Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica
A.a. 2020/2021
Sessione di laurea di Ottobre 2021

Look development per la realizzazione di una sequenza di apertura in CGI con Blender

Relatore:
Prof. Riccardo Antonio Silvio Antonino

Candidato:
Edoardo Mario Amico

Sommario

La trattazione segue le varie fasi di produzione di diverse scene in CGI per la realizzazione di una sequenza di apertura per uno spettacolo teatrale, sotto la direzione dell'azienda di produzione video Robin Studio. All'interno di questa tesi vengono descritte le varie scelte tecniche effettuate nell'ambito del look development finalizzato al fotorealismo, utilizzando come strumento principale il software open source Blender. È stata data particolare enfasi alla descrizione di un workflow basato su di un motore di rendering real-time (EEVEE), su diverse tecniche di modellazione e shading procedurale e sull'ottimizzazione e automazione di alcune fasi di produzione utilizzando Python come linguaggio di scripting.

Indice

Elenco delle figure

Introduzione

La CGI è un ambito della computer grafica sempre più in espansione, con numerosi ambiti di utilizzo. Uno dei principali è ovviamente l'industria cinematografica, nella quale i finanziamenti verso questo tipo di produzione sono molto elevati. Non esistono però solamente le famose case di produzione, ma anche realtà più piccole che si possono definire indipendenti. In questi casi la pipeline di produzione segue necessariamente un percorso diverso, con la partecipazione di figure professionali che vengono definite *generalist*, ossia non con una propensione verso un ambito particolare. Questa tesi ha permesso di partecipare a una produzione in CGI in una azienda di piccole dimensione, con particolare focus (ma non esclusivo) sul look development.

Il look development è una fase fondamentale dello sviluppo di un prodotto audio visivo e consiste nel definire lo stile principale di quest'ultimo, con particolare attenzione alla coerenza tra le varie scene che lo compongono. All'interno di questa tesi, viene descritto il lavoro che è stato svolto durante la produzione di una serie di scene di una sequenza di apertura in CGI per lo spettacolo teatrale *Snow White – La Leggenda*, avente come oggetto la favola di Biancaneve. Il look development in questo caso ha avuto come obbiettivo principale il fotorealismo, ed è stato utilizzato per la maggior parte il software open-source Blender e il motore di rendering real-time EEVEE. La trattazione della tesi, quindi, descrive in prima battuta lo stato dell'arte relativo alla produzione di openings, descrivendone gli stili principali e le tendenze recenti; inoltre, viene descritto il tipico workflow di un prodotto audio visivo, suddiviso nelle fasi di preproduzione, produzione e post-produzione. Segue una panoramica sulla disciplina del look development e delle scelte preliminari fatte per il progetto in questione, quali la decisione su quali software utilizzare (principalmente Blender per modellazione, shading procedurale e rendering e Substance Painter per il texturing di alcuni modelli), focalizzando poi l'attenzione sulla descrizione delle varie tipologie dei motori di rendering attualmente disponibili e sulle motivazioni riguardanti la scelta di utilizzare un motore di rendering real-time quale è EEVEE. Pur avendo molti punti di forza, solitamente un motore di rendering di questo tipo si scontra con le necessità legate al raggiungimento del fotorealismo, proprio per alcuni limiti tecnici intrinsechi della grafica di tipo raster. Vengono quindi presentate alcune tecniche che possono essere utilizzate per ottenere un risultato soddisfacente ed equiparabile a quello prodotto da un motore di rendering basato sul ray tracing. In questo senso, per quanto riguarda la modellazione, viene descritto l'utilizzo sempre più frequente, sia in CGI che in altri ambiti, della tecnica della fotogrammetria, che

permette di ottenere con relativa facilità dei modelli estremamente realistici. Si passa quindi alla fase di preproduzione del progetto, con la descrizione delle references stilistiche sia per quanto riguarda la tipologia di opening che era necessario realizzare, sia per le ambientazioni da ricreare: data la natura e la tematica del progetto, è stato quindi fondamentale confrontarsi con alcuni prodotti audio visivi celebri del genere fantasy. Il capitolo principale è relativo alla produzione vera e propria, in cui vengono descritte le varie scene realizzate. È stato infatti richiesto di realizzare diverse scene e assets, ognuno dei quali ha visto l'applicazione di varie tecniche, sia per quanto riguarda la modellazione che lo shading. Gran parte del lavoro del look development ha interessato in particolare questo ultimo aspetto: sono stati infatti realizzati shader procedurali, che permettono, con un utilizzo limitato o assente di textures, di ottenere materiali che hanno come punto di forza la possibilità di essere modificati significativamente attraverso pochi parametri. Alcune delle scene, inoltre, hanno previsto l'utilizzo di tecniche di simulazione di fluidi mediante il motore Mantaflow, nonché la presenza di sistemi particellari per la realizzazione di terreni con piante ed erba e soprattutto la realizzazione di diversi sistemi di tipo hair per il modello di un personaggio rappresentante un nano; in particolare, per quest'ultimo caso, viene descritto lo stato dell'arte riguardante le principali difficoltà nell'ambito della grafica 3D nella rappresentazione realistica di barba e capelli. Viene presentata anche un'importante alternativa ai sistemi particellari di recente introdotta in Blender, ossia i Geometry Nodes, che aprono le porte a un nuovo tipo di modellazione procedurale dalle infinite potenzialità. È stato inoltre dato particolare interesse alla trattazione di alcune tecniche di ottimizzazione e automazione di alcune fasi di produzione, grazie alla possibilità che Blender offre, data la sua natura open-source, di utilizzare il potente linguaggio di scripting Python, che permette di modificare le funzionalità del software e di aggiungerne di nuove a piacimento.

Obbiettivi

La produzione ha richiesto la realizzazione di diverse scene, molto diverse tra loro. Ogni scena ha permesso di approfondire e toccare con mano diversi aspetti e problematiche nell'ambito della computer grafica. Obbiettivo principale della tesi è quindi descrivere l'esperienza di produzione, approfondire alcuni aspetti di questa branca dell'informazione, e soprattutto descrivere gli strumenti che sono stati utilizzati nelle varie occasioni, evidenziandone la motivazione, i punti di forza ed eventuali compromessi.

Outline

Il lavoro di tesi svolto segue la seguente suddivisione:

1. **Gli opening credits nel cinema e serie TV**: in questo capitolo introduttivo viene descritto l'ambito cinematografico dedicato alla realizzazione di openings (che siano sequenze di apertura o titoli di testa). Viene quindi presentata una

tipica pipeline di produzione per un prodotto di questo tipo, focalizzando quindi l'attenzione sulla differenza tra una pipeline per un prodotto indipendente o al contrario seguito da una grande casa di produzione;

2. **Look development**: nel secondo capitolo viene descritta la disciplina del look development e le sue applicazioni e lo si rapporta al lavoro svolto nel progetto descritto nel seguito della tesi. In particolare, ci si concentra sulla descrizione delle varie tipologie di motori di rendering, sulla fotogrammetria e sulla scelta dei software utilizzati, in funzione degli obbiettivi preposti durante la fase di look development;

3. **Preproduzione**: in questo terzo capitolo si entra nella trattazione del lavoro progettuale svolto, descrivendo quindi il lavoro svolto in fase di preproduzione, in particolare nella ricerca di references stilistiche;

4. **Analisi delle scene**: si passa quindi alla produzione vera e propria. In ogni sezione del capitolo in questione viene descritto il procedimento e le scelte effettuate per la realizzazione delle varie scene, focalizzando l'attenzione sugli aspetti più critici di ognuna, così come eventuali approfondimenti sullo stato dell'arte rispetto ad alcuni argomenti.

Capitolo 1

Gli opening credits nel cinema e serie TV

1.1 Ruolo e cenni storici degli opening credits

Gli *opening credits*, o sequenze di apertura, costituiscono una parte importante nella realizzazione e nell'apprezzamento di un film o serie TV. Storicamente però, non è sempre stato questo l'approccio verso i titoli di testa, ai quali di solito veniva posto un interesse minore e un budget limitato, in base a quanto la produzione riusciva a risparmiare al termine del film [7]. Inoltre, in un primo momento, venivano semplicemente visti come un esercizio di *motion graphic*, che aveva poco a che vedere con il risultato e il successo di un prodotto. I primi esperimenti nell'ambito del cinema a fine '800, con i fratelli Lumière e Georges Méliès, sono privi di qualsiasi forma di titolo di testa. Il primo esempio riconosciuto è il cortometraggio muto "How it feels to be run over" (1900) di Cecil Hepworth, nel quale, al termine della visione, vengono mostrate alcune scritte, non inerenti però con la produzione. Con l'avvento dello *star system* e quindi delle celebrità del cinema intorno al 1910, i titoli di testa hanno iniziato a prendere forma, dando spazio al nome del regista e degli attori presenti e, soprattutto, della casa di produzione. La vera rivoluzione nelle sequenza di apertura si ha verso la metà del '900, con l'illustratore Saul Bass che, in particolare con le sue collaborazioni con Alfred Hitchcock, ha dato inizio al genere, comprendendo le potenzialità creative dei titoli di testa e cercando di coinvolgere il pubblico sin da subito. L'utilizzo della computer grafica in questo settore ha il suo primo riscontro nel 1978, con la sequenza iniziale di "Superman" di Richard Donner, in cui si può vedere un effetto di una scritta tridimensionale che attraversa la camera. Per un uso più frequente di questo tipo di animazioni si deve però aspettare gli inizi degli anni '90, con l'avvento dell'editing non-lineare e il *digital compositing*.

Tuttavia, sebbene nell'ambito del cinema i titoli di testa siano stati abbondantemente rivalutati nel tempo, con l'avvento delle serie TV e l'interesse sempre più crescente verso questo tipo di produzione da parte del pubblico, questi hanno subito una totale rivoluzione, e le risorse ad esse destinate sono state sempre maggiori. Ne è un considerevole esempio l'impegno rivolto alla celebre sequenza di apertura della

serie TV *Game Of Thrones*, alla quale hanno lavorato trentacinque persone per tre mesi di lavoro [8]: il *New York Times* ha commentato questa sequenza, dichiarandone l'inizio di un'"era degli *opening credits* elaborati". Il risultato ha sicuramente contribuito al successo della serie TV in questione, con un prodotto diventato decisamente iconico. Dato il numero sempre più crescente di serie TV, l'obbiettivo è quello di attirare il più possibile il pubblico sin dai primi istanti di visione, cercando di descrivere, in un tempo ristretto, il look visivo e la tematica del prodotto, così come i personaggi che ne saranno protagonisti.

1.2 Struttura e trend recenti

Prendendo in esempio le più recenti sequenze iniziali di famose serie TV, si può notare come la durata tipica è posta tra i 60 e i 90 secondi, con una durata delle varie scene di circa 5-7 secondi, con ovviamente alcune eccezioni decisamente più minimali, come la sequenza di *Lost* o di *Breaking Bad*, la cui durata è limitata a circa 15 secondi. In alcuni casi, la "storia" che viene narrata non è strettamente collegata a quanto verrà poi mostrato nella serie TV e non necessariamente quanto viene mostrato sarà presente nel corso delle puntate. L'obbiettivo tuttavia rimane riuscire a condensare in poco tempo il *mood* principale, cercando di rimarcarne lo stile. Quello che spesso viene mostrato è infatti relativo a oggetti di scena importanti a definire il contesto della narrazione. È presente un uso sempre più ampio della computer grafica e, in alcuni casi, esclusivo, come nel caso della già citata sequenza di *Game Of Thrones*. Molto spesso tuttavia, compaiono gli attori stessi della serie, quindi l'uso di computer grafica è limitato alla realizzazione dell'ambiente circostante e gli attori vengono calati nel contesto tramite utilizzo di green screen. Solitamente l'utilizzo della camera è molto lieve, con lenti movimenti di camera che si avvicinano agli oggetti di scena, effettuandone spesso dei close-ups.

1.3 Pipeline di produzione per un opening

Le tecnologie sulle quali la produzione di un opening si basa variano di molto in base al lavoro di look development che si è svolto, e quindi in base a quali sono gli obbiettivi definiti. Come si è visto, in alcuni casi il prodotto è effettuato totalmente in CGI, quindi la tipica pipeline è quella di un corto animato. Nella maggior parte delle situazioni tuttavia, si ha a disposizione del materiale girato e fotografato mediante l'utilizzo del green screen (o altre metodologie), che deve essere compositato utilizzando del materiale 3D, sia relativo a scene complete, sia di singoli modelli. In questo caso quindi bisogna tenere in conto della fase di integrazione delle due tipologie di materiale (2D e 3D), che si effettua quasi totalmente nella fase finale di post-produzione, ma che inevitabilmente influenza le scelte effettuate nelle fasi precedenti. Nel seguito quindi, verrà proposta una pipeline generica per questo tipo di prodotti, esplicitando anche quella che è stata l'esperienza diretta del progetto

che si è sviluppato e sul quale verte questo scritto, tenendone in conto della sua natura semi-indipendente, anche in relazione alla dimensione ridotta del team.

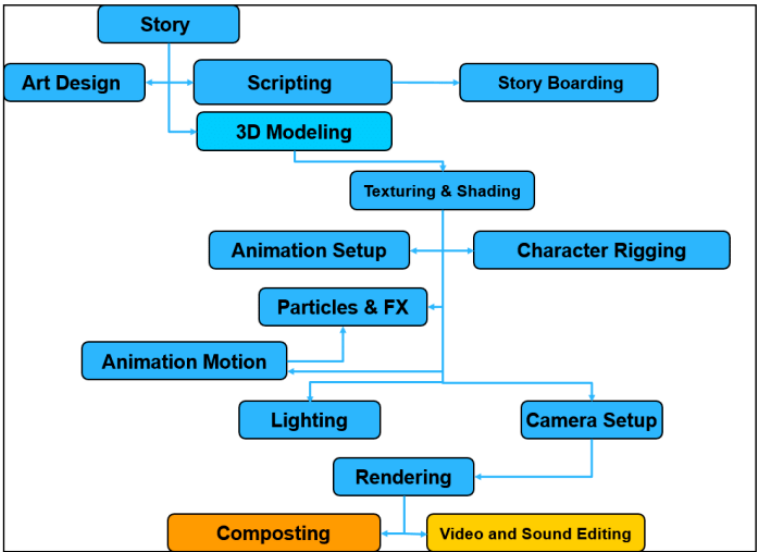

Figura 1.1: Pipeline di un progetto di animazione in 3D. Fonte: Bhatti, Zeeshan & Abro, Ahsan & Rehman, Abdul & Karbasi, Mostafa. (2017). Be-Educated: Multimedia Learning through 3D Animation. [3]

1.3.1 Preproduzione

Durante la fase di preproduzione si inizia a progettare il corto animato e si decidono le scene chiave che questo deve avere, in base anche al team che si occuperà della produzione. Per fare questo bisogna tenere in conto dell'obbiettivo del progetto e a che tipo di pubblico si rivolge. Bisogna quindi riflettere sul budget, sulla durata e sullo stile in generale che si vuole ottenere. Inoltre, avendo a che fare sia con scene 3D che con fotografie o girato, la preproduzione si concentra sul definire cosa deve essere modellato e in che stile, ma anche sul come effettuare lo shooting fotografico, l'utilizzo dei costumi, delle inquadrature e delle luci. Il principale output di questa fase è lo storyboard, ossia una serie di vignette disegnate in ordine cronologico, che descrivono in modo semplice le scene. Non esiste una linea guida universale per la stesura di uno storyboard, ma solitamente si esplicitano la scelta e la durata di ogni scena, le inquadrature, i movimenti di camera e la lunghezza focale, nonchè eventuali dialoghi tra personaggi. L'esecuzione di un buon storyboard è funzionale sia a livello di comunicazione del regista con il team che deve sviluppare le scene, sia al regista stesso al fine di mettere in luce cosa può andare o meno nella messa in scena. Ancora più interessante in questo senso è l'animatic, ossia una versione estremamente semplificata del prodotto finale; molto spesso, nella sua forma di base, consiste nel porre le immagini dello storyboard in sequenza, in modo da rispettare per ogni immagine la durata prefissata. Questo permette di avere un'idea precisa del timing di ogni scena e comprendere se modificare durata o ordine delle scene.

In seguito alla definizione delle scene, si può andare a esplicitarle meglio, focalizzandosi sui punti importanti di ognuna, sullo stile che si vuole ottenere e sulla definizione degli eventuali personaggi e del loro design. È già quindi in questa fase che inizia il lavoro del look development: come primo passo è necessario raccogliere quante più references possibile, per quanto riguarda l'ambientazione delle scene, il look visivo che queste devono avere (quindi scelta delle luci e colori) e comprendendo che tipologie di tecniche saranno necessarie per ottenere il risultato desiderato, sia esse di modellazione, texturing o legate ai VFX.

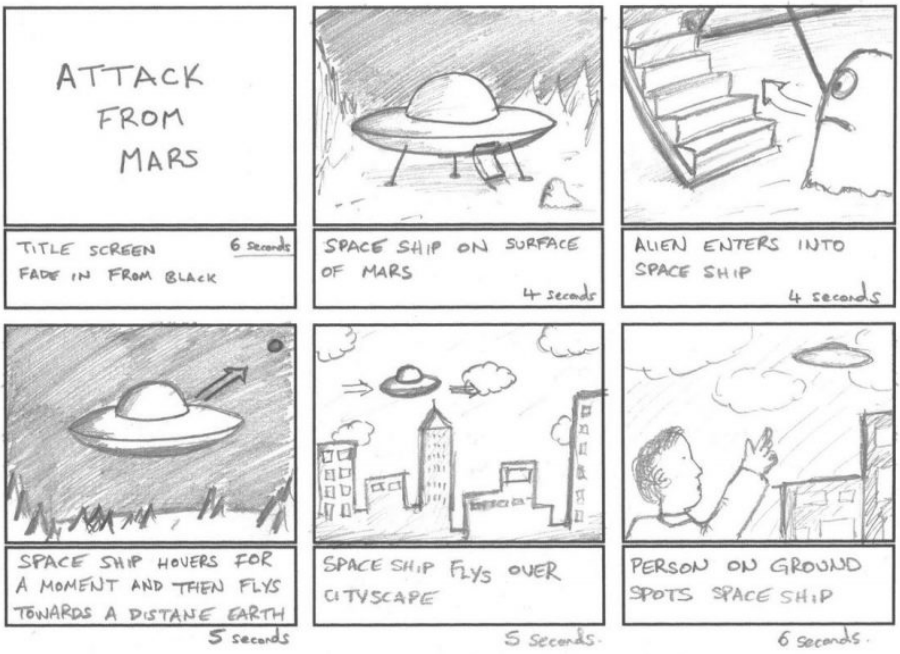

Figura 1.2: Esempio di storyboard. Fonte: Dream Farm Studios

1.3.2 Produzione

La fase di produzione è solitamente particolarmente densa e complessa. Si parte dal layout delle scene, in cui ne viene definita la composizione tramite forme primitive (cubi, cilindri, coni etc.). Questa fase è decisamente importante, al fine di definire le scale relative fra gli oggetti e iniziare con la parte successiva di modeling avendo bene in mente l'integrazione degli stessi nella scena. Il layout permette di ottenere una versione di animatic in 3D, utile come è stato già descritto per il timing delle scene, ma anche per quanto riguarda composizione e movimenti di camera. Si passa quindi alla modellazione degli oggetti da rappresentare. In questa fase è necessario fare delle scelte in base alla tipologia di oggetto che si vuole ottenere e al livello di dettaglio necessario. Di seguito vengono indicate alcune delle principali tecniche di modellazione:

- digital sculpting: si basa sul ricreare in maniera digitale la scultura tramite argilla (clay sculpting) del mondo reale. È una tecnica abbastanza recente e

permette agli artisti di focalizzarsi maggiormente sull'aspetto artistico e creativo, senza essere bloccati dalla componente tecnica. Questa tecnica produce modelli con un alto livello di dettaglio (high-poly), impossibili da usare nell'animazione immediatamente, ma da convertire in modello low-poly tramite retopology;

- boolean modeling: consiste nel combinare tra loro più elementi tramite operazioni booleane (unione, intersezione, differenza), creando un unico oggetto finale. È particolarmente usata in una fase preliminare di layout, in quanto solitamente produce una topologia non perfetta e che necessita di correzioni.

- fotogrammetria: Come descritto meglio nella sezione 2.3.1, la tecnica consiste nell'utilizzare una serie di fotografie di un oggetto da varie angolazione e ricavarne una rappresentazione 3D;

- laser scanning: Tecnica di modellazione innovativa che si basa sull'utilizzare il laser per definire le forme di un oggetto reale e convertirlo in 3D. È solitamente un processo abbastanza veloce e semplice ma che necessita di lavoro successivo al fine di migliorare la geometria generata, spesso molto densa;

- box modeling: È probabilmente la tecnica più utilizzata insieme al subd modeling. Consiste nel suddividere un oggetto in forme più semplici (solitamente cubi o cilindri) e partire da queste con la modellazione tramite estrusioni e trasformazioni lineari. I dettagli più fini vengono aggiunti in un secondo momento;

- subd modeling: È una tecnica di modellazione che permette di ottenere modelli con complessità e livello di dettaglio scalabili in base alle situazioni. Si parte da topologie semplici che vengono suddivise tramite diversi algoritmi (uno dei più diffusi è l'algoritmo definito da Catmull e Clarke) in base al livello di dettaglio richiesto. È spesso necesario avere una topologia che permetta la suddivisione, ossia che non generi risultati indesiderati (topologia quad-based);

- nurbs modeling: NURBS (Non-uniform rational B-spline) sono una rappresentazione matematica di modelli tramite curve smussate. Sono particolarmente utilizzate in modelli architettonici o in visualizzazione di prodotti.

Al termine della fase di modellazione, in base all'ambito di utilizzo dell'asset in questione e alla tecnica di modellazione utilizzata, si può scegliere di effettuare un processo di retopology, ossia la creazione di un modello low-poly avendo come punto di partenza un modello high poly. I dettagli più fini della modellazione high poly verranno riportati nella versione low poly tramite baking delle textures. Questa fase è particolarmente utile nell'ambito dell'animazione, in cui sarebbe impossibile lavorare con modelli riggati high poly, così come nella gestione di scene complesse in cui è preferibile avere un LOD (Level Of Detail) variabile in base alla distanza dalla camera.

Si passa quindi al texturing che consiste nel "vestire" il modello 3D con immagini 2D. Il tipico workflow del texturing consiste nell'effettuare in prima battuta l'UV unwrapping: presa in considerazione una superficie curva, questa viene appiattita su un piano sul quale giace un'immagine. Ad ogni vertice dell'oggetto viene quindi effettuato un mapping tra coordinate 3D e coordinate UV. In seguito, vengono definite le varie mappe dell'oggetto, che specificano le caratteristiche fisiche dello stesso e in che modo interagisce con le fonti luminose. La scelta della tipologia e del numero delle mappe dipende dalle caratteristiche che si intendono ottenere. Negli ultimi anni si è diffuso il workflow PBR (Phisically Based Rendering), essenziale nell'ambito del fotorealismo, che ha standardizzato la tipologia di textures e reso il processo di texturing maggiormente software independent. è possibile definire due differenti tipi di workflow, ossia Metallic-Roughness e Specular-Glossiness, entrambi supportati dai principali motori di rendering, la cui differenza consiste nella tipologia di mappe, spesso speculari tra loro ma equivalenti nella finalità.

In seguito al termine della scena da un punto di vista dei modelli da rappresentare e delle loro textures, è il momento di definire i VFX da realizzare. Questi di solito si suddividono in varie categorie, quali simulazione di particelle (ad esempio derivanti da esplosioni, gocce di pioggia o stormi di uccelli), simulazione dei fluidi, hair grooming e simulazione della dinamica dei peli, simulazione di corpi rigidi o di soft bodies, simulazione dei tessuti. Un tipico workflow in questo ambito consiste nell'effettuare un blocking dell'effetto che si vuole ottenere, decidendone quindi le dinamiche e le proporzioni all'interno della scena. Questa fase di fatto consiste in una simulazione molto semplificata, con una risoluzione abbastanza bassa. Nella fase di allestimento vera e propria, è prassi scomporre l'emissione delle particelle in elementi più semplici, creando quindi diversi sistemi particellari, ognuno con uno scopo ben preciso. Questo tipo di organizzazione del lavoro modulare permette di effettuare modifiche mirate su alcuni aspetti della simulazione, senza intaccare gli effetti già creati che possono essere definitivi.

Per ultimo, troviamo la fase di lighting e quindi di rendering. Queste fasi finali consistono nel posizionare le luci nella scena e posizionare al meglio la camera per creare una composizione interessante. È una fase fondamentale del processo produttivo, finalizzata a dare vita a ciò che è stato fatto finora, caratterizzando e dando senso alla scena che si è creata e indirizzando l'occhio dell'osservatore verso ciò che è importante. È fondamentale in questa fase che il posizionamento delle luci sia coerente con il girato, se presente (ad esempio: se una fotografia ritrae un personaggio illuminato dal lato sinistro dell'inquadratura, deve corrispondere questo tipo di illuminazione nella scena). Per quanto riguarda il rendering, valgono le osservazioni fatta nella sezione 2.6: esistono diverse possibilità riguardo la scelta del motore di rendering, che viene effettuata in base allo stile che si vuole ottenere, ma soprattutto in relazione ai tempi di calcolo e le risorse che sono disponibili in uno studio. In realtà indipendenti, spesso la scelta ricade su motori di tipo raster real-time, che, al momento attuale, permettono di ottenere ottimi risultati, anche se limitati rispetto alle potenzialità del ray tracing. La fase di rendering è inoltre strettamente legata con quella successiva, ossia la post-produzione: si preparano una serie di render

passes, che consistono nel suddividere la scena in gruppi di oggetti di qualsiasi tipo (ad esempio semplici meshes, così come luci o volumetrici), separando ciò che è logicamente distante l'uno dall'altro e sui quali si vuole ottenere una particolare libertà in fase compositiva.

