Alma Mater Studiorum · Università di Bologna

CAMPUS DI CESENA
SCUOLA DI SCIENZE
Corso di Laurea in Scienze e Tecnologie Informatiche

GAME PROGRAMMING: Grafica 3D

Tesi di Laurea in Programmazione

Relatore:
Dott. Mirko Ravaioli

Presentata da:
Tommaso Pantaloni

Sessione III
Anno Accademico 2012-2013

Indice

Elenco delle figure

Introduzione

In questa tesi si vuole descrivere il processo che porta alla creazione di tutta la parte grafica di un videogioco 3D. Questo verrà fatto mediante la realizzazione di un breve gioco esplorativo in prima persona. Inoltre si è scelto di supportare Oculus Rift, un headset di realtà virtuale ancora in fase di prototipazione ma già disponibile agli sviluppatori. Il giocatore si troverà dentro una catacomba alla ricerca di un prezioso artefatto, dovendo stare però attento alle trappole disseminate per l'ambiente.

Il tema in questione è stato deciso prendendo specialmente in considerazione le forte immersione resa possibile dall'Oculus Rift. Infatti il senso di ansia e insicurezza dato dall'esplorare un luogo pericoloso e sconosciuto è enormemente amplificato dal sentirsi effettivamente "dentro" il videogioco.

Nella tesi verranno trattate alcune delle tecniche utilizzate per creare della grafica 3D utilizzabile all'interno di un motore grafico real-time, nello specifico Unity3D. I principali strumenti utilizzati saranno Blender per quanto riguarda la modellazione e Unity3D per quanto riguarda illuminazione e shading.

Essendo il gioco in prima persona (e quindi visto dagli occhi del protagonista) e per di più con supporto per la realtà virtuale, tutta la grafica dovrà mantenere uno standard di realismo piuttosto elevato in modo da non rovinare l'immersione del giocatore con oggetti mal realizzati o fuori luogo.

La piattaforma sul quale il progetto sarà giocabile è il PC. I sistemi operativi supportati saranno Microsoft Windows, Linux, e Mac OS X. Questo perchè Unity3D permette di creare videogiochi multipiattaforma con facilità. Non saranno però supportate le piattaforme mobile in quanto non dispongono della "potenza grafica" necessaria a gestire rendering real-time di buon livello.

Il titolo scelto per il gioco è "Catacomb" in quanto ambientato all'interno di una catacomba.

Capitolo 1

Grafica 3D con Blender e Unity3D

In questo capitolo si descriveranno gli strumenti e le tecniche che verranno poi utilizzate nella creazione del progetto.

Per primo si tratterà di Blender e della modellazione 3D e poi di Unity3D e dello shading e del rendering real-time. Infine si darà un breve descrizione di cos'è Oculus Rift.

1.1 Che cos'è Blender?

Blender è un software per *Computer Graphics* Open Source utilizzato per creare film, effetti visuali, applicazioni interattive 3D e videogiochi. Tra le sue molte funzioni si possono trovare modellazione 3D, texturing, rigging, skinning, simulazioni di particelle, sculpting, animazione, rendering e compositing. Nel corso della tesi si tratteranno solo una piccola parte di queste in quanto ognuna è di per se stessa un tema piuttosto complicato. Si è deciso di utilizzare questo software piuttosto che altri principalmente perchè completamente gratuito e OpenSource (ha una licenza GNU GPL) ma anche perchè universalmente riconosciuto come estremamente valido. Infatti viene utilizzato da professionisti di tutto il mondo sia per videogiochi che per film e pubblicità e la sua diffusione cresce sempre più.

Blender è stato originariamente sviluppato da uno studio di animazione olandese chiamato *Neo Geo* come loro applicazione interna per *Computer Graphics*. Il suo autore principale fu Ton Roosendaal che nel 1998 fondò *NaN (Not a Number Technologies)* per continuare lo sviluppo del programma e distribuirlo. Questa azienda fallì nel 2002 e Ton Roosendal decise di rilasciare il codice sorgente e di adottare per Blender una licenza GNU GPL. Inoltre

fondò la *Blender Foundation*, della quale è tuttora il presidente, per guidare lo sviluppo futuro del software.

Passiamo ora a descrivere più nel dettaglio come e per cosa è stato utilizzato Blender in "Catacomb" .

1.1.1 Introduzione al 3D Modeling

Quando si va a realizzare la grafica 3D per un videogioco una buona parte del lavoro in genere consiste nel creare i modelli 3D per tutti quanti gli oggetti che andranno a comporre le varie scene che lo costituiscono. Il processo che porta alla creazione di un modello 3D è detta *3D Modeling*. La sua difficoltà ovviamente è data dalla complessità dell'oggetto che si vuole ricreare: modellare un bicchiere ad esempio non richiede più di mezz'ora, mentre modellare un personaggio può richiedere benissimo delle settimane.

In generale il *3D Modeling* può essere definito come il processo di sviluppare una rappresentazione matematica di una qualsiasi superficie o oggetto tridimensionali.

Ci sono principalmente due modi di rappresentare un modello:

- **Polygonal modeling:** il modello è rappresentato attraverso dei punti nello spazio 3D (vertici) connessi da dei lati in modo da formare una *Mesh* poligonale. Questo tipo di rappresentazione è il più diffuso, soprattutto nell'ambito dei videogiochi e del real-time rendering in generale, in quanto modelli di questo tipo possono essere renderizzati molto velocemente dall'hardware moderno.

- **Curve modeling:** il modello è rappresentato da delle superfici definite da curve, le quali sono influenzate da dei punti di controllo.

Nel corso della tesi verrà trattata solamente la modellazione poligonale.

In Blender quando si vuole creare un nuovo oggetto la prima cosa da fare è scegliere una Mesh di base dalla quale partire ed inserirla nella scena. Si può scegliere tra: piano, cubo, cerchio, sfera, icosaedro, cilindro, cono, griglia, toroide e testa di scimmia. La scelta va fatta ovviamente pensando a quale delle Mesh proposte sia più facile da modificare per ottenere il risultato finale. In generale ci si troverà molto spesso a partire da un cubo anche quando si devono fare modelli molto complessi come una testa in quanto relativamente facile da adattare ad un vasto numero di esigenze.

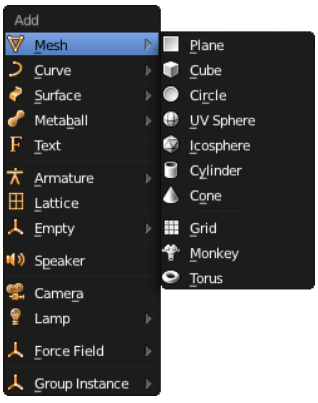


Figura 1.1: Menu di scelta della Mesh

Una volta che si ha una base dalla quale partire si procede a spostare e aggiungere geometria fino ad ottenere il risultato voluto. Quando si vogliono modificare vertici, lati o facce già esistenti gli strumenti principali a nostra disposizione sono:

- **Translate:** permette di traslare tutto o parte del modello all'interno dello spazio 3D
- **Rotate:** permette di ruotare tutto o parte del modello
- **Scale:** permette di scalare tutto o parte del modello

Quando si vogliono aggiungere nuovi vertici, lati o facce gli strumenti più utili sono in genere:

- **Extrude:** permette di estrudere una o più facce.
- **Loop Cut:** permette di aggiungere un nuovo *Edge Loop*. Per *Edge Loop* si intende un insieme di lati interconnessi attraverso una superficie dove in genere l'ultimo che si ricongiunge al primo.
- **Fill:** permette di creare una nuova faccia all'interno dei lati selezionati o di creare un lato tra due vertici selezionati.
- **Delete:** permette di eliminare uno o più vertici, lati o facce.

Altro importante strumento che Blender mette a disposizione è costituito dai *Modifiers*, un inseme di modificatori che possono essere applicati ad un qualsiasi oggetto in modo da conferirgli delle proprietà particolari. Uno molto utile è il *Mirror Modifier* che duplica a specchio la geometria del modello su uno o più assi in modo da poter lavorare più agevolmente su oggetti contraddistinti da due metà identiche.

Un buon utilizzo di quanto messo a disposizione da Blender ci permetterà di creare un un modello tridimensionale di tutto quello che vogliamo. Per rendere più facile questo processo ed ottenere un risultato più realistico è molto utile utilizzare delle immagini di riferimento in modo da essere sempre sicuri di stare rispettando la forma e le proporzioni dell'oggetto che si vuole avere.

Quanto descritto finora verrà poi esaminato più nel dettaglio quando si tratterà nello specifico della creazione di alcuni dei modelli che compongono la grafica del videogioco.

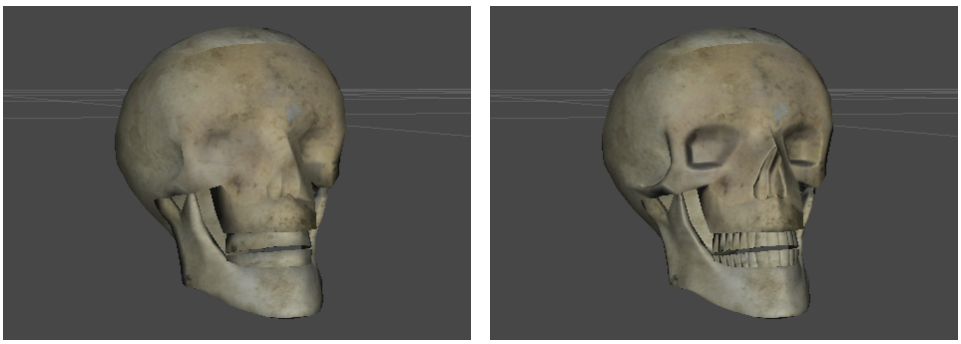
1.1.2 Introduzione al Texture Mapping

Avere un buon modello 3D non è tutto quello che serve per ottenere dei risultati validi. Non avrebbe senso avere dei modelli bellissimi ma con una superficie piatta e monocolore. Per modificare il colore della superficie e come essa reagisce alla luce ci si affida al *Texture Mapping* e allo *Shading*. Per *Texture Mapping* o *Texturing* si intende quel processo che porta ad applicare una immagine su una superficie 3D in modo da aggiungervi dettaglio o colore.

(a) Senza Texture **(b)** Con Texture

Figura 1.2: Esempio di Texture Mapping su una colonna

Oltre all'applicazione della classica Texture di colore ci si troverà molto spesso a volerne utilizzare alcune con scopi differenti, in generale per modificare il comportamento della luce. Alcuni esempi sono il *Normal Mapping*, il *Bump Mapping* e il *Displacement Mapping*. In tutti e tre i casi lo scopo è quello di simulare piccoli incavi o sporgenze su di una superficie senza effettivamente doverli avere come poligoni aggiuntivi nella Mesh.



(a) Senza Normal Map **(b)** Con Normal Map

Figura 1.3: Esempio di Normal Mapping su un teschio

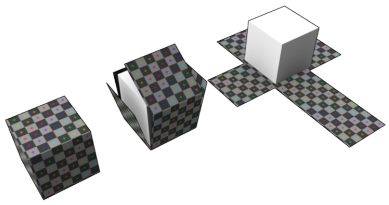


Figura 1.4: Rappresentazione dell'UV Mapping di un cubo

Prima di poter applicare una qualsiasi Texture su di un modello è però necessario eseguire un processo chiamato *UV Mapping*. Questo porta a "mappare" tutta la superficie 3D del modello su di un piano 2D. Questo è fondamentale perchè mentre la Mesh è tridimensionale l'immagine che vi si vuole applicare è strettamente bidimensionale. La complessità di questo processo ovviamente dipende da quanto la geometria del modello è articolata. Inoltre è importante tenere presente come e da che angolazioni l'oggetto verrà normalmente visualizzato. Questo perchè in generale piccole imperfezioni e sporcature saranno presenti nel risultato finale, l'importante è decidere saggiamente dove posizionarle. Queste imperfezioni derivano dal fatto che il modello dovrà necessariamente essere diviso in regioni separate sul piano 2D e quindi saranno presenti dei tagli. Ovviamente è possibile creare delle Texture tali per cui i tagli siano praticamente invisibili ma, a seconda della bontà dell'UV Map che si ha, questo può essere estremamente complicato.

Ci sono vari modi per ottenere delle Texture valide da utilizzare. Per quanto riguarda le Texture per il colore in genere si utilizza un qualche tipo di fotografia o una loro composizione in modo da dare all'oggetto un aspetto quanto più reale possibile. Ovviamente questo potrebbe non essere vero se quello che si cerca di ottenere non è il realismo ma qualcosa di diverso, come ad esempio una qualche ambientazione astratta. Per quanto riguarda le altre Texture (Normal Map, Displacement Map, ecc.) di solito si procede creando un altro modello tridimensionale dell'oggetto al quale si sta lavorando con tutti i dettagli aggiuntivi effettivamente modellati. Le differenze tra i due modelli (e quindi i dettagli) verranno poi impressi su di una Texture che sarà ovviamente applicata sull'oggetto più semplice. In questo modo si riesce a far sembrare un oggetto con pochi poligoni relativamente complesso.

