Grafica

Anno Accademico 2018-2019

Esercitazione 6

Matteo Berti, Matricola 889889

27 giugno 2019

Comandi da tastiera

Tasto	Azione
Esc	Terminazione dell'esecuzione
w/W	Incremento/decremento numero di avvolgimenti del toro
n/N	Incremento/decremento numero di vertici per ogni avvolgimento

* I comandi di navigazione dell'esercitazione 3 sono presenti anche qui.

Note: il codice è stato generato e testato utilizzando le seguenti specifiche:
- OS: Debian GNU/Linux 9 (stretch)
- IDE: CLion 2018.3.4
All'interno della funzione `init()` viene settata la working directory corrente, se si utilizzano parametri diversi da quelli indicati è possibile sia necessario riadattarla.

UPGRADE: integrazione tool di navigazione
I tool di navigazione integrati sono: zoom, pan orizzontale e verticale, traslazione/rotazione/ridimensionamento per ogni oggetto in scena sia in *WCS* che in *OCS*.
Extra: gli shader applicabili alla mesh toro sono accessibili dal menu sotto la voce "Torus options".

1. Lighting
La modifica della posizione della sorgente luminosa avviene all'interno della funzione `modifyModelMatrix()`. Più precisamente si è moltipilicata la matrice corrente risultante dalla trasformazione della sorgente di luce, per la posizione precedente della luce, ottenendo la nuova posizione post-trasformazione. In questo modo la sorgente sarà posizionata correttamente sia dopo una traslazione che una rotazione in *WCS*.

$$l_1 = MV * l_0$$

2. Shading
Lo shading **Phong** è stato implementato tramite i file *v_phong.glsl* e *f_phong.glsl*. Il vertex shader calcola e ritorna in output tre vettori:

- **eye**: un vettore che dal vertice in analisi punta verso la camera.
- **light**: un vettore che dal vertice in analisi punta verso la sorgente luminosa, passata come input allo shader.
- **normal**: il vettore normale al vertice in analisi.

Il fragment shader utilizza questi vettori per calcolare le quattro componenti che formano il colore finale del frammento:

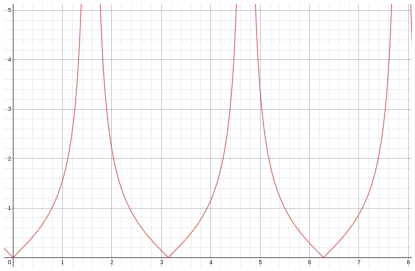
- **emissiva**: in questo caso risulta essere assente.
- **ambiente**: $I_a \; k_a$
- **diffusiva**: $k_d \; (l \cdot n) \; I_l$
- **speculare**: $k_s \cdot I_l \cdot (v \cdot r)^{n_s}$

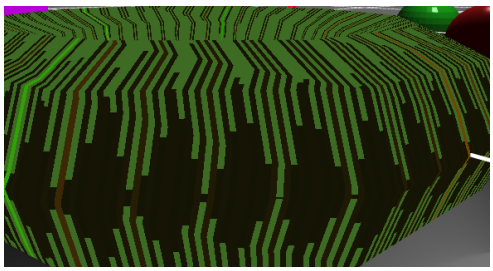
Infine questi elementi sono sommati generando il colore restituito in output.

3. Texture mapping 2D
In questo caso il vertex shader si limita a restituire in output la coordinata texture del vertice, mentre il fragment shader ottiene il colore del texel nella posizione passata dal vertex shader, tramite la funzione `texture()`.

4. Texture mapping 2D + Shading
Questo shader è l'esatta unione dei punti 3 e 4, non sono presenti modifiche rilevanti.

5. Procedural mapping
Questo tipo di texture è generata nella funzione `torusProceduralMapping()`. Lo scopo è quello di simulare il dorso di un rettile tramite i valori assunti dal `valore assoluto` della funzione `tangente`, disegnati in modo speculare al bordo di partenza della texture per toro. Le striscie presenti con valori casuali ma sempre in tonalità verde-marrone sono volute, per dare maggior realismo.





6. Wave motion
La composizione del vertex e fragment shader è del tutto simile al punto 2, con la differenza che nel vertex shader è stata applicata la formula fornita nelle specifiche per creare l'effetto onduoso modificando la coordinata y della posizione del vertice. È stata aggiunta la variabile `time` utilizzando la funzione `glutGet(GLUT_ELAPSED_TIME))` che ritorna il numero di millisecondi trascorsi da quando è stata chiamata `glutInit()`, quindi dall'avvio del programma.

OPZIONALE: Toon shading
Nel vertex shader sono calcolati il vettore che punta alla sorgente luminosa e la normale al vertice. Nel fragment shader sono stati presi in considerazione 5 "strati" di differenti tonalità in base all'intensità di luce ricevuta.