Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Triennale in Informatica

CURVE DI BEZIER E CALCOLO NUMERICO

Tesi di Laurea in Calcolo Numerico

Relatore:
Chiar.mo Prof.
GIULIO CASCIOLA

Presentata da:
GIUSEPPE CAIRO

Sessione II
Anno Accademico 2011-2012

Alla mia famiglia

Introduzione

Pierre Bézier, l'inventore delle curve che portano il suo nome, era un ingegnere francese nato nel 1910 e morto nel 1999. Lavorava come progettista alla Renault, quando negli anni sessanta mise a punto, per il design delle carrozzerie delle auto, uno dei primi sistemi CAD, UNISURF.
La teoria su cui si basa UNISURF nasce da un concetto rivoluzionario per la rappresentazione di curve e superfici: Bézier pensò che le curve dovessero essere adattabili alla forma desiderata, dovessero cioè dare al designer la possibilità di modificarle fino a completa soddisfazione. Ebbe, per ottenere questo, l'idea di combinare la moderna teoria dell'approssimazione e la geometria, in modo da fornire ai progettisti gli strumenti computerizzati analoghi dei loro strumenti convenzionali di lavoro.
Incorporò quindi nel suo progetto, la base di Bernstein e dei punti di controllo, che, con il loro posizionamento, giocano un ruolo centrale nel meccanismo di design interattivo, perché consentono di effettuare, passo dopo passo, le correzioni necessarie fino al raggiungimento del massimo grado di corrispondenza tra forma rappresentata e forma voluta.
I polinomi nella base di Bernstein, descritti da S. Bernstein nel 1912, sono una base di rappresentazione dei polinomi alternativa alla base delle potenze, meglio condizionata numericamente rispetto a quest'ultima e che gode di alcune interessanti proprietà.
Il concetto dei punti di controllo, già implicito nella definizione dei polinomi di Bernstein, fu esplicitato da Bézier.

Per la verità, lui non fu il solo a ricorrere alla base polinomiale di Bernstein e alla esplicitazione di punti di controllo per la definizione di curve e superfici. Anche Paul de Casteljau della casa automobilistica Citroën descrisse in quegli anni un sistema analogo, basato sull'uso di punti pilota chiamati *poles*. E, benché non facesse alcun esplicito riferimento ai polinomi di Bernstein, le caratteristiche delle sue *courbes et surfaces à poles* sono senza dubbio da collegare ad essi, come, per esempio, le proprietà di non negatività e di partizione dell'unità delle funzioni base associate ai *poles*.
Le idee di de Casteljau non ebbero però alcuna diffusione. Rimasero a lungo sconosciute, registrate solo nei documenti interni della Citroën che, dopo un'iniziale sottovalutazione, le adottò.
Quelle di Bézier, pubblicate estensivamente, ebbero invece una grande risonanza e le curve sono oggi universalmente riconosciute con il suo nome, anche se la loro formulazione sembra più vicina nello spirito alla formulazione di de Casteljau che a quella originale di Bézier.

UNISURF fu un modello che, per la sua flessibilità e semplicità rispetto alle tecniche del tempo, risultò il primo ampiamente accettato nella computer grafica e CAD.

Le curve di Bézier sono oggi alla base dei sistemi CAD 2D (Computer Aided Design), CAD 3D, sistemi di disegno, software e linguaggi grafici. Le curve di Bézier sono inoltre uno standard per la rappresentazione di font di caratteri, di modelli CAD in genere e sono utilizzate nello standard SVG (Scalable Vector Graphics) per la grafica vettoriale in alternativa alla grafica bitmap.

Questo lavoro di tesi si occupa di *Mini System*, un software didattico, sviluppato dal Prof. Giulio Casciola e utilizzato nel corso di *Metodi numerici per la grafica*.

Mini System è un programma per il disegno e la manipolazione di curve di Bézier, scritto in C, che utilizza le libreria Xlib di Linux. Il lavoro di tesi consiste nell'implementazione di una nuova interfaccia grafica e nella sostituzione delle librerie Xlib con le librerie SDL che, essendo multipiatta-

forma, potranno consentire in futuro di portare Mini System su altri sistemi operativi.

Nel primo capitolo verranno analizzate le librerie grafiche, in particolare Xlib e SDL.

Nel secondo capitolo verrà esposta la teoria matematica alla base delle curve di Bézier.

Infine, nel terzo capitolo verrà analizzata l'architettura di Mini System e le modifiche apportate al programma, quali la nuova interfaccia grafica e le nuove funzioni di disegno.

Indice

Elenco delle figure

Capitolo 1

Librerie software

L'*interfaccia grafica utente* (in breve GUI: graphical user interface) è la parte di un programma che si occupa dell'interazione tra l'utente e il programma stesso. La GUI utilizza un ambiente grafico, permettendo all'utente di utilizzare l'applicazione software evitandogli di dover imparare una serie di comandi.

Una *libreria software* è un insieme di funzioni pronte per l'uso, tali funzioni possono essere utilizzate dal programmatore senza doverle riscrivere ogni volta che ne ha bisogno.

Una *libreria grafica* è un insieme di funzioni che consentono di creare facilmente delle interfacce grafiche utente.

1.1 Xlib

X Window System è un sistema a finestre progettato al MIT, ed è il gestore grafico standard dei sistemi Unix. Il server distribuisce l'input dell'utente e accetta delle richieste di output da vari programmi client che si trovano sulla stessa macchina su cui X Window è in esecuzione oppure su altre macchine all'interno della rete. *Xlib* è una libreria client del protocollo X Window System scritta in C. Essa contiene delle funzioni utilizzate dai programmi client per interfacciarsi con un X server, tali funzioni permettono

ai programmatori di scrivere i loro programmi senza conoscere i dettagli del protocollo. Le applicazioni che utilizzano direttamente Xlib sono poche, la maggior parte delle applicazioni utilizzano invece altre librerie che usano le funzioni di Xlib per fornire dei toolkit di widget.

Mini System utilizza le librerie Xlib sia per la GUI che per il disegno di curve di Bézier da ciò, in base a quanto detto sopra, si deduce che questa applicazione software non è portabile su altri sistemi operativi.

1.2 SDL

Simple DirectMedia Layer è una libreria multimediale multipiattaforma, scritta in C, che fornisce accesso a basso livello a tastiera, mouse, audio digitale, joystick, hardware 3D tramite OpenGL e video framebuffer 2D. Essa è utilizzata da riproduttori video MPEG, emulatori e molti videogiochi. Essendo una libreria multipiattaforma, SDL permette al programmatore di creare un'applicazione multimediale una sola volta e farla funzionare su vari sistemi operativi.

Figura 1.1: Il logo delle librerie SDL

La libreria SDL ha collegamenti con quasi ogni linguaggio di programmazione, da C++ a Perl, Python, Pascal ecc..

SDL funziona come un *wrapper* che fornisce il supporto alle operazioni 2D su pixel, suoni, gestione degli eventi, temporizzatori ed altro.

La libreria è suddivisa in otto sottosistemi: Video, Audio, CD-ROM, Joystick, gestione eventi, I/O file, processi e Timer. Oltre a questi sottosistemi, vi sono delle librerie ufficiali che aggiungono delle altre funzionalità:

- SDL_image
- SDL_mixer
- SDL_net
- SDL_ttf
- SDL_rtf

In particolare *SDL_image* e *SDL_ttf* sono state utilizzate all'interno di Mini System, per gestire formati di immagine diversi da *BMP* e per fare il *rendering* dei font TrueType.

1.2.1 Architettura

Un *wrapper* è una libreria software che consiste in un *layer* di codice che traduce l'interfaccia di una libreria esistente in una interfaccia compatibile. SDL è un *wrapper* attraverso funzionalità specifiche per i diversi sistemi operativi. La libreria ha lo scopo di fornire un *framework* comune per accedere a queste funzionalità. Il codice di SDL è suddiviso in moduli separati per ogni sistema operativo. Al momento della compilazione di SDL, vengono selezionati i moduli corretti per il giusto sistema operativo.

Su Microsoft Windows, SDL fornisce un wrap per le DirectX, che ha loro volta sono un wrap per il driver video.

Su X11, SDL usa le Xlib per comunicare con il sistema X11 per gli eventi grafici.

Su Mac OS X, SDL usa Quartz.

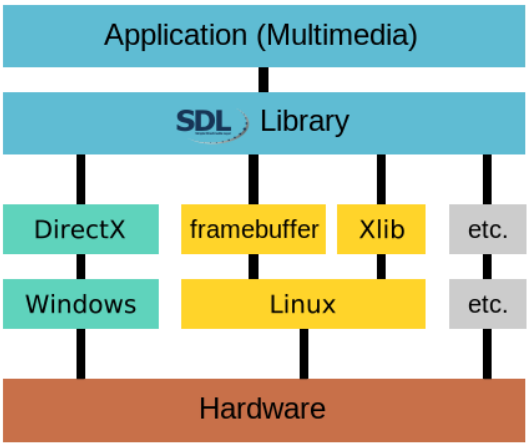

Figura 1.2: Livelli di astrazione per architetture SDL

1.3 Utilizzo di SDL

Ognuno degli otto sottosistemi di cui è composta SDL deve essere inizializzato prima di essere usato. Per fare ciò si utilizza la funzione `SDL_Init` il cui argomento specifica quale sottosistema viene inizializzato.

Esempio 1.3.1.

```
if (SDL_Init(SDL_INIT_VIDEO)<0)
{
    fprintf(stderr,"Error initializing SDL Video:
                            %s\n",SDL_GetError());
}
```

Prima di terminare è necessario chiudere chiamando `SDL_quit`.
Dopo aver inizializzato SDL, bisogna creare una window/viewport, attraverso una chiamata a `SDL_SetVideoMode()`. `SDL_SetVideoMode` ritorna un puntatore a `SDL_SURFACE`.

Esempio 1.3.2. *Viene creata un'area di disegno* 800×600 *a 32 bit color*

```
SDL_Surface *p_screen;
....
```

```
p_screen = SDL_SetVideoMode(800,600,32,SDL_SWSURFACE);
if (p_screen==NULL)
{
    fprintf(stderr,"Error setting video mode:
                              %s\n",SDL_GetError());
}
```

L'ultimo argomento della funzione `SDL_SetVideoMode()` può essere uno dei seguenti flag:

- `SDL_HWSURFACE/SDL_SWSURFACE`: usa video/system memory;
- `SDL_DOUBLEBUF`: usa double buffering;
- `SDL_FULLSCREEN`: usa full screen mode.

1.3.1 SDL_Surface

La struttura `SDL_Surface` descrive un'area grafica, ad esempio una finestra dello schermo.
Una Surface ha un'ampiezza e un'altezza (w e h), un pixel format ecc.. Le funzione SDL che operano su una `SDL_Surface` sono molte, ad esempio:

- `SDL_FillRect`: riempie un rettangolo di una Surface con un colore;
- `SDL_BlitSurface`: fa il *blitting* di una Surface, cioè copia una Surface su un'altra.

Dopo aver finito di usarla, una `SDL_Surface` deve essere liberata con una chiamata a `SDL_FreeSurface`.

1.3.2 Setting e getting di pixel

SDL non fornisce delle funzioni per disegnare un pixel con un colore o leggere il colore di un pixel. È però possibile accedere al *frame buffer* o *video memory* e modificarne o leggerne il contenuto.

Esempio 1.3.3. *Si costruisce una funzione di disegno di un pixel e una funzione per leggere un colore dalla memoria video.*

```
void GC_PutPixel(SDL_Surface *surface,int x,int u,Uint32 pixel)
{
    int bpp=surface->format->BytesPerPixel;
    Uint8 *p=(Uint8 *)surface->pixels+y*surface->pitch+x*bpp;
    ...
    *p=pixel;
    ...
}

Uint32 GC_GetPixel(SDL_Surface *surface,int x,int y)
{
    int bpp=surface->format->BytesPerPixel;
    Uint8 *p=(Uint32 *)surface->pixels+y*surface->pitch+x*bpp;
    ...
    return *p;
}
```

Per poter leggere e scrivere un pixel è necessario bloccare una surface perché si possa accedere direttamente alle informazioni sul pixel mediante la chiamata a `SDL_LockSurface`; successivamente la surface va sbloccata tramite una chiamata a `SDL_UnlockSurface`.

Esempio 1.3.4. *Esempio di utilizzo di GC_PutPixel e GC_GetPixel.*

```
SDL_Surface *p_screen;
Uint32 col_pixel;
    SDL_LockSurface(p_screen);
    col_pixel=GC_GetPixel(p_screen,x,y);
    /* modifica il colore del pixel e lo scrive */
    GC_PutPixel(p_screen,x,y,col_pixel);
    SDL_UnlockSurface(p_screen);
```

1.3.3 Gestione dell'input

Gli eventi generati dalla tastiera e dal mouse sono memorizzati in una struttura SDL_Event.

```
typedef union{
  Uint8 type;
  SDL_ActiveEvent active;
  SDL_KeyboardEvent key;
  SDL_MouseMotionEvent motion;
  SDL_MouseButtonEvent button;
  SDL_JoyAxisEvent jaxis;
  SDL_JoyBallEvent jball;
  SDL_JoyHatEvent jhat;
  SDL_JoyButtonEvent jbutton;
  SDL_ResizeEvent resize;
  SDL_ExposeEvent expose;
  SDL_QuitEvent quit;
  SDL_UserEvent user;
  SDL_SysWMEvent syswm;
} SDL_Event;
```

Per controllare la presenza di eventi nella coda FIFO si utilizza la funzione SDL_PollEvent(SDL_Event *ev che restituisce il valore 1 se ci sono eventi in coda, o altrimenti.

