Lab 6 - Shaders

6.1 Wave Motion

Per questa applicazione era richiesto di implementare tre punti:

- usando il tempo e data l'ampiezza e la frequenza, si richiedeva di applicare una perturbazione su asse y ad ogni vertice dell'oggetto utilizzando una certa formula
- al click sinistro, modificare l'ampiezza
- al click destro modificare la frequenza

Il primo punto è stato applicato nel file `v.glsl` poiché si tratta di un modifica a livello di vertice. Si è semplicemente applicata la formula al vertice utilizzando `gl_Vertex` per ottenere la posizione del vertice e `gl_Position` per fornirgli la nuova generata dalla formula.

Le modifiche successive riguardano il file `wave.c`. Attraverso `glGetUniformLocation`, nella `init` si ottengono i valori di ampiezza, frequenza e tempo. Mentre nella `keyboard`, attraverso la `glUniform1f` si fornisce al vertice un nuovo valore di frequenza o di ampiezza a seconda del click dell'utente. Il valore randomico viene ottenuto grazie alla selezione casuale di un indice del vettore `values` che contiene i valori forniti dal testo.
Di seguito, il risultato:

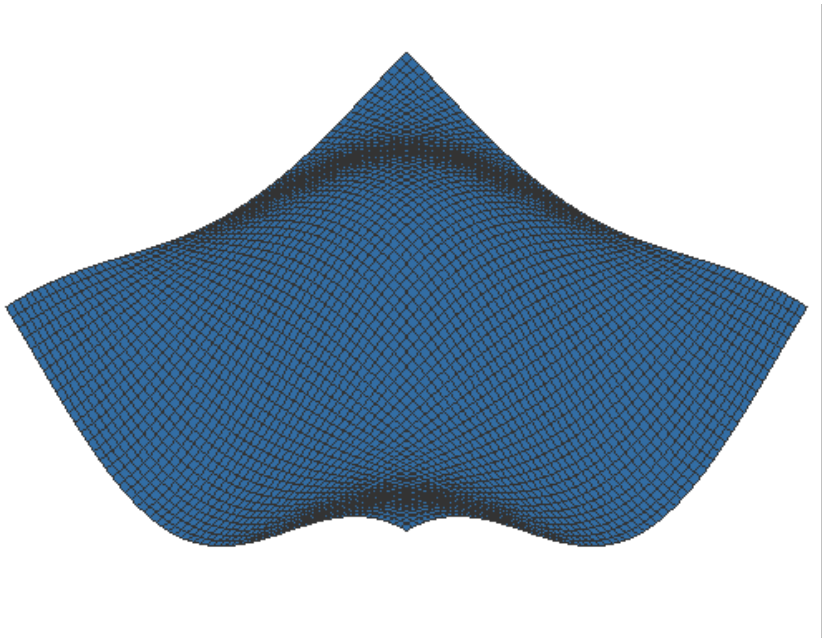


Figura 6.1

6.2 Particle System

L'applicazione del sistema particellare richiedeva di:

- modificare dimensione delle particella a seconda dell'altezza
- muovere le particelle anche nella direzione dell'asse z.

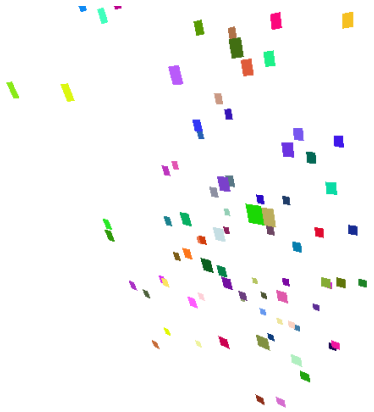


Figura 6.2

Il primo punto è stato implementato modificando la `gl_PointSize` a livello di vertex. Per farlo è necessario abilitare `GL_POINT_SPRITE` e `GL_VERTEX_PROGRAM_POINT_SIZE` e assegnarli un valore composto dal contributo della posizione del vertice negli assi `y` e `z`.

