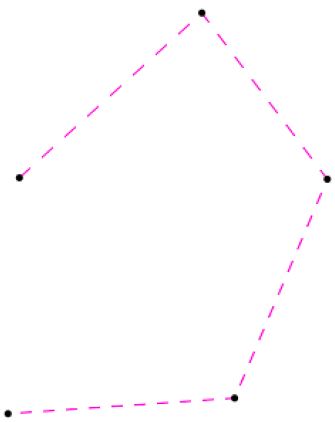
Fondamenti di Computer Graphics LM

Lab 1 – Introduzione a OpenGL e Curve di Bézier

Mattia Fucili

Questa prima esercitazione di laboratorio mirava ad introdurci nel mondo della libreria OpenGL attraverso piccoli esempi di codice che man mano dovevamo modificare per cercare di capire come funziona. Inoltre dovevamo implementare l'algoritmo di De Casteljau per disegnare una Curva di Bézier.

Obiettivo 1



Per prima cosa è stato necessario compilare ed eseguire il programma per capire cosa faceva e come veniva realizzato, quindi leggendo il codice.
Le funzionalità base erano quelle di inserimento di un punto con il click sinistro del mouse. Questo inserimento poteva essere fatto quante volte si voleva, ma il programma al più registrava 64 punti, in caso fossero di più venivano cancellati i primi punti inseriti. Tutti i punti venivano congiunti da un segmento tratteggiato. Nel caso si volessero togliere dei punti a mano è possibile farlo grazie alle due funzionalità fornite dal programma: tasto 'f' per togliere un punto in testa e tasto 'l' per togliere un punto inserito per ultimo. Questi due comandi vengono realizzati nella funzione che gestisce l'interazione con la tastiera nel seguente modo:

```
void keyboard (unsigned char key, int x, int y) {
    switch (key) {
        case 'f':
            removeFirstPoint();
            glutPostRedisplay();
            break;
        case 'l':
            removeLastPoint();
            glutPostRedisplay();
            break;
        case 27:     // Escape key
            exit (0);
            break;
    }
}
```

Infine per imporre il limite a 64 punti è stata definita una costante:

```
#define MAX_CV 64
```

ed ogni volta che si inserisce un nuovo punto si controlla che non siano più di 64 in quel caso viene tolto il primo utilizzando la funzione che implementa la rimozione dei primi punti:

```
if (numCV >= MAX_CV)
        removeFirstPoint ();
```

Obiettivo 2

Per inserire i punti nello schermo e per la loro memorizzazione è stata realizzata appositamente una funzione che cattura i click fatti dal mouse. Per prima cosa bisogna abilitare il fatto di voler “ascoltare” gli eventi che genera il mouse con questo comando:

```
glutMouseFunc(mouse);
```

dove come si può vedere è stata anche inserita una funzione di callback per gestire l’eventuale click del mouse. In questa funzione così definita:

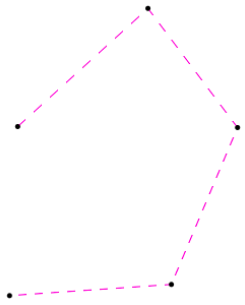
```
if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
    float xPos = ((float) x) / ((float) (WindowWidth - 1));
    float yPos = ((float) y) / ((float) (WindowHeight - 1));
    yPos = 1.0f - yPos;
    float zPos = 0;
    addNewPoint(xPos, yPos, zPos);
    glutPostRedisplay();
}
```

viene catturata la pressione del tasto sinistro del mouse attraverso il controllo delle due costanti definite dalla libreria GLUT. Per memorizzare la posizione del punto viene fatto un calcolo in base alla dimensione della finestra e viene invertito il valore di y perché nella libreria OpenGL l’origine degli assi viene preso in alto a sinistra della finestra. Infine viene messo il valore di z a 0 perché siamo in 2D.