Tastiera

Gli eventi si distinguono in SDL_KEYDOWN (tasto premuto) e SDL_KEYUP (tasto rilasciato).

```
typedef struct{
  Uint8 type;
  Uint8 state;
```

```
  SDL_keysym keysym;
} SDL_KeyboardEvent;
```

In cui `event.key` è una struttura `SDL_KeyboardEvent`; mentre il campo `event.key.keysym.sym` è il codice virtuale in SDL per i tasti.

Mouse

Ci sono due tipi di eventi per il mouse: motion e button (press/release).

```
typedef struct {
  Uint8 type;
  Uint8 state;
  Uint16 x, y;
  Sint16 xrel, yrel;
} SDL_MouseMotionEvent;

typedef struct {
  Uint8 type;
  Uint8 button;
  Uint8 state;
  Uint16 x,y;
} SDL_MouseButtonEvent;
```

Lo stato corrente del mouse si può controllare con `SDL_BUTTON`

1.3.4 Gestione del testo

La libreria SDL_ttf fornisce delle funzioni che permettono di gestire il disegno di font *truetype* su una finestra grafica. Il testo viene reso su una `SDL_Surface` che sarà copiata sullo schermo. Quello che segue è l'elenco di alcune funzioni messe a disposizione da SDL_ttf:

- `TTF_Init`: inizializza SDL_ttf;

- `TTF_OpenFont(char *name,int pts)` carica un font e ritorna un puntatore a `TTF_Font`;
- `TTF_RenderText_xx(TTF_Font*,char*,SDL_Color)` genera una surface con un testo `char*` ed un colore, dove:

  xx==Solid stampa il font senza *alpha-blending*

  xx==Blended stampa il font con *alpha-blending*
- `TTF_FreeFont(TTF_Font *)` rilascia la font memory;
- `TTF_Quit()` chiude SDL_ttf;

1.3.5 Gestione immagini

Per gestire il caricamento e il disegno di immagini nei formati BMP, GIF, JPG, PPM, ecc. si utilizza la libreria SDL_image.
Per caricare un file per essere utilizzato come un'immagine in una nuova surface si usa la funzione

```
SDL_Surface *IMG_Load(const char *file)
```

questa chiama `IMG_LoadTyped_RW` con l'estensione utilizzata per il file che permette di includere tutti i tipi supportati.

1.3.6 Gestione timer

Per la gestione del tempo, SDL fornisce le seguenti funzioni:

- `SDL_GetTicks()` rileva il numero di millisecondi trascorsi dalla inizializzazione di SDL;
- `SDL_Delay()` attende un numero di millisecondi specificato, prima di ritornare;
- `SDL_AddTimer()` aggiunge un timer che chiamerà una funzione di callback dopo uno specificato numero di secondi.

- `SDL_RemoveTimer()` rimuove un timer precedentemente aggiunto con `SDL_AddTimer()`;
- `SDL_SetTimer()` setta una funzione di callback per essere eseguita dopo che siano trascorsi un certo numero di millisecondi.

1.4 La scelta di SDL

Si è deciso di scegliere SDL in quanto, essendo una libreria multipiattaforma, consente di sviluppare delle applicazioni portabili, cioè in grado di funzionare su sistemi operativi differenti, in particolare Linux, Windows e Mac OS X. Un altro motivo che ha fatto ricadere la scelta su SDL è la presenza della libreria *GCGraLib* sviluppata dal Prof. Giulio Casciola. Tale libreria fornisce delle funzioni grafiche di base che si appoggiano su SDL.

1.4.1 GCGraLib

GCGraLib è una libreria, scritta in C, che permette di fare grafica 2D in modo semplice, sfruttando le funzionalità messe a disposizione dalla libreria SDL. Quello che segue è l'elenco di tutte le funzioni fornite dalla libreria:

- `void GC_PutPixel(SDL_Surface *surface, int x, int y, Uint32 pixel);`
- `void GC_GetPixel(SDL_Surface *surface, int x, int y);`
- `void GC_HorizLine(SDL_Surface *s, int x0, int x1, int y, Uint32 color);`
- `void GC_VerticLine(SDL_Surface *s, int x, int y0, int y1, Uint32 color);`
- `void GC_DrawLine(SDL_Surface *s, int x0, int y0, int x1, int y1, Uint32 color);`

- `void GC_DrawRect(SDL_Surface *s, int ax, int ay, int width, int height, Uint32 color);`
- `void GC_DrawCircle(SDL_Surface *s, int xin, int yin, int rad, Uint32 color);`
- `void GC_FillCircle(SDL_Surface *s, int xin, int yin, int rad, Uint32 color);`
- `void GC_DrawText(SDL_Surface *s, TTF_Font *fonttodraw, char fgR, char fgG, char fgB, char fgA, char bgR, char bgG, char bgB, char bgA, char text[], int x, int y, enum textquality quality);`
- `void GC_GetPixelImage(SDL_Surface *surface, int x, int y);`
- `void GC_LoadImage(char *file, int *exitstate);`
- `void GC_DrawImage(SDL_Surface *srcimg, int sx, int sy, int sw, int sh, SDL_Surface *dstimg, int dx, int dy);`

Capitolo 2

Polinomi di Bernstein e curve di Bézier

Ricorrono quest'anno i cento anni della base polinomiale di Bernstein, un matematico ucraino nato nel 1880, che completò i suoi studi alla Sorbona di Parigi, dove sviluppò un'interesse anche per l'ingegneria ed entrò a far parte della École d'Électrotechnique Supérieure.

È difficile prevedere l'evoluzione e gli sviluppi applicativi di una nuova idea matematica. Concetti che a prima vista possono sembrare rivoluzionari, perdono gradualmente rilevanza scientifica, mentre metodi di scarso interesse pratico, introdotti per dimostrare delle dimostrazioni teoriche, possono rivelarsi degli strumenti utili ampiamente accettati in diversi campi.
È questo il caso dei polinomi di Bernstein, introdotti come mezzo per dimostrare la capacità dei polinomi di approssimare qualsiasi funzione continua, con un arbitrario grado di accuratezza, su un dato intervallo. Dopo un'iniziale scarsa rilevanza applicativa, hanno avuto, con l'avvento dei computer, una loro larga applicazione nel design interattivo di funzioni polinomiali, cioè curve e funzioni parametriche, e hanno portato a molti utili algoritmi che ora vengono sempre più adottati in altri campi di applicazione.

2.1 Funzioni polinomiali

Definizione 2.1.1. *Un polinomio è una funzione* $p : \mathbf{R} \to \mathbf{R}$ *tale che*

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

I polinomi sono molto importanti nel calcolo numerico, grazie alle loro buone proprietà; i polinomi, infatti, sono funzioni regolari facilmente memorizzabili e valutabili e sono in grado di approssimare uniformemente qualsiasi funzione continua su un intervallo $[a, b]$, con un arbitrario grado di precisione.

2.1.1 Valutazione di un polinomio: errore inerente

Nella valutazione di un polimomio, l'errore inerente fornisce la misura della variazione del valore del polinomio, conseguente a delle perturbazioni sui valori dei coefficienti.

Esempio 2.1.1.

$$p(x) = a_0 + a_1 x = 100 - x$$

Si può far vedere che perturbando uno dei coefficienti dell'1% il polinomio perturbato sarà

$$\tilde{p}(x) = 100 - (1 - \frac{1}{100})x = 100 - \frac{99}{100}x$$

valutandoli in $x = 101$ si ottiene

$$p(101) = -1 \qquad e \qquad \tilde{p}(101) = 0.01$$

Cioè perturbando un coefficiente dell'1% si ha un errore sul risultato finale del 101%. Questo è dovuto al fatto che il problema della valutazione di un polinomio espresso nella base delle potenze è in generale un problema malcondizionato. Analizzando l'errore inerente mediante la stima

$$\epsilon_{in} \simeq \sum_{i=1}^{n} c_i \varepsilon_i \quad con \quad c_i = \frac{x_i}{f(x_i)} \frac{\partial f(x_i)}{\partial x_i}$$

nel caso di un generico polinomio $p(x)$ nella base delle potenze, si ottiene che l'errore inerente è composto da due componenti:

$$\epsilon_{in1} = \sum_{i=0}^{n} \frac{a_i}{p(x)} \frac{\partial p(x)}{\partial a_i} \varepsilon_i = \sum_{i=0}^{n} \frac{a_i x^i}{p(x)} \varepsilon_i \quad con\ \varepsilon_i = \frac{\tilde{a}_i - a_i}{a_i}$$

e

$$\epsilon_{in2} = \frac{x}{p(x)} \frac{\partial p(x)}{\partial x} \varepsilon_x = \frac{x p'(x)}{p(x)} \varepsilon_x \varepsilon_x = \frac{\tilde{x} - x}{x}$$

da cui si desume che:

- ϵ_{in1} dipende da $p(x)$ e dai valori $a_i x^i$, che possono essere controllati cambiando la rappresentazione del polinomio;
- ϵ_{in2} dipende dal polinomio $p(x)$ e dalla sua derivata. Non può essere eliminato cambiando la sua rappresentazione. In particolare si ha un grande errore relativo per $p(x) \to 0$ e per $p'(x) \to \infty$.

Per ovviare a ϵ_{in1} si può cambiare la base di rappresentazione del polinomio. Le basi di rappresentazione polinomiale sono infinite e, dato un problema, è possibile individuare la base più adatta per trattarlo. La base polinomiale su cui si basano le curve di Bézier è la base di Bernstein.

2.2 Polinomi nella base di Bernstein

Un polinomio di grado n espresso nella base di Bernstein è dato da

$$p(x) = \sum_{i=0}^{n} b_i B_{i,n}(x) \quad con\ x\ \in\ [a, b]$$

dove $B_{i,n}(x)$ sono le funzioni base di Bernstein definite da

$$B_{i,n} = \binom{n}{i} \frac{(b-x)^{n-i}(x-a)^i}{(b-a)^n} \quad i = 0, \ldots, n$$

e $b_0, b_1, \ldots, b_n\ \in \mathbf{R}$ sono i coefficienti nella base di Bernstein.
Rappresentando il polinomio dell'esempio 2.1.1 nella base di Bernstein ed eseguendo l'analisi dell'errore inerente

$$\epsilon_{in} = c_1\varepsilon_1 + c_2\varepsilon_2 + c_3\varepsilon_3$$

si ha che perturbando uno dei coefficienti del'1% il risultato finale ha una variazione dell'1%. Se invece $\epsilon_3 \neq 0$ l'errore inerente potrebbe essere grande indipendentemente dalla base di rappresentazione. Per ovviare a questo problema si può effettuare un cambio di variabile e una traslazione e scala dell'intervallo di definizione.

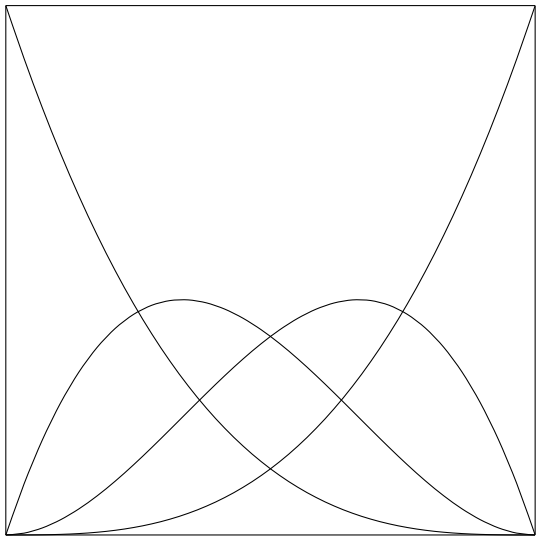

Figura 2.1: I polinomi di Bernstein di grado 3

2.2.1 Proprietà dei polinomi di Bernstein

Quello che segue è un elenco di alcune proprietà dei polinomi di Bernstein.

- Non negatività: $B_{i,n}(t) \geq 0 \quad i = 0, \ldots n \quad \forall t \in [0,1]$;
- Partizione dell'unità:

$$\sum_{i=0}^{n} B_{i,n}(t) = 1 \quad \forall t \in [0,1]$$

- Dalle proprietà precedenti segue che $p(t)$ nella base di Bernstein è una combinazione convessa dei b_i, da cui segue

$$min_i\{b_i\} \leq p(y) \leq max_i\{b_i\} \quad \forall t \in [0,1].$$

- Simmetria: vale la relazione $B_{i,n}(t) = B_{i,n}(1-t)$.

- Condizioni agli estremi: vale

$$B_{i,n}(0) = \delta_{i,0} \quad e \quad B_{i,n}(1) = \delta_{i,n} \quad dove \quad \delta_{i,j} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases}$$

- Unicità del massimo: $B_{i,n}(t)$ ha un solo massimo in $[0,1]$ e questo si ha in corrispondenza di $t = \frac{i}{n}$

- Formula per il prodotto: siano $B_{i_h,n_h} h = 1, \ldots, k$ k polinomi di Bernstein indicizzati da h, poniamo:

$$n_1 + \ldots + n_k = \mathbf{n}, \quad i_1 + \ldots + i_k = \mathbf{i};$$

$$\prod_{h=1}^{k} B_{i_h,n_h}(t) = \frac{\prod_{h=1}^{k} \binom{n_h}{i_h}}{\binom{\mathbf{n}}{\mathbf{i}}} B_{\mathbf{i},\mathbf{n}}(t)$$

  Il prodotto di k polinomi di Bernstein è ancora un polinomio di Bernstein di grado la somma dei gradi dei polinomi fattori.

- Variazione di segno dei coefficienti: un polinomio espresso nella base di Bernstein ha un numero di variazioni di segno minore o uguale al numero di variazione di segno del vettore dei suoi coefficienti (valori o coefficienti nulli non vengono considerati).

