Métodos Discretos em Computação Gráfica

Antonio Elias Fabris Luciano Silva

Computer Graphics and Applied Computational Geometry Project (CGCAP)
Instituto de Matemática e Estatística, Universidade de São Paulo
Caixa Postal 66281, 05315-970, São Paulo-SP, Brazil
aef,lucianos@ime.usp.br

1 Introdução

Um dos objetivos centrais da computação gráfica é produzir imagens a partir de um modelo, processo denominado *renderização*. Um modelo pode ser descrito como um conjunto de objetos gráficos: um objeto gráfico $\mathcal{O}$ consiste de uma família finita $\mathcal{U} = \{U_1, \ldots, U_m\}$ de subconjuntos, $U_i \subset \mathbb{R}^n$, de algum espaço euclidiano $\mathbb{R}^n$, e uma função $F : U_1 \bigcup \ldots \bigcup U_m \rightarrow \mathbb{R}^p$. A família $\mathcal{U}$ é chamada de conjunto de dados geométricos e define a forma (geometria e topologia) do objeto, e f é a função atributo do objeto que define, por exemplo, a cor e textura do objetos. A dimensão da união $U_1 \bigcup \ldots \bigcup U_m$ define a dimensão do objeto.

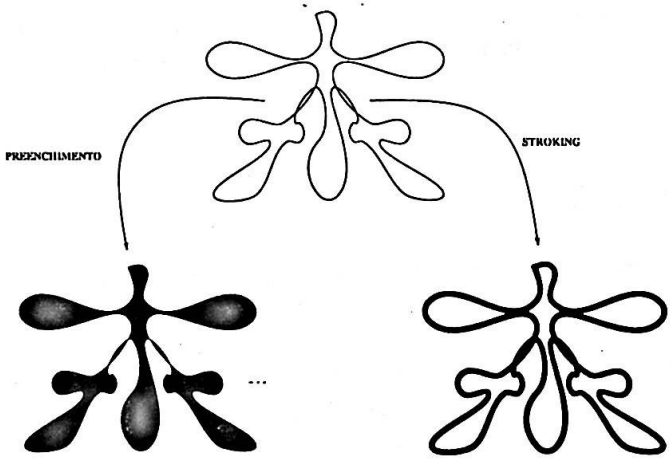
Objetos podem ser combinados de várias maneiras para produzir objetos mais complexos:

- usando operações booleanas, como união, intersecção e diferença;
- usando operações morfológicas, como a soma e subtração de Minkowski.
- uma seqüência de objetos q-dimensionais pode definir um objeto $(q+1)$-dimensional por limitação. Por exemplo, uma curva fechada simples no plano delimita e define uma região 2-dimensional.

A formação de objetos complexos através de operações booleanas tem grande importância em modelamento geométrico. Com base nas operações morfológicas e na limitação, podem ser derivadas várias outras operações importantes. Utilizando a soma de Minkowski,

obtem-se a operação de *stroking* , que consiste numa transformação do tipo caminho-região que modela a forma produzida quando um objeto(chamado *brush*) se move ao longo de uma trajetória, ou comumente, ao longo de um caminho. A partir da operação de limitação, pode-se produzir a operação de preenchimento, cujo função é preencher a região delimitada por alguma seqüência de objetos. As operações de preenchimento e *stroking* constituem um suporte para muitas aplicações gráficas importantes, tais como produção de fontes.

A figura abaixo mostra os efeitos destas duas operações:



Uma imagem é uma função $I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^k$: para cada ponto $p \in \Omega$, $I(p)$ define os atributos de p, como cor e opacidade, por exemplo. Uma classe de imagens muito importante é formada tomando $\Omega = \mathbb{Z}^2$, chamadas de imagens digitais.

No processo de transformação de um modelo em uma imagem notam-se três fases distintas:

Fase de pré-processamento: redução do modelo a uma forma mais simples.

Fase de rasterização: conversão do modelo simplificado em um *bitmap*([Fiu89]).

Fase de pós-processamento: composição dos bitmaps em uma imagem.

Uma taxonomia dos algoritmos de renderização pode ser estabelecida com base no modo em que realizam as três fases anteriores:

Algoritmos baseados no objeto: tratam objeto por objeto do modelo, encontrando os pontos da imagem que são afetados por um determinado objeto. Neste processo, realizam uma projeção explícita do objeto na imagem. Algoritmos baseados em *scan-conversion* são exemplos clássicos desta classe.

Algoritmos baseados na imagem: para cada ponto da imagem a ser produzida, verificam se os atributos deste ponto são afetados por algum objeto(ou objetos) do modelo. Caso afirmativo, calculam o valor desta contribuição. Um exemplo muito familiar desta classe é o algoritmo de *ray-tracing*.

Para exemplificar melhor esta taxonomia, considere-se as duas operações básicas descritas anteriormente: preenchimento e *stroking*. Para a operação de preenchimento, a maioria dos algoritmos de *scan-conversion* trabalham somente com regiões poligonais simples, i.e. sem auto-intersecções, requerem uma prévia linearização e redução da região original em regiões mais simples, por exemplo regiões convexas, trapezoidais ou triangulares. Esta redução pode ser encontrada, por exemplo, na implementação, feita pela Adobe, da linguagem PostScript [AdoB86]. Porém, esta fase de linearização pode levar a problemas numéricos e de robustez dos algoritmos, como os descritos em Forrest [For85] [For88] e Franklin [Fra86]. No caso especial do PostScript da Adobe, alguns exemplos de problemas podem ser encontrados em Perky [Per88] e Pol-Corthout [CorP92].

Para a operação de *stroking*, duas abordagens de *scan-conversion* são geralmente utilizadas. Uma delas consiste em rasterizar o caminho, colocando a *brush* em cada ponto do caminho rasterizado. Whitted [Whi83] e Bleser et al. [BleSM88], por exemplo, realizam este processo com a possibilidade de variação da intensidade, tamanho e forma da *brush* ao longo do caminho. Estes algoritmos foram produzidos para serem conjugados em sistemas de pintura e implementados via tabelas do tipo *look-up*. Strassmann [Str86] descreve outro tipo de algoritmo, com a possibilidade de se controlar vários outros parâmetros. Em [PosF89], Posch apresenta um algoritmo com *brush* circular que somente atualiza os pontos num arco semi-circular, perpendicular à direção do caminho, num ponto corrente do caminho. A segunda abordagem é transformar o problema de *stroking* num problema de preenchimento. Este processo é feito computando-se as equações algébricas do contorno da região de *stroking*. Porém, se a complexidade geométrica dos objetos na operação cresce, as equações tornam-se muito complexas, como descrito em Gosh e Mudur [GhoM84]. Alternativamente, caminhos e *brushes* são usualmente aproximados, em um fase de pré-processamento, por polígonos, como no sistema METAFONT de D.E. Knuth [Knu86], e o resultado da operação pode ser obtido como descrito por Guibas et al. [GuiRS83].

Já os algoritmos baseados na imagem testam a pertinência de um ponto da imagem contra um objeto, sem a necessidade de um pré-processamento prévio. Para isto, existe a necessidade da especificação de um predicado de pertinência de um ponto a um objeto[1], conhecido como teste interior/exterior(*Point Containment test*). A utilização deste predicado de pertinência para todos os pontos da imagem, implementada em software, é considerada muito lenta para o processo de renderização, como já notado por Newmann e Sproull [New79].

Poucos algoritmos baseados na imagem têm sido publicados. Forrest [For85], Tang [Tan88] e Guibas et al. [GuiRS83] descrevem algoritmos para objetos poligonais, detalhando testes de pertinência de pontos em polígonos. O tratamento de objetos mais complexos, como segmentos de curvas de Bézier polinomiais e racionais, pode ser encontrado na tese de doutoramento de Pol e Corthout [CorP92].

Estes exemplos permitem realçar uma série de vantagens dos algoritmos baseados na imagem sobre os baseados no objetos, tais como:

1. Nenhum pré-processamento do objeto é necessário: os algoritmos baseados na imagem podem lidar diretamente com objetos de alta complexidade geométrica. Como nenhum pré-processamento é realizado, não é necessário memória adicional para armazenar estruturas de dados intermediárias. Os algoritmos baseados no objeto realizam um pré-processamento do objetos, geralmente projeções seguidas de linearizações, cujas estruturas intermediárias são armazenadas em alguma ordem para facilitar a fase de rasterização. Em algumas implementações, os objetos são primeiramente reduzidos a regiões mais fáceis de serem rasterizadas, tais como segmentos poligonais(Klassen [KlaS91]).

2. *Clipping* explícito de objetos contra a imagem ou outros objetos não é necessário: simplesmente para cada ponto da imagem, ou ele é afetado ou não por um objeto(ou objetos) do modelo. Nos algoritmos baseados no objeto, o *clipping* precisa ser realizado ou no espaço do objeto ou no espaço da imagem.

3. Facilidade de adaptação com crescimento da complexidade do modelo: somente o predicado de pertinência, que testa se um ponto é afetado ou não por algum objeto, precisa ser especificado. Já nos algoritmos baseados no objeto, a complexidade do processo cresce drasticamente com o aumento da complexidade do modelo, principalmente nos processos de linearização e *clipping*.

[1] Por exemplo, testar se um ponto está na região de preenchimento ou *stroking*.

4. Dada sua simplicidade e provada robustez, os algoritmos baseados na imagem podem ser implementados em *hardware* dedicado. Implementações em hardware de algoritmos baseados no objeto necessitam de uma quantidade considerável de pré-processamento em *software*, fato que pode limitar o poder de modelamento.

5. Os algoritmos baseados na imagem são altamente indicados para implementações paralelas, uma vez que não existe uma dependência entre as computações das contribuições nos pontos da imagem.

Porém, uma das maiores desvantagens dos algoritmos baseados na imagem é a sua complexidade quadrática com respeito à resolução da imagem. Na abordagem baseada no objeto, os algoritmos de *scan-conversion* têm, usualmente, complexidade linear na resolução da imagem. Assim, o desenvolvimento de métodos que reduzam a complexidade dos algoritmos baseados na imagem é de grande interesse computacional.

Este mini-curso pretende introduzir alguns métodos discretos utilizados em algoritmos baseados na imagem, evidenciando alguns resultados importantes no problema do teste interior/exterior. Serão apresentados três problemas clássicos em computação gráfica: preenchimento de regiões, *stroking* e *antialiasing*. Para cada problema, serão mostrados fundamentos teóricos que permitem o desenvolvimento de algoritmos corretos, robustos e eficientes. Alguns algoritmos básicos e outros mais avançados, desenvolvidos pelo CGCAP, serão apresentados, atentando-se para aspectos teóricos e computacionais.

Estas notas estão organizadas da seguinte maneira:

Seção 2: descreve os pré-requisitos da Morfologia Matemática Discreta, que serão utilizados nos capítulos subseqüentes, tais como representações matemáticas de curvas discretas, distâncias discretas e conexidade, operações morfológicas básicas.

Seção 3: introduz o conceito de função de rasterização e descreve uma forma particular destas funções. Esta forma particular permitirá a construção de uma versão discreta do bem conhecido Teorema de Jordan.

Seção 4: com a versão discreta estabelecida, serão evidenciadas aplicações em preenchimento de regiões, *stroking* e *antialiasing*, evidenciando algoritmos onde robustez e eficiência desempenham papel fundamental.

Apesar da escassez de literatura no segmento de teste interior/exterior baseado na imagem, não é pretensão deste mini-curso esgotar o assunto. O leitor interessado encontrará, no final destas notas, referências relevantes da área.

2 Morfologia discreta

A morfologia discreta é uma teoria matemática importante para desenvolvimento de operações no ambiente discreto, isto é, em $\mathbb{Z}^n$. Para a introdução de uma versão discreta do Teorema de Jordan é necessário desenvolver noções geométricas e topológicas neste ambiente. Tais noções são desenvolvidas nesta seção.

2.1 Listas

Uma lista de pontos será definida como uma função de um intervalo de naturais $[0...n]$ em $\mathbb{Z}^2$, ou seja, uma lista de pontos corresponderá a uma seqüência de pontos. Listas serão utilizadas para representar curvas discretas.

Definição 2.1. *Para todo $n \in \mathbb{N}$, o conjunto de funções de $[0...n]$ em $\mathbb{Z}^2$ será denotado por $\Lambda_n = \{L : [0...n] \to \mathbb{Z}^2\}$. As funções L serão chamadas de listas de comprimento n, denotado por $\#L$.*

Usar-se-á L_i para denotar a imagem $L(i)$, ou seja, o i-ésimo ponto de L. Chamar-se-á uma lista de *fechada* se e somente se $L_0 = L_{\#L}$. O conjunto de pontos de uma lista L será denotado por $\langle L_i \rangle_{i=0}^{n}$. Em algumas definições, abusando da linguagem matemática, tal notação também será utilizada para indicar a função L que define uma lista.

Da definição anterior segue que Λ_n é equivalente ao $n+1$-produto cartesiano $(\mathbb{Z}^2)^{n+1}$.

