Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

GRAFICA VETTORIALE E VETTORIALIZZAZIONE

Relatore:
Chiar.mo Prof.
Giulio Casciola

Presentata da:
Davide Nanni

Sessione III
Anno Accademico 2017/2018

Alla mia famiglia

Introduzione

Le curve MD-spline, o multi degree spline, sono una generalizzazione delle B-spline composte da segmenti polinomiali di gradi differenti. Tali curve permettono di modificare il grado di ogni segmento in modo indipendente dagli altri e di conseguenza di rappresentare qualsiasi forma con il minimo grado necessario. Si è voluto sperimentare le MD-spline nell'ambito della grafica vettoriale, dove i formati esistenti presentano limitazioni sul grado o sulla continuità delle curve. Fino a tempi recenti non erano noti algoritmi di calcolo efficienti per MD-spline generiche, rendendone limitato l'impiego; tuttavia, oggigiorno sistemi di calcolo come Mini-System ne permettono la manipolazione alla massima generalità.

L'obiettivo di questo lavoro di tesi è stato l'estensione delle funzionalità del software Mini-System con un sistema di tracing di immagini raster, sia a colori che in scala di grigi, e con algoritmi di riempimento dei poligoni, in modo da migliorarne le potenzialità di utilizzo per disegni vettoriali. Il vantaggio fornito dalle MD-spline in tale ambito consiste sia nella gestione automatica della regolarità di un insieme di curve polinomiali, laddove altri software di grafica vettoriale si limitano alla minima continuità, sia nelle capacità di semplificazione e riduzione dell'espressione dei tratti, propedeutico al risparmio di memoria delle figure salvate.

La tesi è suddivisa in quattro capitoli:

1. Il primo capitolo introduce diversi concetti propri della grafica digitale e della modellazione geometrica. Sono inoltre presentati il forma-

to vettoriale SVG, il pacchetto Mini-System e il software di tracing Potrace.

2. Il secondo capitolo illustra in dettaglio l'algoritmo di Potrace.

3. Il terzo capitolo descrive l'integrazione delle nuove funzionalità in Mini-System. In particolar modo si dà spazio all'implementazione dell'interfaccia software per Potrace.

4. Il quarto capitolo è dedicato alla presentazione dei risultati di varie sperimentazioni sulle funzionalità aggiunte e sui relativi parametri, come dimostrazione dell'idoneità del pacchetto per la grafica vettoriale.

Indice

Introduzione **i**

1 Cenni introduttivi **1**

1.1 Grafica raster e vettoriale 1

1.2 Curve di Bézier 3

1.2.1 Polinomi in forma di Bernstein 4

1.2.2 Algoritmo di de Casteljau 6

1.2.3 Curve in forma parametrica 7

1.2.4 Curve di Bézier 8

1.3 Curve polinomiali a tratti 9

1.3.1 Curve di Bézier a tratti 9

1.3.2 Spline 10

1.3.3 MD-spline 11

1.4 Formato SVG 13

1.5 Mini-System 16

1.6 Potrace 18

2 Algoritmo di Potrace **21**

2.1 Decomposizione in bordi 21

2.1.1 Turn policy 23

2.1.2 Despeckling 24

2.2 Approssimazione con poligono ottimale 25

2.3 Descrizione vettoriale del contorno 27

2.4 Ottimizzazione delle curve 31

3 Integrazione in Mini-System **35**
3.1 Conversione bitmap 35
3.2 Conversione contorni 37
3.3 Gestione immagini 41
3.3.1 Conversione in scala di grigi 41
3.3.2 Thresholding 44
3.4 Gestione parametri 50
3.5 Riempimento dei poligoni 54

4 Risultati **57**
4.1 Thresholding 57
4.2 Tracing 65
4.2.1 Test parametri 65
4.2.2 Risultati del tracing 68
4.3 Comparazione pesi 73

Conclusioni **75**

Bibliografia **77**

Elenco delle figure

1.1 Comparazione tra un'immagine vettoriale e un'immagine raster 2
1.2 Esempi di polinomi base di Bernstein 5
1.3 Esempio di curva di Bézier 8
1.4 Esempio di immagine SVG 15
1.5 Interfaccia utente di Mini-System 17
1.6 Esempio di tracing eseguito da Inkscape 19

2.1 Possibili estensioni di un percorso 23
2.2 Calcolo dell'area di un percorso 24
2.3 Esempi di percorsi 26
2.4 Esempi di poligoni 27
2.5 Esempio di curve di Bézier a due parametri 29
2.6 Esempi di approssimazione dei vertici 30
2.7 Esempio di ottimizzazione di una sequenza di curve 32
2.8 Esempio completo dell'algoritmo di Potrace 33

3.1 Esempio di rappresentazione bitmap di Potrace 37
3.2 Esempio di decomposizione in path 38
3.3 Grafico delle trasformazioni di compressione e decompressione della gamma per sRGB 42
3.4 Diversi metodi di conversione in scala di grigi 43
3.5 Esempio di thresholding con i due metodi globali 46
3.6 Esempio di thresholding con i due metodi locali 48
3.7 Illustrazione di un passo di convoluzione 49

3.8 Esempio di separabilità della convoluzione con kernel gaussiano 51
3.9 Menu delle impostazioni per il tracing 52
3.10 Finestra di dialogo per i parametri di Potrace 52
3.11 Finestra di dialogo per i parametri dei metodi di thresholding 53
3.12 Passo del metodo scan conversion 55
3.13 Esempio di scan conversion 55

4.1 Esempio di thresholding su un'immagine in scala di grigi con illuminazione omogenea 58
4.2 Esempio di thresholding su un'immagine in scala di grigi con illuminazione non omogenea 59
4.3 Esempio di thresholding su un'immagine in scala di grigi con illuminazione non omogenea 60
4.4 Esempio di thresholding su un'immagine a colori con illuminazione parzialmente omogenea 61
4.5 Esempio di thresholding su un'immagine a colori con illuminazione non omogenea 62
4.6 Test sul parametro di soglia per il thresholding a valore fisso . 63
4.7 Test sui parametri per il thresholding locale 64
4.8 Test sul parametro turnpolicy 65
4.9 Test sul parametro turdsize 66
4.10 Test sul parametro alphamax 67
4.11 Test sul parametro opttolerance 67
4.12 Esempio di tracing di una vignetta 68
4.13 Esempio di tracing di una fotografia 69
4.14 Esempio di tracing di una fotografia 70
4.15 Esempio di tracing della scannerizzazione di un documento . . 71
4.16 Esempio di tracing di un disegno tecnico 72

Capitolo 1

Cenni introduttivi

1.1 Grafica raster e vettoriale

In grafica digitale esistono due tipologie di rappresentazione di immagini: **raster** e **vector**. I formati raster, come ad esempio JPEG, PNG e BMP, memorizzano l'immagine come una matrice di pixel e sovente impiegano tecniche di compressione con o senza perdita di informazione. I punti di forza delle immagini raster sono molteplici: si possono generare direttamente da un dispositivo di acquisizione, si possono modificare facilmente sia a livello dei singoli pixel che a scala maggiore, sono gestite ad alte prestazioni dalle schede video, in quanto l'accesso ai pixel richiede una minima computazione, la dimensione dei file relativi è generalmente contenuta. Il difetto principale della grafica raster consiste nell'impossibilità di essere rappresentata correttamente ad una risoluzione maggiore di quella di origine. In tal caso, infatti, si incorre nel fenomeno conosciuto come "effetto sgranato", siccome il software di visualizzazione è incaricato di adattare la matrice dei pixel a disposizione ad una matrice di dimensione maggiore tramite tecniche di riempimento, alle quali è sconosciuta l'informazione visiva del soggetto originale. Le immagini vettoriali sopperiscono a questo specifico problema. Difatti sono costituite dalle linee e curve che delimitano le aree di differente colore e se rappresentate a differente risoluzione vengono ridisegnate senza perdita di qualità. I

Figura 1.1: Comparazione della stessa immagine raffigurante un'automobile come immagine vettoriale (sinistra) e come immagine raster (destra)

formati vector, come ad esempio PS, PDF e SVG memorizzano l'immagine come una sequenza di linee e curve. La maggior parte delle immagini in questo stesso documento sono immagini vettoriali e possono essere ingrandite senza perdita di definizione. Si invita l'eventuale lettore che stia consultando il documento in formato digitale ad ingrandire la figura 1.1; si può notare come l'immagine vettoriale, a sinistra, mantenga sempre la stessa qualità, mentre l'immagine raster, a destra, si sgrani.

La visualizzazione di formati vettoriali, quali *SVG* e *PS*, avviene tramite la tecnica chiamata **rasterizzazione**, o **rastering**, ossia tracciando forme che approssimino i tratti vettoriali su di una superficie di dimensione fissa. Un software di visualizzazione di documenti PDF, ad esempio, effettua la valutazione dei tratti vettoriali in un numero di punti sufficiente a garantirne una buona rappresentazione alla risoluzione attuale ed applica algoritmi di filling per riempire le forme chiuse e algoritmi di anti-aliasing ai contorni delle forme per "ammorbidire" quelle sezioni che presentano un'evidente seghettatura. La qualità che si apprezza in un'immagine vettoriale non deriva perciò unicamente dall'immagine stessa, ma è resa altresì dai procedimenti di visualizzazione che i software attuano a partire da essa. Proprio in tale caratteristica risiede tuttavia il principale svantaggio delle immagini vettoriali, dacché i processi di rasterizzazione richiedono una considerevole mole

di calcoli. Per di più, tali computazioni devono essere eseguite *ex-novo* ad ogni modifica della **viewport**, ossia l'area visibile dell'immagine. Tornando all'esempio della figura 1.1, si può probabilmente notare come ingrandendo considerevolmente l'immagine vettoriale diventi via via meno fluido il passaggio da un livello di zoom al successivo.

Un altro svantaggio delle immagini vettoriali consiste nell'impossibilità di effettuare modifiche relativamente ai pixel; d'altra parte rendono facilmente attuabili mutamenti alle definizioni delle forme, che sono invece molto complessi per immagini raster. Si può quindi concludere come ciascuna tipologia di immagine digitale abbia dei relativi campi di applicazione, per i quali i vantaggi sono più rilevanti rispetto alle limitazioni.

Complementare alla rasterizzazione, la **vettorializzazione**, o **tracing**, effettua il processo di conversione da immagini raster a immagini vettoriali. Tale processo prevede la segmentazione dell'immagine in aree di colore omogeneo e l'approssimazione di tali aree con tratti vettoriali. Trattandosi di un'approssimazione, l'immagine vettoriale risultante presenta differenze tanto più visibili quanto più si osserva in dettaglio, tuttavia un buon algoritmo di tracing mantiene le forme generalmente simili all'originale. Logicamente, maggiore è la risoluzione dell'immagine originale e migliore ne sarà l'approssimazione vettoriale. Sovente, i tratti vettoriali in output da un software di tracing, così come la stessa descrizione delle curve per diversi formati vettoriali, sfruttano le **curve di Bézier** come descrizione matematica. In questo lavoro sfrutteremo le curve MD-spline.

1.2 Curve di Bézier

Una curva di Bézier è una curva parametrica definita utilizzando la base dei polinomi di Bernstein. Sebbene tali polinomi fossero conosciuti già dal 1912, la loro applicazione alla grafica risale a 50 anni più tardi. Il primo studio delle curve di Bézier si deve al matematico francese Paul de Casteljau, il quale ideò nel 1959 un metodo numericamente stabile per la valutazio-

ne dei polinomi in forma di Bernstein. Una loro prima applicazione alla modellazione grafica fu opera dell'ingegnere francese Pierre Bézier, il quale ne pubblicizzò nel 1962 il possibile impiego industriale utilizzando le curve omonime per la progettazione della carrozzeria delle automobili Renault. Le curve di Bézier sono oggi utilizzate in diversi formati di grafica vettoriale come PostScript e SVG e nei relativi software di disegno, nei font TrueType e OpenType, nei software di animazione per la descrizione dei movimenti e in molti altri ambiti.

1.2.1 Polinomi in forma di Bernstein

I **polinomi base di Bernstein** di grado n nell'intervallo $[a, b]$ sono così definiti:

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

e formano una base per lo spazio vettoriale di polinomi di grado massimo n.

Un **polinomio in forma di Bernstein** di grado n è definito come combinazione lineare dei polinomi base di Bernstein dello stesso grado, ovvero

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

dove $b_0, \ldots, b_n \in \mathbb{R}$ sono i coefficienti di Bernstein, anche detti **coefficienti di Bézier**.

Siccome i polinomi sono invarianti per traslazione e scala dell'intervallo di definizione o cambio di variabile, è possibile definire una applicazione da un polinomio $p(x)$ definito nell'intervallo $[a, b]$ ad un polinomio $q(t)$ definito in un intervallo traslato e scalato, come ad esempio $[0, 1]$:

$$x \in [a, b] \to t \in [0, 1] \qquad t = \frac{t-a}{b-a} \ .$$

Ciò risulta utile per ridurre l'errore inerente in fase di valutazione senza il rischio di commettere errori di approssimazione nel calcolo dei coefficienti del polinomio $q(t)$, in quanto una delle proprietà dei polinomi in forma di

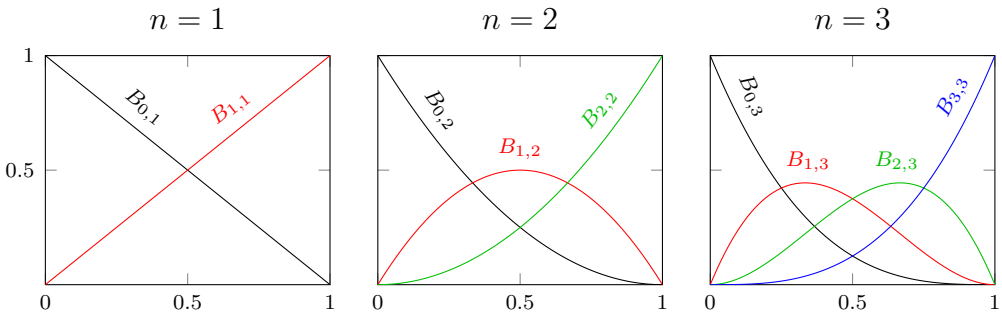


Figura 1.2: Esempi di polinomi base di Bernstein

Bernstein è il preservamento dei coefficienti nel cambio di variabile. Vale infatti

$$B_{i,n}(x) = \binom{n}{i} \left(\frac{b-x}{b-a}\right)^{n-i} \left(\frac{x-a}{b-a}\right)^{i} = \binom{n}{i}(1-t)^{n-i}t^{i} = B_{i,n}(t) \ .$$

La scelta dell'intervallo [0,1] non è casuale, difatti le curve di Bézier sono comunemente definite in tale intervallo. Di seguito assumeremo i polinomi base di Bernstein essere sempre definiti in [0,1]. Seguono alcuni esempi, rappresentati in figura 1.2:

Spazio $\mathbb{P}^1$

$B_{0,1}(t) = 1 - t \qquad B_{1,1}(t) = t$

Spazio $\mathbb{P}^2$

$B_{0,2}(t) = (1-t)^2 \quad B_{1,2}(t) = 2t(1-t) \quad B_{2,2}(t) = t^2$

Spazio $\mathbb{P}^3$

$B_{0,3}(t) = (1-t)^3 \quad B_{1,3}(t) = 3t(1-t)^2 \quad B_{2,3}(t) = 3t^2(1-t) \quad B_{3,3}(t) = t^3$

Alcune proprietà dei polinomi base di Bernstein sono le seguenti:

- Sono non negativi:

$$B_{i,n}(t) \geq 0 \qquad i = 0, \ldots, n \ ;$$

- Costituiscono una partizione dell'unità:

$$\sum_{i=0}^{n} B_{i,n}(t) = 1 \ ;$$

- Valgono:

$$B_{i,n}(0) = \delta_{i,0} \quad \text{e} \quad B_{i,n}(1) = \delta_{i,n} \qquad \text{con} \quad \delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & altrimenti \end{cases} \ ;$$

- $B_{i,n}(t)$ ha unico massimo, per $t = i/n$;

- $B_{i,n}(t)$ è simmetrico rispetto a $t = i/n$, ovvero

$$B_{i,n}(t) = B_{i,n}(1-t) \ ;$$

- Possono essere definiti in termini dei polinomi di base di grado inferiore, secondo la formula di ricorrenza

$$B_{i,n}(t) = tB_{i-1,n-1}(t) + (1-t)B_{i,n-1}(t) \ , \tag{1.1}$$

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

1.2.2 Algoritmo di de Casteljau

L'**Algoritmo di de Casteljau** è un metodo ricorsivo per la valutazione di polinomi in forma di Bernstein. Sebbene possa risultare meno performante dell'approccio diretto

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

guadagna in stabilità numerica. Il metodo deriva dall'applicazione ripetuta della formula ricorrente per i polinomi base di Bernstein sui coefficienti del

polinomio [1]:

$$\begin{aligned} p(t) = \sum_{i=0}^{n} b_i B_{i,n}(t) &\overset{(1.1)}{=} \sum_{i=0}^{n} b_i t B_{i-1,n-1}(t) + \sum_{i=0}^{n} b_i (1-t) B_{i,n-1}(t) \\ &= \sum_{i=0}^{n-1} b_{i+1} t B_{i,n-1}(t) + \sum_{i=0}^{n-1} b_i (1-t) B_{i,n-1}(t) \\ &= \sum_{i=0}^{n-1} [b_{i+1} t + b_i (1-t)] B_{i,n-1}(t) \\ &= \sum_{i=0}^{n-1} b_i^{[1]} B_{i,n-1}(t) \\ &= \dots \\ &= \sum_{i=0}^{0} b_0^{[n]} B_{0,0}(t) = b_0^{[n]} \ . \end{aligned}$$

La valutazione quindi calcola

$$\begin{aligned} p(t) &= b_0^{[n]} \\ \text{con} \quad b_i^{[k]} &= t b_{i+1}^{[k-1]} + (1-t) b_i^{[k-1]} \\ k &= 1, \dots, n \\ i &= 0, \dots, n-k \ . \end{aligned}$$

Notare che agli estremi il valore di un polinomio in forma di Bernstein assume il valore dei coefficienti b_0 e b_n, ovvero

$$p(0) = b_0 \quad \text{e} \quad p(1) = b_n \ .$$

1.2.3 Curve in forma parametrica

Una curva parametrica in $\mathbb{R}^2$ è definita dalla funzione vettoriale $C(t) : [a,b] \to \mathbb{R}^2$ e si esprime come

$$C(t) = \begin{pmatrix} C_x(t) \\ C_y(t) \end{pmatrix} \ ,$$

con $t \in [a,b] \subset \mathbb{R}$ e $C_x(t), C_y(t) : [a,b] \to \mathbb{R}$. Tale funzione associa ad ogni valore del parametro t un punto del piano di coordinate $(C_x(t), C_y(t))$ funzioni dello stesso t tali che il punto appartiene alla curva, ovvero delinea la curva sul piano al variare di t.

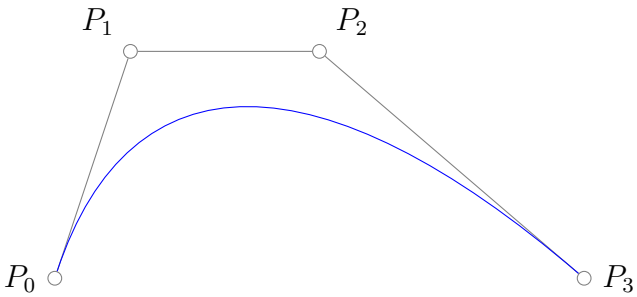


Figura 1.3: Esempio di curva di Bézier cubica

1.2.4 Curve di Bézier

Una **curva di Bézier** $C(t)$ di grado n in $\mathbb{R}^2$ è definita in funzione di un insieme di punti $P_i = (x_i, y_i) \in \mathbb{E}^2$ con $i = 0, \ldots, n$ tramite la formula

$$C(t) = \sum_{i=0}^{n} P_i B_{i,n}(t) = \begin{pmatrix} C_x(t) \\ C_y(t) \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n} x_i B_{i,n}(t) \\ \sum_{i=0}^{n} y_i B_{i,n}(t) \end{pmatrix} \quad t \in [0,1] \ .$$

Gli $n+1$ punti P_i sono chiamati **punti di controllo** e se connessi in ordine da segmenti descrivono il **poligono di controllo**. $C_x(t)$ e $C_y(t)$ sono polinomi in forma di Bernstein. Alcune proprietà delle curve di Bézier sono le seguenti:

- Una curva di grado n è anche una curva di grado m, per qualsiasi $m > n$;
- I punti di controllo P_0 e P_n sono estremi della curva; i punti intermedi non fanno generalmente parte della curva (vedi figura 1.3);
- Una curva degenera in un segmento se e solo se tutti i punti di controllo sono collineari;
- Gli estremi iniziale e finale della curva sono tangenti alla prima e ultima sezione, rispettivamente, del poligono di controllo;
- Una curva può essere suddivisa in punti arbitrari definendo molteplici sottocurve, ognuna delle quali è una curva di Bézier;

- Qualsiasi retta nel piano ha un numero di intersezioni con una curva minore o uguale al numero di intersezioni della stessa retta con il poligono di controllo (**Approssimazione VD**).