1.3.3 Post-produzione

La fase di post-produzione è l'ultima in ordine nella pipeline e si occupa di combinare il materiale prodotto nella fase di produzione, aggiungendone elementi. Il un processo principale in questo senso è il *compositing*, durante il quale si può lavorare separatamente su ogni singolo layer (i render passes). Solitamente, come output della fase di produzione si realizzano diversi render passes che descrivono alcuni elementi separati della luce: quest'ultima viene scomposta nel suo contributo diffuso diretto e indiretto, nella componente responsabile delle riflessioni (*glossy*), anche in questo caso diretta e indiretta e nella componente trasmissiva, diretta e indiretta. Vengono spesso generati una serie di render passes aggiuntivi come il *z-depth pass* (mappa in scala di grigi in funzione della distanza dell'oggetto dalla camera, utile per ricreare l'attenuazione atmosferica in compositing) o il *normal pass*, che contiene le informazioni sulla normale della superficie. Di recente è stato aggiunto in Blender un canale definito *cryptomatte* che permette di creare un layer costituito da diversi colori univoci per oggetto o per materiale. È particolarmente utile in fase di compositing al fine di creare maschere solo per quello che interessa in quel momento, evitando di esportare un layer per ogni oggetto o materiale, che sarebbe decisamente più complesso.

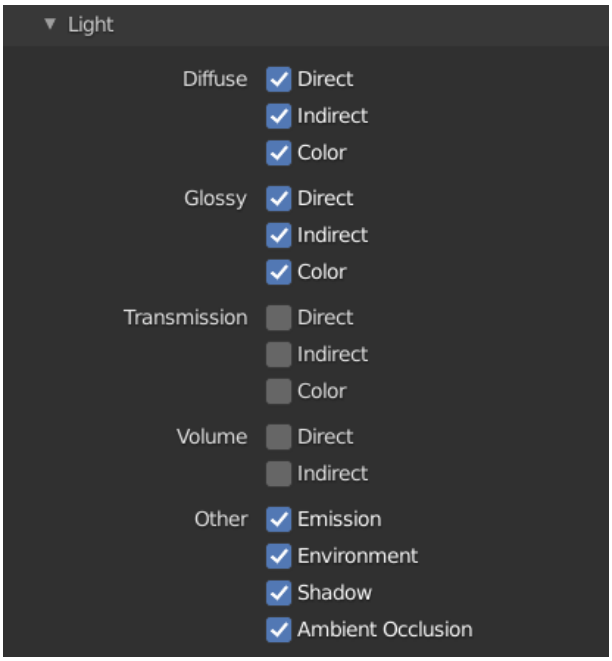

Figura 1.3: Render passes disponibili in Blender

I vari layer, infine, vengono combinati con il materiale girato mediante l'utilizzo del green screen; è infatti compito di chi si occupa della post-produzione di eliminare lo sfondo verde mediante l'utilizzo di filtri di colore e generare un layer con solo il soggetto.

La post-produzione, inoltre, si occupa di aggiungere eventuali effetti speciali in 2D (ad esempio esplosioni o particelle): il motivo per cui si fa questo è che è decisamente più semplice e economico rendere in 2D alcune tipologie di effetti. Inoltre, in quest'ultima fase si effettua il montaggio del materiale girato, il sound design ed eventuale doppiaggio.

Il lavoro svolto in questa fase è particolarmente importante nella realizzazione degli opening, dato che è in questo momento che si effettuano alcune scelte decisive che donano stile e coerenza al prodotto finito, stabilite durante la fase di look-development. Gran parte del look di un prodotto di questo tipo viene reso durante la fase di color grading. Il primo step consiste nella *color correction*: gran parte del girato, così come i render prodotti, vengono esportati in un profilo immagine logaritmico, che permette di preservare al meglio la gamma dinamica, ottenendo quindi più livelli di esposizione per le parti più scure delle immagini. In fase di post-produzione questo tipo di materiale deve essere ripristinato in termini di contrasto, saturazione, esposizione, bilanciamento del bianco, livello del bianco e del nero; il risultato sarà quindi un'immagine corretta in termini di luce e colore, il più fedele possibile alla realtà. La parte più creativa avviene quindi con il *color grading*, in cui, agendo con determinati strumenti come le curve tonali, si può modificare il look complessivo, modificando il quantitativo di un determinato colore (rosso, verde o blu) su alcuni range dinamici rappresentanti le ombre, le luci e le frequenze intermedie.

Capitolo 2

Look development

2.1 Definizione

Il look development è una parte essenziale, in particolare nella fase di preproduzione e postproduzione, di un progetto di animazione 3D, o in generale di un qualsiasi prodotto audio visivo che contempla la presenza di elementi 3D. Consiste essenzialmente nel definire lo stile complessivo delle scene (realistico, iperrealistico, stilizzato etc.), compiendo delle scelte tecniche e artistiche per avvicinarsi il più possibile al risultato desiderato. Comprende al suo interno diverse discipline tra cui, in una prima fase, il lighting, shading e rendering, ma anche compositing e post processing in una fase finale. Nel contesto di un progetto di medie/grandi dimensioni, la figura del look dev artist acquista particolare importanza in quanto, all' aumentare del numero di assets presenti nelle varie scene, diventa non banale riuscire ad ottenere un risultato consistente dal punto di vista stilistico: ogni oggetto 3D deve avere un look definito all'interno del contesto e, nei casi meglio riusciti e più iconici, riconoscibili anche fuori da questo. Diversamente da altre discipline, il ruolo del look development è difficilmente inquadrabile in un momento ben preciso della pipeline di produzione in quanto, dato il suo carattere multi disciplinare, è una figura presente dalla realizzazione dello storyboard fino al color grading finale.

È possibile fare innumerevoli esempi di look development in quanto è un aspetto presente in ogni produzione. Uno tra questi può essere trovato nel lavoro di caratterizzazione delle ambientazioni in *Soul*, tra gli ultimi film della casa di produzione statunitense Pixar Animation Studios. In questo caso, la ricerca stilistica ha portato alla definizione di due stili artistici, entrambi stilizzati ma differenti e ben riconoscibili, per caratterizzare e differenziare il mondo terreno con il mondo ultraterreno. Nel caso del mondo terreno, gli elementi stilizzati si integrano perfettamente in un ambiente generalmente realistico e molto dettagliato, nel secondo caso, invece, è possibile notare la presenza di elementi più semplici e uno shading decisamente più iconico, con scelta di colori freddi e una particolare integrazione tra elementi 2D (o simil 2D) e 3D (figura 2.1 e 2.2).

Per quanto riguarda il progetto descritto in questo lavoro di tesi, *Snow White - La Leggenda*, la fase di look development si è particolarmente incentrata sulla de-

Figura 2.1: Soul (2020) - Mondo terreno

Figura 2.2: Soul (2020) - Mondo ultraterreno

finizione degli obbiettivi stilistici e quindi in una serie di scelte relative a tecniche di modellazione e shading per avvicinarsi allo stile prefissato. È opportuno sottolineare che i render prodotti e mostrati in questo testo non sono necessariamente quelli definitivi, in quanto realizzati in una fase preliminare e soggetti a successive modifiche e rimaneggiamenti durante la fase di post-produzione.

2.2 Look development per *Snow White - La Leggenda*

Il progetto in questione ha previsto la produzione di scene e assets il cui fine era la realizzazione di un opening e materiale pubblicitario per uno spettacolo teatrale basato sull'illustre favola di Biancaneve. La produzione di un corto animato che, come in questo caso, comprende anche la presenza di girato e shooting con attori, porta con sé una serie di scelte artistiche e tecniche che ne compromettono inevitabilmente la resa finale. In un'ottica di rifarsi esteticamente al filone dei moderni trailer di serie TV e al tempo stesso non allontanandosi eccessivamente dalla tradizione iconografica della favola in questione, il lavoro si è concentrato sulla produzione di assets e scene nella loro interezza che avessero come principale caratteristica il fotorealismo. Al tempo stesso, era necessario riuscire ad ottenere questo risultato in tempi

non eccessivamente lunghi e con la possibilità di effettuare modifiche in itinere non distruttive e non proibitive. Postoci questo obbiettivo, il passo successivo è stato quello di fare una serie di scelte, principalmente in fase di preproduzione, riguardanti aspetti relativi alla modellazione, texturing, lighting e rendering. Per quanto riguarda il rendering, infatti, la buona resa di un prodotto finalizzato al fotorealismo è solitamente associata alla scelta di un motore di rendering fisicamente accurato (unbiased, come può essere Cycles in Blender), ma che comporta tempi di calcolo necessariamente lunghi. Questo, tuttavia, si scontrava con il secondo obbiettivo prefissatoci, ossia la flessibilità di produzione. In Blender, l'alternativa a Cycles è EEVEE, recente motore di rendering di tipo biased, che in primis permette all'artista di ricevere un riscontro immediato sulla riuscita della scena e, pur offrendo una resa grafica non eccessivamente accurata, può, con alcuni compromessi e seguendo alcune indicazioni, essere comparabile nei risultati ad un motore unbiased.

2.3 Fotorealismo: come ottenerlo

L'occhio umano riesce sempre con meno facilità a distinguere un'immagine generata al computer da una fotografia. Capita spesso di chiedersi se una determinata scena di un film sia stata realizzata in maniera virtuale o meno, e soprattutto la buona riuscita di una scena dovrebbe riuscire a non far neanche porre questa domanda allo spettatore. Tuttavia, la realizzazione di questa tipologie di scene non è semplice o immediata: richiede pazienza nella realizzazione e attenzione ai dettagli, in quanto il mondo che ci circonda è molto più caotico di quanto si possa pensare ed è particolarmente difficile riuscire a rappresentare il caos all'interno di un software di modellazione.

Nel tempo sono state definite una serie di linee guida che dovrebbero aiutare l'artista nella realizzazione di questa tipologia di scene. Bill Fleming, nel suo libro "Advanced 3D photorealism techniques" [6], descrive dieci principi di base da seguire:

- Caos: L'ordine all'interno della natura è possibile vederlo solo sulle grandi scale, ma più ci si avvicina al piccolo si può notare un caos sempre crescente. Riuscire a realizzare una scena caotica, ma al tempo stesso credibile e naturale, è la base del fotorealismo. Si possono utilizzare le migliori textures a disposizione, ma se applicate a oggetti posti in ordine, non saranno comunque credibili;

- Personalità: anche in una scena senza personaggi umani o animali, dovrebbe essere possibile riuscire a comprendere chi di solito è presente in quella scena, in quanto inevitabilmente ne rilascerà la propria impronta, modificando il territorio rappresentato. Un esempio potrebbe essere dato dalle specie di piante della foresta pluviale tropicale, che hanno una forma necessariamente "scolpita" nelle ere da una pioggia incessante. In questo caso è proprio la pioggia che esprime la propria personalità;

- Aspettativa: nella realizzazione della scena bisognerebbe essere guidati dall'aspettativa di chi la guarda, quindi da una serie di stereotipi che un osservatore si aspetterebbe di trovare in una determinata scena. Questi non devono limitare la creatività, ma indirizzarla verso scelte consapevoli e ragionate;

- Credibilità: Gli oggetti rappresentati devono essere credibili e familiari all'osservatore. È inoltre necessario aggiungere degli oggetti che si potrebbero definire "ancora", ossia per i quali è stato speso del tempo aggiuntivo per renderli realistici; in questo modo, per un osservatore, sarà più facile considerare realistica una scena e passare oltre su oggetti resi meno bene;

- Texture: in questo caso, il termine indica il vero e proprio materiale di ogni oggetto, necessariamente irregolare da un punto di vista di piccole imperfezioni. Anche un oggetto perfettamente liscio nella realtà non sarà mai tale; è quindi necessario rendere una rugosità opportuna per ogni materiale;

- Specularità: Nessun materiale è perfettamente non riflessivo. È necessario dosare quindi il livello di specularità per ogni oggetti, animato e non;

- Polvere e ruggine: ogni materiale è soggetto a invecchiamento e non sarà mai perfettamente pulito. È opportuno aggiungere elementi che indichino questi aspetti, in particolare in posti dove solitamente si accumula (ad esempio nelle cavità) e soprattutto miscelare le varie textures utilizzate al fine di creare una continuità;

- Imperfezioni: In particolare per oggetti naturali è impossibile non notare imperfezioni o rotture. Una roccia sarà naturalmente ricca di pori, increspature e tagli. Anche in questo caso, il quantitativo di imperfezioni deve essere dedotto dall'ambiente circostante e quindi deve essere coerente ad esso;

- Bordi: Qualsiasi oggetto non è mai perfettamente tagliente, ma i suoi bordi saranno sempre in una certa quantità levigati dal tempo. È importante quindi utilizzare tecniche che permettano di rendere questa tipologia di bordi, come in Blender il *Bevel Modifier* o il *Subdivision Surface* con opportuni loop di controllo;

- Spessore degli oggetti: Qualsiasi oggetto ha uno spessore, anche una foglia o un foglio di carta. Al fine di rendere realistici questo tipo di oggetti sarebbe opportuno applicare un piccolo spessore. Non è tuttavia sempre possibile in quanto, se è funzionale alla riuscita di una scena una grande quantità di oggetti di questo tipo, l'aggiunta di uno spessore quanto meno triplica il numero delle facce, quindi è da considerare caso per caso quando è possibile farlo;

- Radiosity: con questo termine l'autore del libro indica la caratteristica della luce di essere riflessa più volte, rendendo praticamente impossibile la presenza di ombre estremamente nette. È tuttavia possibile aggiungere che, in generale, è necessario lavorare particolarmente con l'illuminazione, con l'aggiunta ad esempio di HDRI, che permettono di donare alla scena una luce naturale.

È da sottolineare che il testo citato è datato 1999, ma i principi descritti sono perfettamente applicabili (e vengono applicati) alla computer grafica odierna, che sicuramente offre strumenti aggiuntivi e possibilità superiori rispetto al tempo in cui il libro è stato scritto.

2.3.1 Fotogrammetria e alternative

Nell'ottica di ottenere modelli fotorealistici, è opportuno citare una tecnica sempre più utilizzata, la fotogrammetria. Ha molteplici campi di utilizzo quali quello medico per la rappresentazione 3D accurata di organi, architettonico, per la ricostruzione 3D di reperti archeologici così come nella ricostruzione di mappe geologiche a partire da immagini satellitari. Non per ultimo, è sempre più frequente l'utilizzo nell'ambito dell'industria cinematografica e dei videogiochi: un esempio si può ritrovare nel supporto alla gestione di mesh molto dense in numero di vertici nel nuovo sistema Nanite di Unreal Engine 5, provenienti spesso da una ricostruzione fotogrammetrica: questa tecnologia è una estensione del concetto di livello di dettaglio in relazione alla distanza di un oggetto dalla camera. Durante la fase di importazione di una mesh, questa viene scomposta e ricombinata in un sistema gerarchico di gruppi di triangoli; durante il rendering, questi gruppi di triangoli vengono scambiati nelle loro versioni più dettagliate in base a cosa si sta inquadrando, ottimizzandone la visualizzazione.

Il funzionamento di base della fotogrammetria consiste nel ricostruire un modello 3D da informazioni bidimensionali, quindi da una serie di fotografie di un oggetto reale, derivandone quindi le informazioni sulla posizione, forma e dimensione [15]. Il risultato è quindi un modello estremamente fedele all'oggetto originale. È una tecnica che pone le sue basi teoriche sin dalla meta del '800, ma sta avendo un utilizzo sempre più frequente a partire dalla fine del secolo scorso, complici le nuove possibilità in relazione alle risorse computazionali, così come nuove tecnologie che possono aiutare in alcuni aspetti della ricostruzione 3D, come il Deep Learning. La fotogrammetria è differenziabile in base all'ambito di utilizzo e al tipo di oggetto che si andrà a replicare, in relazione soprattutto alle dimensioni: si parla infatti di *microfotogrammetria* per oggetti fino a 6 cm e ha utilizzo principalmente in laboratori, fotogrammetria *close range* per oggetti fino a un massimo di 100 m e fotogrammetria terrestre per oggetti decisamente più grandi, quindi fino a chilometri di dimensione, per i quali si utilizzano fotografie aeree, satellitari o, sempre più spesso, droni. In particolare, nell'ambito cinematografico, la più utilizzata è la fotogrammetria close range: in questo caso le fotografie sono scattate attorno all'oggetto in esame, con una sovrapposizione di ciò che viene mostrato tra le varie immagini di circa il 70-80%, quindi con un elevato livello di convergenza.

La base teorica della fotogrammetria si può sintetizzare nella definizione delle equazioni di collinearità. In figura 2.3 è schematizzato il sistema di proiezione centrale che, dato un punto nello spazio $A = (X_a, Y_a, Z_a)$, lo proietta su un piano rispetto a un centro di proiezione $O = (X_o, Y_o, Z_o)$ di proiezione distante c dal piano stesso, ottenendo un punto sul piano dalle coordinate $A' = (X'_a, Y'_a)$. La distanza tra

il centro di presa (o centro prospettico) e il piano immagine coincide con la lunghezza focale, se non si tengono in conto delle distorsioni prospettiche. Le coordinate X_a e Y_a sono ottenibili utilizzando la matrice di rotazione, nel quale viene utilizzando l'angolo il punto A' e il centro di proiezione. I fattori r_{ij} sono gli elementi della matrice di rotazione R:

$$X'_a = -c\frac{r_{11}(X_a - X_0) + r_{12}(Y_a - Y_0) + r_{13}(Z_a - Z_0)}{r_{31}(X_a - X_0) + r_{32}(Y_a - Y_0) + r_{33}(Z_a - Z_0)}$$

$$Y'_a = -c\frac{r_{21}(X_a - X_0) + r_{22}(Y_a - Y_0) + r_{23}(Z_a - Z_0)}{r_{31}(X_a - X_0) + r_{32}(Y_a - Y_0) + r_{33}(Z_a - Z_0)}$$

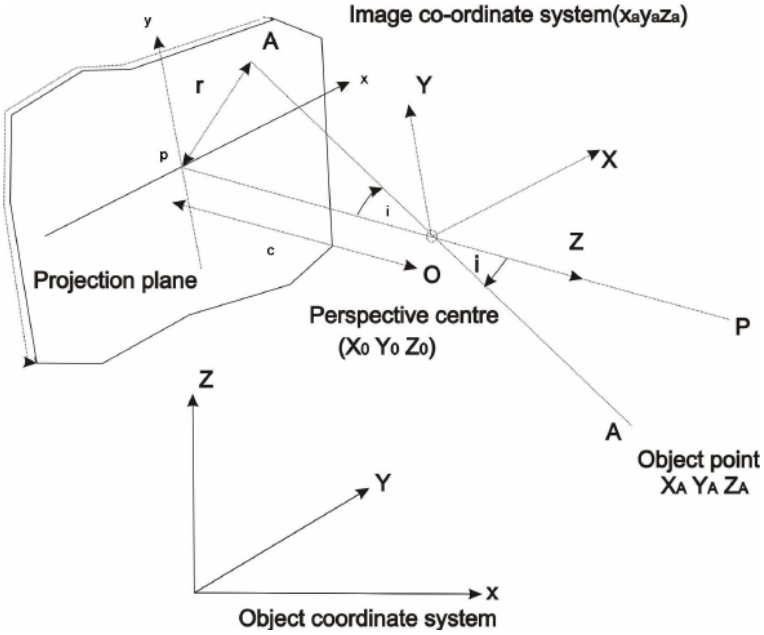

Figura 2.3: Sistema di proiezione centrale. Credits: Alyilmaz, C. & Yakar, Murat & Yilmaz, H.. (2010). Drawing of petroglyphs in Mongolia by close range photogrammetry. Scientific research and essays. 5.

Utilizzando fotocamere non-metriche (quindi di largo consumo), i parametri da tenere in considerazione durante l'acquisizione dei dati sono più delle semplici coordinate 3D delle posizioni delle camere, ma bisogna tenere in conto degli errori intrinsechi generati, quali la distorsione delle lenti e la lunghezza focale. Data tuttavia una singola immagine, non è possibile ottenere informazioni riguardanti le coordinate 3D del punto di un oggetto, ma sono necessarie un numero di immagini almeno pari a due, in modo da ottenere una visione stereoscopica. Note infatti le informazioni sui centri di presa e le orientazioni delle immagini al momento dello scatto è possibile ricostruire la posizione del punto dell'oggetto tramite la definizione dei punti omologhi: il punto sull'oggetto sarà dato dall'intersezione delle due rette che si incontrano nel punto omologo e passanti per i rispettivi centri di presa. Quelli

appena descritti sono i principi sui quali si basa la fotogrammetria aerea, ossia quella più tradizionale, ma possono essere estesi ad altre tipologie.

Nel campo della computer vision ha sempre più importanza una tecnica denominata Structure From Motion (SfM), che ha come scopo la ricostruzione di modelli 3D a partire da immagini 2D con l'aiuto di vettori di movimento, ossia quei vettori che descrivono la transizione tra un'immagine e la successiva. Il principio alla base di questa tecnologia è definito come *motion parallax*, quel fenomeno che indica di quanto un oggetto si "muove" in relazione alla distanza e al movimento dell'osservatore (oggetti più lontani si muovono più lentamente se rapportati ad oggetti più vicini). Tra un'immagine e la successiva bisogna quindi ricercare degli elementi comuni, come ad esempio i contorni di un oggetto. Per fare questo vengono usate tecniche di *object detection*, basate spesso sulla *scale-invariant feature tranform*, che sfrutta la differenza delle Gaussiane, una metodologia che consiste nel sottrarre a un'immagine sfocata ottenuta dall'originale una seconda immagine meno sfocata, ottenendo una terza immagine che avrà caratteristiche spaziali date dal range di frequenze spaziali preservate tra le due immagini (funziona quindi come un filtro passa-banda). Delle features estratte ne viene ricercata una corrispondenza tra le varie immagini: non tutti i match sono corretti, è necessario effettuare un filtraggio tramite algoritmi capaci di ricercare gli outliers. Tramite il tracciamento delle features è possibile ricostruirne le traiettorie descritte e quindi le coordinate 3D e la posizione relativa della camera.

Un'alternativa alla fotogrammetria è possibile ritrovarla nel sistema LIDAR (Laser Imaging Detection and Ranging) scanning, una tecnologia recentemente introdotta anche in alcuni smartphone. Questa sfrutta il principio tipico dei radar nel rilevamento delle distanze, utilizzando però in questo le radiazioni luminose nella lunghezza d'onda degli ultravioletti (tra i 100 e i 400 nm), calcolando il tempo trascorso tra l'emissione dell'impulso e la ricezione del segnale riflesso.

2.4 Scelta dei softwares

Nel mercato della CGI esistono innumerevoli softwares che permettono di affrontare le varie fasi di produzione. Ognuno di essi ha i suoi pro e i suoi contro e solitamente la principale area di competenza in cui eccelle. In produzioni di alto livello, i cui componenti del team tendono a specializzarsi in un determinato ambito (solitamente 3D modelers, texture artists, lookdev artists, technical director, animatori etc.), è comune vedere la compresenza di molteplici softwares che siano "production ready", a lungo testati e capaci di sostenere progetti di grande complessità. Ad esempio, Maya 3D viene considerato lo standard per la fase di modellazione, UV editing e animazione, ZBrush per lo sculpting digitale, Substance Painter per la fase di texturing, Marvelous Designer per il cloth simulation e Houdini per la simulazione di fluidi. Tuttavia, la maggior parte di questi softwares ha costi proibitivi per le produzioni di piccole dimensioni e necessita di tecnici specializzati che conoscano perfettamente l'ambiente di sviluppo di ognuno di essi.

Per la produzione dell'opening di *Snow White - La Leggenda*, data la natura del team di piccole dimensioni ma al tempo stesso essendoci la necessità di ottenere assets e scene anche complesse, con anche la presenza di simulazione di fluidi e hair grooming, la scelta, per la maggior parte del lavoro, è ricaduta su Blender, nella versione 2.93 LTS.
Blender è un software per la grafica 3D open source e gratuito che permette di avere in un unico ambiente tutti gli strumenti necessari alla pipeline di produzione. È quindi possibile modellare, applicare texture e shading, effettuare rigging, animazioni, simulazione di fluidi, rendering, compositing, motion tracking e video editing, il tutto senza mai abbandonare l'interfaccia. La scelta di utilizzare quasi esclusivamente Blender 2.93, oltre per la sua versatilità, è stata dettata da molteplici aspetti, fondamentali per gli obbiettivi prefissatici:

- nell'ambito del look development, Blender offre un potente sistema a nodi che permette di ottenere con estrema facilità shader complessi a carattere fortemente procedurale;

- la presenza di due potenti motori di rendering, Cycles e EEVEE (Extremely Easy Virtual Environment Engine). Come verrà meglio definito in seguito, è stato scelto perlopiù quest'ultimo per il rendering, in particolare per la versatilità che offre e i tempi di rendering decisamente ridotti rispetto a Cycles;

- L'utilizzo di Mantaflow, framework open source specializzato nella simulazione dei fluidi;

- Supporto ad UDIM, evoluzione del sistema di UV mapping e texturing, che permette di suddividere più semplicemente le texture relative a un modello, dando la possibilità di usare diverse texture maps su di uno stesso modello ma a risoluzioni differenti, in base al livello di dettaglio richiesto;

- La possibilità di utilizzare Python come linguaggio di scripting per automatizzare processi e creare addons.

Sebbene Blender permetta di gestire nativamente le textures, in alcuni casi descritti nel seguito di questo scritto, è stato deciso di utilizzare un altro software per questa fase, Substance Painter. Quest'ultimo è un potente software di painting 3D basato su un sistema a livelli (simile a quello utilizzato da Photoshop) che, in seguito all'importazione di modelli 3D, permette di "dipingere" direttamente su di questi. Particolarmente interessante è la presenza di smart materials, materiali "prefabbricati" applicabili direttamente ai modelli 3D, che tengono in conto delle caratteristiche del modello stesso, quali il calcolo delle normali, la curvatura e lo spessore delle superfici, l'ambient occlusion e la posizione della mesh. Per ottenere un risultato ottimale, è però necessario effettuare il baking delle textures, che consiste nel trasferire le informazioni presenti in una mesh ad alto contenuto di dettaglio (high-poly) all'interno di textures, con conseguente alleggerimento del flusso di lavoro. È inoltre presente un motore di rendering interno che offre la possibilità di

osservare in tempo reale il risultato dei materiali applicati. In seguito, è possibile esportare tutte le mappe necessarie al workflow PBR e importarle in Blender.

