Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

SVILUPPO DI SHADER CON TECNOLOGIA HLSL BASATA SUL MOTORE GRAFICO OGRE 3D

Relatore: *Gianluigi Ciocca*

Relazione della prova finale di:

Ludovico Ceciliani

Matricola 830288

Anno Accademico 2019-2020

Indice

Capitolo 1

Capitolo 2

Capitolo 3

Capitolo 4

Capitolo 5

Capitolo 6

Capitolo 1

Introduzione

L’espressione computer graphics, generalmente abbreviata in CG, venne coniata nel 1960, dai ricercatori Verne Hudson e William Fetter [1]. In quegli anni, con questo termine si indicava in maniera molto generale qualsiasi funzionalità sui computer che non sia testo o suono, riferita esclusivamente al mondo 2D. Negli anni ’80 si ebbe il primo contatto tra la computer grafica e il mondo dell’intrattenimento, campo che oggi è in stretto rapporto con essa. Solo nel 1990, con l’inizio della produzione di lungometraggi animati completamente in 3D, la nascita dei primi videogiochi con grafiche tridimensionali e l’impiego di computer per la generazione di effetti speciali nelle pellicole cinematografiche, si ebbe la svolta definitiva. Ed è proprio questo il concetto che abbiamo oggi giorno di CG, che sviluppa la sua espressione nella branca dell’informatica che studia la manipolazione delle immagini usando la matematica e le tecniche computazionali da essa derivante.

Proprio per questo essa viene sfruttata in molti ambiti, sia professionali che industriali, come nell’industria cinematografica, nel ritocco fotografico, nel mondo videoludico, nella tipografia (impaginazione di giornali e riviste), nel campo della metalmeccanica, elettronica, impiantistica, edile e molti altri.
Un settore che in particolar modo si è evoluto insieme alla computer grafica è la stampa 3D [2]. Questa tecnica, nata nel 1986, permette di realizzare oggetti tridimensionali mediante produzione additiva (processo

industriale per fabbricare oggetti fisici partendo dalla loro rappresentazione digitale aggiungendo uno strato sopra l'altro), partendo da un modello 3D. Essa viene utilizzata in campo industriale per la produzione di pezzi, innesti, protesi utilizzabili in praticamente qualsiasi campo, come un dente, un'opera d'arte o persino abitazioni, come avvenuto in Danimarca.

Al fine di rendere l'immagine più realistica, così da poter avere risultati pratici e precisi, si sviluppano costantemente tecnologie per la realizzazione di effetti sempre più veritieri. Alcuni casi sono lo sviluppo di texture sempre più accurate e con risoluzioni maggiori e tecniche per creare particolari effetti, come profondità, riflessione, illuminazione, ombre e altro.
Gli ultimi due citati sono alla base di una qualsiasi scena digitale, e proprio per questo sono due aspetti estremamente complessi, per i quali esistono molteplici tecniche, ognuna con i propri pregi e i propri difetti. Questo perché per la loro elaborazione si devono considerare vari parametri (fonte di luce, la posizione della camera nella scena, la geometria del modello, …) che spesso bisogna ricalcolare più volte. Per citare un esempio, il ray tracing [3] (la tecnica più moderna) permette di calcolare il percorso della luce analizzando i raggi che partono dal punto di vista della telecamera oppure dalle sorgenti luminose.

Lo scopo di questa tesi è quello di mostrare una tra le varie tecniche di illuminazione e realizzazione delle ombre che si possono utilizzare per migliorare la grafica di una scena digitale, utilizzata in ambito odontoiatrico per la produzione di protesi dentali stampabili attraverso la tecnologia delle stampanti 3D.

1.1 Presentazione del progetto

Questo progetto è stato sviluppando sotto l'azienda CIMsystem [4], la quale si occupa della distribuzione di prodotti per il campo odontoiatrico sfruttando il sistema Pyramis [5] per la loro realizzazione attraverso l'utilizzo di stampanti 3D. Esso si prefigge di rendere esteticamente migliore la scena durante le presentazioni dei prodotti mostrati come modelli 3D attraverso scene digitali. Questo miglioramento è stato effettuato in due fasi: nella prima, si è realizzato un materiale scritto in linguaggio HLSL (High Level Shader Language) da applicare al modello, che permette di modificare i valori delle componenti diffusiva e speculare della luce, adattandosi alla posizione della camera. Nella seconda fase, utilizzando le funzionalità del motore grafico OGRE 3D (Object-Oriented Graphics Rendering Engine), producendo l'ombra dell'oggetto, derivante da una luce diffusiva posta sopra la scena, su una piattaforma utilizzata come base.

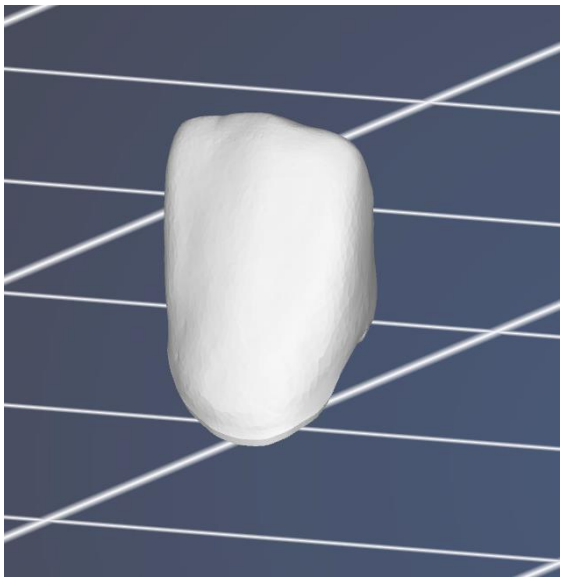

Figura 1.1.1. In questa immagine è visibile il modello di un dente prima dell'applicazione delle modifiche.

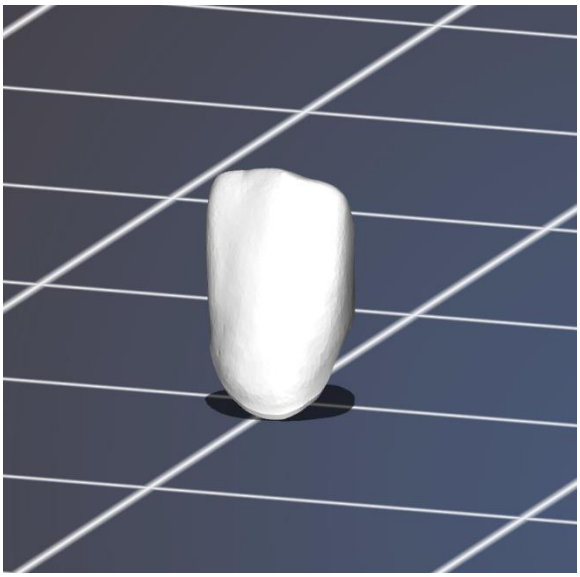

Figura 1.1.2. In questa immagine si può invece vedere lo stesso modello ma con il nuovo materiale e la sua ombra.

1.2 Struttura della tesi

Questa tesi è strutturata nei seguenti capitoli principali:

2. **Tecnologie adottate**

   In questo capitolo verranno descritti gli strumenti principali (Pyramis, OGRE 3D e HLSL) che sono stati utilizzati durante lo svolgimento del progetto, sia sull'aspetto storico che quello tecnico.

3. **Teoria**

   In questo capitolo verrà descritta la teoria alla base del codice e delle soluzioni adottate così da permettere una più chiara comprensione delle scelte effettuate.

4. **Soluzioni adoperate**

   In questo capitolo verranno descritte le soluzioni adottate con le scelte progettuali effettuate per ottenere gli effetti di illuminazione e di ombre cercata.

Capitolo 2

Tecnologie adottate

Nello svolgimento di questo progetto si sono utilizzate diverse tecnologie. Quella più importante è il sistema Pyramis, software sviluppato per venire incontro alle esigenze nate con l'utilizzo delle stampanti 3D in ambito industriale. Esso si basa su diverse applicazioni, la più importante è OGRE 3D, motore grafico in grado di renderizzare una scena digitale modificabile in tempo reale. La realizzazione dei vari effetti visivi dipende dalla scrittura di shader attraverso il linguaggio HLSL, sviluppato sulla tecnologia DirectX di Microsoft.

2.1 Pyramis

Pyramis [5] è un sistema aperto per la gestione della stampa 3D. Esso è in grado di gestire la moltitudine di parametri tecnologici necessari alle varie tecnologie di stampa (SLS [6], sinterizzazione selettiva a laser; SLA [7], stereolitografia; SLM [8], fusione laser selettiva di metalli; DLP [9], elaborazione digitale della luce) e di materiale da stampare con semplicità, utilizzando un'interfaccia utente ottimizzata a seconda delle esigenze.

Figura 2.1.1. Logo del sistema Pyramis.

In base alla scelta della tecnologia in uso, basata sulla stampante 3D a disposizione, Pyramis imposta tutti i parametri in modo totalmente automatico, lasciando oltremodo la possibilità agli utenti “esperti” di intervenire su di essi.

Inoltre, Pyramis utilizza in parallelo i processori delle schede grafiche (GPU), riducendo notevolmente i tempi di calcolo.

Esso è stato implementato basandosi su Visual Studio [10] integrando la libreria Qt [11] per facilitare la realizzazione dell’interfaccia grafica, e il motore di rendering OGRE 3D per la generazione dell’immagine.

2.1.1 Caratteristiche

Pyramis è semplice e intuitivo, molte delle operazioni indispensabili per ottenere il risultato migliore vengono eseguite automaticamente. Durante l’importazione del file del modello, viene effettuata l’analisi dell’oggetto per valutarne la qualità, l’eventuale presenza di imperfezioni, di intersezioni, fori, sovrapposizioni o altri tipi di difetti. Gran parte delle problematiche vengono risolte automaticamente, altre vengono comunque evidenziate affinché si posso intervenire manualmente. Ulteriori strumenti inclusi permettono la gestione dei clusters, il corretto orientamento delle normali e un potentissimo mesh-fix (converte una mesh digitale grezza in una “pulita”, cioè tutte le occorrenze di uno specifico insieme di difetti vengono corrette) che rende la mesh coerente e stampabile.

Ogni volta che viene caricato un oggetto, viene assegnato un profilo di posizionamento e di supporto predefiniti dall’utente. Attraverso

l’interfaccia utente si possono sfruttare più modalità di posizionamento degli oggetti: automatica, manuale e assistita.

- Automatica, con copertura dei punti di minimo, aree in pendenza e dimensioni dei sostegni.
- Manuale, con un singolo click permette di inserire supporti di dimensione e forma configurati.
- Custom, per applicazioni in campi specializzati.

Pyramis mette a diposizione dell’utente diversi strumenti per semplificare la gestione di casi particolari, come quelli utilizzati per la gestione della compensazione in Z e dell’offset 2D, la possibilità di eseguire operazioni booleane e l’utilizzo di una maschera per la correzione delle imperfezioni degli schemi. Inoltre, Pyramis supporta l’importazione di modelli in altri formati, mantenendone le specifiche tolleranze originarie.

Un dispositivo molto potente all’interno di Pyramis è la simulazione 3D, grazie alla quale si può avere una verifica accurata dello slicing (sezionamento, fase intermedia tra la modellazione e la stampa 3D) e hatching (la vera e propria fase di stampa).
Particolare attenzione viene dedicata alla creazione del file di output, che viene messo a punto in accordo coi costruttori in modo da sfruttare al meglio le potenzialità specifiche di ogni singola macchina.
È integrato anche un pannello di comunicazione con la macchina per una gestione real-time integrata.