- Traslazione e scala: i polinomi hanno la proprietà di essere invarianti per traslazione e scala dell'intervallo di definizione, i polinomi di Bernstein, in particolare, hanno la proprietà che, una volta effettuato il mapping $t = \frac{x-a}{b-a}$, i coefficienti rimangono gli stessi. Infatti si può dimostrare che

$$B_{i,n}(x) = \binom{n}{i}(1-t)^{n-i}t^i = B_{i,n}(t).$$

  Quindi un polinomio nella base di Bernstein definito in $[0,1]$ è dato da

$$p(t) = \sum_{i=0}^{n} b_i B_{i,n}(t) \quad t \in [0,1].$$

Questa proprietà è importante perché permette di ovviare al problema della stabilità della valutazione di un polinomio nella base di Bernstein. Quindi se è necessario valutare $p(\overline{x})$, si determina $\overline{t} = \frac{\overline{x}-a}{b-a}$, si calcola $p(\overline{t})$ in $[0,1]$, e sarà $p(\overline{x}) = p(\overline{t})$.

2.2.2 Formula ricorrente

I polinomi di Bernstein possono essere definiti ricorsivamente, tramite la seguente formula ricorrente

$$B_{i,n}(t) = tB_{i-1,n-1}(t) + (1-t)B_{i,n-1}(t)$$

con $B_{0,0}(t) = 1$ e $B_{i,n}(t) = 0 \quad \forall i \notin [0,n]$.

2.2.3 Valutazione mediante algoritmo di de Casteljau

Per valutare un polinomio nella base di Bernstein è possibile utilizzare la formula ricorrente e poi effettuare la combinazione convessa

$$p(\bar{x}) = \sum_{i=0}^{n} b_i B_{i,n}(\bar{x}).$$

In alternativa si può usare l'algoritmo di de Casteljau sui coefficienti del polinomio, che si basa sull'applicazione ripetuta della formula ricorrente.

$$p(t) = \sum_{i=0}^{n} b_i B_{i,n}(t) \overset{def}{=} \sum_{i=0}^{n} b_i t B_{i-1,n-1}(t) + \sum_{i=0}^{n} b_i (1-t) B_{i,n-1}(t) =$$

poiché $B_{-1,n-1} = B_{n,n-1} = 0$

$$= \sum_{i=0}^{n-1} b_{i+1} t B_{i,n-1}(t) + \sum_{i=0}^{n-1} b_i (1-t) B_{i,n-1}(t) =$$

raccogliendo

$$= \sum_{i=0}^{n-1} [b_{i+1}t + b_i(1-t)] B_{i,n-1}(t) =$$

$$= \sum_{i=0}^{n-1} b_i^{[1]} B_{i,n-1}(t) =$$

riapplicando la formula ricorrente più volte, si ottiene:

$$\sum_{i=0}^{0} b_0^{[n]} B_{0,0}(t) = b_0^{[n]}.$$

Si ha quindi il seguente algoritmo:

$$b_i^{[k]} = t b_{i+1}^{[k-1]} + (1-t) b_i^{[k-1]}$$

con $k = 1, \ldots, n$, $i = 0, \ldots, n-k$ e $p(t) := b_0^{[n]}$; in modo esplicito si ha il seguente schema triangolare:

$$\begin{array}{lllll}
b_0^{[0]} & b_1^{[0]} & b_2^{[0]} & \ldots & b_{n-1}^{[0]} \quad b_n^{[0]} \\
b_0^{[1]} & b_1^{[1]} & b_2^{[1]} & \ldots & b_{n-1}^{[1]} \\
\ldots & & & \ldots & \\
b_0^{[n-2]} & b_1^{[n-2]} & b_2^{[n-2]} & & \\
b_0^{[n-1]} & b_1^{[n-1]} & & & \\
b_0^{[n]} & & & &
\end{array}$$

2.3 Approssimazione di forma

Dati $n+1$ punti equidistanti in $[a, b]$, l'approssimazione di Bernstein di grado n, $B_n[f(x)]$ di $f(x)$ è definita come

$$B_n[f(x)] = \sum_{i=0}^{n} f(\xi_i) B_{i,n}(x)$$

con $\xi_i = a + \frac{i}{n}(b-a)$ e $B_{i,n}(x)$ i polinomi di Bernstein di grado n in $[a, b]$. Utilizzando polinomi di grado sufficientemente elevato, tale approssimazione può soddisfare un qualunque errore δ, esiste cioè un valore n_δ tale che

$$|B_n[f(x)] - f(x)| < \delta$$

$\forall x \in [a, b]$ per $n > n_\delta$. In questo caso si dice che l'approssimazione di Bernstein converge uniformemente ad $f(x)$:

$$\lim_{n \to \infty} B_n[f(x)] = f(x) \quad x \in [a, b].$$

2.3.1 Approssimazione VD

Definizione 2.3.1 (Variazione di segno di un vettore)**.** *Sia* $\mathbf{c} = (c_0, c_1, \ldots, c_n)$ *un vettore ad elementi reali; si definisce numero di variazioni di segno di* $\mathbf{c}$ *in senso forte, e lo si indica con*

$$V^-[\mathbf{c}]$$

il numero di variazioni di segno nella sequenza $c_0, c_1, \ldots, c_n$ *in cui si ignorano gli zeri.*

Definizione 2.3.2 (Variazione di segno di una funzione)**.** *Sia* f *una funzione reale in* $[a, b]$*; si dirà che:*

$$V^-[f(z)] = sup_{n+1}\{V^-[f(z_0), \ldots, f(z_n)] \; con \; z_0 < z_1 < 1 \ldots z_n\}.$$

Sia data una funzione $f(x) \in F$ almeno continua nell'intervallo $[a, b]$. Nel caso generale viene individuato uno spazio Φ a dimensione finita $n + 1$, di funzioni reali e una loro base di rappresentazione $\phi_0, \phi_1, \ldots, \phi_n$ tale che ogni $\phi \in \Phi$ è definito da una loro combinazione lineare:

$$\phi = \sum_{i=0}^{n} c_i \phi_i(x).$$

Il problema del'approssimazione variation diminishing consiste nel determinare una trasformazione lineare $s : F \to \Phi$,e chiameremo $\phi^* := s(f(x))$, tale che:

1. se $f(x) \in F$ è lineare, allora $s(f(x))$ è lineare e coincide con $f(x)$;

2. $$V^-[s(f(x))] \leq V^-[f(x)] \quad x \in [a, b]$$

   che vuol dire che la variazione di segno in senso forte della ϕ^* è minore o uguale alla variazione di segno in senso forte della $f(x)$.

Se queste due proprietà sono soddisfatte si ha come conseguenza che per $x \in [a, b]$ vale:

$$\begin{aligned} V^-[s(f(x)) - (a_0 + a_1 x)] & \\ &= V^-[s(f(x)) - s(a_0 + a_1 x)] \; per\ 1 \\ &= V^-[s(f(x) - (a_0 + a_1 x)] \; trasf.\ lineare \\ &= V^-[f(x) - (a_0 + a_1 x)] \; per\ 2 \end{aligned}$$

cioè la $s(f(x))$ non può essere più oscillante della $f(x)$ che approssima, poichè il numero di intersezioni della $s(f(x))$ con una retta è sempre minore o uguale al numero di intersezioni della $f(x)$ con quella retta.

Teorema 2.3.1. *Sia $n \geq 1$, allora $\forall x \in [a, b]$ si ha:*

$$\sum_{i=0}^{n} \xi_i B_{i,n}(x) = x$$

con $\xi_i = a + \frac{i}{n}(b - a)$.

Teorema 2.3.2. *Sia p un polinomio nella base di Bernstein e si consideri come approssimazione di una $f(x) : [a, b] \to \mathbf{R}$ almeno continua, la $B_n[f(x)]$ definita nella sezione precedente; allora $B_n[f(x)]$ è approssimante VD della f(x).*

2.4 Curve di Bézier

2.4.1 Curve in forma parametrica

Sia $\underline{C}(t) : [0, 1] \to \mathbf{R}^d$ una curva d-dimensionale in forma parametrica. Un curva piana di Bézier $\underline{C}(t)$ di grado n può essere definita a partire da un insieme di punti di controllo $P_i \in \mathbf{R}^2 \quad i = 0, \ldots, n$ ed è data da

$$\underline{C}(t) = \sum_{i=0}^{n} P_i B_{i,n}(t) \quad t \in [0, 1]$$

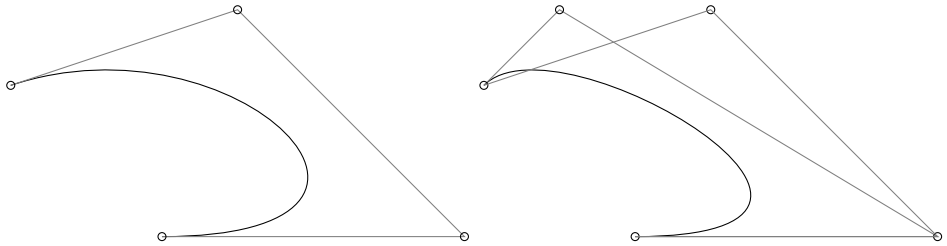

Figura 2.2: Esempio di curva di Bézier con $n = 3$ a sinistra; modifica di un punto di controllo a destra

2.4.2 Proprietà delle curve di Bézier

Le curve di Bézier sono caratterizzate dalle seguenti proprietà.

- Interpolazione degli estremi: la curva passa per i punti di controllo estremi:
$$\underline{C}(0) = P_0 \quad \underline{C}(1) = P_n$$
- Simmetria: le poligonali di controllo $P_0, P_1, \ldots, P_n$ e $P_n, P_{n-1}, \ldots, P_0$ descrivono la stessa curva.
- Invarianza affine: se si applica una trasformazione affine Φ ai punti di controllo, allora la curva resta trasformata interamente da Φ:
$$\sum_{i=0}^{n} \Phi[P_i] B_{i,n}(t) = \Phi[\sum_{i=0}^{n} P_i B_{i,n}(t)].$$
- Guscio convesso: un punto $\underline{C}(\bar{t})$ sulla curva per $\bar{t} \in [0, 1]$ giace nel guscio convesso definito dai punti di controllo.
- Variation diminishing: se una linea attraversa m volte la poligonale di controllo di una curva di Bézier allora la linea può intersecare la curva al più m volte.
- Precisione lineare: se i punti di controllo P_i per $i = 1, \ldots, n-1$ sono equispaziati sul segmento retto definito da P_0 e P_n, allora la curva di Bézier di grado n è l'interpolante lineare fra P_0 e P_n.

- Estrapolazione: per i valori t esterni all'intervallo $[0, 1]$, la curva non rimane in generale all'interno del guscio convesso definito dalla poligonale di controllo. In questo caso si avrà un comportamento imprevedibile e possibili instabilità negli algoritmi di calcolo.

2.4.3 Tecniche di interrogazione

Esistono due tecniche basilari di interrogazione di curve di Bézier: curvatura e torsione. La curvatura di una curva ne descrive la forma. Per le curve di Bézier la curvatura può essere calcolata senza calcolare le derivate. Per $t = 0$ la curvatura di una curva di Bézier è data da:

$$\kappa(0) = 2\frac{n-1}{n}\frac{area[P_0, P_1, P_2]}{||P_1 - P_0||^3}.$$

Per definizione una curva 3D ha curvatura non negativa, per curve di Bézier 2D, la curvatura con segno si ottiene definendo l'area in termini di determinante. La curvatura con segno permette di determinare i punti di flesso.
La torsione misura la torsione di una curva 3D. Per le curve di Bézier, la torsione ha una forma semplice agli estremi, per $t = 0$ si ha:

$$\tau(0) = \frac{3}{2}\frac{n-2}{n}\frac{volume[P_0, P_1, P_2, P_3]}{area[P_0, P_1, P_2]^2}.$$

2.5 Curve di Bézier a tratti

Quando si deve descrivere una forma complessa non si utilizza un'unica curva di Bézier, ma si descrivono dei semplici tratti della forma complessa mediante curve di Bézier di grado basso, cioè si ottiene la forma complessa attaccando più curve di Bézier. Tali curve dovranno raccordarsi tra loro in modo tale che la forma sia priva di irregolarità.

2.5.1 Continuità parametrica

La regolarità di una curva si misura tramite la continuità delle sue derivate. Una curva $\underline{C}(t)$ è detta C^k in u se e solo se

$$\lim_{t\to u^+} \underline{C}^{(i)}(t) = \lim_{t\to u^-} \underline{C}^{(i)}(t) \quad i = 0, \ldots, k$$

Date due curve $\underline{C_1}(t)$ e $\underline{C_2}(t)$ definite su intervalli parametrici consecutivi $[u_1, u_2]$ e $[u_2, u_3]$ si dirà che sono C^k nel punto comune u_2 se e solo se

$$\lim_{t\to u_2^+} \underline{C_2}^{(i)}(t) = \lim_{t\to u_2^-} \underline{C_1}^{(i)}(t) \quad i = 0, \ldots, k$$

2.5.2 Continuità geometrica

Una curva è continua geometricamente (G^k), se e solo se esiste una parametrizzazione regolare C^k della curva (una parametrizzazione di $\underline{C}(t)$ è regolare se $\underline{C}^{(1)}(t)$ non è mai il vettore identicamente nullo. Tutte le curve regolari C^k sono G^k, ma non viceversa.