1.2 Che cos'è Unity3D?

Unity3D è un *Game Engine* che negli ultimi anni ha avuto una enorme diffusione, specialmente tra i piccoli team di sviluppo. Un *Game Engine* è un software appositamente progettato per sviluppare videogiochi. Unity3D è composto principalmente da un motore grafico 3D, un motore fisico, componenti per l'audio, componenti per l'animazione, un IDE grafico per la gestione dei progetti e permette di scrivere i proprio script utilizzando JavaScript,

C# o Boo. Caratteristica fondamentale è che dà la possibilità di compilare il proprio progetto per una notevole varietà di piattaforme: Windows, Linux, Mac OSX, iOS, Android, Windows Phone 8, PlayStation 3, Xbox 360, Wii, BlackBerry 10, Unity Web Player e Adobe Flash. L'IDE è disponibile per Windows e Mac OSX. La licenza è proprietaria ma è comunque disponibile una versione gratuita senza alcune funzioni destinata ad uso privato o a piccole aziende.

1.2.1 Introduzione allo shading

Di tutte le funzioni di cui Unity3D dispone, solamente il motore grafico è interessante ai fini della tesi. In particolare si andranno a creare dei materiali da applicare agli oggetti. I materiali definiscono come l'oggetto reagisce alla luce e sono caratterizzati da una serie di proprietà che possono essere ad esempio delle Texture o dei valori e da uno *Shader*. Questo è essenzialmente una serie di istruzioni che definiscono come le proprietà del materiale vengono usate, assieme alle informazioni sulla luce, per dare all'oggetto un suo aspetto specifico e una sua illuminazione.

Unity3D mette a disposizione fin da subito una serie di Shader di default in modo da poter creare senza troppa fatica dei materiali per gli oggetti più comuni. Se si vuole fare qualcosa di più particolare ci si troverà però costretti a scriverne di propri in uno dei tre modi che Unity mette a disposizione: come *surface shaders*, come *vetex and fragment shaders* o come *fixed function shaders*. Nel corso della tesi si scriveranno solamente dei surface shader. Questi sono appositamente pensati per facilitare l'interazione con la luce, cosa che può diventare notevolmente complessa. La maggior parte del codice è scritto in un linguaggio chiamato *Cg*, abbreviazione di *C for Graphics*, unito ad una parte scritta in un linguaggio specifico di Unity chiamato *ShaderLab*.

1.3 Che cos'è Oculus Rift?

Oculus Rift è un headset di realtà virtuale ancora in fase di prototipazione ma del quale sono già disponibili dei kit per gli sviluppatori. Esso crea delle immagini 3D stereoscopiche in un modo tale da dare un senso di profondità molto simile alla realtà. E' inoltre dotato di alcuni sensori che rilevano la rotazione della testa in modo da dare la possibilità a chi lo indossa di guardarsi attorno nel mondo virtuale come farebbe nel mondo reale. Oculus Rift è prodotto da una azienda americana chiamata *OculusVR* e la prima versione commerciale è attesa entro la fine del 2014.

Ottima è la sua integrazione con alcuni dei Game Engine attualmente sul mercato come Unity3D e Unreal Engine. Durante lo sviluppo di "Catacomb" è infatti bastato utilizzare poco più di quanto messo a disposizione di default dal kit di sviluppo di Oculus Rift per ottenere un ottimo effetto 3D e una coinvolgente navigazione all'interno dell'ambiente.

Capitolo 2

Progettazione

In questo capitolo si parlerà prima della progettazione del videogioco in generale e poi nello specifico di quella grafica. Il primo tipo consiste fondamentalmente nel decidere le meccaniche di gioco che costituiscono quello che è noto come *Gameplay* e, se necessario, nello scrivere la storia. Il secondo tipo di progettazione (quella grafica) invece consiste nel decidere nel dettaglio come dovrà essere l'aspetto del videogioco prima di buttarsi nel realizzare la grafica 3D.

Queste due fasi di progettazione sono entrambe estremamente importanti (soprattutto se non si lavora da soli) in quanto danno una direzione chiara allo sviluppo e rendono il prodotto finale uniforme e coerente al suo scopo. Inoltre definendo bene cosa si vuole raggiungere si minimizzano le possibili incomprensioni tra i vari membri del team di sviluppo e si evitano tempi morti in cui qualcuno si trova a non sapere cosa fare. La fase di progettazione è nota anche come *pre-produzione*.

2.1 Progettazione del videogioco

Quando si decide di sviluppare un videogioco il primo passo è ovviamente avere una idea. Questa può riguardare una particolare meccanica di gioco che si piacerebbe avere, una storia che si ritiene valida oppure anche semplicemente uno stile visuale molto particolare e artistico. Avere una buona idea di partenza è ovviamente fondamentale ma altrettanto importante è la fase seguente, quella della progettazione. Infatti una idea, per quanto possa essere bella, rimane appunto una idea e quindi necessariamente molto vaga e parziale.

Ci si dovrà quindi soffermare a ragionare in maniera sufficientemente approfondita su tutti gli aspetti che dovranno caratterizzare l'esperienza video-

ludica che si vuole andare a creare. Questo compito è generalmente svolto dal cosiddetto *Game Designer*, figura professionale che all'interno del team potrà poi sia andare a ricoprire altri ruoli, come quello di programmatore o di grafico, sia dedicarsi a tempo pieno a questa attività.

Si descrivono ora le principali tematiche delle quali un buon Game Designer si deve occupare all'atto di progettare un nuovo videogioco.

2.1.1 Gameplay

Per *Gameplay* si intendono l'insieme di meccaniche di gioco che caratterizzano l'azione del videogioco. In generale quest'ultimo ricadrà sempre in un genere di Gameplay già noto, visto che fare qualcosa di veramente innovativo è pressoché impossibile. Questi sono innumerevoli e generalmente raggruppati in alcune macro categorie: Action, Action-Adventure, Adventure, Role-Playing, Simulation, Strategy, Sports, Puzzle. Ognuna di queste avrà diversi sottogenere: un gioco Action ad esempio può essere un "Fighting Game", un "Platform Game", uno "Shooter" o molto altro ancora. Come si può immaginare i generi noti sono veramente tantissimi ed è appunto per questo che è praticamente impossibile creare qualcosa che non ricada in nessuno.

(a) Super Mario Bros. **(b)** DooM **(c)** StarCraft

Figura 2.1: Tre famosi giochi di generi molto differenti: (a)Platform, (b)First Person Shooter e (c)Real-Time Strategy

"Catacomb" può essere definito come una breve *avventura in prima persona*. L'essere una avventura è denotato dal fatto che mancano combattimenti o comunque sia della vera azione. Normalmente i giochi "Adventure" hanno però una forte presenza di elementi narrativi, tanto da farne in genere il loro elemento predominante. Nel caso di "Catacomb" non è però così in quanto la storia è pressoché inesistente: il giocatore non ha infatti neanche una vera conoscenza del perchè si ritrovi ad esplorare delle pericolose catacombe. Il gioco può essere invece definito in prima persona perchè l'azione viene vissuta dagli occhi del protagonista, in modo da fare immedesimare il più possibile chi gioca con il proprio personaggio.

Inquadrando il proprio gioco in un genere si ha già una vaga idea delle meccaniche che saranno presenti al suo interno, è però poi necessario andarle a definire nel dettaglio. Il gameplay di "Catacomb" è molto semplice: si esplora l'ambiente in prima persona dovendo stare attenti a quello che si ha attorno e cercando di trovare una croce dorata nascosta all'interno del livello. Ci saranno trappole mortali evitabili solamente prestando attenzione ai dettagli.

Le azioni eseguibili dal giocatore saranno quindi principalmente due: muoversi ed interagire con oggetti. Ovviamente non sarà possibile interagire con tutto ma solamente con alcuni oggetti di particolare interesse, ad esempio dei pulsanti.

Le meccaniche del progetto che si è scelto di sviluppare possono sembrare molto semplici, ed in effetti lo sono. Questo perchè l'obiettivo che "Catacomb" si pone non è tanto quello di offrire un gameplay particolarmente innovativo ma quello di immergere il più possibile il giocatore nell'esplorazione di un ambiente sconosciuto e ostile. Questa scelta è stata fatta soprattutto prendendo in considerazione le immense possibilità di immersione date dall'utilizzo dell'Oculus Rift e quindi della realtà virtuale.

2.1.2 Storia

La storia nel gioco è pressoché assente. Tutto quello che si sa è che il protagonista è un esploratore alla ricerca di una croce dorata nascosta all'interno della catacomba che si sta esplorando.

La scelta di non dare peso alla storia è dovuta dal fatto che essa richiede che al giocatore vengano veicolate in qualche modo notevoli quantità di informazioni. Questo è normalmente fatto in due modi: mediante del testo scritto e/o mediante del testo doppiato. Il primo approccio è reso difficoltoso dalla bassa risoluzione dello schermo dell'Oculus Rift con conseguente difficoltà di lettura del giocatore mentre il secondo dall'impossibilità tecnica del registrare dei dialoghi di qualità adeguatamente alta da non rendere più difficile l'immersione.

2.1.3 Stile visuale

Molta importanza si è deciso di dare invece all'impatto visivo del gioco. Gli ambienti proposti dovranno quindi rendere al meglio il senso di ignoto e di pericolosità propri dell'esperienza che si vuole proporre al giocatore. Inoltre per quanto riguarda lo stile si è scelto di cercare di avere quanto più realismo possibile. Questa decisione è stata dettata dall'Oculus Rift in quanto si è pensato che l'esperienza di realtà virtuale avrebbe giovato notevolmente da

una veste grafica similare al mondo reale. Delle visuali del gioco si parlerà più diffusamente nella sezione seguente.

2.2 Progettazione della scena

Prima di iniziare a produrre qualsiasi tipo di asset grafico per il progetto che si vuole sviluppare è fondamentale avere ben chiaro il risultato che si vuole ottenere e le sensazioni che deve trasmettere. Procedere "alla cieca" facendo oggetto dopo oggetto senza avere chiara una visione d'inseme del contesto porta in genere a dei risultati al massimo mediocri.

Si procede ora a descrivere tutto quello che è stato fatto prima di iniziare a produrre i modelli 3D con relativi materiali.

2.2.1 Ricerca immagini di riferimento

Il primo passo è in genere cercare delle immagini (di solito fotografie) di ambienti simili a quello che si vuole ricreare fino a che non se ne trovano di soddisfacenti. Nello specifico di "Catacomb" si sono andate a ricercare delle foto degli interni di alcune catacombe cristiane e pagane.

Figura 2.2: Tre immagini di riferimento usate per "Catacomb"

Queste immagini dovranno "guidare" il lavoro del 3D artist durante tutto il corso del progetto in modo da avere degli oggetti uniformi nello stile, evitando quindi la presenza di ambienti fuori luogo che diminuirebbero drasticamente l'immersione del giocatore. Infatti è proprio questo che si deve tenere sempre a mente quando si crea la grafica di un videogioco, l'immersione. Non avrebbe minimamente senso avere ad esempio una bellissima porta moderna all'interno di una catacomba.

In "Catacomb" si è deciso di tenere uno stile molto buio e cupo con teschi ed ossa in quantità. I muri dovranno essere per la maggior parte in terra con solo alcune stanze in muratura. Inoltre ci sarà una totale assenza di fonti di illuminazione se non per la torcia tenuta in mano dal giocatore. Tutto questo

per cercare di incutere in chi gioca un certo senso di ansia e di paura mentre esplora l'ambiente non sapendo a cosa andrà in contro proseguendo.

2.2.2 Creazione di una piantina

Una volta che si sa l'effetto che si vuole si può procedere a progettare la disposizione delle varie stanze e degli oggetti principali in esse contenuti, producendo così una piantina della catacomba.

In "Catacomb" si è deciso di fare tre stanze unite da dei corridoi, uno dei quali sarà bloccato da una frana e porterà quindi ad un vicolo cieco. La prima sala sarà quella in cui si trova il giocatore all'inizio della sua breve avventura. E' caratterizzata dall'essere piuttosto ampia e completamente scavata nella terra, sorretta al centro da delle colonne in muratura parzialmente ricoperte da intonaco. Ci saranno poi diverse tombe, sempre scavate, nelle quali si potranno vedere ossa e scheletri. Queste saranno disposte sia a terra che sui muri. Altra caratteristica della stanza è l'ingresso, costituito da una parete in muratura ricoperta di intonaco e da una grata in metallo. Fuori dalla grata si potrà vedere una scalinata piuttosto alta e un bel cielo notturno che inonda l'ambiente di una flebile luce bluastra.