Definição 2.2. *O conjunto de todas as listas será denotado por Λ, ou seja, $\Lambda = \bigcup_{i \in N} \Lambda_i$.*

Neste conjunto podem ser especificados operadores importantes para a formação de novas listas. O primeiro destes operadores será a concatenação:

Definição 2.3. *Sejam L^1 e L^2 duas listas de comprimento n_1 e n_2, respectivamente. L^1 e L^2 podem ser concatenadas se e somente se $L^1_{n_1} = L^2_0$. A concatenação de duas listas nestas condições será a lista $L^1 \uplus L^2$, de comprimento $n_1 + n_2$, definida por:*

$$(L^1 \uplus L^2)_i = \begin{cases} L^1_i & se\ 0 \leq i \leq n_1 \\ L^2_{i-n_1} & se\ n_1 \leq i \leq n_1 + n_2 \end{cases}$$

Observa-se facilmente que, se uma das listas tiver comprimento 0, a concatenação reduz-se à operação identidade.

Definição 2.4. *A reversão é um operador unário* $\mathcal{R} : \Lambda \to \Lambda$, *definido por:*

$$\mathcal{R}(L) = \langle L_{n-i} \rangle_{i=0}^{n}.$$

Claramente $\mathcal{R}(\mathcal{R}(L)) = L$. O próximo operador é o bem conhecido operador de diferença .

Definição 2.5. *O operador de diferença* $\Delta : \Lambda - \Lambda_0 \to \Lambda$ *é definido por:*

$$\Delta(L) = \langle L_{i+1} - L_i \rangle_{i=0}^{n-1}.$$

Como conseqüência imediata destas definições, tem-se a seguinte relação entre estes três operadores:

Teorema 2.1.

$$\mathcal{R}(L^1 \uplus L^2) = \mathcal{R}(L^2) \uplus \mathcal{R}(L^1)$$

$$\Delta(L^1 \uplus L^2) = \Delta(L^1) \uplus \Delta(L^2)$$

$$\Delta \circ \mathcal{R} = -\mathcal{R} \circ \Delta$$

2.2 Métricas discretas e listas conexas

A noção de conectividade em listas pode ser comparada com uma versão discreta de continuidade. Certas restrições de conectividade em listas permitirão obter um corolário importante do Teorema Discreto de Jordan.

Para se desenvolver uma teoria coerente de conectividade no ambiente discreto bidimensional, $\mathbb{Z}^2$ será dotado de uma estrutura métrica.

2.2.1 Estruturas métricas em $\mathbb{Z}^2$

Para se induzir uma métrica em $\mathbb{Z}^2$, utilizar-se-á a seguinte norma geral:

Definição 2.6. *Seja* $V = \{v_i : 0 \leq i \leq n\} \subset \mathbb{Z}^2$ *um sistema finito de geradores de* $\mathbb{Z}^2$, *considerado como um* $\mathbb{Z}$ *-módulo. Defina:*

$$\begin{array}{rccl} |.|_V : & \mathbb{Z}^2 & \to & \mathbb{R} \\ & P & \mapsto & \min\{\sum |a_i| : \sum a_i v_i = P\} \end{array}$$

A partir desta norma, pode ser construída uma métrica canônica, definida por:

$$\begin{aligned} d_V : \ \mathbb{Z}^2 \times \mathbb{Z}^2 &\to \mathbb{R} \\ (P, Q) &\mapsto |P - Q|_V \end{aligned}$$

Variando-se o conjunto de geradores V, as métricas correspondentes produzem as bem conhecidas formas conexas estruturais de $\mathbb{Z}^2$:

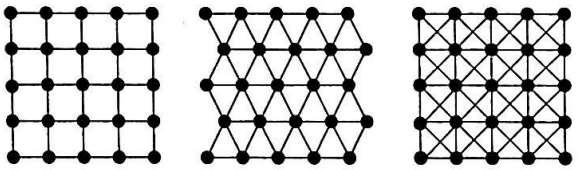


Figura 1: Formas conexas estruturais de $\mathbb{Z}^2$: 4-conexa,6-conexa e 8-conexa.

As métricas correspondentes a estas formas são induzidas, respectivamente, pelas seguintes normas:

- $|.|_{V^4}$, colocando-se $V^4 = \{(1,0),(0,1)\}$
- $|.|_{V^6}$, colocando-se $V^6 = \{(1,0),(1,1),(0,1)\}$
- $|.|_{V^8}$, colocando-se $V^8 = \{(1,0),(1,1),(0,1),(-1,1)\}$

Na próxima seção, freqüentemente $\mathbb{Z}^2$ será imerso em $\mathbb{R}^2$. Esta imersão será a canônica, a menos que explicitado o contrário. Para manter uma compatibilidade de normas entre estes dois espaços, o seguinte teorema relaciona a norma geral $\mathbb{Z}^2$, definida anteriormente, com qualquer norma adotada em $\mathbb{R}^2$.

Teorema 2.2. *Seja* $|.|$ *alguma norma em* $\mathbb{R}^2$. *Definindo* $e = \max\{|(1,0)|_V, |(0,1)|_V\}$, $v = \max\{|V_i|\}$ *e* $\rho = \max\{|R^x| + |R^y| : R \in \mathbb{R} \text{ e } |R| = 1\}$, *tem-se:*

$$e^{-1}\rho^1|P|_V \leq |P| \leq v|P|_V$$
$$v^{-1}|P| \leq |P|_V \leq e\rho|P|$$

2.3 Listas conexas

A norma geral, definida na seção anterior, pode ser extendida para o domínio das listas, da seguinte maneira:

Definição 2.7. *Seja L uma lista de comprimento n. Define-se a norma de L(com relação a um conjunto de geradores V do $\mathbb{Z}$-módulo $\mathbb{Z}^2$) por:*

$$|L|_V = \max_{0 \leq i \leq n} \{|L_i|_V\}.$$

A partir desta norma, tem-se a noção de listas conexas:

Definição 2.8. *Uma lista é m-conexa, $m \in \{4, 6, 8\}$, se $|\Delta(L)|_{V^m} \leq 1$.*

A figura 2 mostra exemplos destas listas:

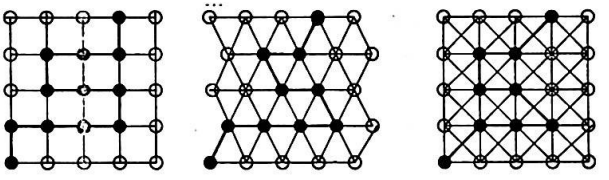


Figura 2: Listas 4-conexa, 6-conexa e 8-conexa.

Advinda desta definição, tem-se a noção de região conexa:

Definição 2.9. *Uma região R (subconjunto finito de $\mathbb{Z}^2$) é dita m-conexa, $m \in \{4, 6, 8\}$, se para todo ponto P, $Q \in R$ existe uma lista m-conexa L, tal que $\{L_i\}_{i=0}^{\#L} \subset R$, $L_0 = P$ e $L_{\#L} = Q$.*

2.4 Operadores de Minkowski

Seja $\mathbf{E} = \mathbb{Z}^2$, munido da estrutura usual de grupo abeliano. Define-se em $\mathbf{E}$ dois operadores importantes, chamados *adição* e *subtração de Minkowski*:

Definição 2.10. *Sejam X e A subconjuntos de $\mathbf{E}$. Define-se a adição $X \oplus A$ e subtração $X \ominus A$ de Minkowski de X por A, por:*

- $X \oplus A = \{x \in \mathbf{E} : (A^t + x) \bigcap X \neq \emptyset\}$

- $X \ominus A = \{x \in \mathbf{E} : A_x \subseteq X\}$.

onde A_x representa o conjunto $A_x = \{a + x : a \in A\}$ e, A^t, o conjunto $A^t = \{a \in \mathbf{E} : -a \in A\}$.

Usualmente a adição de Minkowski é chamada de *dilatação* e, a subtração, de *erosão*. O conjunto A recebe a designação de *elemento estruturante*.

Jean Serra [Ser82] caracteriza de modo mais algébrico estas operações, dando as seguintes definições equivalentes.

- $X \oplus A = \bigcup_{a \in A} X_a$
- $X \ominus A = \bigcap_{a \in A} X_{-a}$

Um enfoque mais geométrico para estas operações pode ser encontrado em Guibas et. al. [GuiRS83], que trabalham com dilatação e erosão com figuras geométricas.

Estas operações permitirão formalizar noções de coerência em rasterização, das quais serão derivados algoritmos, de complexidade quasi-linear na resolução, para algumas operações básicas de rasterização (preenchimento de regiões e *stroking*). Nestas noções de coerência, algumas propriedades básicas destas operações serão necessárias, as quais são listadas a seguir.

Propriedade 2.1. *A soma é um operador monotônico, isto é,*

$$A \subset B \Rightarrow (A \oplus C) \subset (B \oplus C).$$

Propriedade 2.2. *A soma distribui sobre uniões, isto é,*

$$(A \cup B) \oplus C = (A \oplus C) \cup (B \oplus C).$$

Propriedade 2.3. *A soma é um operador associativo, isto é,*

$$(A \oplus B) \oplus C = A \oplus (B \oplus C).$$

Propriedade 2.4. *A soma e a subtração estão relacionadas pela seguinte inclusão*

$$(A \ominus B) \oplus C \subset (A \oplus B) \ominus C.$$

A monotonicidade será utilizada em várias situações de limitação de regiões. A distributividade permitirá construir processos de recursão. A associatividade levará a transformações de elementos estruturantes na operação de *stroking* e, a relação entre as duas operações, para refinamento de testes de coerência.

3 Versão discreta do Teorema de Jordan

Os resultados preliminares necessários para o estabelecimento e prova de uma versão discreta do Teorema de Jordan são introduzidas neste capítulo. Esta versão discreta servirá de base para o desenvolvimento de algoritmos eficientes e corretos para testes de pertinência interior/exterior no problema de preenchimento de regiões delimitadas por curvas discretas.

3.1 Números de rotação

A noção de interior e exterior de uma curva(não necessariamente discreta) é baseada no conceito de *número de rotação* (winding number). Existem, essencialmente, dois modos diferentes de se definir o número de rotação: uma versão analítica, que emprega uma integral complexa de linha, e outra, conhecida como versão geométrica, que conta o número de intersecções diretas com um raio.

Usualmente, a definição que melhor se adequa ao contexto de algum problema de pertinência é escolhida. Nas próximas seções será provado que, sob certas restrições, estas duas versões são equivalentes. A ferramenta matemática utilizada para se estabelecer esta eqüivalência será a *Teoria das Funções-CW.*

3.1.1 Teoria de *cross weight* (CW)

A versão geométrica do número de rotação é baseada na noção de *cross weight(CW).* Informalmente, dadas duas curvas em $\mathbf{R}^2$, o CW destas duas curvas é igual ao número de cruzamentos com orientação esquerda-para-direita menos o número de cruzamentos com orientação direita-para-esquerda, com as definições usuais da Teoria dos Nós ([BurZ85], [Kau87]).

No âmbito das curvas discretas, será definida a noção de função CW. Esta definição será mais geral do que o necessário, praticamente sem nenhuma relação com a versão

analítica do número de rotação. Numa seção posterior, será derivada uma função CW específica para este fim.

Definição 3.1. *Uma função $C : \Lambda \times \Lambda \to \mathbb{Z}$ é chamada de função CW se satisfaz às seguintes condições:*

Distribuição sob concatenação:

$$C(L^1 \uplus L^2, L^3) = C(L^1, L^3) + C(L^2, L^3)$$

$$C(L^1, L^2 \uplus L^3) = C(L^1, L^2) + C(L^1, L^3)$$

Inversão de sinal sob reversão:

$$C(\mathcal{R}(L^1), L^2) = C(L^1, \mathcal{R}(L^2)) = -C(L^1, L^2)$$

Nulidade sob triângulos: *Se L^1 e L^2 são listas fechadas de comprimento 3, então:*

$$\cdots \quad C(L^1, L^2) = 0.$$

Utilizando-se indução no comprimento das listas e as definições anteriores, pode ser provado o seguinte:

Teorema 3.1. *Sejam L^1 e L^2 duas listas fechadas. Então $C(L^1, L^2) = 0$.*

Na prova do Teorema discreto de Jordan, este teorema representa um papel fundamental.

Observa-se, também, que este teorema induz uma relação de equivalência em $\mathbb{Z}^2$:

Corolário 3.1. *Dada uma função CW C e uma lista fechada L^1, a relação $P \equiv Q \Leftrightarrow C(L^1, L^2) = 0$, para toda L^2 com $L^2_0 = P$ e $L^2_{\#L^2} = Q$, é uma eqüivalência em $\mathbb{Z}^2$.*

3.1.2 Versão analítica

A versão analítica do número de rotação emprega uma integral complexa sobre um caminho γ, e constitui uma ferramenta poderosa na Teoria de Funções Analíticas ([Pal91]) e na Teoria de Intersecção em Topologia Diferencial ([GuiP74]):

Definição 3.2. *O número de rotação $w(\gamma, P)$, de um caminho γ, fechado e suave por partes*[2]*, com respeito a P, é dado pela fórmula:*

$$w(\gamma, P) = \frac{1}{2\pi i} \int_{\gamma} \frac{dz}{z - P},$$

onde está admitido $\mathbb{R}^2$ é imerso em $\mathbb{C}$ para o cálculo da integral complexa.

Esta definição possui uma interpretação geométrica bastante simples. Considere-se $\gamma : [a, b] \to \mathbb{C}$ um caminho, fechado e suave por partes, e um ponto $P \in \mathbb{C} - \{\gamma\}$. Fixe-se um círculo $K = K(P, r)$ centrado em P e defina-se a projeção radial de γ em K por $\beta(t) = P + r\left[\frac{\gamma(t)-P}{|\gamma(t)-P|}\right]$, conforme mostrado na Figura 3.