Pyramis è altamente configurabile e personalizzabile: per l’utente inesperto, che viene assistito e guidato durante l’approccio alla stampa; per il professionista, che percepirà la qualità del sostegno fornito dal

software al suo lavoro; per il produttore di stampante, la cui personalizzazione di aspetto e funzionalità permette la cura dei dettagli.

2.1.2 Tecnologie e tecniche

Il sistema Pyramis è in grado di essere utilizzato con le più diffuse tecnologie esistenti di stampa (DLP, SLA, SLS, SLM). Affiancando queste tecniche a una buona conoscenza dei materiali utilizzati, si può controllare completamente il processo di stampa e quindi gestire al meglio la qualità di produzione.
Inoltre, viene consentita la configurazione di diverse stampanti, dando massima flessibilità all'utente sul suo utilizzo.

Per le tecnologie DLP e SLA si utilizza la tecnica dell'offset materiale (figura 2.1.2 e figura 2.1.3). Essa, in base ai parametri macchina e al materiale di stampa, ingrandisce ogni strato del fattore necessario al fine di ottenere la corretta dimensione dopo la polimerizzazione. L'offset è utilizzato anche per eliminare i piccoli difetti dovuti alla diffusione della luce all'interno delle resine. Questa funzionalità è utile anche per l'ottimizzazione dei parametri di stampa.

L'offset laser viene impiegato nelle tecnologie SLS, SLA e SLM per compensare il raggio dello spot luminoso proiettato dal laser (figura 2.1.4).

La tecnica antiliasing è utilizzata per ottenere una maggiore precisione delle superfici stampate tramite modifiche alla intensità della luce proiettata. La tecnologia DLP sfrutta questa tecnica generando immagini

in sfumature di grigio che permettono di eliminare l'effetto "scaletta" lungo il contorno dell'oggetto, ottenendo così una superficie più liscia.

La scala viene spesso affiancata con l'offset per compensare eventuali modifiche strutturale degli oggetti stampati, generate per la natura dei comportamenti fisico-chimici di resine e materiali, oppure dovute ad eventuali successive esposizioni a calore.

Al fine di agevolare l'adesione ottimale tra gli strati, è necessario che la luce penetri nel livello generato in precedenza, creando coesione col materiale già polimerizzato. Dove la luce, invece, incontra la resina degli strati precedenti non polimerizzati, andrà ad aggiungere materiale non necessario. Per risolvere questo comportamento, il software utilizza la tecnica della compensazione Z, la quale modifica le superfici originali disposte sull'asse Z riducendo l'esposizione delle aree interessate

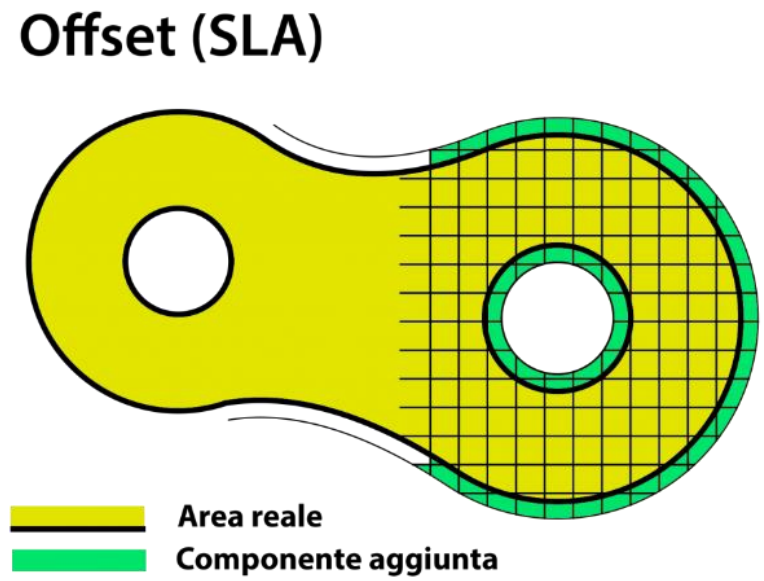

Figura 2.1.2. L'immagine mostra la tecnica dell'offset del materiale per la tecnologia SLA.

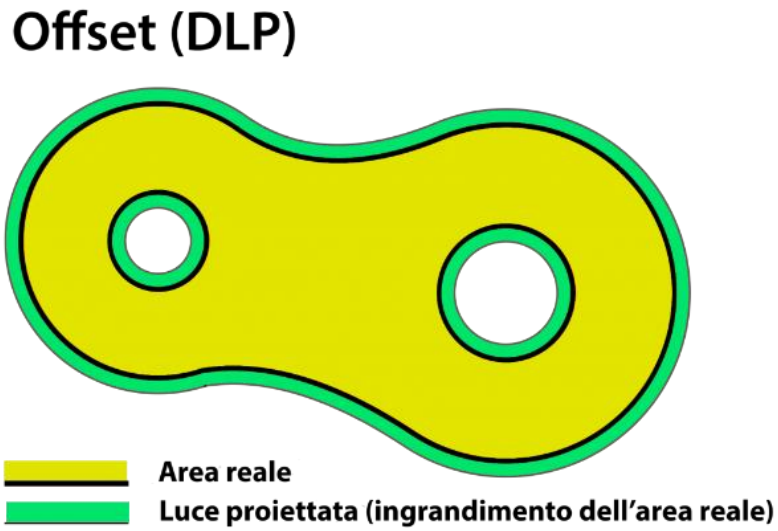

Figura 2.1.3. L'immagine mostra la tecnica dell'offset del materiale per la tecnologia DLP.

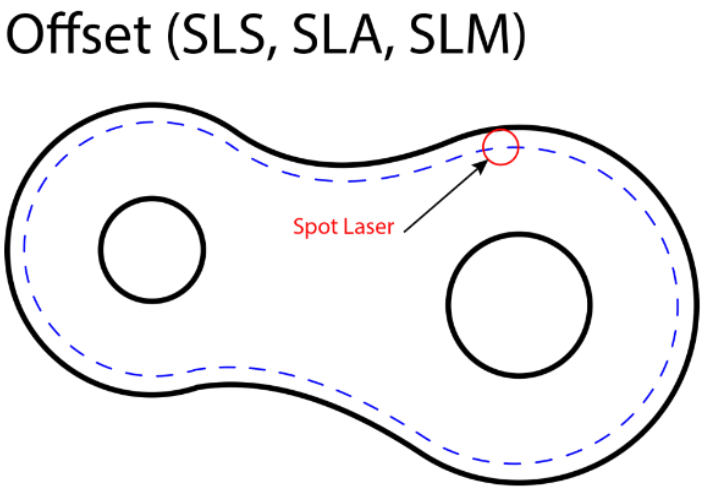

Figura 2.1.4. L'immagine mostra la tecnica dell'offset laser per le tecnologie SLS, SLA e SLM.

2.2 OGRE 3D

OGRE (Object-Oriented Graphics Rendering Engine) [12] è un motore di

Figura 2.2.1 Logo del motore grafico OGRE.

rendering 3D (cioè un software capace di eseguire il rendering di una scena 3D e la sua visualizzazione), flessibile, orientato alla scena e nato nel 2001. È un software libero sotto la licenza MIT (Massachusetts Institute of Technology) con una comunità di utilizzatori molto attiva [13]. È stato il progetto del mese di marzo del 2005 per Sourceforge (piattaforma e sito web che fornisce gli strumenti per portare avanti un progetto di sviluppo software in modo collaborativo tra gli sviluppatori).

Come dice il suo nome, OGRE è “solo” un motore di rendering. Come tale, il suo scopo principale è quello di fornire soluzioni generali per il rendering grafico. Nonostante questo, ci sono alcune strutture dati per la gestione della memoria come vettori e matrici, con il solo scopo di aiuto e semplificazione.

Nel caso in cui qualcuno cercasse una soluzione “tutto in uno” per lo sviluppo di un videogioco o una simulazione bisogna fare attenzione, poiché per esempio non fornisce supporto per l’audio e per la fisica. In generale, si potrebbe pensare che questo sia il maggiore svantaggio per OGRE, ma può essere visto come una caratteristica del motore.

La scelta di rendere OGRE solo un motore grafico lascia agli sviluppatori la libertà di usare le librerie che si desiderano per audio, fisica, input, ….

Il motore è stato utilizzato in un ampio numero di progetti di ambiti diversi, come videogiochi, simulatori, software educazionali, arte interattiva, visualizzazione per scopi scientifici, e altri.

2.2.1 Breve storia di OGRE

- **1999 circa**

  Sinbad (nickname del creatore di OGRE) si rende conto che il suo progetto “DIMClass”, un’idea per creare una libreria per facilitare l’uso delle Direct3D, è diventato talmente astratto da non avere il reale bisogno di basarsi su Direct3D. Inizia così a pianificare una libreria più ambiziosa che può essere indipendente dalle API e dalla piattaforma [14].
- **25 febbraio 2000**

  Viene coniato il nome OGRE e registrato il progetto su Sourceforge. Non viene iniziato subito ilo sviluppo ma si inizia a pensare ad una un’applicazione cross-platform in grado di supportare sia Direct3D che OpenGl.
- **Febbraio 2005**

  Viene rilasciato OGRE v1.0.0 Final “Azathoth”, con una revisione di del sistema delle risorse, aggiunta di hardware pixel buffers, HDR, CECGui e XSI exporter.
- **Marzo 2005**

  OGRE è il “Progetto del Mese” su Sourceforge.
- **Settembre 2005**

  Sinbad annuncia Kadath, un commerciale gestore di scene e esportatore da usare con OGRE, XSI e Blender.
- **31 agosto 2006**

  Viene pubblicato Ankh, primo gioco a utilizzare OGRE.

2.2.2 Caratteristiche generali

OGRE ha una struttura orientata ad oggetti con un'architettura a plugin che consente l'aggiunta di caratteristiche. È un motore basato su scene, con il supporto di vari Scene Managers, come Octree (struttura dati ad albero nella quale ogni nodo interno ha esattamente otto figli) e BSP (Binary Space Partitioning, metodo che permette di suddividere ricorsivamente uno spazio euclideo in sottospazi tramite l'utilizzo di vari piani).

Il motore è completamente multipiattaforma, con il supporto per DirectX e OpenGL [15]. Infatti, esistono dei binari precompilati per Linux, MacOS, e tutte le maggiori versioni di Windows.

OGRE supporta anche shader personalizzati sia con vertex che con fragment program, scritti in vari linguaggi, come GLSL (OpenGL Shading Language), HLSL, Cg (C for Graphics) e linguaggio assembly.

Per la realizzazione di ambienti esterni si ha il supporto per il LOD (Level Of Detail) progressivo che può essere creato automaticamente o manualmente. Questa tecnica consiste nel salvare un modello in una struttura, la mesh progressiva, che permette una scelta più "smooth" per i livelli dei dettagli dipendenti dalla posizione corrente della camera.

Presenta un motore di animazione con il pieno supporto dell'Hardware Weighted Multiple Bone Skinning, che può essere impostato attraverso delle pose per il Full Pose Mixing.

OGRE presenta inoltre un compositing manager per la gestione della posizione delle finestre e i loro confini, con un linguaggio di scripting e postprocessing a schermo intero per effetti come HDR (High Dynamic

Range, tecnica per ampliare l'intervallo tra le aree visibili più chiare e quelle più scure), blooming, saturazione, luminosità, sfumatura e rumore.

È presente anche un sistema per effetti particellari, con rendering estensibile ed effetti personalizzabili.

Le librerie provvedono anche al debugging della memoria ed il caricamento delle risorse dagli archivi.