Da questa sala partirà un breve corridoio che porta ad una piccola camera circolare. Questa avrà come carattere distintivo il pavimento ricoperto di ossa e teschi disposti circolarmente in maniera molto ordinata. I muri saranno invece di mattoni. Altra peculiarità della stanza è quella di essere la prima a contenere una trappola: non appena il giocatore sarà entrato tutte le uscite verranno bloccate da delle barre in ferro e il soffitto comincerà a scendere, con degli spuntoni metallici che lentamente spuntano da dei suoi fori. Se si vorrà evitare di rimanere schiacciati bisognerà prestare attenzione al muro e premere un mattone particolarmente sporgente che funge da "interruttore" del meccanismo.

Fatto questo la discesa del soffitto verrà quindi bloccata e le uscite saranno di nuovo aperte. permettendo così di procedere lungo un altro corridoio. Questo conterrà un bivio con a destra un vicolo cieco e a sinistra l'ultima sala delle catacombe. Qui ci saranno una serie di altari in pietra disposti a semicerchio verso il fondo. Su di essi si potranno notare delle tavolette, sempre in pietra, con sopra incisi dei simboli. Inoltre ai lati dell'ingresso sulle pareti si avranno sei pulsanti, ognuno contraddistinto da un diverso segno. Sotto di essi si potranno notare una serie di buchi dentro i quali saranno nascosti degli spuntoni. Questi usciranno dalla parete uccidendo il giocatore se esso premerà il bottone sbagliato. Il pavimento è ricoperto di piastrelle in pietra, una delle quali è rialzata e con sopra inciso un simbolo. Questa è in realtà un pilastro "incassato" nel terreno e si alzerà quando verrà premuto il

giusto bottone tra quelli presenti nella stanza. Questo pilastro avrà un piccola concavità all'interno della quale si troverà una croce dorata. Questa potrà essere raccolta, azione che determinerà il compimento della propria missione e la fine del gioco. Il pulsante giusto da premere è quello contraddistinto dallo stesso simbolo che può essere visto sopra la mattonella/pilastro.

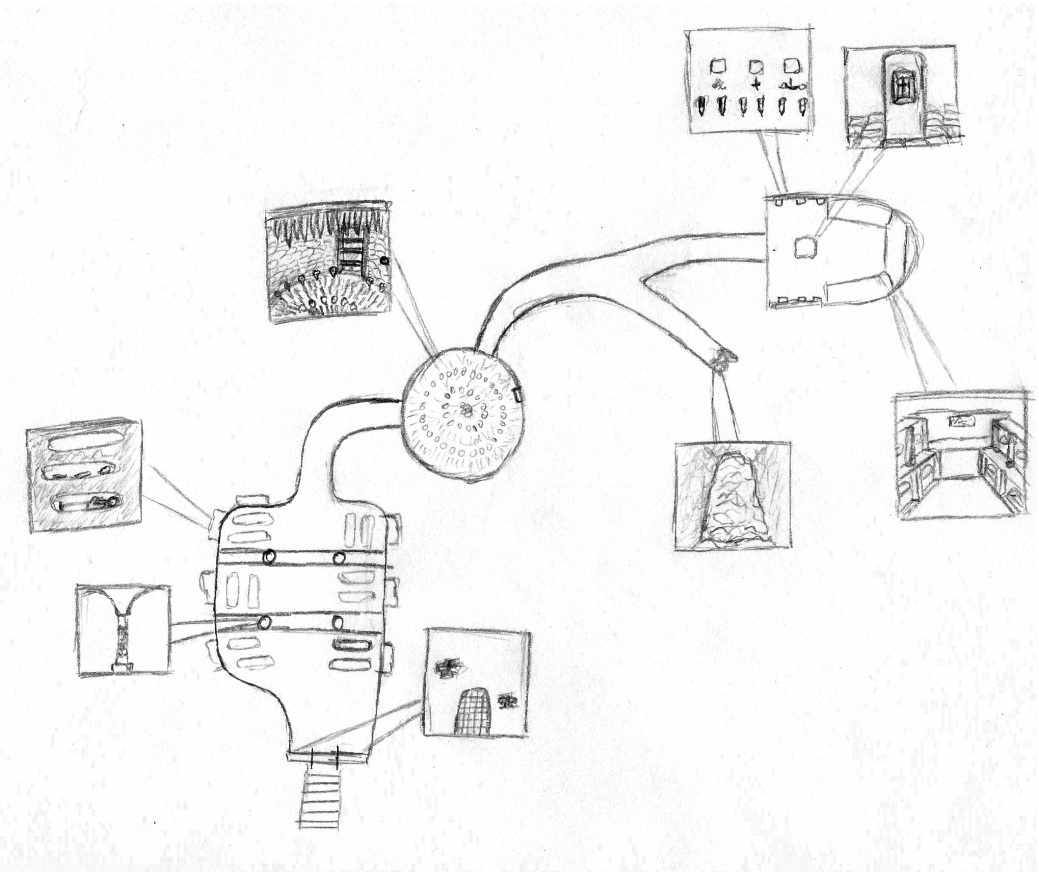


Figura 2.3: Uno schizzo dell'ambiente di "Catacomb"

2.3 Studio della piattaforma target e dei suoi limiti

Quando si crea un videogioco (e soprattutto la sua veste grafica) è fondamentale tenere a mente qual'è la piattaforma sul quale lo si vorrà eseguire. Questo perchè non tutti i dispositivi sono equamente potenti e di conseguenza potrebbe essere necessario limitare la complessità del proprio lavoro. Ad esempio

nel mondo mobile i modelli tridimensionali devono necessariamente avere un contenuto numero di poligoni e l'illuminazione e i materiali essere non troppo complessi. Basti pensare che su iOS per avere un framerate accettabile bisognerebbe non superare i 7000 vertici sullo schermo contemporaneamente mentre su un calcolatore con una buona GPU si possono tranquillamente arrivare ad avere *milioni* di poligoni.

Nel caso di "Catacomb" si è scelto di avere come target il PC. Questo è in generale di gran lunga il dispositivo più facile su cui lavorare visto che oramai l'hardware ha raggiunto standard di prestazioni molto elevati. Inoltre visto che alcune tecniche che si andranno ad utilizzare per aumentare il realismo sono computazionalmente molto impegnative (soprattutto per la GPU) si è deciso di tenere principalmente in considerazione computer di fascia medio-alta. In questo modo si può avere un buon margine di libertà quando si vanno a creare i veri modelli 3D anche se, come vedremo nel prossimo capitolo, non tutto è permesso e si devono adottare delle tecniche per mantenere buoni livelli di dettaglio senza intaccare sulla performance del videogioco.

2.4 Accorgimenti specifici per Oculus Rift

Figura 2.4: Un Oculus Rift

Il poter presentare il videogioco attraverso un mezzo potente come l'Oculus Rift (e quindi la realtà virtuale) è sicuramente un fattore molto positivo. Per sfruttare al meglio le sue potenzialità è necessario stare molto attenti ad alcuni aspetti della grafica tridimensionale. Prima di tutto è fondamentale la dimensione degli oggetti: infatti è fondamentale che tutto quello che va a comporre la scena sia di grandezza verosimile. Sembra a prima vista un compito banale e già di per se stesso necessario alla creazione di qualsiasi videogioco 3D ma purtroppo non è così. Quando ci si guarda attorno nella realtà virtuale l'effetto è del tutto similare al mondo reale e di conseguenza l'ambiente è percepito in maniera molto differente rispetto alla classica visualizzazione attraverso uno schermo. Quello che sembra della grandezza giusta su un monitor non è detto che non risulti sproporzionato una volta che si entra nella VR. Spesso infatti se si lavora senza controllare frequentemente il risultato con l'Oculus Rift ci si ritroverà ad avere oggetti sproporzionatamente grandi e si dovrà quindi procedere a ridurli di dimensioni.

Altra cosa da tenere a mente è lo schermo di cui Oculus Rift è dotato. Questo ha una risoluzione piuttosto bassa e quindi mettere oggetti molto piccoli nel mondo potrebbe risultare quasi inutile in quanto essi potrebbero risultare irriconoscibili. Altra nota sullo schermo è che i colori non sono ottimi e spesso le tonalità viste sul proprio monitor mentre si costruisce il videogioco potrebbero risultare poi notevolmente differenti.

Ultima nota è che la realtà virtuale può causare disorientamenti e vertigini in alcuni soggetti. E' per questo utile fare in modo che il giocatore non possa muoversi troppo velocemente in modo da minimizzare eventuali effetti negativi.

Capitolo 3

Creazione degli asset grafici

In questo capitolo si descriverà in maniera dettagliata il processo pratico di implementazione delle varie componenti grafiche che andranno a definire le visuali del videogioco. Prima si tratterà dei modelli tridimensionali fatti utilizzando Blender e poi di shader, materiali e illuminazione in Unity3D. Inoltre verranno trattate tecniche per ottenere particolari effetti quali la tassellazione e la stereoscopia.

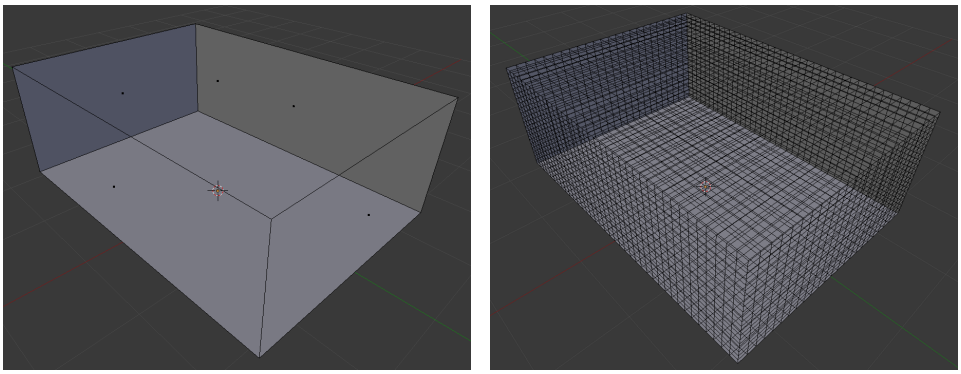
3.1 Modeling di oggetti semplici

Per prima cosa si è scelto di modellare gli oggetti più semplici quali le stanze ed il loro arredo. Questo per fare in modo di avere nel minor tempo possibile almeno una base sulla quale lavorare per implementare effettivamente le meccaniche di gioco e anche per avere delle prime impressioni sull'ambiente che si è progettato.

3.1.1 L'ingresso

Per creare la mesh della prima stanza si è deciso di partire da un cubo, deformandolo poi per dargli circa le proporzioni volute. Ora che si ha una base dalla quale partire è necessario procedere ad aggiungere tutti i dettagli desiderati: bisognerà fare in modo che la forma sia molto più arrotondata e conforme alla piantina che si è fatta e si dovranno inserire le varie tombe scavate.

Il primo passo è aggiungere più vertici in modo da avere molta più libertà nella deformazione del modello. Per far questo si sono inseriti innumerevoli *Edge Loops* su tutta la superficie dell'oggetto fino ad avere una distribuzione uniforme delle facce.



(a) Senza suddivisioni **(b)** Con suddivisioni

Figura 3.1: Il cubo di partenza deformato

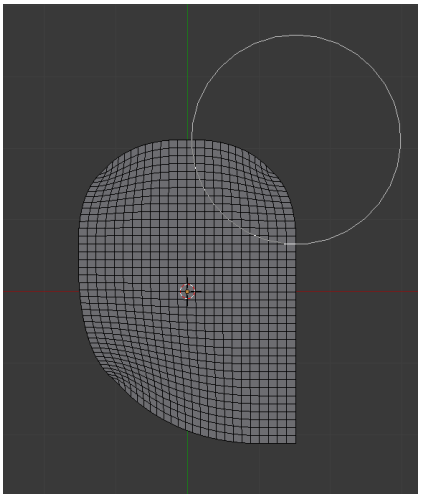
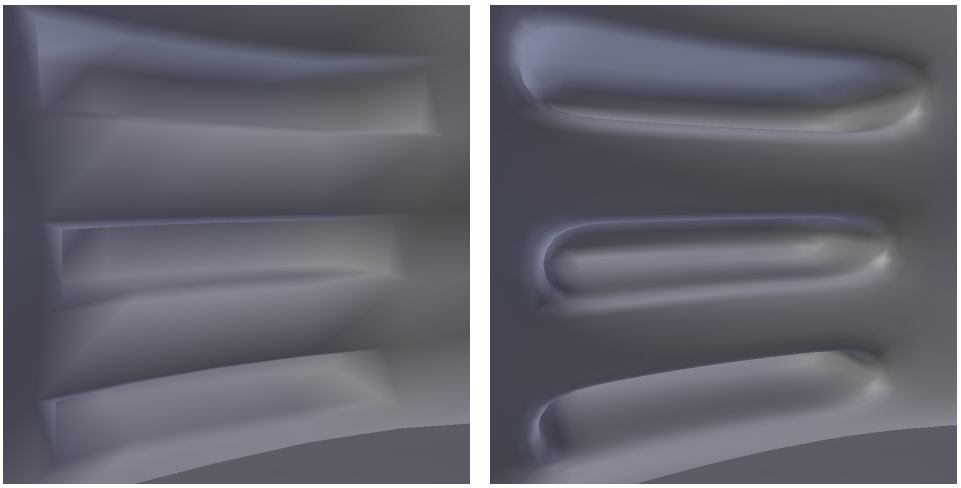


Figura 3.2: Arrotandamento degli spigoli

A questo punto si può procedere a rendere il tutto molto meno squadrato. Bisogna ricordarsi che quello che si sta cercando di ricreare è una stanza scavata nella terra e quindi totalmente priva di angoli vivi. Per ottenere il risultato voluto basta prendere i lati che si trovano agli angoli della mesh e spostarli verso l'interno utilizzando il *Proportional Editing Falloff*. Questa opzione fa si che la geometria vicina a quella che si sta modificando segua le trasformazioni proporzionalmente alla distanza permettendoci così di avere in maniera rapida e semplice delle superfici arrotondate. Senza Proportional Editing Falloff si sarebbe dovuto andare a modificare lato per lato fino ad ottenere una curva soddisfacente. Altra cosa da fare per avere un ambiente più realistico è il rendere meno piatti pavimento, soffitto e muri. Questo risultato può essere ottenuto in maniera estremamente semplice con il Proportianl Editing Falloff selezionando alcune facce e spostandole fino ad ottenere un aspetto soddisfacente.