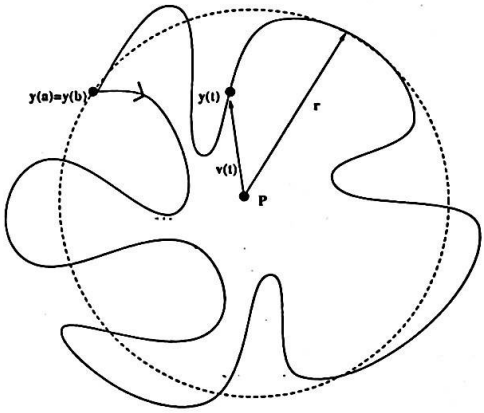


Figura 3: Interpretação geométrica do número de rotação.

Denotando-se por $v(t)$ o vetor radial de P até $\beta(t)$, pode-se relacionar o número de revoluções deste vetor, em torno do ponto P, com a integral que define o número de rotação pelo seguinte resultado:

Teorema 3.2. *O número de rotação $w(\gamma, P)$ conta o número de revoluções completas - o número de revoluções positivas menos o número de revoluções negativas - realizadas por $v(t)$ quando t varia de a para b.*

[2]Isto significa que, tomando-se $\gamma : [a, b] \to \mathbb{C}$, então é possível subdividir $[a, b]$ em um número finito de partes e a restrição de γ a cada uma destas partes existe e, sua derivada complexa $\gamma'(t)$, é contínua e nunca se anula na parte considerada.

Com a definição analítica, pode-se provar o seguinte resultado:

Teorema 3.3. *Seja γ um caminho fechado, suave por partes no plano complexo e $U = \mathbb{C} - \gamma$. Então:*

- *$w(\gamma, P)$ permanece constante quando P varia sobre toda componente de U*
- *$w(\gamma, P) = 0$ se P pertence a uma componente ilimitada de U*
- *Quando γ é simples, i.e. sem auto-intersecções, $w(\gamma, P) = 1$, para todo P na componente limitada de U, ou $w(\gamma, P) = -1$ para tais pontos P.*

A partir deste resultado, já se tem uma forma analítica para se testar se um ponto pertence ou não ao interior de alguma região limitada por um caminho nas condições anteriores. Apesar de simples, a implementação computacional deste resultado é impraticável, pois envolve o cálculo de uma integral complexa: mesmo por métodos numéricos, a tarefa não é das mais fáceis.

Para interpor esta dificuldade, tem-se a versão geométrica do número de rotação, cuja principal característica reside na facilidade de implementação.

3.1.3 Versão geométrica

Para definir uma versão mais implementável do número de rotação e estabelecer uma relação com a versão analítica, será desenvolvida um função CW especial. Para tal fim, fixar-se-á algumas notações:

Definição 3.3.

- *O produto vetorial de P e Q será denotado por $P \times Q$, onde a imersão em $\mathbb{R}^3$ está implícita.*
- *A coordenada z de um ponto $P \in \mathbb{R}^3$ será denotada por P^z.*
- *Considerar-se-á a função sinal $s : \mathbb{R} \to \{-1, 0, 1\}$ como: $s(x) = -1$ se $x < 0$, $s(x) = 0$ se $x = 0$ e $s(x) = 1$, se $x > 0$.*
- *Utilizar-se-á quatro imersões, definidas por:*
  - *A imersão de pontos por $\varphi_\delta : \mathbb{Z}^2 \to \mathbb{R}^2$ por:*

$$\varphi_\delta(P) = P + \delta, \delta \in \mathbb{R}^2.$$

- *A imersão $\varphi_\delta(L)$ de uma lista $L = \langle L_0, L_1 \rangle$, de comprimento 1, como o segmento de reta parametrizado $l(t) = (1-t)L_0 + tL_1 + \delta \in \mathbb{R}^2$, para $0 \leq t \leq 1$.*
- *A imersão de $\varphi_\delta(L)$ de listas $L \in \Lambda_n$ de comprimento $n > 1$ como o polígono (não necessariamente fechado):*

$$\uplus_{0 \leq i \leq n-1} \varphi_\delta(\langle L_i, L_{i+1} \rangle)$$

onde o operador $\uplus$ denota a concatenação de segmentos de reta imersos na forma paramétrica.

- *A imersão canônica, denotada por φ. A imersão canônica da origem será denotada por φ_O.*

Com as notações desta definição, constrói-se a seguir, recursivamente, uma função que desempenhará papel fundamental no relacionamento com a versão analítica.

Definição 3.4. *Seja $\epsilon \in \mathbb{R}^2$ dado e defina-se uma função parcial $\mathcal{W}_\epsilon : \Lambda \times \Lambda \to \mathbb{Z}$ por:*

- $\mathcal{W}_\epsilon : \Lambda_0 \times \Lambda \bigcup \Lambda \times \Lambda_0 \to \mathbb{Z}$ *por:*

$$\mathcal{W}_\epsilon(L^1, L^2) = 0$$

- $\mathcal{W}_\epsilon : \Lambda_1 \times \Lambda_1 \to \mathbb{Z}$ *por:*

$$\begin{cases} \varphi(L^1) \cap \varphi_\epsilon(L^2) = \emptyset \Rightarrow \mathcal{W}_\epsilon(L^1, L^2) = 0 \\ \varphi(L^1) \cap \varphi_\epsilon(L^2) \neq \emptyset \Rightarrow \mathcal{W}_\epsilon(L^1, L^2) = s((\Delta(L^1) \times \Delta(L^2))^z) \end{cases}$$

- $\mathcal{W}_\epsilon : \Lambda_{n_1} \times \Lambda_{n_2} \to \mathbb{Z}$, *com* $n_1, n_2 > 1$, *por:*

$$\mathcal{W}_\epsilon(L^1, L^2) = \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \mathcal{W}_\epsilon(\langle L^1_i, L^1_{i+1} \rangle, \langle L^2_j, L^2_{j+1} \rangle).$$

Nota-se que, como somente uma lista é deslocada de ϵ antes de ser checado se os segmentos das listas têm um ponto em comum, uma assimetria é introduzida na função: pode ser que $\mathcal{W}_\epsilon(L^1, L^2) \neq \pm \mathcal{W}_\epsilon(L^2, L^1)$.

O próximo passo será mostrar que a função parcial definida anteriormente é uma função CW:

Teorema 3.4. *$\mathcal{W}_\epsilon$ distribui sob concatenações e muda de sinal sob reversões.*

...

A prova deste teorema é simples, pois a distribuição sob concatenações segue diretamente da terceira parte da definição de $\mathcal{W}_\epsilon$. A mudança de sinal sob reversões segue da fato que o sinal do produto vetorial muda de sinal quando a orientação de um dos vetores é alterada.

A próxima tarefa é mostrar que $\mathcal{W}_\epsilon$ se anula sob triângulos. Isto pode ser feito checando-se todas as configurações de dois triângulos. Isto pode acarretar um número grande de testes. Para diminuir esta quantidade, utilizar-se-á o seguinte resultado intermediário:

Proposição 3.1. *$\mathcal{W}_\epsilon$ é invariante sob translações e transformações lineares inteiras*[3] *com determinante positivo.*

Utilizando-se este resultado, o número de configurações essencialmente diferentes reduz-se drasticamente.

Ao invés de se provar diretamente a nulidade sob triângulos, será estabelecido um resultado mais geral, que fará, inclusive, a ligação entre as duas versões do número de rotação.

3.1.4 Relação entre a versão analítica e geométrica

O seguinte teorema mostra uma equivalência, sob certas restrições, entre as definições analítica e geométrica do número de rotação:

Teorema 3.5. *Sejam L^1 e L^2 duas listas. Se L^1 é fechada e $\epsilon \notin \varphi(L)$, $\forall L \in \Lambda_1$, então:*

$$\mathcal{W}_\epsilon(L^1, L^2) = \frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^2_{n_2})} - \frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^0_{n_2})}$$

$$\mathcal{W}_\epsilon(L^2, L^1) = \frac{1}{2\pi i}\int_{\varphi_\epsilon(L^1)} \frac{dz}{z - \varphi(L^2_{n_2})} - \frac{1}{2\pi i}\int_{\varphi_\epsilon(L^1)} \frac{dz}{z - \varphi_\epsilon(L^0_{n_2})}$$

Antes de se provar o teorema, uma observação quanto à restrição sobre ϵ. Analisando-se a definição analítica do número de rotação, observa-se que a integral complexa não está definida quando P está no caminho γ. Esta é a razão pela qual o parâmetro de translação ϵ não poder estar na imersão de cada segmento da lista.

Prova. Nota-se, primeiramente, que as versões analíticas utilizadas estão bem definidas, pois os pontos $\varphi_\epsilon(L^2_i)$ nunca estarão no caminho $\varphi(L^1)$, pelo comentário anterior.

[3] A matriz de transformação é composta somente por inteiros.

A prova utilizará indução no comprimento das listas L^1 e L^2. Para isto, serão estabelecidos três casos-base:

- **($n_1 = 0$ ou $n_1 = 1$), n_2 qualquer:**

  Como L^1 é fechada, precisa-se ter $L^1_0 = L^1_1$. Pela definição de $\mathcal{W}_\epsilon$, segue diretamente que

  $$\mathcal{W}_\epsilon(L^1, L^2) = 0.$$

  Por outro lado,

  $$\frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^2_{n_2})} = \frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^0_{n_2})} = 0.$$

- **$n_1 = 2$, n_2 qualquer:**

  Como L^1 é fechada, ela precisa ser da forma $L \uplus \mathcal{R}(L)$. Assim:

  $$\mathcal{W}_\epsilon(L^1, L^2) = 0.$$

  Como a definição analítica também distribui sob concatenações e muda de sinal sob reversões, tem-se:

  $$\frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^2_{n_2})} = \frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^0_{n_2})} = 0.$$

- **$n_1 = 3$, $n_2 = 1$**

  Este caso será provado enumerando-se todas as possibilidades. Aqui somente casos essencialmente diferentes serão checados, uma vez que $\mathcal{W}_\epsilon$ e $w(\gamma, P)$ são invariantes sobre translações e transformações inteiras com determinante positivo. Além disto, $\mathcal{W}_\epsilon$ e $w(\gamma, P)$ mudam de sinal sob reflexões: assim, toda transformação linear não-degenerada pode ser usada.

  - Se $L^2_0 = L^2_1$, tem-se $\mathcal{W}_\epsilon(L^1, L^2) = 0$ e

    $$\frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^2_{n_2})} = \frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^0_{n_2})} = 0,$$

    ou seja, a primeira asserção do teorema vale neste caso.

– Assuma-se, agora, que $L_0^2 \neq L_1^2$. Nota-se que nenhum vértice do triângulo T gerado por $\varphi(L^1)$ pode estar na linha l através de $\varphi_\epsilon(L^2)$. Quando todos os vértices de T estão de um lado de l precisa-se ter $T \cap l = \emptyset$, implicando $\mathcal{W}_\epsilon(L^1, L^2) = 0$. Por outro lado,

$$\frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^2_{n_2})} = \frac{1}{2\pi i}\int_{\varphi(L^1)} \frac{dz}{z - \varphi_\epsilon(L^0_{n_2})} = 0,$$

que pode ser visto cortando-se o plano complexo com um raio partindo de um extremo de l.

Assim, os casos remanescentes são formados quando um vértice de T está de um lado de l, e os outros dois vértices estão do outro lado. Isto significa que exatamente dois lados de T cruzam l. Novamente, pelas restrições sobre ϵ, estas intersecções não podem coincidir com os extremos de $\varphi_\epsilon(L^2)$.

Quando aplicada uma translação e uma transformação linear não-degenerada, L_0^2 é mapeado na origem $\mathcal{O}$, L_1^2 é mapeado em $(1, 0)$ e os dois outros vértices de T estão sobre o eixo x. Um extremo de L^2 pode estar à esquerda, entre ou à direita dos pontos de intersecção. Assim, tem-se seis casos distintos a considerar (não contando a orientação de T):

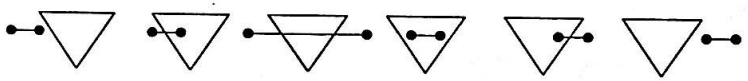


Figura 4: Posições relativas de L^1 e L^2, sem orientação.

Em cada um destes casos, para cada orientação de T, o valor resultante da definição de $\mathcal{W}_\epsilon$ é igual à diferença das integrais complexas.

Isto conclui os casos-base.