1.3 Curve polinomiali a tratti

Sovente le curve di Bézier finora viste risultano inadeguate per la descrizione di forme geometriche complesse, in quanto curve i cui polinomi sono definiti nell'intero intervallo di definizione della curva. Siccome l'interpolazione di n punti richiede una curva di egual ordine, forme complesse sono rappresentate tramite curve di grado elevato, numericamente instabili e di difficile gestione. Inoltre, ciascun punto di controllo condiziona la curva globalmente: una modifica ad un singolo punto influenza la forma dell'intera curva. Per ovviare a suddetti problemi si fa utilizzo delle curve polinomiali a tratti, che sfruttano la suddivisione dell'intervallo di definizione in sottointervalli in modo da poter utilizzare polinomi di grado relativamente basso. Di seguito si introducono brevemente tre tipi di curve polinomiali a tratti: le curve di Bézier a tratti, le spline e le MD-spline.

1.3.1 Curve di Bézier a tratti

Una **curva di Bézier a tratti** è costituita da una sequenza di curve di Bézier, ognuna delle quali è definita in una sezione dell'intervallo canonico. Sia dato un intervallo $I = [a, b]$ e una partizione Δ di detto intervallo così definita:

$$\Delta = \{x_i\}_{i=0}^{q+1} = \{x_i \in [a, b] \mid a = x_0 < x_1 < \ldots < x_{q+1} = b\}$$

Si hanno $q + 1$ sottointervalli $I_i = [x_i, x_{i+1}]$ con $i = 0, \ldots, q$ e altrettante curve $C_i(x)$ di grado n definite in un sottointervallo, le quali costituiscono la curva a tratti $C(x)$:

$$C(x) = C_i(x) \quad \text{con } x \in I_i \ \forall \ i = 0, \ldots, q \ .$$

Riprendendo la parametrizzazione in $[0,1]$, per $x_i \in I_i$ si avrà il corrispondente $t \in [0,1]$ secondo la formula

$$t = \frac{x - x_i}{x_{i+1} - x_i} = \frac{x - x_i}{h_i} \ ,$$

con h_i ampiezza dell'intervallo i-esimo. Riguardo alla continuità nei punti di raccordo, si dice che $C(x)$ è continua C^μ nel punto x_i se

$$C_i^{(j)}(x_i) = C_{i+1}^{(j)}(x_i) \quad \text{con } 0 \leq j \leq \mu,\ \mu \leq n-1 \ ,$$

dove $C_i^{(j)}$ è la derivata j-esima di C_i .

1.3.2 Spline

Sebbene le curve di Bézier a tratti risolvano il problema di globalità delle curve di Bézier, limitando l'influenza dei punti di controllo localmente al tratto di appartenenza, ne rimane difficoltosa la gestione esplicita dei vincoli di continuità. Si rende perciò necessario l'impiego delle **spline**, funzioni polinomiali a tratti definite in modo tale da gestire implicitamente la continuità nei punti di raccordo.

Una spline di grado n definita nell'intervallo $[a,b]$ è identificata da una combinazione lineare di funzioni base dette **B-spline**. Siano $\Delta = \{x_i\}_{i=0}^{q+1}$ il **vettore dei nodi** costituito da una partizione dell'intervallo tale che

$$a = x_0 < x_1 < \ldots < x_{q+1} = b \ ,$$

$M = \{\mu_i\}_{i=1}^{q}$ il **vettore delle continuità**, con $0 \leq \mu_i \leq n-1$ e $\{P_i = (x_i, y_i)\}_{i=0}^{K}$ un insieme di punti di controllo in $\mathbb{R}^2$, con $K = \sum_{i=1}^{q}(n - \mu_i) + n+1$. Lo spazio vettoriale delle spline di grado n è indicato come $S(\mathbb{P}_n, M, \Delta)$ ed è costituito da tutte le spline $s(x)$ di grado n definite come combinazione lineare delle basi B-spline utilizzando i punti di controllo come coefficienti. La definizione di una curva spline di grado n in forma parametrica è quindi la seguente:

$$s(t) = \sum_{i=1}^{K} P_i N_{i,n}(t) = \begin{pmatrix} s_x(t) \\ s_y(t) \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{K} x_i N_{i,n}(t) \\ \sum_{i=1}^{K} y_i N_{i,n}(t) \end{pmatrix} \quad t \in [0,1] \ ,$$

dove $N_{i,n}(t)$ è la base B-spline definita ricorsivamente come

$$N_{i,0}(t) = \begin{cases} 1 & t_i \leq t < t_{i+1} \wedge \ t_i < t_i + 1 \\ 0 & \text{altrimenti} \end{cases}$$
$$N_{i,j}(t) = \frac{t - t_i}{t_{i+j} - t_i} N_{i,j-1}(t) + \frac{t_{i+j+1} - t}{t_{i+j+1} - t_{i+1}} N_{i+1,j-1}(t) \ ,$$

con $j = 1, \ldots, n$. Notare che tali basi sono definite su una partizione nodale estesa

$$\mathcal{T} = \{t_i\}_{i=1}^{2(n+1)+K} \ ;$$

si consulti [2] per una trattazione approfondita.

In base all'omogeneità del vettore delle continuità si configurano due tipi di spline: se la continuità nei nodi è ovunque massima, si parla di spline a *nodi semplici*, altrimenti se le continuità nei nodi sono differenti si parla di spline a *nodi multipli*.

1.3.3 MD-spline

Come abbiamo visto, le tradizionali curve spline sono costituite da segmenti polinomiali di egual grado. Negli scenari in cui sia necessario l'utilizzo di un grado elevato per descriverne anche solo un singolo tratto, si è vincolati ad esprimere tutti i tratti della curva allo stesso grado. Ciò introduce una quantità superflua di punti di controllo e ne diminuisce la praticità in fase di modellazione. Le curve MD-spline, generalizzazione delle spline, sovvengono a tale problema in quanto caratterizzate da una base di funzioni polinomiali a tratti di gradi differenti e quindi costituite da tratti polinomiali di gradi differenti. Segue una descrizione sommaria della definizione matematica delle curve MD-spline, condotta per analogie e differenze con le spline; se ne può trovare una trattazione esauriente in [3].

Spazio polinomiale a tratti

Lo spazio polinomiale a tratti multi-degree generalizza lo spazio polinomiale a tratti delle spline tramite l'introduzione del vettore dei gradi $\mathcal{N} =$

$\{n_i\}_{i=0}^{q}$, dove n_i è il grado del polinomio definito nell'intervallo $I_i = [x_i, x_{i+1}]$ con $i = 0, \ldots, q$. In pratica, si passa dallo spazio $\mathbb{P}_n$ a grado fisso allo spazio $\mathbb{P}_{\mathcal{N}}$ a grado variabile. Lo spazio delle spline multi-degree $S(\mathbb{P}_{\mathcal{N}}, M, \Delta)$ ha dimensione $K = n_q + \sum_{i=1}^{q}(n_{i-1} - \mu_i) + 1$.

Funzioni B-spline multi-degree

Le funzioni base **B-spline multi-degree** $\{N_i\}_{i=1}^{K}$ sono definite in relazione a due partizioni estese $\mathcal{T} = \{t_j\}_{j=1}^{K}$ e $\mathcal{S} = \{s_j\}_{j=1}^{K}$ tali che ogni nodo x_i sia ripetuto $(n_i - \mu_i)$ volte in $\mathcal{T}$ e $(n_{i-1} - \mu_i)$ in $\mathcal{S}$. Sia m il massimo dei gradi in $\mathcal{N}$. La base spline multi-degree di $S(\mathbb{P}_{\mathcal{N}}, M, \Delta)$ è l'insieme $\{N_{i,m}(x)\}$ per $i = 1, \ldots, K$ con ogni $N_{i,n}$ definita su ogni intervallo $[x_j, x_{j+1}] \subset [t_i, s_{i-m+n}]$ tramite una relazione di ricorrenza integrale. Recentemente, in letteratura, sono stati proposti degli algoritmi efficienti per il loro calcolo [4], [5].

Funzioni MD-spline

Le **funzioni MD-spline** dello spazio $S(\mathbb{P}_{\mathcal{N}}, M, \Delta)$ sono, come le spline, costituite da una combinazione lineare delle funzioni base, ma utilizzano le B-spline multi-degree al posto delle B-spline:

$$f(x) = \sum_{i=1}^{K} c_i N_{i,m}(x) \ ,$$

con $x \in [a, b]$ e dove $c_i \in \mathbb{R}$ sono i coefficienti scalari.

Curve MD-spline

Siano $P_i = (x_i, y_i)$ i punti di controllo in $\mathbb{R}^2$ con $i = 1, \ldots, K$. Una **curva MD-spline** è una curva $C(t) \in \mathbb{R}^2$ in forma parametrica le cui componenti $C_1(t)$ e $C_2(t)$ sono funzioni MD-spline:

$$C(t) = \sum_{i=1}^{K} P_i N_i(t) = \begin{pmatrix} C_1(t) \\ C_2(t) \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{K} x_i N_i(t) \\ \sum_{i=1}^{K} y_i N_i(t) \end{pmatrix} \quad t \in [0, 1] \ .$$

1.4 Formato SVG

Scalable Vector Graphics (SVG) è un linguaggio di markup per la descrizione di grafica vettoriale bidimensionale, basato su XML e standardizzato dal World Wide Web Consortium (W3C) dal 1999. Di seguito ne si descrivono la struttura e gli elementi di base [6], tralasciando le caratteristiche più avanzate o recenti, come filtri, animazioni ed integrazione con JavaScript.

L'elemento radice `<svg>` definisce il sistema di coordinate e le dimensioni dell'immagine e della **viewport**, ossia l'area visibile del piano. Di default il sistema di coordinate ha l'origine nell'angolo superiore sinistro e l'asse delle ordinate orientato verso il basso. Se non viene specificata alcuna trasformazione di corrispondenza con unità di misura reali, come *pt* o *cm*, l'unità del sistema di coordinate è associata ad un pixel del dispositivo su cui avviene la visualizzazione.

All'interno dell'elemento radice sono contenuti gli elementi descriventi la sagoma e lo stile di forme semplici e complesse. In aggiunta alle forme basilari, quali *rettangolo*, *cerchio*, *ellisse*, *linea*, *polilinea* e *poligono*, è infatti disponibile l'elemento `<path>` , il cui attributo `d` contiene una sequenza di comandi di disegno. Tutti i comandi sono identificati con una lettera e sono disponibili in due varianti: la lettera maiuscola indica l'interpretazione delle coordinate come coordinate assolute, mentre la lettera minuscola come coordinate relative all'ultimo punto. I principali comandi sono:

- `M x y` / `m dx dy` : muove il cursore di disegno alle coordinate specificate, senza tracciare alcuna linea;
- `L x y` / `l dx dy` : traccia un segmento dalla posizione attuale del cursore fino alle coordinate specificate;
- `H x` / `h dx` o `V y` / `v dy` : traccia un segmento orizzontale o verticale, rispettivamente;

- `Z` / `z` : traccia un segmento dalla posizione attuale del cursore alla prima posizione del path, in modo da chiudere la linea;
- `C x1 y1, x2 y2, x y` / `c dx1 dy1, dx2 dy2, dx dy` : traccia una curva di Bézier cubica, utilizzando la posizione attuale come primo punto di controllo e le coordinate specificate per gli altri tre;
- `S x2 y2, x y` / `s dx2 dy2, dx dy` : come `C` , ma se preceduto da `C` o `S` assegna al primo punto di controllo la coordinata del penultimo punto della curva precedente specchiata rispetto all'ultimo punto della stessa, in modo da avere continuità C^1 nel punto di raccordo, altrimenti assegna la posizione attuale del cursore;
- `Q x1 y1, x y` / `q dx1 dy1, dx dy` : traccia una curva di Bézier quadratica, utilizzando la posizione attuale come primo punto di controllo e le coordinate specificate per gli altri due;
- `T x y` / `t dx dy` : l'equivalente di `S` per `Q` . Se non è preceduto da `Q` o `T` , disegna un segmento dalla posizione attuale del cursore alla coordinata specificata.

Tutti gli elementi di disegno supportano attributi per la definizione dello stile della forma, che comprende il colore e lo spessore delle linee e il riempimento delle forme con tinta unita. Sono inoltre disponibili elementi specifici per definire gradienti lineari, gradienti radiali, pattern e trasformazioni quali traslazione, rotazione, inclinazione e ridimensionamento. Lo standard supporta il rendering dei font, permettendo la visualizzazione di testi lineari o conformi ad un path, e l'integrazione con CSS.

