LAB 03

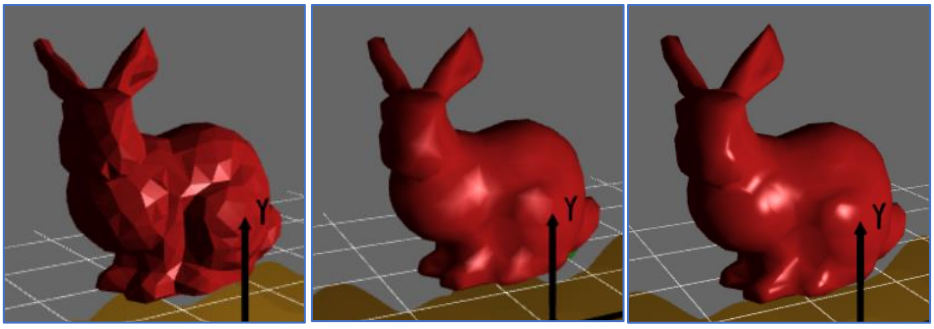
Il laboratorio 3 si è concentrato sull'utilizzo alcune mesh poligonali di oggetti tridimensionali già fornite e la creazione/modifica di fragment e vertex shaders per permettere lighting e shading della scena e aggiungere anche il movimento. Nella prima parte si è presa visione delle funzionalità già esistenti, con alcuni test pratici per prendere familiarità con l'ambiente. Nelle parti successive, invece, le richieste di modifica/implementazione sono state:

- Parte 1:
  - Calcolo delle normali ai vertici per il Phong e il Gouraud Shading (a differenza di quello Flat già implementato);
  - Creazione di un materiale diverso da quelli forniti;
  - Aggiungere un'animazione di tipo "onda" all'height field mesh GridPlane, inizialmente piatto;
  - Implementare lo shading aggiuntivo Toon Shading per la resa non fotorealistica;
- Parte 2: aggiunta dello spostamento orizzontale (camera e punto di riferimento) alle funzionalità della trackball;
- Parte 3: aggiunta delle trasformazioni di traslazione, rotazione e scala per ogni oggetto (relativamente sia al WCS sia all'OCS).

1. Shading

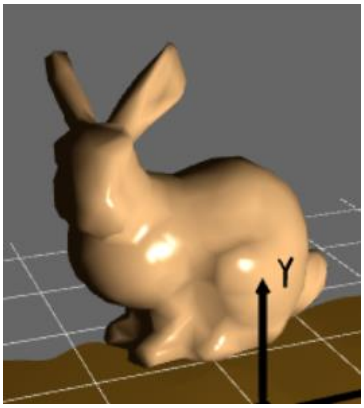
1.1. Calcolo delle normali per il Phong Shading

In generale, le normali ai vertici o alle facce vengono calcolate, se non presenti, al caricamento delle informazioni dal file .obj (`loadObjFile`). Nel caso di normali alle facce, quindi nel caso di Flat Shading, ci si limita a calcolare il valore di una normale (di solito ottenuta dai tre vertici del triangolo) e ad utilizzarla per tutti i pixel della faccia stessa. Il risultato sarà quello di avere la stessa luce (e quindi lo stesso colore per tutti i pixel) su tutta la singola faccia, ottenendo un effetto "faccettato" anche un oggetto in realtà liscio. Di conseguenza, è stato modificato l'algoritmo e sono state calcolate le normali di ogni vertice di ogni faccia. In particolare, per ogni vertice si effettua la somma delle normali di tutti i triangoli che incidono su quel vertice, per poi normalizzare i valori.



(Nell’immagine sopra, in ordine: Flat Shading, Gouraud e Phong Shading)

1.2. Creazione di un materiale diverso



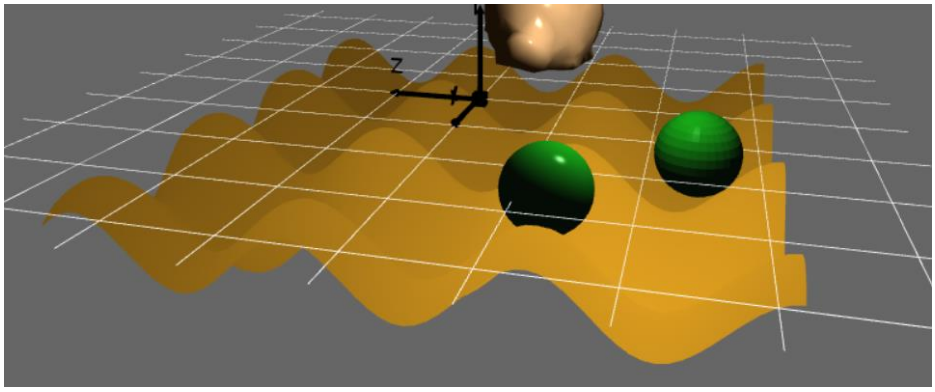
In questo caso, è stato necessario creare un materiale diverso da quelli forniti, specificando la componente ambientale (rose_gold_ambient), ovvero la componente che simula la luce ambiente indiretta, la componente diffusiva (rose_gold_diffuse), che fornisce la frazione di luce che viene riflessa in tutte le direzioni e quindi il colore dell’oggetto, la componente speculare (rose_gold_specular), che indica quando il materiale sia “lucido” e, infine, il l’esponente di riflessione speculare (rose_gold_shiness - visivamente, modifica l’ampiezza dello spot di riflessione della luce).