Una volta che si sono arrotondati tutti gli angoli si può procedere a "scavare" le tombe. Il procedimento è molto semplice: basta selezionare le facce dove si vuole che il loculo venga collocato ed estruderle verso l'esterno della stanza. Sarà poi possibile aggiungere degli edge loop addizionali così da poter arrotondare un po' la forma. Non bisogna però esagerare in quanto il prossimo passo è quello di aggiungere un modificatore chiamato *Subdivi-*

sion Surface che ha il compito di aggiungere altre facce in modo da rendere l'oggetto più "smussato".



(a) Senza modificatori **(b)** Con Subdivision Surface

Figura 3.3: I loculi scavati nel muro

Ultima cosa da fare prima di passare ai modelli dell'entrata e delle colonne è quella di eliminare alcune facce per fare i buchi rappresentati ingresso e uscita della stanza. Questo compito è estremamente facile: basta selezionare le facce che si vogliono cancellare e rimuoverle. Potrebbe poi essere necessario arrotondare a mano il profilo del buco in modo da dargli esattamente l'aspetto desiderato. La base della prima camera si può ora considerare conclusa.

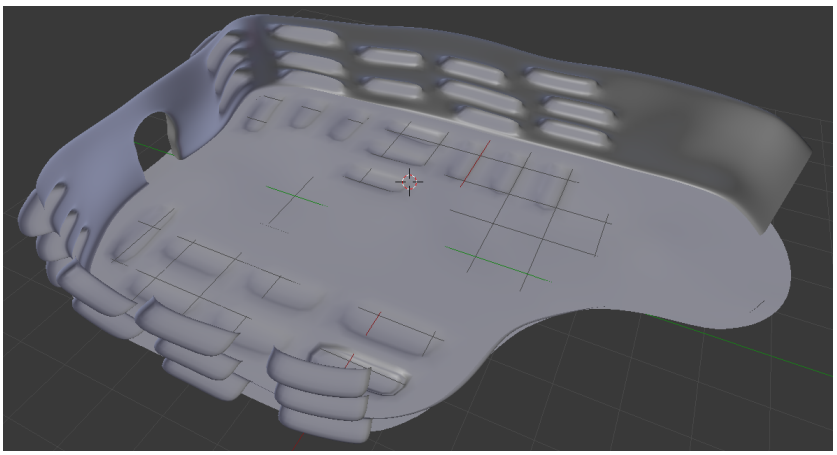


Figura 3.4: Il modello tridimensionale della prima stanza

Si passa ora alle colonne. Queste dovranno essere disposte in due file da due, unite a coppie da degli archi. Per ottenere questo risultato è conveniente partire, come per la stanza, da un semplice cubo. A questo si aggiungeranno due edge loop in modo da avere una faccia quadrata alla base. Questa verrà estrusa verso il basso in modo da formare la colonna. Si possono inoltre eliminare le facce disposte sopra e ai lati del blocco superiore.

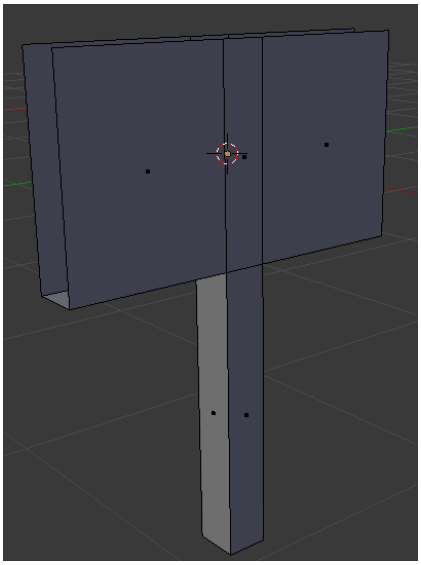


Figura 3.5: Estrusione della colonna

Si possono ora inserire nuovi edge loop alla mesh in modo da dare tutti i dettagli aggiuntivi che si vogliono. Facendo un buon uso di traslazioni, scalature e del Proportional Editing Falloff si riuscirà velocemente ad ottenere esattamente la forma desiderata. Cosa alla quale bisogna prestare attenzione è che l'arco sopra la colonna abbia una forma continua se si affianco due copie del modello. In questo modo è molto più facile riutilizzare l'oggetto anche in ambienti differenti in quanto si possono creare file potenzialmente infinite di colonne.

Per "congiungere" il modello al muro si è deciso di creare un'altra semplice mesh che continui la curvatura dell'arco ma che sia sprovvista della colonna centrale.

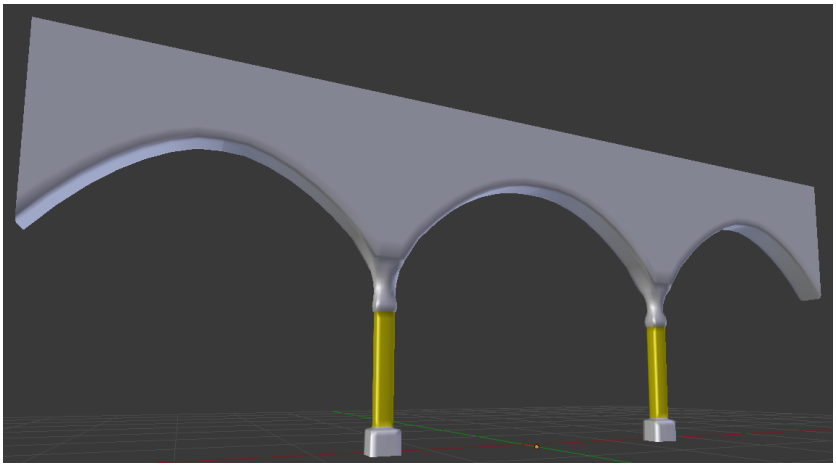


Figura 3.6: Serie di colonne

E' ora il turno dell'ingresso. L'approccio è sempre lo stesso: si parte da un cubo, lo si deforma finché non ha delle dimensioni simili all'oggetto che si vuole ricreare, gli si aggiungono edge loop dove si vuole avere altro dettaglio ed infine si elimina o sposta geometria fino ad ottenere la forma voluta. Se si vogliono creare nette sporgenze o rientranze si utilizza l'estrusione di alcune facce. Nel caso dell'ingresso è utilizzata per fare l'incasso in cui scorrerà la grata che chiude il passaggio.

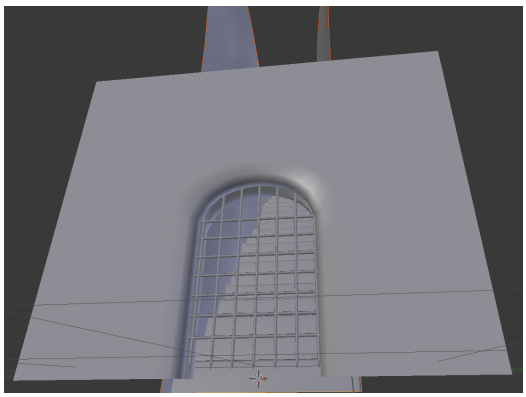


Figura 3.7: L'ingresso finito

Per modellare quest'ultima si parte invece da un cilindro. Questo viene scalato in modo da fargli avere le sembianze di un tubo e poi duplicato, spostato e ruotato fino a formare una griglia.

Siccome è possibile vedere attraverso la grata è necessario avere qualcosa all'esterno. In questo caso si è deciso di mettere una scala scavata nella terra in modo da non dover creare un intero paesaggio. Si è quindi fatto il modello 3D prima di un singolo gradino e poi lo si è duplicato fino ad avere l'intera rampa. Inoltre ai lati si sono messi due piani resi irregolari suddividendoli in piccole facce e spostandone alcune con il Proportional Editing Falloff.

Non rimarrebbe ora che posizionare tutti gli elementi in maniera accurata ma questo verrà fatto in seguito all'interno di Unity3D.

3.1.2 La stanza circolare

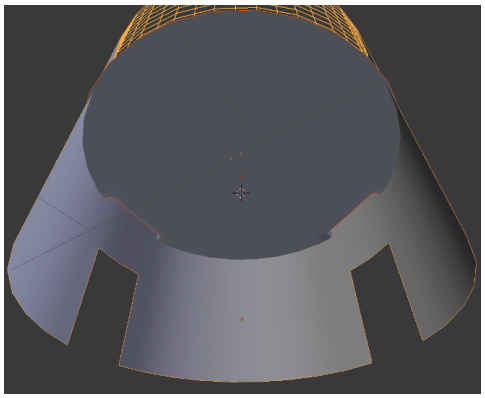
Si passa ora alla seconda stanza, quella circolare. Chiaramente questa volta partire da un cubo sarebbe estremamente controproducente in quanto la sala è di per se stessa cilindrica. Basterà aggiungere quindi un cilindro alla scena e modificarne le dimensioni fino ad ottenere le proporzioni desiderate. Si dovranno poi però eliminare alcune facce per far spazio alle uscite.

Inoltre il soffitto dovrebbe poter scendere visto che costituirà una trappola. Per ottenere questo risultato basterà selezionare tutte le facce che lo compongono e con esse creare un nuovo oggetto distinto.

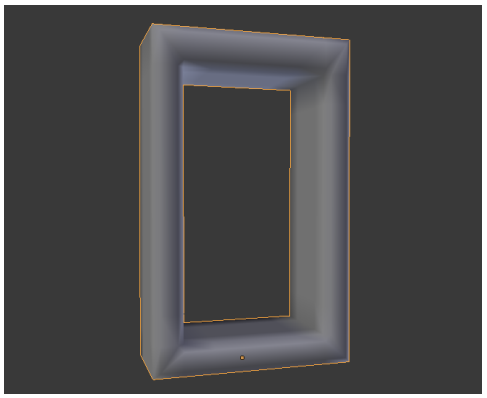
Dei semplici buchi sul muro non possono chiaramente bastare come delle uscite in questo caso. Si è deciso quindi di fare un modello aggiuntivo in modo da rendere il risultato finale più gradevole. Questo consisterà fondamentalmente in uno spesso stipite. Come quasi sempre si parte da un cubo e la tecnica è sempre la stessa: aggiungere edge loop, traslare, ruotare, scalare ed

estrudere. Quando si modellano oggetti che potrebbero essere scomposti in due parti simmetriche, come ad esempio il bordo porta che si è appena fatto, può essere comodo modellare solamente metà dell'oggetto e poi applicarvici un *Mirror Modifier*. Questo fondamentalmente crea una copia speculare della geometria andando così a completare la metà mancante dell'elemento al quale si sta lavorando.

Si può notare che con l'aggiunta degli stipiti il soffitto va modificato visto che scendendo andrebbe a bloccarsi. Per risolvere questo problema è sufficiente inserire delle profonde rientranze in corrispondenza dei due stipiti.



(a) Stanza circolare



(b) Stipite

Solo tre cose mancano ora: gli spuntoni che dovrebbero uscire dal soffitto, il mattone sporgente che doverebbe fungere da bottone e delle barre da usare come blocco per le uscite. Per quanto riguarda gli spuntoni basta partire da un cilindro, scalarlo fino a fargli raggiungere la dimensione desiderata ed aggiungervi la punta. Questo singolo spuntone potrà poi essere duplicato e posizionato quante volte si vuole fino a dare l'effetto desiderato. Ovviamente il soffitto dovrebbe avere un buco in corrispondenza di ognuno di essi ma questo problema verrà risolto in seguito con delle texture.