2.5 Caratteristiche dei diversi motori di rendering

Nel mondo della grafica 3D, esistono due principali approcci al rendering: unbiased e biased. Il primo ha come obbiettivo principale quello di essere fisicamente corretto nel calcolo dell'illuminazione, portando a un risultato spesso fotorealistico. È questo il caso di Cycles in Blender, che utilizza perlopiù algoritmi di rendering fisicalmente corretti e ponendo delle approssimazioni solo in casi limitati. Al contrario, EEVEE è un motore di rendering definito biased, compie ossia una serie di semplificazioni nella fase di shading, non calcolando come realmente la luce si comporta ma effettuando delle approssimazioni. In realtà questo tipo di distinzione non è estremamente netta, in quanto motori unbiased potrebbero tranquillamente fare uso di algoritmi approssimati tipici di quelli biased. All'interno di queste due macro categorie è tuttavia possibile approcciarsi al rendering utilizzando tantissimi diversi algoritmi. È possibile fare un'ulteriore distinzione infatti in motori di rendering che sfruttano la grafica *raster*, motori basati sul *ray casting* ed estensioni quali *ray tracing* e *path tracing*.

Motori di tipo raster

I motori di tipo raster hanno come obbiettivo principale la rappresentazione degli oggetti 3D di una scena su di una griglia di pixel 2D, effettuandone la proiezione sul piano di vista. È il primo metodo di rendering 3D ideato, ed è ancora estremamente utilizzato nell'ambito della grafica real-time, in particolare per quanto riguarda i videogiochi: rispetto all'approccio basato sul ray-tracing, il rendering raster permette di rappresentare gli oggetti 3D con estrema efficienza, riuscendo tranquillamente a raggiungere in tempo reale i 24 frame al secondo (e molti di più), ossia la più bassa frequenza necessaria a fare in modo che l'occhio umano percepisca l'illusione del movimento. Il compito principale di questo tipo di rendering è semplicemente quello di calcolare la proiezione dei modelli sulla superficie di vista e valutarne le occlusioni degli uni con gli altri. Viene eseguita una specifica pipeline che consiste in prima battuta nell'effettuare una trasformazione dei poligoni che compongono gli oggetti in triangoli; successivamente viene proiettato ogni vertice presente nella scena sulla superficie di vista, tenendo in conto del centro di proiezione. In questo senso, la rasterizzazione è un tipo di rendering *object ordered*, ossia parte dall'oggetto stesso per arrivare all'immagine finale. Differentemente dai motori di rendering basati sul ray tracing, in questo caso il colore del singolo pixel non è determinato dal modello di illuminazione applicato a uno specifico punto della scena, ma deve essere interpolato a partire dal colore definito sui singoli vertici, sui quali viene applicato un modello di illuminazione locale. Inoltre, emergono anche questioni relative al *clipping*, ossia cosa deve essere rappresentato sul piano di vista, e in che ordine. Nel caso del clipping,

una volta effettuata la proiezione sul piano di vista, bisogna specificare a priori cosa deve essere mostrato, ossia quali poligoni ne rientrano. Nel caso dell'ordinamento delle superfici visibili, che determina le occlusioni, si utilizza un algoritmo chiamato *z-buffer*, che consiste nel memorizzare, per ogni pixel che appartiene a un oggetto, la distanza dal centro di proiezione. In questo modo è possibile mostrare il pixel corrispondente a un determinato oggetto semplicemente confrontando i valori dello z-buffer per ogni oggetto. Il problema principale dei motori di tipo raster è che per loro natura sono limitati nell'effettuare una serie di operazioni necessarie a rendere una scena fotorealistica, in quanto il comportamento della luce che viene utilizzato non è fisicamente corretto. Esistono però una serie di soluzioni che permettono di generare un'immagine in qualche modo simile a quella prodotta da un motore di rendering *unbiased*.

Riflessioni nei motori di tipo raster

Differentemente da quanto accade per gli algoritmi di illuminazione globali, come può essere il ray tracing, gli algoritmi di illuminazione locali, che vengono applicati sui singoli vertici, non permettono il calcolo delle riflessioni. Una possibile soluzione è l'utilizzo di una *environment map*: dato un oggetto sufficientemente riflessivo, si posiziona un cubo in modo tale che questo lo racchiuda completamente. Per ogni faccia del cubo, si posiziona una telecamera virtuale in modo tale che il piano di vista di ognuna coincida con la faccia. Si ottengono così delle immagini che vanno a costituire una texture map, che verrà applicata al modello. Quella appena descritta è una soluzione che viene definita *mappatura cubica*, ma esiste una seconda soluzione che prende il nome di *mappatura sferica* o *reflection probe*, che al posto di un cubo utilizza una sfera, generando un'immagine deformata della scena. In alcuni casi l'utilizzo di questa tecnica genera artefatti facilmente riconoscibili; ad esempio non è possibile fare in modo che un oggetto concavo generi delle riflessioni con sé stesso.

Un altro algoritmo particolarmente utilizzato nell'ambito delle riflessioni è chiamato Screen Space Reflection (SSR). Viene sempre più utilizzato nell'ambito dei videogiochi in quanto supera alcuni limiti della cube map, come le riflessioni dell'oggetto con sé stesso, ed è possibile utilizzarlo in EEVEE. È un metodo decisamente piò oneroso in termini computazionali in quanto richiede alcuni moduli specifici nella pipeline di rendering, utilizzando diverse mappe come la depth map, position map e normal map, in combinazione tra loro. Queste mappe sono solitamente incluse nell'output del *Geometry Buffer*. L'idea di base consiste nel riutilizzare alcune informazioni già presenti nell'immagine mostrata dalla camera. La soluzione è quindi *view dependent* e non è possibile ottenere riflessioni provenienti da oggetti posti al di fuori di ciò che viene mostrato nella camera. L'SSR si basa su una tecnica chiamata *ray marcing*. Questa consiste nell'individuare delle informazioni riguardanti la scena semplicemente contraendo o espandendo dei vettori rappresentanti dei raggi. Nel caso dell'SSR, per ogni pixel viene generato un raggio, che viene riflesso rispetto alla normale alla superficie. Questo vettore di riflessione viene quindi proiettato sullo schermo utilizzando una matrice di proiezione. Il ray marching quindi consiste nell'espandere la lunghezza del raggio riflesso, modificandone la coordinata z. Data

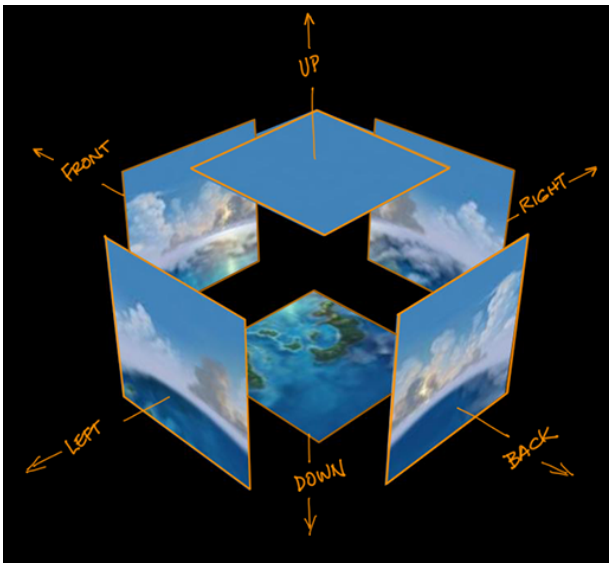

Figura 2.4: Cube mapping. Credits: Scali's OpenBlog

quindi la posizione del raggio, questa viene confrontata con le informazioni presenti nella mappa di profondità; se la coordinata z del raggio è maggiore del valore derivato dalla mappa di profondità in quel punto, allora cè stata una intersezione. Il punto di intersezione verrà quindi usato per effettuare lo shading nel punto di origine del raggio.

Figura 2.5: Screen Space Reflection. Credits: images.nvidia.com

Illuminazione indiretta

Un altro dei punti sul quale i motori di rendering basati sulla rasterizzazione peccano è riguardante l'illuminazione indiretta. Nei motori di rendering di tipo ray

tracing, in particolare in quelli di tipo distribuito, l'illuminazione diffusa è intrisecamente calcolata mediante le varie riflessioni dei raggi. Un modo per sopperire a questa mancanza consiste nel calcolare off-line (quindi effettuando un bake) la luce indiretta, che viene memorizzata in una *light map*. Solitamente viene utilizzata una soluzione *view-indipendent* basata su algoritmi che sfruttano la radiosity, che quindi è indipendente dalla posizione della camera, dando quindi la possibilità di spostare quest'ultima senza effettuare nuovamente il bake. In motori come EEVEE si va a definire un *Irradiance Volume*, ossia un cubo che solitamente racchiude l'intera scena. All'interno di questo, vengono definiti una serie di punti, il cui numero (e quindi la risoluzione) può essere modificata dall'utente. Per ognuno di questi punti viene quindi calcolata una light-map in scala di grigi che verrà moltiplicata con le informazioni derivanti dalle textures degli oggetti. Data la risoluzione decisamente inferiore alla scena nella sua interezza, le informazioni per ogni punto vengono interpolate le une con le altre, dando un risultato più omogeneo. È una tecnica estremamente utilizzata nel campo dei videogiochi. Un'importante limitazione riguarda il fatto che non è possibile applicarla su oggetti dinamici che si muovono all'interno della scena, ma solo su quelli statici, in quanto calcolata prima del rendering. È tuttavia possibile modificare questo comportamento in Blender mediante scripting, facendo in modo che il calcolo venga effettuato per ogni frame, ovviamente aumentando il tempo di calcolo.

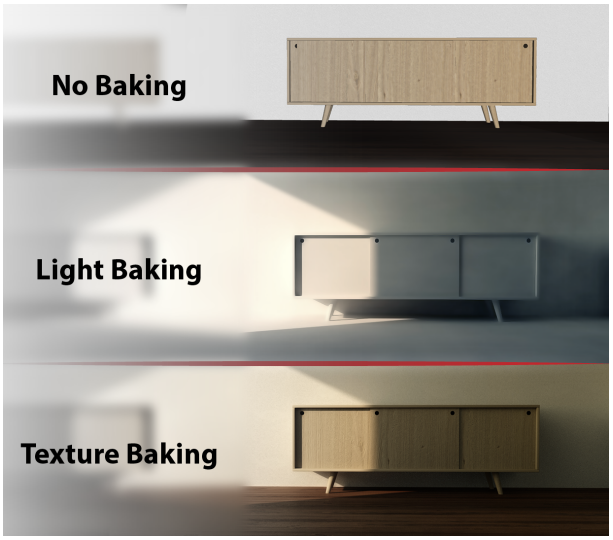

Figura 2.6: Light Baking. Credits: https://vrforcad.com/light-texture-baking

Generazione delle ombre

Anche nel caso della generazione delle ombre, sono necessarie alcune approssimazioni. Il modo in cui EEVEE si approccia a questo problema è mediante le *shadow maps*. Questa tecnica si basa su un algoritmo di rimozioni delle superfici nascoste e consiste nel calcolare uno z-buffer per ogni luce presente nella scena, tenendo quindi

presente della distanza dall'oggetto rispetto alla superficie di vista generata dalla luce. Nel caso di una luce di tipo *spot*, viene generato solo uno z-buffer, mentre nel caso di una luce di tipo *point* invece vengono generati sei differenti z-buffer, idealmente racchiudendo la luce in cubo e proiettando sei diverse camere, una per ogni faccia. Una volta memorizzate le informazioni della distanza dei punti dalla camera, ogni volta che si trova un punto z da renderizzare lo si rapporta alla distanza z_l del primo elemento della geometria trovato, renderizzandolo in ombra se la distanza è maggiore di z_l, il chè significa che è stato occluso da un altro oggetto. In caso contrario viene calcolato lo shading in maniera normale.

Ray casting, ray tracing e path tracing

Anche il ray casting è solitamente descritto come un modo per determinare quale oggetto deve essere rappresentato per ogni pixel, similmente al concetto di z-buffer. Il modo con cui lo fa è basato sul generare un raggio per ogni pixel, la cui origine è determinata dal centro di proiezione. Nella versione di base, il raggio generato termina sulla superficie di un oggetto o, in caso di nessuna intersezione, sullo sfondo. Nel punto di intersezione viene applicato un modello di illuminazione locale e, se parte di un oggetto non viene colpito da nessun raggio, viene renderizzato come in ombra. Il calcolo delle intersezioni ha un costo lineare nel numero di primitive, ma esistono diverse tecniche per ridurre il numero di calcoli necessari a coprire tutte le occlusioni. Un modo consiste nel suddividere lo spazio in volumi gerarchici che contengono gli oggetti in maniera ricorsiva. Questo permette di escludere a priori le intersezioni di un determinato con un oggetto nel caso di assenza di intersezione con il volume che lo racchiude.

L'estensione del ray casting è la sua versione ricorsiva, il *ray tracing*, che consiste nel generare ulteriori raggi in seguito alla prima intersezione, relativi a riflessione, rifrazione e ombra e generarne altri fino a raggiungere un predeterminato numero di ricorsione. La versione appena descritta, che è quella utilizzata per la maggior parte, è definita come *backward ray tracing*, in quanto il raggio generato proviene direttamente dal centro di proiezione e va verso la scena e, dopo una serie di riflessioni e rifrazioni, il raggio potrebbe raggiungere una sorgente luminosa. Al contrario, nel *forward ray tracing*, i raggi generati provengono direttamente dalla sorgente luminosa e, dopo una serie di rifrazioni e riflessioni, potrebbero raggiungere il piano di vista. È un metodo decisamente più oneroso in quanto richiederebbe di generare un numero di raggi decisamente elevato per avere un risultato soddisfacente. Un ulteriore versione è il *path tracing*, che è l'algoritmo utilizzato dal motore Cycles in Blender. L'obbiettivo del path tracing è quello di ridurre il numero di raggi generati, cercando di trovare un compromesso tra tempo di calcolo e risultato ottimale. La differenza rispetto al ray tracing consiste nel generare un numero casuale di raggi per ogni pixel e, ad ogni intersezione, viene generato un nuovo raggio dalla direzione randomica. Maggiore è il numero di raggi generati e migliore sarà il risultato finale, in quanto sarà presente meno rumore. In Blender esistono in ogni caso dei metodi di *denoising*, per cercare di migliorare un'immagine affetta da rumore. Un primo metodo è basato sul *non-linear mean* (NLM), che effettua una media non locale

dell'intera immagine, pesata i ogni punto in base al livello di similarità con il pixel che si sta prendendo in considerazione. Un metodo più efficiente e di recente uscita è la tecnologia di NVidia denominata *Optix Denoiser*, che utilizza tecniche di machine learning per migliorare le immagini generate da algoritmi che sfruttano il metodo di Monte Carlo, così come fa Cycles.

Questo tipo di motore di rendering è utilizzato principalmente *offline* a causa del tempo di calcolo elevato, quindi per il rendering di scene nell'ambito di film e animazione.

2.6 Scelta del motore di rendering

Come accennato, è stato scelto di utilizzare per la maggior parte delle scene EEVEE, in particolare per quelle per le quali era prevista un'animazione. All'interno delle produzioni di prodotti indipendenti si sta sempre più vedendo infatti il passaggio da tecniche di rendering *offline* verso altre in real-time. Uno dei maggior esempi in questo senso è dato dal motore grafico prodotto da Epic Games Unreal Engine, arrivato ora alla versione 5, che permette di ottenere una realizzazione grafica di altissimo livello in tempo reale. Il maggiore punto di forza in questo senso è dato dalla velocità che è possibile sperimentare nelle scelte artistiche, rendendo possibile la visione immediatamente in viewport di ciò che verrà poi mostrato in rendering, permettendo di lavorare a un ritmo decisamente superiore e non limitato dalla necessità di avere una strumentazione di altissimo livello per evitare di avere in viewport un rumore costante, tipico di Cycles. Oltre alle possibilità di visualizzazione migliori in viewport, la scelta di EEVEE porta a una riduzione importante in termini di tempo per quanto riguarda il tempo di render di una scena.

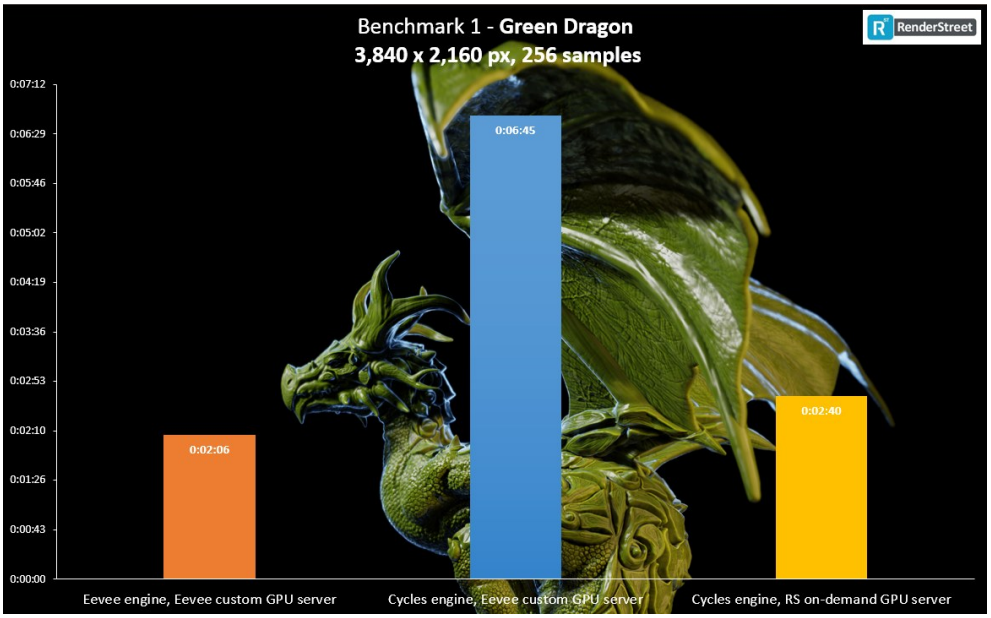

Figura 2.7: Benchmark per il rendering di una scena in EEVEE e in Cycles

La scelta di utilizzare EEVEE porta con sé ovviamente alcune limitazioni, che è stato necessario affrontare e riuscire in qualche modo a superare lavorando in modo specifico con gli strumenti dello shader editor nella realizzazione di materiali. Sebbene infatti Cycles e EEVEE condividano il Principled BSDF, un nodo basato sul PBR shader di RenderMan, che permette di avere in un unico nodo la maggior parte delle informazioni di base relative alla caratterizzazione di un materiale, esistono alcuni nodi specifici per Cycles che in EEVEE non sono presenti. Un esempio è dato dal Principled Hair BSDF, un nuovo nodo che permette in Cycles di ottenere facilmente un materiale specifico per i capelli, variandone alcuni parametri basati sulla fisica (ad esempio il contenuto di melanina all'interno del singolo capello). In EEVEE è quindi necessario realizzare uno shader specifico utilizzando gli altri nodi a disposizione, così come nella realizzazione di materiali di tipo riflessivo e rifrattivo che, come già descritto, sono particolarmente complessi da realizzare in maniera accurata nei motori di render real-time.

È stato tuttavia preferito Cycles per la realizzazione di alcune scene, per le quali era richiesto un singolo render ed erano quindi assenti movimenti di camera.

Capitolo 3

Preproduzione

Le fasi di produzione di un prodotto audiovisivo con elementi di grafica 3D non seguono un ordine rigido e tendono a adattarsi alla tipologia di risultato che si intende ottenere, tenendone in conto la durata, lo stile, la grandezza del team che si occuperà di svilupparlo e il budget. Tuttavia, è possibile in generale effettuare una suddivisione in tre fasi: preproduzione, produzione e postproduzione, come descritto nella sezione 1.3.

La preproduzione si occupa di definire il progetto nella sua interezza, senza entrare eccessivamente nei dettagli tecnici. All'interno di questa fase, il primo passo è il definirsi di un'idea, che è alla base della successiva definizione di script e storyboard. Nel caso del progetto in questione, essendo la tematica commissionata da un cliente esterno, la fase di ideazione si è principalmente focalizzata sulla scelta delle scene da rappresentare e sulla ricerca di un look visivo che si rifacesse ai canoni dell'iconografia fantasy e favolistica. Nel seguito vengono quindi descritte le principali references utilizzate nelle varie occasioni.

3.1 La favola di Biancaneve

Biancaneve, conosciuta anche come *Biancaneve e i Sette nani*, è una favola popolare europea la cui versione più celebre è stata pubblicata nella raccolta Fiabe dei bambini e del focolare, pubblicata nel 1812 dai fratelli Jacob e Wilhelm Grimm, linguisti e filologi tedeschi. La storia racconta di Biancaneve che, rimasta orfana di madre, vive con il padre e con la sua matrigna, una perfida regina a cui interessa solo il suo aspetto fisico e che ogni giorno domanda al suo magico Specchio delle Brame chi sia la più bella del reame. In seguito alla morte del padre, la Regina riduce in sguattera la figliastra. Biancaneve cresce e lo Specchio delle brame informa la Regina, invidiosa della sua bellezza, che è a lei che spetta il primato di più bella. Ordina quindi a un cacciatore di portare Biancaneve in un bosco, ucciderla e portarle il suo cuore in uno scrigno. Il cacciatore, impietositosi della ragazza, non compie l'omicidio e le suggerisce di scappare nel bosco, uccidendo quindi un animale e portandone il cuore alla regina. Biancaneve trova nel bosco una casa, dimora di sette nani che lavorano in una miniera, con i quali riesce a ricrearsi una vita. La Regina, in realtà

una strega, scopre che Biancaneve è ancora viva, raggiunge la casa nel bosco e le offre una mela avvelenata, con la quale la ragazza cade in un sonno profondo. A salvarla sarà quindi il celebre principe, che la risveglierà con un bacio.

La storia è stata resa particolarmente famosa con il film d'animazione statunitense del 1937, *Biancaneve e i sette nani*, prodotto dalla Disney. Primo lungometraggio della casa di produzione, è stato realizzato in seguito a lunghe ricerche stilistiche sotto la guida del concept artist Albert Hurter per quanto riguarda lo stile dei personaggi e delle ambientazioni e di Art Babbitt per le animazioni.

3.2 References

La ricerca di references è stata una fase di fondamentale importanza, al fine di delineare alcuni aspetti guida del progetto. In prima battuta, è stata effettuata una ricerca riguardante lo stile e la struttura degli opening più moderni di famose serie TV. Per quanto riguarda la struttura, gran parte di essi ha una durata tra i 60 e i 90 secondi e le varie scene che lo compongono hanno una durata media di 7/8 secondi. Le scene sono solitamente abbastanza statiche, con lenti movimenti di camera, spesso close-ups, su oggetti 3D o attori. Il lavoro di look development è particolarmente centrale: di fondamentale importanza è la scelta di uno stile predominante in cui è necessaria la coerenza tra le varie scene, in seguito a scelte di shading, illuminazione e post processing. In seguito, le principali references stilistiche tratte da openings, che hanno indirizzato nella scelta delle inquadrature e delle scene da realizzare:

- *Carnival Row*, serie televisiva statunitense del 2019 distribuita da Amazon Prime Video. La trama si basa su uno "scontro" tra creature magiche, immigrate e ghettizzate in una città e gli abitanti umani della stessa. è stata sicuramente la reference principale in quanto l'ambientazione steam punk vittoriana dai caratteri oscuri integrata con la componente favolistica è particolarmente evidente sin dai credits. La figura 2 ritraente un fauno, è stata utile per la scena del nano e sul livello di dettaglio che questo necessitava (3.1, 3.2).

Figura 3.1: Fata in Carnival Row (2019)

Figura 3.2: Fauno in Carnival Row (2019)

- *Daredevil* (2015-2018). In questo caso la reference è stata utile per la scena relativa alla simulazione del fluido del veleno che cade sulla mela, in particolare per la definizione della densità dello stesso (3.3):

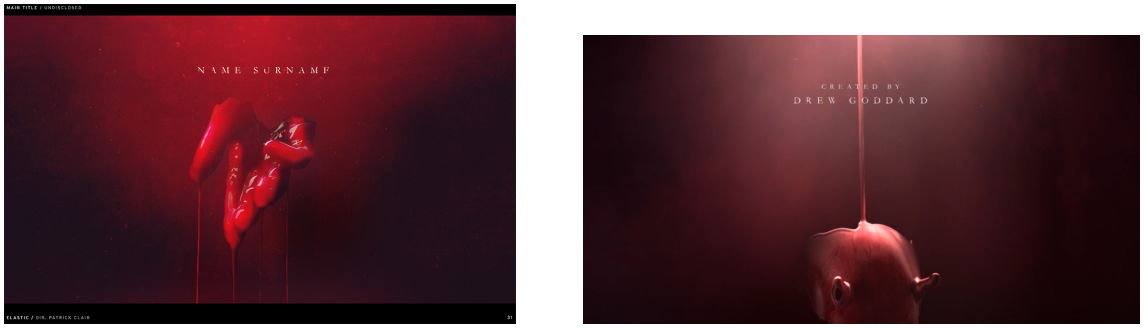

Figura 3.3: Daredevil (2015)

La scelta delle scene da realizzare è stata data dalla necessità di rappresentare le ambientazioni e i personaggi più iconici della storia di Biancaneve, con l'aggiunta di elementi che richiamassero il folklore europeo: lo scrigno contenente il cuore, la mela avvelenata, la pozione, la sala del trono della Regina, un nano, un folletto, il libro incantato. In questo senso era necessario confrontarsi con il celebre film d'animazione, dal quale alcune scene sono state ispirate, come ad esempio il veleno che si adagia sulla mela, avvelenandola (3.4).