Il software presenta esportatori di contenuto per la maggior parte dei programmi di modellazione 3D, come 3D Studio Max, Maya, Blender e altri.

Nelle due figure sottostanti si possono vedere due aspetti tecnici di OGRE.

Nella figura 2.2.2 è rappresentata l'architettura del motore come diagramma delle classi, con la classe Root (radice) al centro e tutte le classi Manager che permettono l'accesso a diversi sottosistemi.

La Root ha un riferimento al corrente SceneManager e un enumeratore che permette di caricare altri tipi di scene grafiche.

Il RenderSystem è una classe astratta che separa la radice dalla sua implementazione, da quella di OpendGL o di Direct3D.

Il ResourceManager è anch'essa una classe astratta che viene suddivisa in sottoclassi da una serie di managers che gestiscono le risorse, come textures, materiali e fonts.

Tutte le classi del diagramma, eccetto RenderSystem, SceneManager e RenderWindow, utilizzano un singleton templato per garantire che sia presente una sola istanza di ogni classe.

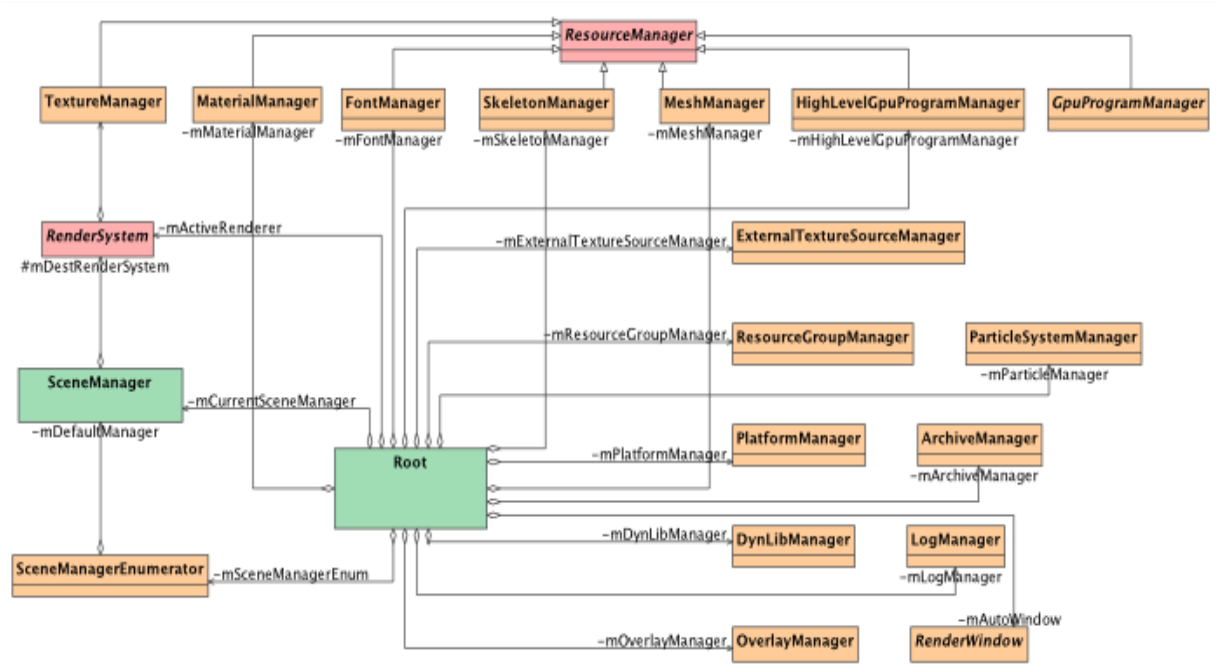

Figura 2.2.2. Questa immagine mostra l'architettura generale di OGRE.

Nella figura 2.2.3 è mostrato il diagramma di sequenza che rappresenta il ciclo di rendering effettuato da OGRE.

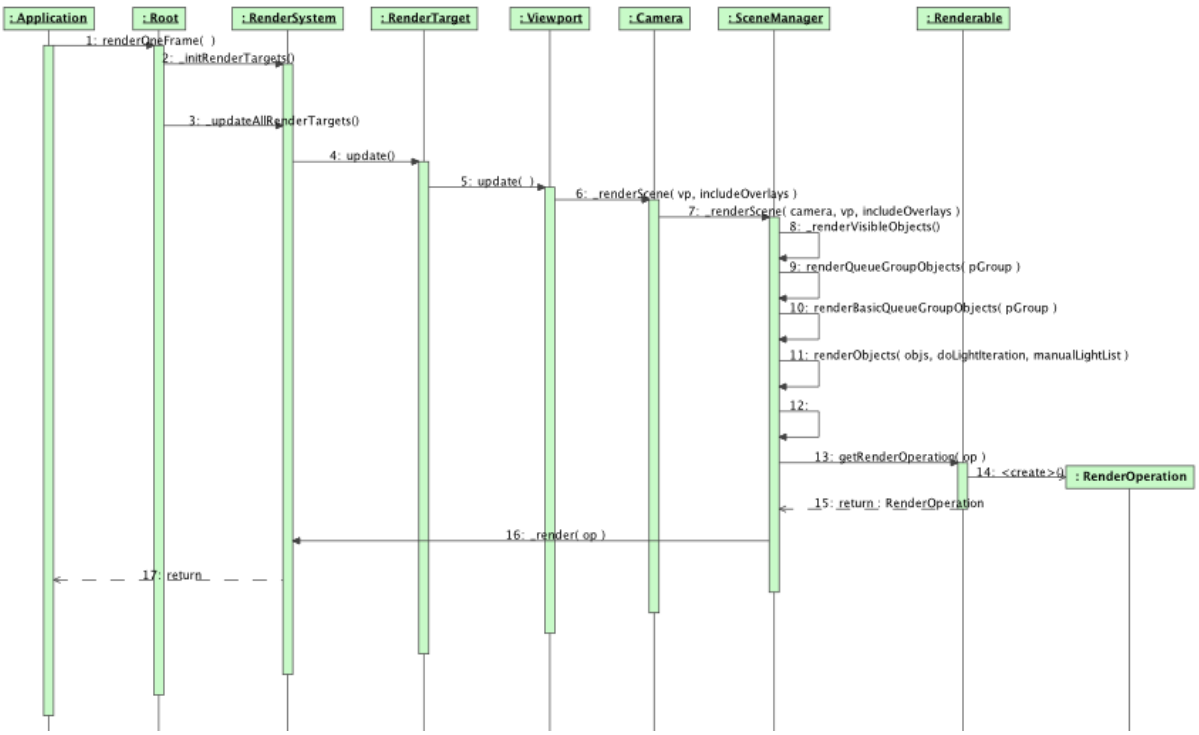

Figura 2.2.3. Questa immagine rappresenta il ciclo di rendering all'interno di OGRE.

2.2.3 OGRE 3D 1.8

Per la realizzazione del progetto si è utilizzata la versione 1.8 di OGRE 3D, chiamata “Byatis” e rilasciata nel maggio del 2012. Oggigiorno esistono due versioni più evolute: OGRE 2.2.4 “Cerberus” e Ogre Next, ognuno con caratteristiche diverse in base alle esigenze dell’utente.

Questa versione supporta l’utilizzo delle Direct3D 9 ed è compatibile con tutti i sistemi operativi principali.

Permette la dichiarazione di materiali all’esterno del proprio codice così da permettere una loro gestione più semplice. OGRE fornisce supporto automatico per molti dei più comuni metodi per parametri costanti, ad esempio le informazioni dello stato della luce oppure le matrici contenenti le informazioni sul mondo.

Il motore permette la gestione di funzioni particolari come l’utilizzo di textures multiple e la generazione e modifica di coordinate della texture. Si ha la possibilità di eseguire diverse iterazioni per avere più effetti così come il supporto a più materiali scelti automaticamente da OGRE in base al migliore in un determinato momento.

Si possono utilizzare texture in formati PNG, JPEG, TGA, BMP, PVRTC, KTX e DDS, inclusi alcuni cosa particolari come texture a una dimensione, cubemaps e textures compresse. Le texture possono inoltre venire fornite e aggiornate in real-time.

È presente una gestione altamente modificabile e flessibile attraverso classi predefinite per l’organizzazione della scena. Attraverso un grafico

di scena gerarchico si possono utilizzare nodi per unire più oggetti e permettere che si seguano durante i loro movimenti. Si ha la possibilità di utilizzare vari effetti speciali gestiti interamente da OGRE, come skybox, skyplanes, skydomes, la trasparenza degli oggetti, la creazione di luci di vario tipo ed effetti particellari. Supporta varie tecniche per la renderizzazione di ombre, sia modulative che adattative, basate su texture o tecnologia stencil (si utilizza per generarle attraverso uno shadow volume).

2.3 HLSL

High Level Shading Language (HLSL) è un linguaggio sviluppato da Microsoft [16] per la creazione di shader da usare in DirectX, ed è molto simile al linguaggio Cg di NVIDIA. In questo progetto si usufruirà della sua versione compatibile con DirectX9.

Esso ha una sintassi simile a C, attraverso la quale si possono scrivere complessi calcoli grafici che possono essere eseguiti molto velocemente dalla GPU. Inoltre, rappresenta il primo passo della pipeline grafica programmabile rendendo quest'ultima interamente programmabile grazie al linguaggio.

Il suo più grande vantaggio è la possibilità di non doversi basare su altre librerie, gli basta unicamente quella di DirectX.

Uno shader in HLSL è formato da tecniche (technique), che a loro volta sono formate da vari passi (pass) [17]. In quest'ultimo si specificano le funzioni che devono essere utilizzate e quale versione dello shader considerare.

Per utilizzare lo shader bisogna inoltre dichiarare un vertex_program e un fragment_program. Questa parte può essere scritta direttamente all'interno del file con estensione material che si vuole utilizzare. Essa serve per indicare quale documento bisogna prende di riferimento per avere lo shader desiderato, quale funzione al suo interno considerare e se le operazioni saranno sui vertex o sui pixel. Di seguito è mostrato un esempio molto semplice di uno script in formato HLSL:

```
// Questo è un commento
vertex_program sample_vs hlsl
{
        source nome_file.hlsl                   // Nome del file in cui è scritto lo shader
        target vs_3_0                           // Per indicare di eseguire le operazioni sui
                                                vertici
        entry_point nome_funzione_vertici       // Quale funzione all'interno del file
                                                considerare
        default _params
        {
                // Qualsiasi parametro richiesto dalla funzione
        }
}

// Le stesse osservazioni valgono anche nel caso in cui si considerano i pixel
fragment_program sample_ps hlsl
{
        source nome_file.hlsl
        target ps_3_0
        entry_point nome_funzione_pixel
```

```
        default _params
        {
                …
        }
}

material SAMPLE
{
        technique                                       // Si dichiara la prima tecnica
        {
                pass                                    // Si dichiara il primo passo
                {
                        vertex_program_ref sample_vs {}         // Riferimento alla parte per i vertici
                        fragment_program_ref sample_ps {}       // Riferimento alla parte per i pixel
                        texture_unit                            // Blocco per l'utilizzo di texture
                        {
                                // Istruzioni per l'utilizzo di una singola texture
                        }
                }
        }
}
```

Capitolo 3

Teoria

In questo capitolo viene trattata la teoria sulla quale si basano le soluzioni adoperate in questo progetto. In particolar modo si mostra in che maniera la luce viene analizzata e calcolata in computer grafica e le sue componenti, e le varie tecniche con le quale si possono generare le ombre, ognuna con i propri vantaggi e svantaggi.