1.3. Animazione wave

Per introdurre un’animazione ad “onda” al piano presente nella scena, bisogna agire, come suggerito dalla consegna, modificando la posizione dei punti del piano con una funzione che dipende da un tempo *t* passata da applicazione a shader, in particolare al vertex shader, che si occupa di processare i punti e non i fragment. Per ottenere l’effetto voluto, la funzione è una sinusoidale del tipo:

$$y_t = y_{t-1} + a\ (\ \sin(wt + 10x_{t-1}) + \sin(wt + 10z_{t-1})\ )$$

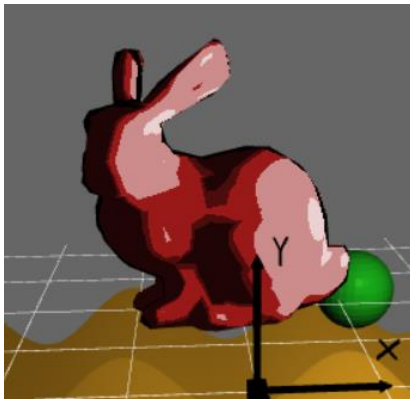
La nuova posizione è calcolata solo per la coordinata y, poiché il movimento ondulatorio è voluto in quella direzione. Inoltre, nello stesso shader, sono calcolate anche le trasformazioni sull’oggetto (con le matrici di proiezione, vista e modello) e il colore punto.



1.4. Toon Shading

L’ultima parte riguardante il lighting è stata l’aggiunta del toon shader (sia vertex sia fragment) per una resa dell’illuminazione di tipo cartoon (vedi immagine). Per riuscire ad ottenere tale resa:

- Nel vertex shader sono eseguite solo le trasformazioni (position, scaling, rotation) tramite le matrici P, V ed M. L'output del fragment shader sono i vettori N, L ed E e saranno l'input per il fragment shader;
- Nel fragment shader è invece gestita la parte vera e propria di shading: vengono calcolati cinque colori della sfumatura a partire dalla componente diffusiva del colore scelto in applicativo e, in base al valore dell'intensità calcolata tramite N, L ed E, al fragment viene assegnato uno dei cinque colori.



2. Spostamento di camera e punto di riferimento in scena

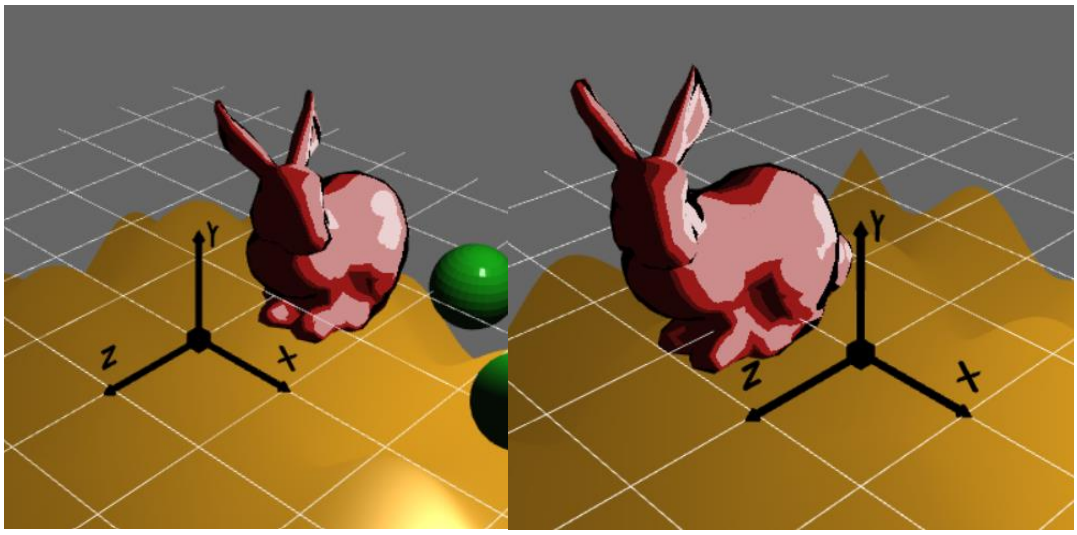
Lo spostamento voluto si ottiene con una combinazione di tasti (CTRL+ scroll wheel up/down), per cui per prima cosa bisogna andare a definire le funzioni di callback chiamate per tali interazioni (tramite `glutMouseFunc`, callback: `moveCameraRight` e `moveCameraLeft`).

Dopodiché, dato che se la camera si muove a destra, gli oggetti si sposteranno a sinistra e viceversa, sarà sufficiente definire un vettore direzione e calcolare l'incremento laterale di posizione e punto di riferimento (la direzione di spostamento è ortogonale all'upVector).

3. Aggiunta delle trasformazioni rispetto a WCS e OCS

L'aggiunta delle trasformazioni rispetto ai due diversi sistemi di riferimento dipende dai modificatori scelti in applicativo, quindi al tipo di trasformazione (Translate = 'g', Rotate='r' o Scale='s') e dal sistema (WCS/OCS, selezionabile da menù). Quando viene utilizzata la rotella del mouse, di verificano tali valori, si calcolano le matrici di traslazione, scala e rotazione, e

- Se è selezionato WCS, l'ordine per la modifica è *matrice_di_trasformazione*matrice_di_modello*(dell'oggetto) ;
- Se è selezionato OCS, l'ordine è *matrice_di_modello*matrice_di_trasformazione* .



Nella prima immagine, il coniglio è stato ruotato attorno all'asse y rispetto al suo OCS, mentre nella seconda immagine rispetto al WCS (di cui gli assi sono rappresentati come oggetto in scena).