Beta-condizioni

Le Beta-condizioni sono delle condizioni sulle componenti di una curva che valgono se e solo se la curva è geometricamente continua. La Beta-condizione per G^0 è identica alla condizione per la continuità parametrica C^0:

$$\lim_{t\to u^+} \underline{C}^{(0)}(t) = \lim_{t\to u^-} \underline{C}^{(0)}(t)$$

Le Beta-condizioni per G^1 dicono che una curva $\underline{C}(t)$ è G^1 se e solo se è G^0 e

$$\lim_{t\to u^+} \underline{C}^{(1)}(t) = \lim_{t\to u^-} \beta_1 \underline{C}^{(1)}(t)$$

per un certo $\beta_1 > 0$. Cioè se due segmenti di curva si congiungono in un modo che sembra C^1, allora le loro derivate prime non sono necessariamente uguali.
La curva $\underline{C}(t)$ è G^2 se e solo se in aggiunta alla Beta-condizione per G^0 e G^1 si ha:

$$\lim_{t\to u^+} \underline{C}^{(2)}(t) = \lim_{t\to u^-} (\beta_1^2 \underline{C}^{(2)}(t) + \beta_2 \underline{C}^{(1)}(t))$$

per un certo β_2.

2.5.3 Join di curve di Bézier

Due curve di Bézier $\underline{C_1}(t) = \sum_{i=0}^{n} P_i B_{i,n}(t)$ per $t \in [0,1]$ e $\underline{C_2}(t) = \sum_{i=0}^{n} Q_i B_{i,n}(u)$ per $u \in [0,1]$ di stesso grado n e punti di controllo P_i e Q_i si raccordano in $t = 1$ per la prima e $u = 0$ per la seconda con continuità parametrica:
C^0 o G^0 se $P_n = Q_0$
C^1 se è C^0 e $(P_n + P_{n-1}) = (Q_1 - Q_0)$
G^1 se è G^0 e $(P_n + P_{n-1}) = \beta_1(Q_1 - Q_0)$
C^2 se è C^1 e $(P_n - 2P_{n-1} + P_{n-2}) = (Q_2 - 2Q_1 + Q_0)$
G^2 se è G^1 e $(P_n - 2P_{n-1} + P_{n-2}) = \beta_1^2(Q_2 - 2Q_1 + Q_0) + \beta_2(Q_1 - Q_0)$.

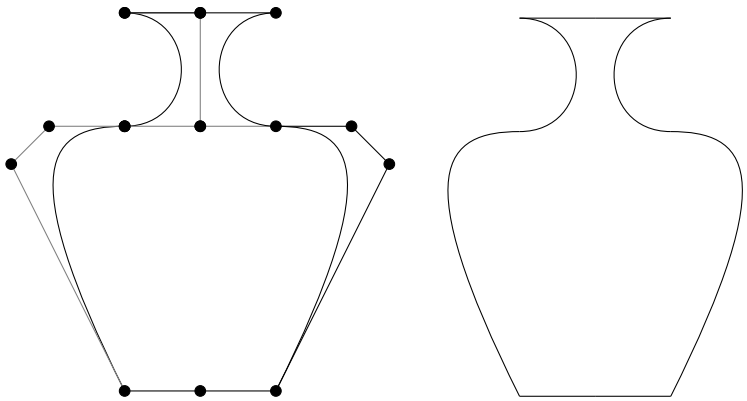

Figura 2.3: Esempio di figura ottenuta mediante join di curve di Bézier

2.6 Interpolazione

Date $n+1$ osservazioni in corrispondenza di $n+1$ punti (x_i, y_i), il problema dell'interpolazione consiste nel determinare una funzione ϕ tale che

$$\phi(x_i) = y_i \quad i = 0, \ldots, n.$$

Nel caso del'interpolazione polinomiale l'esistenza e l'unicità della soluzione è sempre garantita, infatti si dimostra il seguente

Teorema 2.6.1. *Siano dati $n+1$ punti (x_i, y_i), $i = 0, \ldots, n$ con x_i distinti, allora esiste ed è unico il polinomio $\mathbf{P}_n$ che verifica le condizioni*

$$p(x_i) = y_i \quad i = 0, \ldots, n.$$

Dimostrazione. Sia $p(x)$ nella forma

$$p(x) = a_0 + a_1 x + \ldots + a_n x^x$$

imponendo le $n+1$ condizioni di interpolazione

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 \ldots + a_n x_0^n = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 \ldots + a_n x_1^n = y_1 \\ \ldots \\ a_0 + a_1 x_n + a_2 x_n^2 \ldots + a_n x_n^n = y_n \end{cases}$$

È un sistema lineare di $n+1$ equazioni in $n+1$ incognite, la soluzione di tale sistema sarà il polinomio interpolante. In forma matriciale si ha:

$$V\mathbf{a} = \mathbf{y}$$

con

$$V = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \ldots & & & & \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix} \quad \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

dove V è la matrice di Vandermonde; il determinante di questa matrice sarà

$$detV = \prod_{i,j=0, j>i}^{n} (x_j - x_i)$$

il quale risulta sempre non nullo se i punti x_i sono distinti. Quindi il sistema lineare ha un'unica soluzione perciò $p(x)$ esiste ed è unico. □

La dimostrazione del Teorema fornisce un metodo per determinare il polinomio interpolante, ma tale metodo risulta numericamente instabile a causa

del mal condizionamento del problema di risolvere un sistema lineare avente come matrice dei coefficienti la matrice di Vandermonde. Per ovviare a questo problema è possibile rappresentare il polinomio interpolante in un altra base, ad esempio la base di Bernstein. Siano dati $n+1$ punti (x_i, y_i), $i = 0, \ldots, n$ con x_i distinti in $[a, b]$, si vuole trovare un polinomio $p(x)$ nella base di Bernstein tale che

$$p(x_i) = y_i \quad i = 0, \ldots, n$$

Siano (t_i, y_i), $i = 0, \ldots, n$ i punti traslati e scalati in $[0, 1]$ e sia

$$p(t) = \sum_{i=0}^{n} b_i B_{i,n}(t) \quad cont \in [0, 1]$$

il polinomio che deve interpolare i valori assegnati. Imponendo le condizioni di interpolazione si ottiene il sistema lineare:

$$B\mathbf{b} = \mathbf{y}$$

con

$$B = \begin{pmatrix} B_{0,n}(t_0) & B_{1,n}(t_0) & \ldots & B_{n,n}(t_0) \\ B_{0,n}(t_1) & B_{1,n}(t_1) & \ldots & B_{n,n}(t_1) \\ \ldots & & & \\ B_{0,n}(t_n) & B_{1,n}(t_n) & \ldots & B_{n,n}(t_n) \end{pmatrix} \mathbf{b} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Il vettore $\mathbf{b}$, soluzione del sistema lineare darà si il polinomio interpolante in $[0, 1]$ che in $[a, b]$.

Affinché l'errore di interpolazione sia piccolo è necessario che la funzione da interpolare sia almeno continua, in caso contrario l'errore di interpolazione potrebbe essere qualunque anche grandissimo. In alcuni casi, anche se la funzione $f(x) \in C^{\infty}_{[a,b]}$, la successione $p_n(\bar{x})$ dei valori assunti dai polinomi di interpolazione di grado n in un punto $\bar{x} \in [a, b]$ può non convergere ad $f(x)$, per $n \to \infty$.

Esempio 2.6.1. *Per il polinomio di interpolazione in n punti equidistanti della funzione di Runge*

$$f(x) = \frac{1}{1 + x^2} \quad x \in [-5, 5]$$

si può dimostrare che al crescere di n la successione dei polinomi di interpolazione sui punti

$$x_i = \frac{10}{n} i - 5$$

non converge ad f(x).

Se oltre a fare delle ipotesi sulla regolarità della funzione (almeno continua), si utilizzano delle opportune distribuzioni di punti, si può dimostrare la convergenza del polinomio interpolante $p_n(x)$ alla $f(x)$ per $x \in [a, b]$ quando $n \to \infty$. Nonostante quest'ultimo risultato, non si può essere sicuri che si otterrà una buona approssimazione di una funzione, scegliendo opportuni punti di interpolazione e un polinomio di grado alto. Per risolvere questo problema si può utilizzare l'interpolazione polinomiale a tratti.

2.6.1 Interpolazione polinomiale a tratti

L'interpolazione polinomiale a tratti consiste nell'approssimare una funzione con polinomi diversi dia grado basso in diverse parti dell'intervallo di interesse. Siano dati (x_i, y_i) $i = 0, \ldots, m$ con $x_i < x_i + 1$, un interpolante polinomiale a tratti consiste in m polinomi $p_i(x)$, $i = 0, \ldots, m-1$ di grado n, definiti sugli intervalli $[x_i, x_i + 1]$ con le seguenti proprietà:

- $p_{i-1}(x_i) \equiv p_i(x_i) = y_i$
- fissato $k \leq n - 1$, non negativo, deve valere

$$p_{i-1}^{(\ell)}(x_i) \equiv p_i^{(\ell)}(x_i) \quad \ell = 1, \ldots, k \quad i = 1, \ldots, m-1$$

L'interpolante polinomiale a tratti è di classe C^k se è una funzione continua fino alla derivata di ordine k; quando $k = n - 1$ l'interpolante polinomiale a tratti viene detta funzione *spline*. Esistono due metodi di interpolazione polinomiale a tratti molto usati nella pratica: interpolazione globale e interpolazione locale. Per interpolazione globale si intende un metodo che determina l'interpolante a tratti ottenendo i polinomi $P_i(x)$ in modo dipendente dagli altri, mentre l'interpolazione locale ricava ogni polinomio $p_i(x)$ in modo indipendente dagli altri.

2.7 Zeri di polinomi

In letteratura esistono diversi metodi per determinare gli zeri di un polinomio. In pratica spesso è molto utile determinare solo le radici reali. Uno di questi metodi è quello di Lane-Riesenfield basato sulla proprietà di Variation Diminishing dei coefficienti della base di Bernstein. Tale metodo consiste di due procedure ricorsive *Root-Isolate* e *Root*; la prima determina degli intervalli contenenti una radice, controllando che il polinomio abbia solo una variazione di segni dei coefficienti, alla fine si ottiene uno stack contenente dei polinomi che hanno una sola radice nell'intervallo di definizione. La seconda procedura, per ogni polinomio nello stack, determina la sua unica radice.

2.7.1 Metodo di Newton

Il metodo di Newton applicato ai polinomi è molto efficiente nel valutare la derivata prima, una volta valutato il polinomio. Il problema è sapere quando ci si trova nelle condizioni per poter applicare il metodo. Si può dimostrare il seguente teorema.

Teorema 2.7.1. *Sia $f(x) \in C^2_{[a,b]}$ e siano soddisfatte le seguenti condizioni*

1. $f(a)f(b) < 0$
2. $f'(x) \neq 0 \quad \forall x \in [a,b]$
3. $f''(x) \geq 0 \quad \forall x \in [a,b]$
4. *sia c l'estremo di $[a,b]$ nel quale $|f(x)|$ risulta minore e sia*

$$\left| \frac{f(c)}{f'(c)} \right| \leq b - a.$$

Sotto queste ipotesi il metodo di Newton converge all'unica soluzione per un qualsiasi iterato iniziale $x_0 \in [a,b]$.

Se la $f(x)$ è un polinomio nella base di Bernstein è facile verificare le ipotesi del teorema e stabilire se applicare il metodo di Newton. In questo caso basta considerare $x_0 = (a+b)/2$ e applicare il metodo di Newton.
Nella pratica, in molti casi un test per controllare se siamo nelle ipotesi per applicare il metodo di Newton può essere molto costoso. In questi casi conviene localizzare le radici con Root-Isolate, eseguire Newton a partire dal punto medio dell'intervallo, in ognuno degli intervalli individuati e controllare che ogni iterato resti all'interno dell'intervallo di partenza, in questo caso la successione sta convergendo, in caso contrario l'intervallo viene suddiviso in due parti in modo ricorsivo e si esegue Newton su ognuno di essi.

2.8 Applicazioni

2.8.1 Punto interno/esterno ad una curva

Data una curva chiusa piana $\underline{C}(t)$ $t \in [0,1]$ nella forma di Bézier e un punto Q del piano, si vuole determinare se Q è interno o esterno alla curva. Per risolvere questo problema si può pensare di determinare le intersezioni tra la curva $\underline{C}(t)$ e la semiretta uscente da Q; se il numero delle intersezioni è dispari, il punto è interno, altrimenti se è pari o zero, il punto è esterno. Siano

$$\underline{C}(t) = \begin{pmatrix} C_1(t) \\ C_2(t) \end{pmatrix} \quad e \quad ax + by + c = 0$$

rispettivamente l'espressione della curva in forma parametrica con $t \in [0,1]$ e l'equazione della retta in forma cartesiana. Sostituendo le componenti x e y della curva nel'eqauzione della retta si ha:

$$aC_1(t) + bC_2(t) + c = 0.$$

Se $\underline{C}(t) = \sum_{i=0}^{n} P_i B_{i,n}(t)$ con $P_i = (x_i, y_i)^T$, sostituendo si ottiene

$$a\sum_{i=0}^{n} x_i B_{i,n}(t) + b\sum_{i=0}^{n} y_i B_{i,n}(t) + c\sum_{i=0}^{n} B_{i,n}(t) = 0$$

e quindi

$$\sum_{i=0}^{n}(ax_i + by_i + c)B_{i,n}(t) = 0.$$

Si tratta di determinare gli zeri di un polinomio di grado n nell'intervallo $[0, 1]$. Le soluzioni saranno i parametri della curva in corrispondenza dei quali la curva e la retta si intersecano. Valutando la curva in corrispondenza di tali parametri si ottengono le coordinate cartesiane dei punti di intersezione. Quindi per determinare se un punto $Q = (q_x, q_y)^T$ è interno o esterno, si considera la retta orizzontale $y = q_y$ e si determinano le radici di

$$\sum_{i=0}^{n}(y_i - q_y)B_{i,n}(t) = 0$$

si valuta la componente x della curva in corrispondenza delle soluzioni trovate e si contano i valori trovati maggiori di q_x.