3.1 Luce

La fase di lighting (illuminazione) all'interno di una scena digitale è un aspetto parimenti importante e complesso, grazie alla quale si riesce a produrre elaborati che si avvicinano molto alla realtà. Con essa si determina quanta luce viene riflessa verso la camera della scena da ciascun vertice dei triangoli che compongono la mesh del modello.
Inoltre, è strettamente legata alla fase di shading (ombreggiatura) che valuta in che modo la luce è riflessa per ciascun punto interno al triangolo.Alla base di questi due momenti della pipeline grafica deve esserci la presenza una sorgente di luce all'interno di scena. Essa può essere di diversi tipi, in base all'effetto che si vuole emulare. Il motore grafico OGRE ne supporta tre tipologie [18]: point (figura 3.1.1), spotlight (figura 3.1.2) e directional (figura 3.1.3). La prima emette luce in tutte le direzioni dal punto in cui viene posta, la seconda simula l'effetto di una

torcia elettrica, la terza ricrea l'illuminazione presente durante il giorno o con la luna.

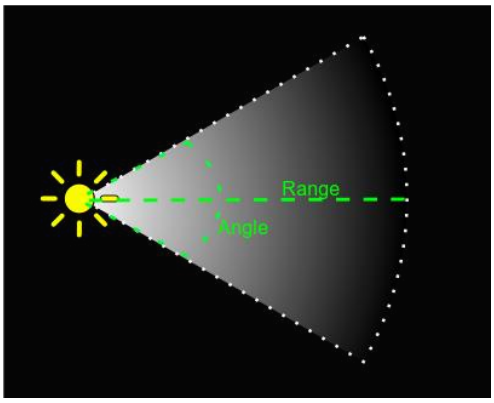

Figura 3.1.2. Rappresentazione di una sorgente luminosa di tipo "spotlight". Essa crea un cono di luce dipendente da un angolo. Anche in questo caso l'intensità dipende dalla distanza.

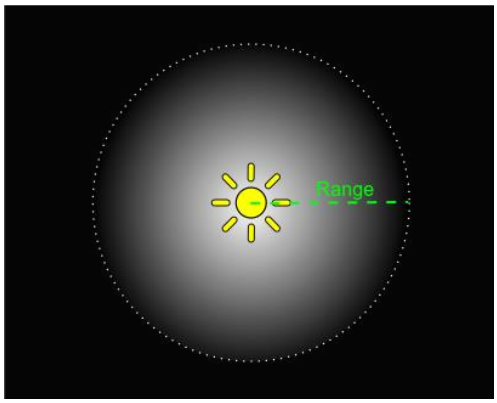

Figura 3.1.1. Rappresentazione di una fonte luminosa di tipo "point". Essa crea una luce uniforme in tutte le direzioni con intensità di variabile in base alla distanza dalla sorgente.

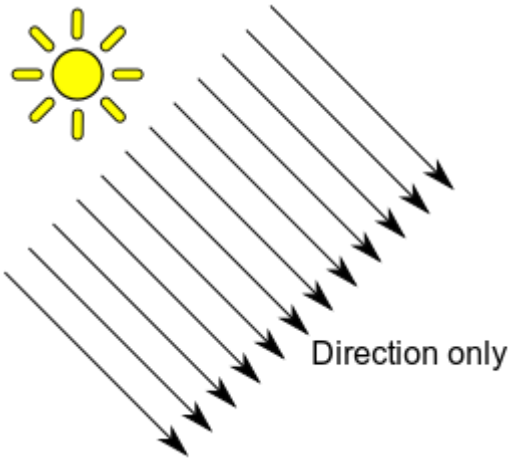

Figura 3.1.3. Rappresentazione di una fonte di luce di tipo "directional". Con essa si simula un effetto giorno, perciò si considera solamente la direzione dei raggi luminosi.

La luce presente all'interno di una scena può essere suddivisa in due categorie: luce locale, con la quale si considera il contributo luminoso

diretto sulla superficie; e luce globale che considera il contributo di luce indiretto, cioè quello proveniente dalle altre superfici.

Il colore di un oggetto viene dato dall'interazione tra la luce che lo illumina e il suo spettro di riflettanza. Questo permette di studiare come, fisicamente, una superficie interagisce con i raggi luminosi in base alle diverse lunghezze d'onda. In base ad esse, si è in grado di vedere un determinato colore dipendente da quella che è in grado di riflettere. La figura 3.1.4 mostra la composizione della luce in base ai valori delle lunghezze d'onda.

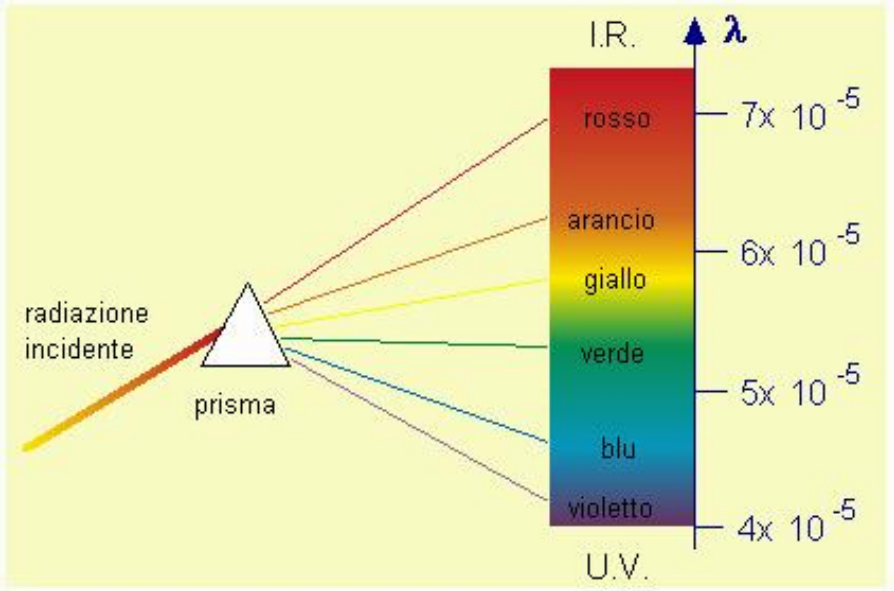

Figura 3.1.4. In questa immagine si mostrano le varie lunghezza d'onda che compongono la luce con i loro rispettivi intervalli. Con la sigla U.V. si indicano le onde che ricadono nel campo dell'ultravioletto, mentre con I.R. si indica il campo dell'infrarosso.

In computer grafica, le lunghezze d'onda che si considerano principalmente sono rossa, blu e verde. In questo modo si sfrutta il modello RGB (Red-Green-Blue) [19]. È uno schema di tipo adattativo che considera questi tre colori in calori percentuali, con la cui variazione si possono generare tutte le gradazioni possibili (figura 3.1.5). Si può utilizzare un ulteriore parametro chiamato alpha (indicato come a) per indicare l'opacità.

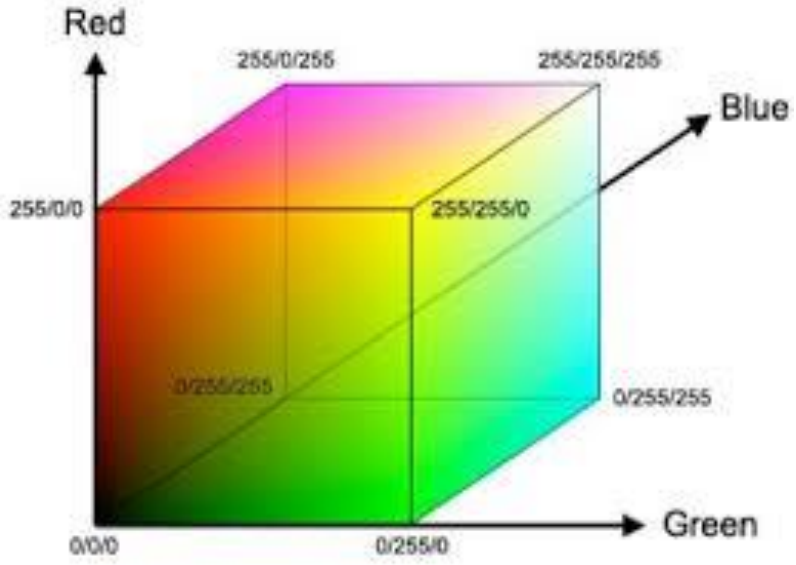

Figura 3.1.5. Questa immagine rappresenta il modello RGB. Inoltre, si mostra il meccanismo delle triplette per avere il colore che si desidera modificando i valori in un intervallo tra 0 e 255.

3.1.1 Luce locale

Per calcolare la luce globale all'interno di una scena in computer grafica si deve studiare l'intensità luminosa. Essa dipende dalla lunghezza d'onda e per questo si utilizza il modello RGB per semplificare le operazioni a soltanto tre colori, il rosso, il verde e il blu. Con la somma di questi tre addendi si trova l'intensità totale prodotta dalla sorgente luminosa.

Inoltre, la luce locale viene scomposta in quattro componenti. Ognuna di esse viene calcolata quasi sempre indipendentemente dalle altre e presenta specifiche proprietà.

La prima componente viene chiamata emissiva. Con essa si identifica la luce emessa dalla superficie, influenzando il colore e facendo sembrare che l'oggetto brilli di luce propria. Per questo motivo dipende soltanto dal materiale utilizzato e non dalla geometria del modello. La figura 3.1.6 mostra un esempio di questa componente rispetto a una luce con lunghezza d'onda nel campo del verde.

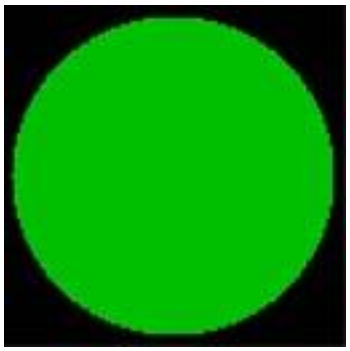

Figura 3.1.6. Esempio di applicazione della componente emissiva della luce.

Essa viene calcolata considerando l'intensità luminosa massima emettibile dal materiale e una costante emissiva avente intervallo tra 0 e 1 compresi:

$$I_{emissiva} = k_{emissiva} * I_{max} \; con \; k \in [0,1]$$

In realtà, le altre tre componenti sono quelle fondamentali che verranno considerate all'interno di questo progetto. Queste sono divise in ambientale, diffusiva e speculare. Unendo solo queste tre si possono raggiungere risultati completi e precisi, come ci mostra la figura 3.1.7.

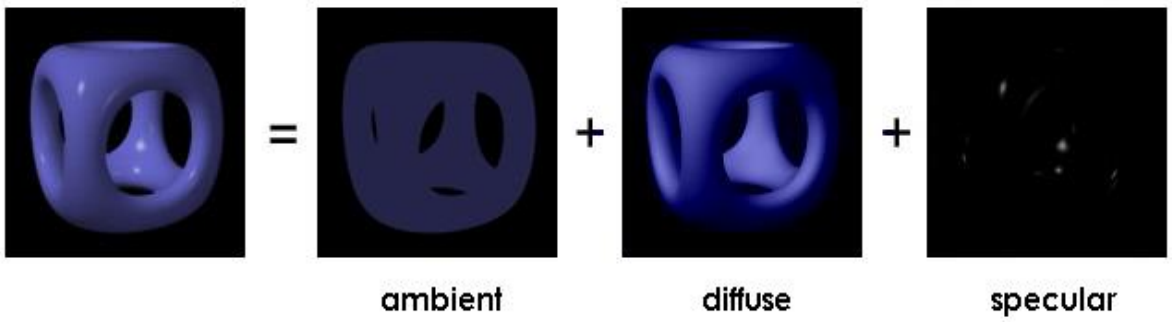

Figura 3.1.7. Con questa immagine si può notare come con l'unione della componente ambientale, diffusiva e speculare si possano raggiungere ottimi risultati.