Il secondo punto invece richiedeva l'elaborazione dell'attributo `vzParam` e di gestire la condivisione di esso con tutti i vertex. È stato quindi gestito attraverso le funzioni `glGetAttribLocation` nella `init` per ottenere la possibilità di modificarlo e `glVertexAttrib1f` nella `draw` per poterlo modificare effettivamente con la velocità del vettore `velocity`.

6.3 Phong Lighting

L'obiettivo della terza applicazione è quello di creare un terzo shader che utilizzi il vero raggio riflesso e non l'Half-Way. A livello di vertex si è quindi ricavato il raggio riflesso R attraverso la funzione `reflect` tra L e N. A livello di fragment si sostituisce Half con il vettore Ray appositamente creato (fornendogli anche un fattore di shininess diverso per marcare le differenze tra i due modelli).
Per quanto riguarda il secondo punto, che richiedeva di rappresentare i tre diversi modelli in un'unica scena, è stata modificata la `draw` in modo da disegnare tre teapot identiche, ma con l'abilitazione di uno dei 3 diversi shader per ogni teapot. È stata inoltre allargata la finestra ed eliminata la gestione della tastiera poiché non serve più cambiare tipo di shader alla pressione dei tasti 1,2,3 in quanto i modelli sono tutti disegnati contemporaneamente. Un esempio del risultato:

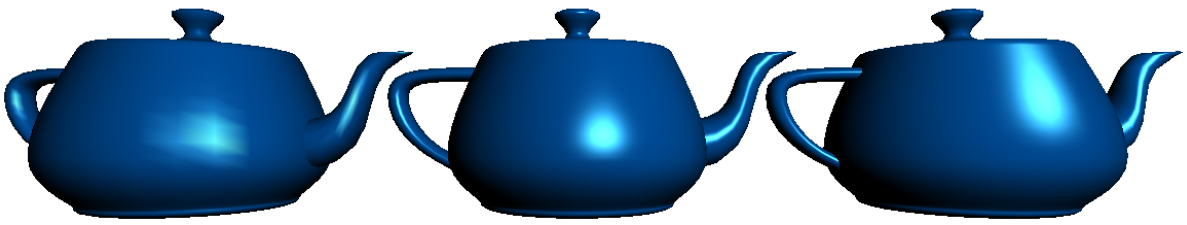


Figura 6.3

6.4 Toon Shading

La quarta applicazione richiedeva di disegnare un'outline sulla teapot in stile cartone animato.

L'idea di base è quella di cambiare colore ai vertici la cui normale forma un angolo maggiore di un certo angolo con il vettore che congiunge vertice e punto di vista. Quindi occorre ottenere per ogni vertex, il vettore E che va dal punto di vista al vertex stesso e ottenere l'angolo utilizzando l'arcocoseno del prodotto scalare tra questi due vettori normalizzati. Il vettore E viene ottenuto a livello di vertex, mentre a livello di fragment si calcola l'angolo e si applica il colore se quest'ultimo supera un certo angolo, ad esempio 74°. La funzione `acos` prende in ingresso un valore compreso tra 0 e π, quindi per rappresentare i 74 gradi è necessario passargli $3.14/10 * 4.1$ ovvero $180/10 * 4.1 = 73.8$.

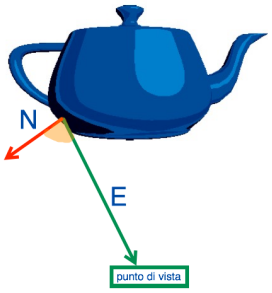
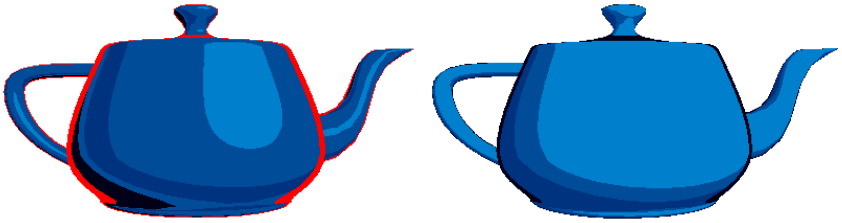