2.8.2 Punti estremi di una curva

Siano

$$\underline{C}(t) = \begin{pmatrix} C_1(t) \\ C_2(t) \end{pmatrix} \quad e \quad \underline{C}'(t) = \begin{pmatrix} C_1'(t) \\ C_2'(t) \end{pmatrix}$$

rispettivamente l'espressione della curva in forma parametrica con $t \in [0, 1]$ e l'espressione della curva derivata prima che rappresenta il vettore tangente alla curva $\underline{C}(t)$ per ogni t; i punti estremi saranno i punti della curva che hanno vettore tangente verticale o orizzontale. Si cercano quindi i punti t^* tali che:

$$C_1'(t^*) = 0 \quad e \quad C_2'(t^*) \neq 0$$

e

$$C_2'(t^*) = 0 \quad e \quad C_1'(t^*) \neq 0$$

Si tratta di determinare le radici di equazioni polinomiali e verificare che l'altra componente non si annulli in corrispondenza di queste soluzioni.
Se si cercano i punti estremi lungo una direzione $D = (d_x, d_y)$, è sufficiente

determinare gli zeri dell'equazione

$$\underline{C}'(t) \cdot D = 0$$

cioè

$$\underline{C}'_1(t)d_x + \underline{C}'_2(t)d_y = 0.$$

2.8.3 Intersezioni tra due curve

Siano $\underline{C}(t)$ e $\mathbf{D}(s)$, $t, s \in [a, b]$, due curve, si vogliono determinare i loro punti di intersezione.
Un metodo molto semplice per trovare tali punti consiste in una fase in cui si suddividono le curve in corrispondenza dei loro punti estremi. In questo modo si ottengono due insiemi di tratti di curva corrispondenti a $\underline{C}(t)$ e $\mathbf{D}(s)$; l'algoritmo continua a confrontare ogni tratto della $\underline{C}(t)$ con quelli di $\mathbf{D}(s)$. Ogni segmento di curva è contenuto in un rettangolo dato dai vertici opposti del segmento di curva. Questo consente di implementare un test in cui si determina se il rettangolo definito dagli estremi di un tratto si interseca con il rettangolo di un altro, in questo caso si suddividono i due tratti a metà e si controllano i quattro accoppiamenti.

2.8.4 Lunghezza di una curva

Sia $\underline{C}(t)$ con $t \in [0, 1]$ una curva di Bézier. Vogliamo determinare la sua lunghezza e progettare curve di lunghezza assegnata mantenendo la stessa forma. La lunghezza di una curva è data da

$$L = \int_0^1 ||\underline{C}'(t)||dt = \int_0^1 \sqrt{x'(t)^2 + y'(t)^2}dt.$$

Per calcolarla è necessario utilizzare una formula di integrazione numerica.
Una volta progettata la curva e calcolato la sua lunghezza L_C, vogliamo determinare una curva $\underline{G}(t)$ di lunghezza L_G avente la stessa forma di $\underline{C}(t)$. Se la curva $\underline{C}(t)$ viene scalata di un fattore $s > 0$ la sua lunghezza viene

scalata dello stesso fattore, sia

$$\underline{G}(t) = \begin{pmatrix} s \cdot x(t) \\ s \cdot y(t) \end{pmatrix}$$

la curva $\underline{C}(t)$ scalata di un fattore s, allora

$$L_G = \int_0^1 \sqrt{[s \cdot x'(t)]^2 + [s \cdot y'(t)]^2} dt = s \int_0^1 \sqrt{x'(t)^2 + y'(t)^2} dt = sL_C.$$

Quindi assegnata L_G si scala la curva $\underline{C}(t)$ di un fattore $s = L_G/L_C$ ottenendo la curva $\underline{G}(t)$ della stessa forma di $\underline{C}(t)$.

2.8.5 Area di una curva

Sia $\underline{C}(t)$ con $t \in [0, 1]$ una curva di Bézier. Vogliamo determinare l'area che tale curva sottende con l'origine degli assi e a progettare curve della stessa forma di area assegnata.
L'area di una curva di questo tipo è data da

$$A = \pm\frac{1}{2} \int_0^1 \underline{C}(t) \times \underline{C'}(t) dt = \pm\frac{1}{2} \int_0^1 [x(t)y'(t) - x'(t)y(t)] dt$$

per calcolarla è necessario usare una formula di integrazione numerica. Una volta progettata un curva $\underline{C}(t)$ di area A_C, siamo interessati ad una curva $\underline{G}(t)$ con la stessa forma di $\underline{C}(t)$, ma di area A_G fissata. La curva $\underline{C}(t)$ è scalata di un fattore s e l'area viene scalata dello stesso fattore al quadrato; sia

$$\underline{G}(t) = \begin{pmatrix} s \cdot x(t) \\ s \cdot y(t) \end{pmatrix}$$

la curva $\underline{C}(t)$ scalata del fattore s, allora sarà

$$A_G = \int_0^1 [s^2 x(t)y'(t) - s^2 x'(t)y(t)] dt = s^2 \int_0^1 [x(t)y'(t) - x'(t)y(t)] dt = s^2 A_C.$$

Quindi data A_G si calcola $s = \sqrt{A_G/A_C}$ e si scala la curva $\underline{C}(t)$ ottenendo una curva $\underline{G}(t)$ della stessa forma e di area desiderata.

Capitolo 3

Architettura di Mini System

3.1 Mini System

Come già detto nell'introduzione, Mini System è un software didattico sviluppato dal Prof. Giulio Casciola. Questo programma consente diverse operazioni.

3.1.1 Operazioni di disegno e modifica di curve

Mini System mette a disposizione le seguenti funzioni di disegno e modifica di curve di Bézier:

New curve: consente di disegnare una curva di Bézier.

Reset curves: cancella tutte le curve di Bézier presenti nell'area di disegno.

Delete curve: cancella la curva attiva.

Modift cp: consente di modificare un punto di controllo di una curva di Bézier, e di ridisegnarla con la nuova poligonale di controllo.

Split: effettua la suddivisione di una curva in un punto selezionato con il puntatore del mouse, oppure di due curve che si intersecano in corrispondenza dei punti di intersezione.

Rotate: consente di ruotare la curva attiva.

Intersect: calcola e visualizza i punti di intersezione tra due curve.

Redraw: ridisegna le curve presenti nell'area di disegno.

Fill: effettua il *filling* delle curve presenti nell'area di disegno.

Join C0/G0: effettua il *join* di due curve con continuità $C0$.

Join G1: effettua il *join* di due curve con continuità $G1$.

Join C1: effettua il *join* di due curve con continuità $C1$.

Tangent vec: disegna i vettori tangenti ad una curva.

Normal vec: disegna i vettori normali ad una curva.

Antialiasing: consente di disegnare delle curve di Bézier con antialiasing.

Int. gl.points: esegue l'interpolazione di un set di punti utilizzando un metodo di interpolazione globale .

Int. lc.points: esegue l'interpolazione di un set di punti utilizzando un metodo di interpolazione a tratti locale.

vis.norm: disegna una linea che collega i punti estremi di una curva di Bézier.

3.1.2 Operazioni di input/output

Mini System consente di salvare su file e di caricare da file delle curve di Bézier, in vari formati.

Open curve: carica una curva da file.

Save curve: salva una curva su file .

Open .all: carica una curva da file .

Save .all: salva una curva su file.

Load gl.points: carica da file dei punti da interpolare con metodo di interpolazione a tratti globale.

Load lc.points: carica da file dei punti da interpolare con metodo di interpolazione a tratti locale.

Save points: salva dei punti su file.

Open ps: carica un file in formato Postscript.

3.1.3 Operazioni di visualizzazione

Mini System consente di effettuare le seguenti operazioni di visualizzazione:

Zoom in/out esegue lo zoom in oppure lo zoom out dell'area di disegno, a seconda del bottone del mouse che viene premuto dall'utente.

Zoom area esegue lo zoom di un area selezionata dall'utente.

Zoom default riporta lo zoom allo stato iniziale.

Scale in/out scala la curva attiva, ingrandendola o rimpicciolendola, a seconda del bottone del mouse che viene premuto dall'utente.

Axes yes/no: Visualizza o cancella gli assi cartesiani.

CP yes/no: Visualizza o cancella i punti di controllo delle curve.

3.1.4 Altre operazioni

Active curve: operazione che consente di attivare una curva.

Curve length: calcola e visualizza la lunghezza di una curva.

Curve area: calcola e visualizza l'area di una curva.

Reverse: inverte l'ordine dei punti di controllo di una curva.

3.1.5 L'interfaccia grafica

L'interfaccia grafica di Mini System si compone di un'area di disegno e di un unico menu, situato nella parte destra della finestra, che contiene tutte le funzioni del programma.

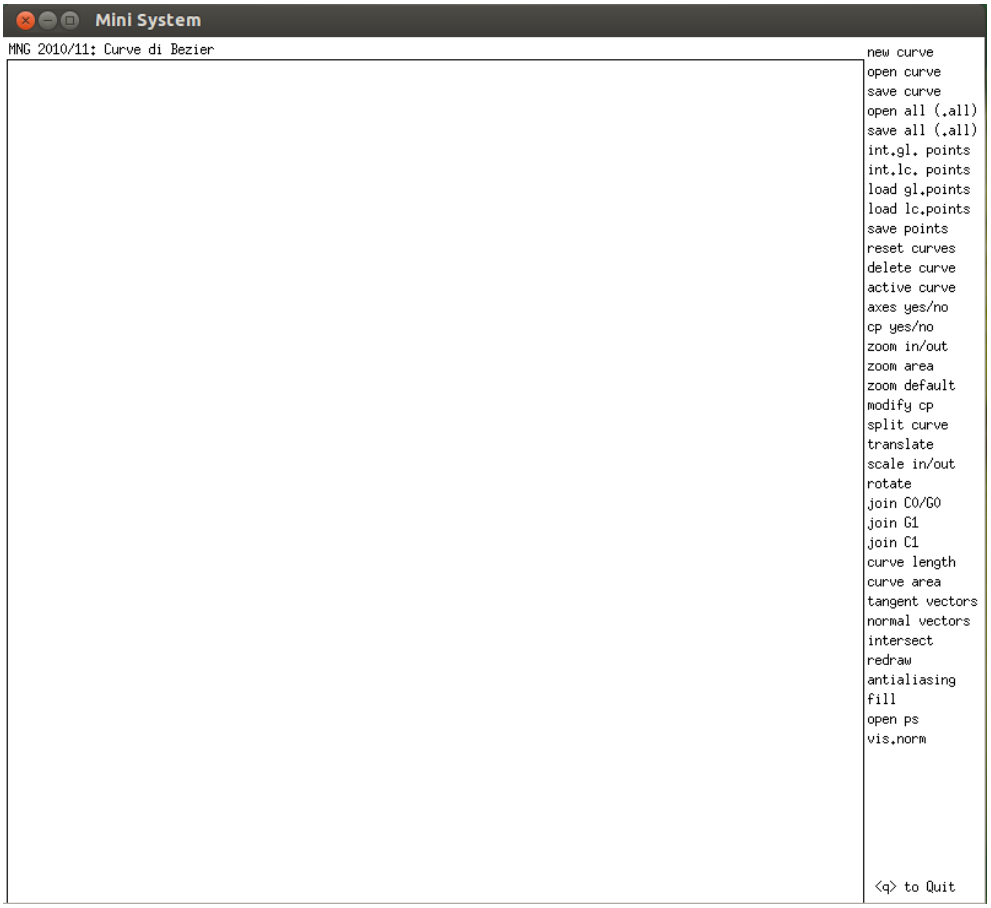

Figura 3.1: La GUI di Mini System

3.2 Struttura del codice

Il codice sorgente di Mini System comprende i file di seguito indicati.

- `main.c` contiene il ciclo principale del programma, le procedure che implementano l'interfaccia grafica, le funzioni di callback e le funzioni di calcolo richiamate da quest'ultime.
- `draw_bres.c` contiene la procedura di disegno di una linea, che utilizza l'algoritmo della linea di Bresenham per effettuare l'antialiasing di una curva. Contiene inoltre le procedure e le funzioni, richiamate dalla procedura di disegno della linea sopra indicata.
- `draw.c` contiene le procedure di gestione della finestra, le procedure di trasformazione Window to Viewport e Viewport to Window e le procedure di disegno per le curve di Bézier.
- `gra_fun.c` contiene le procedure di inizializzazione del sistema XWindow e alcune procedure per il disegno di linee, quadrati, cerchi, ecc. che a loro volta richiamano delle procedure del sistema XWindow.
- `interp.c` contiene le procedure per l'interpolazione di un insieme di punti mediante curva di Bézier.
- `io.c` contiene le funzioni di input/output per caricare da file e salvare su file curve di Bézier.
- `quanc8.c` contiene una funzione, basata sulle formule di Newton—Côtes per il calcolo dell'integrale di una funzione, utilizzata per calcolare la lunghezza e l'area di una curva di Bézier.
- `scan_conv.c` contiene le procedure per lo *scan conversion* di uno o più poligoni.
- `zerip.c` contiene la procedura che implementa il metodo di *Bézier clipping* per curve polinomiali, utilizzata per trovare i punti di intersezione tra due curve.

Lo schema logico ricavato dall'analisi del codice visualizza le dipendenze tra i vari moduli e soprattutto le dipendenze tra la libreria Xlib e il resto del codice.