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
    viewBox="0 0 166.902 167">
<path fill="#1793D1" d="M83.81,0.625c-7.397,18.134-11.858,29.997
    -20.093,47.593c5.049,5.352,11.246,11.584,21.311,18.624
    c-10.82-4.453-18.2-8.924-23.717-13.562C50.772,75.272,34.259,
    106.6,0.75,166.81c26.338-15.2,46.753-24.58,65.78-28.15
    c-0.817-3.52-1.281-7.32-1.25-11.28l0.031-0.85c0.418-16.87,
    9.195-29.845,19.593-28.964s18.478,15.284,18.058,32.154
    c-0.07,3.18-0.43,6.23-1.06,9.06c18.82,3.68,39.02,13.03,65,
    28.03c-5.12-9.43-9.7-17.93-14.06-26.03
    c-6.88-5.33-14.05-12.27-28.69-19.78c10.06,2.62,17.26,5.63,
    22.88,9c-44.4-82.668-47.993-93.651-63.223-129.37L83.81,0.625
    L83.81,0.625z"/>
</svg>
```

Figura 1.4: Codice e rappresentazione di un'immagine SVG raffigurante il logo di Arch Linux. Notare l'utilizzo delle curve di Bézier cubiche.

1.5 Mini-System

Mini-System è un software di disegno vettoriale con supporto a curve di Bézier e MD-spline, scritto in linguaggio C e basato sulle librerie SDL per la gestione di input e superfici.

L'interfaccia utente è composta da quattro sezioni:

- una barra di menu a tendina in alto, contenenti tutte le funzionalità disponibili, quali importazione e salvataggio per i tipi di file supportati, creazione, modifica e trasformazione di curve e opzioni di visualizzazione, valutazione, interpolazione e tracing;
- una barra dei comandi veloci a sinistra, che ospita le opzioni più utilizzate;
- un'area informativa in basso, nella quale sono visualizzate diverse informazioni sulla curva selezionata o in fase di creazione;
- l'area di disegno, nella quale sono visualizzate le curve e sulla quale si operano tutte le operazioni di disegno tramite click del mouse.

Vediamo in dettaglio le funzionalità principali. Mini-System permette la creazione di curve di Bézier a tratti di grado arbitrario e ne gestisce automaticamente la continuità nei punti di raccordo. È possibile scegliere tra le continuità C^0, C^1 e G^1 nelle impostazioni. Inoltre, è possibile definire curve spline e modificare successivamente grado e continuità dei punti di raccordo agendo direttamente sulla rappresentazione della curva. La valutazione delle curve può essere effettuata per mezzo dei polinomi di Bernstein o tramite l'algoritmo di de Casteljau, descritto nella sezione 1.2.2. Oltre all'inserimento di nuove curve è possibile modificare i punti di controllo di curve già definite, effettuarne la suddivisione in due parti o l'intersezione con altre curve, applicare trasformazioni geometriche di rotazione, traslazione e scala e calcolarne lunghezza e area. Le opzioni di visualizzazione comprendono la gestione dello zoom e del rapporto d'aspetto, la visualizzazione adattiva, la

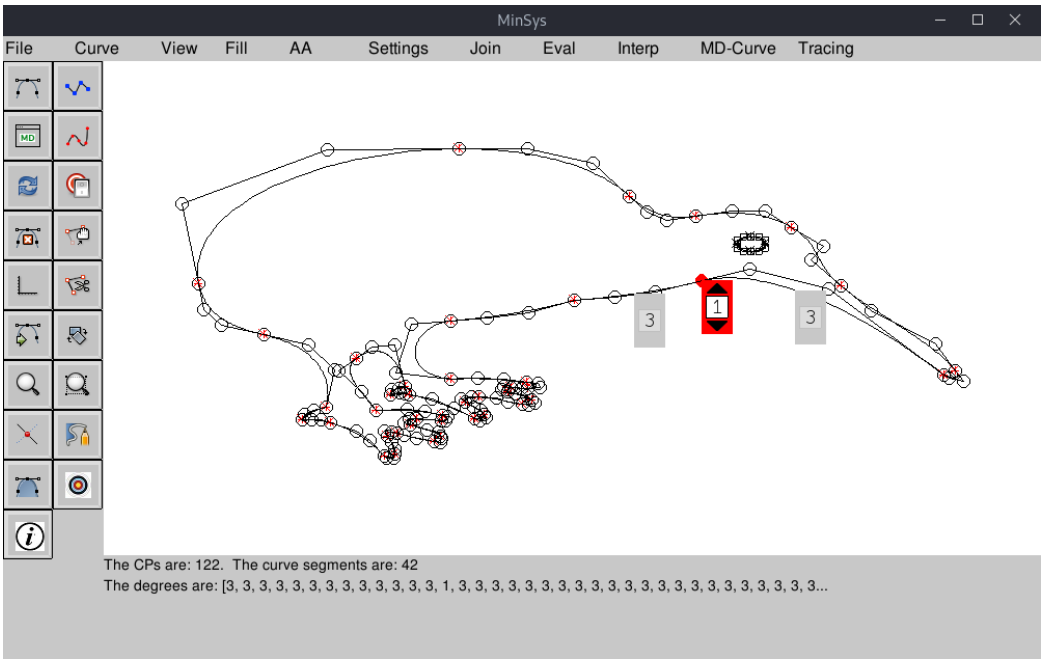


Figura 1.5: Interfaccia utente di Mini-System in fase di modifica della continuità in un punto di raccordo

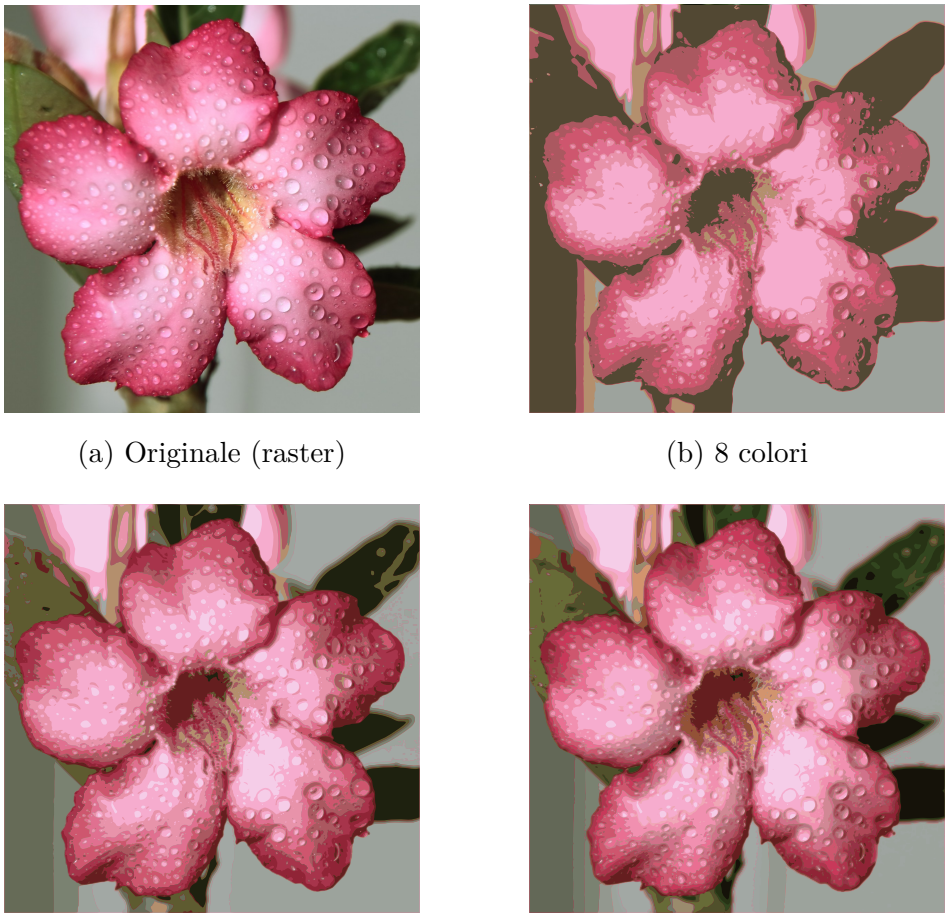
tracciatura di vettori tangenti, normali e di curvatura e diversi algoritmi di anti-aliasing e filling delle curve. In particolare, la visualizzazione adattiva agisce sul metodo di disegno delle curve: mentre normalmente ogni singola curva è disegnata come una successione di N segmenti, con N fissato e indipendente dalla visualizzazione attuale, la metodologia adattiva effettua una suddivisione ricorsiva in due sottocurve fino a che i nuovi tratti sono assimilabili a segmenti rispetto ad un valore di soglia. Si utilizzano quindi questi tratti per il disegno della curva. La differenza si può notare tracciando una curva con curvatura marcata e impostando un livello di zoom elevato: con la visualizzazione normale sono visibili i tratti di suddivisione, mentre con quella adattiva la curva appare perfetta al pixel. Si segnala infine l'implementazione di funzioni per la definizione e modellazione di curve MD-spline, assieme a funzioni per la conversione tra i due tipi di curve.

1.6 Potrace

Potrace è un software di tracing open-source, scritto e mantenuto dal matematico Peter Selinger a partire dal 2001 e distribuito con doppia licenza GPL e commerciale. Tale strumento effettua il tracing di un'immagine bitmap monocroma data in input, fornendone in output la descrizione vettoriale dei contorni. Il nome stesso è un *portmanteau* di "polygon tracer" ed indica l'utilizzo di una rappresentazione intermedia della bitmap basata su poligoni approssimanti. L'utilizzo tipico prevede la creazione di file SVG o PDF da scansioni di documenti, loghi e manoscritti. Sono inoltre supportati altri formati di output, quali EPS, PostScript, DXF, GeoJSON, PGM, Gimppath, e XFig.

Potrace è impiegato come motore di tracing da diversi software liberi e commerciali, come ad esempio l'editor di grafica vettoriale Inkscape, il software per la notazione musicale LilyPond e il software di composizione tipografica FontForge. In particolare, Inkscape è in grado di gestire anche immagini a colori tramite tecniche di quantizzazione. La figura 1.6 mostra un esempio di tracing eseguito da Inkscape a diversi gradi di quantizzazione.

La scelta dell'utilizzo di Potrace come motore di tracing di Mini-System risiede sia negli ottimi risultati da esso ottenuti sia nella presenza di un'esauriente documentazione. L'algoritmo di Potrace è descritto in dettaglio nel prossimo capitolo.



(a) Originale (raster)

(b) 8 colori

(c) 16 colori

(d) 32 colori

Figura 1.6: Esempio di tracing eseguito da Inkscape a diversi gradi di quantizzazione

Capitolo 2

Algoritmo di Potrace

L'idea generale dell'algoritmo di Potrace consiste nella conversione di una bitmap monocroma dapprima in una rappresentazione intermedia, costituita da un'insieme di poligoni che ne approssimino i contorni e successivamente nella descrizione vettoriale di tali poligoni [7]. È suddiviso in 4 fasi:

1. Decomposizione della bitmap in un insieme di bordi;
2. Approssimazione dei bordi con poligoni ottimali;
3. Approssimazione dei contorni dei poligoni con curve ed angoli;
4. Ottimizzazione delle curve.

Andremo di seguito a descrivere ciascuna fase in maggiore dettaglio.

2.1 Decomposizione in bordi

Si consideri la bitmap di dimensione $w \times h$ descritta in un sistema di riferimento cartesiano con origine nell'angolo inferiore sinistro e tale che i punti di coordinate intere corrispondano agli angoli dei pixel, di dimensione unitaria. Ogni punto in tale sistema è quindi adiacente a 4 pixel e i quattro angoli della bitmap hanno coordinate $(0, 0)$, $(0, h)$, (w, h) e $(w, 0)$. Inoltre, assumiamo che i pixel costituenti lo sfondo siano colorati di bianco, mentre

quelli in primo piano di nero. Per convenzione, la parte di piano esterna ai bordi della bitmap si assume costituita da pixel bianchi.

Definiamo **vertice** un punto per cui i pixel ad esso adiacenti non sono tutti dello stesso colore; inoltre, diciamo che esiste un **lato** tra due vertici se la loro distanza euclidea è 1 e se il segmento congiungente i due vertici divide due pixel di colore diverso. Un lato è direzionato in modo da avere sempre un pixel nero alla sua sinistra e un pixel bianco alla sua destra. Chiamiamo G il grafo diretto costruito a partire dalla bitmap con queste definizioni di vertice e lato.

Un **percorso** è una sequenza di vertici $\{v_0, \ldots, v_n\}$ tali che esiste un lato da v_i a $v_{i+1} \quad \forall\, i = 0, \ldots, n-1$ e tali che tutti i lati sono distinti. Un percorso si dice **chiuso** se $v_n = v_0$.

Il processo di decomposizione della bitmap in bordi consiste nella definizione di un insieme di percorsi chiusi su G nel quale ogni lato di G appare esattamente una volta. Tale risultato si ottiene tramite l'esecuzione della seguente procedura ricorsiva:

1. Si crea una copia della bitmap in input;

2. Si sceglie una coppia di pixel di colore diverso, ad esempio scandendo una riga da sinistra verso destra fino ad incontrare il primo pixel nero;

3. Si crea un lato con direzione appropriata e si considera come percorso di lunghezza 1;

4. Si estende il percorso con ulteriori lati contigui finché non si ottiene un percorso chiuso;

5. Si inverte il colore dei pixel interni al percorso e si effettua la chiamata ricorsiva sulla nuova bitmap così definita.

La scelta di nuove coppie si arresta quando non sono più presenti pixel neri nella bitmap in input all'attuale passo di ricorsione. Notare che la direzione

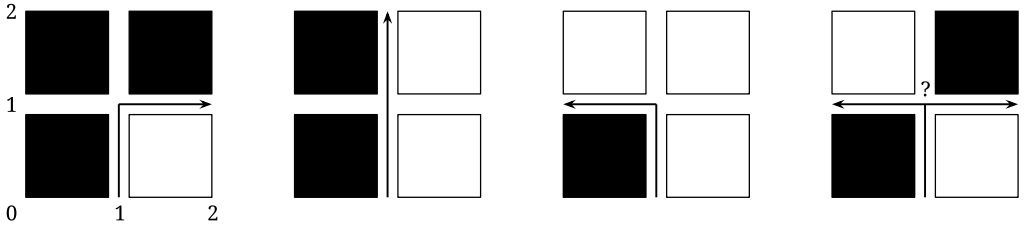


Figura 2.1: Possibili estensioni di un percorso

dei lati creati fa sempre riferimento al colore dei pixel nella bitmap originale. I percorsi definiti in questa fase sono trattati singolarmente nelle fasi successive.

2.1.1 Turn policy

In fase di estensione del percorso sono possono presentarsi situazioni di ambiguità, nelle quali la scelta del lato successivo è effettuabile tra due possibili lati di direzioni opposte, nominalmente **left** e **right**. In scenari come quello rappresentato dall'ultimo caso nella figura 2.1 è infatti lecito estendere il percorso in una o l'altra direzione, senza effetti di sorta sulla riuscita esecuzione dell'algoritmo. Ovviamente la scelta avrà invece influenza sulla forma dei percorsi chiusi.

In Potrace tale scelta è subordinata al valore di un'impostazione, detta **turn policy**, il cui dominio è costituito dalle seguenti politiche:

- **left/right** : si effettua sempre la scelta sinistra/destra;
- **black/white** : si effettua la scelta che connette le regioni nere/bianche (riferendosi ai colori della bitmap originale);
- **minority/majority** : si effettua la scelta che connette le regioni del colore che appare con meno/più frequenza in un intorno del punto;
- **random** : si effettua una scelta pseudo-randomica.

La politica di default è **minority**.

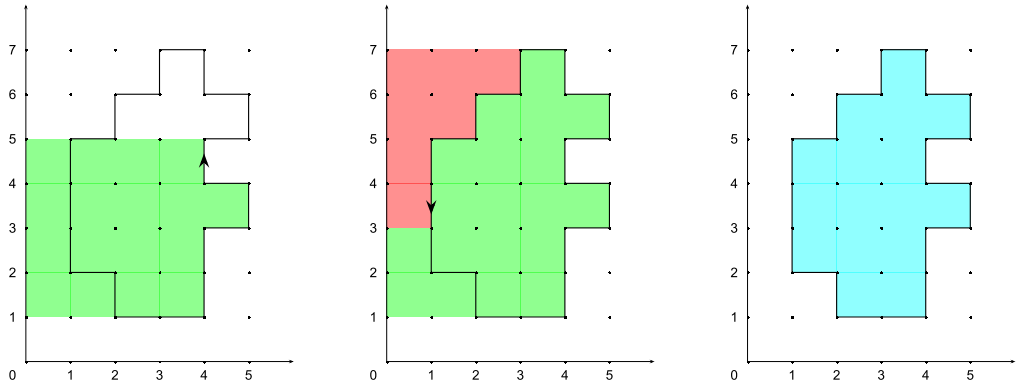


Figura 2.2: Calcolo dell'area di un percorso

2.1.2 Despeckling

Prima di passare alla fase successiva, vengono rimossi dall'insieme tutti i percorsi la cui area è minore di un determinato valore soglia. Tale procedimento, detto **despeckling**, agisce come filtro per la riduzione del rumore. L'area di un percorso p è calcolata durante la sua estensione, sommando ad ogni spostamento il prodotto tra l'ascissa del punto e l'ordinata della direzione relativa, ovvero calcolando

$$Area = \sum_{i=0}^{n-1} x_i(y_{i+1} - y_i) \ ,$$

con $p = \{v_0, \ldots, v_n\}$ e $v_i = (x_i, y_i)$. L'insieme degli spostamenti verso l'alto risulta nell'addizione dell'area a sinistra del bordo destro del percorso, mentre l'insieme degli spostamenti verso il basso risulta nella sottrazione dell'area a sinistra del bordo sinistro, come illustrato in figura 2.2. Gli spostamenti laterali non influiscono sul risultato. Siccome ogni percorso definito è chiuso, il valore finale intero corrisponde all'area dello spazio di piano interno al percorso. Il valore di soglia per l'area può essere modificato tramite il parametro intero positivo `turdsize` .

2.2 Approssimazione con poligono ottimale

La seconda fase dell'algoritmo di Potrace si occupa di approssimare un percorso chiuso con un poligono ottimale. Si descrive di seguito il processo di individuazione dei possibili lati del poligono e si introduce la definizione di ottimalità di un determinato insieme di lati.

Dati due punti $a = (x_0, y_0)$ e $b = (x_1, y_1)$, si definisce **massima distanza** $d(a, b) = \max(|x_1 - x_0|, |y_1 - y_0|)$ e si denota con $\overline{ab}$ il segmento congiungente i due punti.

Dato un percorso non chiuso $p = \{v_0, \dots, v_n\}$ ne definiamo la *direzione* di indice i per $i = 0, \dots, n-1$ come $v_{i+1} - v_i$, ovvero la coppia delle differenze delle componenti $(x_{i+1} - x_i,\ y_{i+1} - y_i)$. Diciamo inoltre che p è **approssimato** dal segmento $\overline{ab}$ se

$$\begin{cases} d(v_0, a) \leq 1/2 \\ d(v_n, b) \leq 1/2 \\ \forall\ i = 1, \dots, n-1 \quad \exists\ c_i \in \overline{ab} \ \mid\ d(c_i, v_i) \leq 1/2 \end{cases}$$

e che è **retto** se è approssimato da un segmento e al contempo non contiene tutte e quattro le direzioni. Possiamo osservare che un percorso retto contiene solo sottopercorsi retti e che un percorso non retto è contenuto solo da sovrapercorsi non retti. L'immagine 2.3 presenta alcuni esempi di possibili percorsi, di cui solo (a) e (c) sono retti. In particolare, (d) non è retto poiché contiene tutte e quattro le direzioni. Notare che i punti corrispondono ai vertici e quindi agli angoli dei pixel della bitmap, mentre i quadrati ne rappresentano un intorno di massima distanza 1/2. Per verificare che un percorso sia retto ci si serve della **proprietà della tripla**: un percorso $p = \{v_0, \dots, v_n\}$ che non contiene tutte e quattro le direzioni è retto $\iff \forall (i, j, k) \quad 0 \leq i < j < k \leq n \quad \exists\ w \in \overline{v_i v_k} \ \mid\ d(w, v_j) \leq 1$. L'implementazione in Potrace permette di ricavare tutti i sottopercorsi retti di un percorso chiuso di lunghezza n in tempo $\mathcal{O}(n^2)$.

Passiamo ora a descrivere i segmenti costituenti i poligoni. Dato un percorso chiuso $p = \{v_0, \dots, v_n\}$ definiamo **differenza ciclica** di due indici i, j

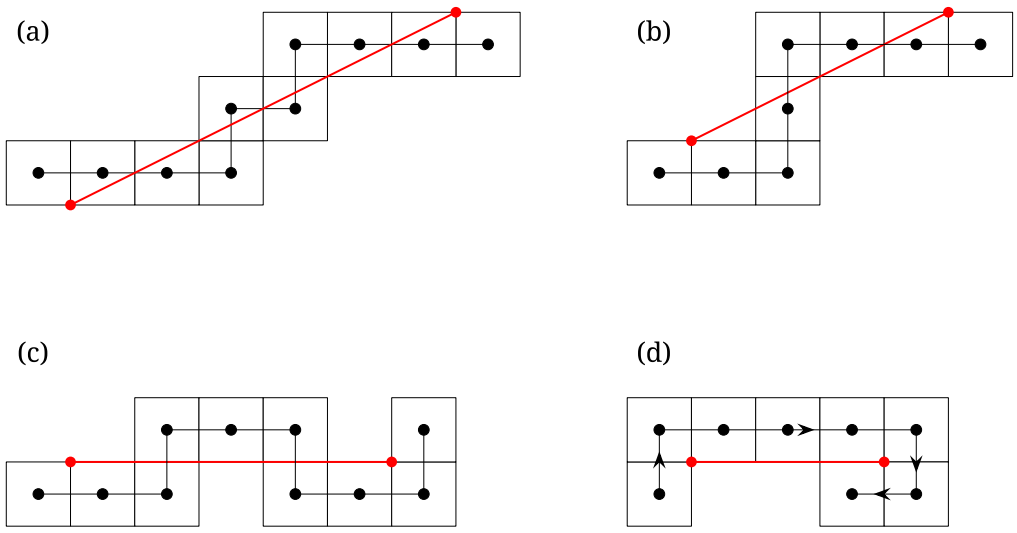


Figura 2.3: Esempi di percorsi

sul percorso come

$$i \ominus j = \begin{cases} j - i & i \leq j \\ j - i + n & \text{altrimenti} \end{cases}.$$

Diciamo che esiste un **possibile segmento** da i a j se $i \ominus j \leq n - 3$ e il sottopercorso $p_{i-1,j+1}$ è retto. Tale segmento può quindi essere esteso di un punto in entrambe le direzioni mantenendone la rettilineità. Siccome qualsiasi percorso di lunghezza 3 è retto, esiste sempre un possibile segmento da i a $i + 1$.

Un **poligono** è una sequenza di m indici $i_0 \leq \ldots \leq i_{m-1}$ su un percorso chiuso tali che esiste un possibile segmento da i_k a $i_{k+1 \bmod m}$ per $k = 0, \ldots, m - 1$. Di tutti i possibili poligoni per lo stesso percorso se ne vuole trovare l'ottimale. Il primo criterio è il numero di segmenti: tra due diversi poligoni si preferisce quello costituito dal minor numero di segmenti. Nel caso abbiano lo stesso numero di segmenti, il secondo criterio si basa sul calcolo di una **penalità**. Ad ogni possibile segmento da i a j si associa il segmento retto $\overline{v_i v_j}$ e si calcola la penalità $P_{i,j}$ come il prodotto tra la lunghezza euclidea di $\overline{v_i v_j}$ e la deviazione standard della distanza euclidea tra i

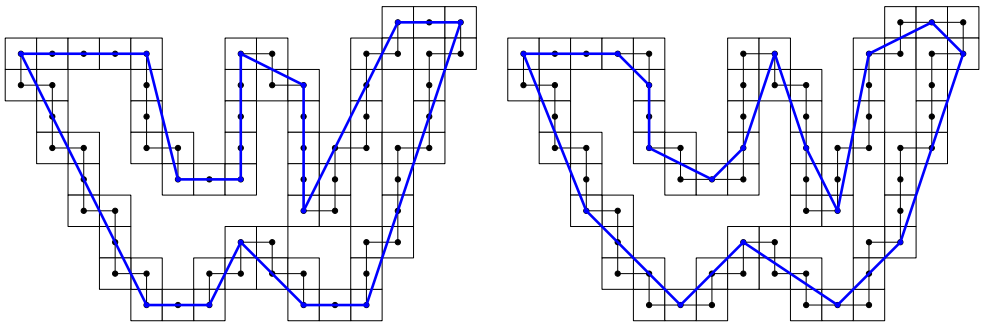


Figura 2.4: Il poligono ottimale e uno non ottimale per lo stesso percorso

punti del percorso e $\overline{v_i v_j}$, ovvero

$$P_{i,j} = |v_j - v_i| \cdot \sqrt{\frac{\sum_{k=i}^{j} \mathrm{dist}(v_k, \overline{v_i v_j})^2}{j \ominus i + 1}} \ ,$$

dove $\mathrm{dist}(p, \overline{ab})$ è la distanza euclidea di un punto p dalla linea descritta da a e b e si assume che la sommatoria sia circolare, ossia che calcoli $k \mod n$, con n lunghezza del percorso chiuso.

Si può ridurre la ricerca del poligono ottimale al problema di ricerca del cammino minimo su un grafo orientato. Consideriamo il percorso chiuso $p = \{v_0, \ldots, v_n\}$ come un grafo diretto, con vertici $\{v_0, \ldots, v_n - 1\}$ e archi $\{(i, j) \quad \forall\ i, j$ per cui esiste un possibile segmento da i a $j\}$. Ad ogni cammino sul grafo da v_0 a v_n associamo la penalità (k, P), con k il numero degli archi del cammino e P la somma delle penalità dei possibili segmenti tra archi successivi. Tali coppie sono ordinate lessicograficamente: $(k, P) < (k', P')$ se $k < k'$ o se $k = k'$ e $P < P'$. Il poligono ottimale è quindi quello che minimizza la penalità del cammino e si trova in tempo $\mathcal{O}(nm)$, con n lunghezza del percorso e m massima lunghezza dei possibili segmenti.

2.3 Descrizione vettoriale del contorno

La terza fase dell'algoritmo si occupa di approssimare il contorno di un poligono con curve di Bézier ed angoli. Il primo passo da compiere è il pas-

saggio da coordinate intere a coordinate reali. Siccome l'insieme dei vertici di un poligono è un sottoinsieme di punti del percorso che il poligono stesso approssima, essi coincidono con vertici della bitmap come definiti nella sezione 2.1 e sono perciò identificati da coordinate intere nel sistema di riferimento della bitmap originale. Si vuole ora approssimare il poligono con segmenti definiti tra punti di coordinate non intere.

Dato un poligono $\{i_0, \ldots, i_m - 1\}$ relativo al percorso chiuso $\{v_0, \ldots, v_n\}$ calcoliamo per ogni coppia consecutiva di vertici (i_k, i_{k+1}) la retta $L_{k,k+1}$ che minimizza la somma dei quadrati delle distanze euclidee dei punti $v_{i_k}, \ldots, v_{i_{k+1}}$ dalla retta, con v_{i_k} vertice del percorso relativo al vertice i_k del poligono. Per ogni tripla di vertici consecutivi $\langle i_{k-1}, i_k, i_{k+1} \rangle$ associamo ad i_k un punto a_k tale che la massima distanza $d(a_k, v_{i_k}) \leq 1/2$ e che sia minima la somma dei quadrati delle distanze euclidee da a_k a $L_{k-1,k}$ e $L_{k,k+1}$. In particolare, se l'intersezione delle due rette è interna al quadrato unitario di centro v_{i_k} posizioniamo a_k in tale punto. La retta $L_{k,k+1}$ è calcolata a partire dai punti $v_{i_k} = (x_{i_k}, y_{i_k}), \ldots, v_{i_{k+1}}$; è passante per il punto $(E(x_k), E(y_k))$, con

$$E(x_k) = \frac{1}{i_{k+1} \ominus i_k + 1} \sum_{t=i_k}^{i_{k+1}} x_t \quad e \quad E(y_k) = \frac{1}{i_{k+1} \ominus i_k + 1} \sum_{t=i_k}^{i_{k+1}} y_t \ ,$$

ed ha inclinazione data dall'autovettore associato al più grande autovalore della matrice

$$\begin{pmatrix} a & b \\ b & c \end{pmatrix}, \quad \text{con} \quad \begin{aligned} a &= E(x_{k+1}^2) - E(x_k + 1)^2 \\ b &= E(x_{k+1} y_{k+1}) - E(x_{k+1}) E(y_{k+1}) \\ c &= E(y_{k+1}^2) - E(y_{k+1})^2 \end{aligned} \ .$$

La sequenza di vertici $a_0, \ldots, a_{m-1}$ così calcolati definisce il poligono sul quale si effettua l'approssimazione.

La classe delle curve utilizzate per descrivere il contorno del nuovo poligono è una restrizione della classe di curve di Bézier cubiche. Data la forma descritta nella sezione 1.2.4, si considerano le curve convesse per le quali le rette $\overline{P_0P_1}$ e $\overline{P_2P_3}$ si intersecano nel punto O e $P1 \in \overline{P_0O}$, $P2 \in \overline{OP_3}$. Assumiamo, tramite trasformazione lineare, che $P_0 = (-1, 0)$, $P_3 = (1, 0)$, $O = (0, 1)$. In

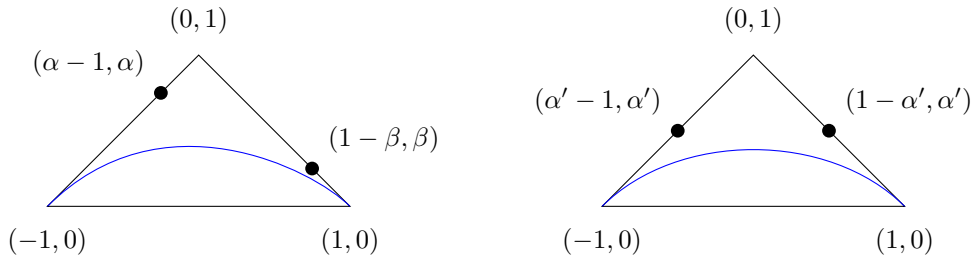


Figura 2.5: Esempio di curve di Bézier a due parametri: a sinistra la forma normale, a destra la forma approssimata

tal modo si può descrivere una curva tramite due parametri $\alpha, \beta \in [0, 1]$, con $P_1 = (\alpha - 1, \alpha)$ e $P_2 = (1 - \beta, \beta)$. Chiamiamo le curve così definite **Curve di Bézier a due parametri**. Inoltre, siccome due curve a due parametri sono simili se sottendono aree di egual misura, con area

$$A = \frac{3}{10}(2\alpha + 2\beta - \alpha\beta) = \frac{3}{10}(4 - (2 - \alpha)(2 - \beta)) \ ,$$

si può approssimare una curva di parametri α, β con una curva di parametri

$$\alpha' = \beta' = 2 - \sqrt{(2 - \alpha)(2 - \beta)} \ ,$$

come illustrato in figura 2.5.

L'approssimazione del contorno consiste nell'individuazione degli spigoli e nel calcolo dei parametri delle curve che meglio approssimano i vertici $a_0, \ldots, a_{m-1}$. Siano $b_0, \ldots, b_{m-1}$ i punti medi dei lati del poligono, ovvero $b_i = (a_i + a_{i+1})/2$. Per ogni i, consideriamo l'angolo $b_{i-1} \,..\, a_i \,..\, b_i$ e il quadrato unitario centrato in a_i. Si traccia la retta $L_i \parallel \overline{b_{i-1}b_i}$ tale che intersechi il quadrato e sia minima la distanza da $\overline{b_{i-1}b_i}$. Siano c il punto di intersezione tra L_i e $b_{i-1}a_i$, $\gamma = |\overline{b_{i-1}c}| \ / \ |\overline{b_{i-1}a_i}|$, $\alpha = 4\gamma/3$ e $\alpha_{max} = 1$. Se $\alpha > \alpha_{max}$, allora non esiste una curva di Bézier nella classe di curve considerata che connetta b_{i-1} e b_i e che sia tangente a L_i; si traccia quindi uno spigolo di lati $\overline{b_{i-1}a_i}$ e $\overline{a_ib_i}$. Se $\alpha \leq \alpha_{max}$ si traccia la curva di Bézier cubica convessa che connette b_{i-1} e b_i con parametro α, tangente a $\overline{b_{i-1}a_i}$, L_i e $\overline{a_ib_i}$. Nel caso $\alpha < 0.55$ si imposta a tale valore, in modo da evitare curve eccessivamente

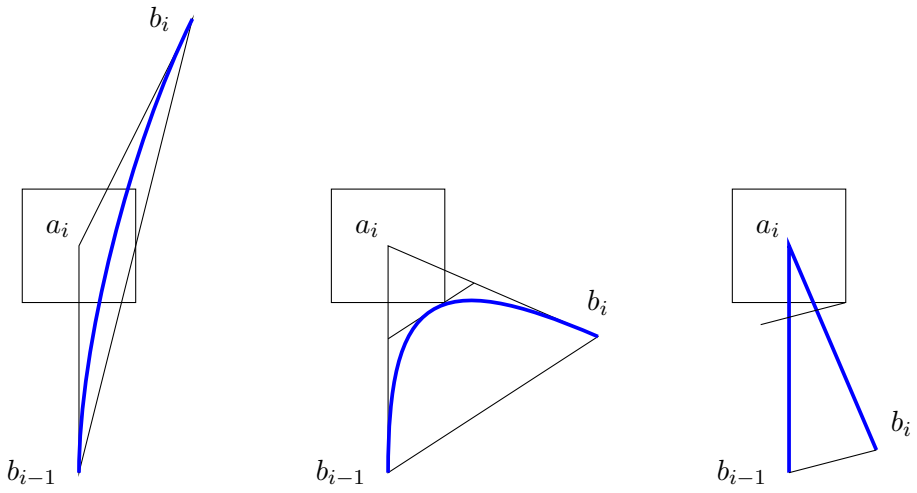


Figura 2.6: Esempi di approssimazione dei vertici

"piatte". Tale valore deriva dal valore teorico

$$\alpha_0 = \frac{4}{3}(\sqrt{2} - 1) \approx 0.552285 \ ,$$

che fornisce la miglior approssimazione di una curva di Bézier ad un quarto di circonferenza. Notare che la scelta di uno spigolo è favorita sia per angoli stretti che per segmenti definenti l'angolo lunghi. La figura 2.6 presenta le tre possibilità di configurazione degli spigoli in fase di approssimazione.

Alphamax

Il risultato del rilevamento degli spigoli dipende dal valore di α_{max}, il quale può essere modificato tramite il parametro `alphamax` . Valori minori risultano in un maggior numero di spigoli, mentre valori maggiori portano ad un contorno più morbido. Con $\alpha_{max} = 0$ non vengono tracciate curve e il risultato è un poligono. Con $\alpha_{max} > 4/3$ non viene tracciato alcuno spigolo.

2.4 Ottimizzazione delle curve

L'ultima fase dell'algoritmo si occupa di ottimizzare ulteriormente il contorno sostituendo una sequenza di curve adiacenti con una singola curva che le approssima, in modo da semplificarne la descrizione senza introdurre differenze visibili rispetto al risultato della fase precedente.

Si cercano sequenze di sole curve consecutive, di curvatura concorde, per cui si ha un cambio di direzione totale minore di 179 gradi. Dati i punti $b_0, \ldots, b_n$, si cerca quindi una curva da b_0 a b_n che possa approssimare le curve intermedie. Una possibile curva approssimante C è tangente a $\overline{b_0 a_1}$ e $\overline{a_n b_n}$; tracciando le rette su tali segmenti si può trovare l'intersezione O. Considerando la classe di curve a due parametri introdotta nella sezione 2.3, rimane solamente da trovare il parametro α. Imponiamo che l'area delimitata da C sia uguale all'area totale racchiusa tra le curve intermedie ed il segmento $\overline{b_0 b_n}$.

Il passo successivo consiste nel verificare se C è una buona approssimazione delle curve intermedie tramite un controllo sulle tangenti e nell'assegnare ad essa una penalità. Per ogni $i = 1, \ldots, n-1$ si trova il punto $z_i \in C$ tale che la tangente a C in z_i sia parallela al segmento $\overline{a_i a_{i+1}}$ e sia $d_i = \mathrm{dist}(z_i, \overline{a_i a_{i+1}})$. Per ogni $i = 1, \ldots, n$ si trova il punto $z'_i \in C$ tale che la tangente a C in z_i sia parallela al segmento $\overline{b_{i-1} b_i}$ e sia $d'_i = \mathrm{dist}(z'_i, L_i)$ positivo se z'_i è nello stesso semipiano di a_i rispetto a L_i e negativo altrimenti, con L_i come definita nella sezione 2.3.

L'approssimazione è accettabile se per ogni $i = 1, \ldots, n-1$, $d_i \leq \varepsilon$, per ogni $j = 1, \ldots, n$, $d'_j \geq -\varepsilon$ e la proiezione ortogonale di z_i su $\overline{a_i a_{i+1}}$ si trova tra a_i e a_{i+1}, con ε costante governata dal parametro `opttolerance` , di default impostato al valore 0.2.

Se la curva è accettabile si assegna un valore di penalità dato dalla somma dei quadrati di tutte le distanze d_i e d'_i. Si utilizza un algoritmo di ricerca del cammino minimo su un grafo pesato per decomporre una sequenza di curve in una approssimazione accettabile, ottimizzando sulla coppia (n, p), con n numero di segmenti e p penalità totale.

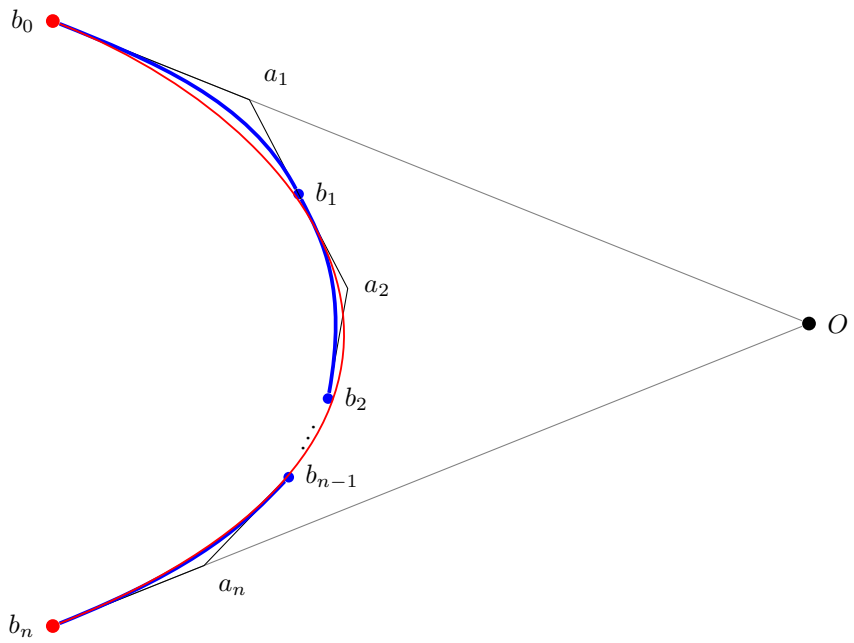
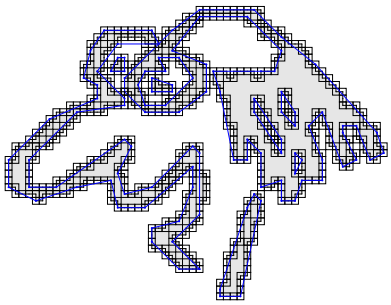
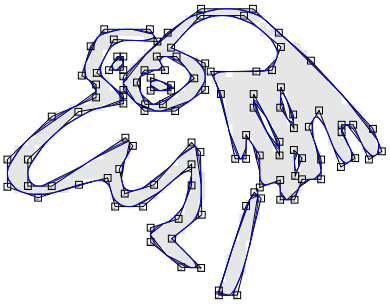


Figura 2.7: Esempio di ottimizzazione di una sequenza di curve. Il tratto rosso è la curva approssimante.

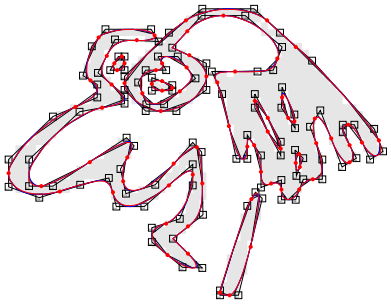
(a) Bitmap



(b) Bordi e poligono ottimale



(c) Approssimazione vettoriale



(d) Ottimizzazione

(e) Risultato

Figura 2.8: Esempio completo dell'algoritmo di Potrace. In (b) si può notare come la politica **minority** mantenga uniti i pixel bianchi interni ai capelli e come il **despeckling** abbia rimosso un singolo pixel bianco. In (c) sono presenti i quadrati unitari, i segmenti L_i e i valori di α. In questo esempio sono stati tracciati pochi spigoli, dato che il rilevamento degli spigoli funziona meglio a risoluzioni maggiori (le dimensioni originali della bitmap sono 57×46 pixel). In (d) la curva blu è l'originale, quella rossa l'ottimizzata. Il numero di tratti è stato ridotto da 112 a 68, ovvero del 40%.

Capitolo 3

Integrazione in Mini-System

In questo capitolo sarà descritta l'integrazione della libreria di Potrace in Mini-System e sarà introdotto il principale algoritmo di riempimento dei poligoni disponibile nel pacchetto come funzionalità di visualizzazione.

Per il corretto utilizzo della libreria si rendono necessarie due fasi di conversione tra strutture dati: la prima fase, precedente al tracing, svolge la conversione dell'immagine caricata da surface SDL a formato bitmap di Potrace; la seconda fase, successiva al tracing, si occupa di adattare le definizioni dei contorni in output dalle strutture definite dalla libreria alle strutture di curve proprie di Mini-System. Si sono inoltre implementate tecniche di gestione delle immagini non monocrome, quali conversione in scala di grigi e thresholding.

3.1 Conversione bitmap

Richiamiamo quanto detto nella sezione 2.1: nella logica di Potrace, una bitmap di dimensione $w \times h$ è descritta in un sistema di riferimento cartesiano, con ogni pixel di dimensione unitaria ed i cui angoli sono posizionati nei punti di coordinate intere. Introduciamo la notazione $[x, y]$ per indicare il pixel i cui angoli hanno coordinate (x, y), $(x,\ y+1)$, $(x+1,\ y+1)$, $(x+1,\ y)$. Inoltre, ricordiamo che tale bitmap è monocroma.

Una bitmap è rappresentata in memoria [8] come istanza della struttura

