ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Corso di Laurea Magistrale in
Ingegneria Informatica

Navigazione interattiva in scena con modelli mesh 3D
LAB 3

Fondamenti di Computer Graphics M

Cristian Davide Conte 0001034932

Anno Accademico: 2022-2023

Contents

Chapter 1

Calcolo e memorizzazione delle normali ai vertici per i modelli mesh poligonali

Per quel che riguarda il calcolo delle normali, la funzione "loadObjFile" tenta prima di leggerle dal file obj (mesh) e qualora non fossero già presenti (nessun header "vn"), le calcola dinamicamente. L'utente può scegliere se calcolare le normali alle facce (1 per triangolo/faccia), o quelle ai vertici (3 per triangolo/faccia) tramite menù contestuale (tasto destro).
A seconda dell'opzione selezionata otterremo risultati differenti.

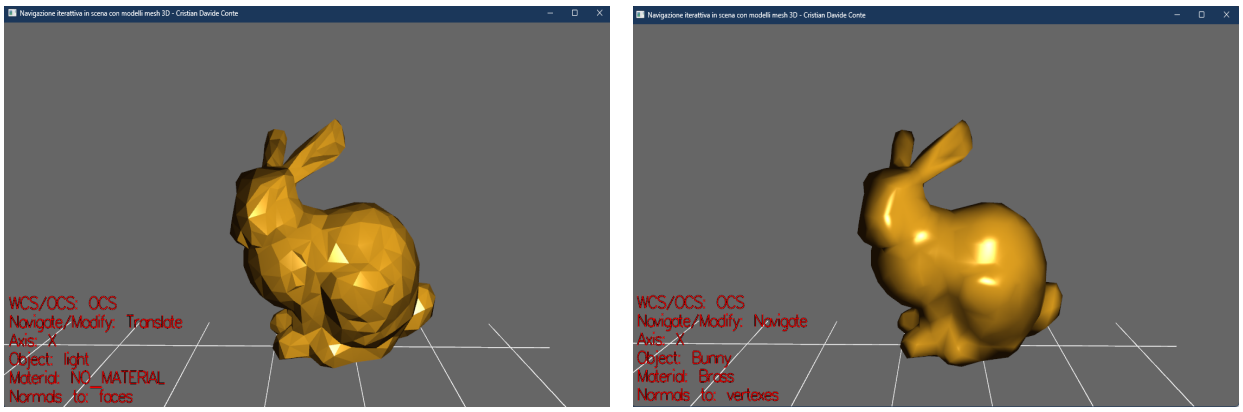

Figure 1.1: Bunny - Normali alle facce (sx) vs Normali ai vertici (dx)

Chapter 2

Creazione di un nuovo materiale

Il nuovo materiale che si è scelto di creare cerca di essere il più vicino possibile a quello proposto nelle imagini del Bunny Toon-Shading del pdf della consegna (la tonalità più chiara, azzurina).

Di seguito le sue caratteristiche in termini di luce ambient/diffusa/riflessa:

```
glm::vec3 custom_material_ambient = { 0.1, 0.3, 0.5 },
          custom_material_diffuse = { 0.3, 0.3, 1.0 },
          custom_material_specular = { 0.1, 0.3, 0.5 };
GLfloat custom_material_shininess = 50.0;
```

Queste proprietà sono poi state assegnate ad un nuovo materiale, tramite la funzione "init", in modo tale da poter essere utilizzato dagli oggetti in scena.

```
materials[
    MaterialType::CUSTOM_MATERIAL
].name = "Custom material";

materials[
    MaterialType::CUSTOM_MATERIAL
].ambient = custom_material_ambient;

materials[
    MaterialType::CUSTOM_MATERIAL
].diffuse = custom_material_diffuse;

materials[
    MaterialType::CUSTOM_MATERIAL
].specular = custom_material_specular;
materials[
    MaterialType::CUSTOM_MATERIAL
].shininess = custom_material_shininess;
```

Il materiale può a questo punto essere assegnato a qualsiasi oggetto in scena durante la sua fase di init.