La componente ambientale è quella che permette di generare una luce omnidirezionale in grado di illuminare tutte le superfici in maniera uniforme. Per questo motivo essa è indipendente dalla geometria del modello.

Essa viene calcolata come il prodotto tra una costante, chiamata riflettanza del materiale, per la luce ambientale e l'intensità di essa:

$$I_{ambientale\ della\ superficie} = k_{ambientale} * I_{ambientale}$$

La componente diffusiva indica la quantità di luce che viene riflessa dalla superficie. Grazie ad essa si possono generare le ombreggiature.

L'intensità della luce riflessa è uguale in tutte le direzioni, per questo il fattore che influisce su di essa è la direzione di riflessione. Questo valore lo si può calcolare in due modi: il primo considerandolo come l'angolo tra la normale della superficie e il vettore della luce incidente, il secondo come il prodotto scalare tra il versore della normale e quello della luce incidente. Inoltre, se si assume il caso in cui questo prodotto risulta negativo si ha la superficie in ombra, e quindi bisogna prendere il massimo tra 0 e il valore ricavato. Tutto questo ci fa capire che la componente diffusiva dipende non solo dal materiale ma anche dalla geometria dell'oggetto.

In conclusione, si è in grado di calcolare questo fattore attraverso la seguente formula:

$$I_{diffusiva} = k_{diffusiva} * I_{luce\ incidente} * \cos\theta =$$

$$= k_{diffusiva} * I_{luce\ incidente} * \left(\hat{N} \cdot \hat{L}\right) =$$

$$= k_{diffusiva} * I_{luce\ incidente} * \max\left\{0, \hat{N} \cdot \hat{L}\right\}$$

L’ultima componente è quella speculare. Essa è, come quella diffusiva, la quantità di luce che viene riflessa dalla superficie data una luce incidente. La differenza sta nel fatto che questa quantità non è uniforme in tutte le direzioni, andando a generare una direzione preferenziale in cui si ha un picco maggiore di luce emessa. Per questo motivo nel calcolo di questa componente si deve tener conto anche del punto di vista.

Per riuscire a calcolare questa componente si possono utilizzare varie tecniche. La più intuibile è attraverso il calcolo del coseno dell’angolo sotteso il vettore di riflessione e quello di vista, ma esistono tecniche più economiche e precise come il modello di luce speculare di Blinn-Phong che verrà trattato nella prossima sezione.

L’ultimo elemento da considerare è il fatto che la luce possa venir attenuata. Con lo scopo di risolvere questo caso, si utilizza un fattore di attenuazione dipendente dalla luce presente e dalla geometria dell’oggetto.

3.1.2 Modello di luce speculare di Blinn-Phong

Il modello di luce speculare di Blinn-Phong [20] è una modifica del modello di riflessione di Phong. Questi due metodi consistono nel calcolare la luce speculare grazie alle direzioni dei vettori presenti nella scena.

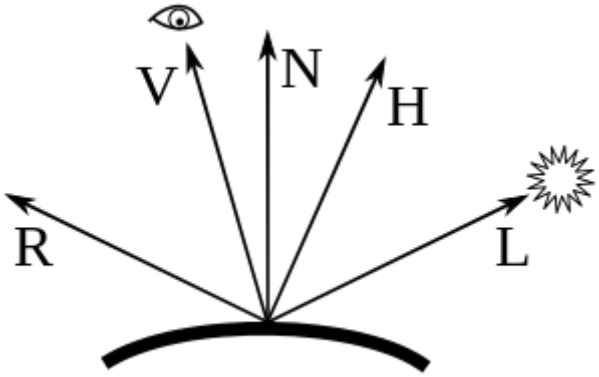

Figura 3.1.8. In questa immagine è rappresentata la situazione vettoriale presente durante lo studio della componente speculare.

La formula di Phong permette di ottenere la componente speculare attraverso il prodotto scalare tra il vettore di riflessione R e la normale N della superficie. Lo svantaggio di questa tecnica è come ottenere il vettore R essendo computazionalmente oneroso. Per ovviare questo problema si considera il vettore medio H tra la direzione del punto di vista V e il raggio incidente L. In questo modo si ottiene la formula

$$I_{speculare} = k_{speculare} \times I_{luce\ speculare} \times \left(\widehat{H} \cdot \widehat{N}\right)^{n}$$

La presenza dell'esponente *n* serve per modulare gli spike di luce sulle superfici. Più è alto questo valore più essi saranno concentrati. Spesso il valore di $I_{luce\ speculare}$ coincide con l'intensità luminoso della componente diffusiva.

L'algoritmo di Blinn-Phong risulta più veloce rispetto a quello di Phong nel caso in cui l'osservatore e la luce sono posizionati molto distanti, come nei casi di luci di tipo directional e camere ortografiche/isometriche. Infatti, in questi casi il vettore medio si può considerare costante visto che sia la direzione dell'osservatore che quella del raggio di incidenza convergono individualmente alle distanze remote di queste circostanze. Inoltre, il

vettore H può essere calcolato una volta per ogni luce per poi essere usata per l'intero frame, o nel caso in cui la luce e il punto di vista rimangono relativamente nella stessa posizione.

Nella figura 3.1.9 si nota che le due tecniche producono risultati molto simili tra di loro e di come, variando il valore dell'esponente, cambi l'effetto finale.

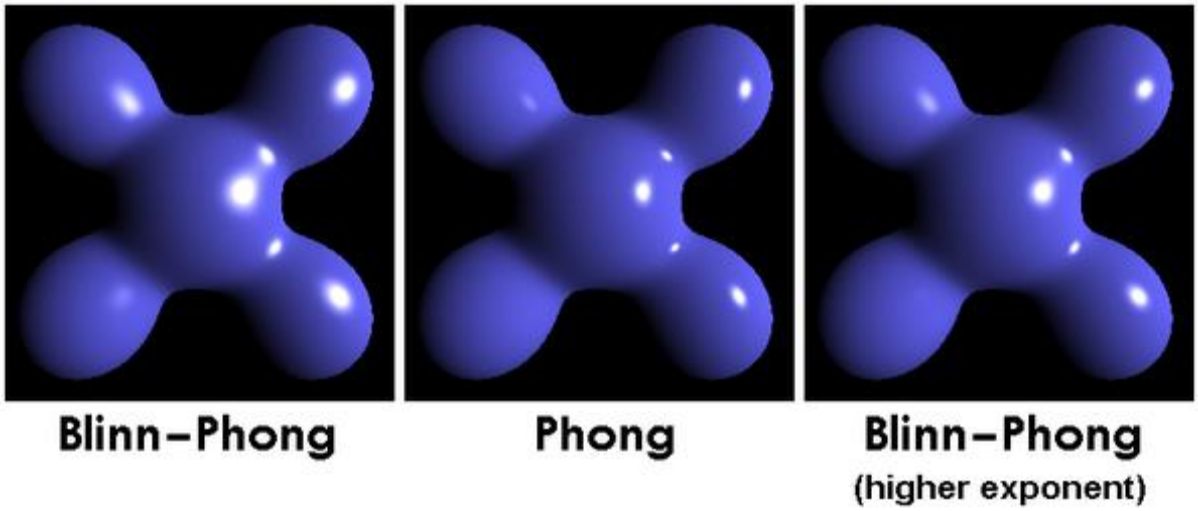

Figura 3.1.9. Nell'immagine si può vedere lo stesso modello realizzato con tecniche diverse per calcolare la luce speculare.

3.1.3 Luce globale

Nello studiare la luce globale non si considerano più i vertici dei triangoli della mesh ma i punti interni ad essi. Per calcolare la loro influenza si utilizzano gli algoritmi di shading. La figura 3.1.10 mostra l'effetto che questo tipo di luce produce sugli oggetti utilizzando diversi algoritmi.

Figura 3.1.10. Con questa immagine si possono notare le diverse tecniche di shading applicate ad uno stesso modello

Il Flat shading è ottimo per realizzare superfici spigolose. Questo perché l'algoritmo assegna lo stesso valore di intensità di luce emessa da ogni punto della superficie. Esso consiste nel calcolare la normale di ogni faccia e di seguito l'intensità luminosa usando l'equazione di luce locale. Questo valore viene così attribuito a tutti i punti, rendendo molto visibili le diverse facce presenti nel modello 3D, creando delle discontinuità.

Per realizzare superfici più lisce, esistono due algoritmi: il Gourand shading e il Phong shading.

Il primo calcola le normali di ogni vertice di ogni faccia, considerando le normali di quelli condivisi come la media delle facce incidenti. Di seguito viene determinato il contributo di luce associandolo ad ogni vertice. Infine, si ottengono i punti interni per interpolazione. Il problema di questo shader è nella renderizzazione degli spike di luce sulle superfici lucide.

Il Phong shading è molto simile. Vengono calcolate le normali dei vertici ma a differenza del Gourand shading vengono trovate le normali dei punti interni per interpolazione. In questo modo è possibile determinare il colore del punto localmente così da riuscire anche a rendere visibili l'effetto della luce speculare su superfici lucide.

3.2 Ombre

La presenza di ombre all'interno di una scena digitale è molto importante e allo stesso tempo molto complessa. Per riuscire ad ottenere il risultato migliore OGRE supporta diverse tecniche per implementare le ombre, ognuna con i propri vantaggi e svantaggi, così da poter ottenere il miglior risultato voluto.

Esistono due categorie base: Stencil Shadows e Texture-based Shadows. Inoltre, esistono due metodi per renderizzare le ombre, le Modulative Shadows che rendono la scena più scura nelle zone in ombra, e l'Additive Light Masking con il quale aumenta il contributo di luce nelle aree non in ombra.

Un'altra opzione sono le Integreted Texture Shadows grazie alle quali si ha controllo completo sulle shadow texture utilizzate dall'applicazione [21].

3.2.1 Stencil Shadows

Stencil Shadows è un metodo con cui si genera una "maschera" per la scena usando uno stencil buffer. Questa può essere usata per escludere le aree dalle fasi di rendering successive, e può essere sfruttata anche per includere o escludere le zone in ombra. Essendo quindi possibile solamente scegliere se zone "attive" o "disattivate", si generano delle ombre nette, con i bordi accentuati.

Questa tecnica si basa su uno shadow volume che divide la scena digitale in due aree, quelle in ombra e quelle no. Esse vengono renderizzate escludendo la sagoma del shadow caster (l'oggetto che produce l'ombra) dalla luce. Dove questi shadow volumes intersecano altri oggetti, lo stencil buffer viene aggiornato, così da permettere alle operazioni seguenti di distinguere tra luci e ombre. L'esatto procedimento con cui avviene il rendering delle ombre dipende comunque da quale metodo viene utilizzato tra Modulative Shadows e Additive Light Masking.

Grazie alle Stencil Shadows si ha la possibilità che un oggetto possa sia ricevere che generare ombre, permettendo così l'auto-ombreggiatura.

Il vantaggio di questa tecnica è la possibilità di avere una semplice auto-ombreggiatura utilizzando anche hardware di fascia bassa, permettendo di tenere sotto controllo il proprio numero di poligoni, a differenza delle Texture-based Shadows che richiedono macchine più moderne.

Gli svantaggi sono però numerosi, specialmente su hardware moderni. Questo perché essendo una tecnica geometrica, più è alto il numero di poligono più sarà elevato il costo computazionale, penalizzando dettagli presenti nelle mesh utilizzate. Per questo motivo si può avere un effetto a collo di bottiglia, con la creazione di ombre nette senza avere la possibilità di modificarle.