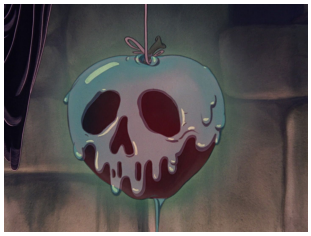

Figura 3.4: Mela avvelenata - Biancaneve e i Sette Nani (1937)

Tuttavia, l'idea era quella di allontanarsi dallo stile più "infantile" e cartoonesco rappresentato dal film della Disney, e conferire all'opening dei tratti più cupi e gotici. Una tra le references principali in questo senso è stata data dalla trilogia capolavoro di Peter Jackson, *Il Signore degli Anelli*. In particolare, la scena del bosco è stata influenzata dal look visivo della foresta di Fangorn (3.6), caratterizzata da toni scuri e tendenzialmente in controluce, mentre le scelte di modellazione per l'ampolla di veleno, così come la scena della sala del trono e in generale negli elementi decorativi di ogni scena si rifanno agli ornamenti di Rivendell (3.5), avamposto degli elfi di Arda. La rappresentazione del nano, inoltre, è stata realizzata in modo tale da renderlo più realistico e simile ai nani de *Il Signore degli Anelli*, senza però distaccarsi eccessivamente dall'iconografia dei nani del film d'animazione della Disney.

Figura 3.5: Rivendell - Il Signore degli Anelli (2001)

Figura 3.6: Foresta di Fangorn - Il Signore degli Anelli (2001)

Capitolo 4

Analisi delle scene

4.1 Bosco autunnale

4.1.1 Introduzione

La scena relativa al bosco in autunno è stata la prima realizzata in quanto non direttamente collegata alla realizzazione dell'opening, ma finalizzata alla produzione di materiale preliminare a fine pubblicitario. Per *Snow White - La Leggenda*, sono state infatti realizzate diverse locandine ritraenti alcune scene chiave della favola in questione, una tra queste il bosco nel quale Biancaneve trova la casa dei nani.

4.1.2 Versioni della scena

La realizzazione di questa scena ha seguito un percorso particolarmente sperimentale, con la produzione di diverse scene che sono state proposte al cliente. L'idea di base era la realizzazione di un bosco durante la stagione autunnale, con al centro della scena Biancaneve (4.2). Inizialmente era stata proposta una versione dai toni particolarmente cupi (4.1) la cui reference principale è stata la foresta di Fangorn (3.6).

Questa tuttavia è stata modificata al fine di ottenere una composizione differente: in particolare, è stato scelto di rappresentare un "tunnel" di alberi in primo piano e in lontananza una radura molto illuminata.

La necessità di muoversi verso questa seconda scelta, è stata data principalmente da motivazioni relative al lighting in relazione al materiale da compositare. In seguito infatti alla fase di shooting fotografico, è stata realizzata una foto ritraente Biancaneve che regge con una mano la mela avvelenata. La foto è stata realizzata con un setup di lighting chiamato *3-points-lighting*, che consiste in una backlight che illumina il soggetto alle spalle e due luci (key light, luce principale e fill ligth, che rende meno *dure* le ombre) poste di fronte al soggetto, rispettivamente a $-45°$ e $45°$ rispetto ad esso (4.3). Al fine quindi di rendere la scena coerente con il risultato dello scatto, è stata realizzata un'illuminazione particolarmente controluce (fig. 4.13)

Figura 4.1: Render della prima versione, realizzato con EEVEE

4.1.3 Requisiti della scena

La scena necessitava di essere il più fotorealistica possibile. In tal senso, non essendo destinata ad essere background di alcuna animazione, è stato possibile avere più libertà riguardo al motore di rendering, scegliendo quindi Cycles per la versione finale. Il più grande ostacolo tuttavia è consistito nella rappresentazione degli alberi e del fogliame: sebbene sia possibile infatti modellare alberi anche tramite l'utilizzo di addon che rendono più immediato e procedurale il flusso di lavoro, rimane difficile renderli realistici, in particolare per quelli più vicini alla camera. In tal senso è stato necessario utilizzare un approccio misto, che ha previsto l'utilizzo di modelli fotogrammetrici per quanto riguarda quelli più vicini alla camera (foreground e middleground) e modelli realizzati tramite tecniche più tradizionali per il background, organizzando quindi la scena per livelli di dettaglio decrescenti in relazione

Figura 4.2: Una delle reference per la scena del bosco

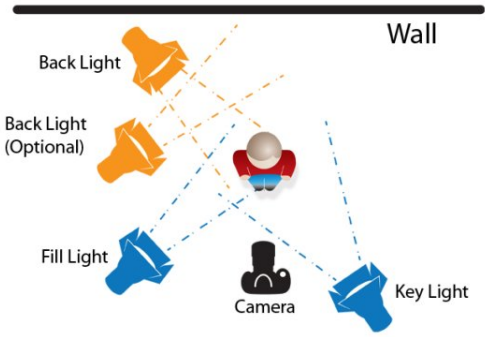

Figura 4.3: 3-Points-Lighting

alla distanza dalla camera.

4.1.4 Organizzazione del progetto

La realizzazione di una scena di questo tipo prevede uno studio preliminare riguardo l'organizzazione dei vari file .blend in modo da non appesantire eccessivamente il carico sulla CPU e GPU, specialmente in fase di allestimento della scena finale. I modelli 3D prodotti tramite fotogrammetria, sono infatti particolarmente densi dal punto di vista di numero di vertici e con facilità si raggiunge la saturazione della memoria della GPU (VRAM), all'interno della quale vengono memorizzate le informazioni relative alle coordinate 3D dei vertici delle mesh durante la fase di rendering. Inoltre, il numero dei modelli 3D da utilizzare è stato abbastanza elevato ed era necessario quindi organizzarli separatamente dalla scena finale. È stato quindi

creato un file .blend apposito che racchiudesse tutti gli assets da utilizzare. La collezione principale di questo file, chiamata *_Assets*, è stata quindi aggiunta tramite link alla scena da allestire, inserendola in una scena separata. All'occorrenza, è stato necessario modificare i vari modelli singolarmente; per fare ciò è possibile andare ad agire direttamente sul file Assets.blend o, in alternativa, usare il nuovo sistema di Library Overrides. Quest'ultimo è stato aggiunto in Blender a partire dalla versione 2.81 ed è destinato a rimpiazzare l'obsoleto sistema Proxy, permettendo di effettuare multiple modifiche dello stesso modello di cui è stato fatto il link, ad esempio aggiungendo o togliendo modificatori, così come andare direttamente ad agire sulle informazioni della mesh (edit mode). In questo modo, possono essere sovrascritti alcuni tipi di data block (l'*unità di base* di Blender nella gestione dei vari tipi di dato) e le proprietà di queste sovrascritture, modificate. Le proprietà che non vengono modificate dalla sovrascrittura vengono aggiornate quando i *library data* (ossia i dati relativi al link, che contengono informazioni sui file .blend esterni) cambiano.

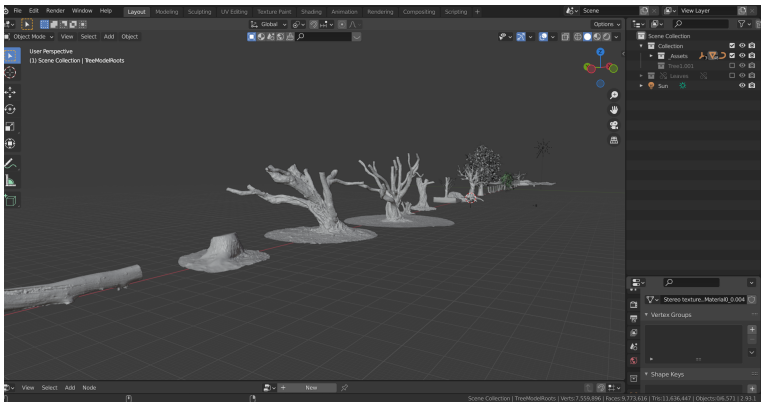

Figura 4.4: Snapshot dell'assets blender file

L'organizzazione della scena finale ha visto un ulteriore passaggio, relativo alla suddivisione per layer della stessa. Al fine di non appesantire ulteriormente la scena è stata fatta una suddivisione per layer, ossia foreground, middleground e background da compositare in seguito. Il vantaggio di una suddivisione di questo tipo è quello di effettuare render separati per i vari layer e quindi non ripetere l'operazione di rendering sull'intera scena in caso di modiche su un singolo layer. Per fare questo, Blender mette a disposizione i *View Layers*, che permettono di selezionare per ogni view layer le collections da utilizzare per la viewport e per il rendering. In alcuni casi è però necessario che collection oscurate interagiscano con la collection corrente per quanto riguarda il lighting (ad esempio il foreground, in questo caso , doveva essere influenzato dal middle ground, in quanto gli alberi presenti in quest'ultimo dovevano generare dell'ombra sugli oggetti in primo piano). Per fare ciò, è possibile definire come *holdout collections* le collections che devono con il view layer corrente, senza che ne sia fatto il render. In tal modo si andrà a creare una maschera nel render finale del layer in questione.

4.1.5 Modellazione parametrica di alberi

La realizzazione della scena ha previsto la presenza di numerosi modelli di alberi. Come già accennato nella sezione 4.1.3, alcuni di questi sono modelli fotogrammetrici, per ognuno dei quali è stata fatta una riduzione del numero dei vertici tramite modificatore *Decimate*, che effettua il merge progressivo dei vertici della mesh in input tenendo in conto del parametro *ratio*, o utilizzando il più recente modificatore *Remesh*, che crea una nuova topologia only quads seguendo la curvatura della superficie in input. Non era tuttavia possibile utilizzare esclusivamente modelli fotogrammetrici per diverse ragioni:

- Il numero dei vertici rimane abbastanza elevato, anche a seguito del remesh, al fine di preservare la coerenza con il modello originale;
- I modelli fotogrammetrici sono tendenzialmente incompleti. L'acquisizione dei dati della mesh, come descritto nella sezione 2.3.1, non permette di ottenere una rappresentazione completa dell'albero, completa e dettagliata in ogni aspetto, e di fatto sono *tagliati* in prossimità della diramazione dei rami dal tronco principale;
- Non sono facilmente modificabili e personalizzabili. I modelli provengono da fotografie reali di alberi, perciò non è possibile ottenere alberi di qualsiasi forma. In alcuni casi, tuttavia, sono stati utilizzati vari modificatori posti nello stack (come il *Curve Modifier* o il *Simple Deform*) al fine di modificare in parte la forma di alcuni dei modelli fotogrammetrici utilizzati.

È stato quindi necessario realizzare dei modelli di alberi *ad hoc*, che in alcuni casi avessero anche la caratteristica di poter essere modificati in itinere al fine di adattarsi al meglio alla scena.

L-System

La modellazione di piante e alberi costituisce un problema non banale nel campo della computer grafica, in quanto hanno solitamente una struttura abbastanza complessa, anche se in qualche modo standardizzabile. Un modo per modellare gli alberi in maniera semi-automatica è rappresentato dagli L-System, o sistema di Lindenmayer, ossia un sistema complesso in grado di descrivere in maniera formale la crescita e quindi la forma delle piante. Esistono vari tipi di L-System; il più semplice è deterministico e senza contesto, ossia ad un determinato input corrisponde sempre uno stesso output e non dipende da fattori esterni (ad esempio fattori atmosferici). Il modo in cui vengono rappresentati è mediante una sequenza di simboli (assioma), alla quale viene applicata una o più regole di produzione in parallelo. La stringa iniziale viene quindi modificata in modo ricorsivo fino ad ottenere una stringa finale, che rappresenterà una serie di caratteri ognuno con una regola di modellazione: su di un piano è possibile immaginare un cursore definito dalla terna (x, y, α) (posizione e orientamento) che si sposta sul piano disegnando (o non disegnando) dei segmenti. Il metodo appena descritto permette di descrivere una struttura lineare.

Una tipologia più accurata viene definita *Bracketed L-System*, che permette di descrivere delle ramificazioni in maniera ricorsiva tramite l'utilizzo dei simboli *[* e *]*. È possibile inoltre utilizzare una versione stocastica degli L-System, che associa ad ogni regola di produzione una possibilità di esecuzione, rendendo quindi casuale il risultato finale.

M-Tree e Sapling Addon

Nel caso in questione è stato scelto di utilizzare due addon, in particolare gli addons *MTree* e *Sapling*, quest'ultimo già incluso con Blender e solamente da abilitare.

L'MTree Addon (Modular Tree) permette di creare alberi tramite curve in maniera modulare, utilizzando un sistema a nodi simile a quello utilizzato dal sistema di shading interno a Blender. L'idea di base consiste nel dividere logicamente un albero in sezioni, partendo dal basso e aggiungendo man mano elementi: si parte dal tronco principale, dal quale partono le prime diramazioni con i rami principali, i quali si suddividono a loro volta in rami più piccoli e così via. Nel sistema a nodi, si aggiunge quindi un *trunk node*, la base del nostro albero. A questo è possibile aggiungere vari *branch nodes*, ognuno corrispondente a una ulteriore diramazione. Nelle ultime versioni è stato aggiunto anche un *root node*, che permette di aggiungere le radici. Troviamo quindi il *twig node*, con il quale è possibile generare dei rami più piccoli con foglie che possono venire istanziate tramite il nodo *Tree parameters*. Quest'ultimo offre una personalizzazione ad alto livello di ciò che viene mostrato sulla viewport, permettendo di aggiungere un sistema particellare di tipo hair all'albero, aggiungendo le foglie. Inoltre è possibile definire la risoluzione dell'albero finale e il tipo di visualizzazione, *preview* che lascia le varie parti dell'albero separate e le mostra a una risoluzione più bassa e *final*, con il quale si finalizza l'albero, connettendone le varie parti. Ognuno dei nodi descritti, permette di customizzare diversi aspetti dell'albero, quali la lunghezza e il raggio dei rami o del tronco, l'angolo che viene generato in seguito a una diramazione, la forma stessa dei rami tramite il parametro *random*, così come la rotazione verso il basso degli stessi.

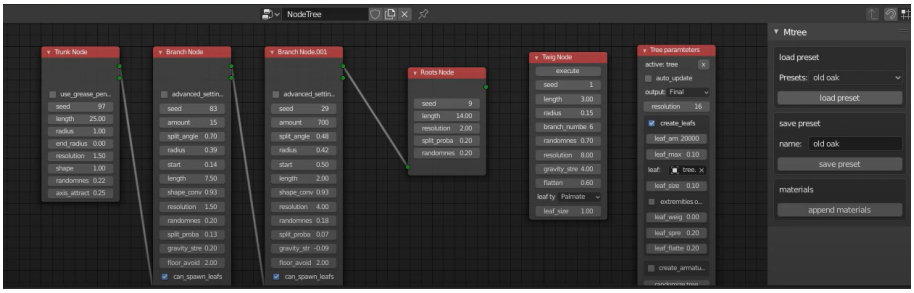

Figura 4.5: Sistema a nodi dell'addon MTree

Il secondo addon, *Sapling Tree*, permette la creazione dell'albero tramite interfaccia grafica e non con sistema a nodi. La generazione dell'albero segue otto macro settaggi (*Geometry*, *Branch Radius*, *Branch Splitting*, *Branch Growth*, *Pruning*, *Leaves*, *Armature*, *Animation*), molto simili ad MTree per quanto riguarda la tipologia

di parametri, producendo risultati praticamente sovrapponibili. È stato tuttavia preferito quasi principalmente il primo descritto per la possibilità più immediata di modifica del modello generato, la generazione di alberi simili al variare del parametro *seed*, così come la visualizzazione a più risoluzioni dello stesso.

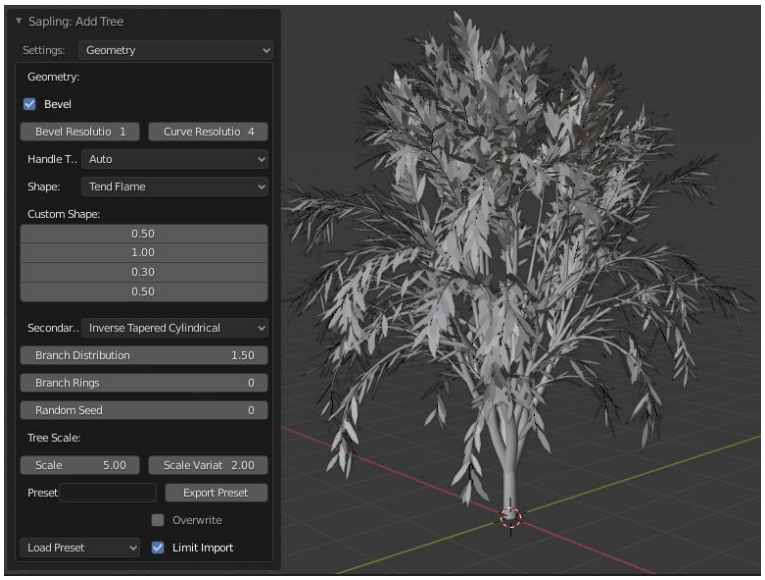

Figura 4.6: Interfaccia dell'addon Sapling Tree

In seguito alla fase di modellazione, è stato effettuato il texturing. La fase di UV unwrapping è stata automaticamente generata in seguito alla generazione dei modelli dai rispettivi addon. Sono state quindi aggiunte delle texture di tronchi di quercia per il materiale del legno, mentre per le foglie, i vari piani aggiunti tramite hair particles, sono stati texturizzati utilizzando immagini di foglie e variandone la tonalità con il nodo Hue/Saturation/Value in modo da ottenere colori autunnali.

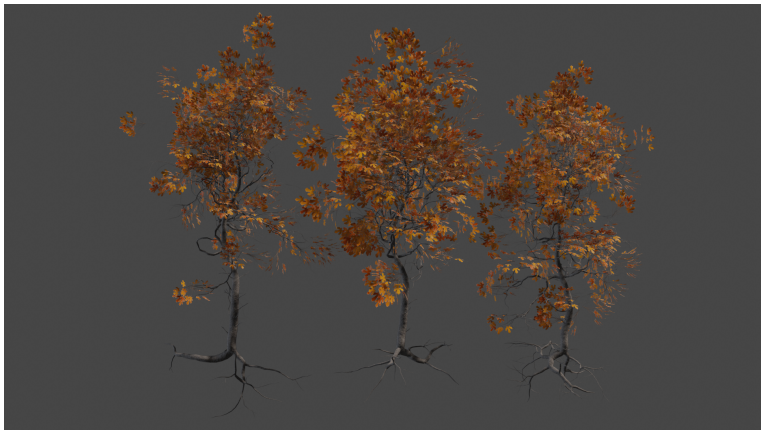

Figura 4.7: Render di alcuni degli alberi realizzati

4.1.6 Realizzazione delle radici

Oltre agli alberi realizzati in precedenza, era necessario arricchire la scena con radici che coprissero parte del terreno. Quest'ultimo è stato creato partendo da un piano suddiviso più volte, al quale è stato applicato un modificatore di tipo *Displace* che disloca i vertici della mesh in base all'intensità di una texture, nel caso in questione di tipo *Cloud*, ottenendo quindi una superficie irregolare. Un modo di modellare le radici sarebbe stato tramite curve di Bezier, alle quali applicare la stessa texture degli alberi. Tuttavia, la difficoltà maggiore è fare in modo che queste rimangano ancorate al terreno. L'idea è stata quindi quella di partire da un modello generico di albero realizzato tramite l'addon *Sapling Tree*, che è stato ruotato di 180° rispetto all'asse Y in modo tale che le ramificazioni fossero orientate verso il terreno. È stato quindi posto un object di tipo empty alla base del tronco. Sono stati infine aggiunti due modificatori:

- *Hook modifier*: è utilizzato per deformare una mesh, utilizzando come riferimento un altro oggetto, in questo caso di tipo empty. Compie un'operazione simile a quella effettuata dal proportional editing in edit mode. Nel caso in questione, questo modificatore permette di non "appiattire" totalmente il modello sul terreno a causa del prossimo modificatore;

- *Shrinkwrap modifier*: muove ogni vertice della mesh verso il punto più vicino di un'altra mesh, ossia il terreno. È stato scelto di utilizzare il metodo *Project* in modo tale da essere effettivo solo rispetto all'asse Z.

Tramite questi modificatori, è stato possibile avere delle radici che si adattassero automaticamente al terreno, indipendentemente dalla conformazione di quest'ultimo (fig. 4.8).

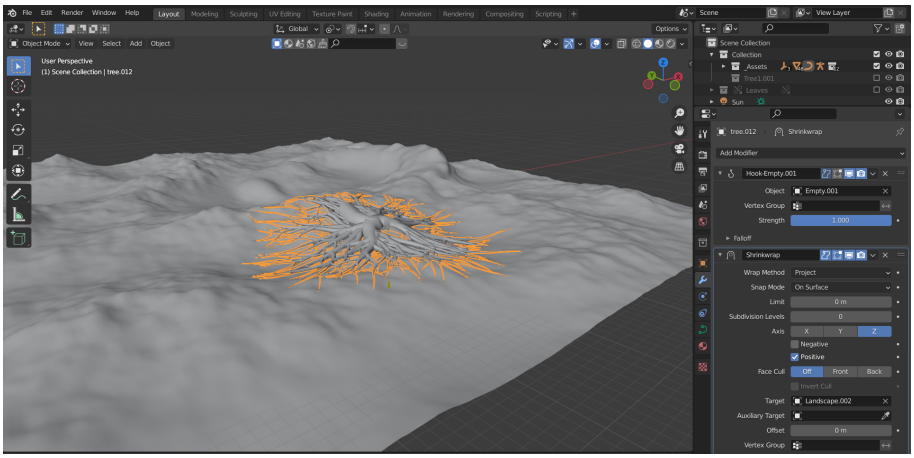

Figura 4.8: Visualizzazione in viewport delle radici

4.1.7 Hair Particle System

Il terreno, al fine di risultare fotorealistico, è stato ricoperto di piante, erba, foglie e pietre. Per collogare in maniera randomica ma controllata un gran numero di oggetti, Blender offre un sistema di distribuzione di meshes chiamato *Hair Particles*, una sotto categoria delle regolari particelle. Al fine di rendere la distribuzione non omogenea, è stato fatto un lavoro preliminare sulla mesh del terreno che consiste nell'assegnare dei *pesi* ai vertici che compongono la mesh. Questa operazione, chiamata *Weight painting*, si effettua selezionando la modalità apposita e dipingendo direttamente sulla mesh, utilizzando uno sistema di colori che va dal blu al rosso, e tra di essi i colori dello spettro del visibile secondo lo schema convenzionale chiamato *ROYGBIV*, permettendo quindi di assegnare un peso da 0.0 a 1.0 ai vertici selezionati. In figura 4.10 si può notare in vista ortografica il risultato del Weight Painting, realizzato in funzione della posizione e rotazione della camera, della sua lunghezza focale e della posizione degli alberi già sul terreno.

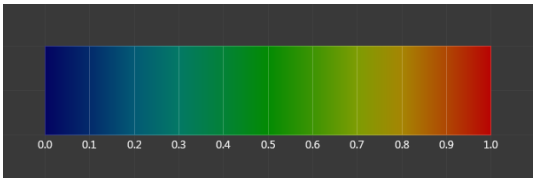

Figura 4.9: Spettro di colori per il weight painting

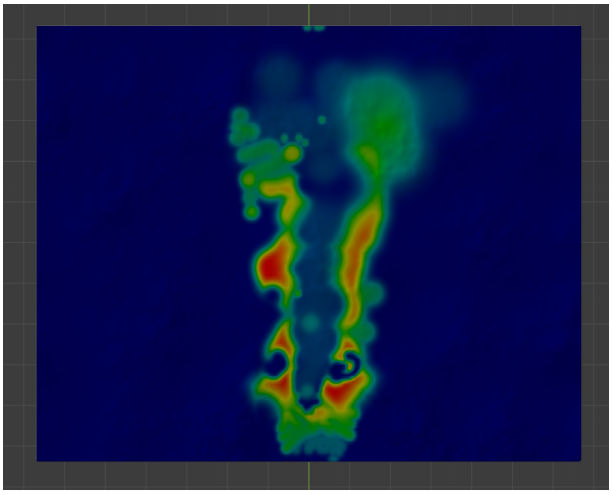

Figura 4.10: Weight painting applicato al terreno, in vista ortografica

Al terreno sono stati quindi applicati quattro diversi sistemi particellari, uno per ogni collezione contenente gli oggetti da distribuire. Sono stati utilizzate, per quanto riguarda gli oggetti di piante e erba, alcune collezioni provenienti dalla versione gratuita dell'asset pack *Graswald*, che offre un discreto numero di specie diverse di piante, la cui topologia e i materiali sono stati accuratamente studiati al fine di ottenere le caratteristiche tipiche di riflessione, trasparenza e colore che permettono

di raggiungere un buon livello di fotorealismo. I modelli di fiori sono stati invece realizzati partendo dal pistillo (la parte centrale del fiore) al quale è stato aggiunto un ulteriore sistema particellare di semplici piani texturizzati partendo da foto reali di petali a realizzazione della corolla. Per ogni sistema particellare, è stato configurato un numero variabile di elementi da distribuire (100-200 ca. oggetti per fiori e piante e 10000 ca. per l'erba) e in alcuni di essi è stata impostata la distribuzione interpolata mediante il pannello *children*. In questo modo vengono istanziati nuovi oggetti dalle proprietà di trasformazione relative rispetto all'oggetto *parent*, andando ad alleggerire il carico di lavoro sulla CPU. La distribuzione è stata personalizzata in modo da rendere randomici (entro certi limiti) i valori di rotazione rispetto alla normale alla superficie e la scala. Infine è stato inserito nel pannello relativo ai vertex group, al parametro densità, il weight painting realizzato precedentemente.