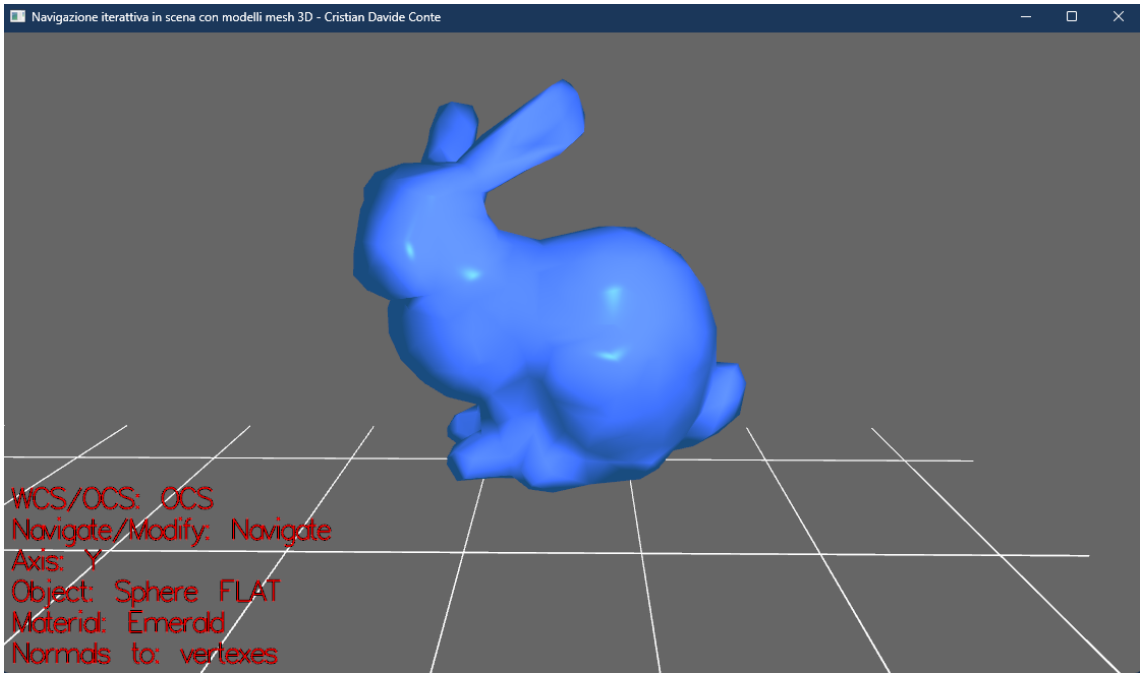

Figure 2.1: Bunny - Custom Material

Chapter 3

Wave Motion

Al fine di creare l'effetto "wave motion", all'interno del vertex shader "v_wave.glsl" richiediamo in ingresso la posizione del vertice ("aPos" copiato dentro "v") e andiamo a modificare la sua coordinata y seguendo la seguente legge del moto ondoso:

$$v_y = a * sin(wt + 10v_x) * sin(wt + 10v_z)$$

Il tempo "t" viene richiesto in ingresso al vertex shader tramite la variabile "current_time" ed è calcolato dalla funzione "get_current_time". Il binding della variabile glsl "current_time" con il codice OpenGL viene fatto all'interno della funzione "initShader". Il valore restituito da "get_current_time" viene passato allo shader tramite la "drawScene".

A questo link è possibile scaricare una breve demo del wave motion:
`https://github.com/CristianDavideConte/Fondamendi-di-Computer-Graphics/blob/main/LAB_03/Waves%20demo.mp4`