Oltre a questi svantaggi, esistono vari problemi specifici. Uno di questi è la mancata disponibilità dei vertex programs, OGRE può soltanto escludere gli shadow volumes a una distanza finita dall'oggetto. Quindi, se un oggetto si trova troppo vicino al una luce, non si ha una distanza

sufficiente per garantire a tutti gli oggetti di venire ombreggiati adeguatamente dall'oggetto.

Un altro problema si ha nel posizionamento del far plane della camera. Infatti, il motore grafico cambia le opzioni del far plane internamente per far sì che esso non venga più considerato (viene semplicemente posizionato all'infinito). In questo modo si evitano artefatti causati dal clipping delle zone oscure alla fine delle linee disegnate sui shadows volume, al costo di una piccola parte della precisione della profondità.

Inoltre, questa tecnica può essere utilizzata solo quando una lista di bordi (edge-list) è stata generata per tutta la geometria di una mesh. I tools officiali e gli esportatori lo fanno automaticamente, ma se si crea una mesh personale bisogna fare attenzione nell'utilizzarla con le Stencil Shadows, altrimenti OGRE assume che quel particolare modello non deve generare ombre.

3.2.2 Texture-based Shadows

La tecnica Texture-based Shadows è, come indica il nome, basata sull'utilizzo di texture, nella quale vengono salvate le informazioni per renderizzare le ombre prodotte dagli oggetti caster dal punto di vista della luce e visibili sugli oggetti chiamati receivers. Il vantaggio principale è il basso aumento computazionale nel caso dell'incremento dei dettagli nella geometria, non essendo necessario eseguire calcoli per ogni triangolo della mesh.

Molto del lavoro di rendering eseguito con le texture shadows è fatto dalle schede grafiche, rendendo così questo metodo estremamente vantaggioso nell'utilizzo di quelle più moderne. Inoltre, sono molto personalizzabili, con la possibilità di utilizzarle all'interno di shaders dove se ne ha bisogno.

Lo svantaggio principale delle texture shadows è, essendo delle texture, l'avere una risoluzione fissa, rendendo così i pixel molto distinguibili nel caso in cui essa venga allungata. Per contrastare questo effetto esistono diverse maniere.

Una di queste è la scelta di una base di proiezione (projection basis). La più semplice proiezione avviene elaborando le ombre dalle prospettive delle luci in scena attraverso opzioni regolari della camera. Questo potrebbe produrre brutti effetti visivi, e per questo OGRE fornisce varie opzioni di projection basis applicabili alla shadow camera (camera utilizzata per la realizzazione della Texture Shadow):

- *Uniform*, la più semplice;
- *Uniform Focussed*, una normale camera di proiezione con l'eccezione di concentrarsi nell'area che la principale camera di vista (view camera) sta guardando;
- *Light Space Perspective Shadow Mapping* (LiSPSM), che si concentra e distorce lo shadow frustum basato sulla view camera;
- *Plane Optimal*, la quale cerca di ottimizzare le ombre fedelmente per un singolo receiver plane (piano sul quale vengono poste le ombre);
- *Parallel Split Shadow Mapping,* tecnica introdotta con la versione 1.6 di OGRE, permette di alleviare il problema della distribuzione dei

texels della shadowmap e i pixel su schermo dividendo il tronco di vista in blocchi parallelli lungo l'asse Z.

Un altro metodo è il filtering. Esso consiste nel semplice fatto di utilizzare la shadow texture più volte per creare ombre con bordi più smussati. Si utilizza come approccio il Percentage Closest Filtering (PCF) che permette di utilizzare diverse variabili dipendenti dal numero e dal pattern dei campioni che vengono presi.

Una tecnica molto semplice per avere un risultato migliore è di utilizzare una texture più grande, sfruttando la GPU veloce con un aumento della memoria.

Un ulteriore problema è con l'utilizzo delle point lights. Questo perché le texture shadows richiedono di renderizzare la texture nella direzione della luce, così che questo tipo di luci omnidirezionali richiedono sei elaborazioni per coprire tutte le direzioni che le ombre possano creare. Per questa ragione, OGRE supporta principalmente luci direzionali e spotlight; si possono però utilizzare le point lights se posizionate fuori dalla camera rendendole come riflettori.

OGRE permette di poter gestire personalmente vari attributi per ottenere il risultato voluto. Alcuni esempi sono la possibilità di scegliere il numero massimo di texture da utilizzare, la grandezza della texture, la distanza in cui le ombre terminano e personalizzare la shadow camera utilizzata.

Nell'utilizzo di questa tecnica si deve dichiarare per ogni singolo oggetto della scena se esso funge da caster o da receiver. OGRE setta

automaticamente gli oggetti come caster, ma lascia la possibilità all'utente di farlo personalmente.

Un metodo diverso che si basa comunque sull'utilizzo di texture è l'Integrated Texture Shadows. Con questa tecnica si può andare direttamente nello shader di un modello a impostare le opzioni con cui utilizzare le shadow texture, dando responsabilità al programmatore di impostare le operazioni da eseguire per ottenere il risultato voluto.

3.2.3 Modulative Shadows

Modulative Shadows è la tecnica che produce ombre oscurando una scena già renderizzata con un colore deciso dallo sviluppatore. Prima, viene per l'appunto elaborata la scena con i modelli che devono essere ombreggiati normalmente, di seguito si esegue un passo modulativo per ogni luce, così da oscurare le aree in buie. Infine, si renderizzano gli oggetti che non devono ricevere ombre.

Questa tecnica produce un modello luminoso inaccurato, visto che rende le zone in ombra più buie uniformemente, indipendentemente dalla quantità luminosa che sarebbe comunque presente in quell'area.

Comunque, con le Modulative Shadows si ha un buon risultato con un costo computazionale molto inferiore rispetto a metodi più "corretti", come l'Additive Light Masking, oltre a combinarsi bene anche con l'utilizzo di mappe di luci precalcolate che le luci adattative non fanno.

Il fatto principale da considerare è che l'utilizzo di più fonti luminose può provocare delle ombre eccessivamente scure (nei punti in cui le ombre si sovrappongono, fenomeno intuitivamente giusto ma errato fisicamente) e artefatti quando questo metodo viene utilizzato insieme alle stencil shadows.

3.2.4 Additive Light Masking

Con la tecnica Additive Light Masking si renderizza la scena molte volte, ognuna che rappresenta un singolo contributo di luce la cui influenza viene mascherata nelle aree in ombra. Ogni passo viene combinato con quello precedente così che quando si completano le operazioni, tutti i contributi luminosi sono stati accumulati correttamente nella scena, e ad ogni luce viene impedito di influenzare aree che non dovrebbero essere in grado di colpire per via delle ombre. È un metodo molto efficace che produce risultati molto realistici al prezzo di diversi passi di rendering.

OGRE libera l'utente da molto del lavoro e permette di usare lo stesso materiale sia nel caso in cui si utilizza questa tecnica o no.

Per usufruire dell'Additive Light Masking, il motore grafico categorizza automaticamente i passaggi definiti nel materiale in tre tipologie:

1. *Ambient*. Con questo passo si includono tutti i passaggi base che non usano nessuna luce particolare, ossia quelli che si verificano anche nel caso in cui non sia presente una luce ambientale. Questa

parte avviene sempre per prima e setta il valore iniziale della profondità e il colore della luce ambientale se applicabile. Inoltre, include qualunque contributo di luce emissiva e auto illuminazione. Solamente le texture che influenzano la luce ambientale vengono renderizzate in questo passo, come le ambient occlusion maps.

2. *Diffusive/Specular*. Viene calcolata la componente diffusiva e speculare per ogni luce, ed ogni passo contribuisce al colore di queste due componenti luminose da parte di una singola luce come riflessione. Le aree on ombra da questa luce sono mascherate e quindi non più aggiornate. La maschera del colore così ottenuta viene aggiunta al colore presente nella scena. Come per il passo precedente, non vengono utilizzate textures oltre a quelle utili per il calcolo dell'illuminazione come le normal maps.
3. *Decal*. In questo passo si aggiunge la texture finale del colore alla scena, la quale viene modulata dalla luce accumulata calcolata durante i passaggi precedenti.

Il limite della Additive Light Mask è di non poter venire combinata bene con le luci statiche precalcolate della scena. Questo perché sono basate sul principio che l'ombra è l'assenza di luce, ma visto che le luci statiche includono di per sé le zone di luce e buio, la tecnica non può rimuovere l'illuminazione per creare nuove ombre. Si ha però la possibilità di utilizzare questo metodo esclusivamente come soluzione per l'illuminamento, altrimenti si devono utilizzare le Integrated Texture Shadows per combinare le luci statiche a seconda dell'approccio che si è scelto.

Capitolo 4

Soluzioni adoperate

La parte pratica di questo progetto è avvenuta in due fasi. Nella prima si è scritto uno shader in grado di calcolare la luce locale per ogni modello presente nella scena sfruttando il modello di luce speculare di Blinn-Phong (l'algoritmo scritto si basa su uno presente nella sezione Tutorial di OGRE [22]). Per la seconda parte vengono utilizzate le potenzialità del motore grafico OGRE 3D applicando la tecnica per le ombre basate sulle texture di tipo modulativo.

4.1 Shader per l'illuminazione

La realizzazione dello shader per calcolare l'illuminazione locale di un modello è stata fatta all'interno di due file: uno con estensione material con il quale si sono dichiarati i parametri da utilizzare durante i calcoli e il documento hlsl, in cui sono presenti le varie operazioni eseguiti. Per il calcolo della luce speculare si è utilizzata la formula di Blinn-Phong, con la possibilità di scegliere sia il suo colore sia quello della componente diffusiva.

Lo shader viene utilizzato come materiale per tutti i modelli della scena che vengono importati dall'utente, di conseguenza esso è in grado di fornire buoni risultati indipendentemente dalla geometria degli oggetti.

Nelle immagini seguenti si può notare effettivamente come cambi l'aspetto dei modelli utilizzando un materiale base oppure quello realizzato dallo shader.

Figura 4.1.1. Questa immagine mostra un modello diverso rispetto a quello mostrato nella parte introduttiva realizzato con un materiale di base. Si può notare che l'oggetto possiede un colore molto uniforme, con un'auto-ombreggiatura al minimo e la mancanza della luce speculare. Questo perché per la realizzazione del materiale si sono utilizzati soltanto le istruzioni base per l'impostazione del colore, scegliendo personalmente i valori delle triplette RGB per le varie componenti della luce.