- Suponha-se o teorema válido para $n_1 = m \geq 3$ e n_2 fixado. Seja L^1 uma lista de comprimento $m + 1$. Assim:

$$\mathcal{W}_\epsilon(L^1, L^2) = \mathcal{W}_\epsilon(\langle L_0^1, \cdots, L_{m+1}^1\rangle \uplus \langle L_{m+1}^1, L_{m-1}^1, L_m^1, L_{m+1}^1\rangle, L^2) = \mathcal{W}_\epsilon(L^3 \uplus L^4, L^2),$$

onde os comprimentos de L^3 e L^4 são m e 3, respectivamente. Logo:

$$\mathcal{W}_\epsilon(L^3 \uplus L^4, L^2) = \mathcal{W}_\epsilon(L^3, L^2) + \mathcal{W}_\epsilon(L^4, L^2)$$

$$= \frac{1}{2\pi i}\int_{L^3}\frac{dz}{z-L_{n_2}^2} - \frac{1}{2\pi i}\int_{L^3}\frac{dz}{z-L_0^2} + \frac{1}{2\pi i}\int_{L^4}\frac{dz}{z-L_{n_2}^2} - \frac{1}{2\pi i}\int_{L^4}\frac{dz}{z-L_0^2}$$

$$= \frac{1}{2\pi i}\int_{L^3}\frac{dz}{z-L_{n_2}^2} + \frac{1}{2\pi i}\int_{L^4}\frac{dz}{z-L_{n_2}^2} - \frac{1}{2\pi i}\int_{L^3}\frac{dz}{z-L_0^2} - \frac{1}{2\pi i}\int_{L^4}\frac{dz}{z-L_0^2}$$

$$= \frac{1}{2\pi i}\int_{L^3\uplus L^4}\frac{dz}{z-L_{n_2}^2} - \frac{1}{2\pi i}\int_{L^3\uplus L^4}\frac{dz}{z-L_0^2}$$

$$= \frac{1}{2\pi i}\int_{L^1}\frac{dz}{z-L_{n_2}^2} - \frac{1}{2\pi i}\int_{L^1}\frac{dz}{z-L_0^2}$$

Isto mostra que a primeira asserção do teorema é válida para $n_1 = m+1$ e n_2 fixo.

Por outro lado, assumindo-se o teorema válido para n_1 fixo e $n_2 = m \geq 3$, tem-se para uma lista L^2 de comprimento $m+1$:

$$\mathcal{W}_\epsilon(L^1, L^2) = \mathcal{W}_\epsilon(L^1, \langle L_i^2\rangle_{i=0}^{m} \uplus \langle L_i^2\rangle_{i=m}^{m+1})$$

$$= \frac{1}{2\pi i}\int_{L^1}\frac{dz}{z-L_m^2} - \frac{1}{2\pi i}\int_{L^1}\frac{dz}{z-L_0^2} + \frac{1}{2\pi i}\int_{L^1}\frac{dz}{z-L_{m+1}^2} - \frac{1}{2\pi i}\int_{L^1}\frac{dz}{z-L_m^2}$$

$$= \frac{1}{2\pi i}\int_{L^1}\frac{dz}{z-L_{m+1}^2} - \frac{1}{2\pi i}\int_{L^1}\frac{dz}{z-L_0^2}$$

Isto mostra que a primeira asserção do teorema é válida para n_1 fixo e $n_2 = m+1$.

Uma prova similar pode ser feita para a segunda asserção do teorema. □

Um corolário imediato deste teorema estabele a nulidade sob triângulos de $\mathcal{W}_\epsilon$.

Corolário 3.2. *Para ϵ satisfazendo à condição do teorema anterior, $\mathcal{W}_\epsilon$ anula-se sob triângulos.*

Isto pode ser provado observando-se que o teorema implica que $\mathcal{W}_\epsilon(L^1, L^2) = 0$ quando L^1 e L^2 são fechadas. Instanciando as listas para triângulos (listas fechadas de comprimento 3), o corolário segue imediatamente.

Uma questão importante sobre o teorema refere-se à existência do parâmetro de translação ϵ. O resultado seguinte contempla sua existência:

Teorema 3.6. *Seja $q \in \mathbb{Q} - \mathbb{Z}$ e $r \in \mathbb{R} - \mathbb{Q}$ dados. Se $\epsilon = (q, r)$ ou $\epsilon = (r, q)$, então $\epsilon \notin \varphi(L)$, $\forall L \in \Lambda_1$.*

3.2 A translação ϵ

Técnicas de translação(ou perturbação) de pontos são largamente utilizadas em algoritmos de computação gráfica([Gla95]) e geometria computacional([Ede87]). Problemas fundamentais, como intersecção, freqüentemente utilizam-se destas técnicas para tratamento de casos especiais. As técnicas mais conhecidas, geralmente, utilizam perturbações pequenas ou aleatórias.

No problema específico das listas, a Figura 5 mostra porque a translação ϵ é introduzida e seus efeitos.

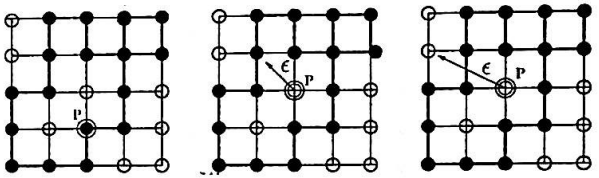


Figura 5: Perturbações em pontos.

Na esquerda, o ponto P não tem número de rotação definido, pois está sobre um dos segmentos da curva: isto pode ser contornado transladando P de ϵ, antes de se calcular efetivamente o número de rotação. Caso esta translação não seja efetuada com cuidado, podem aparecer problemas como os mostrados no centro e à direita da figura. No centro, o diagrama mostra que a introdução da translação ϵ poderia também mover pontos do interior para as bordas, recaindo no caso da esquerda. No caso da direita, a translação foi grande demais, chegando-se a alterar o número de rotação do ponto P.

O principal problema encontrado nestas técnicas é controlar, de maneira efetiva, o tamanho da translação ϵ. Conforme mostra a figura, translações pequenas podem não ser satisfatórias, ou seja, dado $\epsilon \neq \mathcal{O}$, existe uma lista $L \in \Lambda_1$ tal que $\varphi(L) \cap \varphi(\langle \mathcal{O}, \epsilon \rangle) \not\subset \{\mathcal{O}\}$. Em outras palavras, a translação ϵ pode mover os pontos para fora de alguma borda, alterando seus números de rotação. ···

Para se evitar tal situação, fixar-se-á um grau de liberdade para ϵ. Será deixada sua direção indeterminada, porém seu tamanho será tomado infinitesimal. Isto significa que, para todo segmento de linha com um extremo na origem e outro num ponto de coordenadas inteiras, pode ser determinado (usando somente a direção) em que semiplano ϵ está contido e, para todo segmento não passando pela origem, ϵ sempre estará do mesmo lado da origem.

Para se provar tal asserção, utilizar-se-á a seguinte definição:

Definição 3.5.

$$\Lambda^n = \{L : |\Delta(L)|_V \leq n\}.$$

Claramente $\Lambda^n \subset \Lambda^{n+1}$ e $\Lambda = \bigcup_{n\in\mathbb{N}} \Lambda^n$. A seguinte proposição afirma que, para toda lista L, existe uma translação ϵ que não mapeia P para além de L.

Proposição 3.2. *Seja $q \in \mathbb{Q} - \mathbb{Z}$ e $r \in \mathbb{R} - \mathbb{Q}$ dados. Então para todo $n \in \mathbb{N}$, existe um $\lambda > 0$ tal que , $\forall L \in \Lambda^n, P \in \mathbb{Z}^2, 0 < \mu(q,r) \leq \lambda$, tomando-se $\epsilon = \mu(q,r)$, vale $\varphi(\langle P, P+\epsilon\rangle) \cap \varphi(L) \subset \{P\}$.*

Isto significa afirmar que, para um λ suficientemente pequeno, substituindo-se ϵ por $\mu(q,r)$, com $0 < \mu(q,r) \leq \lambda$, $\mathcal{W}_\epsilon$ não é afetado em $\Lambda^n \times \Lambda^n$. Além disto, pode-se provar que esta invariância de $\mathcal{W}_\epsilon$ em $\Lambda^n \times \Lambda^n$ não depende do tamanho de ϵ, mas somente de sua direção. Isto é contemplado pela seguinte proposição:

Proposição 3.3. *Seja $q \in \mathbb{Q} - \mathbb{Z}$ e $r \in \mathbb{R} - \mathbb{Q}$ dados. Então para todo $n \in \mathbb{N}$, existe um $\lambda > 0$ tal que , $\forall L^1, L^2 \in \Lambda^n, P \in \mathbb{Z}^2, 0 < \mu_1(q,r) \leq \mu_2(q,r) \leq \lambda$, vale que*

$$\mathcal{W}_{\mu_1(q,r)}(L^1, L^2) = \mathcal{W}_{\mu_2(q,r)}(L^1, L^2).$$

Um detalhe extremamente importante neste processo de minimização dos efeitos da translação ϵ é manter a propriedade CW de $\mathcal{W}_\epsilon$. Isto é analisado pelo próximo resultado:

Teorema 3.7. *O limite $\lim_{\lambda\downarrow 0} \mathcal{W}_{\lambda\epsilon}(L^1, L^2)$ existe para toda L^1 e L^2. Além disto, a função limite:*

$$\mathcal{W}_\rho(L^1, L^2) = \lim_{\lambda\downarrow 0} \mathcal{W}_{\lambda\epsilon}(L^1, L^2)$$

é uma função CW.

Para uma conveniência de notação do tamanho e direção de ϵ, tomar-se-á a forma polar de ϵ por (r,ρ). Na seqüência, utilizar-se-á a notação $\mathcal{W}_\rho$ para indicar $\mathcal{W}_{(r,\rho)}$ e será tomada a versão analítica correspondente por:

$$\frac{1}{2\pi i}\int_{\varphi(L)} \frac{dz}{z - \varphi_\rho(P)}.$$

Se um ponto P não pertence a um segmento poligonal, a infinitesimalidade de r garante que P nunca será movido para ou além deste segmento, após a translação. Porém, se P pertence a algum segmento poligonal, a irracionalidade da tangente ρ permite garantir que P nunca permanece neste segmento após a translação.

3.3 O Teorema de Jordan

O Teorema da Curva de Jordan é um resultado importante em topologia de baixa dimensão, estabelecido por Camille Jordan (1838-1892) e provado rigorosamente por Oswald Veblen (1880-1960) em 1905. Este resultado afirma o seguinte:

Teorema 3.8. *Seja γ uma curva homotópica a S^1, i.e. sem auto-intersecções, imersa em $\mathbb{R}^2$. Então o complemento de γ, com respeito a $\mathbb{R}^2$, tem exatamente duas componentes, cada qual tendo γ como borda. Uma destas componentes (o interior de γ) é um conjunto limitado, e a outra, o exterior de γ, um conjunto ilimitado.*

Apesar de parecer intuitiva, a prova deste teorema requer argumentos sofisticados de topologia algébrica, como homotopia e homologia([Mas91]).

3.4 Versão discreta do teorema

Esta seção introduz uma versão discreta do Teorema da Curva de Jordan. Esta versão contempla listas arbitrárias, inclusive com auto-intersecção.

Teorema 3.9. *Seja ρ um ângulo com tangente irracional. Dada uma lista fechada L^1, a função CW $\mathcal{W}_\rho$ divide o plano $\mathbb{Z}^2$ em um número finito de regiões com pontos P de igual número de rotação:*

$$\frac{1}{2\pi i}\int_{\varphi(L)} \frac{dz}{z-\varphi_\rho(P)}.$$

Precisamente uma destas regiões é infinita, contendo pontos com número de rotação igual a 0. Além disto, quando para uma lista L^2 de comprimento n_2 valer:

$$\frac{1}{2\pi i}\int_{\varphi(L)} \frac{dz}{z-\varphi_\rho(L_0^2)} \neq \frac{1}{2\pi i}\int_{\varphi(L)} \frac{dz}{z-\varphi_\rho(L_{n_2}^2)},$$

então necessariamente precisa-se ter:

$$\varphi(L^1)\cap\varphi(L^2)\neq\emptyset.$$

Prova. O corolário 3.1 mostra como construir a partição dada pela função $\mathcal{W}$ e a lista L^1. O teorema 3.5 mostra que as regiões contém pontos com números de rotação iguais. Para mostrar que precisamente uma das regiões é infinita, basta notar que para todo ponto P com $|P| > |L^1|$ tem-se:

$$\frac{1}{2\pi i}\int_{\varphi(L)} \frac{dz}{z - \varphi_\rho(P)} = 0.$$

Logo, segue-se que somente um número finito de pontos tem número de rotação diferente de zero.

Para provar a última asserção do teorema, observa-se que, por definição de $\mathcal{W}_\epsilon$, tem-se para todo ϵ:

$$\frac{1}{2\pi i}\int_{\varphi(L)} \frac{dz}{z - \varphi_\rho(L_0^2)} \neq \frac{1}{2\pi i}\int_{\varphi(L)} \frac{dz}{z - \varphi_\rho(L_{n_2}^2)} \Rightarrow$$
$$\mathcal{W}_\epsilon(L^1, L^2) \neq 0 \Rightarrow$$
$$\varphi(L^1) \cap \varphi_\epsilon(L^2) \neq \emptyset.$$

Como os polígonos gerados por φ_ϵ são conjuntos fechados para todo ϵ, no limite $r \downarrow 0$ a intersecção não é vazia também. □

Este teorema afirma que, sob certas condições, as imersões poligonais de duas listas precisam ter um ponto em comum. O teorema a seguir afirma que, sob certas condições de conectividade, elas têm um ponto em comum.