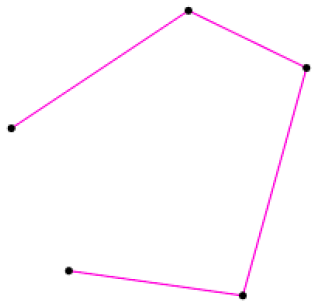
Obiettivo 3

In questo punto era necessario cercare di modificare lo stile della linea che congiunge i punti disegnati. Si modifica lo stile della linea con il seguente comando:

```
glEnable(GL_LINE_STIPPLE);
```



```
glEnable (GL_LINE_STRIP);
```



Obiettivo 4

Una prima rappresentazione di una Curva di Bézier ci è stata richiesta attraverso l'utilizzo di un algoritmo già fornito da OpenGL. Per fare ciò è stato necessario dapprima abilitare il disegno di curva inserendo il seguente comando:

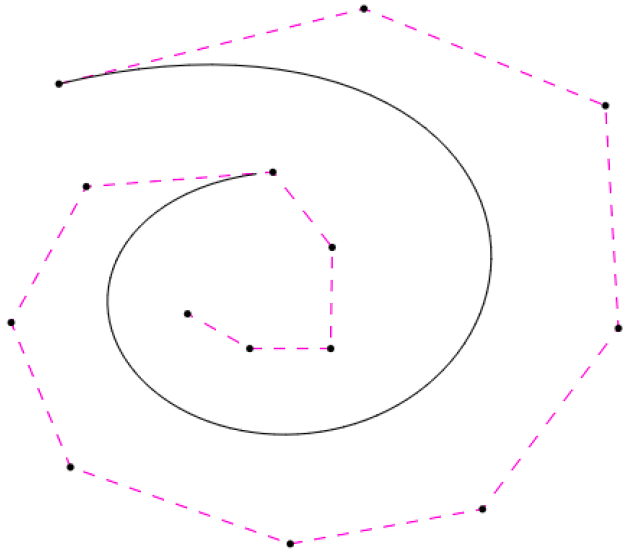
```
glEnable(GL_MAP1_VERTEX_3);
```

e poi disegnare la curva con il seguente metodo:

```
glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, numCV, &CV[0][0]);

for(int i = 0; i < 100; i++)
    glEvalCoord1f((GLfloat) i / 100.0);
```

il risultato è il seguente:

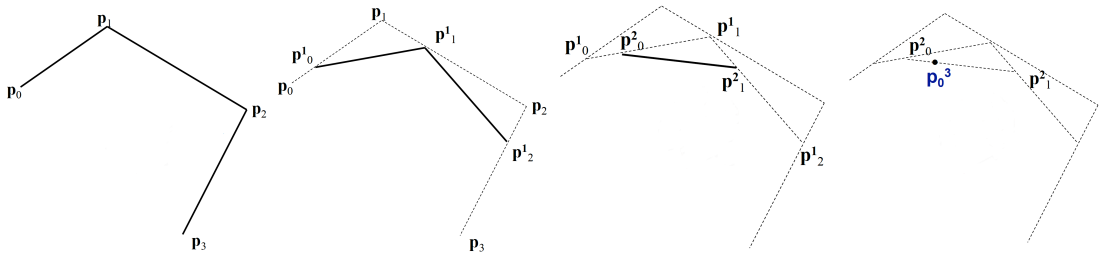


Il problema di questo algoritmo è che considera solo i primi 10 punti di controllo inseriti dopodiché non viene più aggiornata la curva derivante.

Obiettivo 5

In questo punto veniva richiesto a noi di implementare l'algoritmo di De Casteljau per risolvere i problemi derivanti da quello fornito di base dalla libreria.
Una Curva di Bézier, di grado 2 come serve a noi, è una curva costruita a partire dal seguente polinomio di controllo: $C(t) = P_0(1-t)^2 + P_1 2t(1-t) + P_2 t^2$ che in forma generale è così espresso:
$C(t) = \sum_{i=0}^{N} P_i B_i^n(t)\ con\ t \in [0,1]$ e $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$. Tanto più l'intervallo t viene suddiviso in punti tanto più la curva che ne deriva è precisa. La particolarità di questa curva è che approssima i punti ma non li interpola, se non il primo e l'ultimo punto.