Figura 4.11: Render del terreno con i particle systems

4.1.8 Geometry Nodes

A partire dalla versione 2.93 di Blender, è possibile sperimentare i Geometry Nodes nella loro versione non definitiva, che avverrà verosimilmente nella versione 3.0. I Geometry Nodes (GM da ora in poi) possono essere visti come un nuovo modo di modellare, utilizzando i nodi come nello Shader Editor al fine di modificare o processare una mesh preesistente. Le potenziali applicazioni sono pressochè illimitate, in quanto i GM aprono le porte a un tipo di modellazione procedurale tipica di sistemi come quello presente in Houdini: in generale, viene creata una base per un determinato modello che può essere modificato a seconda dei contesti di utilizzo tramite semplici parametri. Nella versione attuale, tuttavia, i GM sono ancora in fase sperimentale e non-production-ready, e le funzionalità disponibili, limitate.

Nel caso della scena in questione, i GM sono stati utilizzati in alcuni casi come alternativa agli hair particle systems per istanziare oggetti, principalmente per le seguenti motivazioni:

- Gli hair particle systems non permettono di avere sistemi particellari nidificati. Un esempio di questo utilizzo è dato dall'oggetto di un albero che ha un sistema particellare per la distribuzione delle foglie sui rami. A patto di non applicare il modificatore relativo al sistema particellare e quindi realizzare un'unica mesh di albero e foglie, non è possibile distribuire l'oggetto albero con foglie sul terreno. Applicare il modificatore, tuttavia, significa lavorare con una mesh particolarmente densa di vertici, impossibile da istanziare mediante particles in numeri elevati.

- I Geometry Nodes, nella versione attuale, sono decisamente più efficienti in termini prestazionali degli hair particle systems nel gestire numeri elevati di oggetti da istanziare.

In questa scena, i GM sono stati quindi utilizzati per l'istanziazione delle foglie per il modello di un albero utilizzato per il background. In fig. 4.12 è mostrato il setup dei nodi, composto da:

- *Group Input*: utilizza come input del sistema dei nodi la mesh principale, in questo caso quella dell'albero;

- *Point Distribute*: distribuisce i punti sulla superficie dell'input group, tenendo in conto del weight painting effettuato sui rami definito dal vertex group *Group* e della densità specificata dall'omonimo parametro;

- *Attribute Randomize*: il termine attributo descrive alcuni tipi di dato presenti in un data-block che rappresenta la geometria. Alcuni di essi possono essere randomizzati tramite il nodo *Attribute Randomize*. In questo caso, è stata resa random, entro determinati limiti, la scale e la rotazione dei rami contenenti le foglie.

- *Point Intance*: ad ogni punto definito dal nodo *Point Distribute* viene istanziato l'oggetto o la collezione selezionati.

- *Join Geometry*: vengono unite le geometrie costituite dalla mesh originale con la mesh dei rami e foglie.

Medesimo procedimento è stato effettuato per l'istanziazione sul terreno del background dell'albero appena descritto, insieme a foglie, erba e fiori.

4.1.9 Lighting e render finale

Il lighting della scena ha previsto la presenza di una HDRI (High Dynamic Range Imaging) ossia un'insieme di immagini panoramiche a 360° con vari livelli di esposizione, che permette di ottenere una luce diffusa naturale, e una sun light, al fine di

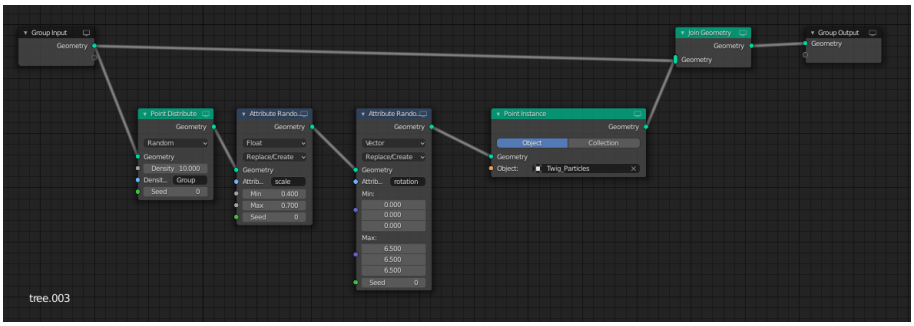

Figura 4.12: Geometry nodes per l'istanziazione delle foglie per il modello di un albero

rendere più decise le ombre. Il render è stato effettuato utilizzando Cycles; in conseguenza di questo, era necessario fare in modo che le riflessioni della luce rendessero il middleground particolarmente in ombra, andando a indicare la presenza di molti alberi. Per fare questo è stato posizionato un cubo a racchiudere middleground e foreground, al quale sono state eliminate la faccia frontale e superiore, in modo da creare un ostacolo alla luce proveniente dal lato destro dell'immagine.

I vari layer (background, middleground e foreground) sono stati esportati separatamente e, all'interno di ogni layer, sono stati esportati quanti più layer intermedi possibili, renderizzando separatamente *file* di alberi, al fine di facilitare il lavoro di post-produzione, ottenenendo la maggiore libertà compositiva possibile. Sono stati esportati anche alcuni pass aggiuntivi come il Z-depth pass per quanto riguarda il background, al fine di creare in post processing l'attenuazione atmosferica.
In fig. 4.13 è mostrata la composizione dei vari layers prima della fase di post-produzione.

Figura 4.13: Render finale, sul quale verrà effettuata la fase di post processing e compositing

4.2 Bosco invernale

4.2.1 Introduzione

Al termine della realizzazione della scena relativa al bosco in autunno, è stato richiesto di realizzare una scena simile ma con il bosco durante il periodo invernale, quindi con neve che coprisse gran parte del terreno e degli alberi (fig. 4.14).

Figura 4.14: Una delle reference per la scena invernale

La difficoltà maggiore è stata quella relativa all'utilizzo di assets invernali; era infatti necessario ricercare modelli fotogrammetrici (e non) o modellarli, simili a quelli utilizzati per la scena precedente, di fatto ricreando ex novo un ulteriore file .blend contenente gli assets invernali. La ricerca e creazione di assets per la scena del bosco in autunno (cfr. sezione 4.1.4) ha richiesto un notevole sforzo in termini di tempo, non più disponibile in quanto vicina la deadline relativa alla presentazione delle prime scene al cliente finale per l'approvazione. Era quindi necessario trovare un modo di reimpiegare gli assets per la scena autunnale ed è stato possibile farlo in breve tempo grazie al sistema di scripting in Blender che si basa sul linguaggio Python.

4.2.2 Shader procedurale per la neve

Il primo passo nella realizzazione della scena è consistito nel capire come rendere al meglio la neve. Uno dei possibili modi è tramite l'aggiunta di geometria, quindi partendo da un determinato modello 3D, selezionare le superfici di questo che dovrebbero avere della neve, duplicarle e separarle dall'oggetto iniziale, estruderle per aggiungere del volume e applicare all'oggetto così creato uno shader per la neve. Un procedimento di questo tipo è però abbastanza limitante sono diversi punti di vista:

- È lento e poco efficiente, impossibile da attuare su ogni oggetto dell'assets file in tempi ragionevoli;
- Poco flessibile, in quanto la neve ha una posizione predefinita, mentre dovrebbe adattarsi alla posizione e rotazione del modello posizionato nella scena;
- L'aggiunta di geometria renderebbe ancora più complessa, da un punto di vista computazionale, la scena.

Si è deciso quindi di rendere la neve solo da un punto di vista di shader. Nel caso si fosse deciso di utilizzare Cycles per la versione finale, sarebbe stato possibile ottenere tramite il *True Displacement* l'effettivo dislocamento della mesh nei punti in cui è presente la neve, agendo direttamente nello shader editor. Il *True Displacement*, tuttavia, non è presente in EEVEE, motore di rendering utilizzato in questo caso, e viene reso in questo caso mediante una semplice *Bump map*.

Lo shader finale è stato realizzato su tre livelli, in quanto, una volta creato lo shader della sola neve, era necessario aggiungerlo allo shader preesistente (senza neve), tenendo in conto di alcune caratteristiche dell'oggetto e delle textures già utilizzate. Lo shader relativo alla sola neve è completamente procedurale, non utilizza quindi textures esterne, ma una serie di nodi; in particolare il nodo principale *Principled BSDF*, al quale è stato applicato un colore di base tendente al bianco e vari altri nodi principalmente di tipo noise e voronoi, collegati ai layers relativi a *normal*, *clearcoat* (strato speculare esterno, da aggiungere al di sopra degli altri), e *subsurface* (definisce in che modo la luce si comporta all'interno di un materiale traslucido). Il materiale così creato è stato quindi inserito all'interno di uno *sha-*

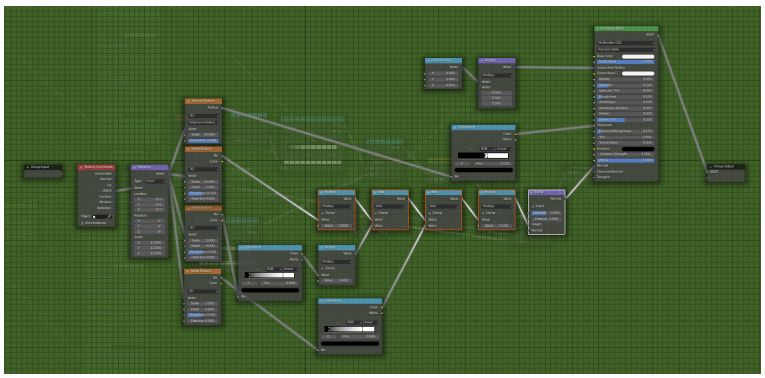

Figura 4.15: Shader procedurale per la neve

der group (denominato *Snow*), che permette di raggruppare più nodi e trattarli da quel momento in poi come un unico nodo, rendendo più semplice l'annidamento di shaders.

Al secondo livello, quindi, il gruppo *Snow*, viene aggiunto allo shader preesistente utilizzando informazioni relative alla superficie del modello. In particolare, il nodo *Geometry*, che fornisce informazioni geometriche riguardo l'oggetto 3D, è stato utilizzato per ottenere la normale alla superficie del modello, relativamente alla componente Z, tramite il nodo *Separate XYZ*. Si presuppone, infatti, che la neve si depositi sulla superficie dall'alto verso il basso. Agendo in questo modo, lo shader della neve verrà applicato automaticamente al modello tenendo in conto della rotazione attuale dello stesso all'interno della scena, rendendo l'integrazione di tutti gli oggetti presenti molto più naturale e fotorealistica. Alla componente Z della normale alla superficie, viene aggiunta una seconda componente, anch'essa relativa all'asse Z, ma derivante dalla normal map dello shader originale. In alcuni casi, nei modelli fotogrammetrici utilizzati non era presente una normal map e, in questo caso, lo shader creato non ne terrà conto, utilizzando come valori di default per descrivere la normale (0, 0, 0). L'utilizzo della normal map in questo contesto permette di fare in modo che la neve si adagi al modello tenendo in conto delle insenature descritte dalla componente z della normale alla superficie. L'insieme di queste informazioni vengono quindi utilizzate sotto forma di maschera come *factor* per il *Mix Node*, che mixa il gruppo *Snow* allo shader del modello senza neve. Anche

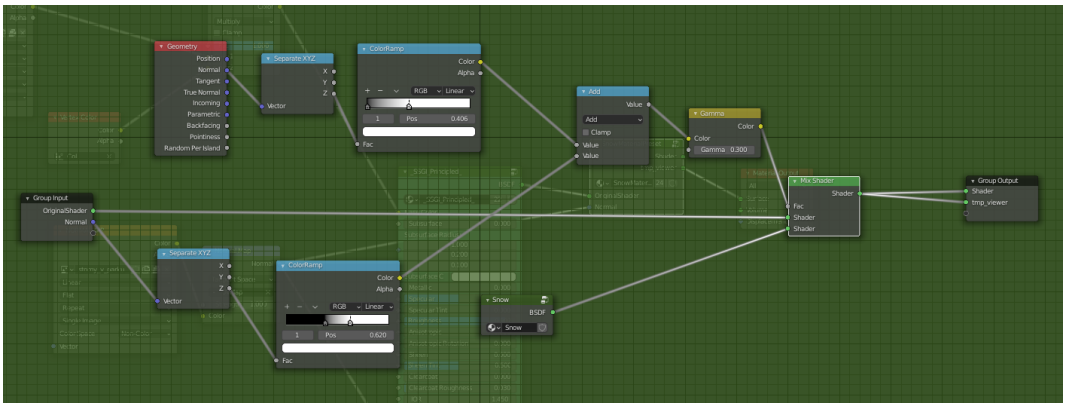

Figura 4.16: Secondo livello dello shader

in questo caso, è stato creato uno shader group, denominato *Snow Material Preset*, che comprende i nodi creati nel primo e secondo livello e che riceve come input lo shader BSDF originale e la normal map.

Al terzo livello (fig. 4.17), quindi, troviamo semplicemente l'applicazione dello *Snow Material Preset* allo shader originale. In figura 4.18 è possibile osservare lo stesso modello fotogrammetrico, prima e dopo l'applicazione dello *Snow Material Preset*.

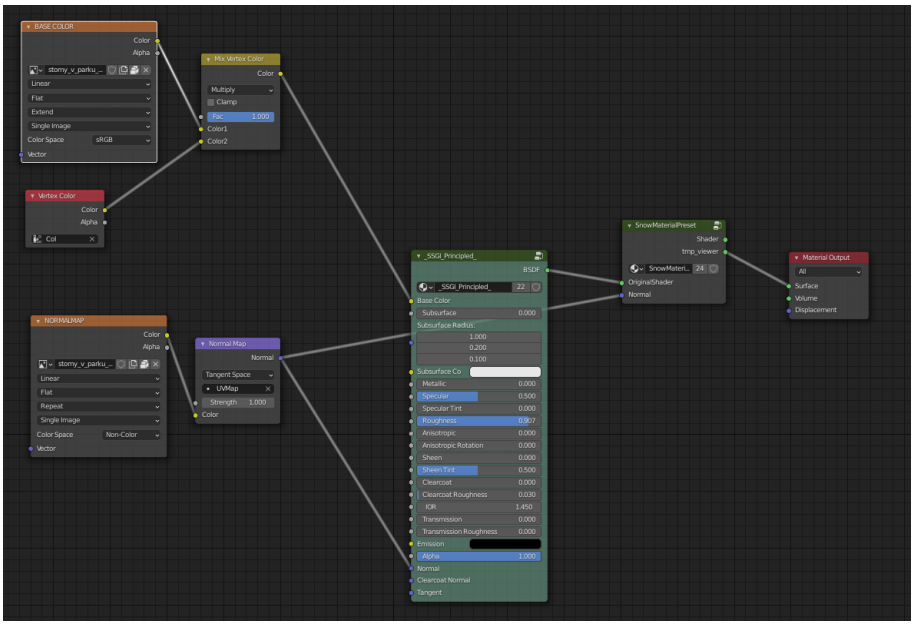

Figura 4.17: Terzo livello dello shader per un tipico setup dei modelli utilizzati

4.2.3 Automazione di operazioni con Python

Blender offre una API (Application Programming Interface) in Python che permette di modificare le funzionalità del programma così come di aggiungerne di nuove. All'interno di Blender è presente un interprete Python che è caricato all'apertura del programma e rimane attivo finchè non viene chiuso. L'ambiente offerto dall'interprete (che quindi compila un'istruzione per volta) è tipico di qualsiasi ambiente di sviluppo Python, con l'aggiunta dei moduli *bpy*, che permette l'accesso ai dati di cui fa uso Blender, e *mathutils*, che fornisce una serie di funzioni matematiche. È possibile accedere a qualsiasi tipo di dato di cui fa uso Blender e modificarlo (tramite *bpy.data*), così come qualsiasi elemento della user interface.

È possibile utilizzare Python per molteplici scopi, ad esempio per la modellazione procedurale o creazione di addon utili a effettuare le più svariate operazioni. Nel caso in questione, Python è stato utilizzato per automatizzare l'aggiunta dello *Snow Material Preset* a tutti gli oggetti presenti nel file asset, realizzando un setup simile a quello rappresentato in fig. 4.17.

Lo script realizzato (fig. 4.19) prende innanzitutto in considerazione una selezione di oggetti, sulla quale viene effettuato un ciclo for e viene richiamato lo *Snow Material Preset* come gruppo di nodi. Per ognuno degli oggetti, ne viene selezionato il materiale e viene posto l'insieme dei nodi nella variabile *nodes*. Viene quindi ritornato il nodo del *Material Output*, presente in ogni materiale e posta la sua posizione in una variabile. Dato che non necessariamente è presente un nodo di tipo *Principled BSDF* e, anche se fosse presente, questo potrebbe non essere l'ultimo nodo collegato all'output, viene memorizzato in *m_princ* l'ultimo nodo collegato all'input *surface* del material output utilizzando il metodo *from_node* applicato alla lista di *links*, ossia l'insieme dei collegamenti tra i nodi. Tramite metodo *get*, viene ritornato

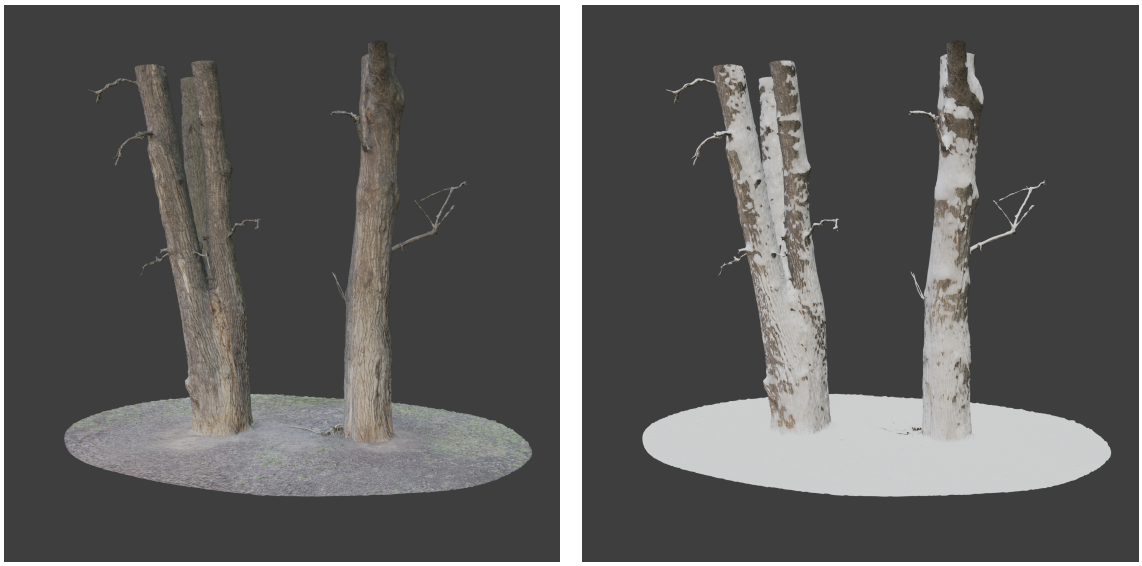

Figura 4.18: Stesso modello fotogrammetrico, senza e con lo shader della neve applicato

il nodo relativo alla normal map, se disponibile, effettuando una ricerca per nome. Viene quindi definita la posizione del nuovo gruppo che si andrà a istanziare, prendendo in considerazione la posizione del material output. Vengono quindi effettuati i collegamenti tra quest'ultimo e il gruppo *Snow Material Preset*, che avrà in input l'ultimo nodo che era collegato al material output e la normal map.

4.2.4 Composizione della scena, lighting e conclusioni finali

Una volta terminata la preparazione dei modelli 3D con la neve in un nuovo file assets, è arrivato il momento di comporre la scena. Dopo la creazione del terreno, realizzato in modo simile al terreno della scena autunnale (cfr. 4.1.6), è stato scelto di rappresentare, anche in questo caso, una radura, attraversata da un sentiero che parte da alcuni alberi in primo piano. Per lo shader del terreno sono stati utilizzati due materiali diversi, realizzati tramite due diverse texture per il suolo, e ad entrambi è stato applicato lo *Snow Material Preset*. I due materiali sono stati utilizzati al fine di ottenere una distinzione tra terreno con neve e sentiero, agendo in quest'ultimo caso direttamente nel preset della neve e riducendone la diffusione tramite la *color ramp* relativa alla normale sull'asse z. Per quanto riguarda il factor per il *Mix Node* tra i due materiali, in questo caso è stato realizzato tramite vertex paint, creando di fatto una maschera. Il vertex paint è un'alternativa al texture paint: in quest'ultimo caso, si va a creare una texture da usare come maschera, utilizzando una colorazione in scala di grigi come fattore da 0 a 1 (da nero a bianco), a cui associare l'uno o l'altro materiale. In modo simile, tramite vertex paint è possibile definire un colore direttamente associato ai vertici della mesh. È possibile definire più vertex paints sulla stessa mesh e accedere a questi canali tramite lo Shader Editor, utilizzando l'omologo nodo. La composizione della scena ha previsto il posizionamento di una

```python
import bpy

so=bpy.context.selected_objects
snow_group=bpy.data.node_groups["SnowMaterialPreset"]

for obj in so:
    mat=obj.active_material

    nodes=mat.node_tree.nodes

    m_out=nodes.get("Material Output")
    m_out_loc=m_out.location

    m_princ=m_out.inputs[0].links[0].from_node

    normal=nodes.get("Normal Map")

    group = nodes.new(type = 'ShaderNodeGroup')
    group.location[0]=m_out_loc[0]
    group.location[1]=m_out_loc[1]+250

    group.node_tree = snow_group

    mat_links=mat.node_tree.links
    mat_links.new(m_princ.outputs[0],group.inputs[0])

    if normal!=None:
        mat_links.new(normal.outputs[0],group.inputs[1])

    mat_links.new(group.outputs[0],m_out.inputs[0])
```

Figura 4.19: Script per l'aggiunta dello *Snow Material Preset*

camera con lunghezza focale 35mm. Posto il sentiero in funzione della posizione della camera, il resto della scena è stata realizzato piazzando i vari modelli 3D con la neve e aggiungendo diversi sistemi particellari, in modo analogo a quanto descritto nella sezione 4.1.7, rappresentanti principalmente piante e sassi. Anche in questo caso il lighting è stato realizzato tramite una HDRI coerente con la scena di un paesaggio invernale e una sun light. In aggiunta, è stato inserito un cubo che racchiudesse l'intera scena al quale è stato assegnato un materiale volumetrico tramite il *Principled Volume*, a bassa densità, a simulare della foschia. In questo caso è stato utilizzato EEVEE per il render. Dovendo comunque rendere delle zone più in ombra di altre senza appesantire eccessivamente la scena con modelli 3D di alberi, è stato utilizzato uno stratagemma: sono stati istanziati dei piani, i quali sono stati suddivisi più volte. Sulla mesh generata, è stata effettuata una selezione randomica delle facce di circa il 30%, che sono state eliminate. In questo modo è stato possibile degli ostacoli alla luce generata dalla sun light, creando quindi delle ombre "artificiali" che ne riducessero l'afflusso, ma non bloccandola completamente (fig. 4.20).

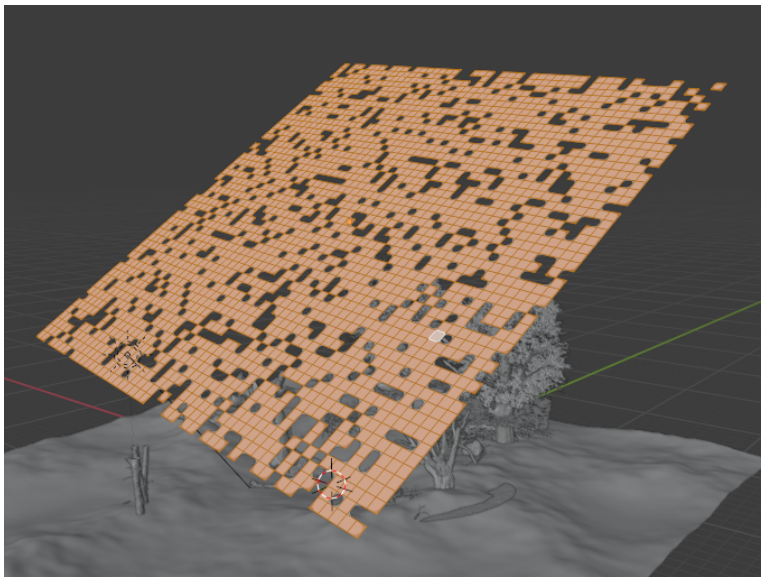

Figura 4.20: Esempio di utilizzo dei piani per la creazione delle ombre

Il livello di dettaglio raggiunto in questa scena non è particolarmente elevato in quanto era sicuramente necessaria ulteriore lavorazione. Il render prodotto è infatti preliminare, ossia finalizzato alla presentazione al cliente di un concept. È stato tuttavia scelto di focalizzare l'attenzione su altre tipologie di scene e accantonare quella in questione. Ciononostante, la realizzazione della scena è stata funzionale all'acquisizione di competenze relative alla velocizzazione e organizzazione del flusso produttivo tramite l'utilizzo di Python come linguaggio di scripting.

Figura 4.21: Render finale della scena invernale

4.3 Realizzazione dell'asset dell'ampolla

4.3.1 Introduzione

L'asset dell'ampolla contenente il veleno è stato particolarmente importante in quanto presente in diverse scene, tra le quali quella relativa alla simulazione del fluido del veleno che cade sulla mela. In questo caso, sono state utilizzate diverse references di ampolle, come quella in fig.4.22, ma le più utili sono state quelle relative alla realizzazione degli ornamenti. Come già accennato nella sezione 3.2, questi ultimi sono stati influenzati dalle forme sinuose e armoniche utilizzate nella saga de *Il Signore degli Anelli* per la caratterizzazione degli ambienti relativi ai luoghi in cui vivono gli Elfi, e in particolare Rivendell (vedi fig. 3.5).