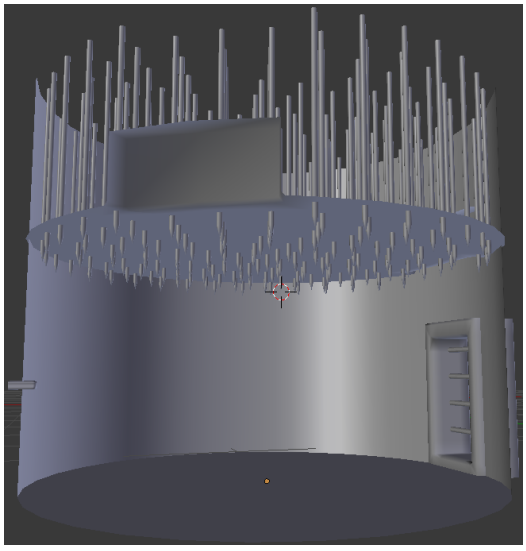


Figura 3.9: Seconda stanza ultimata

Per quanto riguarda il mattone sporgente e le barre l'approccio è sempre quello descritto per gli altri elementi. Creare modelli 3D come quelli fatti

fin'ora non è particolarmente complicato: è solamente importante conoscere gli strumenti che si hanno a disposizione e avere una buona pazienza nel posizionare accuratamente la geometria.

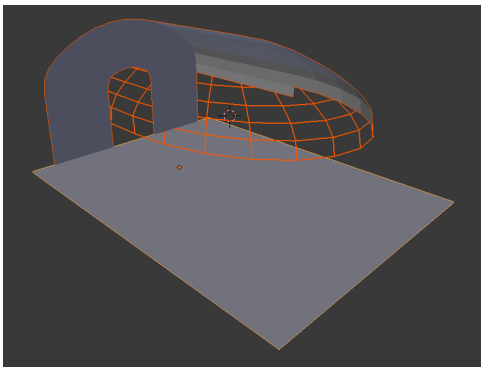
3.1.3 La stanza degli altari

La terza sala, quella con la serie di altari, è sicuramente la più complicata a livello di realizzazione. Questo a causa appunto degli altari che hanno una forma tutto sommato articolata. Per prima cosa conviene comunque sia fare pavimento pareti e soffitto. Questi tre elementi non devono per forza essere contenuti nella stessa mesh ma possono anche essere separati se riteniamo che sia più comodo lavorarci.

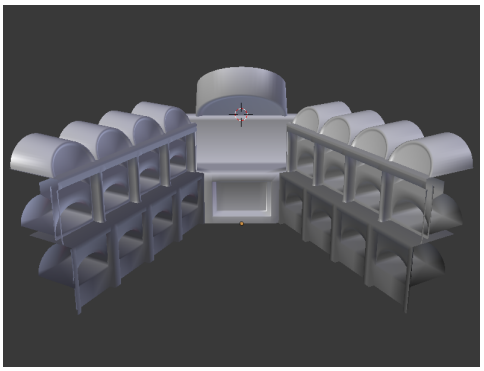
Fatta la base della stanza si può passare a realizzare il primo altare. Il metodo è chiaramente sempre quello spiegato. Unica piccola differenza è che potrebbe tornare utile aggiungere più di una mesh base per ottenere più rapidamente il risultato voluto. Inoltre potrà essere necessaria un po' più di pazienza.

Una volta finito un singolo altare potrà essere duplicato e spostato per formare l'intera fila. Si noterà però che le due terminazioni andranno in qualche modo coperte. A questo scopo si è deciso di utilizzare una specie di lastra di pietra.

Ultimate e correttamente posizionate le due file si può procedere a modellare l'altare più grande che dovrà essere disposto nel mezzo. Nel farlo unica cosa alla quale bisogna stare attenti è che non lasci dei buchi e che completi perfettamente il semicerchio.



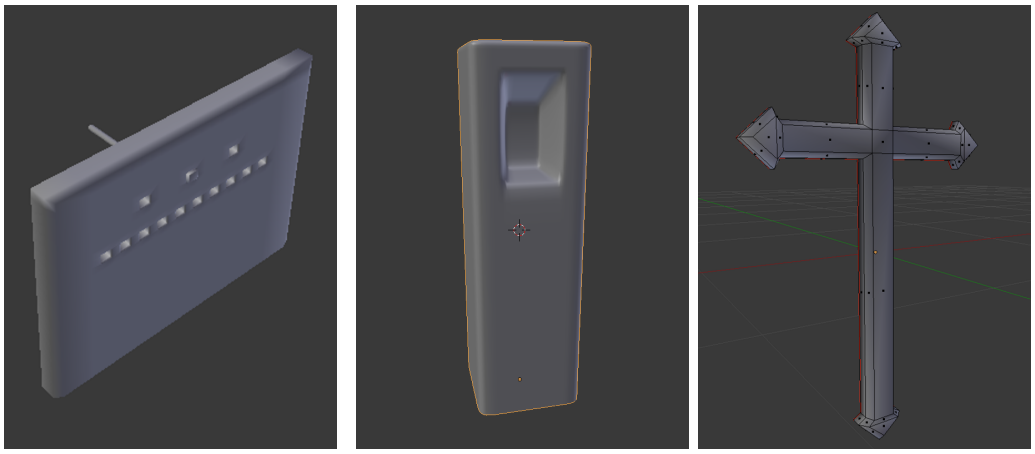
(a) La terza stanza



(b) Il semicerchio di altari

Gli oggetti rimanenti sono ora: la parete con i pulsanti e gli spuntoni, le tavolette incise, il pilastro incassato nel terreno e la croce da mettere nella sua cavità. Tutti questi possono essere modellati in maniera piuttosto semplice

con il metodo visto fin'ora. Bisogna notare che tutte le incisioni non vanno fatte in tre dimensioni sulla mesh: i dettagli verranno aggiunti poi con le texture.



(a) La parete dei bottoni **(b)** Il pilastro **(c)** La croce

3.1.4 I corridoi

Realizzare i corridoi è molto semplice e il metodo è sempre quello usato per la maggioranza degli altri modelli. Ovviamente bisogna fare in modo che le terminazioni si congiungano bene con le stanze. Inoltre il secondo corridoio deve avere un bivio e una frana alla fine di una delle due biforcazioni. Per fare il modello della frana è sufficiente partire da un piano, aggiungervi facce e rendere la superficie irregolare in modo da dargli un aspetto un po' roccioso.

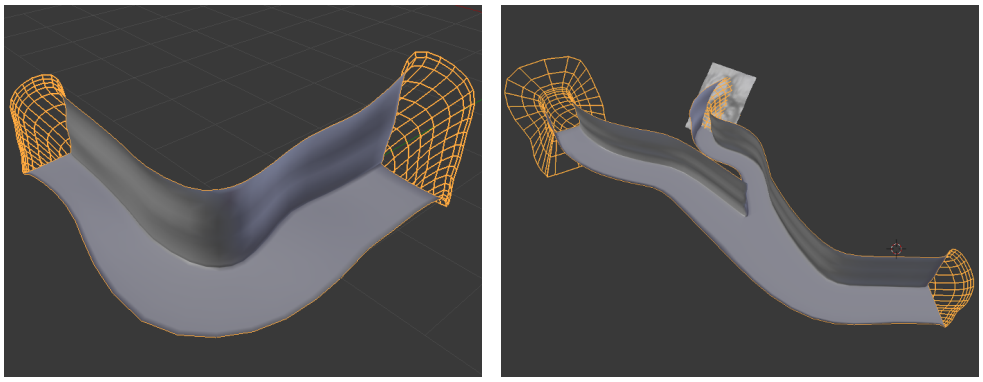


Figura 3.12: I due corridoi

3.2 Modeling di oggetti complessi

In generale i modelli più complessi da realizzare sono quelli di esseri viventi o di cose a essi legati. Nello specifico si tratterà di uno scheletro.

3.3 Il teschio

Per primo si è deciso di partire solamente dal teschio. Si dovranno quindi realizzare cranio, mandibola e denti. Quando si vuole modellare qualcosa di un po' complicato è molto utile importare delle immagini di riferimento direttamente in Blender in modo da poterci quasi "ricalcare" sopra.

Il procedimento per un primo modello base è sempre quello esposto nel capitolo precedente. Fondamentale è ruotare spesso la visuale in modo da controllare che la propria mesh stia venendo bene da tutte le prospettive. Inoltre bisogna sempre tenere d'occhio le immagini di riferimento e fare in modo che il modello le rispetti in maniera sufficientemente accurata. Il lavoro per ottenere dei buoni risultati è piuttosto lungo e richiede notevoli dosi di pazienza. Un primo modello base è stato realizzato in circa un paio di giorni.

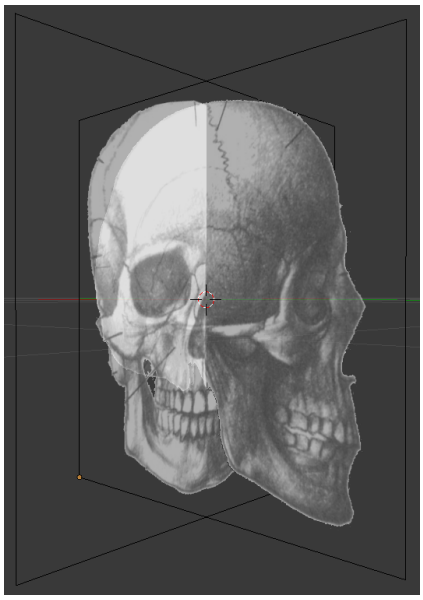
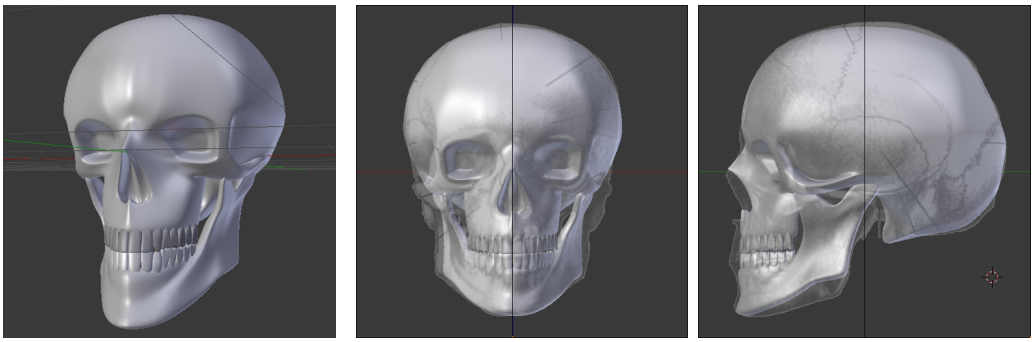


Figura 3.13: Immagini di riferimento per il teschio



(a) Vista prospettica **(b)** Vista frontale **(c)** Vista laterale

Figura 3.14: Primo modello del teschio

Si può aggiungere anche il modificatore Subdivision Surface per avere una superficie molto più arrotondata. Seguendo i riferimenti e aggiungendo geometria mano a mano che serve, le probabilità di ottenere un modello "pulito" sono molto basse. Una mesh può considerarsi pulita se non è presente geometria inutile e se la densità di facce rispecchia la quantità di dettaglio necessario.

Una volta fatto il primo modello si esegue un processo chiamato *retopologizzazione*. Questo consiste nel creare una nuova mesh da modellare "ricalcando" quella precedente stando però attenti a dove e come le facce vengono posizionate, oltre che al loro numero. In questo Blender ci aiuta con i sui tool di *snapping*. Questi servono ad "ancorare" i vertici che si muovono a della geometria preesistente e ci permettono di fare in modo che tutta la superficie del nuovo modello possa seguire la forma di quello precedente senza quasi alcuno sforzo. Bisogna però stare molto attenti a dove le facce vengono posizionate in modo che la nuova topologia sia più pulita ed ordinata possibile. In generale si cerca anche di evitare quando possibile la presenza di facce triangolari.

