Report LAB01

Nicola Modugno

prof.ssa Serena Morigi

Abstract—In questo elaborato viene descritto lo sviluppo e l'estensione del programma `LAB_01.cpp`, mirato alla manipolazione interattiva di punti di controllo su un piano 2D tramite OpenGL e GLFW. A partire da una versione base che consente l'inserimento e la cancellazione di punti, sono state implementate una curva di Bézier mediante l'algoritmo di de Casteljau e un sistema di modifica dinamica dei punti tramite trascinamento. In alternativa, è stata inoltre integrata l'interpolazione Catmull-Rom per ottenere una curva fluida che attraversa i punti di controllo. Il progetto dimostra l'efficacia delle callback GLFW nella gestione degli eventi e nella modellazione interattiva di geometrie.

1. Introduzione

Il programma di partenza `LAB_01.cpp` permette all'utente di inserire punti di controllo (Control Points, CP) in una finestra grafica 2D mediante clic del mouse, visualizzando un poligono di controllo che connette i punti tramite segmenti. I punti vengono memorizzati in un array bidimensionale `vPositions_CP[MaxNumPts][2]` contenente le coordinate (x, y).

La gestione dell'input avviene tramite callback di GLFW, che permettono il rilevamento e la gestione in tempo reale degli eventi dell'utente. Il sistema consente la cancellazione dei punti di controllo premendo il tasto `f` (rimozione del primo punto) o `l` (rimozione dell'ultimo punto), attraverso la funzione `key_callback()`.

Le specifiche dell'esercizio richiedevano di:

1. Testare i controlli da tastiera e mouse.
2. Analizzare l'uso delle callback GLFW per la cattura degli eventi.
3. Implementare la curva di Bézier tramite l'algoritmo di de Casteljau.
4. Consentire il trascinamento dei punti di controllo.
5. Integrare nel programa in alternativa una tra le seguenti
   (a) disegno di una curva di B´ezier interpolante a tratti (Catmull-Rom Spline)
   (b) disegno di una curva di B´ezier mediante algoritmo ottimizzato basato sulla suddivisione adattiva
   (c) disegno di una curva di B´ezier composta da tratti cubici, dove ogni tratto viene raccordato con il successivo con continuità C0,C1, o G1 a seconda della scelta utente (da keyboard).

2. Utilizzo delle Callback di OpenGL e GLFW

Il progetto è strutturato in due file principali: `LAB_01.cpp` che gestisce la scena e contiene le strutture dati e `gestione_callback.cpp`che contiene le callback per la gestione dell'input. Le due parti comunicano attraverso variabili globali, come `vPositions_CP`, `mouseOverIndex` e `isMovingPoint`. La base del funzionamento interattivo è rappresentata dall'impiego delle callback offerte da GLFW, che consentono di reagire agli eventi generati dall'utente, come il movimento del mouse, la pressione di un tasto o un clic. Queste callback sono state assegnate a funzioni specifiche: `mouse_button_callback()` rileva i clic del mouse per inserire nuovi punti o attivarne la modifica; `cursor_position_callback()` permette l'aggiornamento in tempo reale delle coordinate durante il trascinamento; infine, `key_callback()` interpreta la pressione dei tasti `f` e `l` per eliminare i punti. L'interazione tra utente e grafica è resa possibile da un ciclo di rendering continuo che aggiorna la scena dopo ogni modifica. Per rilevare se il mouse si trova sopra un punto esistente, si scorre l'array `vPositions_CP` e si confronta la distanza con il cursore, secondo la formula euclidea:

$$d = \sqrt{(x_{mouse} - x_i)^2 + (y_{mouse} - y_i)^2}$$

Se $d < 0.06$, il punto viene considerato "attivo" e modificabile. In questo caso, `mouseOverIndex` viene aggiornato con l'indice del punto selezionato. Durante il trascinamento, la variabile `isMovingPoint` è impostata a `true`, e `cursor_position_callback` richiama la funzione `modifyPoint()` per aggiornare in tempo reale la posizione del punto selezionato. Al rilascio del tasto del mouse, `isMovingPoint` viene settata a `false`, confermando la nuova posizione.

