Lab 1 - Curve di Bézier

I punti dell'esercitazioni che verranno trattati in questa relazione sono i seguenti:

- Sostituire alle routine di OpenGL il disegno della curva mediante algoritmo di De Casteljau.
- Integrare nel programma in alternativa uno dei seguenti punti:
  - disegno di una curva di Bézier mediante algoritmo ottimizzato basato sulla suddivisione adattiva.
  - disegno interattivo di una curva di Bézier composta da tratti cubici, dove ogni tratto viene raccordato con il successivo con continuità C_0, C_1, o G_1 a seconda della scelta utente da keyboard.
- Permettere la modifica della posizione dei punti di controllo tramite trascinamento con il mouse.

1.1 Struttura

1.1.1 Scenari

L'ultimo punto è stato implementato insieme agli altri due punti. Nel secondo punto si è scelto di implementare entrambi gli scenari. L'applicativo è quindi composto da 3 scenari selezionabili tramite la pressione dei tasti 1,2,3 dalla tastiera:

- Disegno di una curva di Bézier con algoritmo di De Casteljau
- Disegno di una curva di Bézier con suddivisione adattiva
- Disegno interattivo di più curve di Bézier con tre possibili tipi di continuità fra di esse

Per rappresentare il concetto di scenario, è stata utilizzata una enumerazione e una variabile *currentState* che mantiene lo scenario corrente. Nella funzione *keyboard* sono quindi stati aggiunti dei *case* per i tasti 1,2 e 3 che inseriscono il medesimo valore nella variabile *currentState* e richiamano la *glutPostRedisplay()* per ridisegnare la scena.

1.1.2 Highlight Point

Fin dal principio, si è mantenuto un occhio di riguardo all'ultimo punto dell'esercitazione, quindi si è scelto di implementare una funzione che evidenziasse il punto più vicino al puntatore del mouse, per renderlo ben visibile all'utente nel caso in cui volesse spostarlo. Per evidenziare il punto al passaggio del mouse è necessario inserire nel main la funzione *glutPassiveMotionFunc(highlightPoint)* che richiama appunto la funzione *highightPoint* ogni volta che il mouse

viene mosso. Questa funzione ricava la posizione del cursore sullo schermo e converte da *pixel* a coordinate *x,y* della scena ed esegue un controllo su tutti i punti ottenendo quello che approssimativamente ha le stesse coordinate della posizione del cursore. Questo punto verrà selezionato e messo dentro la variabile *selectedPoint*. Nella funzione *display()*, al momento di disegnare tutti i punti di controllo della curva, si applica il colore rosso tramite *glColor3f(1.0f, 0.2f, 0.2f)* e si ingrandisce il punto tramite *glPointSize(9)*.

1.1.3 Display

La funzione *display()* deve disegnare nell'ordine:

- **Punti di controllo**: l'unica modifica effettuata si ha al momento della definizione dello spessore e del colore del punto. Si applicano diversi colori e spessori nei seguenti casi in cui il punto corrente è:
  - quello selezionato dal cursore (già chiarito sopra)
  - un punto di giunzione nello scenario INTERACTIVE (verrà chiarito in seguito)
  - un normale punto di controllo interno.
- **Lati della poligonale di controllo**: l'unica alterazione si ha nel colore della poligonale.
- **La curva di Bézier**: prima di disegnare ogni punto della curva, viene fatto un controllo su quale scenario è stato selezionato:
  - **DE CASTELJAU**: viene semplicemente chiamata la routine *drawBezierCurve* per disegnare una curva di Bézier. Ne verranno illustrati i vari passi nella sezione Algoritmo di De Casteljau.
  - **SUBDIVISION**: analogamente richiama la funzione `drawSubdividedCurve` che si occupa di suddividere e di disegnare la curva con suddivisione adattiva.
  - **INTERACTIVE**: fa uso di una variabile globale *degree* che definisce il grado massimo per ogni curva e permette di disegnare curve consecutive di grado pari al suo valore. Utilizzando questo valore e il numero di punti di controllo totale, effettua calcoli e ottiene valori da dare in pasto alla routine *drawBezierCurve*, la quale viene chiamata in due casi: per ogni curva che ha raggiunto il grado massimo e per la curva che è attualmente in costruzione, che quindi non raggiunge il grado massimo per curva. Ulteriori dettagli verrano illustrati nella sezione Modalità interattiva