Teorema 3.10. *Seja* $(m_1, m_2) \in \{(4,4),(4,6),(4,8),(6,4),(6,6),(8,4)\}$, L^1 *e* L^2 *listas* m_1*-conexa e* m_2*-conexa, respectivamente, não necessariamente fechadas. Se* $\mathcal{W}_\rho(L^1, L^2) \neq 0$, *então* $\{L_i^1\} \cap \{L_j^2\} \neq \emptyset$. *Este resultado não é válido para* $(m_1, m_2) \in \{(6,8),(8,6),(8,8)\}$.

Estes dois últimos teoremas serão utilizados na próxima seção para o desenvolvimento de algoritmos eficientes e corretos para o preenchimento de regiões delimitadas por curvas discretas, contorno de caminhos e operações de *antialiasing*.

4 Aplicações

A beleza matemática dos resultados anteriores não está só encerrada no cunho teórico. Esta seção mostra três aplicações importantes de tais resultados em computação gráfica.

4.1 Preenchimento de regiões

Preenchimento de regiões é uma importante operação gráfica de transformação de um caminho fechado numa região, que encontra aplicações em algoritmos de iluminação e texturização, fontes, dentre outros.

4.1.1 Funções de rasterização

No contexto da operação de preenchimento, considera-se o processo de rasterização do interior(e do exterior) de uma lista L consistindo de dois estágios:

- Construção de uma relação de eqüivalência em $\mathbb{Z}^2$, atribuindo a cada região de uma classe de eqüivalência um único número (por exemplo, o número de rotação com relação a L).
- Atribuição de um valor binário ao pontos da imagem, tal que todos os pontos em uma classe de eqüivalência tenham um mesmo valor, geralmente não único.

O primeiro estágio requer a especificação de uma função de partição, a ser calculada para cada ponto de uma região de interesse. A funcionalidade requerida pelo segundo estágio pode ser provida por uma função muito simples[4].

As funções que fazem o particionamento da região de interesse em classes de eqüivalência são conhecidas como *funções de rasterização*:

Definição 4.1. *Uma função de rasterização $Q(L,P)$ é uma função $Q : \Lambda_c \times \mathbb{Z}^2 \rightarrow \mathbb{Z}$, onde Λ_c denota o conjunto das listas fechadas.*

Especificando-se uma função de rasterização $Q(L,P)$, obtém-se a numeração de cada classe de eqüivalência. Estas classes são, no máximo, enumeráveis. Dada uma lista fechada L e um ponto P na imagem, calcula-se $Q(L,P)$ para se obter a qual classe de eqüivalência P pertence.

4.1.2 Extremização

Extremização é uma propriedade desejada em funções de rasterização. Isto expressa a propriedade que, quando um conjunto de polígonos simples que se sobrepõem uns aos outros, a rasterização também produz regiões que se sobrepõem umas as outras.

[4]Por exemplo, classes de eqüivalência no interior da curva recebem valor 0, e as do exterior, valor 1.

Para se formalizar a noção desta propriedade, precisar-se-á de algumas definições:

Definição 4.2. • *Um polígono $L \in \Lambda_c$ de comprimento 3 é chamado constante se e somente se $\Delta(L)_0 \times \Delta(L)_1 = 0$.*

- *Um polígono $L \in \Lambda_c$ de comprimento 3 é chamado simples se e somente se não é constante.*

- *Um polígono $L \in \Lambda_c$ de comprimento $n > 3$ é chamado simples se e somente se*

$$\forall_{0 \leq i,j \leq n} : \varphi(\langle L_i, L_{i+1} \rangle) \cap \varphi(\langle L_j, L_{j+1} \rangle) \neq \emptyset \Rightarrow (|i-j|+1) \bmod n \leq 2.$$

- *O interior de um polígono simples é definido como o conjunto dos pontos que não tem número de rotação 0 com respeito ao polígono. Pontos do polígono não fazem parte do interior.*

- *A orientação de um polígono simples é dita definida positiva quando os pontos do interior têm número de rotação $+1$ e, definida negativa, quando têm número de rotação -1.*

Os polígonos simples são objetos topológicos homotópicos ao círculo unitário S^1 e têm o interior como um conjunto simplesmente conexo. Assim, a orientação de um polígono simples é bem definida.

Definição 4.3. *Um conjunto de polígonos é chamado um conjunto de extremidades se e somente se*

- *todos os polígonos são simples e com orientação definida positiva*

- *todos os polígonos podem ser agrupados por rotação dos índices de pontos, concatenação e remoção de extremidades, em um polígono simples.*

Utilizando-se estas ferramentas, pode-se formalizar a noção de extremização:

Definição 4.4. *Uma função de rasterização $\mathcal{Q}(L, P)$ é dita ter a propriedade de* extremização *se e somente se:*

- $\mathcal{Q}$ é invariante sob rotação de índices

  Se $L^1 \uplus L^2$ é fechada, então $\mathcal{Q}(L^1 \uplus L^2, P) = \mathcal{Q}(L^2 \uplus L^1, P)$.

- $\mathcal{Q}$ anula-se sobre concatenações da forma $L^1 \uplus \mathcal{R}(L^1)$

  $\mathcal{Q}(L^1 \uplus \mathcal{R}(L^1), P) = 0.$

- $\mathcal{Q}$ distribui sob concatenações

  Se L^1 e L^2 são listas concatenáveis, então $\mathcal{Q}(L^1 \uplus L^2, P) = \mathcal{Q}(L^1, P) + \mathcal{Q}(L^2, P)$.

- $\mathcal{Q}$ é simples

  Se L é simples, então para todo P tem-se $\mathcal{Q}(L, P) \in \{0, 1\}$ se L tem orientação definida positiva, e $\mathcal{Q}(L, P) \in \{-1, 0\}$ se L tem orientação definida negativa.

Esta definição permite obter um resultado importante sobre funções de rasterização:

Teorema 4.1. *As regiões produzidas por uma função de rasterização(com a propriedade de extremização), aplicada a um conjunto de extremidades, têm intersecção duas-a-duas vazia. Além disto, a união destas regiões é igual ao conjunto das regiões produzidas pela aplicação da função de rasterização ao polígono simples resultante do agrupamento das extremidades.*

4.1.3 Rasterização via funções CW

A função CW $\mathcal{W}_\rho$, estabelecida na seção anterior, pode ser usada para construir uma função de rasterização. Para listas fechadas, o valor do número de rotação relacionado

$$\frac{1}{2\pi i} \int_{\varphi(L)} \frac{dz}{z - \varphi_\rho(P)}$$

será utilizado como a função de enumeração de classes de eqüivalência.

Definição 4.5. *Defina-se a função de rasterização $\mathcal{F}_\rho : \Lambda \times \mathbb{Z}^2 \to \mathbb{Z}$ por:*

$$\mathcal{F}_\rho(L, P) = \lim_{Q \to \infty} \mathcal{W}_\rho(L, \langle Q, P \rangle)$$

Nota-se que o domínio de $\mathcal{F}$ é maior que o requerido pela definição de função de rasterização. Este levantamento de domínio é utilizado para gerar resultados fundamentais sobre a função. Um destes resultados é mostrado a seguir:

Proposição 4.1. *Se L é uma lista fechada, então:*

$$\mathcal{F}_\rho(L,P) = \frac{1}{2\pi i}\int_{\varphi(L)} \frac{dz}{z-\varphi_\rho(P)}.$$

Utilizando-se este resultado, pode-se provar que a função de rasterização definida anteriormente, com o auxílio de $\mathcal{W}_\rho$, possui a propriedade de extremização:

Teorema 4.2. *$\mathcal{F}_\rho$ é uma função de rasterização com a propriedade de extremização.*

Alguns outros resultados adicionais acerca de $\mathcal{F}_\rho$ são listados a seguir.

Proposição 4.2.

- **$\mathcal{F}_\rho$ anula-se sobre triângulos constantes**

  Se L é um triângulo constante, então $\mathcal{F}_\rho(L,P) = 0$.

- **$\mathcal{F}\rho$ distribui sobre concatenações gerais**

  Se L^1 e L^2 são listas concatenáveis, não necessariamente fechadas, então $\mathcal{F}_\rho(L^1 \uplus L^2, P) = \mathcal{F}_\rho(L^1,P) + \mathcal{F}_\rho(L^2,P)$.

- **$\mathcal{F}_\rho$ é invariante sobre mudanças de escala**

  $\forall_{a\in\mathbf{Z}-\{0\}} : \mathcal{F}_\rho(aL,aP) = \mathcal{F}_\rho(L,P)$.

Estes resultados adicionais serão utilizadas para provas de corretudes de algoritmos para a operação de preenchimento de região.

4.1.4 Polígonos

Em qualquer implementação da função de rasterização $\mathcal{F}_\rho$ definida anteriormente, um limite para ρ precisa ser escolhido. Algumas escolhas deste limite são mais fáceis de se implementar, tais como $\rho \uparrow 0$ ou valores limites de $\frac{1}{2}k\pi$. Nesta seção, será construída uma implementação eficiente de $\lim_{\rho\uparrow 0} \mathcal{F}_\rho$.

Definição 4.6. *Defina-se a função de rasterização $\mathcal{F}$ como:*

$$\mathcal{F} = \lim_{\rho\uparrow 0} \mathcal{F}_\rho.$$

Como $\mathcal{F}$ distribui sob concatenações, o primeiro passo da implementação de $\mathcal{F}$ será uma soma das contribuições dos lados poligonais da lista. Nesta implementação, existem vários casos nos quais a contribuição de um segmento poligonal $L = \langle L_0, L_1 \rangle$ é 0 ou de cálculo imediato, podendo ser detectados facilmente.

A seguir, serão enumerados alguns casos suficientes para a implementação do limite. Os dois primeiros casos mostram como calcular a função de rasterização $\mathcal{F}(L, P)$, em relação a uma região retangular limitante de L, chamada *bounding box*.

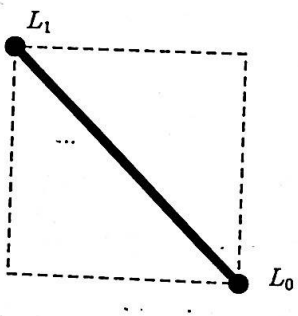


Figura 6: Bounding box de um segmento poligonal L.

O primeiro caso consiste em verificar se o ponto P está na seguinte região:

Proposição 4.3. *Se* $P^y > \max\{L_i^y\}$ *ou* $P^y \leq \min\{L_i^y\}$ *ou* $P^x < \min\{L_i^x\}$, *então* $\mathcal{F}(L, P) = 0$.

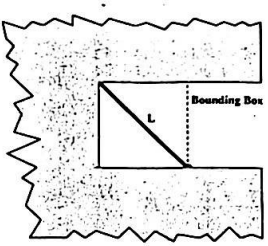


Figura 7: Caso em que $\mathcal{F}(L,P) = 0$.

O segundo caso verifica se P está na seguinte região:

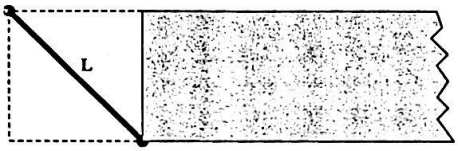


Figura 8: Caso em que $\mathcal{F}(L,P) = s(L_0^y - L_1^y)$.

Proposição 4.4. *Se* $P^y \leq \max\{L_i^y\}$ *e* $P^y > \min\{L_i^y\}$ *e* $P^x \geq \max\{L_i^x\}$, *então* $\mathcal{F}(L,P) = s(L_0^y - L_1^y)$.

O próximo resultado mostra como calcular a função de rasterização dentro da bounding box:

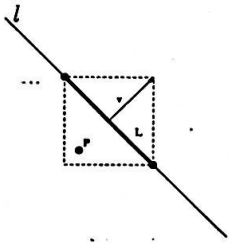


Figura 9: Elementos para cálculo dentro da bounding box.

Proposição 4.5. *Sejam* $\min\{L_i^y\} < P^y \leq \max\{L_i^y\}$, $\min\{L_i^x\} < P^x \leq \max\{L_i^x\}$ *e* l *a linha passando através do segmento poligonal de interesse* L.

- *Se l é paralela ao eixo x, então* $\mathcal{F}(L,P) = 0$.

- *Se l não é paralela ao eixo x, então* $\Delta(L)^y \neq 0$. *Seja v o vetor perpendicular a l, tal que* $v^x > 0$ *e* $v = \pm(\Delta(L)^y, -\Delta(L)^x)$. *Assim, l pode ser descrita como* $(x,y) \cdot v = c$ *e vale :*

$$P \cdot v \geq c \Rightarrow \mathcal{F}(L,P) = s(L_0^y - L_1^y)$$

$$P \cdot v < c \Rightarrow \mathcal{F}(L,P) = 0.$$

Estes casos cobrem todas as possibilidades de posicionamento do ponto P em relação ao segmento poligonal L. Assim, o primeiro passo da implementação será detectar todos estes casos. A função referente a tal implementação será, doravante, denominada **Contribuição** e terá o protótipo **Contribuição (Ponto L0,Ponto L1)**.

Tendo-se a contribuição para cada um dos segmentos poligonais da lista, utiliza-se a propriedade de distribuição sob concatenação para se calcular o valor da função de rasterização para um ponto P:

Algoritmo 1.1 Cálculo de número de rotação de poligonais.