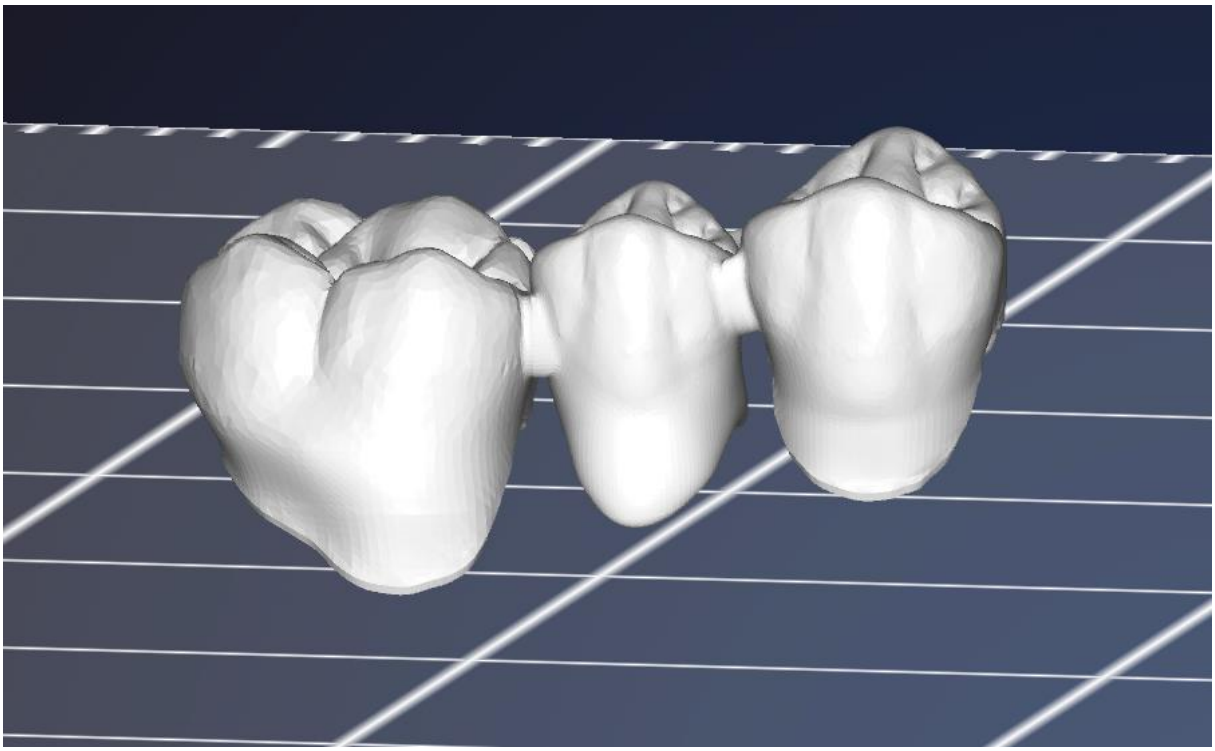

Figura 4.1.2. In questa figura è mostrato lo stesso modello di quella sopra ma con il materiale prodotto grazie allo shader realizzato per l'illuminazione. A differenza dell'immagine precedente, si nota la presenza della componente speculare che fornisce un effetto di riflessione sul materiale. Inoltre, visto che lo shader analizza punto per punto, si ha un'auto-ombreggiatura migliore e il colore non risulta più così uniforme.

4.1.1 Documento material

In questo file sono presenti le dichiarazioni e le definizioni del vertex program, fragment program e il materiale. Grazie ad essi si è in grado di ottenere il materiale voluto attraverso le istruzioni all'interno del documento hlsl, proclamando i parametri globali da considerare.

Nella prima parte si prepara l'ambiente per l'esecuzione dei calcoli rispetto ai vertici delle mesh. Si inizia dichiarando il documento con estensione hlsl

nel quale è presente la funzione utilizzata per le operazioni, e il nome della stessa.

Inoltre, si dichiara il bersaglio (i vertici) su cui eseguire le istruzioni e, attraverso le direttive per il preprocessore, il numero massimo di luci che vengono considerate all'interno della scena (in questo caso il valore è cinque).

Infine, si dichiarano i parametri da utilizzare per i calcoli, cioè la matrice per la proiezione del mondo di vista, e la posizione della camera e delle luci all'interno dello spazio.

Nel fragment program le istruzioni iniziali sono simili a quelle del vertex program, tranne per l'ovvio che le operazioni sono sui pixel. Ciò che cambia sono i parametri considerati in questa fase.

Le variabili globali considerate sono le direzioni e le distanze delle luci reflesse all'interno dell'object space (lo spazio in relazione a un sistema ottico nel quale sono collocati gli oggetti che devono essere immaginati dal sistema), i parametri delle luci di tipo spotlight, il colore della luce ambientale, diffusiva e speculare interne alla scena.

Vengono dichiarati alla fine i valori della componente diffusiva e speculare della luce riflessa dal modello, cioè il colore appartenente dell'oggetto, e il valore dell'esponente per gli spike di luce. Questi tre numeri sono situati in questa parte del codice per permettere allo sviluppatore di poterli modificare facilmente in caso di modifiche, senza dover cercare all'interno del codice.

La parte del materiale è molto semplice visto che comprende in questo caso solamente i riferimenti al vertex program e il fragment program.

4.1.2 Documento hlsl

All'interno di questo documento si trovano le istruzioni utilizzate per ottenere il materiale voluto. Esse sono racchiuse all'interno delle due funzioni principali richiamate nel vertex e fragment program, più una sotto funzione per il calcolo della luce.

Si inizia istanziando due strutture dati, per definire un vertice in ingresso e uno in uscita. Nel primo caso vengono salvate la sua posizione e la normale associata, nell'altro si ha in aggiunta la direzione di vista e la direzione della luce.

La funzione che agisce sui vertici prende in ingresso un vertice e i parametri richiesti nel vertex program. Essa calcola i componenti della struttura dati per il vertice in uscita: la posizione è ottenuta dalla moltiplicazione tra la matrice di proiezioni del mondo di vista e la posizione del vertice in ingresso, mentre per la normale e la direzione di vista si normalizzano per la prima la normale del vertice d'ingresso e per la seconda la differenza tra la posizione della camera e la posizione del vertice. A questo punto viene calcolato l'ultimo valore, la direzione della luce, per il vertice in uscita considerando il limite di luci della scena. Si possono avere due casi in base al valore della coordinata omogenea della

posizione della luce. Nel momento in cui essa vale zero, si considera il solo vettore XYZ associato, altrimenti la differenza tra le coordinate della posizione della luce e la posizione del vertice di ingresso.

Per il calcolo dei punti interni ai triangoli della mesh, oltre ai parametri definiti all'interno nel fragment program, viene preso in ingresso il vertice ottenuto con i calcoli precedenti. Questa funzione esegue in maniera ciclica, pari al valore limite di luci, la normalizzazione di ogni direzione della luce e il calcolo dei valori delle componenti diffusiva e speculare grazie alla funzione di supporto. Infine, si restituisce il colore come la somma di questi due numeri e la componente ambientale.

La funzione di supporto prende in ingresso diversi parametri: la normale, la direzione di vista e la direzione della luce contenuta nel vertice di ingresso, le direzioni e le distanze sempre della luce ma all'interno dell'object space, i parametri delle luci di tipo spotlight, il valore di attenuazione della luce, l'esponente per il calcolo degli spike luminosi e le informazioni riguardanti il colore dichiarati nel fragment program.

In questo metodo viene applicato l'algoritmo di Blinn-Phong per il calcolo dell'intensità luminosa considerando i vari tipi di luci applicabili nella scena. Si ha un primo momento in cui vengono calcolati il prodotto scalare tra i vettori N e L, l'angolo tra N e il vettore medio H, il loro prodotto scalare e la luminosità presente nella scena (i simboli con cui sono rappresentati i vettori fanno riferimento alla figura 3.1.8 presentata nel capitolo precedente). Tutti questi valori verranno utilizzati nelle fasi successive per calcolare le componenti diffusive e speculari utilizzate per la somma finale.

A questo punto, in base ai parametri utilizzati per definire una luce di tipo spotlight, si possono ottenere due possibilità, quella in cui si ha questa tipologia di sorgente luminosa e quella che racchiude tutti gli altri casi possibili.

Quest'ultimo caso è il più semplice da calcolare, attraverso le due formule seguenti:

$$spec += pow(saturate(dotNH), specShine) * specColor * luminosity$$

$$diff += (saturate(dotNL)) * diffColor * luminosity$$

Come si può vedere viene utilizzata la funzione presente in HLSL *saturate* che permette di bloccare un valore specifico in un intervallo tra 0 e 1.

Nella circostanza in cui si considera una luce spotlight, bisogna prima di tutto calcolare, attraverso il prodotto scalare, il coseno tra la direzione frontale della luce (spotlight facing direction) e quella definita come pixelToLight fornita come parametro d'ingresso alla sotto funzione. Questo valore serve per distinguere due possibilità. Se esso è maggiore del parametro Y della luce, significa che il pixel è all'interno del cono generato dalla fonte luminosa e si può trovare la sua componente diffusiva con questa formula:

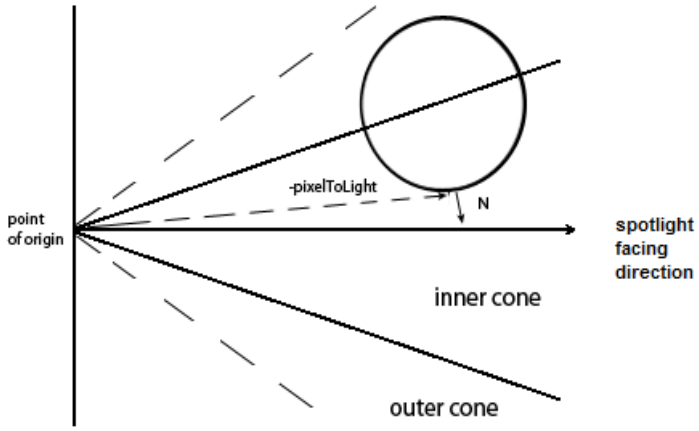

Figura 4.1.3. Questa immagine serve per dare chiarezza sui vettori considerati nel caso un cui si stia analizzando una luce di tipo spotlight.

$$diff \mathrel{+}= dotNL * diffColor * luminosity * (1 - (spotLightParams.x - dotPLd) / (spotLightParams.x - spotLightParams.y))$$

Se il valore è invece maggiore alla componente X della luce, di conseguenza il pixel è esterno al cono. In questo caso si utilizza invece l'istruzione seguente

$$diff \mathrel{+}= dotNL * diffColor * luminosity$$

Infine, si calcola la componente speculare nel caso in cui il prodotto scalare risulta maggiore di 0, grazie al codice

$$spec \mathrel{+}= pow(saturate(dotNH), specShine) * specColor * luminosity$$

4.2 Attivazione delle ombre

Per permettere ad OGRE di riuscire a calcolare le ombre presenti nella scena, si devono effettuare due passaggi fondamentali: uno nel quale viene dichiarata la tecnica che si vuole utilizzare e tutte le istruzioni per riuscire a migliorare l'effetto visivo; l'altro consiste nel creare una luce in scena grazie alla quale vengono generate le ombre. Infine, con l'istruzione

setCastShadows, si identificano quali oggetti producono buio e quali lo ricevono [23].

