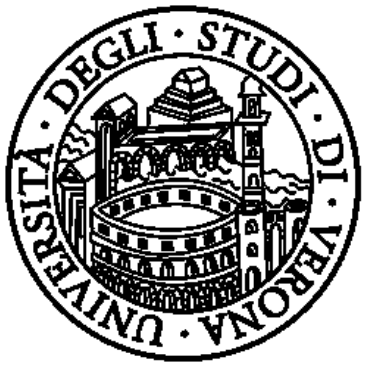

UNIVERSITÀ di VERONA

Università degli studi di Verona – Dipartimento di Ingegneria e scienze informatiche

ESERCITAZIONE AGGIUNTIVA 1

- Programmazione Grafica -

Prof. Referente
Umberto Castellani

A cura di
Davide Zanellato
13/09/24

INDICE

1. Modello

Tramite il sito mixamo.com ho scaricato un modello statico con le relative texture, diffusa, speculare e normale. Il modello e tutte le sue caratteristiche, come vertici, normali e coordinate texture, vengono caricate nell'applicazione tramite Assimp.
I modelli vengono creati dagli 3D Artists utilizzando appositi strumenti di modellazione 3D, come blender, che consentono di creare forme complesse e di applicare loro delle texture tramite un processo chiamato UV-mapping. Gli strumenti generano quindi automaticamente tutte le coordinate dei vertici, le normali e le coordinate delle texture, esportandole in un formato di file che è possibile utilizzare all'interno di una applicazione. Per analizzare questi file ed estrarre tutte le informazioni rilevanti in modo da poterle archiviare in un formato che OpenGL possa comprendere, viene utilizzata la libreria Assimp che è in grado di importare decine di formati di file. Quando si importa un modello tramite Assimp, viene caricato l'intero modello in un oggetto scena che contiene tutti i suoi dati. Assimp ha quindi una raccolta di nodi e ognuno di essi contiene degli indici che fanno riferimento ai dati memorizzati nell'oggetto scena e inoltre ogni nodo può avere un numero qualsiasi di figli.

Il processo di caricamento di un modello prevede diversi passaggi:

- caricare tutti i dati in un oggetto Scene;
- recuperare ricorsivamente gli oggetti Mesh corrispondenti a ciascuno dei nodi;
- elaborare ogni oggetto Mesh per recuperare i dati dei vertici, gli indici e le sue proprietà dei materiali.

Ogni modello di solito è composto da diversi sottomodelli chiamati Mesh e tramite la loro combinazione è possibile ottenere l'oggetto finale. Una singola Mesh è la rappresentazione minima di ciò che bisogna possedere per disegnare un oggetto in OpenGL, dati dei vertici, indici e proprietà dei materiali.

Nell'applicazione si utilizza la classe "Mesh.h" che definisce due strutture: "Vertex" rappresenta un singolo vertice di una mesh e contiene gli attributi: posizione del vertice, il vettore normale, le coordinate texture, il vettore tangente e il vettore bitangente; la struttura "Texture" rappresenta una texture utilizzata dalla mesh e contiene l'ID della texture, il tipo e il percorso del file. La classe "Mesh" rappresenta una mesh completa e contiene un vettore di vertici che la compongono, un vettore di indici per il suo rendering e un vettore di texture applicate ad essa. Il costruttore invece inizializza i dati della mesh e chiama il metodo "setupMesh()" per configurare i buffer. In questo metodo vengono quindi inizializzati i buffer VAO, VBO, EBO e configurati i puntatori degli attributi del vertice: posizione, normali, coordinate texture, tangente, bitangente, bone ID e Weights.

La funzione “Draw()” attiva e associa le texture corrette e disegna la mesh utilizzando i dati nei buffer. Per selezionare le texture viene utilizzata una convenzione, ogni texture diffusa è denominata “texture_diffuseN”, ogni texture speculare è denominata “texture_specularN” e ogni texture normale è denominata “texture_normalN”.

