Computer Graphics - Relazione Laboratorio 1

Keran Jegasothy

Indice

1 Curva di Bézier con algoritmo di de Casteljau

1.1 Consegna

Disegnare la curva di Bézier a partire dai punti di controllo inseriti, utilizzando l'algoritmo di de Casteljau.

1.2 Soluzione

Innanzitutto, si è proceduto con l'analisi dell'algoritmo di deCasteljau.

$$
\begin{aligned}
&input: \quad p_i, t \qquad\qquad \text{n degree of the curve}\\
&p_i^0(t) = p_i \qquad\qquad \text{p}_\text{i} \text{ control points i=0,..,n}\\
&for \quad i = 1,..,n \qquad\qquad \text{t evaluation parameter}\\
&\quad for \quad j = 0,..,n-i\\
&\quad\quad p_j^i(t) = (1-t)p_j^{i-1}(t) + tp_{j+1}^{i-1}(t)\\
&\quad end\\
&end
\end{aligned}
$$

Successivamente, lo speudocodice sopra riportato è stato tradotto in codice eseguibile nell'ambiente di sviluppo compatibile con la libreria OpenGL. Più precisamente, è stata costruita una funzione che prende in input due valori float e applicando l'algoritmo di DeCasteljau determina il punto corrispondente alla curva.

```
void deCasteljau(float t, float* result) {
    int i, k;
    float coordX[MaxNumPts], coordY[MaxNumPts];
    for (i = 0; i < NumPts; i++) {
        coordX[i] = PointArray[i][0];
        coordY[i] = PointArray[i][1];
    }
    for (i = 1; i < NumPts; i++) {
        for (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];
    result[2] = 0.0;
}
```

2 Curva di Bézier con suddivisione adattiva

2.1 Consegna

Disegno di una curva di Bézier mediante algoritmo ottimizzato basato sulla suddivisione adattiva.

2.2 Soluzione

A seguito dell'analisi dell'algoritmo di suddivisione adattiva,è stato realizzato un metodo che applicasse tale algoritmo. Il codice è stato reperito da una repository ed opportunamente integrato all'interno del progetto.
A tale metodo, viene dato in input un'array e i punti di controllo. A partire dei punti di controllo situati nelle due estremità, vengono calcolati i coefficienti della retta. Mentre, per i punti interni, vengono calcolati le distanze dalla retta e se è maggiore di 0.01, la funzione viene richiamata ricorsivamente con l'array divisa in due, altrimenti i punti vengono presi in considerazione per il disegno della curva.

3 Modifica della posizione dei punti di controllo

3.1 Consegna

Permettere la modifica della posizione dei punti di controllo tramite trascinamento con il mouse.

3.2 Soluzione

La modifica della posizione dei punti di controllo avviene tramite le funzioni callback *passive* e *motion*.

```
void passive(int x, int y) {
    float xPos = -1.0f + ((float)x) * 2 / ((float)(width));
    float yPos = -1.0f + ((float)(height - y)) * 2 / ((float)(height));

    for (int i = 0; i < NumPts; i++) {
        float dist = sqrt(pow(PointArray[i][0] - xPos, 2) + pow(PointArray[i][1] - yPos, 2));
        if (dist < 0.03) {
            mouseOverIndex = i;
            glutPostRedisplay();
            return;
        }
        else {
            mouseOverIndex = -1;
        }
    }
}

void motion(int x, int y) {
    float xPos = -1.0f + ((float)x) * 2 / ((float)(width));
    float yPos = -1.0f + ((float)(height - y)) * 2 / ((float)(height));

    if (isMovingPoint) {
        PointArray[movingPoint][0] = xPos;
        PointArray[movingPoint][1] = yPos;
    }

    glutPostRedisplay();
}
```