NúmeroDeRotaçãoPoligonal(Lista L,Ponto P)
Entrada: L - pontos da lista
P - ponto de intersse.
Saída: Valor de $\mathcal{F}(L,P)$.
1: wn←0
2: **para** i=0 **até** Comprimento(L)-1 **faça**
3: wn← wn+Contribuição(L[i]-P,L[i+1]-P)
4: **fim para**
5: Devolva(wn)

O translação do passo 3 do algoritmo é uma otimização para acelerar o processo de cálculo. Esta otimização é conhecida como *normalização* do ponto P.

4.1.5 Curvas de Bézier Polinomiais

O algoritmo apresentado na seção anterior refere-se exclusivamente a polígonos. Porém, em muitas aplicações, tais como fontes, ocorrem elementos gráficos importantes, como curvas. Esta seção analisa uma extensão do algoritmo anterior para curvas de Bézier discretas.

Curvas de Bézier podem ser incorporadas ao algoritmo anterior convertendo-as para listas poligonais e calculando a função de rasterização nestas listas. O algoritmo a ser exibido utiliza um método híbrido para estes passos, combinando conversão e cálculo, produzindo significantes otimizações. Um ponto importante reside no fato que a conversão preservará a propriedade de extremização.

Em primeiro lugar, formalizar-se-á a noção de curva de Bézier, com pontos de controle numa lista.

Definição 4.7. *Seja L uma lista de comprimento n. Para $t \in [0,1]$ defina-se $\mathcal{B}_L(t)$, a curva de Bézier de L, por:*

$$\mathcal{B}_L(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} L_i.$$

Desta definição também surge uma nova maneira de se intepretar listas: uma lista também pode definir uma curva de Bézier, onde o comprimento de lista, usualmente, determina o grau da curva.

Uma operação essencial que pode ser definida em curvas de Bézier é a *subdivisão*. Subdivisões de curvas de Bézier resultam em duas curvas de Bézier, chamadas de *parte esquerda* e *parte direita*.

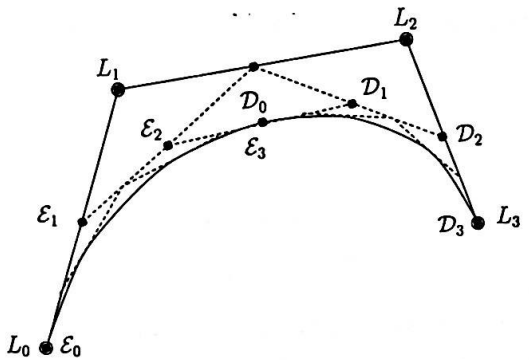


Figura 10: Partes esquerda e direita de uma curva de Bézier.

Definição 4.8. *Seja L uma lista de comprimento n. Define-se a parte esquerda $\mathcal{E}$ de $\mathcal{B}_L$ por:*

$$\mathcal{E} : \Lambda \to \Lambda$$

$$\mathcal{E}(L)_i = 2^{-i} \sum_{j=0}^{i} \binom{i}{j} L_j$$

e a parte direita $\mathcal{D}$ *por:*

$$\mathcal{D} : \Lambda \to \Lambda$$
$$\mathcal{D}(L)_i = 2^{i-n} \sum_{j=i}^{n} \binom{n-i}{j} L_j.$$

Um dos aspectos essenciais do teste interior/exterior de Pol e Corthout [CorP92] é o uso exclusivo de aritmética inteira. Assim, da definição, a divisão por uma potência de 2 precisa produzir um resultado exato. Para implementar uma subdivisão em um algoritmo que manipula somente inteiros, precisa-se mapear as coordenadas produzidas pela subdivisão no conjunto do inteiros. Para isto, serão introduzidas as *funções aproximadoras*:

Definição 4.9. *Uma função não-decrescente* $d : \mathbb{R} \to \mathbb{Z}$ *é chamada uma função aproximadora se e somente se* $\forall_{x \in \mathbb{R}, y \in \mathbb{Z}}$, $d(y) = y$ *e se* $x \in [y, y+1]$, *então* $d(x) = y$ *ou* $d(x) = y + 1$.

As funções de truncamento e arredondamento são exemplos familares desta classe de funções. Neste contexto, serão redefinidas as noções de *parte esquerda* e *parte direita* de uma curva de Bézier, no ambiente discreto:

Definição 4.10. *Seja* L *uma lista de comprimento* n. *Define-se a parte esquerda discreta* $\mathcal{E}'$ *de* $\mathcal{B}_L$ *por:*

$$\mathcal{E}' : \Lambda \to \Lambda$$
$$\mathcal{E}'(L)_i \mapsto d(\mathcal{E}(L)_i)$$

e parte direita discreta $\mathcal{D}'$ *por:*

$$\mathcal{D}' : \Lambda \to \Lambda$$
$$\mathcal{D}'(L)_i \mapsto d(\mathcal{D}(L)_i).$$

Estes operadores de subdivisão discreta formarão a base para a conversão curva-lista poligonal. Esta conversão será construída sobre o paradigma de divisão-e-conquista, cujo critério de parada será a distância entre dois argumentos da lista.

Definição 4.11. *Seja* L *uma lista de comprimento* n. *Seja* $e_{d_8}(L)$ *o diâmetro 8-conexo de* L, *ou seja,* $e_{d_8}(L) = \max_{i,j}\{|L_j - L_i|_{V^8}\}$.

Define-se o operador conversão curva-polígono $\mathcal{V}$ *por:*

$$e_{d_8}(L) \leq 1 \Rightarrow \mathcal{V}(L) = \langle L_0, L_n \rangle$$

$$c_{d_s}(L) > 1 \Rightarrow \mathcal{V}(L) = \mathcal{V}(\mathcal{E}'(L)) \uplus \mathcal{V}(\mathcal{D}'(L)).$$

...

Neste processo de conversão, dois fatos precisam ser verificados:

- o critério de parada é sempre atingido após um número finito de subdivisões
- $\mathcal{V}(\mathcal{E}'(L))$ e $\mathcal{V}(\mathcal{D}'(L))$ sempre podem ser concatenadas

Para o primeiro fato, como todas as definições e cálculos são invariantes sob translações, pode-se assumir, sem perda de generalidade, que o espaço que contém os pontos de controle de uma curva de Bézier de grau n, representada por uma lista L, seja $[-m \ldots m] \times [-m \ldots m]$, onde m é alguma constante pré-definida.

Assim, pode ser provado o seguinte limitante superior:

Teorema 4.3. *Após $k = \lceil \log_2 m \rceil + n$ subdivisões, a curva de Bézier resultante $\mathcal{E}'^k$ satisfaz o critério de parada.*

Um resultado semelhante pode ser provado para $\mathcal{D}'^k$.

Para o segundo fato, precisar-se-á da seguinte proposição:

Proposição 4.6. *Seja L uma lista de comprimento n e p o comprimento de $\mathcal{V}(L)$. Então $\mathcal{V}(L)_0 = L_0$ e $\mathcal{V}(L)_p = L_n$.*

Verificado que o processo de conversão é válido, discutir-se-á um detalhe sutil: o processo produz uma lista 8-conexa.

Proposição 4.7. *Para toda lista L, $\mathcal{V}(L)$ é uma lista 8-conexa.*

A prova desta proposição é feita diretamente por indução no número de subdivisões. A utilidade deste detalhe está estritamente relacionada com uma simplificação no algoritmo para computar $\mathcal{F}(\mathcal{V}(L), P)$: somente listas 8-conexas precisarão ser suportadas.

Os modelos que o algoritmo irá tratar serão formados de listas de listas representativas de curvas de Bézier. Seja L uma destas listas de listas, com L^i_j representando o j-ésimo ponto de controle da i-ésima lista. Para formar um contorno fechado a ser preenchido, estas curvas de Bézier precisam estar conectadas, ou seja, $L^i_{\#L^i} = L^{i+1}_0$ e $L^{\#L}_{\#L^{\#L}} = L^0_0$. A operação de preenchimento é feita pelo cálculo de $\mathcal{F}(\uplus_i \mathcal{V}(L^i), P)$: como $\mathcal{F}$ distribui sob concatenações, isto pode ser reescrito como $\sum_i \mathcal{F}(\mathcal{V}(L^i), P)$.

Um ponto importante sobre a conversão curva-polígono refere-se à preservação da propriedade de extremização. Para provar isto, utilizar-se-á a seguinte proposição:

Proposição 4.8. *O operador reversão comuta com o operador de conversão curva-polígono, ou seja,*

$$\mathcal{R} \circ \mathcal{V} = \mathcal{V} \circ \mathcal{R}.$$

A prova desta proposição vem imediatamente das definições.

Teorema 4.4. *O processo de rasterização $\sum_i \mathcal{F}(\mathcal{V}(L^i), P)$, de uma seqüência fechada de curvas representada por uma lista de listas, possui a propriedade de extremização.*

Para um cálculo eficiente de $\sum_i \mathcal{F}(\mathcal{V}(L^i), P)$, mecanismos de limitação por *bounding boxes* serão utilizados, analogamente ao caso dos polígonos. Curvas de Bézier contínuas possuem uma propriedade chamada *casco convexo*: os pontos de uma curva sempre podem ser escritos como combinações lineares convexas dos pontos de controle da curva. Esta propriedade terá analogias no caso discreto, utilizando-se *bounding boxes*, como mostrado pela seguinte proposição:

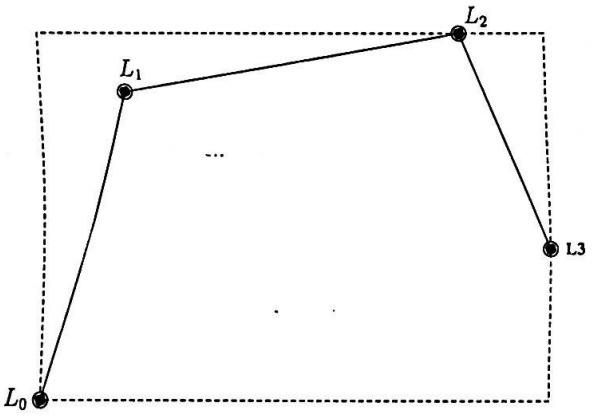


Figura 11: Bounding box usando os pontos de controle da curva.

Proposição 4.9. *Seja L uma lista de comprimento n e $\mathcal{V}(L)$ uma lista de comprimento m. Então:*

$$\min\{L_i^x\} \leq \min\{\mathcal{V}(L)_j^x\} \leq \max\{\mathcal{V}(L)_j^x\} \leq \max\{L_i^x\}$$
$$\min\{L_i^y\} \leq \min\{\mathcal{V}(L)_j^y\} \leq \max\{\mathcal{V}(L)_j^y\} \leq \max\{L_i^y\}$$

Para se estabelecer os casos de computação imediata de $\mathcal{F}(L, P)$, o primeiro passo será extender as proposições 4.3 e 4.4, para listas de qualquer comprimento:

Proposição 4.10. *Seja L uma lista de comprimento n. Assim, tem-se:*

- *Se* $P^x > \min\{L_i^x\}$ *ou* $P^y > \min\{L_i^y\}$ *ou* $P^y < \min\{L_i^y\}$, *então* $\mathcal{F}(L,P) = 0$
- *Se* $P^x \geq \max\{L_i^x\}$ *então* $\mathcal{F}(L,P) = \frac{1}{2}\left(s\left((L_0 - P)^y + \frac{1}{2}\right) - s\left((L_n - P)^y + \frac{1}{2}\right)\right)$.

O seguinte resultado permite obter dois resultados imediatos de cálculo da função de rasterização, para a conversão curva-polígono:

Proposição 4.11. *Seja L uma lista de comprimento n. Assim, tem-se:*

- *Se* $P^y \leq \max\{L_i^y\}$ *e* $P^y > \min\{L_i^y\}$ *e* $P^x \geq \max\{L_i^x\}$, *então* $\mathcal{F}(\mathcal{V}(L),P) = 0$
- *Se* $P^y > \min\{L_i^x\}$ *ou* $P^y > \min\{L_i^y\}$ *ou* $P^y < \min\{L_i^y\}$, *então* $\mathcal{F}(\mathcal{V}(L),P) = \frac{1}{2}\left(s\left((L_0 - P)^y + \frac{1}{2}\right) - s\left((L_n - P)^y + \frac{1}{2}\right)\right)$.

A expressão $\frac{1}{2}\left(s\left((L_0 - P)^y + \frac{1}{2}\right) - s\left((L_n - P)^y + \frac{1}{2}\right)\right)$ não parece muito implementável com o uso de artimética inteira. Para contrapor esta dificuldade, será definida uma função sinal, no âmbito dos inteiros, e provada uma expressão eqüivalente:

Definição 4.12. *Define-se a função sinal inteiro* $s' : \mathbb{Z} \to \{0,1\}$ *por:*

$$s'(x) = \begin{cases} 1 & se\ x < 0 \\ 0 & se\ x \geq 0. \end{cases}$$

Com esta definição, pode ser provado o seguinte resultado de equivalência entre as funções-sinal:

Teorema 4.5. $\frac{1}{2}\left(s\left((L_0 - P)^y + \frac{1}{2}\right) - s\left((L_n - P)^y + \frac{1}{2}\right)\right) = s'\left((L_n - P)^y\right) - s'\left((L_0 - P)^y\right)$.

Estes resultados são suficientes para se construir uma extensão no algoritmo de rasterização de polígonos para curvas de Bézier discretas. A função correspondente a está implementação será chamada **ContribuiçãoBézier (Lista L)**.