1.2 Algoritmo di De Casteljau

Questo scenario richiama semplicemente la routine `drawBezierCurve(int firstPointIndex, int curveDegree)`. Essa richiede l'indice (del vettore di punti di controllo) del punto da cui deve iniziare a disegnare la curva di Bézier e il grado che deve averes. La funzione inizialmente alloca un vettore temporaneo di dimensione *curveDegree+1* che manterrà tutti gli stadi dell'algoritmo di De Casteljau. Questo viene inizializzato con i punti di controllo che vanno da *firstPointIndex* a *curveDegree*. L'algoritmo di De Casteljau consiste in un'interpolazione eseguita tra un punto della curva e il successivo tramite il parametro t, come segue (pseudocodice):

```
for (int i = 1; i <= curveDegree; i++) {
  for (int j = 0; j <= curveDegree - i; j++) {
    temp[j] = Lerp(temp[j],temp[j+1],t); //per ogni coordinata
  }
}
```

dove `Lerp` è una semplice interpolazione richiamata per ogni punto e per ogni coordinata:

```
float Lerp(float start, float end, float t)
  return (1 - t) * start + t * end;
```

L'algoritmo viene ripetuto per ogni variazione del parametro t (di 0.01, contenuta nella variabile TPRECISION e modificabile) e al suo termine posiziona il punto da disegnare all'indice 0 del vettore temp, attraverso la primitiva *glVertex3f*. Questa primitiva, se ripetuta più volte all'interno di una sezione definita dalle primitive *glBegin(GL_LINE_STRIP)* e *glEnd()*, permette di disegnare la curva di Bézier.

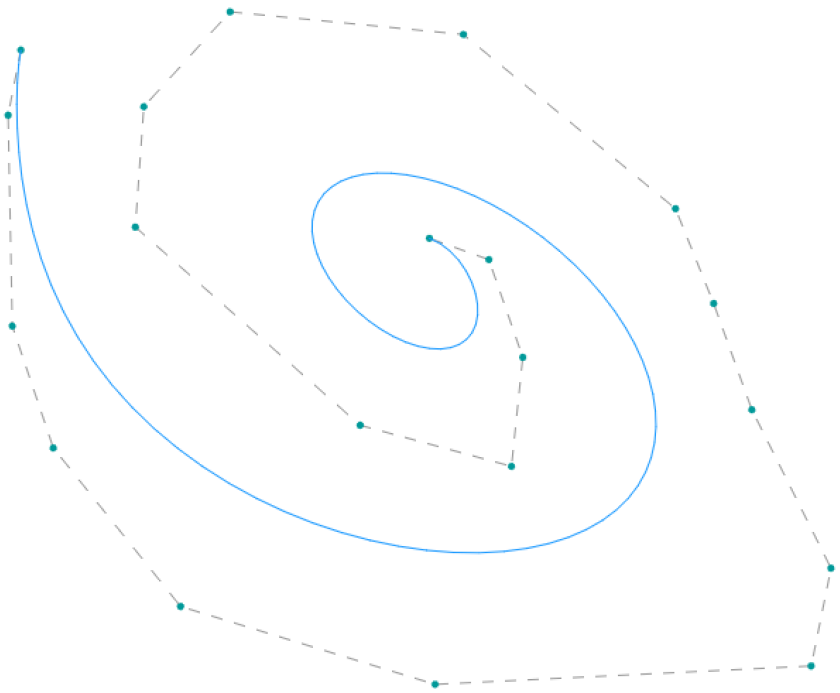


Figura 1.1: Un esempio del risultato

1.3 Suddivisione Adattiva

Questo scenario richiama la routine `drawSubdividedCurve(float points[MAX_CP][3])` che prende un vettore di punti che forma una curva e applica il *flatTest* a tutti i punti interni. Il test si basa sul confronto con un parametro TOL di tolleranza entro cui la curva è considerata *flat* e quindi deve essere disegnata. Se il test è positivo, la curva viene disegnata con un insieme di segmenti che collegano i punti del vettore. Se viceversa il test è negativo, viene suddivisa la curva in due sotto-curve con stesso numero di punti e viene richiamata ricorsivamente la ruotine `drawBezierCurve`. La suddivisione viene effettuata considerando ciò che si ottiene applicando *De Casteljau* con parametro $t = 0.5$. Utilizzando le slide del corso (figura 1.2), si può illustrare meglio come è possibile ottenere le due sotto-curve nel caso di una curva iniziale di 4 punti. La parte *gialla* rappresenta la curva iniziale, mentre la parte arancione rappresenta la prima sotto-curva e quella rossa, la seconda.