Per importare il modello nell’applicazione si utilizza la classe “Model.h” che definisce un vettore “textures_loaded” per memorizzare tutte le texture caricate finora ed evitare di caricarle più di una volta e un vettore “meshes” che contiene tutte le mesh che compongono il modello. Successivamente il costruttore inizializza il modello caricando il file specificato. La funzione “loadModel()” carica un modello dal file specificato utilizzando l’importer di Assimp e memorizza le mesh risultanti nel vettore meshes. All’importer possono essere specificate diverse opzioni che impongono ad Assimp di eseguire dei calcoli su dati caricati: “aiProcess_Triangulate” specifica che se il modello non è interamente costituito da triangoli, Assimp deve trasformare tutte le forme primitive in triangoli; “aiProcess_FlipUVs” capovolge le coordinate della texture sull'asse y durante l'elaborazione; “aiProcess_CalcTangentSpace” calcola automaticamente tangente e bitangente. La funzione “processNode()” elabora un nodo della scena ricorsivamente, processando ciascuna mesh situata nel nodo e ripetendo il processo per i nodi figli. La funzione “processMesh()” elabora una singola mesh estraendo i vertici, le normali e le coordinate texture; estrae gli indici di ciascuna faccia della mesh; elabora i materiali della mesh per caricare le texture diffuse, speculari, normali. La funzione “loadMaterialTextures()” controlla tutte le texture di un determinato tipo, se la texture è già stata caricata, viene saltata, se invece la texture non è ancora stata caricata, viene elaborata dal file e memorizzata. La funzione “TextureFromFile()” carica un'immagine da file utilizzando “stb_image”, genera una texture OpenGL, la configura e restituisce l'ID della texture.

2. Lighting

Per implementare il lighting nell’applicazione ho utilizzato il modello di illuminazione di Phong che è costituito da tre componenti: illuminazione ambientale, diffusa e speculare. L’illuminazione ambientale è la costante di illuminazione presente anche quando è buio, infatti in questa circostanza è comunque presente una qualche forma di luce e quindi gli oggetti non sono quasi mai completamente al buio. L'illuminazione diffusa simula l'impatto direzionale che un oggetto luminoso ha su un altro oggetto ed è il componente visivamente più significativo del modello di illuminazione. L’illuminazione speculare simula il punto luminoso di una luce che appare sugli oggetti lucidi.

L'illuminazione diffusa conferisce all'oggetto maggiore luminosità quanto più i suoi fragments sono allineati ai raggi di luce di una sorgente luminosa. L'obiettivo è quello di misurare l'angolo con cui il raggio di luce tocca il fragment e se il raggio di luce è perpendicolare alla superficie dell'oggetto, la luce ha il maggiore impatto. Per misurare l'angolo tra il raggio di luce e il fragment utilizziamo il vettore normale, ovvero un vettore perpendicolare alla superficie del fragment. Il raggio di luce rappresenta il vettore di direzione della luce che si ottiene dalla differenza tra la posizione della luce e la posizione del fragment. L'angolo tra il raggio di luce e il vettore normale del fragment può quindi essere calcolato con il prodotto scalare. Minore è l'angolo tra due vettori unitari, più il valore del prodotto scalare è vicino all'1, quando l'angolo tra entrambi i vettori è di 90 gradi, il prodotto scalare diventa 0, mentre se l'angolo tra entrambi i vettori è maggiore di 90 gradi, il risultato del prodotto scalare diventerà in negativo.

L'illuminazione speculare si basa sul vettore di direzione della luce, sui vettori normali dell'oggetto, sul vettore di direzione della vista e sulle proprietà riflettenti delle superfici. Bisogna calcolare un vettore di riflessione riflettendo la direzione della luce attorno al vettore normale. Successivamente si calcola il vettore di direzione della vista con la sottrazione tra la posizione dell'osservatore nello spazio mondo e la posizione del fragment. Infine l'angolo tra il vettore di riflessione e il vettore di direzione della vista si ottiene con il prodotto scalare, minore è l'angolo tra loro, maggiore è l'impatto della luce speculare.

3. Mappe diffuse e speculari

Le mappe diffuse rappresentano il modo in cui si fa riferimento alle texture nelle scene illuminate e servono per impostare i colori diffusi di un oggetto per ogni singolo fragment. Le mappe speculari vengono utilizzate per controllare quali parti dell'oggetto dovrebbero mostrare un riflesso speculare con intensità variabile, si tratta di texture tipicamente in bianco e nero che definiscono le intensità speculari di ogni parte dell'oggetto.

4. Luce direzionale, puntiforme e spotlight

Nell'applicazione ho implementato tre diversi tipi di luce: direzionale, puntiforme e spotlight.

La luce direzionale rappresenta una sorgente luminosa lontana i cui raggi luminosi sono paralleli tra loro e provengono dalla stessa direzione, indipendentemente da dove si trova l'oggetto e/o l'osservatore.

Per modellare una luce direzionale, nell'applicazione ho definito un vettore di direzione della luce e i tre vettori che rappresentano le componenti ambientale, diffusa e speculare.