Como $\mathcal{F}$ distribui sob concatenação, o algoritmo para calcular a contribuição de uma lista de listas, para um ponto P, é simplesmente uma soma de contribuições de listas normalizadas.[5]

[5]Novamente, a normalização de listas é utilizada para acelerar a computação.

Algoritmo 1.2 Cálculo de número de rotação para Curvas de Bézier.

NúmeroDeRotaçãoBezier(Lista de listas L,Ponto P)

Entrada: L - lista L de listas contendo pontos de controle de curvas de Bézier discretas
P - Ponto de interesse.

Saída: Valor de $\mathcal{F}(L, P)$.

```
1: wn←0
2: para i← 0 até Comprimento(L)-1 faça
3:    wn← wn+ContribuiçãoBézier(L[i]-P)
4: fim para
5: Devolva(wn)
```

Pol e Corthout [CorP92] também descrevem as curvas de Bézier racionais no espaço discreto, cujo cálculo das funções de rasterização recai no caso anterior por transformações via projeções.

4.1.6 O algoritmo quadrático

Estabelecido um método de teste interior/exterior, o algoritmo para realizar a operação de preenchimento aparece de maneira natural: simplesmente para cada ponto P da imagem a ser produzida, verifica-se se P está no interior ou exterior.

Algoritmo 1.3 Preenchimento de regiões quadrático.

Filling(Lista L)

Entrada: L - Lista poligonal ou lista de listas (curvas de Bézier)

Saída: Interior de L.

```
1: para cada ponto P da imagem faça
2:    se L é poligonal então
3:       teste← NúmeroDeRotação(L,P)
4:    senão
5:       teste← NúmeroDeRotaçãoBezier(L,P)
6:    fim se
7:    se teste≠0 então
8:       Pintar ponto P
9:    fim se
10: fim para
```

Indicando por $custo_{jordan}$ a complexidade do teste interior/exterior[6] e a resolução da

[6]Observa-se que esta complexidade é linear no número de pontos da curva discreta.

imagem por r, tem-se a seguinte complexidade para o algoritmo:

$$O(r^2)custo_{jordan},$$

pois, para cada ponto, é realizado um teste interior/exterior.

Apesar de polinomial na resolução, este algoritmo é considerado lento na maioria das aplicações práticas. A próxima seção aborda alguns métodos para reduzir esta complexidade, utilizando-se a noção de *coerência*.

4.1.7 Redução da complexidade

A utilização do teste interior/exterior de Pol e Corthout [CorP92], no algoritmo trivial para preenchimento, tem comportamento quadrático com respeito à resolução. Algumas técnicas alternativas têm comportamento linear na resolução: *scan-conversion* [Fol96], por exemplo, passa duas vezes sobre uma *scan-line*. Em primeira instância, pode parecer que a utilização do teste de Pol e Corhout não seja interessante, devido à discrepância de complexidade: contudo, existem mecanismos para reduzir a complexidade de utilização do teste à patamares quasi-linear e sub-linear na resolução. Estes mecanismos são conhecidos como *testes de coerência*.

4.1.8 Coerência

Para o estabelecimento efetivo dos testes de coerência, é necessária a formalização do conceito de coerência na operação de preenchimento.

Definição 4.13. *Seja L uma lista fechada e $\mathcal{F}$ uma função de rasterização. Uma região R é chamada coerente, com respeito a L, se e somente se R é um subconjunto de uma classe de eqüivalência simples, induzida em $\mathbb{Z}^2$ por $\mathcal{F}$.*

Isto é equivalente a dizer que uma região R é coerente, com respeito a L, se e somente se R ou está totalmente contida no interior de L ou em seu exterior. Embora a definição seja matematicamente precisa, ela é pouco operacional.

O próximo resultado traduz a relação de coerência entre uma região e uma curva em termos mais implementáveis.

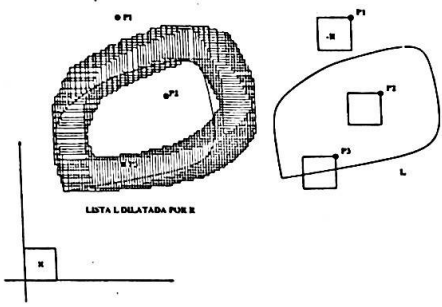


Figura 12: Teste de coerência.

Teorema 4.6. *Seja* $(m_1, m_2) \in \{(4,4),(4,6),(4,8),(6,4),(6,6),(8,4)\}$. *Se* L *é uma lista fechada,* m_1*-conexa, e* R *uma região* m_2*-conexa, então se* $P \notin \{L_i\} \oplus R$, *tem-se que* $P - R$ *é coerente com respeito a* L, *onde* $\oplus$ *denota a soma de Minkowiski usual.*

Este resultado estabelece que a coerência de uma região $P - R$ é implicada quando um simples teste interior/exterior com uma região dilatada $(L \oplus R)$ retorna negativo. A Figura 12 mostra alguns exemplos de testes, onde tem-se as regiões indexadas por $P1$ e $P2$ coerentes e, a indexada por $P3$, não-coerente.

A recíproca do resultado anterior não é verdadeira. Para isto, basta considerar uma lista da forma $L = L_1 \uplus \mathcal{R}(L_1)$: como o interior de L não contém pontos, existe uma região $P - R$ coerente, com $P \in \{L_i\} \oplus R$.

4.1.9 Coerência com quadtrees

O teste anterior pode ser implementado, de maneira eficiente, utilizando a bem conhecida estrutura de dados *quadtrees*. O algoritmo é baseado no artigo original de Hunter e Steiglitz [HunS79], que aborda *quadtrees* no contexto de representações em modelagem geométrica. No algoritmo a seguir, foi modificada apenas a questão de classificação com base no teste acima:

Algoritmo 1.4 Preenchimento de regiões com quadtrees.

FillingComQuadtrees(Região R,Lista L)

Entrada: R - região de ineteresse para preenchimento
L - Lista poligonal ou lista de listas (curvas de Bézier)

Saída: Interior de L.

1: se R é coerente com respeito a L **então**
2: Escolher um representante P de R
3: Fazer w←NúmeroDeRotaçãoPoligonal(Lista L,Ponto P) ou w←NúmeroDeRotaçãoBezier(Lista L,Ponto P) de acordo com o tipo de lista
4: se w≠0 **então**
5: Pintar P
6: **senão**
7: Subdividir R em quatro partes R1,R2,R3,R4
8: FillingComQuadtrees(R1,L)
9: FillingComQuadtrees(R2,L)
10: FillingComQuadtrees(R3,L)
11: FillingComQuadtrees(R4,L)
12: **fim se**
13: **fim se**

A Figura 13, mostrada na próxima página, simula algumas etapas do funcionamento para uma entrada do algoritmo:

Claramente este algoritmo recursivo sempre pára, pois, no máximo, atinge-se uma região R formada por um único ponto e tais regiões são coerentes com respeito a L.

Observa-se, facilmente, que as folhas da árvore resultante da recursão são as regiões coerentes e, os nós internos, regiões não coerentes.

Hunter e Steiglitz [HunS79], Samet [Sam90], mostram que o número de nós em uma *quadtree* é de ordem $O(r + p)$, onde p é o perímetro da lista e r o nível máximo de subdivisão. Este nível pode atingir a resolução do espaço de imersão da lista. Como o número de testes interior/exterior, necessários para se construir a *quadtree*, depende linearmente do número de nós, o número de testes também é de ordem $O(r + p)$.

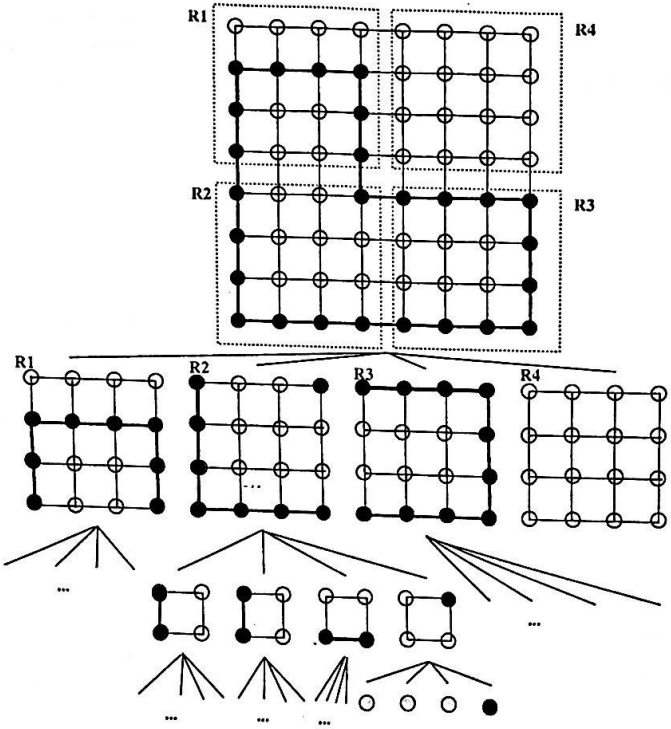


Figura 13: Algumas etapas do algoritmo de preenchimento com quadtrees.

Assim, o algoritmo com *quadtrees* tem complexidade de utilização de testes interior/exterior de Pol e Corthout [CorP92]

$$O(r+p)custo_{jordan}.$$

4.2 Coerência maximal

A técnica com *quadtrees* reduz significativamente a complexidade de utilização dos testes interior/exterior a um limite quasi-linear na resolução. Porém, o tamanho da região coerente encontrada em cada folha não é ideal, como mostra o exemplo da próxima página.

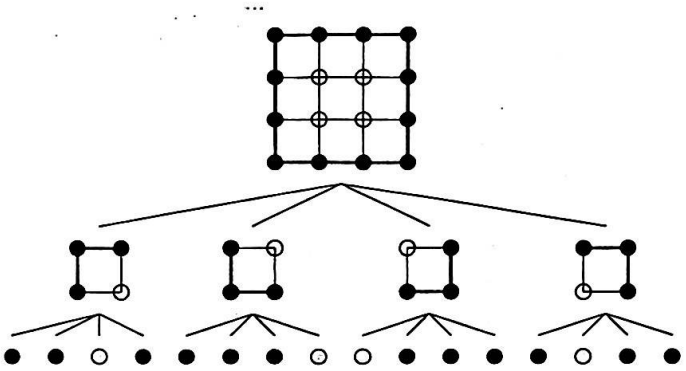


Figura 14: Regiões internas não-maximais.

As folhas contendo regiões coerentes internas poderiam ser agrupadas numa única região, coerente com L. Isto provém do fato que o teste com *quadtrees* encontra regiões coerentes retangulares, não necessariamente maximais com relação à coerência.

Porém, a definição operacional de coerência do Teorema 4.6 pode ser refinada para a produção de um algoritmo que encontra as regiões maximais, com número de testes interior/exterior independente da resolução.

Este resultado é fortemente baseado no Teorema Discreto de Jordan, que afirma que pontos dentro de uma mesma componente conexa, resultante da excisão da lista de $\mathbb{Z}^2$, têm o mesmo número de rotação.

O algoritmo é formado de dois passos:

- detecção de um representante de uma componente conexa do interior
- propagação da cor nesta componente

Primeiramente, serão mostrados o algoritmos e procedimentos auxiliares. Em seguida, serão analisadas a corretude e complexidade dos procedimentos e do algoritmo.

A fase de detecção, mostrada na próxima página, tenta encontrar um ponto no interior da lista, fazendo uma busca pela vizinhança 8-conexa de um ponto da lista.

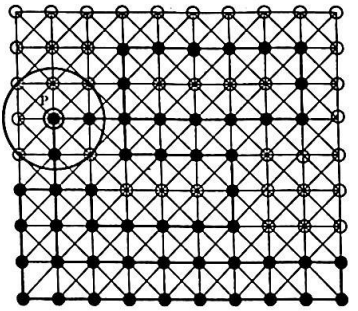


Figura 15: Detecção num ponto P

Function 1.1 Detecção de um ponto do interior.

Detecção(Ponto P,Lista L,Ponto Q): Booleano
Entrada: P - ponto da lista ...
L - Lista poligonal
Saída: TRUE - P tem vizinho interior, devolvido em Q
FALSE - P não tem vizinho interior.
1: **para cada** vizinho V de P, com cor diferente de COR_INTERIOR **faça**
2: se V está no interior de L **então**
3: Q←V
4: Devolva (TRUE)
5: **fim se**
6: **fim para**
7: Devolva (FALSE)

Como as curvas são conexas, observa-se facilmente que o número de testes interior/exterior é menor ou igual a 6: os dois vizinhos do ponto P que estão na curva não precisam ser testados. Assim, esta função tem complexidade $O(1)custo_{jordan}$.

Dado um ponto Q pertencente ao interior de uma lista L, o procedimento Propagação, mostrado na próxima página, assinala COR_INTERIOR a todos os pontos da região coerente que contém Q.