```
struct potrace_bitmap_s
{
    int w, h;            /*  width and height, in pixels  */
    int dy;              /*  scanline offset in words     */
    potrace_word *map;   /*  pixel data, dy*h words       */
};
```

Assumiamo che `potrace_word` sia definito come tipo intero senza segno e che abbia dimensione N bit. La bitmap è divisa, dal basso verso l'alto, in h **scanline** orizzontali, ognuna delle quali è suddivisa in blocchi di N bit da sinistra a destra. Ogni blocco è contenuto in una `potrace_word` , con il pixel più a sinistra corrispondente al bit più significativo. I pixel neri sono rappresentati dal valore binario 1, quelli bianchi dal valore 0. Se il numero di bit della scanline non è divisibile da N, la word più a destra è riempita con un **padding** di zeri. L'informazione della scanline n inizia all'indirizzo di $map[n \cdot dy]$ e il valore del pixel $[x, y]$ è espresso come

```
pixel(x,y) = ((map + y*dy)[x/N] & (1 << (N-1 - x%N)) ? 1 : 0
```

In figura 3.1 è rappresentato un esempio di organizzazione in memoria di una bitmap nel formato di Potrace. La relativa struttura `potrace_bitmap_s` è così costituita:

```
    w = 36;
    h = 12;
    dy = 2;
    map[22]    =   0xfff0fc02;    map[23]    =   0x00000000;
    map[20]    =   0x7ff1fe02;    map[21]    =   0x00000000;
    map[18]    =   0x3ff3ff07;    map[19]    =   0x00000000;
    map[16]    =   0x1ff7ff87;    map[17]    =   0x00000000;
    map[14]    =   0x0ff7cf8f;    map[15]    =   0x80000000;
```

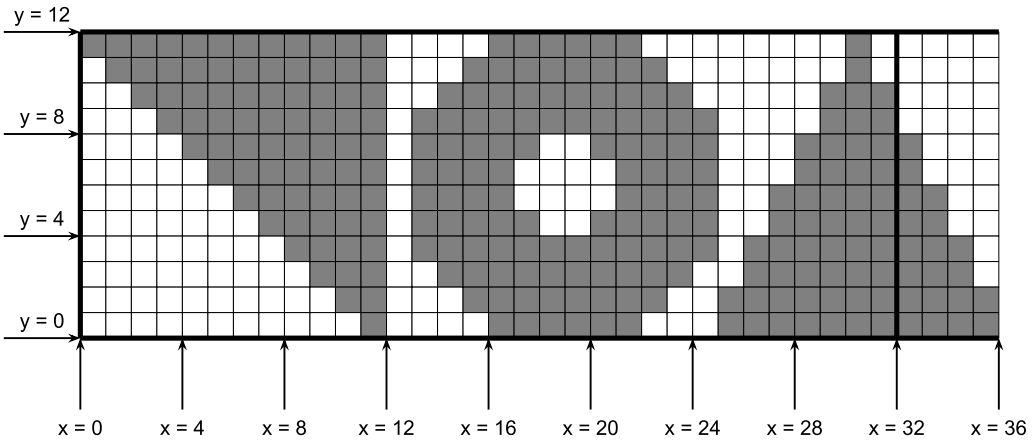


Figura 3.1: Esempio di rappresentazione bitmap di Potrace

```
map[12]   =  0x07f7878f;   map[13]   =  0x80000000;
map[10]   =  0x03f7879f;   map[11]   =  0xc0000000;
map[8]    =  0x01f7cf9f;   map[9]    =  0xc0000000;
map[6]    =  0x00f7ffbf;   map[7]    =  0xe0000000;
map[4]    =  0x0073ff3f;   map[5]    =  0xe0000000;
map[2]    =  0x0031fe7f;   map[3]    =  0xf0000000;
map[0]    =  0x0010fc7f;   map[1]    =  0xf0000000;
```

3.2 Conversione contorni

Come già introdotto nella sezione 2.1, l'output di Potrace consiste in un insieme di percorsi (**path**) che approssimano i contorni della bitmap, ognuno dei quali è costituito da una sequenza di spigoli e curve (sezione 2.3).

I path sono organizzati [8] in strutture ad albero, dacché sono assimilabili a curve chiuse non intersecate. In particolare, ogni percorso è caratterizzato da un segno, positivo o negativo, corrispondente rispettivamente ad una zona di primo piano o di sfondo. Poiché le due zone si alternano, ogni nodo dell'albero ha come figli e come padre path di segno opposto al proprio. La figura 3.2 mostra un esempio di decomposizione di un'immagine in una foresta

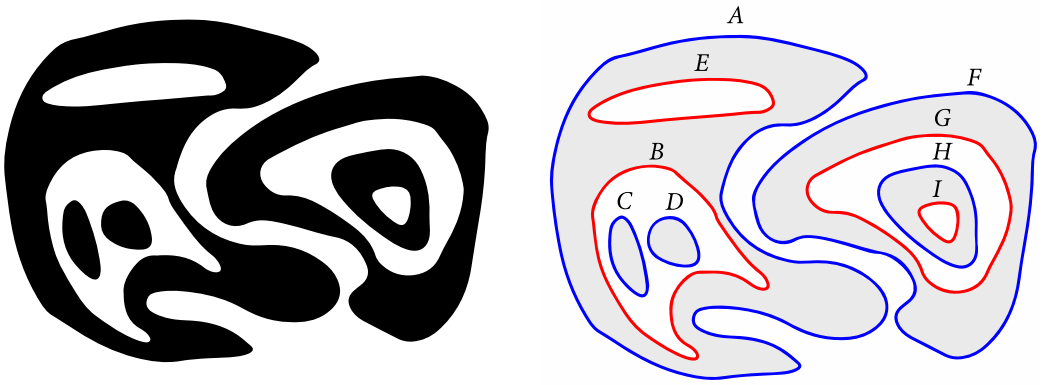


Figura 3.2: Esempio di decomposizione in path

di path, blu se positivi e rossi se negativi. La struttura `potrace_path_s` rappresentante un percorso è così definita:

```
struct potrace_path_s
{
    int area;                           /* enclosed area  */
    int sign;                           /* '+' or '-'     */
    potrace_curve_t curve;              /* vector data    */
    struct potrace_path_s *next;        /* list structure */
    struct potrace_path_s *childlist;   /* tree structure */
    struct potrace_path_s *sibling;     /* tree structure */
    struct potrace_privpath_s *priv;    /* private state  */
};
typedef struct potrace_param_s potrace_param_t;
```

I puntatori `childlist` e `sibling` sono i riferimenti alla lista dei nodi figli e al fratello destro, rispettivamente. Il puntatore `next` è utilizzato per fornire una rappresentazione dell'albero come lista, costruita in modo che

- per ogni C_1 e C_2 path tali che C_1 è contenuto in C_2, quest'ultimo ha indice inferiore rispetto a C_1;
- ogni path positivo è seguito dai path figli.

La struttura di base `potrace_curve_s` rappresenta una curva chiusa, ovvero una sequenza di `n` tratti in cui l'ultimo punto dell'ultimo tratto coincide con il primo punto del primo tratto.