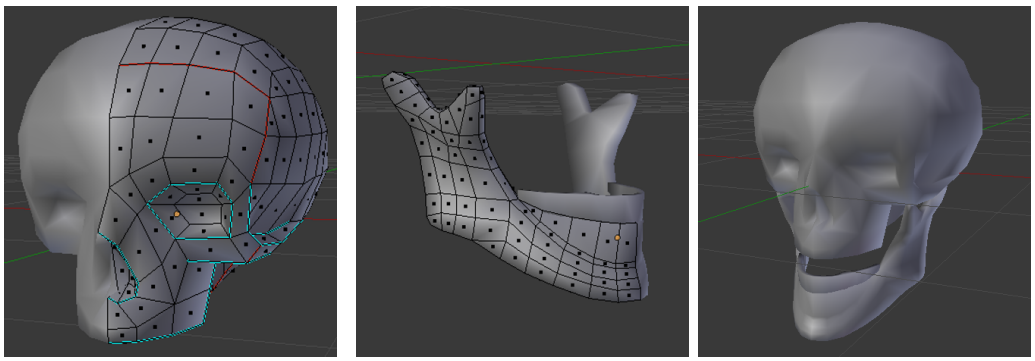


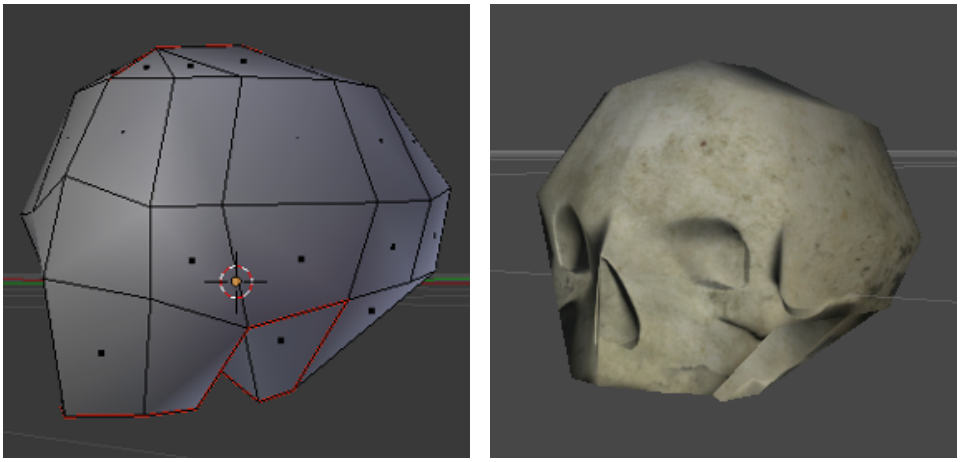
Figura 3.15: Teschio retopologizzato

Per *topologia* di un modello tridimensionale si intende come la geometria è disposta per formare la sua superficie.

Eseguire il processo di retopologizzazione partendo da un modello estremamente denso di poligoni e dettagli non essenziali per crearne una versione molto più semplice ci permette poi di utilizzare delle tecniche per simulare tutti i particolari che sono andati perduti nel secondo oggetto. Queste tecniche verranno esaminate nella sezione riguardante il texturing.

Può essere utile retopologizzare un modello più volte per crearne varie versioni di complessità poligonale differente. In "Catacomb" ad esempio è stata fatta anche una versione estremamente "Low-Poly" (con pochi poligoni)

del teschio. Questo per avere la possibilità di usarla un gran numero di volte nelle scene del gioco senza doversi preoccupare di possibili cali di prestazioni.



(a) Senza Texture **(b)** Con Texture

Figura 3.16: Teschio estremamente Low-Poly

3.4 Lo scheletro

Il corpo umano è molto complesso e quindi per "Catacomb" si è valutato che avere un modello tridimensionale di uno scheletro completo avrebbe richiesto decisamente troppo tempo. Si è deciso quindi di approcciarsi al problema in modo diverso: i corpi scheletrici contenuti nei vari loculi saranno vestiti con una tonaca. In questo modo si ha comunque un buon effetto visivo risparmiandosi però molto lavoro.

Sarà comunque sia necessario creare un modello molto grezzo di cassa toracica, bacino, gambe e braccia in quanto simulare una veste senza la concezione di quello che si ha sotto può risultare estremamente complicato. Per questo motivo si è deciso di partire, come per il teschio, importando in Blender delle immagini di riferimento da seguire.

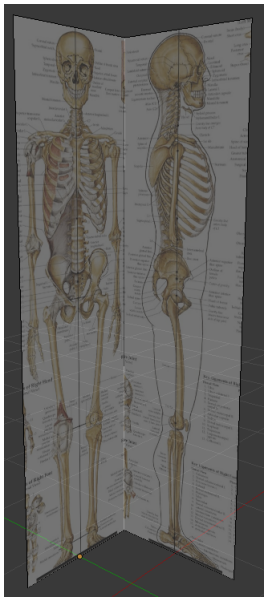
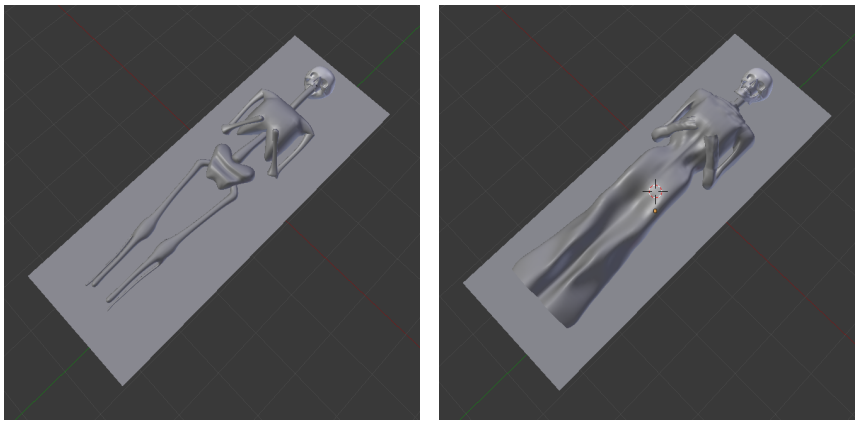


Figura 3.17: Immagini di riferimento per lo scheletro

Una volta create la base di partenza con le tecniche viste fino ad ora si potrà procedere a modellare la tonaca. Nel progetto questo è stato fatto creando un cilindro e posizionandolo e scalandolo attorno allo scheletro in modo da farglielo avvolgere completamente. Si

Sono poi aggiunti innumerevoli edge loop in modo da avere una buona densità di poligoni e quindi maggiore libertà nella deformazione. In seguito con il Proportional Editing Falloff e molta pazienza si è fatto si che la superficie del cilindro si adagiasse in maniera sufficientemente precisa su quella della mesh sottostante, cercando di dare al tutto una aspetto più simile possibile a quello della stoffa. Può essere molto utile posizionare sotto allo scheletro un piano in modo da rendersi conto di che effetto farà il modello una volta adagiato dentro un loculo.



(a) Senza tonaca **(b)** Con tonaca

Figura 3.18: Modello grezzo dello scheletro

Fatto questo sarà necessario avere dei modelli per mani e piedi visto che nonostante il vestito saranno comunque visibili. Essendo anche questi elementi non banali è utile partire da delle immagini di riferimento in modo da avere un risultato più realistico possibile. Una volta fatte le mesh ovviamente andranno posizionate in modo conforme alle pieghe della veste. Come al solito bisogna prestare estrema attenzione alle proporzioni.

(a) Mani **(b)** Piedi **(c)** Scheletro ultimato

3.5 UVMapping e Texturing

Ora che si hanno tutti i modelli tridimensionali che andranno a comporre l'ambiente del videogioco è necessario applicarvici sopra delle texture, altrimenti tutto sarebbe al massimo colorato in modo "piatto". Prima di fare questo è però necessario assicurarsi che ogni mesh abbia una propria UVMap. Come detto nel primo capitolo una *UVMap* è una mappatura dell'oggetto 3D su un piano bidimensionale necessaria ad applicarce delle immagini sulla superficie di un modello. Per lo scopo di questa tesi non avrebbe senso entrare nel dettaglio dell'UVMapping di ogni singolo oggetto modellato. Si forniranno invece delle linee guida del metodo utilizzato.

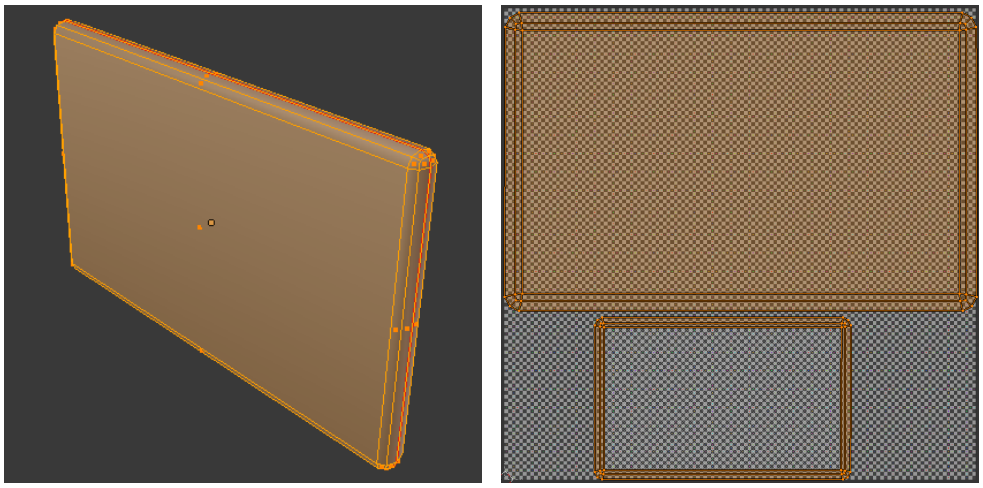
Il primo passo per una buona mappatura è identificare dove si vuole che il modello tridimensionale venga tagliato. Questo va fatto perchè è ovviamente impossibile mantenere la contiguità di tutte le facce su di un piano: la mesh va quindi divisa in "regioni". Basterà selezionare tutti i lati dove si pensa ci debba essere un taglio ed eseguire in Blender il comando *Mark Seam*. Per renderli facilmente riconoscibili questi saranno colorati in rosso.

Dopo di ciò bisognerà fare un *Unwrap* del modello. Questo ci darà una mappatura di base secondo i tagli da noi indicati cercando di minimizzare la distorsione delle facce. Si può poi procedere a modificare a mano l'UVMap come ci sembra più opportuno e funzionale.

Per un buon risultato è fondamentale tenere a mente dove si vuole che si abbia continuità della texture che verrà poi applicata in modo da poter posizionare adeguatamente le varie regioni dell'UVMap. Ad esempio il muro della stanza circolare dovrà chiaramente sembrare continuo nonostante la necessità di posizionarvi un taglio per poterlo trasporre su un piano. La sua mappatura sarà una serie di facce in cui quella più a destra e quella più a sinistra nella mesh sono contigue. Per assicurarsi che questo taglio non si faccia notare basterà posizionare nell'UVMap questi due poligoni in modo che siano esattamente alla stessa altezza e che uno sia completamente a sinistra e l'altro completamente a destra della mappa. In questo modo applicando poi una texture *seamless* il taglio sarà del tutto impossibile da notare.

Per *texture seamless* si intende una immagine fatta in modo tale da poter essere ripetuta e affiancata infinite volte senza mostrare alcun tipo di discontinuità. Questo tipo di texture sono molto usate nel mondo della grafica 3D.

Anche se il processo di UVMapping sembra molto semplice in realtà non lo è. Per ottenere dei risultati soddisfacenti c'è bisogno di una accurata pianificazione e, come al solito, di pazienza ed esperienza. Se si commettono



(a) Modello 3D **(b)** UVMap

Figura 3.20: La tavoletta e la sua UVMap

errori l'effetto finale potrebbe essere altamente irrealistico e presentare forti distorsioni dell'immagine.

Una volta che tutti i modelli sono stati "UV mappati" si potrà procedere alla creazione delle varie Texture necessarie ad ottenere una qualità visuale soddisfacente. Ne saranno necessarie tre tipologie differenti: texture di colore, normal maps e displacement maps.

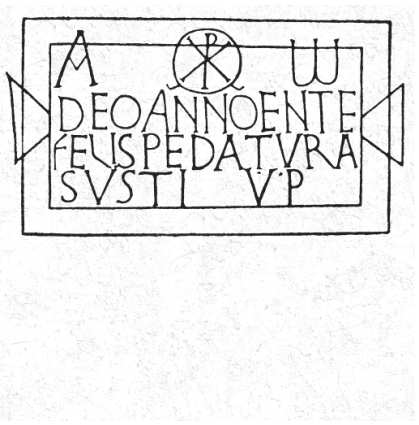
3.5.1 Texture di colore

La tipologia di texture più basilare utilizzata in *Computer Graphics* è quella per dare il colore ad un oggetto. Il suo funzionamento è estremamente semplice: l'immagine viene "disposta" sulla superficie del modello tridimensionale a seconda della sua mappa UV ed essa conferisce il colore del materiale.

A seconda dell'oggetto e della sua UVMap ci si troveranno ad usare principalmente o texture seamless generiche oppure delle texture prodotte ad hoc. Le prime vengono in genere usate per dare idea del materiale di cui è fatto il modello mentre le seconde a dare dei dettagli specifici ad un singolo oggetto. Mentre le prime sono utilizzabili in una enorme quantità di situazioni le seconde sono altamente specifiche. E' possibile che nello stesso modello si utilizzino entrambe le tipologie.

In "Catacomb" le texture seamless sono state principalmente utilizzate per muri, soffitti e pavimenti mentre per oggetti specifici come le tavolette con le incisioni sono state disegnate texture apposite.

(a) Texture seamless per il muro di mattoni



(b) Texture ad hoc per una tavoletta

Figura 3.21: Esempi di texture

3.5.2 Normal mapping

Altro tipo di texture largamente utilizzato è costituito dalle normal map. Per capire cosa fanno è necessario avere la nozione di *vettore normale*. Per *vettore normale* ad una superficie si intende quel versore (vettore di modulo uno) a tre dimensioni che ne è perpendicolare.