3. Curva di Bézier: Algoritmo di de Casteljau

La funzione seguente implementa l'algoritmo di de Casteljau per calcolare un punto sulla curva di Bézier.

```
void deCasteljau(float t, float* result) {
    float coordX[MaxNumPts], coordY[MaxNumPts];

    for (int i = 0; i < NumPts; i++) {
        coordX[i] = vPositions_CP[i][0];
        coordY[i] = vPositions_CP[i][1];
    }

    for (int i = 1; i < NumPts; i++) {
        for (int k = 0; k < NumPts - i; k++) {
            coordX[k] = (1 - t) * coordX[k] + t * coordX
    ↪ [k + 1];
            coordY[k] = (1 - t) * coordY[k] + t * coordY
    ↪ [k + 1];
        }
    }

    result[0] = coordX[0];
    result[1] = coordY[0];
}
```

Code 1. Algoritmo di de Casteljau

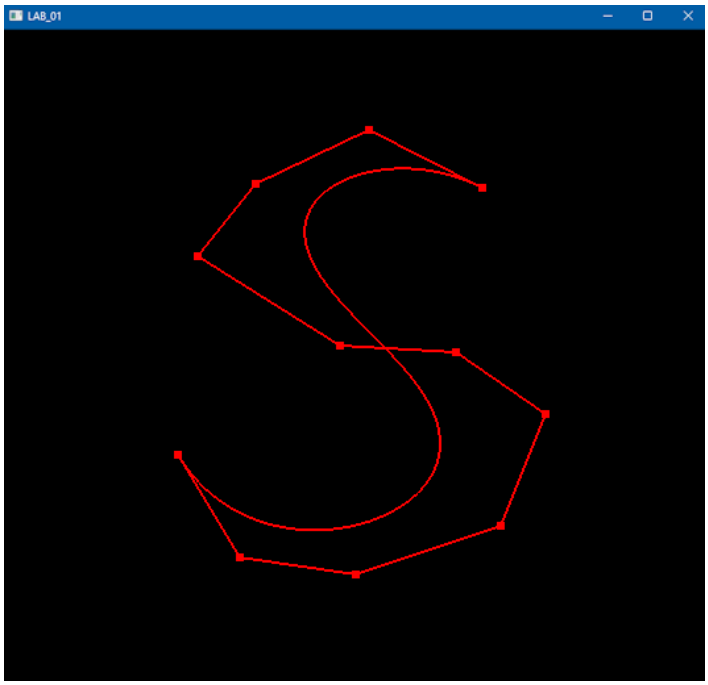

Figure 1. Resa della curva con Bézier

L'algoritmo funziona copiando inizialmente le coordinate dei punti in due array temporanei. Successivamente, esegue iterazioni di interpolazione lineare tra coppie di punti adiacenti, fino a ottenere un

singolo punto, che corrisponde alla posizione finale della curva per un determinato parametro t. Sebbene la funzione sia correttamente funzionante, nel programma finale è stata sostituita con un'alternativa più coerente con la richiesta del laboratorio. Sebbene implementata, nel programma finale la curva di Bézier è stata commentata per lasciare spazio alla Catmull-Rom.

4. Catmull-Rom Spline

In base alla traccia, è stata scelta l'opzione **(a)**: la curva Catmull-Rom interpolante.

La curva Catmull-Rom è una spline interpolante a tratti che garantisce continuità nella prima derivata (C1) ed è progettata per passare attraverso i punti di controllo. A differenza delle curve di Bézier, che tendono a "stare dentro" il convesso dei punti di controllo, la Catmull-Rom attraversa i punti generando una curva più aderente al profilo desiderato.

Matematicamente, ogni segmento della curva viene descritto da un polinomio cubico parametrico della forma:

$$p(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3, \quad u \in [0,1]$$

In questo contesto, ogni tratto della curva viene calcolato utilizzando quattro punti consecutivi p_0, p_1, p_2, p_3 appartenenti all'insieme dei punti di controllo. Il tratto in esame parte da p_1 e termina in p_2, ovvero $p(0) = p_1$ e $p(1) = p_2$.