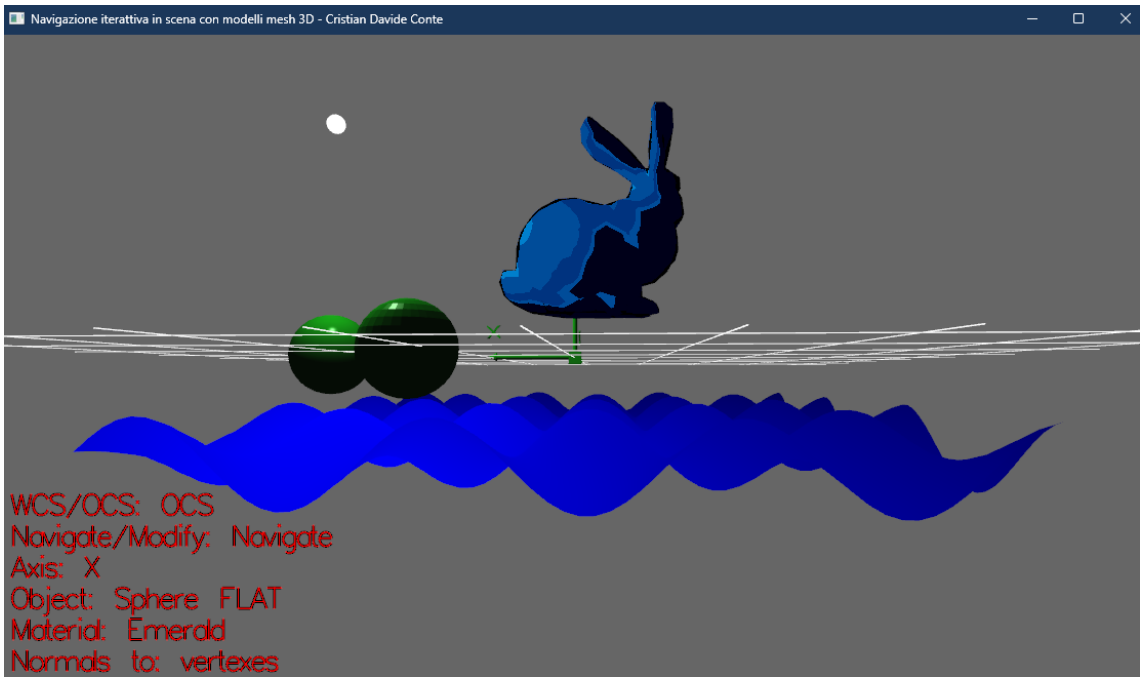

Figure 3.1: Wave motion

Chapter 4

Toon Shading

Il Toon Shading è una tecnica che permette di colorare le diverse parti (fragment) di un oggetto in scena, con sfumature dello stesso colore (blu nel nostro caso), a seconda dall'intensità della luce che le colpisce. Il vertex shader "v_toon.glsl" è implementato esattamente come "v_blinn.glsl" (segue il modello di riflessione di blinn-phong), mentre il calcolo dell'intensità luminosa e del conseguente colore dei fragment avviene all'interno di "f_toon.glsl". In pratica quello che viene fatto all'interno del fragment shader non è altro che la selezione del colore del fragment tramite la variabile "intensity" che rappresenta (il coseno del) l'angolo d'incidenza della luce sul fragment (prodotto scalare tra raggio luminoso e normale del fragment): maggiore è l'angolo di incidenza più scura sarà la sfumatura di blu applicata.

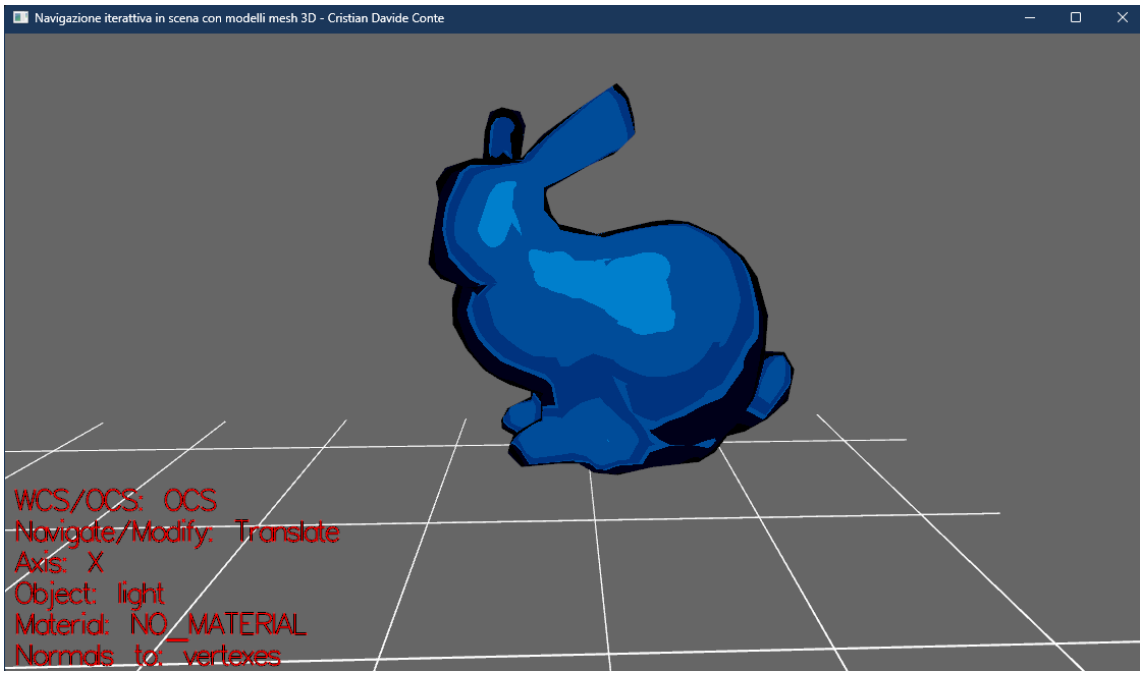

Figure 4.1: Bunny - Toon Shading

Chapter 5

Camera Motion

La posizione della camera in scena è definita dai vettori:

- VUP, il view up vector
- w, la view direction
- u, il vettore di spostamento orizzontale della camera

Al fine di permettere un movimento orizzontale della camera, è necessario:

- ottenere VUP
- ottenere w (posizione camera - posizione target)
- ottenere il vettore spostamento (cross product tra VUP e w, stessa direzione di u)
- sommare il vettore spostamento alla posizione attuale della camera (C)
- sommare il vettore spostamento alla posizione attuale del target (A)

Queste operazioni sono implementate tramite le funzioni "moveCameraLeft" e "moveCameraRight".

Chapter 6

Trasformazione degli oggetti in scena

Per l'ultimo punto si richiede di gestire le transformazioni degli oggetti in scena tramite la funzione "modifyModelMatrix" che va implementata. Prima di applicare le transformazioni, va notato che alla funzione "modifyModelMatrix" vengono passati i seguenti input:

- translation_vector
- rotation_vector
- angle
- scale_factor

Lo "scale_factor" deve essere applicato solo alla "WorkingAxis" selezionata, per cui prima viene creato un vettore "scaling_vector" unitario (vec3) che moltiplicherà la componente relativa all'asse selezionato per lo "scale_factor".
A questo punto sono solo due sono i possibili sistemi di riferimento che l'utente può selezionare per le trasformazioni:

- OCS
- WCS

Nel caso OCS basterà applicare (moltiplicare) direttamente le transformazioni alla matrice "M" dell'oggetto.
Nel caso WCS invece è necessario prima invertire la matrice "M" di trasformazione dell'oggetto, applicare le trasformazioni ed invertire nuovamente "M".