```
struct potrace_curve_s
{
    int n;                      /* number of segments       */
    int *tag;                   /* array of segment types   */
    potrace_dpoint_t (*c)[3];   /* array of control points. */
};
typedef struct potrace_curve_s potrace_curve_t;
```

L'array `tag` ha dimensione n e rappresenta il tipo del tratto i-esimo: `POTRACE_CURVETO` indica una curva di Bézier, mentre `POTRACE_CORNER` indica uno spigolo. L'array `c` ha dimensione $n \times 3$ e contiene i punti caratteristici dei tratti. Se l'i-esimo tratto è una curva di Bézier, allora `c[i][0]` e `c[i][1]` sono i punti di controllo e `c[i][2]` è il punto finale, altrimenti `c[i][0]` è inutilizzato, `c[i][1]` è il vertice dello spigolo e `c[i][2]` è il punto finale. Siccome il punto iniziale di ogni tratto coincide con il punto finale del precedente non è necessario memorizzarlo esplicitamente.

La memorizzazione di una curva di Bézier in Mini-System è affidata ad una lista di istanze della struttura `bezier_t` , definita come segue:

```
typedef struct bezier_t
{
    int order;              /*  ordine della curva          */
    real_t xcp[MAX_ORDER];  /*  coord. x del cp             */
    real_t ycp[MAX_ORDER];  /*  coord. y del cp             */
    real_t zero, one;       /*  dominio parametrico         */
    struct bezier_t *next;  /*  il campo next punta ad una
                                struttura dello stesso tipo  */
} Bezier_t;
```

La conversione da `potrace_curve_s` a `bezier_t` è triviale. Nel caso in cui il tratto sia una curva, è sufficiente impostare `order = 4` e copiare le coordinate dei rispettivi punti

```
ns->xcp[j+1] = c[i][j].x;
ns->ycp[j+1] = c[i][j].y;
```

con i indice del tratto e $j = 0, \ldots, 2$. Nel caso in cui il tratto sia uno spigolo, si creano due curve, `ns` e `ns2`, con `order = 2` : nella prima si copiano le coordinate

```
ns->xcp[0] = sp.x;
ns->ycp[0] = sp.y;

ns->xcp[1] = c[i][1].x;
ns->ycp[1] = c[i][1].y;
```

mentre nella seconda

```
ns2->xcp[0] = c[i][1].x;
ns2->ycp[0] = c[i][1].y;