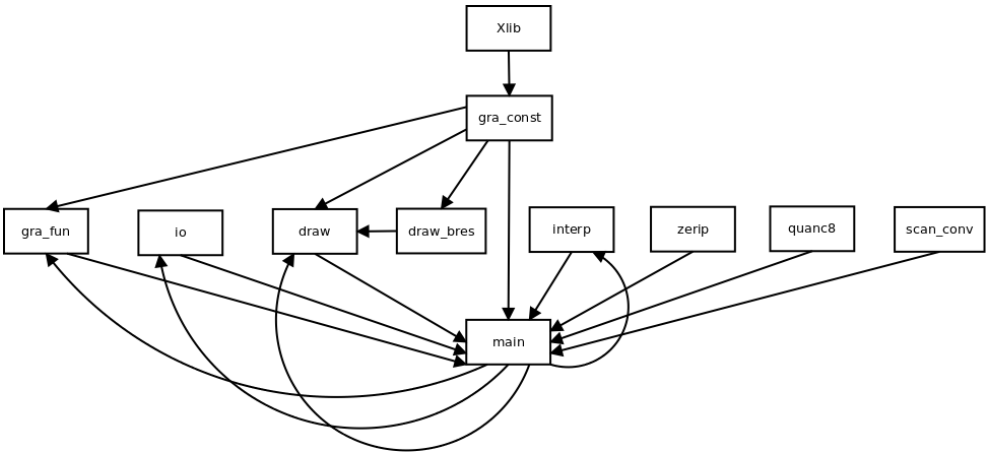

Figura 3.2: La struttura del codice di Mini System

3.3 Le modifiche effettuate

Dall'analisi della struttura del codice, si evince che la GUI non è separata dalle funzioni di calcolo e di disegno, di conseguenza una sua eventuale modifica influirebbe su tutto il resto. Per ovviare al problema, il presente lavoro modifica la struttura di Mini System con l'introduzione di una nuova interfaccia grafica, basata sulle librerie SDL, separa questa dalle funzioni di calcolo e di disegno, sostituisce queste ultime con quelle della libreria GCGraLib.

3.4 La nuova interfaccia grafica

Nel progettare la nuova interfaccia grafica, si è scelto di utilizzare una barra di menu a tendina ed una barra di bottoni situata sulla parte sinistra della finestra. L'area situata a destra della barra dei bottoni e sotto la barra dei menu costituisce l'area di disegno. La barra dei menu contiene tre menu a tendina:

- File contiene le funzioni di input/output;
- Curve contiene le funzioni di manipolazione di curve di Bézier;

- View contiene le funzioni di zoom, di scala di curve di Bézier e di visualizzazione degli assi cartesiani e dei punti di controllo.

La barra dei bottoni contiene due colonne di bottoni, con i quali è possibile eseguire delle operazioni di creazione, cancellazione e manipolazione di curve di Bézier ed altre funzioni, tra le quali lo zoom in/out e lo zoom di un'area.

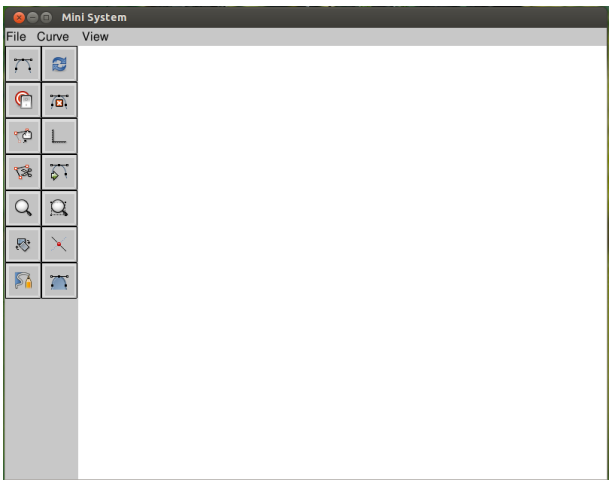

Figura 3.3: La nuova GUI di Mini System

3.5 Implementazione della nuova GUI

Poiché la libreria SDL è una libreria multimediale che non fornisce gli strumenti per creare un'interfaccia grafica, si sono dovuti implementare gli elementi che costituiscono una GUI, utilizzando quanto messo a disposizione da SDL. Nell'implementare la nuova GUI, si è scelto di creare un modulo button che fornisce le funzioni che creano e visualizzano i bottoni, un modulo buttonbar per la gestione di una barra di bottoni, un modulo menu per creare e visualizzare dei menu a tendina e una libreria menubar per la gestione di una barra di menu.

3.5.1 I moduli button e buttonbar

Il modulo button fornisce le funzioni che consentono di creare e posizionare, in un punto preciso dello schermo, dei bottoni di dimensione 48×48 pixel contenenti un'icona, oppure dei bottoni contenenti un testo. Sono inoltre in grado di visualizzare un'etichetta esplicativa, che compare quando il puntatore del mouse viene posizionato su un bottone. Il modulo buttonbar inizializza e visualizza una barra di bottoni ed è in grado di gestire gli eventi che la riguardano.
Si sono utilizzati i moduli sopradescritti per creare la barra dei bottoni a due colonne, contenenti un'icona, in cui viene visualizzata un'etichetta ogniqualvolta il mouse viene posizionato su un bottone.
Il file `button.h` contiene la seguente struttura:

```
typedef struct button
{
    SDL_Surface *button_s;
    SDL_Surface *icon;
    SDL_Rect button_r;
    SDL_Rect icon_r;
    SDL_Rect iconpress_r;
    char label[LABELLENGTH];
    int pressed;
} Button;
```

Gli elementi `button_s` e `icon` sono delle superfici SDL, mentre `button_r` e `icon_r` sono i rettangoli che fanno riferimento rispettivamente alle superfici sopradescritte. L'elemento `iconpress_r` è il rettangolo in cui viene visualizzata l'icona contenuta all'interno di un bottone quando questo viene cliccato, cioè quando la variabile `pressed` è uguale a 1. L'elemento `label` è una stringa di testo contenente l'etichetta associata al bottone, che, se si tratta di un bottone di testo, viene visualizzata all'interno dello stesso.

Il file `buttonbar.h` contiene la seguente definizione di tipo:

```
typedef Button ButtonBar[MAXROW][MAXCOL];
```

cioè la barra dei bottoni è una matrice di `Button`.

3.5.2 I moduli menu e menubar

Il modulo menu contiene le strutture dati su cui si basano i menu a tendina e le procedure di inizializzazione, di visualizzazione dei menu e la gestione degli eventi che li riguardano.
Il modulo menubar si occupa di inizializzare, visualizzare e gestire una barra di menu.
Il file `menu.h` contiene le seguenti strutture:

```
typedef struct menuentry
{
    char label[ENTRY_L];
    int selected;
    SDL_Rect entry_r;
} MenuEntry;

typedef struct MENU
{
    MenuEntry menu[NUM_ENTRY];
    SDL_Rect menu_r;
    int numentry;
} Menu;
```

La struttura `MenuEntry` definisce una entrata di un menu a tendina; l'elemento `label` è la stringa di testo da visualizzare, l'elemento `selected` indica se la stringa di testo vada evidenziata o meno, `entry_r` è il rettangolo all'interno del quale viene visualizzata l'etichetta.
La struttura `Menu` definisce un menu a tendina; l'elemento menu è un array di `MenuEntry`, `menu_r` è il rettangolo all'interno del quale viene visualizza-

to l'intero menu, mentre `numentry` rappresenta il numero delle entrate del menu.

Il file `menubar.h` contiene la struttura:

```
typedef struct menubar
{
        Menu menubar[NUM_MENU];
} MenuBar;
```

cioè la barra dei menu è un array di menu.

3.6 Le nuove funzioni di disegno

Le nuove funzioni di disegno si sono ottenute sostituendo le funzioni di disegno della libreria Xlib nel file `gra_fun.c`, con le corrispondenti funzioni della libreria GCGraLib.

3.7 Separazione della GUI dalle funzioni di calcolo e disegno

Per separare la GUI dal resto del codice, si è creato un livello intermedio tra l'interfaccia e le funzioni di calcolo, in modo tale che la GUI utilizzi dei moduli che trasferiscono i comandi dati dall'utente alle funzioni di calcolo, le quali successivamente interagiscono con la GUI per visualizzare in output, mediante le funzioni di disegno, i risultati delle loro elaborazioni. In questo modo la parte che si occupa dei calcoli dipende solo indirettamente dalla libreria SDL, quindi se in futuro si vorrà cambiare la GUI sarà sufficiente modificare quest'ultima e i moduli del livello intermedio, senza andare a modificare le funzioni di calcolo.

Per realizzare tutto questo si sono implementati i seguenti moduli: event, callback e screen.
Il modulo event si occupa della gestione degli eventi, quali il movimento

del puntatore del mouse, la pressione e il rilascio dei pulsanti del mouse, il ridimensionamento della finestra; per ognuno di questi eventi c'è una routine di gestione. Il modulo traduce inoltre gli eventi generati dalla libreria SDL in eventi indipendenti dalla libreria, cosicché cambiando la libreria grafica basterà sostituire, nel modulo event, gli eventi SDL con gli eventi specifici della nuova libreria.
Il modulo screen fornisce delle operazioni riguardanti lo schermo, quali la pulitura, la sua memorizzazione in una variabile temporanea, in modo da poterlo ripristinare in una fase successiva dell'elaborazione e il *blitting* della superficie del viewport sulla superficie dello schermo principale.
Il modulo callback, come suggerisce il nome, fornisce le funzioni di callback, cioè le azioni associate alla pressione di un bottone o alla selezione di una voce di un menu. Queste funzioni sono state spostate dal file `main.c` al file `callback.c`.

Per rendere ancora più modulare il codice sorgente si sono effettuate le seguenti modifiche:

- le funzioni del menu curve sono state spostate dal file `main.c` al file `curve.c`;
- le funzioni del menu view sono state spostate dal file `main.c` al file `view.c`;
- le operazioni riguardanti le curve di Bézier sono state spostate dal file `main.c` al file `bezier.c`.

Conclusioni

Questo lavoro di tesi descrive il procedimento seguito per effettuare un'operazione di refactoring sul programma Mini System.
L'intervento ha riguardato in particolare la sostituzione dell'interfaccia grafica, basata sulla libreria Xlib, con un altra basata sulla libreria SDL, la sostituzione delle funzioni di disegno della libreria Xlib con quelle della libreria GCGraLib e la riorganizzazione del codice.
Per comprendere meglio li lavoro svolto, sono state introdotte le librerie Xlib e, soprattutto, le librerie SDL con alcuni esempi di utilizzo. Si è passati poi ai polinomi di Bernstein e alle curve di Bézier, sviluppandone nel dettaglio il discorso.
Si è proceduto successivamente ad un'analisi di Mini System con relativa rilevazione e rappresentazione grafica della struttura e quindi alla spiegazione delle modifiche apportate.
Dall'analisi del codice è emerso il fatto che la GUI non è separata dalle altre parti del programma. Si è quindi creata una nuova interfaccia grafica, alla quale si sono collegate le funzioni di calcolo e di disegno; si è riorganizzato il codice, spostando tutte le funzioni definite in `main.c` su dei moduli separati; si è separata la GUI dalle funzioni di disegno e di calcolo, rendendo queste indirettamente dipendenti dalla libreria SDL.

Il lavoro svolto potrà consentire in futuro di portare Mini System su altri sistemi operativi, apportando soltanto delle piccole modifiche, in quanto SDL è una libreria multipiattaforma. Inoltre se si vorrà sostituire la GUI o se si vorranno cambiare le librerie grafiche non sarà necessario modificare le

funzioni che si occupano dei calcoli, ma solo l'interfaccia grafica e le funzioni di disegno.

Appendice A

Guida utente di Mini System

A.1 Operazioni di disegno e modifica di curve di Bézier

A.1.1 Disegno di una curva

Per disegnare una fare clic sul pulsante *new curve* come mostrato in figura A.1; fare clic sull'area di disegno in corrispondenza dei punti in cui vanno

Figura A.1: Bottone per la creazione di una nuova curva

posizionati i punti di controllo della curva, per terminare l'operazione fare clic con il pulsante destro del mouse, a questo punto compare nell'area di disegno la curva di Bézier Per disegnare un'altra curva ripetere lo stesso procedimento, quando vengono disegnate più curve, la curva attiva è quella che ha dei cerchi come punti di controllo.

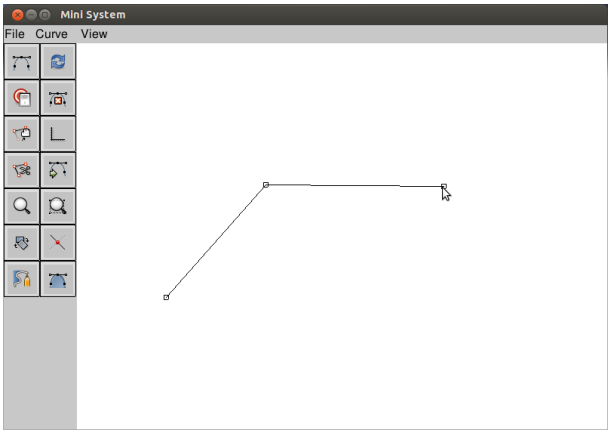

Figura A.2: Posizionamento dei punti di controllo

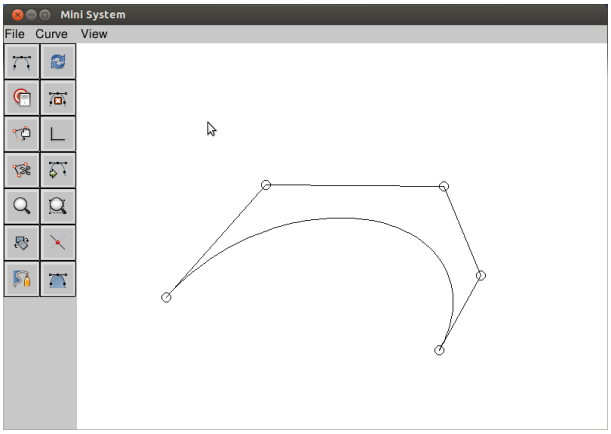

Figura A.3: Disegno di una curva di Bézier

A.1.2 Attivazione di una curva

Tutte le operazioni che Mini System è in grado di effettuare vengono eseguite sulla curva attiva. Quando nell'area di disegno sono presenti più di una curva, se si vuole eseguire un'operazione su una curva diversa da quella attiva è necessario attivarla. Per attivare una curva fare clic sul bottone *active curve*, (fig. A.4) e poi fare clic sulla curva che si desidera attivare.