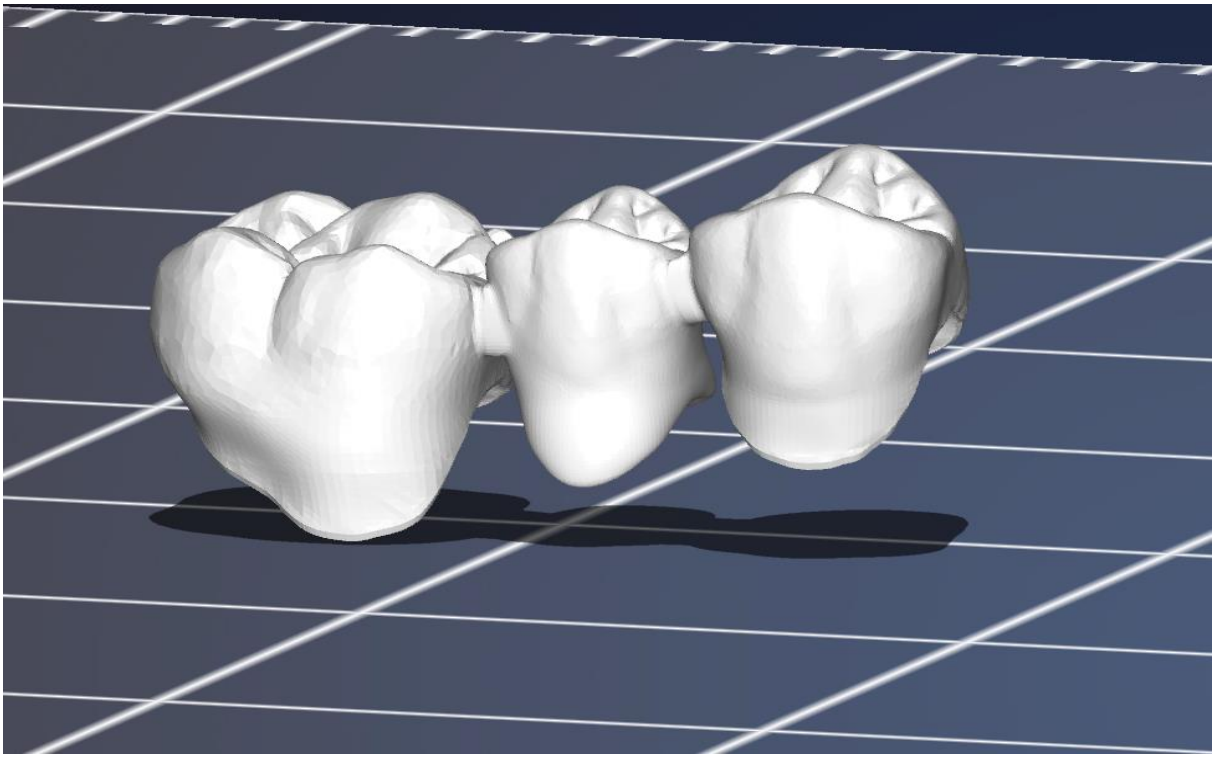

Figura 4.2.1. Questa immagine rappresenta lo stesso modello presente nelle figure 4.1.1 e 4.1.2 con l'aggiunta dell'ombra prodotta da esso su una piattaforma.

4.2.1 Dichiarazione della tecnica

In questo progetto si è utilizzata la tecnica basata sulla texture di tipo modulative. Questa scelta è stata fatta sulla base che Pyramis utilizza mesh con geometrie complesse e in questa maniera si riesce ad abbassare il costo computazionale insieme all'utilizzo del metodo Modulative Shadows. Per indicare ad OGRE di renderizzare le ombre si utilizza l'istruzione

setShadowTechnique(SHADOWTYPE_TEXTURE_MODULATIVE)

che imposta automaticamente gli oggetti come caster, cosa che viene evitata attraverso l'istruzione scritta nel paragrafo precedente.

A questo punto, grazie a specifici comandi, si settano le opzioni per migliorare la resa grafica. Quelli utilizzati sono state:

- *setShadowColour*, per rendere l'ombra più chiara;
- *setShadowTextureSize*, che permette di scegliere la grandezza della texture per evitare l'effetto aliasing sui bordi;
- *setShadowTextureCount*, grazie al quale si sceglie il numero massimo di shadow textures utilizzabili;
- *setShadowDirLightTextureOffset*, per influenzare la distanza della posizione della camera dal centro della texture shadow nel caso di utilizzo di una luce direzionale;
- *setShadowFarDistance*, con il quale indicare la distanza alla quale non vengono più generate le ombre.

L'ultimo passaggio consiste nello scegliere la shadow camera da utilizzare. Per avere il risultato migliore si è scelta la PSSM camera, con la quale si possono utilizzare più texture contemporaneamente.

4.2.2 Fonte luminosa

La sorgente luminosa scelta è di tipo directional. Essa è molto tenue considerando il fatto che serve solo per la generazione delle ombre visto che l'illuminazione dell'intera scena è svolta a parte.

Essa è posizionata in alto, perpendicolarmente alla piattaforma sulla quale vengono posizionati gli oggetti e rese visibili le zone oscure.

4.3 Limiti e tentativi

Grazie alle soluzioni adottate per il progetto, si sono riusciti dei buoni risultati eseguiti in tempi molto brevi e adattabili a tutti i possibili modelli utilizzabili. Allo stesso tempo, si hanno però alcuni difetti.

Per quanto riguarda lo shader addetto alla realizzazione del materiale attraverso il calcolo dell'intensità luminosa, si può notare dalle immagini che in alcune parti della mesh sono abbastanza visibili i triangoli che la compongono, limitando l'effetto di liscio della superficie. Si sono provati a realizzare altri shader per provare a migliorare questo difetto, ma alla fine grazie a quello adottato si ha il materiale migliore.

La tecnica utilizzata per la renderizzazione delle ombre produce ottimi risultati con ombre aventi bordi definiti. Il problema con essa è la mancanza della possibilità che le zone buie generate dagli oggetti siano visibili anche su altri modelli e non solo la piattaforma.

Per cercare di ovviare questo limite si è cercato di utilizzare il metodo delle Integreted-texture sfruttando la tecnologia del Shadow Mapping [24]. Essa consiste nel calcolare una mappa per le ombre attraverso uno specifico shader. Andando ad agire sul materiale dell'oggetto la si può integrare, ma durante il corso del progetto l'utilizzo di questa tecnica andava a creare

delle problematiche con la semplice generazione delle ombre, creando degli artefatti sui modelli; per questo motivo si è infine adoperato il metodo base per la visualizzazione delle zone buie.

È stato anche provato a dare un effetto di riflessione alla piattaforma. Per fare questo, si è lavorato su due possibilità: la creazione di una cubemap [25] (texture che rappresenta una versione specchiata del mondo come se fosse un cubo) grazie ad uno shader e l'utilizzo delle funzionalità integrate in OGRE.

Col primo metodo si è riusciti a creare la cubemap ed applicarla alla piattaforma, ma la riflessione che essa generava era sbagliata e non adattabile alla piattaforma.

Il secondo metodo invece richiede delle impostazioni particolari che senza una buona conoscenza del codice alla base del sistema Pyramis, non è possibile realizzare senza ottenere dei conflitti.

Capitolo 5

Conclusioni

Nel corso di questa tesi si è trattato lo studio di shader per la realizzazione di particolari effetti grafici all'interno di una scena digitale, in particolar modo illuminazione e ombreggiatura, utilizzabili dal sistema Pyramis durante la visualizzazione di modelli stampabili grazie a stampanti 3D. Per la realizzazione di questi programmi si è usufruito delle potenzialità del motore di rendering grafico OGRE 3D e il linguaggio HLSL per la loro scrittura. Applicando la teoria alla base dell'illuminazione, soprattutto il modello di luce speculare di Blinn-Phong, si è riuscito a creare un materiale per i modelli in grado di riflettere la luce adattandosi alla posizione della camera, con un buon risultato con basso costo computazionale considerando la complessità delle mesh utilizzate. Inoltre, con le potenzialità del motore grafico OGRE, si è riuscito a generare ombre da parte degli oggetti su una piattaforma, dando così un maggiore realismo alla scena. La possibilità di miglioramento è vasta, considerando lo sviluppo della computer grafica e della tecnologia delle schede grafiche in continua evoluzione. L'auspicio è di ottenere effetti più realistici non solo in campo videoludico o cinematografico, ma anche in quello industriale, così da poter fornire anteprime di prodotti non solo più belli visibilmente ma più precisi, in grado di fornire una versione con pochi errori e alta attendibilità.

Capitolo 6

Riferimenti

[1] GeeksforGeeks, "Computer graphics," [Online]. Available: https://www.geeksforgeeks.org/introduction-to-computer-graphics/. [Accessed Gennaio 2021].

[2] Britannica, «3D printing,» [Online]. Available: https://www.britannica.com/technology/3D-printing. [Consultato il giorno Gennaio 2021].

[3] A. J. v. d. Ploeg, "Interactive Ray Tracing," Maggio 2019. [Online]. Available: https://web.archive.org/web/20190520020303/http://www.few.vu.nl/~kielmann/theses/avdploeg.pdf. [Accessed Gennaio 2021].

[4] CIMsystem, «CIMsystem Dental,» [Online]. Available: https://www.cimsystem.com/dental/it/. [Consultato il giorno Gennaio 2021].

[5] CIMsystem, «Pyramis,» [Online]. Available: https://www.cimsystem.com/dental/it/products/pyramis-panoramica/. [Consultato il giorno Gennaio 2021].

[6] Springer, "Selective Laser Sintering," [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4615-1469-5_5. [Accessed Gennaio 2021].

[7] Wikipedia, «Stereolitografia,» [Online]. Available: https://it.wikipedia.org/wiki/Stereolitografia. [Consultato il giorno Gennaio 2021].

[8] Wikipedia, «Fusione laser selettiva dei metalli,» [Online]. Available: https://it.wikipedia.org/wiki/Fusione_laser_selettiva_di_metalli. [Consultato il giorno Gennaio 2021].

[9] Wikipedia, «Digital Light Processing,» [Online]. Available: https://it.wikipedia.org/wiki/Digital_Light_Processing. [Consultato il giorno Gennaio 2021].

[10] Microsoft, «Visual Studio,» [Online]. Available: https://visualstudio.microsoft.com/it/. [Consultato il giorno Gennaio 2021].

[11] Qt, "Qt," [Online]. Available: https://www.qt.io/. [Accessed Gennaio 2021].

[12] OGRE, "OGRE," [Online]. Available: https://www.ogre3d.org/. [Accessed Gennaio 2021].

[13] OGREWiki, "OGREWiki," [Online]. Available: http://wiki.ogre3d.org/Home. [Accessed Gennaio 2021].

[14] OGRE, "Story of OGRE," [Online]. Available: http://wiki.ogre3d.org/Brief+history+of+OGRE. [Accessed Gennaio 2021].

[15] OGRE, "High-level Programs," [Online]. Available: https://ogrecave.github.io/ogre/api/1.11/_high-level-_programs.html. [Accessed Gennaio 2021].

[16] Microsoft, "High-level shader language (HLSL)," [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl. [Accessed Gennaio 2021].

[17] OGREWiki, "Using HLSL in OGRE," [Online]. Available: http://wiki.ogre3d.org/HLSL. [Accessed Gennaio 2021].

[18] OGRE, "Lights, Camera and Shadows," [Online]. Available: https://ogrecave.github.io/ogre/api/1.10/tut__lights_cameras_shadows.html#:~:text=Ogre%20provides%20three%20types%20of,Light%20works%20like%20a%20flashlight.. [Accessed Gennaio 2021].

[19] GeeksforGeeks, «The RGB color model,» [Online]. Available: https://www.geeksforgeeks.org/computer-graphics-the-rgb-color-model/. [Consultato il giorno Gennaio 2021].

[20] Wikipedia, "Blinn-Phong reflection model," [Online]. Available: https://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_reflection_model. [Accessed Gennaio 2021].

[21] OGRE, "Shadows," [Online]. Available: https://ogrecave.github.io/ogre/api/1.12/_shadows.html. [Accessed Gennaio 2021].

[22] JaJDoo, "Shader guide for per-pixel Blinn-Phong," [Online]. Available: http://wiki.ogre3d.org/White+Phong+part+3+-+JaJDoo+Shader+Guide+-+Basics. [Accessed Gennaio 2021].

[23] OGREWiki, "How to start with shadows," [Online]. Available: http://wiki.ogre3d.org/Shadows. [Accessed Gennaio 2021].

[24] OGRECave, "Shadow Mapping," [Online]. Available: https://ogrecave.github.io/ogre/api/1.11/_shadow_mapping_ogre.html. [Accessed Gennaio 2021].

[25] Microsoft, "Cube Mapping," [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/direct3d9/cubic-environment-mapping. [Accessed Gennaio 2021].