Una luce puntiforme rappresenta una sorgente luminosa con una determinata nella scena, che illumina in tutte le direzioni e i cui raggi di luce svaniscono in lontananza. Il valore di attenuazione è calcolato tramite una formula basata sulla distanza di un fragment dalla sorgente luminosa, che poi moltiplichiamo con il vettore di intensità della luce.

$$F_{att} = \frac{1.0}{K_c + K_l * d + K_q * d^2}$$

Per calcolare il valore di attenuazione definiamo tre termini configurabili: una costante "Kc", un lineare "Kl" e un quadratico "Kq", mentre d rappresenta la distanza dal fragment alla sorgente luminosa. Infine questo valore di attenuazione calcolato viene moltiplicato con il vettore di intensità della luce. Per modellare una luce puntiforme, nell'applicazione ho definito un vettore di posizione della luce, i tre vettori che rappresentano le componenti ambientale, diffusa e speculare e i tre valori costante, lineare e quadratico per il calcolo dell'attenuazione.

Una spotlight rappresenta una sorgente luminosa che si trova da qualche parte nell'ambiente e che, invece di diffondere raggi di luce in tutte le direzioni, li diffonde solo in una direzione specifica. Il risultato è che solo gli oggetti entro un certo raggio dalla direzione della spotlight sono illuminati, mentre tutto il resto non lo è. Una spotlight è definita da una posizione nello spazio mondo, una direzione e un angolo cutoff che specifica il raggio della torcia. Bisogna quindi calcolare, per ogni fragment, se si trova tra le direzioni del cutoff della torcia e, in tal caso, illuminarlo. Per modellare una spotlight sotto forma di torcia, nell'applicazione ho definito un vettore di posizione e direzione della luce, tre vettori che rappresentano le componenti ambientale, diffusa e speculare che cambiano in base al fatto che la torcia si accesa o spenta, i tre valori costante, lineare e quadratico per il calcolo dell'attenuazione e un valore per l'angolo e "cutoff" e "outerCutOff".

Una torcia è una spotlight presente nella posizione dell'osservatore e solitamente puntata dritta davanti alla sua prospettiva e di conseguenza la posizione e direzione della luce vengono continuamente aggiornate in base alla posizione e all'orientamento dell'osservatore. Il valore di "outerCutOff" viene utilizzato per creare dei bordi lisci e morbidi, possiamo immaginare il valore di cutoff che specifica il raggio del cono interno e il valore di "outerCutOff" che specifica il raggio del cono esterno.

Se un fragment è tra il cono interno e quello esterno, viene calcolato un valore di intensità tra 0.0 e 1.0, se il fragment è all'interno del cono interno, la sua intensità è uguale a 1.0, mentre se il fragment è all'esterno del cono esterno la sua intensità è 0.0.

$$I = \frac{\theta - \gamma}{\epsilon}$$

Per calcolare il valore di intensità, è necessario ottenere “theta” come prodotto scalare tra il vettore che punta dal fragment alla sorgente luminosa e il vettore che rappresenta la direzione verso cui punta la torcia. Si calcola successivamente “epsilon” eseguendo la sottrazione tra “cutoff” e “outerCutOff” e infine si determina l’intensità “I” con la sottrazione tra “theta” e “outerCutOff” e dividendo il tutto per epsilon. Per fare questo si utilizza la funzione clamp i cui argomenti sono impostati a 0.0 e 1.0 per assicurare che i valori di intensità non finiscano al di fuori dell'intervallo [0, 1]. Quando vengono impostati i valori angolari di “cutoff” e “outerCutOff”, in realtà si calcola il valore del coseno dell’angolo specificato, in quanto, nel fragment shader, il prodotto scalare tra il vettore che punta dal fragment alla sorgente luminosa e il vettore che rappresenta la posizione verso cui punta la torcia, restituisce un valore del coseno e questo dato sarà successivamente utilizzato nella sottrazione, che non può essere eseguita in presenza di un angolo e di un valore del coseno.

5. Normal mapping e Tangent space

Nell’applicazione ho utilizzato il normal mapping e implementato il tangent space.

Il normal mapping viene utilizzato quando una superficie non è perfettamente piatta, ad esempio se presenta crepe o buchi, e consiste nell’utilizzare un vettore normale diverso per ogni fragment invece di un solo vettore normale per l’intera superficie. Utilizzando le normali per fragment possiamo ingannare l'illuminazione facendole credere che una superficie sia composta da tanti piccoli piani rendendola superficie molto più dettagliata. Per implementare il normal mapping ho utilizzato una normal map, ovvero una texture che memorizza i dati delle normali per ogni fragment. Le componenti x, y e z di un vettore normale vengono memorizzate nei rispettivi componenti colore RGB, i vettori normali variano tra -1 e 1 e vengono quindi mappati tra [0, 1], ovvero l’intervallo di valori che possono essere assunti dai componenti RGB. Le normal map hanno un colore tendente al blu in quanto tutte le normali sono strettamente puntate verso l'esterno, verso l'asse z positivo (0, 0, 1).