Figura 6.4

Due esempi del risultato:



(a) 74°, rosso (b) 80°, nero con altre modifiche colore

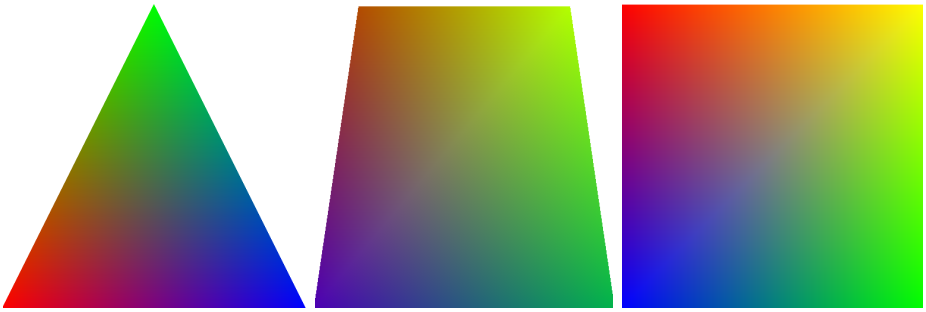
6.5 Morphing

In questa applicazione era richiesto di applicare il colore al triangolo e variare il colore in funzione del tempo. In alternativa era possibile modificare la forma da triangolo a quadrato. Sono state implementate entrambe le interpolazioni tra colore e forme. A livello di vertex, è stato aggiunto il vettore dei colori che deve assumere il quadrato ed è stata modificata la modalità con cui viene fornito il colore al vertice: analogamente ai vertici, si è utilizzata la funzione `mix` tra il colore corrente e il colore finale a seconda del tempo. Nel sorgente C, è stata modificata la funzione `draw` in modo che disegnasse quadrati (`GL_QUAD`) e all'interno, per ogni vertice del quadrato:

- si fornisce il colore del vertice
- si fornisce al vertex, la posizione finale e il colore finale
- si disegna il vertice iniziale

La figura iniziale viene rappresentata come triangolo, ma in realtà un quadrato con due vertici coincidenti. Attraverso l'interpolazione, i due vertici divergono e vanno a posizionarsi in modo da formare il quadrato finale.

Il risultato finale:



(a) Triangolo iniziale (b) Durante l'interpolazione (c) Quadrato finale

6.6 Bump Mapping

L'obiettivo di questa applicazione era modificare l'input data per ottenere un effetto particolare di bump mapping sulla superficie. È stato quindi diviso il quadrato in quattro parti, ognuna con un particolare valore randomizzato o ottenuto tramite seni, coseni o modulo. Il risultato è il seguente:

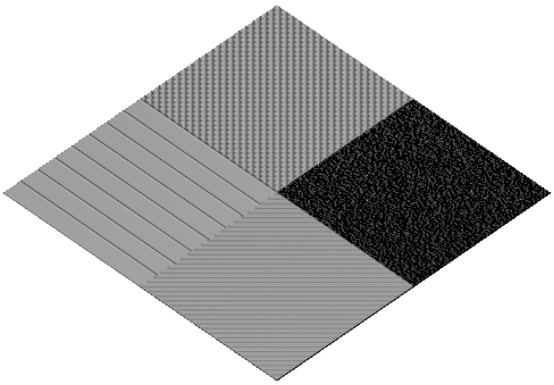


Figura 6.5

6.7 Cube environment mapping

Nell'ultimo esercizio era richiesto di sostituire i colori
del cubo con immagini ed applicare l'environment cube mapping. Sono state scelte sei immagini dell'esercitazione scorsa e sono state mappate sul cubo che andrà ad applicare la texture alla teapot. Il funzionamento è analogo a quell'esercitazione scorsa.

Un esempio del risultato:

(a) (b)