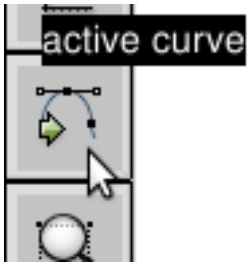

Figura A.4: Bottone per l'attivazione di una curva

A.1.3 Cancellazione e reset di curve

Per cancellare una curva fare clic sul bottone *delete curve*, come mostrato in figura A.5, la curva attiva verrà cancellata. Se si vuole cancellare una curva

Figura A.5: Bottone per la cancellazione di una curva

diversa da quella attiva è necessario attivarla con il bottone *active curve* (fig. A.4) e poi cancellarla con il bottone *delete curve*.

Per fare il reset, cioè cancellare tutte le curve presenti nell'area di disegno, fare clic su *reset curves* (fig. A.6)

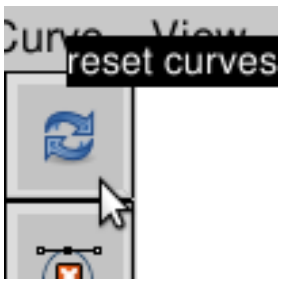

Figura A.6: Bottone per il reset di curve

A.1.4 Modifica dei punti di controllo

Per modificare la posizione di un punto di controllo, fare clic sul bottone *modify cp* (fig. A.7), successivamente selezionare con il pulsante sinistro del

Figura A.7: Bottone per la modifica di un punto di controllo

mouse il punto di controllo che si vuole modificare (fig. A.8), infine fare

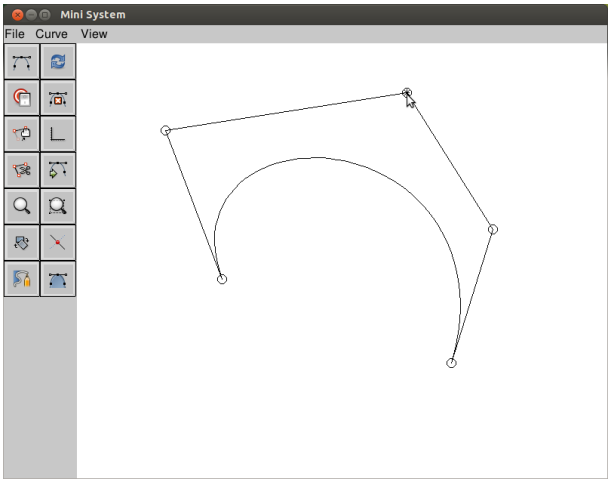

Figura A.8: Selezione di un punto di controllo

clic nel punto dell'area di disegno in cui si desidera posizionare il punto di controllo, la curva verrà ridisegnata con la nuova poligonale di controllo (fig. A.9).

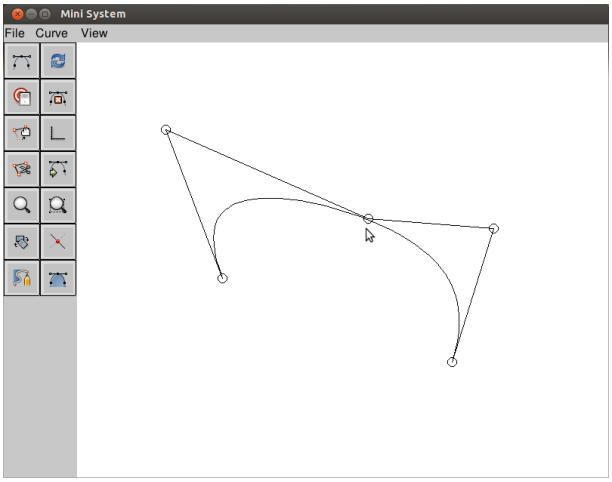

Figura A.9: Curva di Bézier ridisegnata con la nuova poligonale di controllo

A.1.5 Suddivisione di una curva

Per suddividere una curva in un punto, fare clic sul bottone *split* (fig. A.10), e successivamente fare clic sul punto della curva nel quale si vuole suddividere (figg. A.11 e A.12).

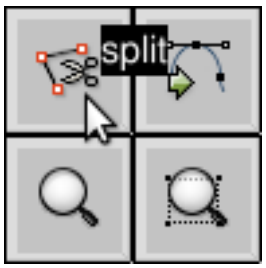

Figura A.10: Bottone per la suddivisione di una curva

A.1.6 Ridisegnare lo schermo

Per ridisegnare lo schermo fare clic sul pulsante *redraw* (fig. A.13) oppure selezionare l'omonima voce dal menu *curve*.

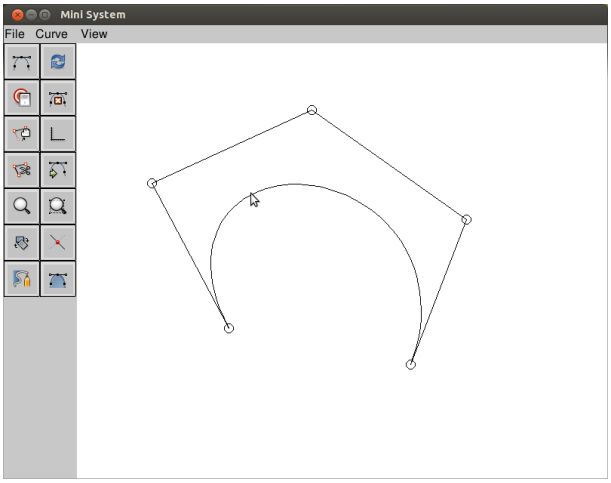

Figura A.11: Selezione del punto in cui suddividere

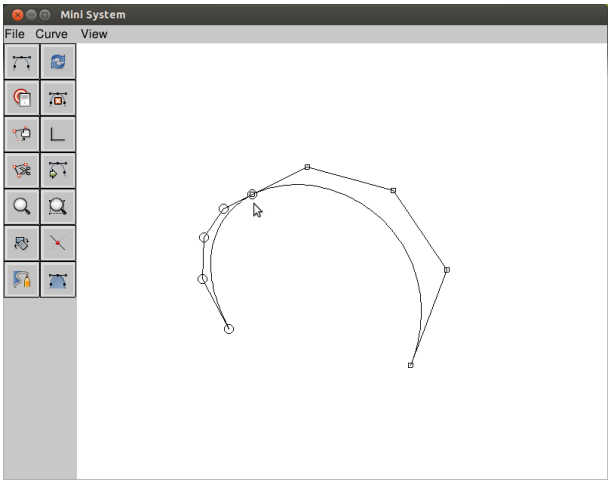

Figura A.12: Curva suddivisa

Figura A.13: Bottone per ridisegnare lo schermo

A.1.7 Filling di regioni

Per effettuare il *filling* di regioni, fare clic sul pulsante *fill* (fig. A.14). Le

Figura A.14: Bottone per il filling di regioni

regioni dello schermo racchiuse da curve di Bézier risulteranno riempite (fig. A.15).

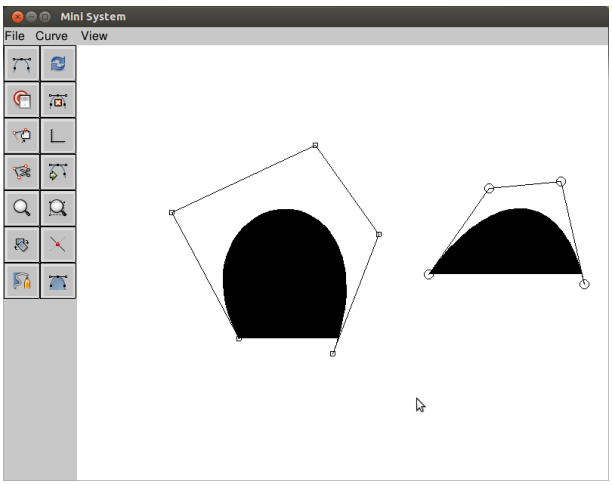

Figura A.15: Filling di regioni

A.1.8 Intersezione di curve

Per trovare i punti di intersezione fra due curve, fare clic sul bottone *intersect* (fig. A.16) e selezionare col mouse la prima e la seconda curva delle quali calcolare l'intersezione. Se le curve si intersecano compariranno degli asterischi nei punti di intersezione (fig. A.17) A questo punto è possibile

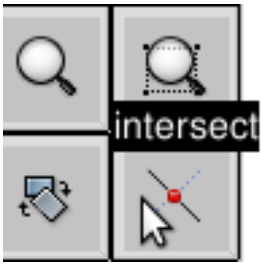

Figura A.16: Bottone intersect

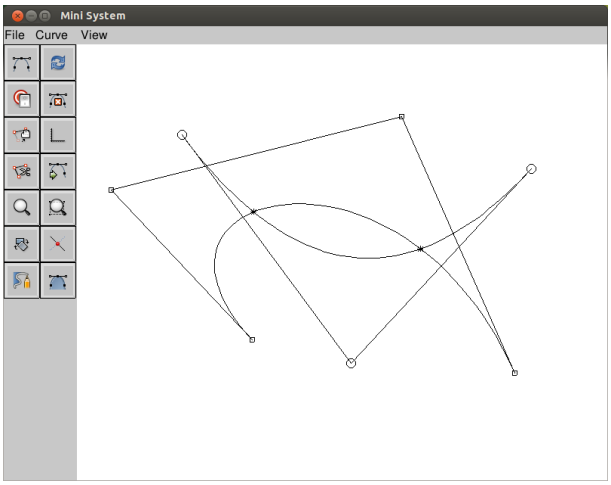

Figura A.17: Punti di intersezione fra due curve

suddividere le curve nei punti di intersezione facendo clic sul bottone *split*.

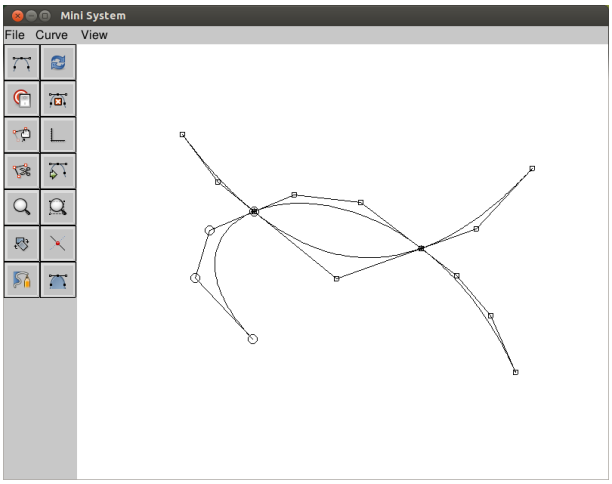

Figura A.18: Suddivsione di curve nei punti di intersezione

A.1.9 Rotazione

Per ruotare una curva fare clic sul bottone *rotate* (fig. A.19), successivamente fare clic sull'area di disegno, nel punto in cui si vuole far ruotare la curva.

Figura A.19: Rotazione di una curva

A.2 Operazioni di input/output

Le operazioni di input/output, quali apertura e salvataggio di file, sono tutte contenute nel menu *File*. Ad esempio se si vuole aprire una curva

memorizzata in un file, selezionare *Open curve* dal menu *File*, a questo punto comparirà una maschera in cui inserire il nome del file contenente la curva (fig. A.20), per confermare basta premere il tasto Invio oppure fare clic sul bottone *OK*. Le altre operazioni di input/output sono analoghe.

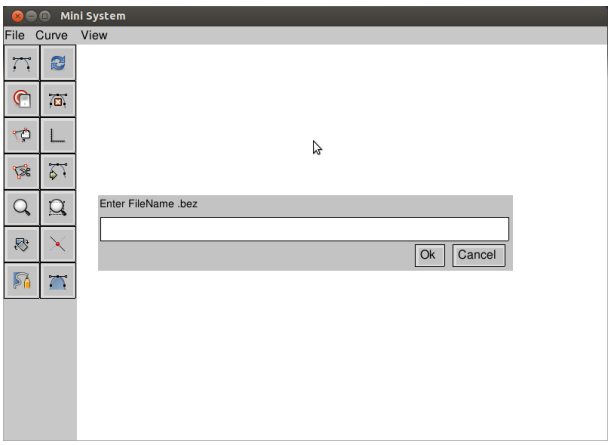

Figura A.20: La maschera di input/output

A.3 Operazioni sulle curve di Bézier

Le operazioni che è possibile effettuare sulle curve di Bézier sono contenute nel menu *Curve*.

A.3.1 Join di curve

Per effettuare il join di due curve, con continuità C0, C1 o G1, scegliere la voce corrispondente dal menu *Curve*, successivamente selezionare con il mouse la prima e la seconda curva di cui si vuole eseguire il join. A queste punto le due curve risulteranno unite.