Nel fragment shader prima di tutto bisogna recuperare le normali per ogni fragment dalla normal map e invertire il processo di mappatura delle normali sui colori RGB rimappando il colore normale campionato da [0, 1] a [-1, 1]. Successivamente vengono utilizzati i vettori normali campionati per effettuare i calcoli necessari ad implementare il modello di illuminazione di Phong.

In una normal map tutti i vettori normali puntano sempre verso l'asse z positivo, quindi se si modificasse la direzione del piano, che inizialmente puntava anch'esso verso l'asse z positivo, ad esempio puntando verso l'asse y positivo, l'illuminazione risulterebbe scorretta.

Per risolvere questo problema viene utilizzato il tangent space, uno spazio di coordinate in cui i vettori della normal map puntano sempre verso la direzione z positiva e tutti gli altri vettori di illuminazione vengono trasformati rispetto a questa direzione z positiva.

Per prima cosa calcolo la matrice TBN, formata da tre vettori perpendicolari allineati lungo la superficie della normal map: tangente, bitagente e normale. Essa permette di trasformare i vettori normali dallo spazio tangente locale in coordinate mondiali o di vista. I vettori tangente e bitangente vengono calcolati automaticamente specificando l'apposita opzione per l'importer di Assimp e, insieme al vettore normale, vengono passati al vertex shader. Per la creazione della matrice è necessario trasformare tutti i vettori TBN nel sistema di coordinate spazio mondo moltiplicandoli con la matrice model.

Prendiamo quindi l'inverso della matrice TBN che trasforma qualsiasi vettore dallo spazio mondo allo spazio tangente e utilizziamo questa matrice per trasformare le variabili di illuminazione rilevanti nello spazio tangente, esclusa la normale che vi si trova già. Questa operazione viene eseguita nel vertex shader invece che nel fragment shader in quanto la posizione della luce e la posizione dell'osservatore non vengono aggiornate ad ogni esecuzione del fragment shader e inoltre è possibile calcolare i vettori di illuminazione nei vertici e lasciare che l'hardware grafico esegua l'interpolazione. Quindi tutte le variabili di illuminazione rilevanti trasformate nello spazio tangente e successivamente inviate al fragment shader dove vengono utilizzate per eseguire i calcoli necessari ad implementare il modello di illuminazione di Phong con tre diversi tipi di luci: direzionale, puntiforme e spotlight.

6. Trasformazioni e rendering

Successivamente vengono inizializzate le matrici “projection”, “view” e “model”.

La matrice “projection” si definisce utilizzando la funzione glm::perspective che permette di creare un frustum che definisce lo spazio visibile, tutto ciò che si trova al di fuori del frustum non finirà nel volume dello spazio clip e quindi verrà ritagliato. Il primo parametro definisce il valore del campo visivo e imposta la grandezza dello spazio di visualizzazione. Il secondo parametro imposta il rapporto di aspetto che viene calcolato dividendo la larghezza della finestra per la sua altezza. Il terzo e il quarto parametro impostano il piano vicino e lontano del frustum.
La matrice “view” trasforma le coordinate mondiali nel view space e permette di modificare la posizione della telecamera nella nostra scena.

La matrice “model” trasforma le coordinate dell’oggetto dallo spazio locale allo spazio mondo e permette di eseguire trasformazioni per traslarlo, ridimensionarlo e/o ruotarlo con l’obiettivo di collocarlo nella scena in una determinata posizione e con un determinato orientamento. Poiché la moltiplicazione tra matrici non è commutativa è importare eseguire le trasformazioni nell’ordine corretto: prima la scala, poi la rotazione e infine la traslazione. Inoltre la matrice “model” deve essere inizializzata esplicitamente alla matrice identità impostando il valore della sua diagonale ad 1.0, se questa operazione non viene eseguita, il risultato sarebbe una matrice nulla con tutti gli elementi a 0 e tutte le successive operazioni finirebbero anch'esse per produrre una matrice nulla.

Per far ruotare la luce e telecamera attorno alla scena si utilizza il seno per creare la coordinata x e il coseno per creare la coordinata z, le quali rappresentano un punto in un cerchio. Ricalcolando le coordinate nel tempo è possibile attraversare tutti i punti nel cerchio e l’effetto risultante consiste nel far ruotare la telecamera e la luce attorno alla scena.

Infine, dopo aver posizionato gli oggetti nello spazio mondo, viene eseguito il rendering del modello e del cubo di luce.