Per ottenere la forma esplicita del polinomio, si parte da alcune condizioni di interpolazione e continuità. Da esse si deduce che:

$$\begin{aligned} p(0) &= c_0 = p_1 \\ p(1) &= c_0 + c_1 + c_2 + c_3 = p_2 \\ c_1 &= \frac{1}{2}(p_2 - p_0) \\ c_1 + 2c_2 + 3c_3 &= \frac{1}{2}(p_3 - p_1) \end{aligned}$$

Attraverso questi vincoli, si possono determinare in modo univoco i coefficienti del polinomio. Nel programma `LAB_01.cpp`, la funzione `catmullRom()` riceve come parametro il valore normalizzato $t \in [0,1]$ e calcola il punto corrispondente sulla curva nel segmento definito da quattro punti adiacenti. Di seguito si mostra il codice implementato:

```
void catmullRom(float t, float* result) {
    if (NumPts < 4) return;

    int segment = (int)(t * (NumPts - 3));
    float localT = (t * (NumPts - 3)) - segment;

    if (segment < 0 || segment >= NumPts - 3) return;

    float* p0 = vPositions_CP[segment];
    float* p1 = vPositions_CP[segment + 1];
    float* p2 = vPositions_CP[segment + 2];
    float* p3 = vPositions_CP[segment + 3];

    float t2 = localT * localT;
    float t3 = t2 * localT;

    result[0] = 0.5f * (
        (2.0f * p1[0]) +
        (-p0[0] + p2[0]) * localT +
        (2.0f * p0[0] - 5.0f * p1[0] + 4.0f * p2[0] - p3[0]) * t2 +
        (-p0[0] + 3.0f * p1[0] - 3.0f * p2[0] + p3[0]) * t3
    );

    result[1] = 0.5f * (
        (2.0f * p1[1]) +
        (-p0[1] + p2[1]) * localT +
        (2.0f * p0[1] - 5.0f * p1[1] + 4.0f * p2[1] - p3[1]) * t2 +
        (-p0[1] + 3.0f * p1[1] - 3.0f * p2[1] + p3[1]) * t3
    );
}
```

Code 2. Algoritmo di Catmull-Rom

Questa funzione viene integrata all'interno della funzione `drawScene()`, dove la curva viene costruita punto per punto. Il parametro t viene suddiviso in 100 intervalli equidistanti, e ad ogni iterazione il punto calcolato viene memorizzato nell'array `vPositions_C`, il quale verrà successivamente inviato alla GPU per la visualizzazione. Nella Figura 2 è mostrata la resa della curva ottenuta mediante Catmull-Rom Spline. Confrontandola con la Figura 1 si può osservare come la curva Bézier tende a non passare per i punti di controllo, limitandosi risulando molto più approssimativa, mentre la Catmull-Rom li attraversa, rendendo l'interazione visiva più coerente con l'input dell'utente.

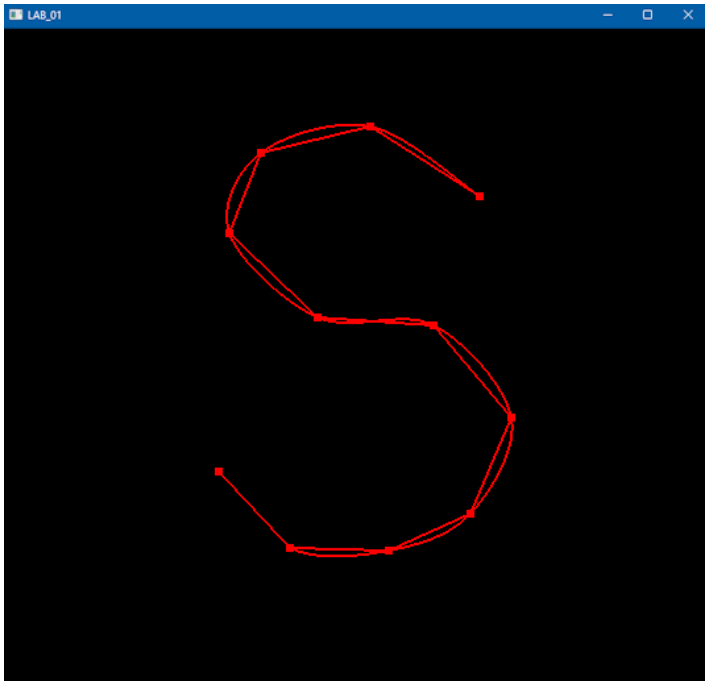

Figure 2. Resa della curva con Catmull-Rom Spline