A.3.2 Vettori normali e tangenti

Per visualizzare i vettori normali o tangenti ad una curva selezionare le voci *Normal vec* o *Tangent vec* dal menu *Curve*. Ora compariranno i vettori

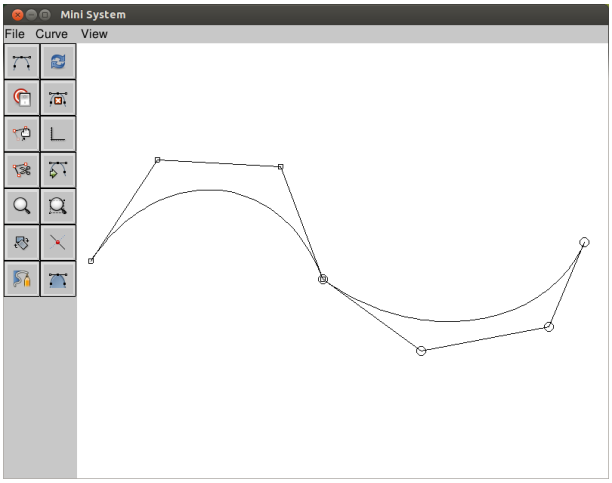

Figura A.21: Join C0/G0 di due curve

normali o tangenti alla curva attiva(figg. A.22, A.23).

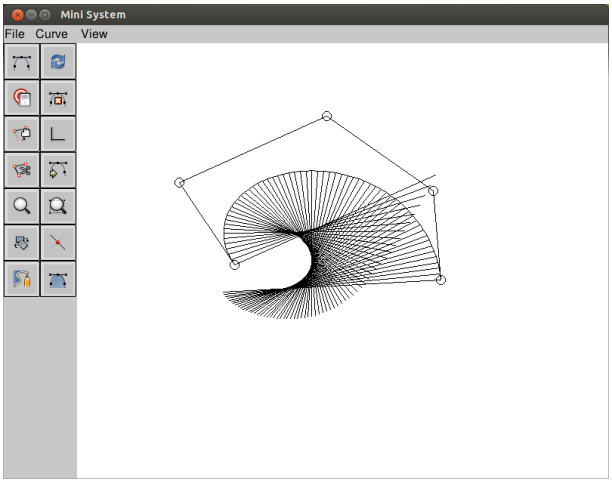

Figura A.22: Vettori normali ad una curva

A.3.3 Antialiasing

Per eseguire l'antialiasing di curve scegliere l'omonima voce dal menu *Curve*, ora tutte le curve verranno disegnate con antialiasing (fig. A.24) fino a quando non verrà selezionata nuovamente la voce *Antialiasing*.

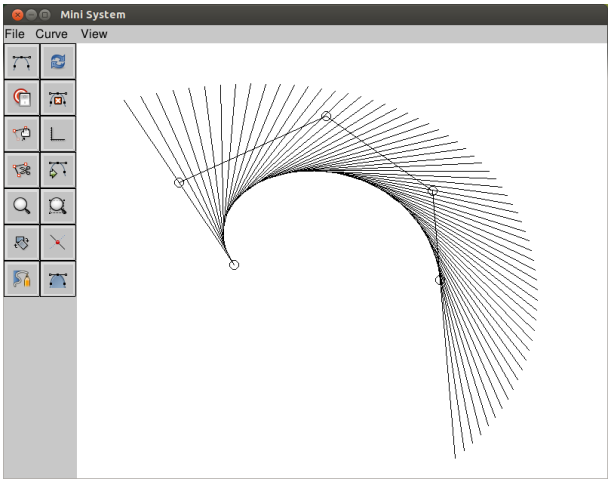

Figura A.23: Vettori tangenti ad una curva

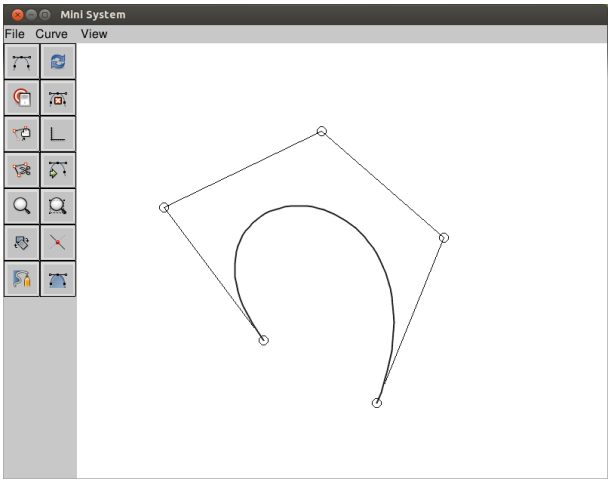

Figura A.24: Disegno con antialiasing

A.3.4 Ridisegno, intersezione, attivazione

Queste operazioni sono analoghe a quelle che è possibile eseguire premendo i corrispondenti bottoni.

A.3.5 Traslazione

Per traslare una curva selezionare la voce *Translate* dal menu *Curve* e fare clic sul punto dell'area di disegno in cui si vuole eseguire la traslazione della curva attiva.

A.3.6 Interpolazione

Per interpolare un insieme di punti tramite una curva di Bézier, utilizzando un metodo di interpolazione polinomiale a tratti, selezionare la corrispondente voce dal menu *Curve* (*Int. gl.points* per interpolazione globale a tratti, *Int. lc.points* per interpolazione locale a tratti). Successivamente fare clic nell'area di disegno, in corrispondenza dei punti che si vogliono interpolare (in ogni punto compare una x). Per terminare premere il tasto destro del mouse. A questo punto compare la curva che interpola i punti dati (figg. A.25, A.26).

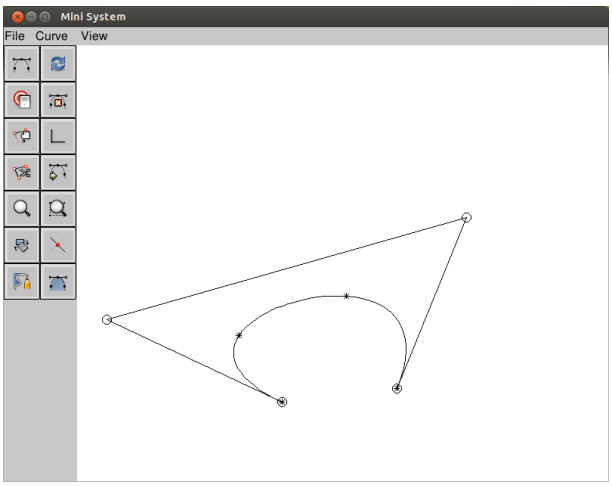

Figura A.25: Interpolazione globale a tratti

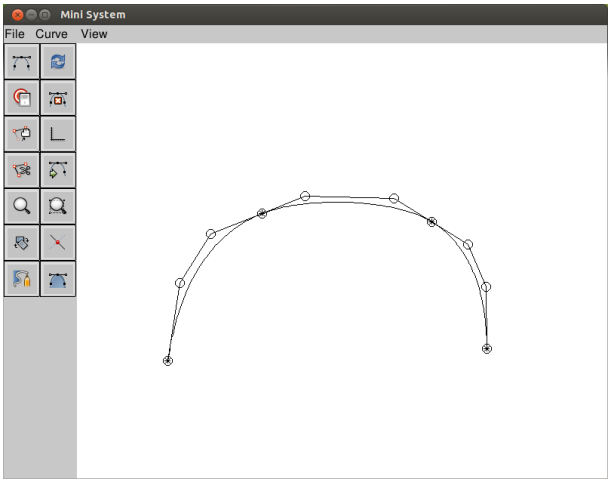

Figura A.26: Interpolazione locale a tratti

A.3.7 vis.norm

Per disegnare una linea che congiunge i punti estremi della curva attiva, selezionare *vis.norm* dal menu *Curve*.

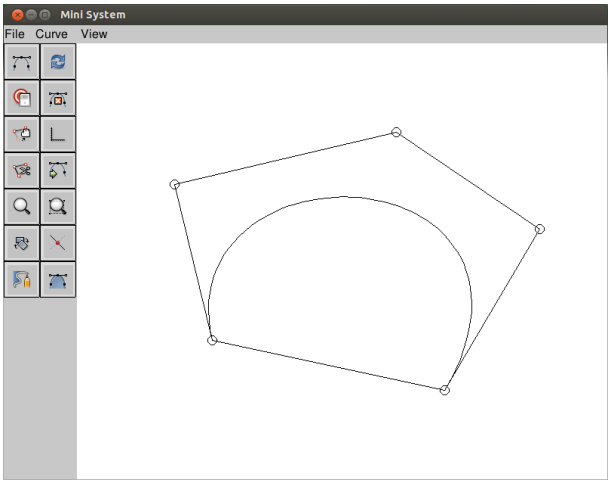

Figura A.27: vis.norm

A.3.8 Lunghezza e area

Per visualizzare la lunghezza o l'area di una curva selezionare la voce *Curve length* oppure la voce *Curve area* dal menu *Curve*, comparirà sullo schermo una maschera contenente i dati che si vogliono visualizzare (fig. A.28).

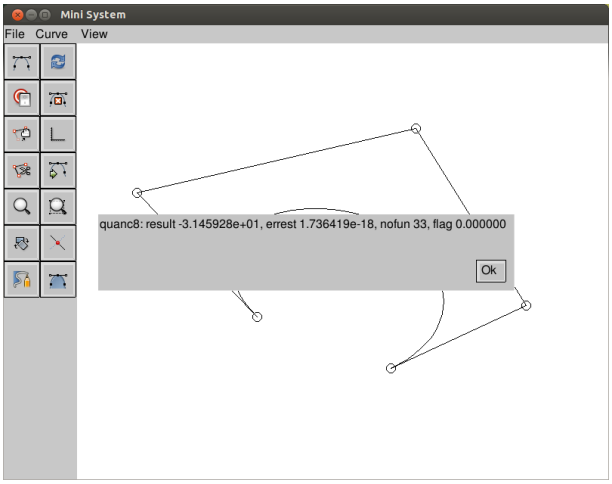

Figura A.28: Area di una curva

A.3.9 Inversione dell'ordine di una curva

Per invertire l'ordine dei punti di controllo, selezionare la voce *Reverse* dal menu *Curve* e selezionare con il mouse la curva di cui si desidera invertire l'ordine.

A.4 Operazioni di visualizzazione

Le operazioni di visualizzazione sono contenute nel menu *View*, per effettuare una di queste operazioni basta selezionarla dal suddetto menu.

A.4.1 Visualizzazione dei punti di controllo e degli assi

Per visualizzare o meno i punti di controllo è sufficiente fare clic sul bottone *cp yes/no* (fig. A.29) oppure selezionare la voce omonima nel menu *View*. Per visualizzare gli assi si procede in modo analogo.

Figura A.29: Il bottone di visualizzazione dei punti di controllo

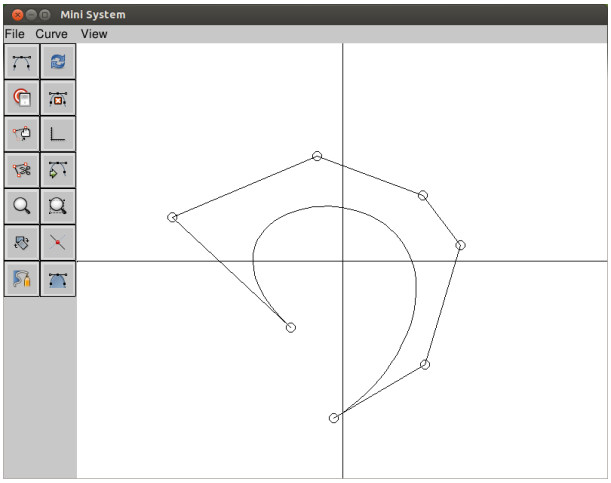

Figura A.30: Visualizzazione degli assi cartesiani

A.4.2 Zoom in/out

Per effettuare lo zoom fare clic sul bottone *zoom in/out* (fig. A.31, successivamente posizionare il mouse su un punto dell'area di disegno. A questo punto premere il tasto sinistro del mouse per effettuare lo zoom in o il tasto destro per lo zoom out.

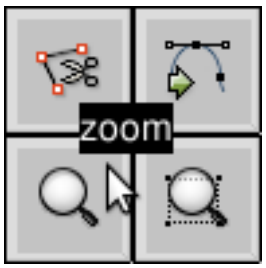

Figura A.31: Zoom in/out

A.4.3 Zoom di un area

Per effettuare lo zoom fare clic sul bottone *zoom area* (fig. A.32, selezionare

Figura A.32: Zoom area

sull'area di disegno due vertici opposti di un rettangolo che costituisce l'area da ingrandire. A questo punto l'area selezionata risulterà ingrandita.

A.4.4 Zoom default

La funzione *zoom default* riporta la curva alle sue dimensioni originali.

A.4.5 Scale in/out

La funzione *scale in/out* come suggerisce il nome serve a scalare le curve presenti nell'area di disegno; una volta selezionata la voce dal menu *View* fare clic col tasto sinistro del mouse per ingrandire, col tasto destro per rimpicciolire, l'immagine sullo schermo.

Bibliografia

[1] X Window System,
`http://it.wikipedia.org/wiki/X_Window_System`

[2] Simple DirectMedia Layer,
`http://www.libsdl.org/`

[3] G. Casciola, *Polinomi nella base di Bernestein e Curve di Bézier*, Dispense per il corso Metodi Numerici per Matematica e Informatica (2010).

[4] R.T. Farouki, *The Bernstein polynomial basis: a centennial retrospective* (2012).

[5] G.Farin, J.Hoschek, M.S.Kim, *Handbook of Computer Aided Geometric Design*, Elsevier (2002).