In una normal map ogni pixel dell'immagine rappresenta un vettore a tre dimensioni costruito sulla base dei valori dei canali RGB (rosso, verde e blu) che la costituiscono. Questo vettore verrà moltiplicato al vettore normale alla superficie di un modello nel punto corrispondente basandosi sulla sua UVMap. In questo modo si può fare in modo che la luce si comporti come se delle superfici piane siano in realtà ricurve dando quindi l'impressione di un maggiore dettaglio.

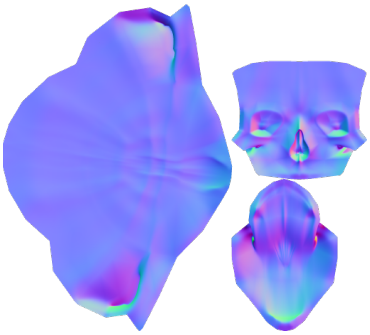


Figura 3.22: Normal map del teschio

Ovviamente disegnare a mano una normal map è pressoché impossibile. Queste vengono generalmente prodotte in due modi:

- si prendono due modelli simili, uno dei quali ha effettivamente modellati tutti i dettagli che si vogliono, e si "stampano" su di una texture tutte le differenze tra le normali alle superfici dei due. In questo quando la luce interagirà con l'oggetto senza dettagli come se li avesse.

- si cerca di creare mediante degli algoritmi specifici una normal map a partire da una semplice immagine (normalmente la texture di colore dell'oggetto) valutando ad esempio ombreggiature e zone contigue della stessa tonalità.

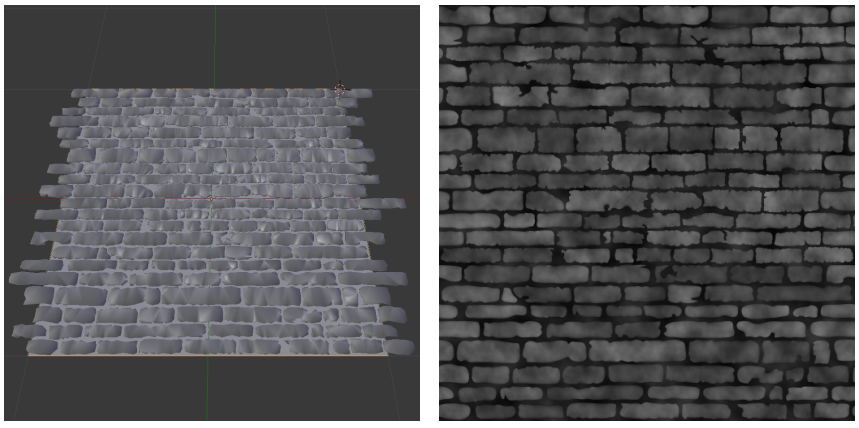
In "Catacomb" si è fatto uso di entrambe le metodologie appena descritte. La prima è stata usata ad esempio per il teschio e i muri di mattoni mentre la seconda per terra, intonaco e pietra.

3.5.3 Displacement mapping

Una displacement map è una texture in scala di grigi con fondamentalmente lo stesso scopo della normal map: salvare i dettagli di un modello su di una texture in modo da non doverli avere nella mesh. E' diverso però il suo modo di agire: ogni punto dell'immagine indicherà di quanto il vertice corrispondente secondo la UVMap deva essere "fisicamente" spostato lungo il suo vettore normale.

Come si può intuire per avere un buon risultato si dovrà quindi applicare su di un modello con una alta densità di vertici. Per questo motivo viene largamente utilizzata in collaborazione con la tassellazione, tecnica che verrà descritta nella prossima sezione.

Anche le displacement map possono essere prodotte o salvando su una texture le differenze tra due superfici (*Baking*) o cercando di "intuirle" partendo da delle immagini.

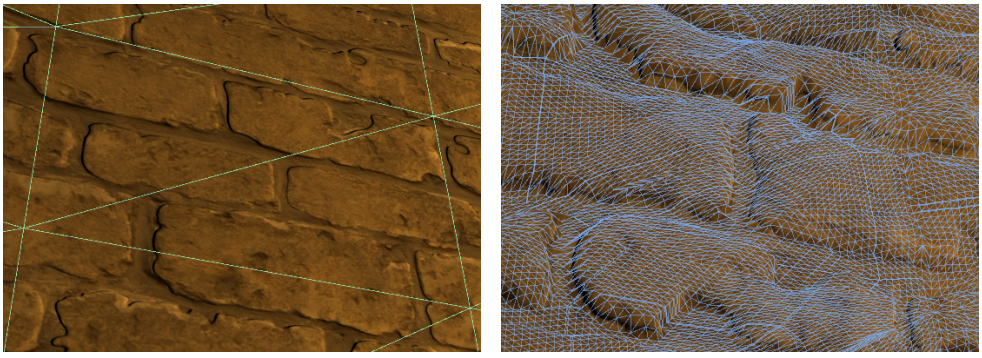


(a) Modello di partenza **(b)** Texture ottenuta

Figura 3.23: Baking di una displacement map

3.6 Tassellazione e stereoscopia

La tassellazione è una tecnica che consiste nel suddividere le facce di un modello in un numero variabile di sotto-facce in modo da avere più densità di vertici. In questo modo è possibile, in congiunzione ad esempio con una displacement map, incrementare dinamicamente i dettagli "fisicamente" presenti sulla superficie di un oggetto. In generale la quantità di facce aggiunte viene calcolata in base alla distanza della telecamera in modo da rendere più dettagliati solamente gli oggetti più vicini dove la maggiore definizione della superficie può essere effettivamente apprezzata.



(a) Muro non tassellato **(b)** Muro tassellato

Figura 3.24: Esempio di tassellazione

La stereoscopia è una tecnica sia per la realizzazione che per la visione di immagini con lo scopo di trasmettere una illusione di effettiva tridimensionalità. In "Catacomb" è particolarmente importante in quanto si fa uso di realtà virtuale. Esistono vari modi per ottenere una immagine stereoscopica e quasi tutti prevedono l'unione di due immagini con varie tecniche. Queste due immagini saranno una per l'occhio sinistro e una per l'occhio destro.

Nel caso dell'Oculus Rift le due immagini servono affiancate e ad esse va applicata una apposita distorsione, la quale verrà "controbilanciata" dalle lenti attraverso le quali si visualizzerà il tutto. Il risultato finale sarà estremamente realistico e anche il campo di visione sufficientemente ampio (110 gradi).

3.7 Creazione shader e materiali in Unity3D

Ora che si hanno a disposizione tutti i modelli e le relative texture si può procedere alla creazione dei materiali in Unity3D. Questi controlleranno come i vari oggetti reagiranno alla luce e come le texture verranno utilizzate per

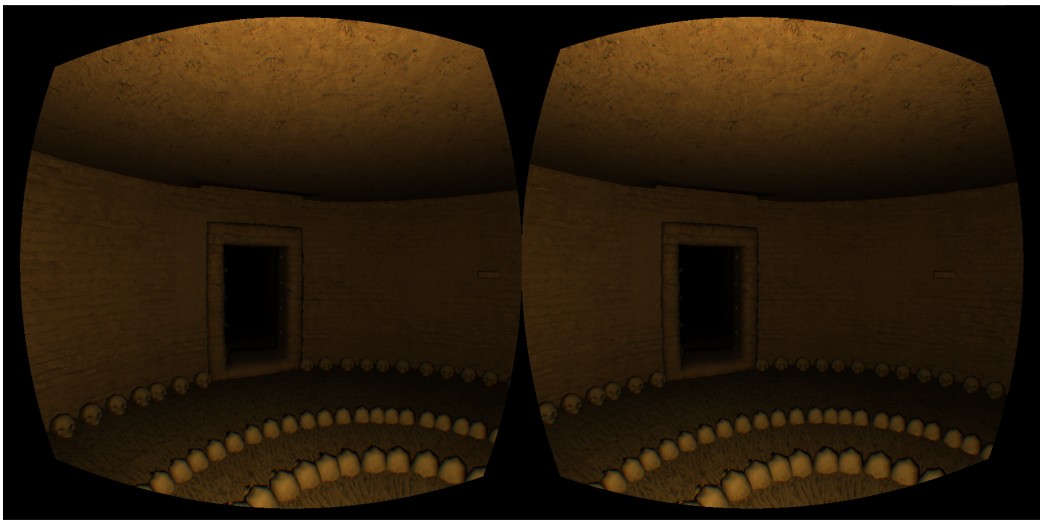


Figura 3.25: Immagine stereoscopica per Oculus Rift

raggiungere l'aspetto visivo finale. Tutto questo è ottenuto attraverso degli shader. Per comprendere nel dettaglio il loro funzionamento si propone un esempio di *Surface Shader* utilizzato in "Catacomb" .

```
Properties {
        _BasePar ("Base Height",Float)=0
        _Parallax ("Height", Float) = 0.5
        _DecalParallax ("Decal Height", Float) = 0.5
        _MainTex ("Base (RGB) ", 2D) = "white" {}
        _DecalTex ("Decal (RGB) ", 2D) = "white" {}
        _DecalMask("DecalMask",2D)="black"{}
        _BumpMap ("Normalmap", 2D) = "bump" {}
        _ParallaxMap ("Heightmap (A)", 2D) = "black" {}
        _DecalBumpMap ("Decal Normalmap", 2D) = "bump" {}
        _DecalParallaxMap ("Decal Heightmap (A)", 2D) =
           "black" {}
        _EdgeLength ("Edge length", Float) = 10
        _Tiles ("Tiles",Float)=1
        _DecalTiles("Decal Tiles",Float)=1
        _MaskTiles("Mask Tiles",Float)=1
}
SubShader {
        Tags { "RenderType"="Opaque" }

        CGPROGRAM
```

```
#pragma surface surf Lambert addshadow
   vertex:disp tessellate:tessEdge
#include "Tessellation.cginc"

struct appdata {
        float4 vertex : POSITION;
        float4 tangent : TANGENT;
        float3 normal : NORMAL;
        float2 texcoord : TEXCOORD0;
        float2 texcoord1 : TEXCOORD1;
};

float _EdgeLength;
float _Parallax;
float _BasePar;
float _DecalParallax;

float4 tessEdge (appdata v0, appdata v1, appdata
   v2)
{
        return UnityEdgeLengthBasedTessCull
           (v0.vertex, v1.vertex, v2.vertex,
           _EdgeLength,(_BasePar+_Parallax) *
           1.5f);
}

sampler2D _ParallaxMap;
sampler2D _DecalParallaxMap;
sampler2D _DecalMask;
float _Tiles;
float _DecalTiles;
float _MaskTiles;

void disp (inout appdata v)
{
        float d1 =_BasePar +
           tex2Dlod(_ParallaxMap,
           float4(v.texcoord.xy*_Tiles,0,0)).r *
           _Parallax;
        float d2 = tex2Dlod(_DecalParallaxMap,
           float4(v.texcoord.xy*_DecalTiles,0,0)).r
           * _DecalParallax;
        float m = tex2Dlod(_DecalMask,
```

```
                float4(v.texcoord.xy*_MaskTiles,0,0)).r;

            v.vertex.xyz += v.normal * lerp(d1,d2,m);
        }

        sampler2D _MainTex;
        sampler2D _DecalTex;
        sampler2D _BumpMap;
        sampler2D _DecalBumpMap;

        struct Input {
            float2 uv_MainTex;
            float2 uv_DecalTex;
            float2 uv_BumpMap;
            float2 uv_DecalBumpMap;
            float2 uv_DecalMask;
        };

        void surf (Input IN, inout SurfaceOutput o) {
            fixed4 tex = tex2D(_MainTex,
                IN.uv_MainTex);
            fixed4 decal =
                tex2D(_DecalTex,IN.uv_DecalTex);
            fixed decmask =
                tex2D(_DecalMask,IN.uv_DecalMask);
            tex =lerp(tex,decal,decmask);
            float3 norm=UnpackNormal(tex2D(_BumpMap,
                IN.uv_BumpMap));
            float3
                decalNorm=UnpackNormal(tex2D(_DecalBumpMap,
                IN.uv_DecalBumpMap));
            norm=lerp(norm,decalNorm,decmask);

            o.Albedo = tex.rgb;
            o.Alpha = tex.a ;
            o.Normal = norm;
        }
        ENDCG
}
```

Si può subito notare come il codice sia diviso in due blocchi ben distinti: Properties e SubShader. Il primo costituisce l'"interfaccia" con Unity3D: in esso vanno tutte quante le proprietà di cui lo shader ha bisogno e al-

le quali andranno poi assegnati dei valori. Il secondo definisce l'effettivo comportamento della superfici che ne fanno uso.

Una proprietà è scritta seguendo la sintassi _NomeVariabile ("nomeVisualizzatoInEditor", tipo) = valoreDiDefault.

