Relazione laboratori

Corso di Computer Graphics - Università di Bologna

Daniele Polidori
daniele.polidori2@studio.unibo.it

a.a. 2022-23

1 LAB-01

Punto 4b. Nello svolgimento del primo laboratorio, per quanto riguarda il punto 4, ho scelto di svolgere l'opzione *b*.
Oltre ai VAO già in uso per la curva di base, ho creato una seconda coppia di VAO per disegnare il tratto cubico che di volta in volta viene creato. Questo, una volta completato, viene attaccato alla curva di base: i punti relativi al tratto cubico vengono aggiunti a quelli della curva di base, così da liberare lo spazio per un possibile nuovo tratto cubico, e così via.
L'utente può scegliere la continuità con cui attaccare i tratti cubici alla curva di base: la continuità C^0 premendo il tasto *0* (scelta di default), C^1 premendo il tasto *1* e G^1 premendo il tasto *g*. Tale continuità rimarrà selezionata finché non ne verrà scelta una differente.
I tratti successivamente creati verranno raccordati con la continuità selezionata. Siano $p_0, \ldots, p_n$ i punti della curva di base e $v_0, \ldots, v_3$ i punti del tratto cubico, le continuità vengono applicate nella maniera seguente. v_0 viene eliminato, sarà infatti sostituito da p_n. A partire dalle coordinate di p_{n-1} e p_n vengono calcolate le coordinate che deve assumere il punto v_1, così che la continuità venga soddisfatta. Viene applicata la formula (1) per ottenere la continuità C^1 e la formula (2) per la continuità G^1: tali formule vengono applicate separatamente sulle coordinate x e sulle y dei rispettivi punti, con $\Delta = p_3 - p_2$. Per la continuità G^1, il punto verrà posizionato in modo tale che il tratto $v_1 - p_n$ sia lungo la metà del tratto $p_n - p_{n-1}$. Infine, i punti v_1, v_2 e v_3 vengono aggiunti alla curva di base, in qualità di, rispettivamente,

p_{n+1}, p_{n+2} e p_{n+3}.

$$v_1' = p_n + \Delta = 2 * p_n - p_{n-1} \tag{1}$$

$$v_1' = p_n + \frac{\Delta}{2} = p_n + \frac{p_n - p_{n-1}}{2} \tag{2}$$

Punto 5. Per realizzare lo spostamento dei punti tramite trascinamento con il mouse, mi sono servito delle funzioni callback di OpenGL: `glutMouseFunc()` e `glutMotionFunc()`.
Tramite la prima, quando viene premuto il tasto destro del mouse, controllo se mi trovo sopra un punto. Per facilitare la presa, considero un intorno delle coordinate dei punti: dato un punto (x, y), anziché le coordinate strette x e y, cerco dei valori più laschi, rispettivamente negli intervalli $[x - 0.01, x + 0.01]$ e $[y - 0.01, y + 0.01]$. Se il mouse si trova sopra un punto, questo viene "agganciato" (i.e. un puntatore punta alla sua cella di memoria) e quando poi rilascio il tasto, il punto viene "sganciato".
Tramite la seconda, invece, catturo la posizione del mouse durante il trascinamento del punto. Se un punto è stato agganciato, sostituisco le coordinate di tale punto con le coordinate del mouse in ogni istante. I valori continuano a mutare fino a che il punto non viene sganciato, ovvero finché il tasto del mouse non viene rilasciato.

2 LAB-02

Nello svolgimento del secondo laboratorio, ho deciso di riprodurre un cielo notturno dove si trovano dieci piccole fonti di luce (detti aloni) e un cerchio di luce più grande (detto sole), che segue il movimento del mouse.
Quando il sole (i.e. il mouse) passa sopra un alone, quest'ultimo scompare (dando l'effetto di venire inglobato dal sole). Se il sole tocca uno dei bordi dello schermo, si incorre in una penalità: tutti gli aloni inglobati tornano alla loro posizione iniziale (in questo modo la partita ricomincia da capo).
Lo scopo del gioco è inglobare (i.e. passare con il sole sopra) tutti gli aloni.
Presentiamo ora il modo in cui sono stati realizzati i vari oggetti presenti nella scena: il cielo, gli aloni, il sole e i sistemi particellari.