ns2->xcp[1] = c[i][2].x;
ns2->ycp[1] = c[i][2].y;
```

In entrambi i casi il punto iniziale `sp` è recuperato dall'ultimo punto del tratto precedente tramite la formula `sp = c[i ? i-1 : n-1][2];` . Una volta convertite tutte le curve in curve di Bézier, si effettua la chiamata a `bez2mdC0` su ognuna di esse per effettuarne l'ulteriore conversione a curve MD-spline con continuità C^0 nei punti di raccordo e si cerca di aumentarne la continuità tramite la funzione `knot_simplify_md` .

3.3 Gestione immagini

Siccome Potrace richiede una bitmap monocroma in input, si rende necessario gestire opportunamente le immagini in scala di grigi o a colori. Considerato che anche Mini-System lavora in monocromia, risulta più adatto convertire le immagini a tale profondità colore piuttosto che effettuarne la segmentazione, ossia la separazione in regioni di colori attigui, con algoritmi come k-means o SRM (Statistical Region Merging). L'importazione di immagini per il tracing si articola quindi nelle seguenti fasi:

1. rilevamento della profondità colore dell'immagine in input;
2. conversione in scala di grigi, se l'immagine è a colori;
3. conversione a monocromia (**thresholding**), se l'immagine è in scala di grigi.

Per la prima fase non ci si serve della definizione di profondità colore resa disponibile dalla struttura della surface SDL in quanto essa si basa sui metadati dell'immagine stessa, i quali sono indicativi del formato e non del contenuto. Ad esempio, un file JPEG potrebbe essere segnalato come immagine a colori dai metadati, ma non contenere alcun pixel colorato. Tale rilevamento si effettua quindi scandendo l'intera matrice e verificando se esiste almeno un pixel con componenti colore diverse tra loro, nel qual caso si ha un'immagine a colori, o un pixel con componenti uguali tra loro ma diverse da 0 o 255, nel qual caso si ha un'immagine in scala di grigi.

3.3.1 Conversione in scala di grigi

La conversione da formato RGB a scala di grigi consiste nella sostituzione del codice colore di ogni pixel dell'immagine con la rappresentazione $RGB(Y, Y, Y)$, ove Y rappresenta la **luminanza relativa**, ossia la misura dell'intensità luminosa per unità di area, ed è calcolata come somma pesata delle tre componenti colore lineari:

$$Y = (w_r \cdot R_{lineare} + w_g \cdot G_{lineare} + w_g \cdot B_{lineare}) \ .$$

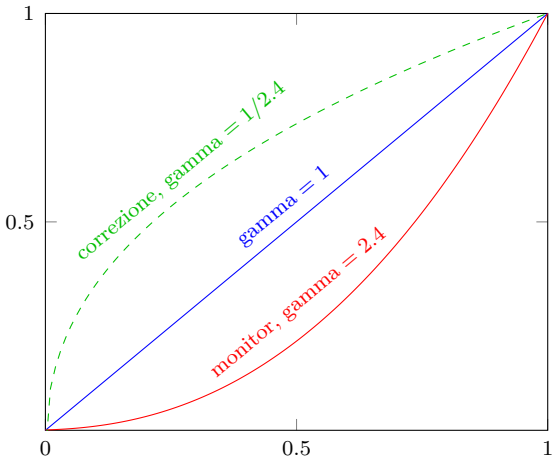


Figura 3.3: Grafico delle trasformazioni di compressione e decompressione della gamma per sRGB

Nel caso l'immagine sia definita in uno spazio colore **gamma-compresso**, come ad esempio **sRGB**, i valori delle componenti lineari si ottengono dai valori in rappresentazione non lineare applicando una trasformazione di espansione della gamma (figura 3.3). Si assume che le immagini in input siano definite nello spazio sRGB in quanto specifica standard per sistemi digitali e Internet. La trasformazione di espansione della gamma per sRGB è così definita [9]:

$$C_{lineare} = \begin{cases} \frac{C_{sRGB}}{12.92} & C_{sRGB} \leq 0.04045 \\ (\frac{C_{sRGB}+0.055}{1.055})^{2.4} & \text{altrimenti} \end{cases},$$

dove C_{sRGB} è qualunque delle componenti gamma-compresse e $C_{lineare}$ è la corrispettiva componente lineare.

A questo punto, si potrebbe pensare di assegnare ai tre pesi lo stesso valore 1/3, ottenendo quindi la media delle componenti. Tuttavia, tale decisione non produce il risultato ottimale, in quanto la percezione dell'occhio umano della luminosità relativa all'intensità del colore cambia a seconda della tonalità. Esistono numerosi standard di specifica del formato di immagine che definiscono pesi diversi per meglio approssimare la luminosità percepita a seconda della gamma colore. Lo spazio sRGB utilizza i colori primari de-

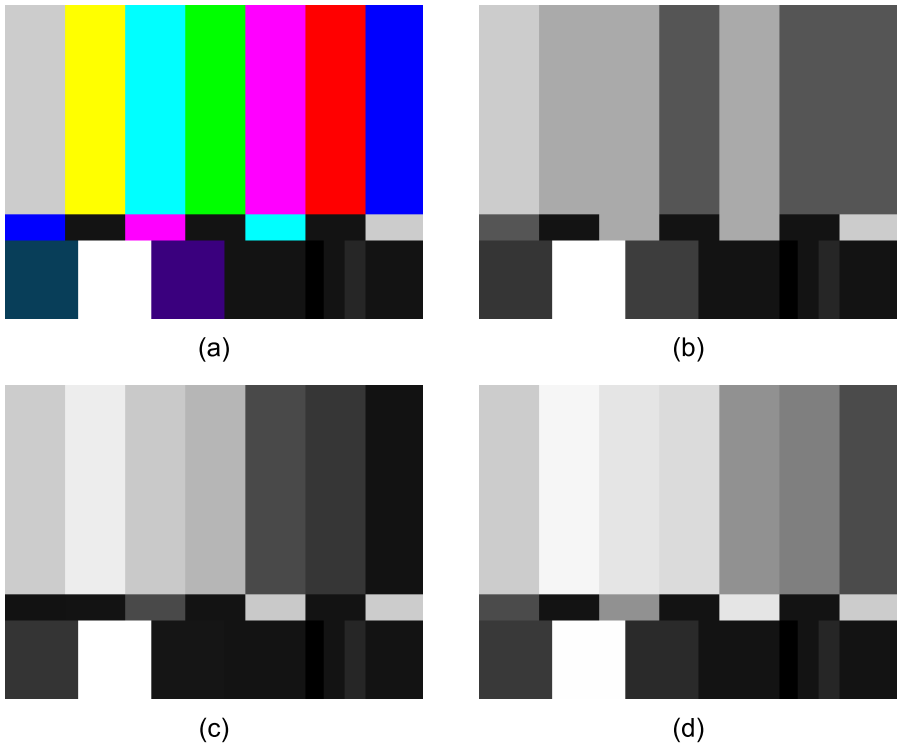


Figura 3.4: Diversi metodi di conversione in scala di grigi. (a) è l'immagine originale; (b) è ottenuta tramite media delle componenti colore; (c) è il risultato del calcolo del **luma**, ossia la media pesata delle componenti gamma-compresse; (d) è il risultato del calcolo della **luminanza**. Luma e luminanza condividono gli stessi coefficienti nella specifica BT.709; la differenza è dovuta unicamente alla compressione della gamma.

finiti nella **ITU-R Recommendation BT.709**, la quale definisce inoltre i seguenti coefficienti per il calcolo della luminanza [10]:

$$w_r = 0.2126, \quad w_g = 0.7152, \quad w_b = 0.0722 \ .$$

Il valore di luminanza così calcolato deve quindi essere riconvertito nello spazio gamma-compresso prima di ottenerne la rappresentazione RGB. La trasformazione di compressione della gamma per sRGB è così definita [9]:

$$C_{sRGB} = \begin{cases} 12.92 \cdot C_{lineare} & C_{lineare} \leq 0.0031308 \\ 1.055 \cdot C_{lineare}^{\frac{1}{2.4}} - 0.055 & \text{altrimenti} \end{cases} .$$

3.3.2 Thresholding

Le immagini in scala di grigi devono necessariamente essere convertite in bitmap monocrome per poter essere gestite da Potrace. Tale procedimento viene effettuato tramite **thresholding**, ossia convertendo tutti i pixel il cui livello di grigio è minore o maggiore di un determinato valore soglia a nero o bianco, rispettivamente. Tale valore di soglia può essere fissato globalmente e quindi valevole per l'intera immagine, oppure calcolato localmente per ogni pixel come somma pesata dei livelli di grigio di un intorno del pixel in esame. Si sono implementati due metodi globali, a valore fisso e adattivo, e due metodi locali, che sfruttano la tecnica di convoluzione con kernel tipica di diversi filtri per l'elaborazione digitale di immagini.

Valore fisso

Il valore di soglia di default per il thresholding globale a valore fisso è di 128 su una scala di 256 livelli di grigio ed è modificabile dall'utente tramite l'apposita maschera. Tale valore ha risultati migliori quanto più bilanciata è la distribuzione dei livelli di grigio nell'immagine rispetto al centro della scala. In caso di distribuzioni non uniformi si può utilizzare il **Metodo di Otsu**, che introduce un criterio di valutazione di "separabilità" utilizzato per il calcolo del valore di soglia ottimale.

Metodo di Otsu

L'idea di base dell'algoritmo consiste nello scegliere il valore di soglia che divida l'istogramma dei livelli di grigio dell'immagine in due classi, per sfondo e primo piano, tali che sia massima la varianza interclasse [11].

Siano n_i il numero di pixel al livello i e $N = \sum_{i=1}^{L} n_i$ il numero di pixel totali, con L numero dei livelli. Si considera l'istogramma come distribuzione di probabilità:

$$p_i = \frac{n_i}{N} \qquad p_i > 0, \ \sum_{i=1}^{L} p_i = 1 \ .$$

Si supponga ora di dividere i pixel in due classi C_b e C_f con valore di soglia k. Si introducono le definizioni di ripartizione di probabilità e valore medio e di media totale:

$$w(k) = \sum_{i=1}^{k} p_i \qquad \mu(k) = \sum_{i=1}^{k} i\ p_i \qquad \mu_T = \mu(L) = \sum_{i=1}^{L} i\ p_i\ .$$

Le probabilità e i valori medi delle classi sono rispettivamente dati da

$$w_b = \sum_{i=1}^{k} p_i = w(k) \qquad w_f = \sum_{i=k+1}^{L} p_i = 1 - w(k)$$

e

$$\mu_b = \sum_{i=1}^{k} \frac{i\ p_i}{w_b} = \frac{\mu(k)}{w(k)} \qquad \mu_f = \sum_{i=k+1}^{L} \frac{i\ p_i}{w_f} = \frac{\mu_T - \mu(k)}{1 - w(k)}\ .$$

È ora possibile introdurre diversi criteri di valutazione della scelta del valore di soglia. Il calcolo più semplice si ha con il seguente criterio:

$$\eta = \frac{\sigma_B^2}{\sigma_T^2}\ ,$$

dove σ_B^2 e σ_T^2 sono rispettivamente la varianza interclasse e la varianza totale, così definite:

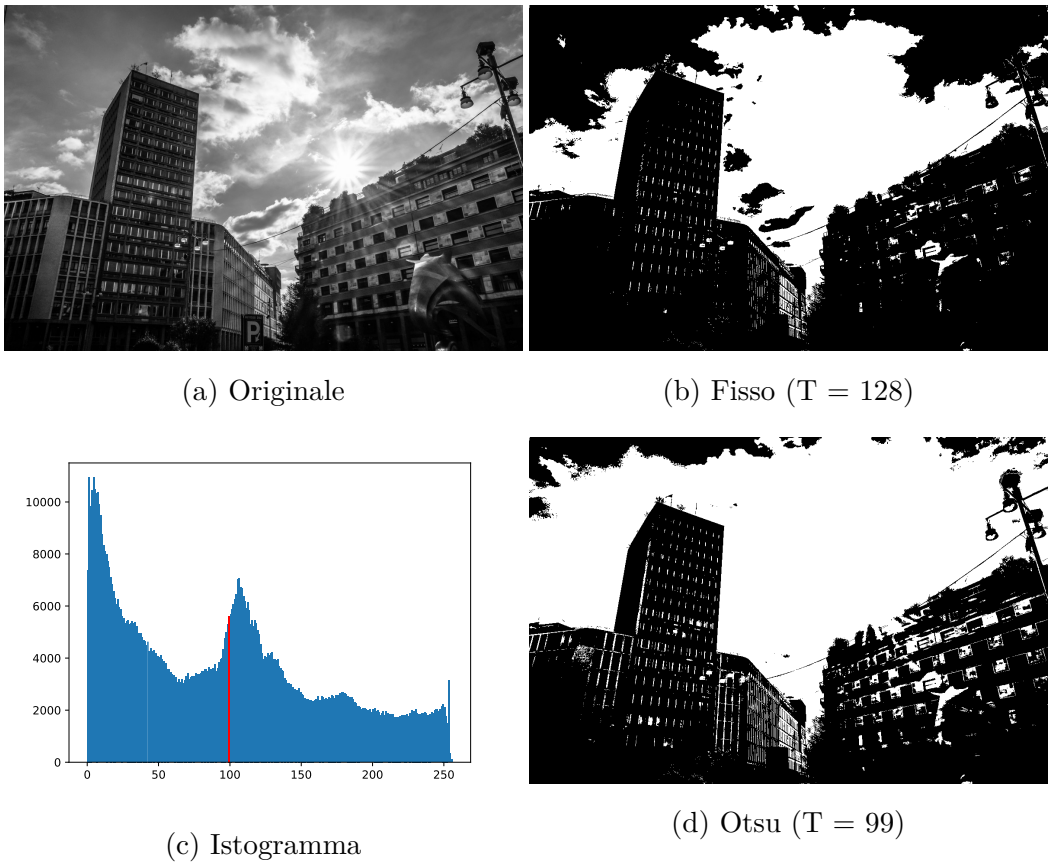
$$\sigma_B^2 \ = \ w_b(\mu_b - \mu_T)^2 + w_f(\mu_f - \mu_T)^2 \ = \ w_b w_f(\mu_f - \mu_b)^2$$

$$\sigma_T^2 = \sum_{i=1}^{L}(i - \mu_T)^2 p_i\ .$$

La ricerca si riduce quindi ad un problema di ottimizzazione, in cui si cerca il valore di k tale che sia massimo il valore del criterio η. Dato che σ_B^2 è definita in funzione di k e σ_T^2 è invece indipendente da essa, è equivalente massimizzare il solo σ_B^2, ossia trovare il valore di soglia ottimo k^* tale che sia

$$\sigma_B^2(k^*) = \max_{1 \le k < L} \sigma_B^2(k)\ .$$

La figura 3.5 presenta una comparazione dei due metodi di thresholding globale per un'immagine di esempio. Si può notare come il risultato del metodo di Otsu mantenga un maggior numero di dettagli nelle facciate dei palazzi e separi distintamente la forma del lampione, ma al contempo approssimi meno accuratamente la forma delle nuvole.



(a) Originale

(b) Fisso (T = 128)

(c) Istogramma

(d) Otsu (T = 99)

Figura 3.5: Esempio di thresholding con i due metodi globali. In (c) la linea rossa rappresenta il valore di soglia ottimo per il metodo di Otsu.

Metodi locali

Il valore di soglia nel thresholding locale è calcolato per ogni pixel tramite convoluzione con kernel quadrato, ossia effettuando per ogni pixel dell'immagine la sovrapposizione di una matrice quadrata di lato dispari con centro nel pixel in esame e calcolando la somma pesata dei livelli di grigio dei pixel interni all'area di sovrapposizione utilizzando i valori della matrice come pesi. Al risultato così ottenuto si sottrae una costante positiva C utile a ridurre il rumore. La formula calcolata è quindi

$$T = (\sum_{y=1}^{N}\sum_{x=1}^{N} K[y][x] \cdot I[c_y - \lceil\frac{N}{2}\rceil + y][c_x - \lceil\frac{N}{2}\rceil + x]) - C \ , \tag{3.1}$$

dove K è il kernel di dimensione $N \times N$, con N dispari e $N \leq I_h \wedge N \leq I_w$, I è la matrice di dimensione $I_h \times I_w$ contenente i livelli di grigio dei pixel dell'immagine e (c_x, c_y) è il centro di sovrapposizione del kernel. Si assume che il kernel sia normalizzato, ossia che

$$\sum_{y=1}^{N}\sum_{x=1}^{N} K[y][x] = 1 \ .$$

I due metodi di convoluzione implementati, **media** e **gaussiana**, differiscono unicamente per i pesi del kernel. Nel primo caso vale

$$K[y][x] = \frac{1}{S} \quad \forall x, y.\ 1 \leq x, y \leq N \ ,$$

risultando quindi nel calcolo della media aritmetica. Nel secondo caso invece si approssima la funzione gaussiana in due dimensioni, definita come

$$g(x, y) = \frac{1}{2\pi\sigma} \cdot \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \ ,$$

considerando l'origine posizionata nel centro della matrice. L'espressione calcolata è quindi

$$K[y][x] = \frac{1}{2\pi\sigma} \cdot \mathrm{e}^{-\frac{(\lceil N\rceil - x)^2 + (\lceil N\rceil - y)^2}{2\sigma^2}} \quad \forall x, y.\ 1 \leq x, y \leq N \ ,$$

con $\sigma = 0.3 \cdot ((N-1) \cdot 0.5 - 1) + 0.8$, come da sorgenti OpenCV [12]. La figura 3.6 presenta una comparazione dei due metodi di thresholding globale per la

(a) Media ($N = 11,\ C = 4$) (b) Gaussiana ($N = 11,\ C = 4$)

Figura 3.6: Esempio di thresholding con i due metodi locali

stessa immagine della figura 3.5a. Si può notare come entrambi i risultati prodotti siano rappresentativi dei contorni dei soggetti raffigurati e che in generale il metodo gaussiano produca un'immagine più pulita.

Gestione dei bordi

Per ogni passo di convoluzione su un pixel per cui vale

$$\min(\min(c_y,\ I_h - c_y + 1),\ \min(c_x,\ I_w - c_x + 1)) \leq \lfloor \frac{N}{2} \rfloor$$

il calcolo della somma pesata necessiterebbe l'accesso ai livelli di grigio di pixel esterni ai bordi dell'immagine. Esistono diversi metodi per ovviare al problema:

- **Estensione**: si estende l'immagine di $2 \cdot \lfloor \frac{N}{2} \rfloor$ pixel in entrambe le dimensioni, riempiendo ogni riga/colonna dell'estensione con lo stesso colore del pixel sulla stessa riga/colonna facente parte del bordo più vicino ed ogni quadrato rimanente agli angoli con lo stesso colore del più vicino pixel costituente un angolo dell'immagine originale. La convoluzione è effettuata sul rettangolo originale;

- **Avvolgimento**: si convertono le coordinate dei pixel (p_y, p_x) esterni ai bordi in $((p_y - 1) \bmod I_h + 1,\ (p_x - 1) \bmod I_w + 1)$;

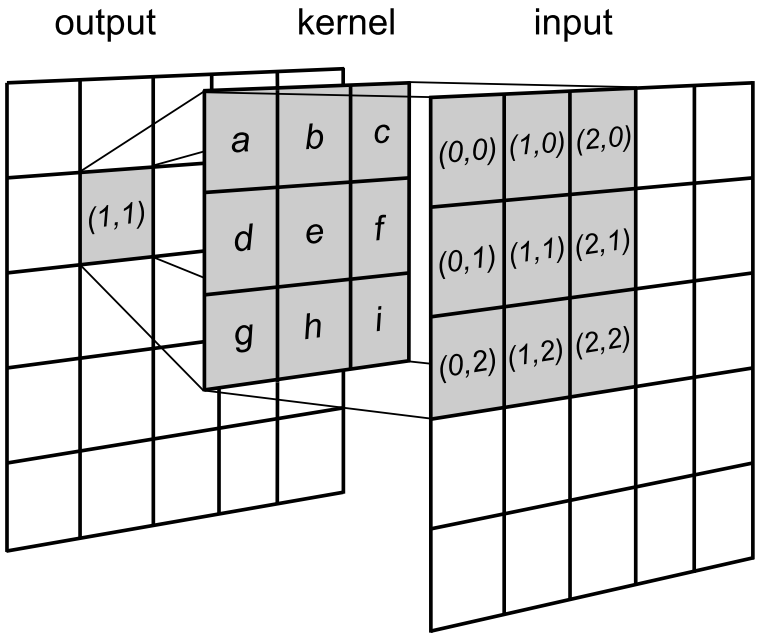


Figura 3.7: Illustrazione di un passo di convoluzione

- **Specchiamento**: si converte ciascuna coordinata p_c dei pixel esterni ai bordi in

$$c' = \begin{cases} (I_l - p_c) \bmod I_l & p_c < 1 \\ (I_l - p_c) \bmod I_l + 1 & p_c > I_l \end{cases},$$

  dove I_l è la lunghezza del lato dell'immagine relativo alla coordinata;

- **Scarto**: non si esegue il passo di convoluzione sui pixel per i quali si necessiterebbe l'accesso a pixel esterni ai bordi dell'immagine, scartandoli per la bitmap di output e di fatto riducendone la dimensione;

- **Potatura**: si "taglia" la parte di kernel corrispondente all'area esterna all'immagine, adattando la normalizzazione sui pesi rimasti.

Si è implementato il metodo di estensione per entrambi i metodi locali.