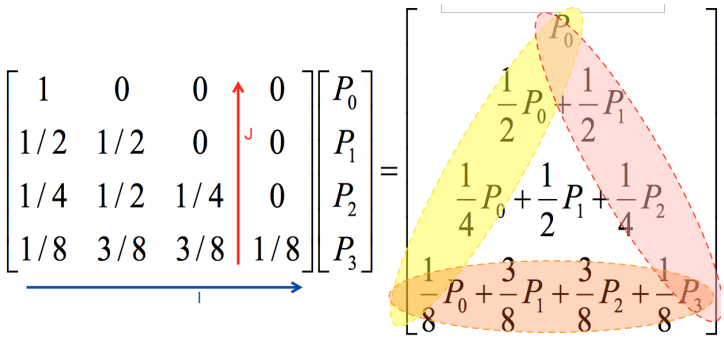
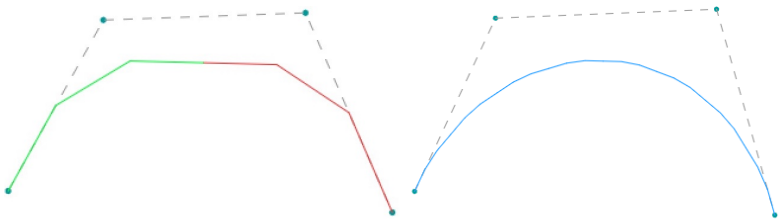


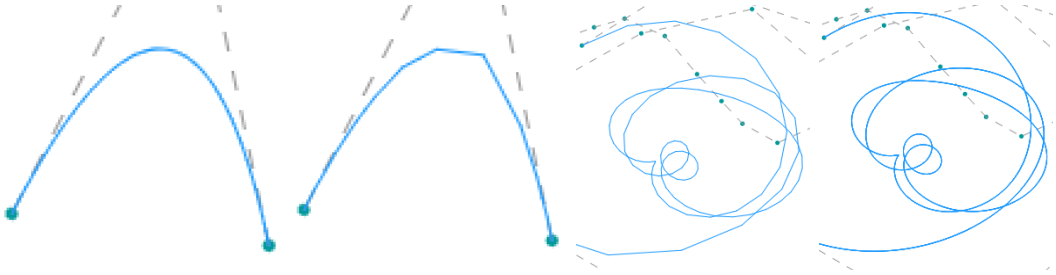
Figura 1.2

Un esempio del risultato:



(a) Con una sola subdivide (b) Risultato finale

Nota: se si utilizza un TOL elevato con questa modalità, in quanto approssimazione, si hanno scarsi risultati quando vi sono pochi punti, mentre quando ve ne sono molti si ha un ottimo riscontro visivo. Ecco un confronto con il normale algoritmo di De Casteljau con un parametro t con precisione 0.01.



(a) De Casteljau (b) Subdivision (c) De Casteljau (d) Subdivision

1.4 Modalità interattiva

Questo scenario permette di disegnare una curva costituita da più curve di Bézier con un grado massimo definito. Questo implica che durante la costruzione del curva vi siano sotto-curve *complete*, ovvero che raggiungono il grado massimo e una sola possibile sotto-curva *in costruzione*, ovvero che ha grado minore al grado massimo. In entrambi i casi viene richiamata la routine *drawBezierCurve*, ma con diversi parametri di ingresso. Per la curva in costruzione, fa uso di:

- **newCurveBegin**: indica l'indice del vettore dei punti di controllo in cui inizia la curva nuova. È ottenuto sottraendo al numero di punti di controllo i *surplusPoints*, ovvero i punti che compongono la curva in costruzione. Questo valore è ottenuto secondo i seguenti calcoli:

```
int completedCurves = (numCP - 1) / degree;
int surplusPoints = (numCP - 1) % (degree * (completedCurves > 0 ?
    completedCurves : 1)) + 1;
```

- **tempDegree** definito come *surplusPoints - 1*

Per fare in modo di disegnare tutte le curve complete, si ciclano tutti i punti di controllo e si richiama la routine solo se:

- vi è almeno una curva completa (`completedCurves > 0`)
- il punto corrente non è il primo punto di controllo (`c!=0`)
- il punto corrente è un punto di giunzione (`c%degree == 0`)

Per meglio comprendere il funzionamento di queste espressioni, riporto l'esempio di 9 punti di controllo con grado massimo per curva pari a 3. I valori che si ottengono sono i seguenti:

- *completedCurves* = (9 - 1) / 3 = **2** –> 2 curve complete.
- *surplusPoints* = (9 - 1) % (3 × 2) + 1 = 8 % 6 + 1 = **3** –> 3 punti della curva in costruzione. Ricordo che il settimo punto (punto 6 in figura 1.3) rappresenta un punto di giunzione ed è quindi parte sia della curva completa 2, sia della curva in costruzione;

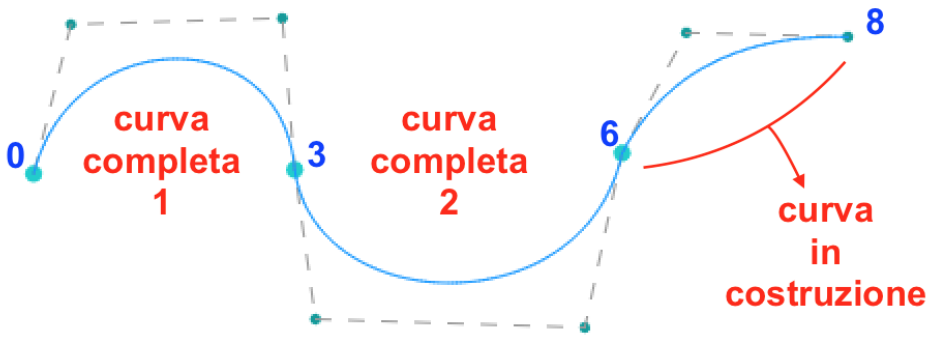


Figura 1.3

1.4.1 Modifica della continuità nei punti di giunzione

Per questo scenario si è scelto di dare la possibilità all'utente di modificare la continuità dei punti di raccordo. È infatti possibile cliccare con il tasto centrale (o click della rotellina) su un punto di raccordo per visualizzare un piccolo menù contenente "C_0, C_1, G_1".

Per creare il menù è stata utilizzata la primitiva *glutCreateMenu()* e sono state aggiunte le 3 voci con *glutAddMenuEntry* e il tasto centrale è stato attaccato tramite la *glutAttachMenu()*. `changeContinuity(int sel)` è la funzione che a seconda della selezione, applica una posizione diversa al punto successivo al punto di giunzione.

Per poter impostare una diversa continuità in ogni punto di giunzione è necessario tenere in memoria un vettore *junctionContinuity* che viene periodicamente aggiornato nelle funzioni di aggiunta e rimozione dei punti di controllo e in *changeContinuity*.

Nel caso di aggiunta di un punto di controllo successivo ad un punto di raccordo con continuità C_1 o G_1, è necessario proiettare il punto alla stessa distanza che intercorre tra il punto di raccordo e il punto precedente (nota: per G_1 si è scelto di proiettare allo stesso modo di C_1, per semplicità). Per fare in modo che il punto sia aggiunto *in continuità*, basta applicare una interpolazione con parametro t = -1 tra il punto di giunzione e il punto precedente e si ottengono le coordinate del nuovo punto.