Figura 4.22: Esempio di reference per l'ampolla

4.3.2 Modellazione tramite modificatori

La modellazione di un oggetto di questo tipo può essere fatta in vari modi, in base agli obbiettivi prefissati. In questo caso, era necessario ottenere un modello 3D facilmente modificabile nelle forme e proporzioni, in modo da andare incontro alle eventuali richieste di modifica da parte del cliente finale in tempi ragionevoli. Per questo motivo, è stato fatto un uso consistente di modificatori, ottenendo la forma finale modellando il minimo indispensabile. In particolare, è stato modellata solo

una porzione dell'intera ampolla, definendone sostanzialmente solo il profilo (fig. 4.23).

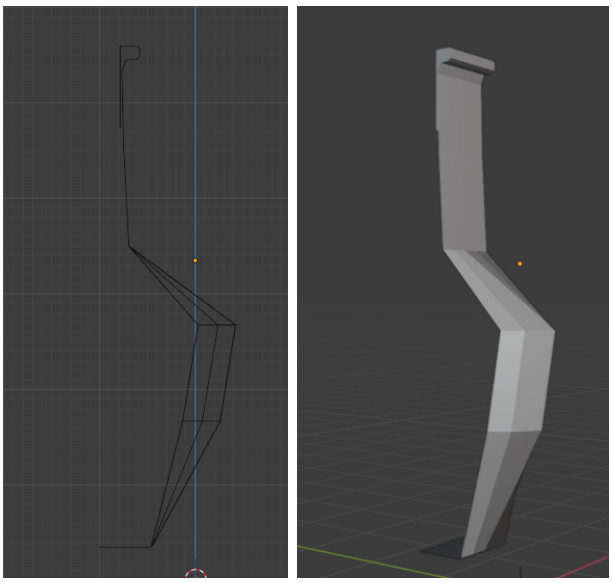

Figura 4.23: Sezione dell'ampolla

Avendo ottenuto questa forma di base, qualsiasi modifica sarebbe stata possibile semplicemente traslandone i vari spigoli. Come si può vedere in fig.4.24, che mostra

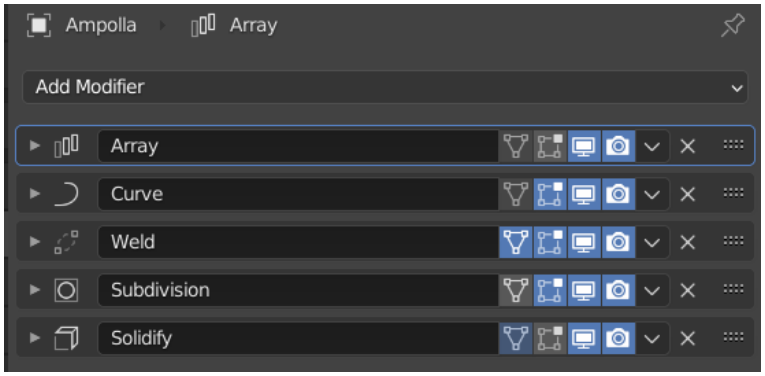

Figura 4.24: Stack dei modificatori per l'ampolla

l'intero stack dei modificatori, il passo successivo è stato l'aggiunta del modificatore *Array*, che crea una o più copie dell'oggetto (si è deciso di ottenere otto ripetizioni della sezione di base) su un determinato asse, tenendo in conto di un offset che in questo caso è stato posto posto a zero. Il modificatore *Curve* permette invece di deformare la mesh tramite una curva, quindi un altro oggetto, definendone un asse di deformazione, in questo caso l'asse X globale. È stato utilizzata a tal fine una curva di tipo *Bezier Circle*, in quanto le ripetizioni ottenute tramite l'array modifier dovevano essere deformate a formare una rotazione completa di 360°. Il modificatore *Weld* permette di effettuare un *merge* dei vertici al di sotto una determinata distanza, posta in questo caso a 0.1 unità di Blender. Il motivo di questo modificatore è

che nei punti di giunzione tra le varie sezioni si andava a creare un'apertura, dovuta ai modificatori precedenti e in particolare al Curve modifier. Troviamo quindi il modificatore *Subdivision Surface*, che utilizza il classico algoritmo Catmull-Clarke per suddividere e rendere più *morbida* la mesh, e il modificatore *Solidify*, che estrude la mesh e ne crea uno spessore. In figura 4.25 si può osservare il risultato finale.

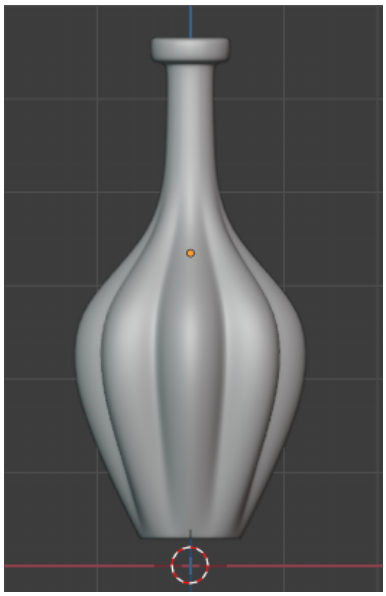

Figura 4.25: Mesh ottenuta in seguito all'utilizzo dei modificatori

4.3.3 Modellazione degli ornamenti

Definita la forma principale dell'ampolla, sono stati modellati diversi ornamenti. Alcuni di essi, come ad esempio il tappo, sono stati modellati con tecniche standard, quindi partendo da forme base (principalmente cubi e cilindri) e applicando estrusioni e scalamenti fino a raggiungere il risultato desiderato. In praticamente tutti i casi descritti, è stato aggiunto il modificatore *Subdivision Surface*. L'aggiunta di questo modificatore permette di avere diversi livelli di dettaglio definiti dal numero delle suddivisioni, che possono essere diversi in preview e in render. Tuttavia, questo tipo di utilizzo, prevede una mesh che viene definita *sub-d ready*, ossia che abbia, oltre che le forme di base, degli *edge loops* di sostegno, che servono a definire dei limiti nella creazione della mesh "curva", risultato dell'algoritmo *Catmull-Clarke*. Una tecnica leggermente diversa è stata usata per i due ornamenti posti sul collo dell'ampolla. In questo caso è stata creata una copia provvisoria dell'ampolla alla quale sono stati applicati tutti i modificatori. Fatto questo, sono state selezionate le facce sulle quale posizionare gli ornamenti, separate dall'oggetto creato e estruse. In questo modo si è ottenuto una perfetta aderenza di questi alla mesh originale. Al risultato così ottenuto, sono stati aggiunte delle mesh rappresentanti dei cuori. Questi sono stati modellati partendo da un piano del quale è stato fatto il merge ottenendo un solo vertice che è stato estruso fino a raggiungere la forma desiderata.

È stato quindi aggiunto un modificatore di tipo *Solidify*, un modificatore *Subdivision Surface* e un modificatore *Decimate*, con un fattore di merge molto alto. L'obbiettivo di quest'ultimo era ottenere una mesh che fosse particolarmente decimata e con spigoli netti, andando a ricreare la tipica forma dei cristalli. Una geometria così creata è stata utile anche in fase di shading per ricrearne le riflessioni. Per effettuare le ripetizioni della mesh su ogni sezione dell'ampolla, in questo e in altri casi che ne necessitavano, è stato aggiunto un modificatore array con otto ripetizioni e un offset gestito da un oggetto di tipo empty esterno, posizionato al centro della mesh relativa all'ampolla. Quest'ultimo è stato ruotato di un angolo pari a 360°/n° ripetizioni, in modo da ottenere una ripetizione circolare uniforme.

Per la modellazione del manico, degli ornamenti che formano il piedistallo e per la catenella che ruota attorno al collo dell'ampolla, sono state utilizzate delle curve di tipo *Nurbs Path*, che permettono di avere particolare controllo sulla modellazione. A queste è stato aggiunto il parametro Bevel, che ne definisce uno spessore, di tipo Profile, che permette di ottenere un curva con degli spigoli abbastanza morbidi. Nel caso della catenella, la curva è stata utilizzata come oggetto per il modificatore *Curve*, applicato alla mesh di base degli anelli, alla quale è stato aggiunto il modificatore Array, con la spunta *Fit Curve*, in modo da avere un numero di ripetizioni sufficienti a coprire l'intera curva.

Figura 4.26: Modello 3D dell'ampolla con gli ornamenti

4.3.4 Shading e texturing

Lo shader per l'ampolla è stato reso più difficile da rendere in maniera convincente per la scelta di utilizzare EEVEE come motore di rendering nelle scene in cui questa compariva. Questo perchè il modo in cui gestisce le riflessioni EEVEE (e in generale i motori di rendering real-time) non è particolarmente accurato a causa di limiti tecnici imposti sul calcolo di riflessioni e rifrazioni, che non prevedono l'utilizzo del ray tracing, come descritto nella sezione relativa ai motori di rendering (cfr. 2.5). Il principale problema in questo senso è stato dato dal fatto che era necessario non solo ricreare le riflessioni relative all'ampolla, ma anche al liquido che a sua volta conteneva, anch'esso costituito da materiale di tipo speculare. In EEVEE non è possibile vedere attraverso un materiale di tipo *Glass* un altro materiale dello stesso tipo. È stato tuttavia possibile ottenere un risultato soddisfacente utilizzando alcuni stratagemmi.

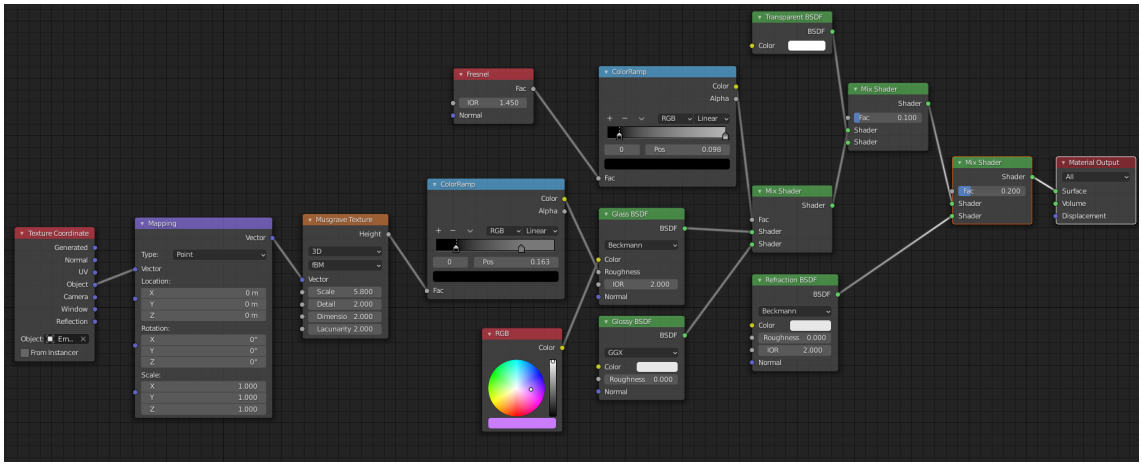

Figura 4.27: Nodi per lo shading dell'ampolla

Per i materiali di tipo riflessivo, è necessario innanzitutto abilitare nel pannello relativo alle impostazioni del materiale la spunta *Alpha Blend*, che utilizza un fattore di trasparenza in funzione del canale alpha delle textures utilizzate, e il parametro *Screen Space Reflection*, approssimazione del ray tracing per i motori real time per il calcolo delle riflessioni. La maggior parte delle riflessioni e rifrazioni è stata ottenuta tramite il nodo *Glass*, al quale è stato collegato un nodo di tipo *RGB values* per definirne il colore principale e un nodo *Musgrave Texture*, il cui mapping è pilotato da un oggetto empty posto al centro dell'ampolla, collegato all'input *Roughness*, al fine di definire delle riflessioni non uniformi su tutta la superficie. Al fine di aumentare il numero di riflessioni, è stato aggiunto un nodo *Glossy*, con un fattore definito dal node *Fresnel*. Questo definisce quanta luce viene riflessa su di una superficie e quanta ne viene rifratta, tenendo in conto dell'angolo compreso tra la normale alla superficie e dell'angolo di visione. In questo modo si va a definire un maggiore livello di riflessioni dato sui "contorni" dell'oggetto, in base al posizionamento della camera. Gli ultimi due nodi, *Transparent BSDF* e *Refraction BSDF*, servono a superare in qualche modo il limite imposto da EEVEE riguardo alla presenza di più materiali

di tipo *Glass*, aggiungendo un livello di trasparenza e di rifrazione che permettono di vedere attraverso il primo materiale.

Il resto dei materiali, utilizzati per gli ornamenti, è stato realizzato tramite il *Principled BSDF*, andando a aumentare il valore relativo al *Metallic* e riducendo la *Roughness*, ottenendo un materiale simile all'oro. In più, in seguito alla fase di UV unwrapping, sono state aggiunte delle texture collegate al nodo *Displacement*, in modo da ricreare delle decorazioni su alcune parti degli ornamenti.

4.3.5 Modellazione e shader procedurale per il veleno

Per quanto riguarda il veleno, in questo caso non c'è stata alcuna simulazione di fluidi, in quanto il modello doveva essere utilizzato per alcune scene statiche, differentemente dalla situazione descritta nella sezione successiva. È stata quindi ricavata una mesh direttamente da quella dell'ampolla, in seguito all'aver applicato su una mesh temporanea i modificatori. Alla mesh così ottenuta, è stato aggiunto un modificatore di tipo *Displace* utilizzando una mesh di tipo Cloud, selezionando come target un vertex group relativa alla superficie superiore del liquido. Agendo su bassi valori del parametro strength è possibile ottenere una superficie non totalmente piatta.

Lo shader per il veleno consta di due parti, una relativa alla superficie, e uno relativo al volume. Il setup nel primo caso è simile a quello utilizzato per l'ampolla. Il

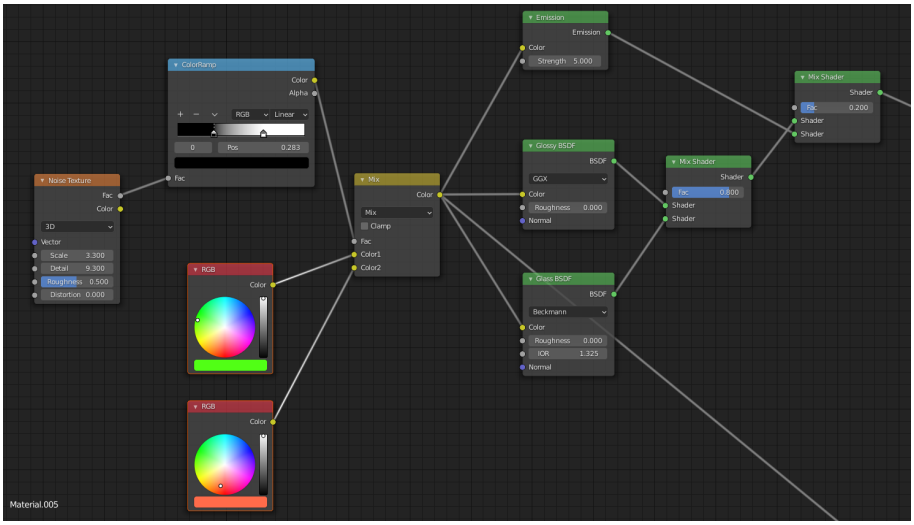

Figura 4.28: Shader per la superficie del veleno

colore è costituito da due tinte, il cui factor è definito da una texture di tipo noise. Il colore così ottenuto è stato utilizzato come input del nodo *Glass*, *Glossy* e *Emission*, nonchè per il *Principled Volume*. Quest'ultimo è stato utilizzato per definire un volume interno al liquido. Differentemente da Cycles, in EEVEE non è possibile definire un volume a partire da una qualsiasi superficie. È stato quindi necessario, tramite un mapping adeguato, andare a posizionarlo al centro della mesh, utilizzando un nodo di tipo *Gradient Texture*, che genera un valore interpolato partendo dal

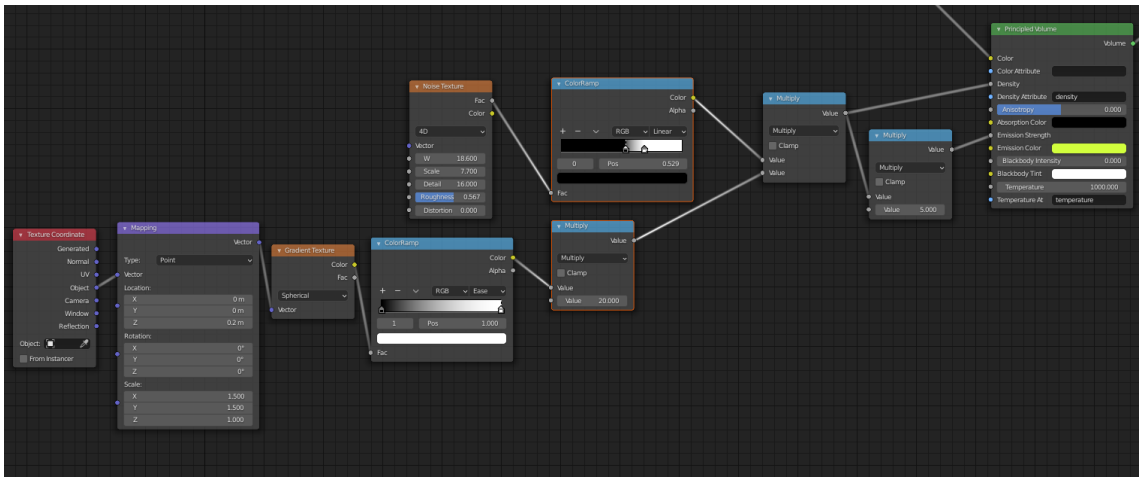

Figura 4.29: Shader per il volume del veleno

vector che gli si passa come input. Il risultato ottenuto è stato moltiplicato per l'output del *Noise Texture*, definendo un volume irregolare, e inserito nei valori riferiti alla densità e alla forza dell'emissione del principled volume.

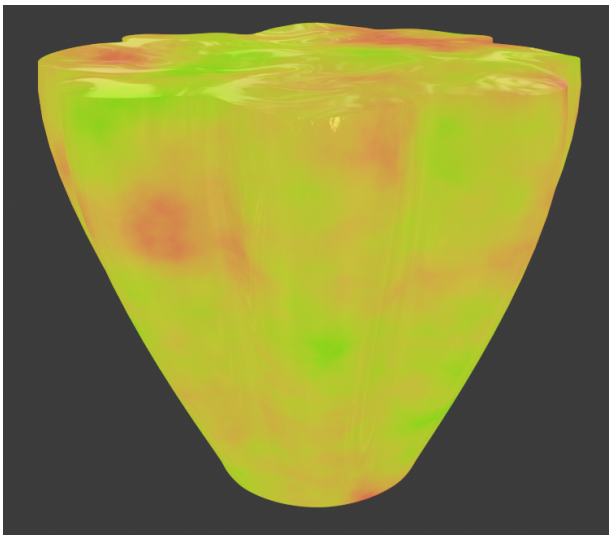

Figura 4.30: Oggetto 3D per il veleno

In figura 4.31 è possibile vedere un render dell'ampolla completa.

Figura 4.31: Render dell'ampolla in un semplice environment

4.4 Scena della simulazione del fluido del veleno

Un'altra scena realizzata è stata quella relativa alla simulazione del fluido del veleno che cade sulla mela. L'obbiettivo era quello di rappresentare un fluido leggermente viscoso, che si adagiasse sulla mela, sorretta da una mano. Per quanto riguarda le

references, si fa riferimento alla sezione 3.2, e in particolare alle scene dell'opening della serie TV *Daredevil* (fig.3.3). Nel caso in questione, la mano e la mela erano già in praticamente definite, in quanto risultato della fase di shooting fotografico. Era quindi necessario costruire la scena e la simulazione ricreando una rappresentazione 3D coerente a partire da una foto, in modo tale da donare profondità a quest'ultima e permettere un leggero movimento di camera, come descritto dallo script.

4.4.1 Realizzazione dell'environment e projection mapping

È stato scelto di partire dalla rappresentazione dell'environment. Il motivo per il quale è stata fatta questa scelta è stato dettato dal fatto che, al fine di effettuare una simulazione convincente, era innanzitutto necessario definire l'ambiente nel quale questa si sarebbe svolta e capire con quali oggetti il fluido doveva interagire. Partendo da una foto scattata con l'ausilio del green screen, è stato scelto di ricreare in 3D la mano, la mela e aggiungere parte della manica attorno al braccio, partendo direttamente dalla versione scontornata della foto. Il fine ultimo era infatti, anche in questo caso, il fotorealismo e quindi era d'aiuto a tal fine sfruttare le proporzioni e le textures derivanti dalla foto. La modellazione della mano è stata effettuata in prima battuta tramite lo *Skin Modifier*: estrudendo un singolo vertice, si va a definire la forma principale (palmo e dita). Lo *skin modifier*, insieme al *Subdivision Surface*, permette di ottenere una geometria generando una estrusione con un raggio definito per-vertice, modificabile tramite il comando Ctrl-A. Le proporzioni più dettagliate sono state definite tramite sculpting. Tuttavia, non era necessario ottenere una mesh perfettamente corrispondente a quanto mostrato in foto, in quanto utilizzando il canale alpha della foto, sarebbe stato possibile ricreare le trasparenze sul materiale della mesh. La mela è stata invece creata utilizzando un cubo al quale è stato applicato un modificatore *Subdivision Surface* e, anche in questo caso, sono state necessarie alcune correzioni tramite lo sculpting. La manica della veste è invece in semplice piano suddiviso, scolpito al fine di ricreare la forma desiderata. Come già accennato, le texture di mela e mano sono state applicate direttamente dalla foto. La tecnica sulla quale si basa questa scelta si chiama *projection mapping*, che permette di proiettare una texture su di una mesh, così come potrebbe fare un proiettore su una superficie: l'uv unwrapping generato è stato creato utilizzando l'opzione *Project from View* e posizionandolo in relazione alla foto. Il materiale, in entrambi i casi, è stato realizzato con un semplice setup, utilizzando la foto come canale color del nodo *Diffuse*, mixato con il nodo *Transparent* tramite il canale alpha della foto.

4.4.2 Simulazione del fluido in Blender con Mantaflow

La simulazione doveva rappresentare la caduta del liquido sulla mela. Come tipologia di liquido, si è scelto di realizzare un fluido semi-viscoso, che avesse quindi della forza di attrito sulla superficie di contatto nella forma $\vec{F} = -k \cdot n \cdot \vec{v}$, in cui k dipende dalla forma della superficie dell'oggetto e n dipende dalle proprietà del liquido. Si

è partiti con la definizione del dominio, ossia l'oggetto che deve contenere l'intera simulazione, racchiudendone all'interno il collo dell'ampolla e la mela. Sono stati quindi definiti come *effector*, ossia oggetti che interagiscono con il liquido tramite collisione, l'oggetto della mela e una seconda mesh, costituita da un cilindro cavo con un determinato spessore, a rappresentare il collo della bottiglia. È infatti opportuno nelle simulazioni di fluidi non utilizzare le mesh "originali", ma alcune mesh più semplici che le approssimano, al fine di non appesantire eccessivamente il carico di lavoro sulla CPU. È stato quindi posizionato all'interno del cilindro una sfera di tipo *inflow*, che è responsabile della generazione del fluido all'interno della simulazione. Per facilitare la fuoriuscita del fluido dall'effector rappresentato dal cilindro, è stata definita una forza iniziale sulla componente x pari a $x = -3\frac{m}{s}$, in accordo con la posizione e rotazione dell'ampolla. Data l'origine del fluido abbastanza ridotta in termini di dimensioni rispetto alla grandezza del dominio, è stata necessaria una risoluzione abbastanza elevata, impostandola a 150. Quest'ultima rappresenta il numero di suddivisioni del domain sul lato lungo in *voxels*, celle 3D all'interno delle quali vengono memorizzate le informazioni sulla simulazione, ossia il comportamento del fluido all'interno di esse. Al fine di ottenere una simulazione più precisa, è stato aumentato anche il valore di *Timesteps minimum* e *Timesteps maximum*, rispettivamente a 2 e 8, definendo quindi un tetto minimo e massimo nel numero di *steps*, il numero di calcoli relativi alla posizione delle particelle del fluido, calcolate per ogni frame. La viscosità del fluido è stata definita nel pannello relativo, con un valore di 0.001. Al termine della simulazione è stato effettuato il baking della mesh, definendo il *particle radius* a 1.2 *unità di griglia*, andando quindi a definire un raggio intorno alla singola particella. La mesh che si va a generare terrà in conto di questo valore in modo da considerare quanta area attorno a una particella deve essere considerata facente parte del liquido. Un valore più alto in questo senso permette di ottenere un liquido più omogeneo, da un punto di vista della mesh generata.

Figura 4.32: Render di un frame della simulazione, a cui seguirà la fase di post-processing

4.5 Realizzazione del modello 3D del nano

Un'ulteriore scena prevedeva la modellazione di un mezzo busto rappresentante un nano. Anche in questo caso, l'obbiettivo principale era riuscire a ottenere un personaggio che fosse abbastanza fotorealistico e che avesse al tempo stesso delle proporzioni e una mimica facciale accattivante.

Nel momento in cui ci si approccia alla modellazione di un personaggio umanoide, si fa spesso riferimento alla definizione di *Uncanny Valley*. Questo concetto nasce in ambito robotico, ma è stato successivamente esteso al mondo della Computer Animation e in generale della CGI. È stato introdotto per la prima volta nel 1970 da Masahiro Mori, professore di robotica presso il Tokyo Institute of Technology, per descrivere la reazione da parte di un essere umano alla vista di un robot umanoide: egli spiega come la realizzazione di un robot dalle fattezze umane, incrementa il senso di gradevolezza agli occhi di una persona, ma fino al punto in cui si raggiunge la cosiddetta *uncanny valley*, in cui la vista di una figura dalle caratteristiche all'apparenza simili a quelle umane, produce una sensazione di disagio crescente. Il saggio da lui scritto, come egli stesso sottolinea, non è stato frutto di uno studio realizzato con metodo scientifico, ma derivante da una serie di osservazione pratiche svolte nella realizzazione di protesi della mano e si propone principalmente come punto di riferimento per i designer di robot umanoidi.