Per implementare questo tipo di curve è necessario seguire l'algoritmo di De Casteljau che prevede $n-1$ step dove n è il grado della Curva di Bézier. Questo algoritmo sfrutta l'interpolazione lineare definita: $Lerp(t,a,b) = (1-t)a + tb$.
Questa funzione ad ogni passo, in base al parametro t che viene passato, calcola un punto in mezzo ad ogni coppia di punti di controllo, fino ad arrivare ad ottenere un singolo punto. In questo passerà la Curva di Bézier.



Il codice è stato implementato nel seguente modo:

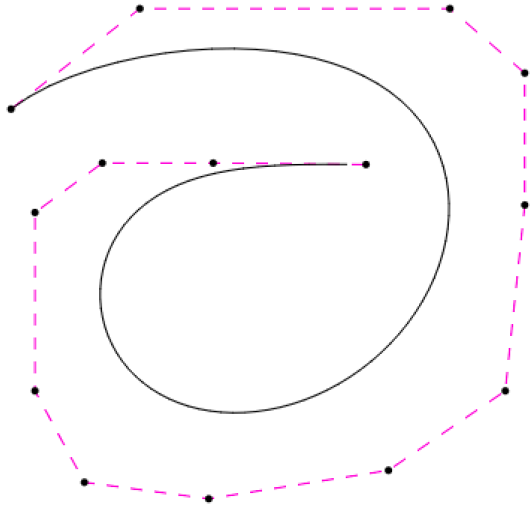
```
void decasteljau(float t, float* result) {
    float coords[MAX_CV][2];
    int i, j;

    for(i = 0; i < numCV; i++) {
        coords[i][0] = CV[i][0];
        coords[i][1] = CV[i][1];
    }

    for(i = 1; i < numCV; i++) {
        for(j = 0; j < (numCV - i); j ++) {
            coords[j][0] = (1 - t) * coords[j][0] + t * coords[j + 1][0];
            coords[j][1] = (1 - t) * coords[j][1] + t * coords[j + 1][1];
        }
    }

    result[0] = coords[0][0];
    result[1] = coords[0][1];
}
```

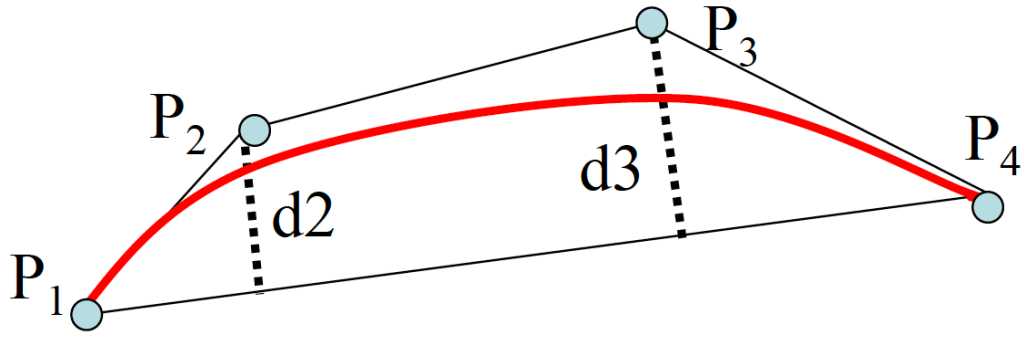
e da come risultato:



che come si può vedere tiene conto di più di 10 punti rispetto al vecchio algoritmo.
Viene invocato passando come t un valore che viene ogni volta incrementato di 0.01.