La funzione peculiare di questo scenario è la `movePoint()` che viene chiamata ogni volta che l'utente trascina un punto con il tasto destro del mouse. La funzione applica **sempre** la variazione δx e δy sommandole alle coordinate del **punto selezionato** (ottenuto con la `highlightPoint()`). Anche questa funzione fa uso delle *completedCurves* e se ve ne è almeno una, distingue i casi in cui il punto che l'utente sta spostando sia un punto che precede o che segue un punto di raccordo. Vi è inoltre la distinzione sulla continuità del punto di giunzione:

- se **il punto selezionato è un punto di giunzione** allora applico le variazioni δx e δy anche ai due punti, precedente e successivo.
- se **il punto selezionato PRECEDE un punto di giunzione**, allora si controlla la continuità nel punto di giunzione:
  - C_1: si applica la variazione **invertita di segno** dello spostamento al punto **successivo al punto di giunzione**
  - G_1: si utilizza la funzione `getRotationPoints(centerPoint)` per ottenere le nuove coordinate del punto **successivo al punto di giunzione**
- se **il punto selezionato SEGUE un punto di giunzione**, allora si controlla la continuità nel punto di giunzione e analogamente si applicano nuove coordinate al **punto precedente al punto di giunzione**.

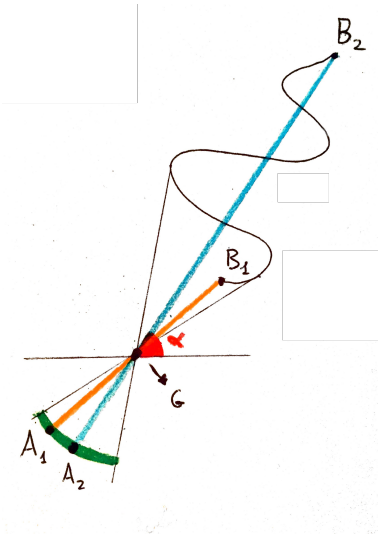


Figura 1.4

La funzione `getRotationPoints(centerPoint)` prende in ingresso l'indice del punto di giunzione attorno a cui dovrà far ruotare il punto opposto al *selectedPoint*. Prendiamo come esempio l'immagine 1.4 per illustrarne il funzionamento.

G è il punto di giunzione con continuità G_1 e B_1 è il punto selezionato. Se lo si trascina nello spazio, il punto A_1 dovrà seguirlo mantenendo sempre la distanza dal centro, muovendosi quindi sulla traiettoria circolare verde. Alla prima chiamata, la funzione calcola il raggio di questa traiettoria e lo salva globalmente per non doverlo ricalcolare ad ogni spostamento del punto B_1. Ad ogni chiamata, la funzione calcola l'angolo α attraverso la distanza tra G e B_2 e lo applica in seno e coseno alle nuove coordinate del punto A_2.

Il risultato finale è il seguente. Si possono notare i 3 diversi esempi di continuità in 3 diversi punti di giunzione.

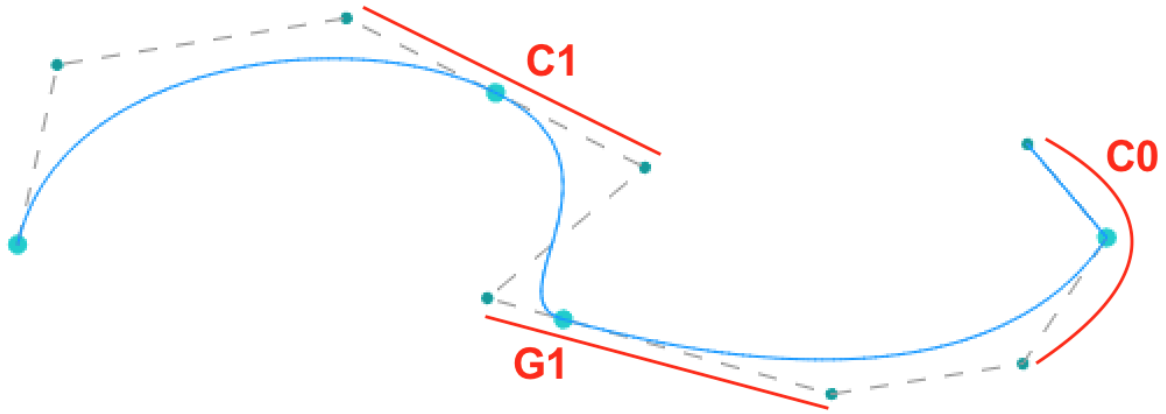


Figura 1.5