Il blocco SubShader inizia con dei tag che definiscono alcune caratteristiche dello shader (in questo caso che è opaco). Subito a seguire si può vedere la stringa *CGPROGRAM*: questa sta ad indicare che da qui fino ad *ENDCG* si sta usando il linguaggio *CG*.

Il *#pragma* da alcune direttive al compilatore:

- *surface surf:* indica che si tratta di un Surface Shader e che la funzione nella quale avviene lo shading è chiamata *surf*.
- *Lambert:* indica il modello di illuminazione da usare. In questo caso si usa il modello di Lambert integrato in Unity3D. Volendo se ne possono scrivere di propri.
- *vertex:disp* indica che verranno effettuate operazioni ai vertici oltre che alla superficie nella funzione *disp*.
- *tessellate:tessEdge* indica che verrà effettuata della tassellazione nella funzione *tessEdge*.

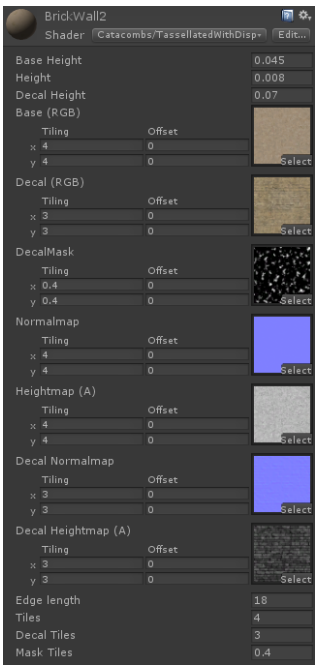


Figura 3.26: Proprietà di un materiale

Si possono inoltre inserire altre direttive con scopi diversi, come ad esempio *addshadow* che permette alle ombre dinamiche di essere proiettate su di un materiale facente uso di questo shader.

Come si può notare vengono anche dichiarate una variabile per ogni proprietà del blocco Properties. I principale tipi di variabile che si hanno in CG sono: sampler2D (per le texture), fixed, half e float. Gli ultimi tre sono tutti numerici con precisione differente e possono anche essere seguiti da un numero n per avere un vettore a n dimensioni (ad esempio float3).

Sono poi presenti due strutture dati: appdata e Input. Entrambe servono ad accedere ai dati del modello tridimensionale al quale sarà applicato lo shader. Questi dati comprendono ad esempio mappe UV, posizioni dei vertici, vettori normali e altro ancora.

Il funzionamento dello shader proposto è tutto sommato piuttosto semplice. Nella funzione "tessEdge" si effettua la tassellazione inserendo nuove

facce in base ad un valore impostabile dell'utente e alla lunghezza dei lati della mesh in modo da mantenere la densità di geometria costante. In "disp" si ha il displacement dei vertici sulla base di tre differenti texture di cui una funge da maschera per valutare quale delle altre due abbia più peso in un determinato punto. Anche in "surf" vengono usati dei set di tre texture con lo stesso principio, uno per il colore e uno per le normali. Si è deciso di usare tre texture invece di una per dare maggiore varietà alla superficie: infatti la prima immagini costituisce la base, la seconda una sorta di variazione che gli può essere applicata sopra e la terza una maschera in scala di grigi che indica in ogni punto se si vede di più la prima texture o la seconda. Un esempio di utilizzo pratico è nei muri ricoperti da intonaco (prima texture) dove in alcuni punti (seconda texture) si vedono i mattoni sottostanti (terza texture). Nelle ultime righe di codice della funzione "surf" vengono settate le varie caratteristiche della superficie appena calcolate (in questo caso colore e normali).

Non si sta a descrivere singolarmente ogni shader scritto in quanto il principio è sempre lo stesso, saranno presenti solamente alcune varianti: ad esempio non tutti fanno uso di tassellazione o di displacement maps e non tutti sono strutturati con texture su più livelli.

Unity3D mette anche disposizione una buona quantità di shader già fatti che soddisfano le esigenze più comuni.

Si può procedere ora a creare i materiali. Per fare questo basta aprire Unity3D e crearne uno nuovo in una cartella a piacimento. Si avrà quindi la possibilità di dare valori o texture a tutte le proprietà dello shader. Una volta finita la configurazione si potrà assegnarlo ad uno o più oggetti, conferendogli così l'aspetto desiderato. E' importante notare che più materiali possono avere lo stesso shader in quanto esso definisce il comportamento in generale e non è riferito ad un caso specifico. Ogni materiale facente uso dello stesso shader potrà quindi assegnare valori diversi alle varie proprietà.

Figura 3.27: Risultato in-game dello shader proposto

3.8 Luci ed illuminazione

Come si può immaginare le luci sono fondamentali visto che senza di esse sarebbe impossibile vedere tutto quello che si è fatto fin'ora. Non si entra nel dettaglio di come queste funzionano e sono implementate in quanto si tratta di un argomento estremamente complesso che richiederebbe anche uno studio di come agiscono in fisica. Si tratterà invece brevemente come utilizzarle efficacemente in Unity3D.

Quando si vuole illuminare una scena il primo passo è chiaramente inserire delle nuove luci. Queste possono essere principalmente di tre tipi:

- **Point Light:** luce emessa a partire da un punto centrale verso tutte le direzioni che colpisce tutto quello che è dentro il suo raggio, ad esempio come una lampadina.
- **Spotlight:** luce che colpisce tutto quello all'interno di un cono definito da un angolo e una lunghezza, come ad esempio un faro.
- **Directional Light:** luce posizionata ad una distanza indefinita e che colpisce tutta la scena, come il Sole.

Una volta inserita nella scena il tipo di luce desiderata si potranno modificare alcuni suoi parametri quali raggio, colore e intensità. Inoltre sarà possibile specificare se dovrà proiettare ombre dinamiche o no, potendo anche decidere la qualità di queste.

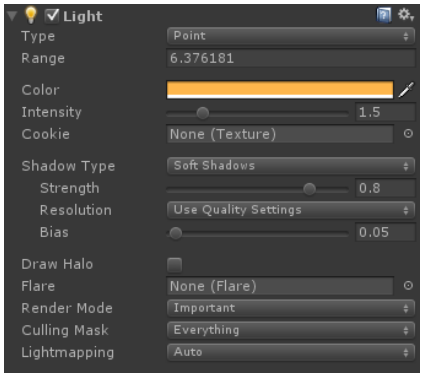


Figura 3.28: Pannello per la configurazione di una luce

Se le luci che si usano sono statiche (non si muovono durante il corso del videogioco) Unity3D mette anche a disposizione la possibilità di pre-calcolare le loro ombre in modo da alleggerire il carico di lavoro della GPU e avere anche un risultato più preciso.

All'interno di una scena si possono avere contemporaneamente sia luci dinamiche che statiche. In generale è buona norma cercare di limitare il più possibile quelle del primo tipo se non si vuole rischiare di incorrere in considerevoli cali di prestazioni.

Quando si configura una luce bisogna prestare molta attenzione a colore e intensità in quanto con valori sbagliati si rischia di compromettere l'aspetto delle superfici degli oggetti. Ad esempio una luce troppo colorata potrebbe completamente stravolgere la tonalità di una texture portando ad una perdita di dettagli.

Capitolo 4

Sviluppi futuri

A questo punto si può dire che in "Catacomb" si abbia tutto sommato una grafica discreta. Ovviamente però beneficerebbe notevolmente, sia dal lato visivo che da quello delle prestazioni, da alcuni lavori aggiuntivi.

4.1 Mesh aggiuntive

Innanzitutto sarebbe veramente molto utile avere più oggetti generici senza un preciso scopo da utilizzare per decorare le stanze in modo da renderle meno monotone. Questi potrebbero essere ad esempio delle altre tipologie di ossa, dei mucchietti di terra, delle ragnatele o dei contenitori. Per quanto questi oggetti siano di semplice realizzazione la quantità necessaria a dare un buon senso di varietà è comunque notevole, rendendo questo lavoro piuttosto lungo e laborioso.

Altra cosa della quale sarebbe molto utile avere un modello 3D è la parte dello scheletro non ancora fatta, che include tutto tranne testa mani e piedi. C'è bisogno però di molto tempo e di abilità visto che il corpo umano (anche se solo le ossa) non è facile da mantenere sempre ben fatto e proporzionato.

Si potrebbe anche perfezionare il modello della tonaca, utilizzando magari tecniche più avanzate come la simulazione dei tessuti. In questo modo il risultato sarebbe sicuramente molto più fisicamente corretto.

4.2 Rigging e Skinning

Sempre per dare maggiore varietà sarebbe molto utile poter mettere lo scheletro in pose diverse oltre a quella nella quale lo si è modellato: questo per avere la possibilità di ripeterlo senza dare un forte senso di monotonia. Per ottenere questo ci si affida a *Rigging* e *Skinning*.

Per *Rigging* si intende quel processo che porta alla creazione di un insieme gerarchico di "ossa" interconnesse detto *scheletro* o *rig*.

Per *Skinning* si intende invece quella procedura che porta a legare la mesh di un oggetto ad un rig.

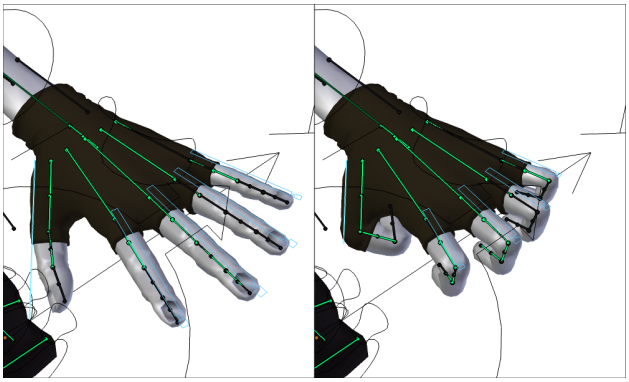


Figura 4.1: Un esempio di rig

In questo modo basterà muovere lo scheletro per spostare le varie parti di un modello. Questo approccio è fondamentale quando si vuole animare qualcosa, soprattutto se si parla di figure umanoidi. Nel caso di "Catacomb" verrebbe comunque sia solamente utilizzata per cambiare posa allo scheletro presente all'interno delle varie tombe

4.3 Level of Detail

Per avere delle prestazioni maggiori si potrebbe decidere di fare dei modelli semplificati degli oggetti presenti da utilizzare in lontananza. Ovviamente quando il giocatore sarà vicino il modello dovrà essere sostituito con quello più dettagliato.

Questo approccio è noto in letteratura come *Level of Detail* (in genere abbreviato in *LOD*). Concettualmente è molto semplice implementare una tecnica del genere. Il problema risiede infatti nel fare i modelli semplificati, cosa che a seconda della complessità della propria scena può diventare notevolmente laboriosa. Ovviamente si può anche decidere di cercare un compromesso ed implementare i LOD solamente per alcuni elementi particolarmente complessi in modo da poterli comunque sia usare senza troppi pensieri.

Capitolo 5

Conclusioni

Il lavoro necessario a creare tutta la grafica che serve a dare un aspetto dignitoso ad un videogioco tridimensionale è sicuramente lungo ed impegnativo e richiede doti che spaziano da quelle tecniche a quelle artistiche. Infatti per avere dei risultati sia belli esteticamente che sufficientemente performanti è necessario da una lato avere delle buone basi teoriche di quello che sta dietro alla *Computer Graphics* e al rendering real-time e dall'altro occhio per i dettagli e per le proporzioni.

Altra dote fondamentale è la pazienza: ottenere precisamente l'effetto visuale che si ha in mente richiede in generale molto tempo per aggiustare alla perfezione sia modeli 3D che materiali.

Chiaramente quello che è stato trattato nella tesi è solamente una parte limitata di quello che sta dietro ad un videogioco essendo stati ad esempio completamente omessi la programmazione delle logiche di gioco e l'audio. Questi argomenti sono stati però trattati da Miche Sasso nella tesi "Game Programming: Sviluppo di un Gioco".

Si è parlato invece del *3D modeling* in Blender e di alcuni strumenti che quest'ultimo ci mette a disposizione per rendere più veloce il lavoro e dello shading in Unity3D, portando esempi concreti di materiale prodotto per "Catacomb" . Si è poi presentato Oculus Rift, uno strumento di realtà virtuale che sta conoscendo una buona diffusione nonostante non sia ancora un prodotto finito e che probabilmente cambierà nel corso degli anni come i videogiochi vengono percepiti e vissuti.

Inoltre si è spiegato come sia importante, prima di buttarsi a capofitto nella modellazione e nella creazione di un videogioco in generale, studiare approfonditamente quello che si vuole ottenere cercando immagini e referenze. Questo permette non solo di rendersi conto meglio se quello che si ha in mente possa effettivamente funzionare, ma anche di lavorare meglio e molto più velocemente.

Bibliografia

[1] Roland Hess, *Blender Foundations: The Essential Guide to Learning Blender 2.6*, Focal Press, 2010

[2] *The Blender Wiki*, "http://wiki.blender.org/"

[3] Unity Technologies, *Unity Documentation*, "http://unity3d.com/learn/documentation"