Obiettivo 6

In realtà l'algoritmo appena implementato presenta un grosso carico computazionale perché se due punti di controllo sono pressoché alla stessa altezza, quindi possono essere congiunti con una linea retta, questo algoritmo comunque applica De Casteljau e quindi crea 100 punti per ogni tratto della curva. Per risolvere questo problema si può ottimizzare l'algoritmo basandolo su una suddivisione adattativa. Questa ottimizzazione permette di controllare se due punti circa si trovano alla stessa altezza, rispetto ai punti di inizio e fine del tratto considerato, e quindi se possono essere congiunti da una linea retta.
Per implementare questo algoritmo ho pensato di calcolare la retta passante per due punti tramite questa formula $\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1}$ e calcolare la distanza di un punto dalla retta con l'opportuna formula $d(P,r) = \frac{|ax_p+by_p+c|}{\sqrt{a^2+b^2}}$ che graficamente assomiglia:



Dopo aver implementato le funzioni per i calcoli geometrici ho scritto questa funzione che sfrutta le precedenti funzioni per controllare se è possibile disegnare una linea retta oppure se è necessario applicare De Casteljau

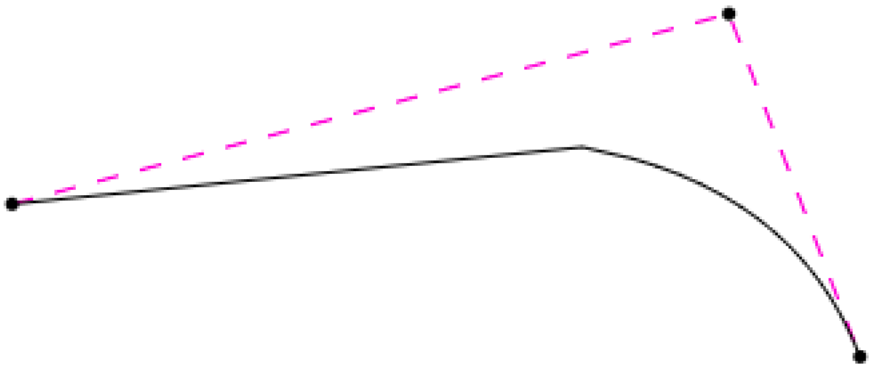
```
void straightLine(float points[MAX_CV][3], int n) {
    if(checkPoints(points, n)) {
        glVertex3f(points[0][0], points[0][1], points[0][2]);
        glVertex3f(points[n - 1][0], points[n - 1][1], points[n - 1][2]);
    } else {
        float firstHalfPoints[MAX_CV][3], secondHalfPoints[MAX_CV][3];

        adaptiveDeCasteljau(points, n, 0.5, firstHalfPoints, secondHalfPoints);

        straightLine(firstHalfPoints, n);
        float reversedSecondHalfPoints[MAX_CV][3];
        int index = 0;
        for(int i = n - 1; i >= 0; i--) {
            reversedSecondHalfPoints[index][0] = secondHalfPoints[i][0];
            reversedSecondHalfPoints[index][1] = secondHalfPoints[i][1];
            reversedSecondHalfPoints[index][2] = secondHalfPoints[i][2];

            index++;
        }
        straightLine(reversedSecondHalfPoints, n);
    }
}
```

Il risultato è il seguente:



Obiettivo 7

Per permettere la modifica della posizione dei punti di controllo tramite trascinamento con il mouse ho aggiunto una funzionalità in più nella callback del mouse dove si va a catturare il click destro del mouse.

```
if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        float xPos = ((float) x) / ((float) (WindowWidth - 1));
        float yPos = ((float) y) / ((float) (WindowHeight - 1));
        yPos = 1.0f - yPos;                 // Flip value since y position is from top row.
        float zPos = 0;

        for(int i = 0; i < numCV; i++) {
            if(CV[i][0] <= (xPos + 0.01) && CV[i][0] >= (xPos - 0.01) && CV[i][1] <= (yPos +
0.01) && CV[i][1] >= (yPos - 0.01) && CV[i][2] <= (zPos + 0.01) && CV[i][2] >= (zPos - 0.01))
{
                flag = true;
                pos = i;
                break;
            }
        }
    }
```

inoltre si va a controllare che il click sia avvenuto in un certo intorno compreso nella dimensione del punto di controllo, così da poter individuare esattamente quale si vuole spostare.
Infine ho aggiunto una nuova callback per la mouseMotion:

```
glutMotionFunc(mouseMotion);
```

dove mi salvo tutte le posizioni attraversate dal punto quando viene spostato e le disegno con il comando:

```
glutPostRedisplay();
```