Nel caso della modellazione del nano, il risultato non doveva apparire eccessivamente cartoonesco, ma doveva risultare un personaggio semi-umano credibile, avendo comunque bene in mente l'iconografia rappresentata dai tipici personaggi rappresentati dal film Disney. A questo fine, è stata posta particolare attenzione

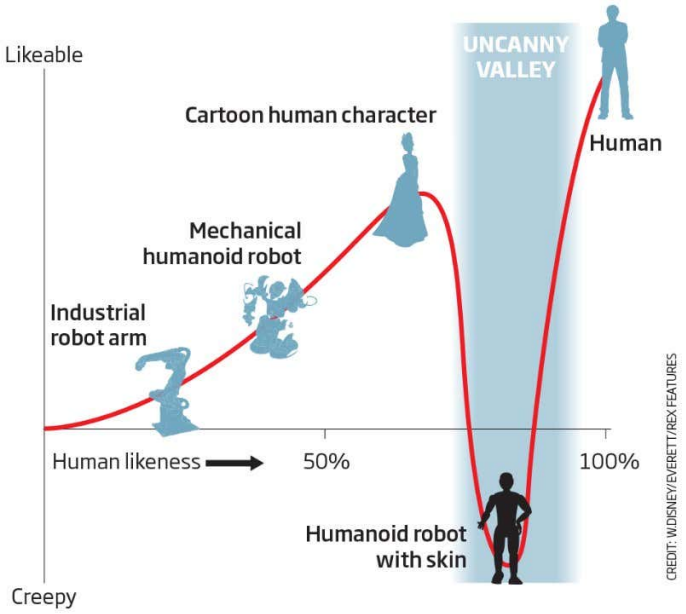

Figura 4.33: Grafico che mostra il concetto di *Uncanny Valley*

alle proporzioni facciali e alla creazione dei materiali, in particolar modo per la pelle. Per quanto riguarda la references principali, si fa riferimento alla sezione 3.2, e in particolare alla figura 3.2, in relazione alla scelta compositiva e di lighting che si cercava di ottenere. Oltre queste, sono state proposte alcune references direttamente dal cliente finale, di cui un esempio in fig.4.34, in cui si possono notare alcune delle caratteristiche che il personaggio avrebbe dovuto avere, ossia un anziano, sorridente nano.

4.5.1 Sculpting in Blender: tipologie e realizzazione della mesh

Esistono diverse metodologie e workflow per ottenere un modello tramite sculpting. Prendendo in esempio la creazione di un personaggio umanoide, solitamente si parte con la definizione delle forme di base tramite le principali primitive, andando a evidenziare le proporzioni che si cerca di ottenere. Si vanno quindi a creare diverse mesh che vanno a rappresentare le varie parti del personaggio e si cerca di tenerle separate quanto più possibile, unendole solo in una fase più avanzata, in modo tale da avere la possibilità di lavorare singolarmente sulle varie parti, rendendo la gestione del progetto più semplice. Lo sculpting viene quindi effettuato su ognuna di queste mesh e, in seguito all'unione di queste, si andranno ad appianare le differenze nei punti di congiunzione. In generale il processo di sculpting prevede la creazione di una geometria particolarmente densa, al seguito della quale, in base all'ambito di utilizzo della mesh, si procede con la fase di retopology. In Blender, un modo utile per la generazione della mesh high-poly consiste nell'utilizzare la funzione *dynamic*

Figura 4.34: Esempio di reference per il nano

topology, che, differentemente dal solo sculpting che modifica la mesh agendone solo sulla forma, permette di aggiungere o togliere dettaglio a parti della mesh tramite una tessellatura dinamica in relazione ad alcuni fattori, come ad esempio la distanza del modello rispetto alla vista che si sta utilizzando. Sebbene in teoria questo metodo permetta di ottenere una geometria non eccessivamente complessa se gestita alla perfezione, di fatto si ottiene una mesh che necessità praticamente sempre di una fase di retopology. Inoltre, da un punto di vista computazionale, la dynamic topology porta un particolare overhead, dovuto alla non uniformità della mesh. Un altro metodo recentemente introdotto in Blender prevede l'utilizzo della funzione di *Voxel Remesh*, che sfrutta la libreria OpenVDB per generare una geometria manifold (ossia una geometria di cui può essere effettuato l'unfold generando una geometria 2D, con tutte le normali direzionate nella stessa direzione) perfettamente distribuita. Questo metodo è particolarmente utile per la fase di *blocking*, ossia nella definizione delle proporzioni principali.

Nel caso in questione è stata utilizzata una terza via, ossia tramite l'utilizzo del potente modificatore *Multires Modifier*. Quest'ultimo permette di suddividere la mesh in modo simile a quanto accade con il *Subdivision Surface*, ma con la possibilità di modificare la versione suddivisa tramite sculpting. Inoltre, permette di differenziare il numero di suddivisioni in relazione all'ambiente di visualizzazione (Sculpt, Viewport e Rendering). Il tipico utilizzo di questo modificatore prevede l'utilizzo di una mesh di base a bassa risoluzione la cui topologia è composta solo da quads; non è possibile quindi utilizzare questa tecnica insieme alle altre presentate precedentemente. Solitamente si realizza in primis la mesh high poly tramite

la dynamic topology, si effettua il retopology della mesh generata e infine si utilizza il *Multires Modifier* o per effettuare il baking della versione high-poly su quella low-poly o per continuare con la fase di sculpting. Nel caso particolare, si è partiti da una mesh low-poly, generata tramite l'addon MBLab, che permette di creare in modo parametrico figure umane. Il motivo per il quale si è utilizzato questo addon è stato in primo luogo la possibilità di lavorare sin da subito con una mesh dotata di una buona topologia e inoltre ha permesso di ottenere un viso dalle proporzioni anatomiche corrette in maniera immediata. Tuttavia, la mesh generata è stata pesantemente modificata tramite sculpting al fine di ottenere il risultato desiderato e le proporzioni anatomiche sono state ovviamente accentuate.

Figura 4.35: Mesh di base generata tramite l'addon MBLab

Dei pennelli disponibili all'interno del workspace dedicato allo sculpting sono stati principalmente utilizzati i seguenti pennelli:

- *Grab*, permette di muovere un gruppo di vertici, in modo simile al proportional editing in edit mode;
- *Draw*, muove i vertici verso l'interno o l'esterno della mesh, in relazione alla normale media dei vertici;
- *Crease*, crea dei rientramenti o delle creste sulla superficie, "schiacciando" al tempo stesso i vertici adiacenti;
- *Clay*, simile al pennello Draw, ma con un'incidenza più morbida.

In particolare, l'obbiettivo era la ricreazione di un personaggio anziano, quindi era necessario ricreare già in questa fase alcune imperfezioni della pelle e soprattutto le rughe. In particolare, a questo fine è stato utilizzato il già citato pennello *Crease*. Inoltre, è stato utilizzato il pennello *Draw* insieme a una texture in scala di grigi (di fatto una height map), rappresentante le increspature tipiche delle rughe. Agendo sulla forza del pennello, è stato possibile ricrearle sulla mesh.

È stato poi modellato un cappello e una veste, partendo da alcune primitive (piani e cubi), per poi completare il lavoro sempre tramite sculpting. In questa fase è stato molto utile il pennello *Cloth*, che simula la fisica dei tessuti in maniera più semplice e immediata rispetto alla simulazione classica.

Figura 4.36: Mesh per il nano in seguito alla fase di sculpting

4.5.2 Baking, texturing e shading

Terminata la fase di modellazione, si passa quindi alla fase di texturing. Per fare questo, come primo passo era necessario effettuare il baking delle informazioni contenute nella mesh high-poly, su di un modello più gestibile per quanto riguarda il numero di vertici. Infatti, sebbene nel caso in questione questo non fosse estremamente necessario, in quanto il modello creato non doveva essere soggetto a deformazioni dovute ad animazione, la fase del baking è stata comunque portata a termine al fine di effettuare un texturing più efficiente e per fare in modo di ottenere un livello di dettaglio abbastanza elevato, senza necessariamente aumentare in maniera troppo elevata il numero di suddivisioni in Blender. Si è deciso infatti di iniziare la fase di definizione delle textures in Substance Painter, che richiede una versione low-poly e una high-poly al fine di ottenere un buon risultato. Prima però di effettuare l'esportazione dei modelli 3D relativi alle due versioni, bisogna effettuare l'UV mapping, ossia la traduzione della superficie da coordinate 3D in coordinate 2D tipiche delle classiche immagini bitmap. È stato scelto di approcciare questa fase utilizzando un nuovo workflow sempre più impiegato nelle fasi di texturing denominato UDIM (U DIMension), che effettua un offset automatico dei cosiddetti UV tiles (lo spazio generato per ogni UV creata), permettendo di assegnare diverse immagini a risoluzioni differenti per ognuno di essi (fig. 4.37). In questo modo, necessitando di un maggiore livello di dettaglio sul viso del nano rispetto al resto, sono stati assegnati due diversi tile per i due uv mapping generati, con una risoluzione delle textures rispettivamente di 4k e 2k (fig. 4.38).

Importate le due versioni del modello in Substance Painter, è stato effettuato il baking delle textures, utilizzando quasi tutte le mappe disponibili; il programma ne utilizza diverse, al fine di permettere un texturing più accurato, tenendo in conto delle informazioni della mesh high poly. In particolare, le mappe presenti sono:

- *Normal*: viene generata la *Tangent Space Normal Map*. Il calcolo delle normali

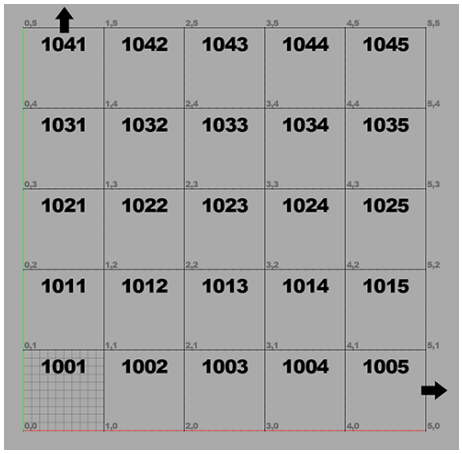

Figura 4.37: Rappresentazione degli udim tiles

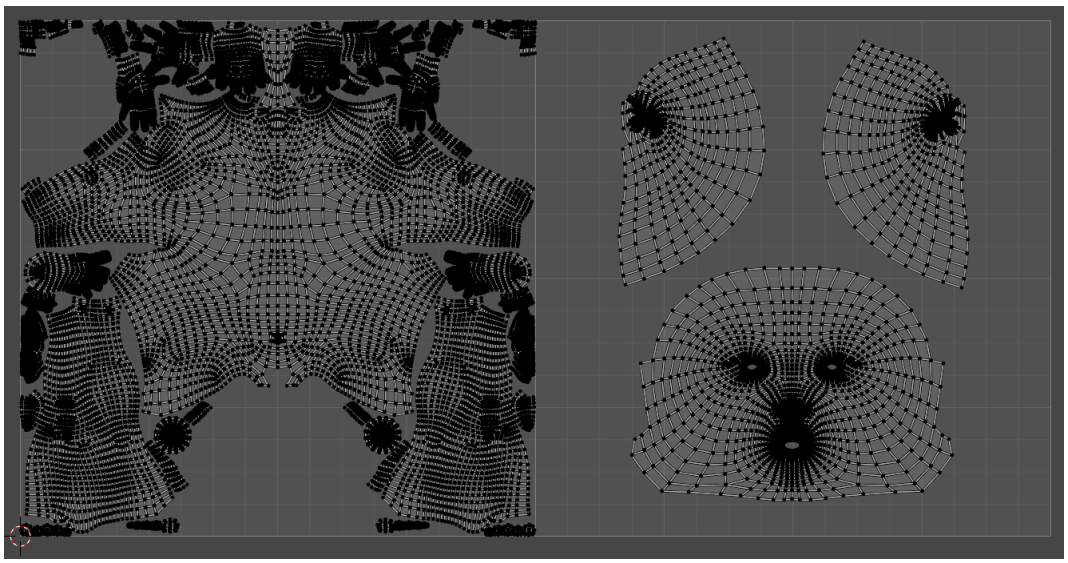

Figura 4.38: Risultato dell'uv unwrapping del modello 3D, su due differenti tiles

viene effettuato direttamente sulla superficie dell'oggetto, tenendo in conto dell'orientamento di ogni faccia (spazio tangente)[12].

- *World Space Normal*: differentemente dalla mappa precedente, in questo caso viene effettuato il calcolo delle normali è relativo all'intero oggetto, in base alle sue coordinate nel mondo;
- *ID*: per ogni materiale viene assegnato un ID, utile per effettuare alcune operazioni come il masking di una texture in relazione alle altre;
- *Ambient Occlusion*: viene generata una mappa che contiene le informazioni riguardanti le ombre relative all'ambiente;
- *Curvature*: mappa contenente le informazioni sulla curvatura della mesh, evidenziandone le increspature;

- *Position*: viene generata una mappa contenente tutte le posizioni relative a ogni vertice, ottenendo quindi informazioni sulla posizione dell'oggetto nel mondo;

- *Thickness*: vengono immagazzinate le informazioni relative allo spessore della mesh.

Figura 4.39: Schermata di Substance Painter durante la fase iniziale di texturing

In seguito al baking, sono stati aggiunti vari livelli nello stack, similmente a come si farebbe in Photoshop, utilizzando le mappe precedenti e aggiungendo dettagli come pori, alcune macchie, variazioni di colore in relazione all'ambient occlusion e al subsurface scattering (fig. 4.39).

Il lavoro fatto in Substance Painter è stato in realtà un punto di partenza, in quanto il resto è stato realizzato in Blender, aggiungendo dettagli a quanto fatto in precedenza, utilizzando uno stile di lavoro più improntato sul procedurale. Le textures create sono quindi state esportate e inserite nei vari canali del Principled BSDF all'interno dello Shader Editor di Blender. All'interno di quest'ultimo, sono state create diverse mappe, agendo su vari tipi di textures (come Voronoi e Noise), al fine di aggiungere alcuni particolari come nei, variazioni di colore sulla pelle, pori e vene. Il sistema di nodi ottenuto è particolarmente complesso e il numero di nodi elevato. Al fine di non appesantire la trattazione, in figura 4.40 ne viene mostrata solo una parte: come già accennato, le textures sono state utilizzate come punto di partenza, e in particolare in questo caso è stata utile la mappa relativa al diffuse color, input di due diversi nodi del tipo *Hue/Saturation/Value* che, combinati con alcune textures procedurali relative al rumore, hanno permesso di realizzare delle striature di colore violaceo sulla pelle a rappresentare i capillari e delle variazioni di colore sulla pelle in modo da non renderla perfettamente omogenea.

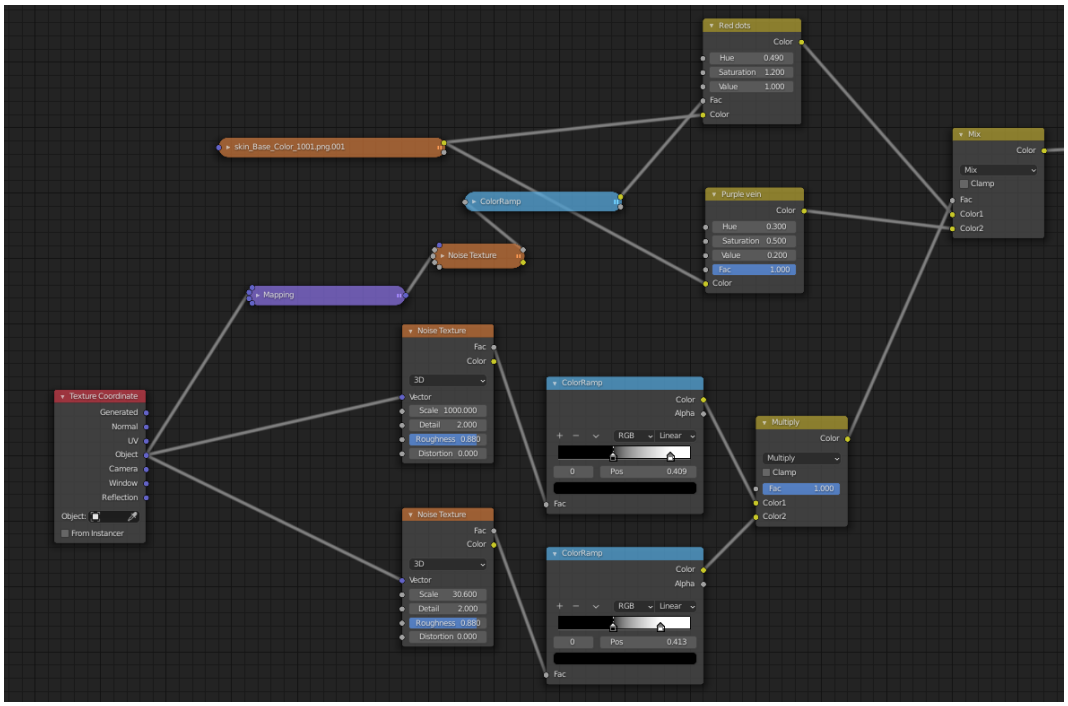

Figura 4.40: Parte del sistema a nodi utilizzato per il materiale della pelle

4.5.3 Hair grooming e realizzazione dello shader

Al personaggio creato era richiesta l'aggiunta di barba, sopracciglia e ciglia, anch'essa il più fotorealistica possibile. La modellazione dei capelli a fini realistici e il modo in cui questi vengono gestiti dal software è un compito particolarmente complesso e difficile da risolvere. Il motivo principale è dato dalla necessità di rappresentare un numero molto elevato di elementi, circa 100000 nel caso dei capelli, ognuno dei quali ha un diametro molto ridotto e non esiste un unico standard nella loro modellazione, in quanto possono essere visti sia come un numero molto alto di *strands* (fili), o come un unico volume. Inoltre, non siamo a conoscenza di alcune caratteristiche fisiche dei capelli reali, rendendo ancora più complesso riuscire a generare uno "schema di modellazione corretto"[10]. Nella maggior parte dei casi, i capelli sono considerati all'interno dei softwares come dei fili, ossia curve monodimensionali in uno spazio 3D. Questo accade anche nel caso di Blender, in cui i sistemi di tipo *Hair* sono una sottocategoria delle particelle e in particolare, sono considerate delle particelle la cui vita è di 100 frames e il cui cammino è stato calcolato in precedenza, generando dei segmenti, tramite interpolazione dei punti di controllo [5]. Il modo in cui i capelli (o la barba in questo caso) vengono posizionati è anch'esso soggetto a vari tipi di worflows. È possibile ad esempio, al fine di semplificare il lavoro, posizionare i capelli su una superficie 2D che viene mappata nello spazio 3D. Nel caso in questione, è stato scelto di non posizionare la barba direttamente sulla mesh del modello, ma utilizzare delle meshes ausiliari (superfici composte da piani estrusi), per i seguenti motivi:

- Data la natura del modello, creato utilizzando il *Multires Modifier*, è impossibile effettuare il passo di *styling* del sistema di *hair* in quanto il numero di suddivisioni è soggetto a cambiare in base al tipo di visualizzazione e il posizionamento delle particelle cambia in relazione alla geometria sottostante;

- È in ogni caso più semplice e più efficiente da un punto di vista computazionale lavorare con delle mesh più semplici;

- È possibile creare differenti sistemi particellari suddivisi per ogni mesh, rendendo più semplice la fase di *styling* in quanto si vanno a creare dei limiti ben definiti su dove questi devono posizionarsi. In questo modo, è stato utile una suddivisione logica dei sistemi particellari utilizzati, differenziandoli in base alle zone: ono stati infatti definiti due sistemi particellari per le sopracciglia, due per le ciglia, e tre per la barba, rispettivamente due per i lati e uno per il mento. In questo modo, anche la modifica di uno di essi non andava a inficiare il lavoro svolto sugli altri.

Sono stati quindi creati dei piani suddivisi ed estrusi, posizionati sulla superficie del modello utilizzando il *Shrinkwrap Modifier*, utilizzando un workflow simile a quello relativo alla retopology. Il numero di suddivisioni dei piani è stato quindi definito a priori e non più modificato. Una volta deciso il numero di particelle da emettere (nell'ordine di 1000), sono stati modificati alcuni parametri, come ad esempio *Hair Shape*, che va a definire il diametro delle meshes dei peli, sulla punta e sulla base, in modo da renderli realistici. Utilizzando il workspace relativo allo styling, *Particle Edit*, è stata definita una forma generale per ogni sistema particellare utilizzando i vari pennelli presenti (*Comb*, *Smooth*, *Add*, *Length*, *Puff* e *Cut*) e, al fine di aggiungere variazione al risultato, è stato utile selezionare solo alcune delle curve, agendo su una selezione randomica e effettuando lo styling separatamente per ognuna di queste. Gran parte della randomicità tipica di un sistema di questo tipo è stata ottenuta aggiungendo sul parametro *Children*. In questo modo, vengono aggiunte un gran numero di meshes aggiuntive per ogni curva *parent* e posizionate in un intorno di queste ultime. È possibile quindi agire sulla lunghezza delle mesh *figlie*, impostandone una lunghezza inferiore e un *threshold* che determina quante di queste ne verranno affette. Sono stati quindi utilizzati i sotto pannelli *Clump*, con un fattore positivo in modo da aggregare tra loro le curve, formandone dei *ciuffi* (fig. 4.41 [1]), e *Roughness*, che definisce una variazione nella forma delle curve, rendendo random la posizione dei punti di controllo delle curve, agendo sia sul parametro *random* (quantità di randomicità) e *size* (ampiezza della variazione in relazione alla lunghezza complessiva della curva).

È stato quindi creato un materiale da attribuire al sistema particellare. In Cycles è ora possibile utilizzare uno shader apposito, *Principled Hair BSDF*, che permette molto facilmente di realizzare uno shading credibile per i capelli in quanto fisicamente corretto. Non è tuttavia disponibile in EEVEE, motore di rendering usato in questo caso, quindi è stato necessario creare uno shader *ad hoc*, ricercando anche in questo

[1] Fonte: Blender Documentation

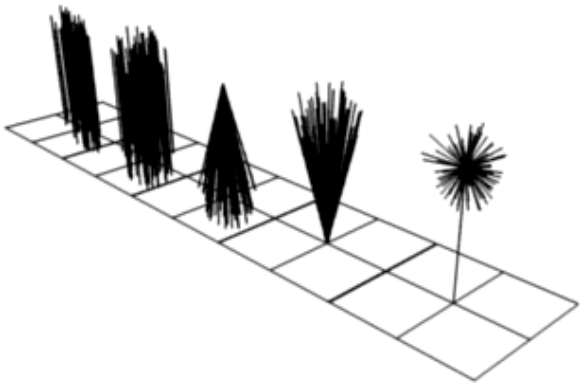

Figura 4.41: Immagine che esemplifica il concetto di clump

un risultato quanto più accurato possibile, anche tenendo in conto dei limiti tipici del motore di rendering utilizzato. La realizzazione dello shader ha previsto la

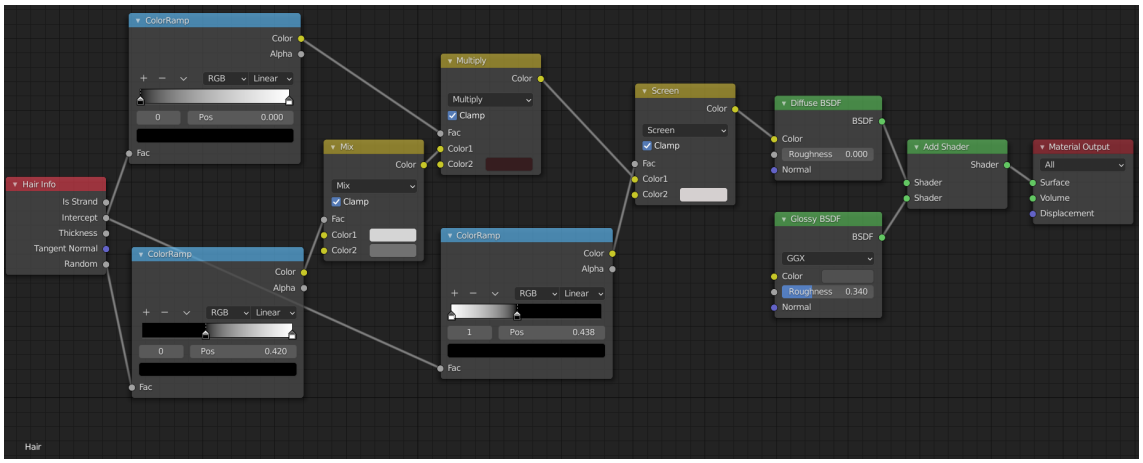

Figura 4.42: Schema a nodi per lo shader di barba, ciglia e sopracciglia

creazione di una serie di mappe, utilizzando il nodo *Hair Info*, che permette di accedere alle informazioni del sistema particellare *hair*. Come si può vedere in figura 4.42, sono stati in particolar modo usati i canali *Intercept* e *Random*: il primo permette di accedere alla posizione lungo le singole curve, ponendo in output un valore di 1 per la punta e 0 per la radice; il secondo seleziona in maniera randomica le curve del sistema, permettendo di ottenere variazioni di colore in maniera semplice. Utilizzando questi due nodi, sono state infatti realizzate alcune variazioni di colore, dal grigio chiaro al grigio scuro e aggiungendo delle tonalità rossicce. Tramite il nodo *Add Shader*, è stato aggiunto uno shader a bassa intensità (ossia con un grigio abbastanza scuro) di tipo *Glossy*, in modo da aggiungere lucentezza al risultato.