Ottimizzazione

Si calcola ora la complessità computazionale della procedura di convoluzione. Per semplificarne l'espressione, assumiamo che l'immagine di input sia quadrata, ossia $I_h = I_w = L$, mentre vale la precedente definizione di kernel. La complessità temporale dell'operazione, come definita in (3.1), è quindi $\mathcal{O}(L^2N^2)$. Esistono diverse tecniche di ottimizzazione di operazioni di convoluzione; di seguito è descritta la suddivisione in due passi tramite separazione del kernel.

Definizione. Un kernel di convoluzione $N \times N$ è **separabile** se può essere scritto come prodotto di due matrici di dimensione N.

Un esempio di kernel separabile è quello utilizzato dall'**Operatore di Sobel**, un filtro largamente utilizzato per il rilevamento dei contorni:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}.$$

In particolare, un kernel è separabile se ha rango 1, in quanto il prodotto di due matrici monodimensionali è una matrice le cui righe sono ognuna necessariamente una combinazione lineare del fattore matrice riga.

Siccome l'operazione di convoluzione è associativa, si può suddividere la convoluzione con un kernel separabile in due convoluzioni con i kernel che lo compongono, applicando la prima di esse all'immagine originale per ottenere un'immagine intermedia e la seconda all'immagine intermedia, ottenendo così l'immagine di output (figura 3.8). In tal modo si riduce la complessità temporale a $\mathcal{O}(L^2 2N) = \mathcal{O}(L^2 N)$. Entrambi i metodi locali utilizzati hanno kernel separabile [13]; si è quindi scelto di implementarli in forma separata.

3.4 Gestione parametri

L'interfaccia utente di Mini-System offre numerose possibilità di personalizzazione del funzionamento del software, sia per quanto riguarda i parametri

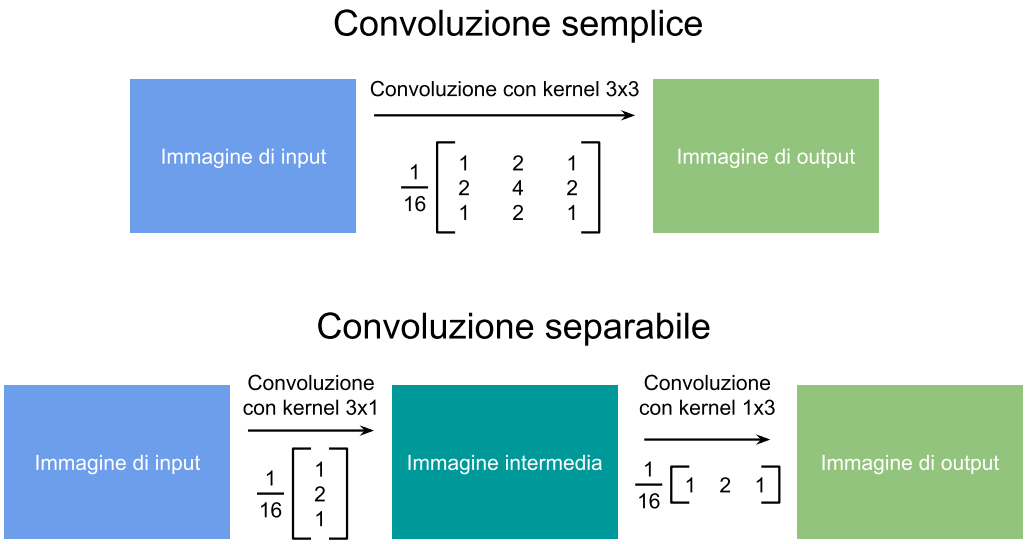


Figura 3.8: Esempio di separabilità della convoluzione con kernel gaussiano

di calcolo, valutazione e disegno, sia per quanto concerne la visualizzazione delle curve. Volendo rimanere coerenti con la presente scelta, si è fornito l'accesso ai parametri più rilevanti di thresholding e tracing tramite apposite finestre di dialogo. Tutte le impostazioni aggiunte sono raggiungibili dal menu "Trace" presente nella barra dei menu (figura 3.9). Da qui si può direttamente scegliere il metodo di thresholding da utilizzare o si può selezionare la voce relativa alla finestra di dialogo dei parametri che si vogliono modificare.

I parametri disponibili per la calibrazione del processo di tracing comprendono tutti quelli citati nel secondo capitolo, ovvero `turnpolicy` , `turdsize` , `alphamax` e `opttolerance` , ai quali si aggiunge il parametro booleano `opticurve` , il quale permette di attivare o disattivare l'ultima fase di ottimizzazione delle curve. La figura 3.10 presenta la finestra di dialogo tramite la quale si possono modificare suddetti parametri, con i rispettivi valori di default. Riepiloghiamo gli effetti ed i domini di ciascun parametro:

- `turnpolicy` : imposta la decisione in caso di ambiguità nell'estensione del percorso; valore di un'enumerazione a scelta tra **black**/**white** (con-

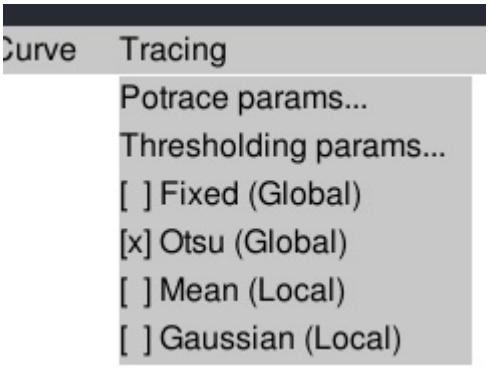


Figura 3.9: Menu delle impostazioni per il tracing

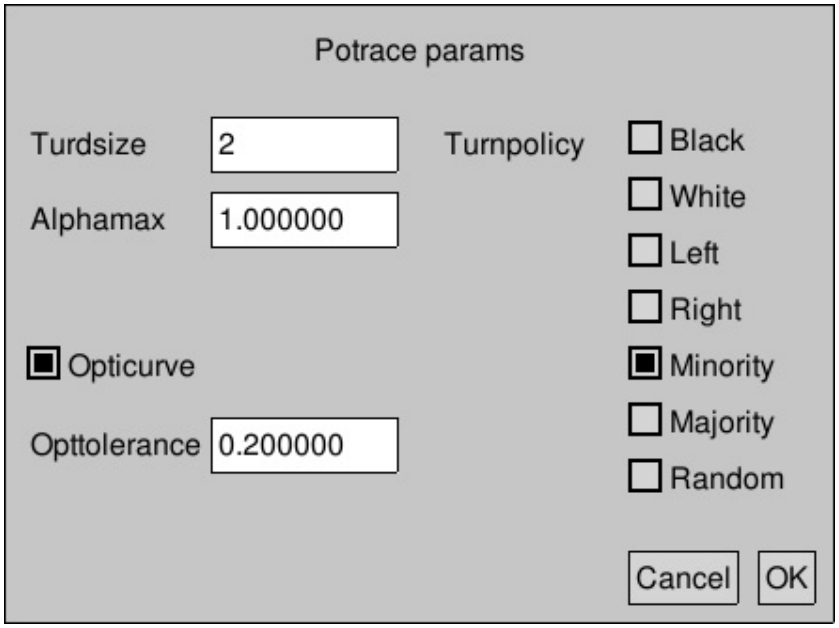


Figura 3.10: Finestra di dialogo per i parametri di Potrace

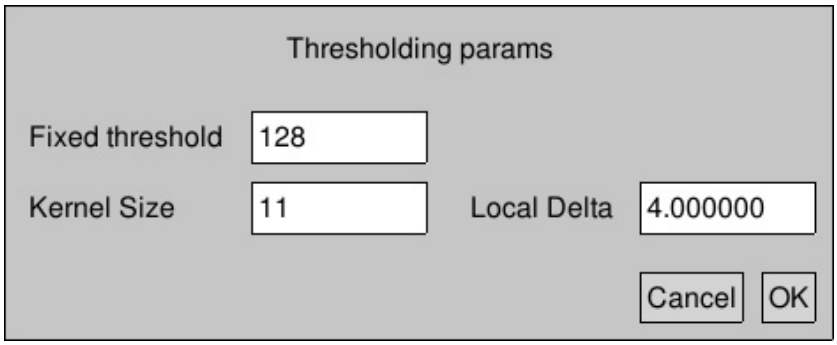


Figura 3.11: Finestra di dialogo per i parametri dei metodi di thresholding

nessione di regioni del rispettivo colore), **left/right** (direzione fissa), **minority/majority** (connessione di regioni del colore con la rispettiva frequenza in un intorno), **random** (scelta pseudo-randomica).

- `turdsize` : valore di soglia dell'area per il despeckling; intero ≥ 0.
- `alphamax` : valore di soglia per il rilevamento degli spigoli, agisce sulla morbidezza delle forme; reale da 0.0 (massima spigolosità) a 1.3334 (assenza di spigoli).
- `opttolerance` : definisce la quantità di errore ammesso nella semplificazione delle curve; reale ≥ 0.0; valori maggiori diminuiscono il numero delle curve finali a discapito della precisione.
- `opticurve` : stato di attivazione della fase di ottimizzazione delle curve; valore booleano.

I parametri disponibili per la personalizzazione dei metodi di thresholding sono tre, rappresentati con i rispettivi valori di default in figura 3.11:

- `Fixed threshold` : valore di soglia per il thresholding a valore fisso; intero da 0 a 255.
- `Kernel size` : dimensione N del kernel per entrambi i metodi locali; intero dispari $3 \leq N \leq 255$.

- `Local delta` : costante C sottratta dal valore risultante un passo di convoluzione, per entrambi i metodi locali; reale $0 \leq C \leq 255$.

3.5 Riempimento dei poligoni

Il riempimento dei poligoni, o **filling**, è una delle funzionalità caratteristiche di numerosi software di grafica sia raster che vettoriale. Il formato SVG altresì supporta la definizione di filling delle forme con colori, sfumature e pattern. Sono disponibili diverse metodologie di filling, quali algoritmi a scansione, a inondazione (*flood filling*) e di test interno-esterno. Mini-system integra il metodo **scan conversion** come opzione di visualizzazione nel rastering delle curve, ossia non legato alle singole curve e non permanente, di modo da fornire all'utente la possibilità di valutare velocemente l'aspetto d'insieme delle forme disegnate.

Come suggerisce il nome, scan conversion è un metodo a scansione, la cui idea consiste nell'intersecare delle linee di scansione con il poligono e riempire i pixel tra coppie di punti di intersezione. Sia C il canvas di dimensione $C_y \times C_x$ nel quale sono definite le curve e sia G un generico poligono chiuso sul canvas. Per ogni $y = 0, \ldots, C_y$ si calcolano i punti $P = (P_x, P_y) \in G$ di ordinata y, ovvero i punti di intersezione tra il poligono e la scanline per $x = 0, \ldots, C_x$ e si inseriscono in una lista $L = [P_i]_{i=0}^{N}$. Si ordina la lista in modo che sia $P_{i_x} < P_{(i+1)_x} \quad \forall\, i = 0, \ldots, N-1$. Si divide la lista in coppie di punti (P_1, P_2) e si procede a tracciare una linea di estremi P_1 e P_2 per ogni coppia.

Il principale vantaggio del metodo scan conversion rispetto ad altri algoritmi consiste nella velocità di esecuzione: il calcolo infatti risulta pressoché immediato anche con una considerevole quantità di curve, in quanto lineare nella dimensione del canvas e nel numero dei tratti. Tuttavia, ciò va a discapito della qualità del risultato finale, in quanto non si utilizzano le informazioni di forma delle figure per riempirne solo l'interno. Con forme aperte infatti capita sovente che sia riempita l'area tra linee di forme diverse. Inoltre, se si

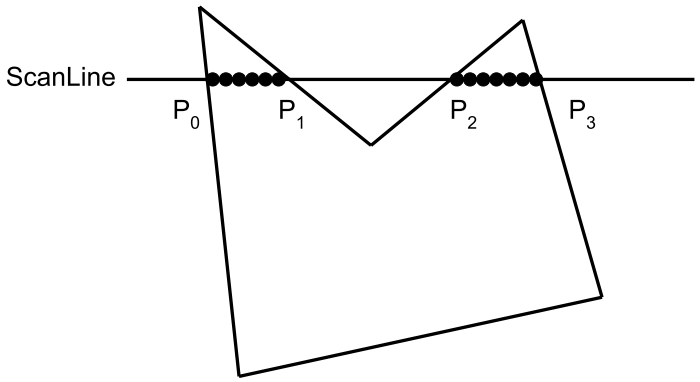


Figura 3.12: Passo del metodo scan conversion

adatta la scansione alla superficie della viewport, si ha una rappresentazione errata ogni qual volta il rettangolo che la definisce nel canvas reale interseca una qualsiasi forma. Per ovviare a tale problema, l'implementazione in Mini-System considera l'intera area reale di definizione delle curve, ma effettuandone una restrizione tra i punti di minore e maggiore coordinata, per entrambi gli assi, in modo da contenere tutte le forme definite.

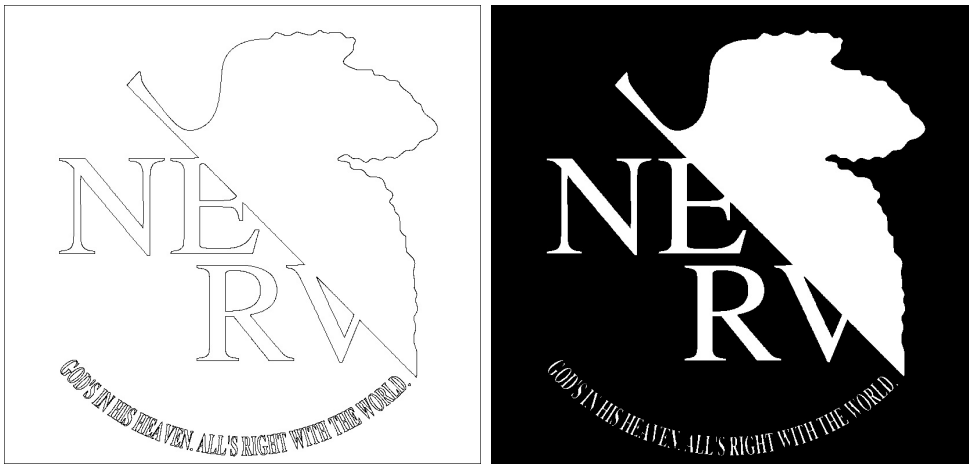


Figura 3.13: Esempio di scan conversion

Capitolo 4

Risultati

In questo capitolo saranno presentati i risultati dei processi di thresholding e tracing implementati in Mini-System su diverse tipologie di immagini e se ne effettuerà un'analisi di efficienza di rappresentazione comparando le immagini raster originali alle relative immagini vettoriali in formato SVG. In particolare, si vedranno le differenze intrinseche tra i metodi di thresholding ed i risultati ottenuti in immagini che presentano un'illuminazione della scena non uniforme; in seguito si esibiranno i risultati di tracing su illustrazioni, fotografie, disegni tecnici e documenti di vario genere. Per entrambi i processi saranno sperimentati diversi valori per i parametri disponibili.

4.1 Thresholding

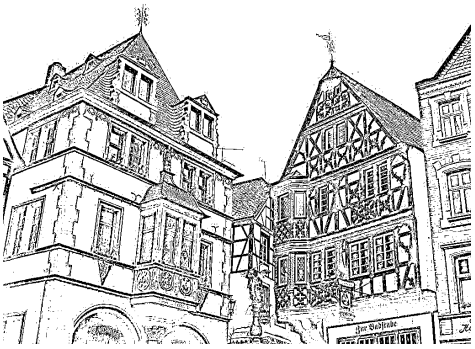
Come si è già avuto modo di notare nel terzo capitolo, i risultati dei metodi di thresholding globale applicati alle fotografie tendono a mantenere le caratteristiche di illuminazione, mentre i metodi locali agiscono molto più similarmente a dei filtri di estrazione dei contorni, adattandosi alle sfumature di grigio di un intorno per ogni pixel. Vediamo alcuni esempi nelle figure 4.1, 4.2, 4.3 (in scala di grigi), 4.4 e 4.5 (a colori).

(a) Originale

(b) Fisso (T = 128)

(c) Otsu (T = 152)

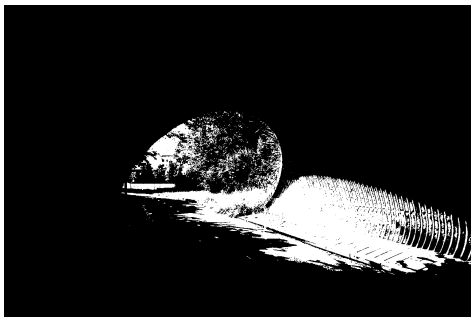
(d) Media ($N = 11,\ C = 4$)



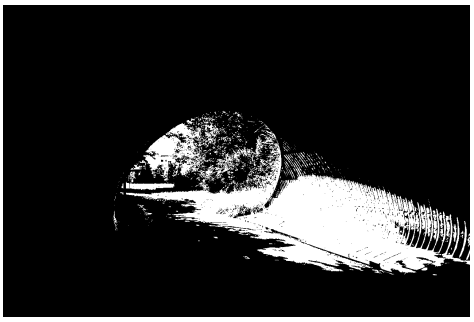
(e) Gaussiana ($N = 11,\ C = 4$)

Figura 4.1: Esempio di thresholding su un'immagine in scala di grigi con illuminazione omogenea. Entrambi i metodi globali risultano in una buona approssimazione; Otsu migliora Fisso nelle aree "bruciate" come lo spigolo a sinistra, a discapito dei dettagli. I due metodi locali riproducono i contorni delle figure; Gaussiana tende a rilevare contorni più sottili e produce meno artefatti rispetto a Media.

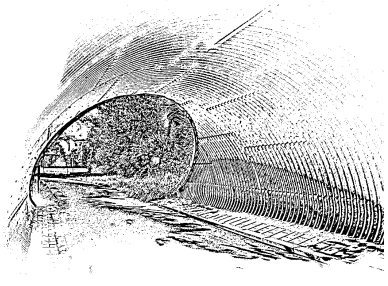
(a) Originale



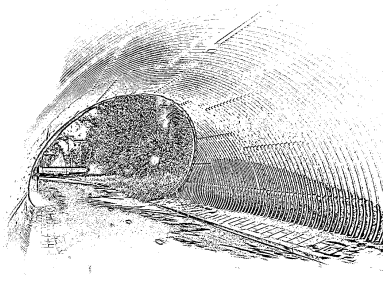
(b) Fisso (T = 128)



(c) Otsu (T = 108)



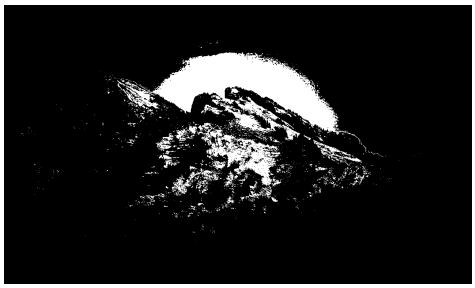
(d) Media ($N = 11,\ C = 4$)



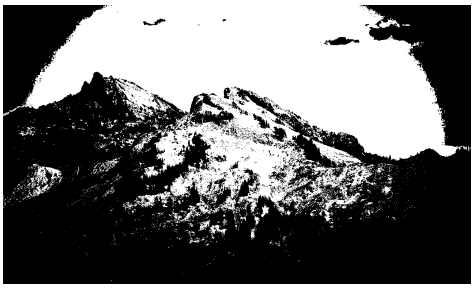
(e) Gaussiana ($N = 11,\ C = 4$)

Figura 4.2: Esempio di thresholding su un'immagine in scala di grigi con illuminazione non omogenea. I metodi globali approssimano la luminosità in modo similare, in quanto il valore di soglia di Otsu è vicino al valore di soglia di Fisso. I metodi locali rilevano i contorni con precisione; Gaussiana produce un'immagine più pulita.

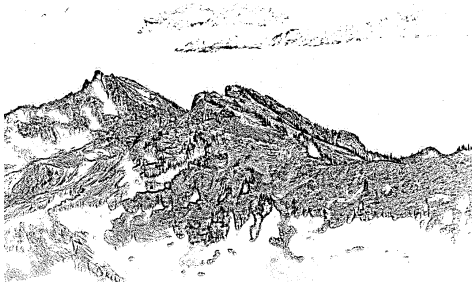
(a) Originale



(b) Fisso (T = 128)



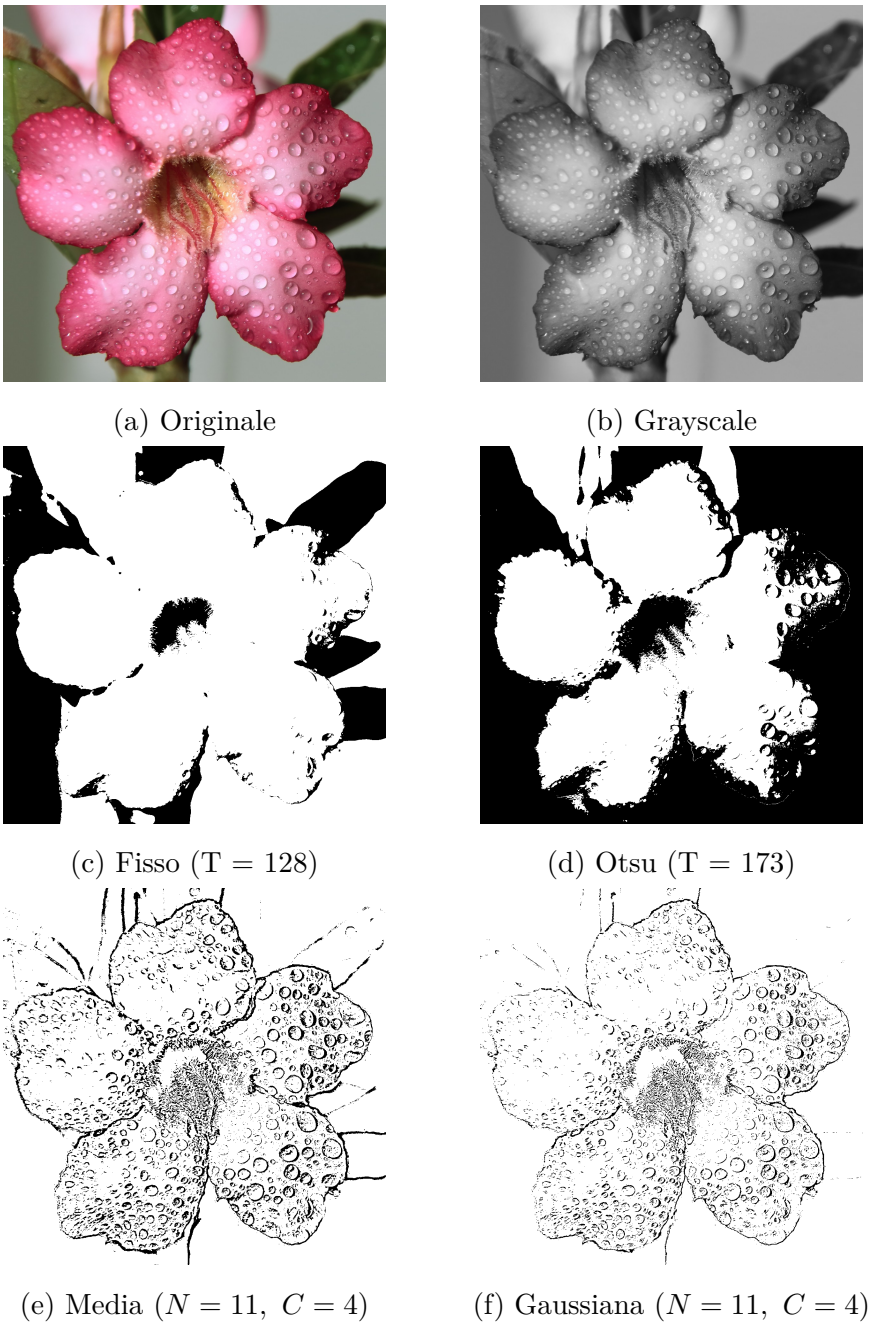
(c) Otsu (T = 62)



(d) Media ($N = 11,\ C = 4$)

(e) Gaussiana ($N = 11,\ C = 4$)

Figura 4.3: Esempio di thresholding su un'immagine in scala di grigi con illuminazione non omogenea. In questo caso Otsu riesce a produrre un risultato migliore di Fisso poiché la distribuzione dei valori di grigio è compressa verso le tonalità scure: il valore di soglia scelto e più distante dal valore del metodo fisso rispetto all'esempio precedente.

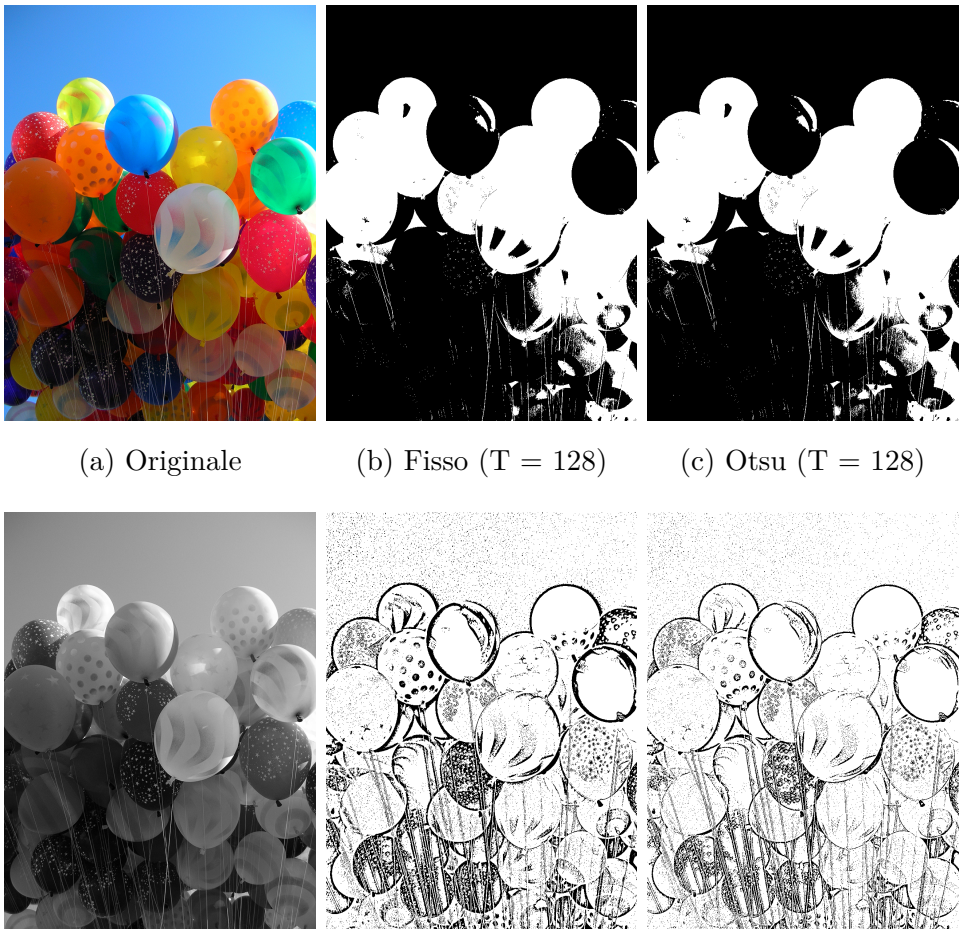


(a) Originale (b) Grayscale

(c) Fisso (T = 128) (d) Otsu (T = 173)

(e) Media ($N = 11,\ C = 4$) (f) Gaussiana ($N = 11,\ C = 4$)

Figura 4.4: Esempio di thresholding su un'immagine a colori con illuminazione parzialmente omogenea. In Fisso si distingue solo una forma approssimativa del fiore, mentre Otsu produce un'approssimazione migliore, distinguendo anche alcune gocce nella parte destra dell'immagine. Entrambi i metodi locali rilevano ottimamente i contorni.

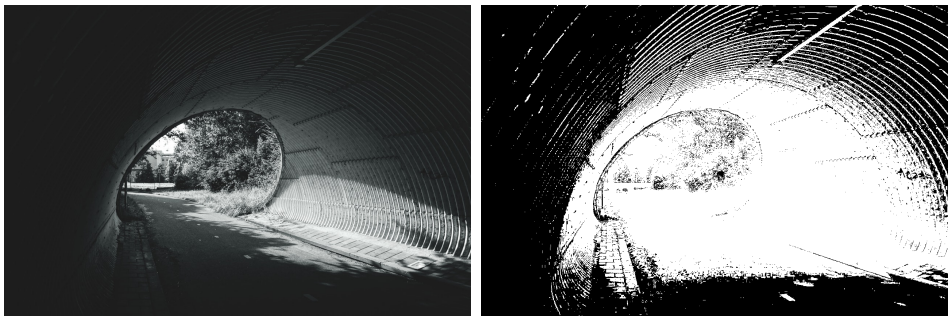


(a) Originale (b) Fisso (T = 128) (c) Otsu (T = 128)

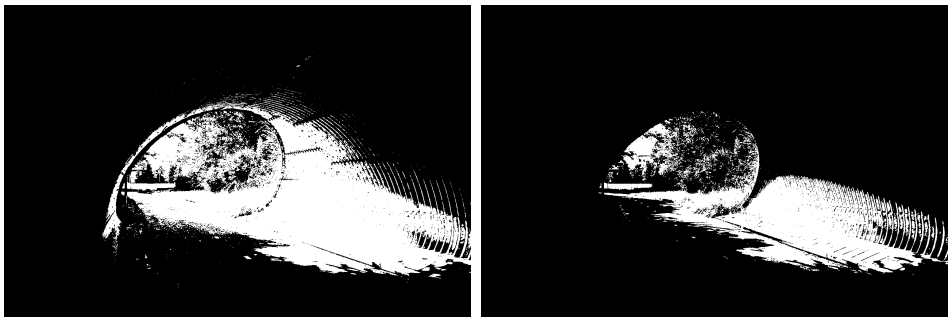
(d) Grayscale (e) Media ($N = 11,\ C = 4$) (f) Gaussiana ($N = 11,\ C = 4$)

Figura 4.5: Esempio di thresholding su un'immagine a colori con illuminazione non omogenea. I due metodi globali producono il medesimo risultato, in quanto Otsu calcola lo stesso valore di soglia di Fisso. I metodi locali definiscono una buona approssimazione dei contorni.

Vediamo ora come i parametri di thresholding influenzano i risultati ottenuti. Per quanto riguarda il valore di soglia per il metodo fisso, si ha banalmente che valori maggiori produrranno immagini più scure, mentre valori minori risulteranno in immagini più "bruciate". Riprendiamo l'esempio della figura 4.2 in figura 4.6.



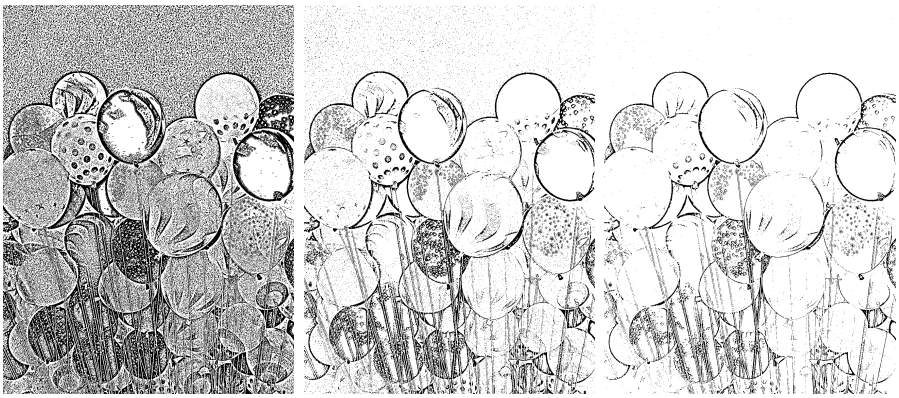
(a) Originale (b) Fisso (T = 32)



(c) Fisso (T = 64) (d) Fisso (T = 128)

Figura 4.6: Test sul parametro di soglia per il thresholding a valore fisso

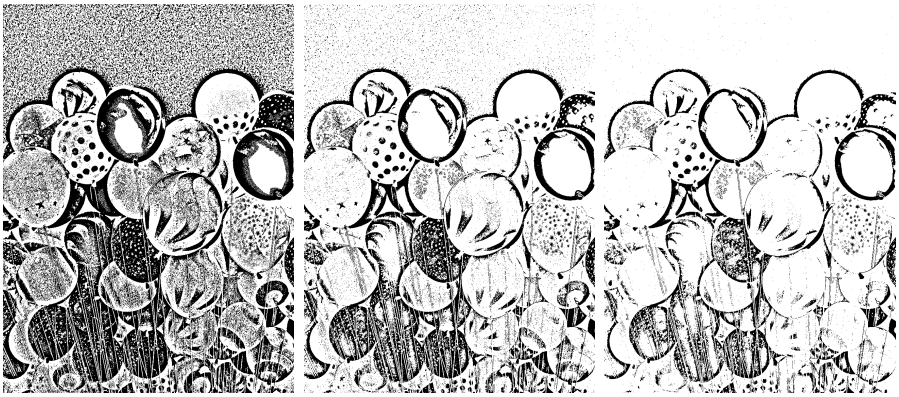
Il parametro `Local delta` per i metodi locali funge da filtro per il rumore: valori maggiori causano una riduzione del rumore, ma allo stesso tempo tendono ad "erodere" i dettagli dei bordi. Il parametro `Kernel size` regola la dimensione dell'intorno sul quale si applica il passo di convoluzione per ogni pixel. Valori maggiori tendono a inspessire i contorni con un effetto "alone" e generano più rumore. I valori ottimali, oltre che dall'algoritmo scelto, dipendono dalla composizione dell'immagine, dalla distribuzione dei valori di grigio, dai gradienti nelle sfumature e in ultimo dal risultato che si vuole ottenere. In figura 4.7 sono rappresentati i risultati del metodo con kernel gaussiano applicato alla stessa immagine in figura 4.5, con diversi valori dei parametri per i metodi locali.