Cielo. Ho preso l'oggetto del cielo dal programma `2D_Jumping_Ball.cpp` e l'ho collegato a `VAO_CIELO` e a `VAO_ANIMAZIONECIELO`. Ne ho modificato la posizione e la scala (modificando l'input delle funzioni `translate()` e `scale()`, nella funzione `drawScene()`), in modo tale da riempire tutto lo schermo a disposizione. Ne ho modificato anche il colore (cambiando il valore

delle variabili `col_top` e `col_bottom`, nella funzione `init()`), così da rendere il cielo notturno.
Ho aggiunto poi un'animazione al cielo. Per farlo, ho preso l'oggetto dell'alone del sole dal programma `2D_Jumping_Ball.cpp`. Ne ho modificato la scala (modificando l'input della funzione `scale()`, nella funzione `drawScene()`), in modo tale da ingigantire l'alone nello scermo. Ne ho modificato il colore e la trasparenza (cambiando il valore della variabile `col_bottom_sole`, nella funzione `disegna_luce()`), così da renderlo un chiarore vagamente riconoscibile. Infine, nella funzione `drawScene()`, ho creato contemporaneamente cinque aloni nel cielo, in posizioni casuali. Questi, a intervallo di tempo regolare, cambiano posizione, sempre in maniera casuale. L'effetto ottico che ne deriva è quello di un cielo dinamico, mostrando un vago movimento indistinguibile sullo sfondo.

Aloni. Ho preso l'oggetto dell'alone del sole dal programma `2D_Jumping_Ball.cpp` e l'ho collegato a `VAO_SOLE`. Ne ho modificato la scala (modificando l'input della funzione `scale()`, nella funzione `drawScene()`), in modo tale da renderlo una piccola fonte di luce. Ne ho modificato anche il colore (cambiando il valore della variabile `col_bottom_sole`, nella funzione `disegna_sole()`), rendendolo più chiaro.
All'inizio della partita, nella funzione `drawScene()`, creo contemporaneamente dieci aloni nel cielo, in posizioni casuali. Successivamente vengono mostrati soltanto quelli che ancora non sono stati inglobati.

Sole. Ho preso l'oggetto del sole (con il suo alone) dal programma `2D_Jumping_Ball.cpp` e l'ho collegato a `VAO_SOLE`. Ne ho modificato la scala (modificando l'input della funzione `scale()`, nella funzione `drawScene()`), per renderlo leggermente più piccolo. Ne ho modificato anche il colore (cambiando il valore delle variabili `col_top_sole` e `col_bottom_sole`, nella funzione `disegna_sole()`), rendendolo più chiaro.
Infine, ho agganciato il sole al movimento del mouse: ho sostituito l'input della funzione `translate()` con delle variabili che memorizzano la posizione attuale del mouse (il loro valore viene continuamente aggiornato attraverso la funzione callback di OpenGL `glutPassiveMotionFunc()`).
In caso di vittoria, l'alone del sole viene ingrandito progressivamente (senza mai fermarsi). Per realizzare questo effetto ho usato cinquanta chiamate ricorsive alla funzione `aumentaScala_aloneSole()`, attraverso la funzione `glutTimerFunc()`, ciascuna delle quali applica un piccolo incremento alla scala dell'alone del sole.

Sistemi Particellari. Ho preso l'oggetto del sistema particellare dal programma `2D_PS.cpp` e l'ho collegato a `VAO_SISTEMAPARTICELLARE`. Tale oggetto è stato utilizzato in più punti.
Come prima cosa, l'ho collegato al movimento del mouse (attraverso la funzione `glutPassiveMotionFunc()`) e ne ho cambiato il colore (impostando il giallo nel valore della variabile `rgb`, presente nella funzione).
Inoltre, nella funzione `drawScene()`, ho creato, in ciascuna posizione degli aloni, un piccolo sistema particellare di colore arancione (indicato nel valore della variabile `rgb`, presente nella funzione). Tale sistema particellare viene mostrato soltanto nelle posizioni degli aloni che non sono stati ancora inglobati.
Infine, ho utilizzato questo oggetto per segnalare visivamente un'azione negativa, quando essa viene compiuta dall'utente. Infatti, quando il sole tocca un bordo dello schermo, il cielo si riempie di un sistema particellare di colore rosso (indicato nel valore della variabile `rgb`, nella funzione `glutPassiveMotionFunc()`), diffuso uniformemente in tutta l'area dello schermo.