In figura 4.43 è mostrato il render finale del modello.

Figura 4.43: Render finale del modello del nano

4.6 Scena della sala del trono

L'ultima scena da realizzare consisteva nella rappresentazione della sala del trono della regina. L'idea alla base prevedeva di ricreare lo spazio circostante al trono

con su seduta la regina, intorno al quale sono appesi un gran numero di specchi che lentamente ruotano su sé stessi. Anche in questo caso è stata centrale la ricerca di references; una celebre sala del trono è quella di *Minas Tirith*, capitale del regno di Gondor nell'universo rappresentato ne *Il Signore degli Anelli* (fig. 4.44), utilizzata principalmente per quanto riguarda alcune scelte decorative; altre references sono state trovate tramite il sito *Artstation*, che permette di accedere a un gran numero di opere d'arte digitali. Un esempio è mostrato in fig. 4.45, in particolar modo per quanto riguarda il posizionamento delle vetrate.

Proprio la necessità di aggiungere delle grandi vetrate alle spalle della regina, insieme alla presenza di numerosi specchi, hanno reso la scena abbastanza complessa da gestire in EEVEE per quanto riguarda la fase di lighting. Nel seguito verranno esposte alcune scelte tecniche fatte in questo senso e finalizzate a una resa naturale della luce all'interno della scena.

Figura 4.44: Minas Tirith - *Il Signore degli Anelli*

Figura 4.45: Esempio di reference utilizzata (credits: Edward Barons)

4.6.1 Modellazione della sala del trono

Il primo passo è stata la modellazione della sala del trono. Questa fase non è stata particolarmente complessa, in quanto l'ambiente da realizzare non era eccessivamente impegnativo e sono state quindi utilizzate tecniche di base di *box modelling*. È stato tuttavia utile il ricorso al metodo di modellazione tramite *booleans*, che consiste nell'utilizzare il modificatore *Boolean Modifier* su di una mesh di base, utilizzando altre meshes di supporto per effettuare le operazione di unione, differenza e intersezione con quest'ultima. L'utilizzo di questo modificatore produce risultati migliori e predicibili su mesh di tipo *manifold*, in caso contrario possono verificarsi degli errori dovuti al risolutore matematico che Blender utilizza. Il punto di forza di un workflow di questo tipo è che è possibile cambiare il risultato finale semplicemente modificando (ad esempio traslando e scalando) la mesh utilizzata per l'intersezione, senza andare ad agire direttamente sulla mesh originale. È possibile infatti ottenere lo stesso risultato agendo direttamente sulla geometria originale, ma si perderebbe la versatilità che questo metodo offre. Il modificatore *boolean* è stato utilizzato in particolare per il posizionamento delle vetrate e per la creazione del punto luce al di sopra del trono.

Per quanto riguarda gli altri elementi presenti, sono state realizzate delle scalinate tramite il modificatore *Array* applicato su un semplice cubo scalato sull'asse X, utilizzando un offset unitario per la ripetizione sugli assi Y e Z. In più sono state create le inferriate delle vetrate, così come per il punto luce, tramite il modificatore *Wireframe*. Il funzionamento di quest'ultimo si basa sull'effettuare un'iterazione per ogni faccia della mesh, trasformando ogni lato di questa in un poligono a quattro facce, permettendo di modificarne lo spessore tramite il parametro *Thickness*. Sono stati realizzati degli ornamenti sulle vetrate tramite le curve, e in particolare *Nurbs Path*, con un metodo simile a quello descritto nella sezione 4.3.3 relativa all'ampolla. Anche il design che si è cercato di riprodurre, riprende lo stesso stile utilizzato in precedenza, in ottica di coerenza tra le varie scene in relazione al concetto di look development, come descritto nella sezione 2.1 del capitolo relativo. I modelli invece relativi al trono e agli specchi sono stati realizzati da un altro membro del team (Edoardo Toso), principalmente tramite tecniche di sculpting e utilizzando il modificatore *Displacement* con l'ausilio di textures. In molti degli assets creati, è stato aggiunto il modificatore *Bevel*. Nell'ambito del box modeling finalizzato al fotorealismo, questo modificatore è particolarmente utile, in quanto permette di "smussare" i lati della mesh che soddisfano alcune condizioni, come ad esempio l'angolo che si crea tra le facce e di cui il lato è condiviso o tramite un valore inserito manualmente sui lati interessati (*Bevel weight*). È infatti praticamente impossibile vedere nel mondo reale degli oggetti perfettamente spigolosi ed è importante riportare questo fattore nella grafica 3D. È possibile ottenere un risultato molto simile o utilizzando il modificatore *Subdivision Surface* insieme ad alcune edge loops di controllo, o tramite il *bevel* manuale, che aggiunge geometria alla mesh; in entrambi i casi, tuttavia, si perde la possibilità di modificare facilmente la mesh ottenuta e ritornare ad uno stato precedente.

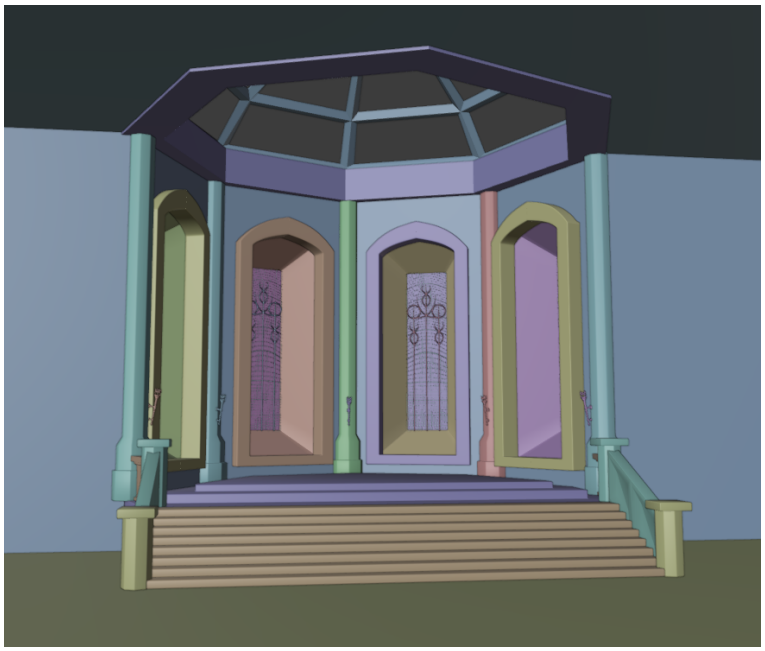

Figura 4.46: Risultato della fase di modellazione

4.6.2 Shading e texturing

La fase di shading e texturing è ovviamente preceduta da quella di UV unwrapping. La maggior parte dei modelli realizzati ha una geometria semplice, quindi questa fase è stata abbastanza immediata. Un esempio di un tipico uv unwrapping si può osservare per le colonne, cha hanno un unico *edge seam*, che serve all'algoritmo di unwrap per effettuare l'appiattimento da modello 3D su di una superficie 2D, posto su di un *edge loop* che le attraversa completamente in maniera perpendicolare rispetto alla base e selezionato in modo tale da non essere visibile alla camera. Un procedimento simile è stato fatto per gli altri modelli. L'applicazione delle textures è stato effettuato utilizzando il solito nodo *Principled BSDF*, al quale sono state collegate immagini, suddivise sui vari canali, relative al cemento, a mattoni e a marmo, alcune in parte modificate per quanto riguarda il canale *Diffuse*, per ottenere i colori desiderati, o il canale *Roughness*, al fine di variare il modo in cui la luce viene riflessa sulle superfici, creandone delle imperfezioni.

Un materiale particolarmente interessante è stato quello relativo alle vetrate. Quest'ultimo doveva rappresentare una sorta di mosaico, simile alle tipiche fantasie utilizzate nelle chiese ma più irregolare. In fig. 4.47 ne si può osservare un esempio, in cui si può intuire di che tipo di shader era necessario creare: deve essere caratterizzato da diversi colori traslucidi, permettendo alla luce di attraversarli e riflettere i colori sull'ambiente circostante.

La definizione di questo materiale è quindi molto legata alle varie scelte di lighting esposte in seguito. Data la somiglianza del rappresentazione a mosaico rispetto al diagramma di tipo Voronoi, è stato fondamentale il nodo omonimo all'interno dello Shader Editor. Il diagramma in questione viene costruito, nel caso più semplice

Figura 4.47: Il ciclo di vetrate della Sainte-Chapelle di Parigi (fonte: Wikipedia)

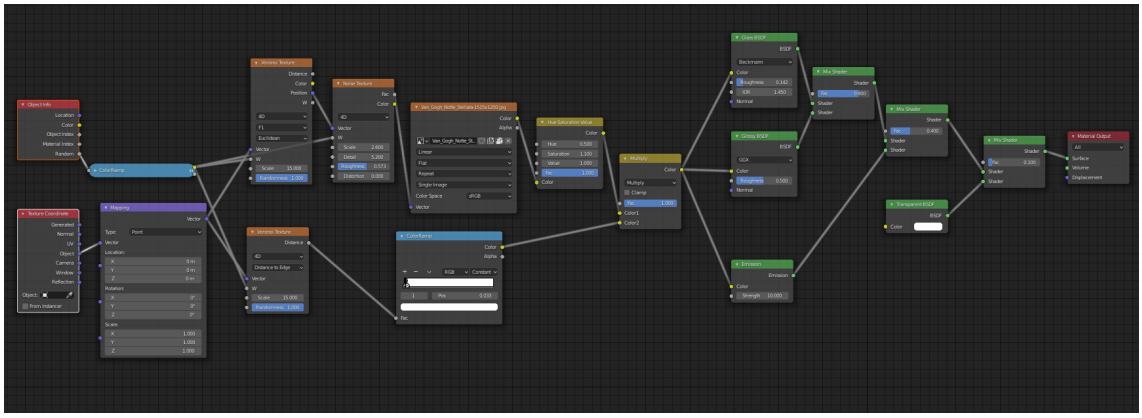

Figura 4.48: Sistema dei nodi per il materiale relativo alle vetrate

di un spazio bidimensionale, definendo un insieme di punti S su di un piano e suddividendolo in tante regioni (o celle) $V(p)$ quanti sono i punti p definiti, in cui $p \in S$, in modo tale che tutti i punti definiti all'interno della regione siano il più possibile vicini a p rispetto che agli altri punti. La definizione matematica è data dalla formula seguente [14]:

$$R_k = \{x \in X | d(x, P_k) \leq d(x, P_j), \forall j \neq k\}$$

Come mostrato in figura 4.48 in cui è possibile vedere l'intero sistema dei nodi utilizzato, sono stati utilizzati due nodi di tipo Voronoi, esattamente identici per i parametri riguardanti la randomicità e la scale ma differenti nella tipologia di output: il primo utilizza la distanza euclidea per il calcolo delle distanze dell'insieme dei punti definiti nella regione *V(p)* e ne viene utilizzato l'output relativo alla posizione come input del socket relativo al *Vector* di un nodo *Noise*.

Il nodo Voronoi serve a ricreare la fantasia del mosaico, utilizzando la formula precedentemente citata; il nodo Noise, invece, il cui output è utilizzato come input nel socket *Vector* di una texture, modifica l'immagine, rendondola impossibile da decifrare e ricollegare all'originale. La texture in questione è *La Notte Stellata* di Van Gogh ed è stata decisa in base allo schema di colori che si intendeva rappresentare (principalmente variazioni di blu e di verde). Il secondo nodo Voronoi è stato invece impostato in modo tale da mostrare la distanza dei punti dai lati in una mappa in scala di grigi. Modificando questa mappa in una *Color Ramp* e rendendo netta la differenza tra i lati e l'interno della regione, è stato possibile ricreare delle divisioni più nette tra le varie regioni, combinando il risultato del primo Voronoi e del noise con il secondo mediante metodo *Multiply*. Inoltre, essendoci quattro diverse vetrate, al fine di variarne la composizione, è stato utilizzato il canale *Random* del nodo *Object Info* che permette di definire un una valore differente in base all'istanziazione dell'oggetto. Questo valore è stato utile a pilotare il valore *W* dei vari nodi Voronoi e Noise, che rappresenta il *seed* del rumore creato.

Figura 4.49: Render in viewport di una delle vetrate

La parte successiva dello shading è stato invece relativa alla scelta vera e propria dei materiali, combinando tra loro nodi di tipo *Glossy*, *Glass*, *Transparent* e *Emission*. In particolare questi ultimi servono a simulare un passaggio di luce in EEVEE, difficilmente ottenibile utilizzando solamente il nodo *Glass*. Anche in questo caso, così come è stato fatto nel materiale del vetro relativo all'ampolla, nel pannello relativo alle impostazioni del materiale, è stato impostata la trasparenza di tipo *Alpha Blend* ed è stata inserita la spunta *Screen Space Reflection*.

4.6.3 Lighting in EEVEE

L'illuminazione di una scena di questo tipo è particolarmente impegnativa in un ambiente di rendering come quello di EEVEE, in quanto difficile da rendere in modo realistico, essendo un luogo chiuso in cui devono essere presenti un gran numero di luci e colori provenienti principalmente dalle vetrate. Questo è stato complicato dalla presenza di materiali altamente riflessivi come gli specchi, le cui riflessioni realistiche sono difficili da rendere in un motore di rendering basato sul raster e non sul ray-tracing.

L'illuminazione ha previsto la presenza di una HDRI, rappresentante l'interno di una cattedrale, in modo da ottenere una scelta di colori e luci coerente con la scena in questione. È stata inoltre aggiunta una *Sun light*, in modo da creare delle ombre più nette provenienti dalla parte superiore della sala del trono. Una fase fondamentale è stata relativa alla preparazione della scena al baking delle luci indirette. In questo caso, più che nelle altre scene che comunque hanno beneficiato di questo workflow, era necessario aggiungere il contributo delle luci indirette; in Blender, queste ultime sono considerate tutte le luci che provengono da una sorgente luminosa, sebbene non sia totalmente corretto da un punto di vista fisico [4]. In questo senso, anche la luce proveniente dalla HDRI viene considerata luce indiretta. In EEVEE, la luce indiretta consta di due componenti, relativi alla luce diffusa e a quella speculare e le informazioni derivate da entrambi vengono memorizzate in una cache statica, ossia che non può essere aggiornata ad ogni frame. Sono stati quindi aggiunti due oggetti, chiamati *Reflection Probe* e *Irradiance Volume*, che sono responsabili della memorizzazione delle informazioni relative rispettivamente al carattere speculare e a quello di diffusione delle luce: ad ogni punto definito all'interno di questi oggetti e denominato *probe*, viene calcolato il comportamento della luce. Sono stati posizionati in modo da racchiudere la scena rappresentata nella sua interezza. Sono stati inoltre aggiunti dei *Reflection Planes*, uno per ogni specchio; questi non contribuiscono al baking della luce indiretta, in quanto vengono memorizzate solo le informazioni indipendenti dall'osservatore (ossia dalla camera). Il baking della luce indiretta permette di rappresentare il comportamento della luce nelle sue riflessioni; tuttavia, EEVEE non permette di mostrare il comportamento delle luce fatta passare attraverso un oggetto semi-trasparente (come appunto le vetrate), quindi rappresentare i colori derivanti dalla rifrazione della luce nel materiale. Come capita spesso quindi, nel momento in cui bisogna fare i conti con un motore di rendering di questo tipo, bisogna cercare dei compromessi che in qualche modo simulino il comportamento reale della luce. Una possibile alternativa al baking, è quella di posizionare manualmente delle luci in modo tale da simulare il comportamento della luce indiretta, come descritto in "RenderMan, Theory and Practice" [2]. Un espediente trovato, è stato quello di aggiungere tre luci di tipo *Spot* per ogni vetrata, ognuna di un colore diverso scelto in base ai colori delle vetrate stesse, in modo che, attraversando il materiale delle vetrate, producessero dei risultati il più possibile naturale.

4.6.4 Animazione procedurale

Come accennato, è stato richiesto di fare in modo che gli specchi ruotassero lentamente su sé stessi, mentre la camera si avvicinava alla regina seduta sul treno. Dato il numero abbastanza elevato di specchi, l'animazione di questi è stata affrontata in modo procedurale. In particolare, è stato inserito un keyframe al frame iniziale per ogni specchio, definendone la rotazione di partenza. Successivamente, agendo su di un singolo specchio, è stato aggiunto un modificatore F-Curve di tipo *Noise* all'interno del *Graph Editor*, applicato alla curva di rotazione sull'asse Z. I modificatori F-Curve permettono di ottenere un workflow non distruttivo riguardante le anima-

zioni, modificando i parametri della curva in ogni momento. Inizialmente, dato che è stato inserito un solo keyframe, la curva iniziale era di tipo costante, quindi nessuna rotazione sull'asse considerato. Con l'aggiunta del modificatore, la curva è stata modificata in modo da rendere random la rotazione sull'asse Z. È stata impostato il metodo *Replace*, che permette di aggiungere rumore nel range -0.5 e +0.5 e sono stati settati gli altri parametri (*Scale*, *Strenght*, *Offset*, *Phase*) in modo da ottenere il risultato desiderato. È stato poi sufficiente selezionare tutti gli oggetti relativi agli specchi e copiare i dati relativi all'animazione dal primo selezionato, tramite *Copy Animation Data*. In seguito, è stato modificato l'offset per ognuno dei modificatori applicati, in modo da ottenere un'animazione differente per ogni oggetto, lasciando invariati gli altri parametri.

4.6.5 Depth map tramite Deep Learning da una singola immagine RGB

In un ambiente 3D il render della depth map o z-depth è di immediata realizzazione, in quanto intrinsecamente calcolata durante le fasi di rendering. L'utilizzo della depth map è fondamentale nella realizzazione di environments in quanto utile a ricreare una serie di effetti in fase di compositing, come l'attenuazione atmosferica o il bokeh. Nel caso in questione era necessario utilizzare una depth map per la foto 2D relativa alla regina seduta sul trono, e utilizzarla come texture per il displacement, realizzando quindi una mesh 3D in grado di reagire alla luce. Senza questa informazione di profondità il movimento di camera avrebbe facilmente rivelato la natura 2D del piano in cui è stata posizionata la texture relativa alla regina.

La realizzazione di depth map da una singola immagine RGB è un task molto complesso all'interno della computer vision e impossibile da realizzare prima dell'avvento degli strumenti odierni. In particolare, sono stati scritti vari paper che trattano l'argomento e lo risolvono tramite il deep learning. Sono state testate varie soluzioni, che si appoggiano a tipologie specifiche di reti neurali, quali le Convolutional Neural Network o le GAN. In molti casi tuttavia oltre la singola immagine è necessario avere informazioni aggiuntive, quali la lunghezza focale utilizzata o una seconda immagine relativa a una vista stereoscopica. Una possibile implementazione è descritta in *High Quality Monocular Depth Estimation via Transfer Learning*[1]. Con transfer learning si intende l'utilizzo di una rete addestrata precedentemente su un task simile, quindi utilizzata per un secondo training. In questo modo è possibile ottimizzare il tempo usato per il calcolo per lo specifico task che si vuole ottenere. Utilizzando vari dataset come Kitti, ImageNet e NYUv2, in questo caso è stata realizzata una rete utilizzando un modello di *autoencoder* di tipo convoluzionale, con l'ausilio delle skip connections. Come input dell'encoder sono state utilizzate delle immagini RGB, il cui output è stato confrontato tramite una loss-function con il *ground truth* dell'immagine, ossia la depth map associata.

Esistono vari applicativi che permettono di sfruttare il machine learning per questo tipo di obbiettivi. Nel caso in questione, la foto della regina è stata processata

tramite un tool online chiamato *Runway ML*, che offre molteplici strumenti basati sul machine learning e deep learning per la computer vision.

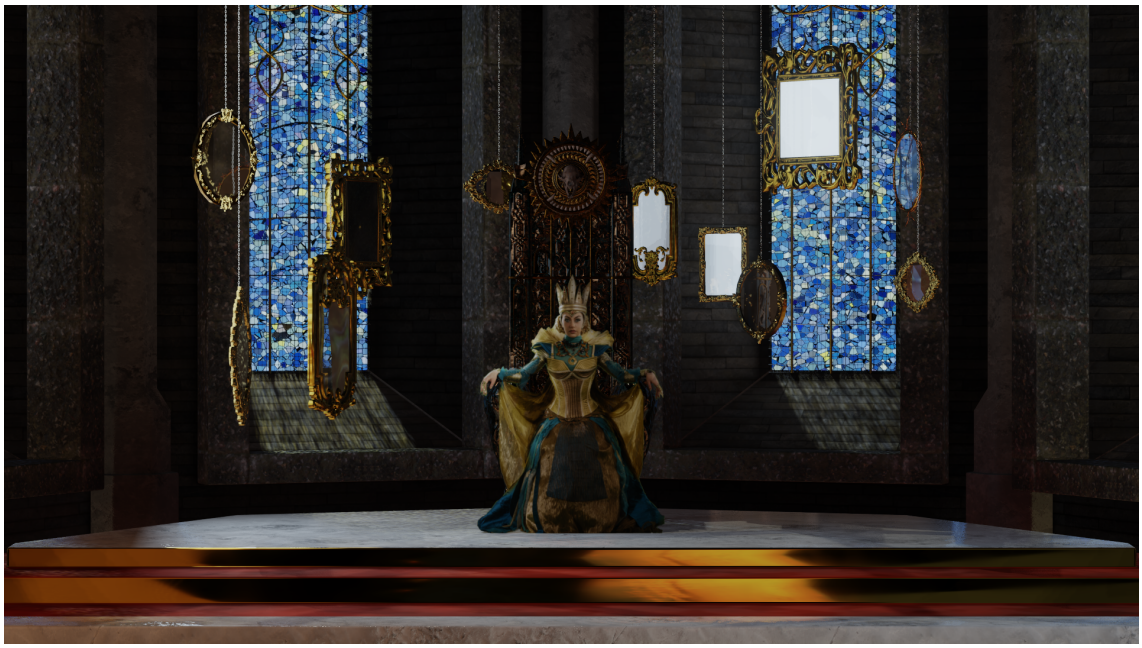

Figura 4.50: Render della sala del trono

Conclusioni

Il lavoro di tesi condotto ha permesso di approfondire e vedere con mano i vari step relativi alla realizzazione di un opening, comprendendone la pipeline e cercando di ottimizzarla quando possibile. Il lavoro di look development ha inoltre permesso di effettuare una serie di scelte volte a dare la possibilità a un team di piccole dimensioni di realizzare un cortometraggio in computer animation: questo è stato possibile grazie a una ricerca di un workflow di lavoro basato sulla modellazione e il texturing procedurale, così come l'utilizzo di scripting in Python, permettendo di ridurre il tempo di realizzazione degli assets necessari e focalizzandosi maggiormente sulla composizione della scena.

È ovviamente possibile in questo senso continuare a ricercare e applicare nuove tecnologie e metodi di ottimizzazione: i nuovi strumenti della computer grafica permettono di ottenere grandi risultati un tempo impossibili per team che non fossero diretti da grandi case di produzione. In particolare, la modellazione procedurale così come il deep learning applicato alla computer vision sono strumenti da sperimentare e riuscire ad affiancare alle pipeline di produzione esistenti.

Bibliografia

[1] Ibraheem Alhashim e Peter Wonka. *High Quality Monocular Depth Estimation via Transfer Learning*. 2019. arXiv: 1812.11941 [cs.CV].

[2] Dana Batali et al. "RenderMan, Theory and Practice". In: *SIGGRAPH 2003* 9 (2003), pp. 31–72.

[3] Zeeshan Bhatti et al. "Be-Educated: Multimedia Learning through 3D Animation". In: *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND EMERGING TECHNOLOGIES* 1 (dic. 2017), pp. 13–22.

[4] Blender Documentatio. *Indirect Lighting*. Last updated on 09/15/2021. URL: https://docs.blender.org/manual/en/latest/render/eevee/render_settings/indirect_lighting.html.

[5] Blender Documentation. *Hair*. Last updated on 09/12/2021. URL: https://docs.blender.org/manual/en/latest/physics/particles/hair/introduction.html.

[6] Bill Fleming. *Advanced 3d Photorealism Techniques*. John Wiley & Sons Inc, 1999.

[7] Melis Inceer. "An Analysis of the Opening Credit Sequence in Film". In: *CUREJ: CollegeUndergraduate Research Electronic Journal* (2008). URL: http://repository.upenn.edu/curej/65.

[8] Katie Ingram. *A brief history of TV shows' opening credit sequences*. 2016. URL: https://theweek.com/articles/632836/brief-history-tv-shows-opening-credit-sequences.

[9] David Lettier. *Screen Space Reflection*. 2019. URL: https://lettier.github.io/3d-game-shaders-for-beginners/screen-space-reflection.html.

[10] Kelly Ward et al. "A Survey on Hair Modeling: Styling, Simulation, and Rendering". In: *IEEE Transactions on Visualization and Computer Graphics* 13 (2007). DOI: 10.1109/TVCG.2007.30.

[11] Wikipedia. *Global Illumination*. Last edited on 13 April 2021. URL: https://en.wikipedia.org/wiki/Global_illumination.

[12] Wikipedia. *Normal Mapping*. Last edited on 19 August 2021. URL: https://en.wikipedia.org/wiki/Normal_mapping.

[13] Wikipedia. *Shader*. Last edited on 22 August 2021. URL: https://en.wikipedia.org/wiki/Shader.

[14] Wikipedia. *Voronoi Diagram*. Last edited on 3 September 2021. URL: https://en.wikipedia.org/wiki/Voronoi_diagram.

[15] H. Yilmaza, Murat Yakar e Ferruh Yildiz. "DIGITAL PHOTOGRAMMETRY IN OBTAINING OF 3D MODEL DATA OF IRREGULAR SMALL OBJECTS". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XXXVII (2008). URL: https://www.isprs.org/proceedings/XXXVII/congress/3b_pdf/23.pdf.