(a) Gaussiana ($N = 11,\ C = 0$)

(b) Gaussiana ($N = 11,\ C = 4$)

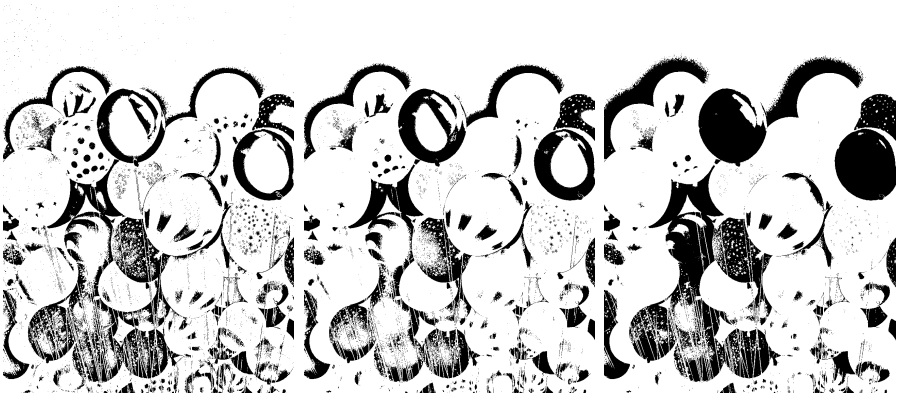
(c) Gaussiana ($N = 11,\ C = 8$)



(d) Gaussiana ($N = 33,\ C = 0$)

(e) Gaussiana ($N = 33,\ C = 4$)

(f) Gaussiana ($N = 33,\ C = 8$)



(g) Gaussiana ($N = 63,\ C = 8$)

(h) Gaussiana ($N = 127,\ C = 16$)

(i) Gaussiana ($N = 255,\ C = 16$)

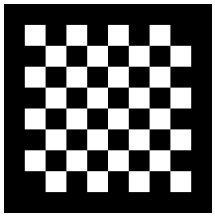
Figura 4.7: Test sui parametri per il thresholding locale

4.2 Tracing

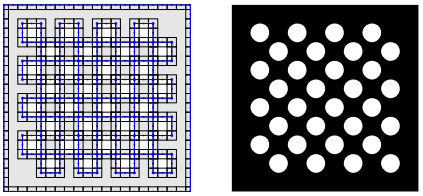
In questa sezione saranno presentati esempi degli effetti dei diversi parametri di Potrace sull'output e si includeranno i risultati del tracing eseguiti da Mini-System su diverse tipologie di immagine. Le immagini raffiguranti i risultati a seguire sono tutte immagini vettoriali in formato SVG.

4.2.1 Test parametri

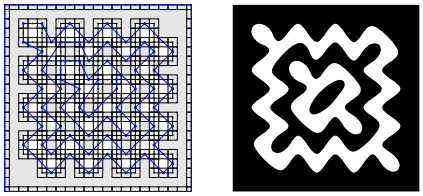
turnpolicy



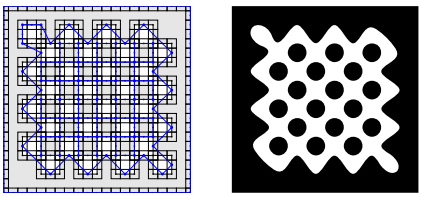
(a) Originale ($20 \times 20px$)



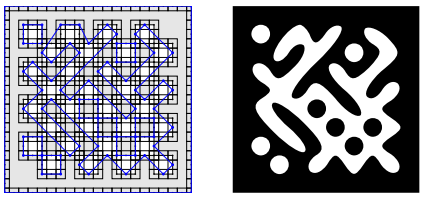
(b) black, left, majority



(c) right, minority



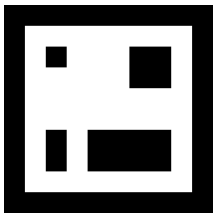
(d) white



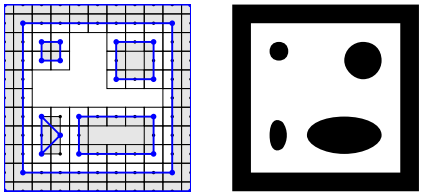
(e) random

Figura 4.8: Test sul parametro turnpolicy. A sinistra il poligono ottimale, a destra il risultato in formato vettoriale. Si può notare come ad esempio la politica black tenda effettivamente a connettere le zone nere, mentre la politica white le zone bianche.

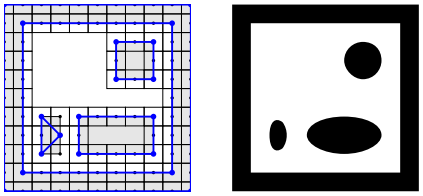
turdsize



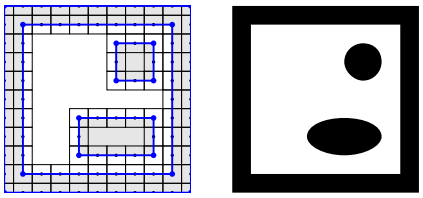
(a) Originale ($8 \times 8px$)



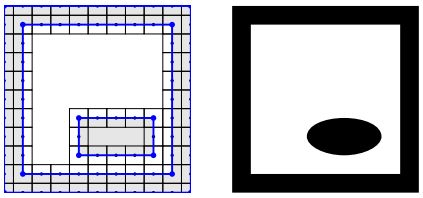
(b) turdsize = 0



(c) turdsize = 1



(d) turdsize = 2



(e) turdsize = 4

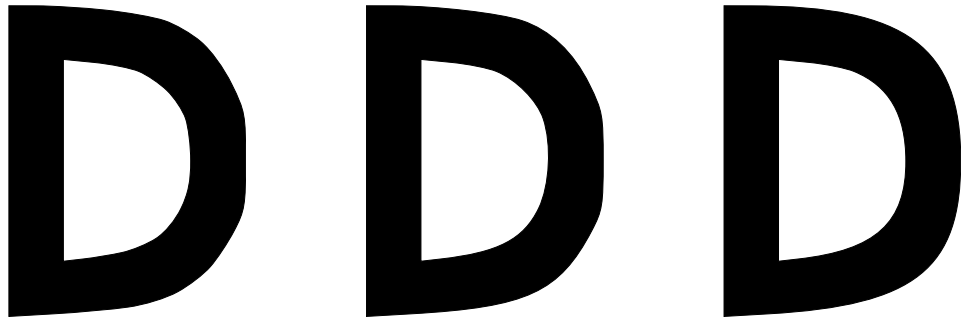
Figura 4.9: Test sul parametro turdsize. A sinistra il poligono ottimale, a destra il risultato in formato vettoriale. Si può notare come le figure con area maggiore del valore di soglia vengano escluse.

alphamax

(a) $\alpha_{max} = 0.0$ (b) $\alpha_{max} = 0.6$ (c) $\alpha_{max} = 1.0$ (d) $\alpha_{max} = 1.3$

Figura 4.10: Test sul parametro alphamax. Si noti come valori minori aumentano il numero di spigoli e valori maggiori rendano le curve più morbide.

opttolerance

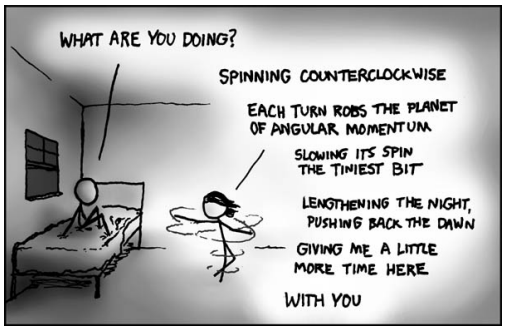


(a) $\varepsilon = 0.0$ (b) $\varepsilon = 0.2$ (default) (c) $\varepsilon = 1.0$

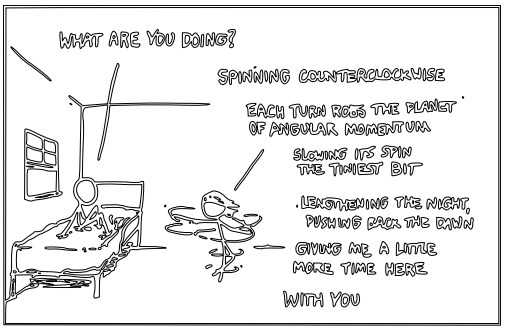
Figura 4.11: Test sul parametro opttolerance. Si noti come l'ottimizzazione produca di riflesso curve più regolari.

4.2.2 Risultati del tracing

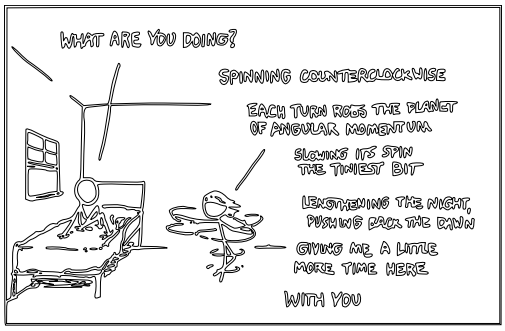
Vediamo ora i risultati del tracing eseguiti da Mini-System ed esportati in formato SVG direttamente dal software. Verranno presentate diverse tipologie di immagini, quali disegni, fotografie e scannerizzazioni di documenti. Notare che le strutture delle curve in Mini-System non comprendono l'informazione per il filling, di conseguenza le immagini vettoriali esportate visualizzano unicamente i contorni.



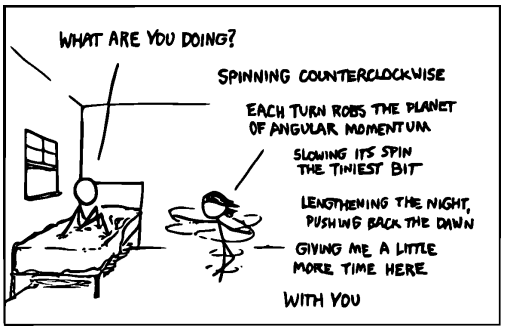
(a) Originale



(b) Local delta= 4.0 (default)



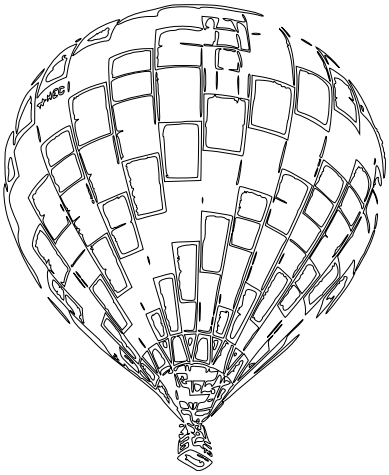
(c) Local delta= 5.0



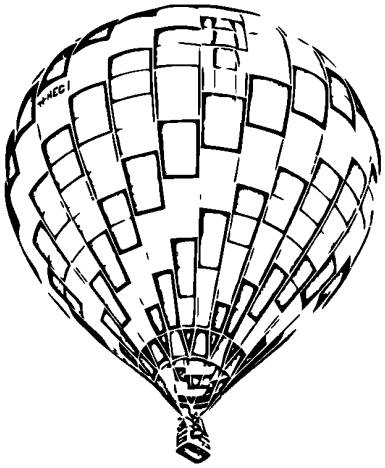
(d) Scan conversion

Figura 4.12: Esempio di tracing di una vignetta di xkcd con thresholding gaussiano; (c) è ottenuto impostando il parametro Local delta = 5.0 per rimuovere i punti superflui presenti in (b); (d) è uno screenshot post tracing con il riempimento tramite scan conversion attivo.

(a) Originale



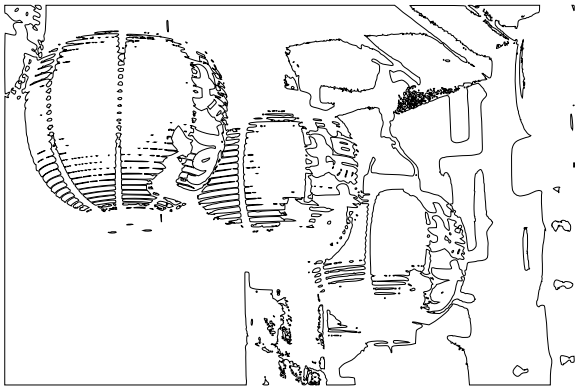
(b) Vettoriale



(c) Scan conversion

Figura 4.13: Esempio di tracing di una fotografia con thresholding gaussiano; i parametri utilizzati sono turnpolicy = black e turdsize = 6 per il tracing e quelli di default per il thresholding; (c) è uno screenshot post tracing con il riempimento tramite scan conversion attivo.

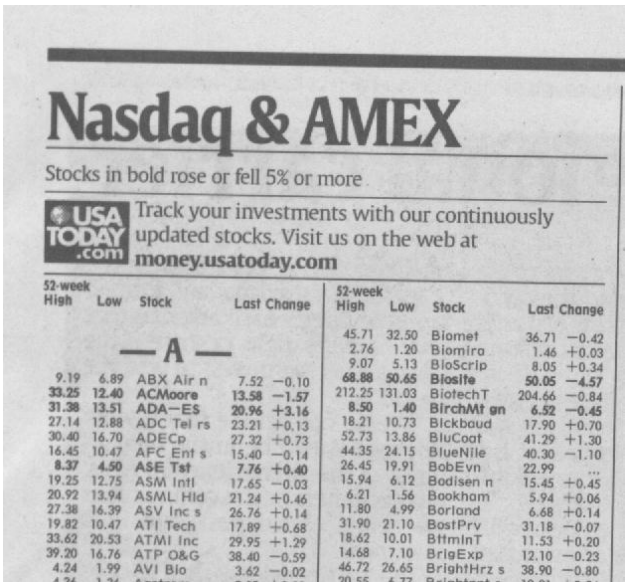
(a) Originale



(b) Otsu

(c) Scan conversion

Figura 4.14: Esempio di tracing di una fotografia con il metodo di Otsu; (c) è uno screenshot post tracing con il riempimento tramite scan conversion attivo.



Nasdaq & AMEX

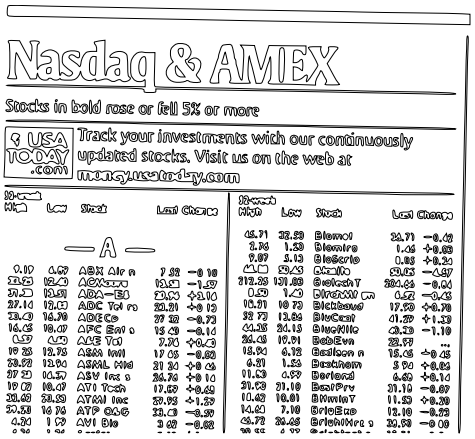
Stocks in bold rose or fell 5% or more

USA TODAY .com Track your investments with our continuously updated stocks. Visit us on the web at money.usatoday.com

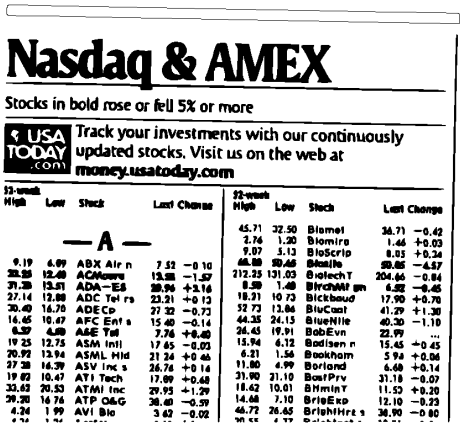
52-week High	Low	Stock	Last	Change
		— A —		
9.19	6.89	ABX Air n	7.52	−0.10
33.25	12.40	ACMoore	13.58	−1.57
31.38	13.51	ADA—ES	20.96	+3.16
27.14	12.88	ADC Tel rs	23.21	+0.13
30.40	16.70	ADECp	27.32	+0.73
16.45	10.47	AFC Ent s	15.40	−0.14
8.37	4.50	ASE Tst	7.76	+0.40
19.25	12.75	ASM Intl	17.65	−0.03
20.92	13.94	ASML Hld	21.24	+0.46
27.38	16.39	ASV Inc s	26.76	+0.14
19.82	10.47	ATI Tech	17.89	+0.68
33.62	20.53	ATMI Inc	29.95	+1.29
39.20	16.76	ATP O&G	38.40	−0.59
4.24	1.99	AVI Bio	3.62	−0.02

52-week High	Low	Stock	Last	Change
45.71	32.50	Biomet	36.71	−0.42
2.76	1.20	Biomira	1.46	+0.03
9.07	5.13	BioScrip	8.05	+0.34
68.88	50.65	Biosite	50.05	−4.57
212.25	131.03	BiotechT	204.66	−0.84
8.50	1.40	BirchMt gn	6.52	−0.45
18.21	10.73	Bickbaud	17.90	+0.70
52.73	13.86	BluCoat	41.29	+1.30
44.35	24.15	BlueNile	40.30	−1.10
26.45	19.91	BobEvn	22.99	...
15.94	6.12	Bodisen n	15.45	+0.45
6.21	1.56	Bookham	5.94	+0.06
11.80	4.99	Borland	6.68	+0.14
31.90	21.10	BostPrv	31.18	−0.07
18.62	10.01	BtmInT	11.53	+0.20
14.68	7.10	BrigExp	12.10	−0.23
46.72	26.65	BrightHrz s	38.90	−0.80

(a) Originale

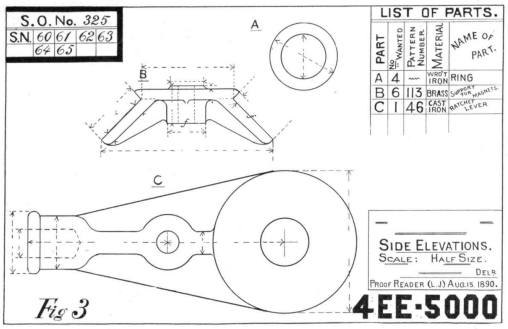


(b) Otsu

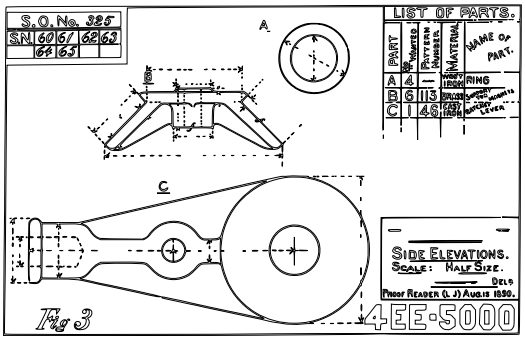


(c) Scan conversion

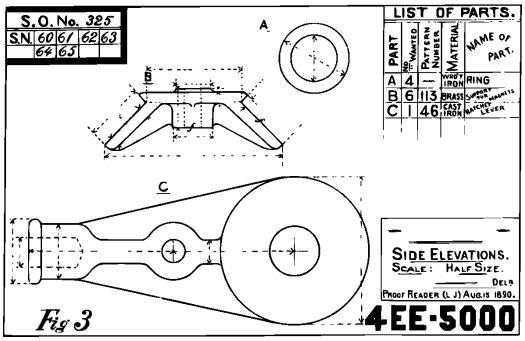
Figura 4.15: Esempio di tracing della scannerizzazione di un documento con il metodo di Otsu; (c) è uno screenshot post tracing con il riempimento tramite scan conversion attivo.



(a) Originale



(b) Otsu



(c) Scan conversion

Figura 4.16: Esempio di tracing di un disegno tecnico con il metodo di Otsu; (c) è uno screenshot post tracing con il riempimento tramite scan conversion attivo.

4.3 Comparazione pesi

Come nota finale, compariamo il peso in memoria tra le immagini raster originali e le immagini vettoriali derivanti dal tracing. Oltre alle ottimizzazioni effettuate sulle curve sia da Potrace che dalle funzioni di gestione delle curve MD-spline, Mini-System utilizza in fase di salvataggio in SVG delle tecniche di compressione che sfruttano la specifica del linguaggio. Ad esempio, nella descrizione di un `path` è possibile omettere il delimitatore precedente ad una coordinata (spazio o virgola) se questa è preceduta da una lettera o se è negativa (il segno meno agisce da delimitatore). In più, si è scelto di limitare a 6 il numero di cifre significative nella memorizzazione delle coordinate dei punti di controllo delle curve.

Nonostante queste accortezze, quasi la totalità delle immagini vettoriali prodotte in fase di testing hanno un peso sensibilmente maggiore delle relative immagini raster. Tale fenomeno è dovuto principalmente a quattro fattori:

1. le immagini su cui si è eseguito il tracing sono in formato JPEG, la cui efficienza in peso è direttamente correlata alla perdita di qualità in fase di compressione;

2. le immagini raster sono memorizzate a livello di file in rappresentazione binaria, mentre SVG è a tutti gli effetti un formato testuale;

3. si sono utilizzate immagini relativamente piccole, per le quali l'approssimazione vettoriale non produce risultati ottimali e tali che, come conseguenza del punto precedente, il rapporto tra la descrizione vettoriale e la rappresentazione come matrice di pixel è a favore di quest'ultima;

4. il formato SVG supporta solo curve di Bézier al massimo cubiche, vincolando curve di grado maggiore o definite in spazio multi-degree alla conversione in tale rappresentazione.

Riassumiamo i pesi e il loro rapporto per i due tipi di immagine:

Figura		Raster (JPEG)	Vettoriale (SVG)	Ratio
Vignetta	4.12	44.2 KB	117 KB	2.6
Fotografia mongolfiera	4.13	130 KB	131 KB	1
Fotografia lanterne	4.14	192 KB	284 KB	1.5
Scannerizzazione	4.15	71 KB	385 KB	5.4
Disegno tecnico	4.16	114 KB	215 KB	1.9

Per un confronto equo, un formato vettoriale, la cui caratteristica peculiare è il mantenimento di alta qualità a qualsiasi risoluzione, dovrebbe perlomeno essere confrontato con immagini bitmap o con formati che prevedono compressione senza perdita di informazione, come PNG. In tal caso, si ha che il file SVG risultante è generalmente comparabile o persino più leggero dell'immagine raster originale. Ad esempio, utilizzando alcune delle immagini precedenti nel formato originale PNG, si hanno i seguenti risultati:

Figura		Raster (PNG)	Vettoriale (SVG)	Ratio
Vignetta		90 KB	119 KB	1.3
Fotografia mongolfiera		717 KB	100 KB	0.14
Disegno tecnico		381 KB	215 KB	0.56
Logo	3.13	54 KB	43 KB	0.8

Il peso diverso di alcuni risultati rispetto alla precedente tabella è dovuto alla maggiore qualità delle immagini in input. Si può quindi notare come un formato vettoriale come SVG, nonostante le limitazioni correlate, possa essere generalmente più efficiente di un formato raster senza perdita di qualità alla stessa risoluzione, con il vantaggio del mantenimento della qualità anche a risoluzioni maggiori e della possibilità di attuare modifiche a livello delle forme, non attuabile con immagini raster.

Conclusioni

In questa tesi si è presentato il lavoro di integrazione della libreria di tracing Potrace nel software di disegno vettoriale Mini-System, in modo da poter ampliare le possibilità di sperimentazione delle curve MD-spline nell'ambito della grafica vettoriale. Ciò ha compreso la conversione dei formati di descrizione delle curve e la gestione delle immagini in input, con conversione in scala di grigi e thresholding. Oltre a ciò, sono stati introdotti diversi miglioramenti di interfaccia e gestione input/output, compresa una libreria sperimentale per il disegno dell'interfaccia utente.

I risultati mostrati evidenziano l'idoneità del software e dei paradigmi di definizione delle forme su cui si basa all'utilizzo per la grafica vettoriale. Tuttavia, il campo teorico delle curve a tratti a grado variabile è ancora relativamente giovane e ne mancano ad oggi adozioni su vasta scala, sia a livello di software che a definizione di formati di file che le supportino. Tale situazione apre la porta a diversi sviluppi futuri.

Innanzitutto, si lascia Mini-System come piattaforma per sperimentare ulteriori e più efficaci algoritmi di ottimizzazione delle curve mediante le definizioni MD-spline. Inoltre, siccome gli attuali formati di immagine vettoriale presentano numerose limitazioni sulla definizione delle curve, un passo importante della grafica vettoriale si potrebbe concludere proprio tramite la definizione di un nuovo formato che supporti lo spazio multi-degree. Si andrebbero così a coniugare i vantaggi di ottimizzazione della rappresentazione propri delle MD-spline alle caratteristiche di flessibilità e praticità propri del mondo della grafica vettoriale.

Bibliografia

[1] G. Casciola. “Dispensa di Calcolo numerico: metodi numerici per il calcolo”, 2016.

[2] C.V. Beccari, G. Casciola, S. Morigi. “Dispense del corso di Analisi Numerica e Modellazione Geometrica”, 2017.

[3] A. Benetti. “Modellazione geometrica con MD-spline”, 2016.

[4] C. V. Beccari, G. Casciola, S. Morigi. “On multi-degree splines”, Comput. Aided Geom. Des. 58 (2017), 8–23, Sep. 2017.

[5] D. Toshniwal, H. Speleers, R. R. Hiemstra, T. J. R. Hughes. “Multi-degree smooth polar splines: A framework for geometric modeling and isogeometric analysis”, Comput. Methods Appl. Mech. Eng. 316 (2017), 1005–1061, Nov. 2016.

[6] “SVG: Scalable Vector Graphics”, MDN Web Docs, `https://developer.mozilla.org/en-US/docs/Web/SVG`.

[7] P. Selinger. “Potrace: a polygon-based tracing algorithm”, Sep. 2003.

[8] P. Selinger. “Potrace Library API”, 2001-2017.

[9] “Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB”, IEC 61966-2-1:1999, Oct. 1999.

[10] "Parameter values for the HDTV standards for production and international programme exchange", Rec. ITU-R BT.709-4, Mar. 2000.

[11] N. Otsu. "A Threshold Selection Method from Gray-Level Histograms", IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62-66, Jan. 1979.

[12] Intel Corporation (2000-2008), Willow Garage Inc. (2009), Itseez Inc. (2014-2015). OpenCV (4.0.1), cv::getGaussianKernel() : 81.

`https://github.com/opencv/opencv/blob/c9ad5779f2803dcc91a9938142209128d30b22d1/modules/imgproc/src/smooth.cpp`

[13] S. T. Birchfield. "Image Processing and Analysis", pp. 215-233, Jan. 2017.