinery.

[9] Gerald Oster. Auditory beats in the brain. *Scientific American*, 229(4):94–103, 1973.

[10] Misha Sra, Ken Perlin, Luiz Velho, and Mark Bolas. Novel interaction techniques for collaboration in vr. In *Extended Abstracts of the 2018*

CHI Conference on Human Factors in Computing Systems, CHI EA '18, page 1–8, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Jayant Thatte and Bernd Girod. Towards perceptual evaluation of six degrees of freedom virtual reality rendering from stacked omnistereo representation. *Electronic Imaging*, 2018(5):352–1, 2018.

[12] Bob G. Witmer and Michael J. Singer. Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoperators and Virtual Environments*, 7(3):225–240, 1998.

Sitografia

[1] Felipe Lira, Andre McGrail, 2019, How the Universal Render Pipeline unlocks games for you - Unite Copenhagen, https://www.youtube.com/watch?v=ZPQdm1T7aRs&t=584s

[2] Piotr Korzuszek, 2016, Choosing Between Forward or Deferred Rendering Paths in Unity, http://blog.theknightsofunity.com/forward-vs-deferred-rendering-paths/

[3] Brackeys, 2017, Intro To Unity Timeline, https://www.youtube.com/watch?v=G_uBFM3YUF4

[4] Ciro Continisio, 2018, Extending Timeline: A Practical Guide, https://blogs.unity3d.com/2018/09/05/extending-timeline-a-practical-guide/

[5] James Bouckley, 2017, Unite Europe 2017 - Extending Timeline Wiht Your Own Playables, https://www.youtube.com/watch?v=uBPRfcox5hE

[6] Game Dev Guide, 2019, Extending Timeline in Unity to make a Subtitle track , https://www.youtube.com/watch?v=12bfRIvqLW4

[7] Andy Borrell, 2015, Unity's UI System in VR, https://developer.oculus.com/blog/unitys-ui-system-in-vr/

[8] Chris Pruett, 2018, Unite Berlin 2018 - Going for Speed Maximizing Performance on Oculus Go, https://www.youtube.com/watch?v=BQyZ9yYWT18&t=864s

[9] Remi Palandri, Samuel Gosselin, and Chris Pruett, 2018, Optimizing Oculus Go for Performance, https://developer.oculus.com/blog/optimizing-oculus-go-for-performance/

[10] Ciro Continisio, 2019, What's new in Shader Graph - Unite Copenhagen, https://www.youtube.com/watch?v=L6mU2rkT5To

[11] Gabor Szauer, 2018, Look Ma, No Jutter! Optimizing Performance Across Oculus Mobile - Unite LA, https://www.youtube.com/watch?v=JUiO64KbAHA

[12] Gabor Szauer, 2018, Tech Note: Unity settings for mobile VR https://developer.oculus.com/blog/tech-note-unity-settings-for-mobile-vr/

[13] Unity Technologies, 2020, Nine ways to optimize your game development - Unity expert tips to help you ship (eBook), https://create.unity3d.com/nine-ways-to-optimize-game-development

[14] Rubén Torres Bonet, 2020, Unity Draw Call Batching: The Ultimate Guide, https://thegamedev.guru/unity-performance/draw-call-optimization/

[15] Lucas Rizzotto, 2019, Optimizing Your Unity VR Game (14 Tips and Tools), https://www.youtube.com/watch?v=w0n4fuC4fNU&list=PL-6ROxZ1TRcv1Ry5wx8Bfu0a_xZV2_MX8&index=14&t=105s

[16] Umbra 3D, 2018, Introduction to Occlusion Culling, https://medium.com/@Umbra3D/introduction-to-occlusion-culling-3d6cfb195c79

[17] Kristina Hougaard, 2013, Occlusion Culling in Unity 4.3: The Basics, https://blogs.unity3d.com/2013/12/02/occlusion-culling-in-unity-4-3-the-basics/

[18] Jennifer Nordwall, 2017, Unite Europe 2017 - Bake it 'til you make it: An intro to Lightmaps, https://www.youtube.com/watch?v=u5RTVMBWabg

[19] Jennifer Nordwall, 2019, Baking at the speed of light: Lightmapping for beginners - Unite Copenhagen, https://www.youtube.com/watch?v=pcXpY_IBSeU

[20] Ciro Continisio, 2020, Harnessing Light with URP and the GPU Lightmapper | Unite Now 2020, https://www.youtube.com/watch?v=hMnetI4-dNY