Procedimento 1.1 Propagação na componente interior.

```
Propagação(Ponto Q)
 Entrada:  Q - ponto do interior de uma lista
 Saída:    A todos os pontos da componente conexa que contém Q
           é atribuída a cor COR_INTERIOR
 1: Fila.criaFila()
 2: Fila.insere(Q)
 3: Cor(Q)←COR_INTERIOR
 4: enquanto Fila.Vazia()≠TRUE faça
 5:    P←Fila.retira()
 6:    para cada vizinho T de P tal que cor(T)≠COR_INTERIOR faça
 7:       cor(T)←COR_INTERIOR
 8:       Fila.insere(T)
 9:    fim para
10: fim enquanto
```

O tipo de propagação utilizado é baseado em busca em largura no grafo definido pela estrutura conexa de $\mathbb{Z}^2$. A Figura 16 mostra a um exemplo da ação deste algoritmo.

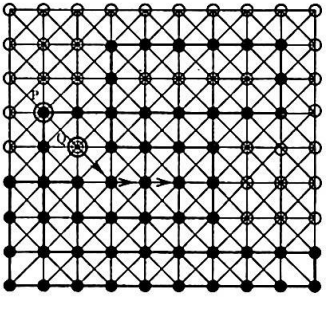


Figura 16: Propagação de cor a partir de um ponto Q.

Observa-se facilmente que este procedimento não faz nenhum teste interior/exterior. Porém, faz um outro tipo de operação básica: uma atribuição de cor ao pontos que estão no interior. Como o interior pode ocupar quase toda a imagem, esta operação faz $O(r^2)$ atribuições de cor na imagem toda, onde r representa a resolução da imagem.

Especificados a função de detecção de um ponto de uma componente interior e um

procedimento para preenchimento desta componente, um algoritmo para preenchimento de todas as componentes interiores pode ser contruído:

Algoritmo 1.5 Preenchimento de regiões com coerência maximal.

```
FillingCoerênciaMaximal(Regiao R,Lista L)
 Entrada:  L - lista descritora da curva discreta
           R - região onde L está imersa
 Saída:    Interior de L
 1: para cada ponto P∈R faça
 2:    cor(P)←COR_EXTERIOR
 3: fim para
 4: se L é uma lista de pontos de controle então
 5:    L←Conversão_Poligonal(L)
 6: fim se
 7: para cada ponto P∈L faça
 8:    se Detecção(P,L,Q)=TRUE então
 9:       Propagação(Q)
10:    fim se
11: fim para
```

Observa-se que, para cada ponto P da curva, é realizada uma chamada à função *Detecção(P,L,Q)*. Somente neste ponto é realizado algum teste interior/exterior. Logo, este algoritmo tem complexidade $O(p)$ em relação ao número de testes interior/exterior, onde p representa o perímetro da curva. A conversão poligonal, no caso das curvas de Bézier, pode ser efetuada com complexidade $O(p)$ e a propagação, no máximo, faz $O(n^2)$ atribuições de cor aos pontos da imagem.

Para assegurar o funcionamento correto da função Detecção, do procedimento Propagação e do algoritmo FillingCoerênciaMaximal, precisar-se-á das seguintes definições:

Definição 4.14. *Define-se o interior de uma lista fechada L, int(L), por:*

$$int(L) = \{P \in \mathbb{Z}^2 : \mathcal{F}(L, P) \neq 0\}.$$

Assim, o interior de uma lista$L$ corresponde aos pontos P, cujo número de rotação com respeito a L é diferente de 0. Estes pontos são chamados de *pontos interiores.*

Definição 4.15. *Um caminho (ou um $\mathbb{Z}^2$-caminho) entre dois pontos P_1 e P_2, chamados extremos, é uma seqüência $\{V_1 = P_1, V_2, \ldots, V_{n-1}, V_n = P_2\}$, onde $d(V_i, V_{i+1}) = 1$, para $i = 1, \ldots, n-1$. Os pontos V_i, $i \neq 1, n$, são chamados internos.*

Nota-se que esta definição é análoga à definição de lista: porém, caminhos não permitem a repetição de pontos consecutivos.

Uma conseqüência imediata desta definição é que, se W é uma região m-conexa, então sempre existe um caminho ligando P_1 e P_2, para todo P_1, $P_2 \in W$.

Proposição 4.12. *Seja L uma lista fechada e $P_1 \in int(L)$. Então existe um caminho, com extremos P_1 e P_2, com $P_2 \in L$, cujos pontos internos estão contidos em $int(L)$.*

A proposição anterior afirma que todo ponto do interior de uma lista L pode ser alcançado a partir de algum ponto desta lista.

Definição 4.16. *Seja W uma região e L uma lista fechada. W é chamada coerente conexa maximal se W é m-conexa, $\mathcal{F}(L,P) = \mathcal{F}(L,Q)$ para todo $P,Q \in W$ e se $V \subset \mathbb{Z}^2$ satisfaz:*

- $W \subseteq V$
- *V m-conexa*
- $\exists T \in V$, $\exists S \in W$, *tal que* $\mathcal{F}(L,T) = \mathcal{F}(L,S)$

então $V = W$.

Exemplos triviais de regiões coerentes conexas maximais são as regiões interiores do Teorema Discreto de Jordan.

Proposição 4.13. *Se um ponto Q é encontrado pela função* Detecção(P,L,Q), *então o procedimento* Propagação(Q) *atribui cor COR_INTERIOR a todos os pontos $P \in W$, onde W é uma região coerente conexa maximal que contém Q.*

O teorema a seguir mostra que o algoritmo *FillingCoerênciaMaximal(R,L)* encontra as regiões coerentes conexas maximais de L:

Teorema 4.7. *Seja L uma lista fechada. Então, o algoritmo* FillingCoerênciaMaximal(R,L) *encontra as regiões coerentes conexas maximais de L.*

Estabelecidas as provas de funcionamento dos algoritmos, a próxima seção compara suas complexidades.

4.2.1 Comparação das complexidades teóricas

Os três algoritmos apresentados anteriormente praticamente são formados de duas operações básicas:

- **teste**, que decide se um ponto está dentro ou fora da região de preenchimento;
- **atribuição**, que atribui cores (preto ou branco)[7] aos pontos testados.

Assim, a comparação téorica das complexidades destes algoritmos será feita com base nestas operações.

A complexidade da fase de teste já foi estabelecida para os três algoritmos ao longo deste capítulo, sumarizada na Tabela 1. Os algoritmos quadrático e com *quadtrees* fazem somente uma atribuição a cada ponto da imagem a ser gerada: assim, fazem $O(r^2)$ atribuições, onde r representa a resolução da imagem. O algoritmo de coerência maximal faz, no máximo, duas atribuições de cor a um mesmo ponto da imagem: uma para atribuição da cor COR_ EXTERIOR e desenho da curva, e outra, para os pontos do interior. Logo, este algoritmo faz também $O(r^2)$ atribuições.

A tabela abaixo compara a complexidade destas duas operações básicas para os três algoritmos:

Algoritmo	Número de Testes	Número de Atribuições
Quadrático	$O(r^2)$	$O(r^2)$
Coerência com quadtress	$O(r+p)$	$O(r^2)$
Coerência maximal	$O(p)$	$O(r^2)$

Tabela 1: Comparação das complexidades teóricas para preenchimento de região.

O passo mais custoso num algoritmo de preenchimento de região é justamente computar quem está dentro ou fora da região de interesse (fase de teste). A simples observação da Tabela 1 evidencia que o algoritmo com coerência maximal reduz significativamente a complexidade teórica da fase de teste, comparada com os outros dois algoritmos. Porém, esta redução pode parecer estar mascarada pela notação assintótica da complexidade: assim, foram implementados os três algoritmos, para poligonais e curvas de Bézier, a fim de evidenciar a redução da complexidade. Alguns detalhes de implementação e vários resultados práticos dos algoritmos são analisados nas duas próximas seções.

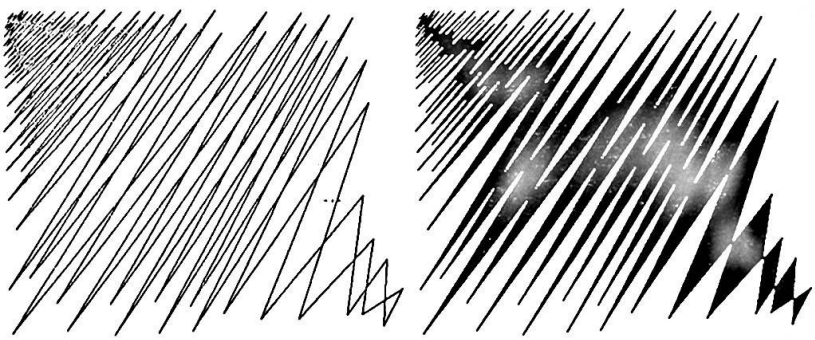
[7] Ou, alternativamente, cor do interior e exterior.

4.2.2 Alguns resultados numéricos

A partir dos algoritmos anteriores, foram efetuados vários testes para averiguar o comportamento da implementação e comparar resultados em situações práticas. Mediu-se apenas a quantidade de testes interior/exterior feitos pelos algoritmos, uma vez que as atribuição de cor aos pontos é, essencialmente, a mesma.

O ambiente dos testes foi um computador IBM Pentium 200 MHz. A resolução das imagens é sempre 512×512, a menos que explicitado o contrário. As imagens geradas foram armazenadas no formato *bitmap*, escaladas a 40%, convertidas para PostScript e incluídas nesta dissertação através do pacote **graphicx**, disponível para LaTeX. A impressão foi efetuada numa impressora HPLaserJet 5Si a 600 dpi.

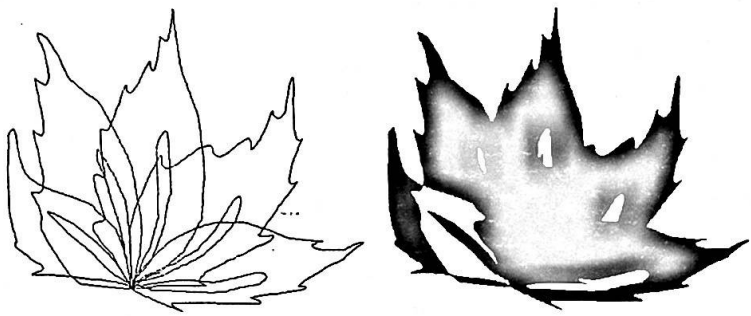
Os dois primeiros resultados mostram a habilidade do algoritmo com coerência maximal, juntamente com o teste interior/exterior de Pol e Corthout [CorP92], em detectar pequenos detalhes, conforme mostra o exemplo das poligonais, e tratar várias sobreposições de curvas, como mostra o exemplo das curvas de Bézier.



(a) Poligonais (b) Poligonais preenchidas

Figura 17: Preenchimento de poligonais.

Um ponto importante refere-se à distribuição das curvas na imagem a ser produzida, pois ela é fator determinante na quantidade de regiões coerentes para as *quadtrees*. Assim, foram gerados exemplos desta distribuição e comparados os resultados.



(a) Curvas de Bézier (b) Curvas preenchidas

Figura 18: Preenchimento de curvas de Bézier.

O primeiro exemplo mostra o caracter japonês Hon, formado por 28 segmentos de curvas de Bézier, que se distribuiu por grande parte da imagem.

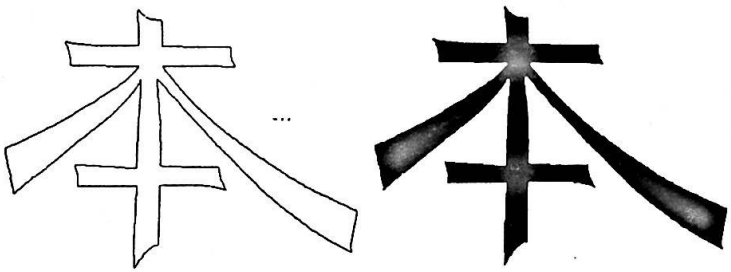


Figura 19: Preenchimento do caracter Hon.

A tabela 2, mostrada a seguir, compara o número de testes efetuados por cada um dos algoritmos.

Algoritmo	Número de Testes
Quadrático	262.144
Coerência com quadtrees	5.329
Coerência maximal	4.200

Tabela 2: Comparação do número de testes para o caracter Hon.

O segundo exemplo foi construído com outro caracter japonês, Ni, concentrado no eixo vertical médio da imagem. Este caracter foi formado por 14 segmentos de curvas de Bézier.

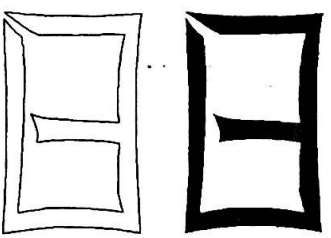


Figura 20: Preenchimento do caracter Ni.

A Tabela 3, mostrada a seguir, traz o número de comparações para este exemplo.

Algoritmo	Número de Testes
Quadrático	262.144
Coerência com quadtrees	5.206
Coerência maximal	4.347

Tabela 3: Comparação do número de testes para o caracter Ni.

O terceiro exemplo concentrou o caracter japonês Na, formado por 22 segmentos de curvas de Bézier, na parte superior esquerda da imagem.

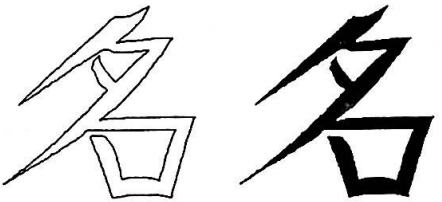


Figura 21: Preenchimento do caracter Na.

A Tabela 4 mostra o resultado do número de comparações:

Algoritmo	Número de Testes
Quadrático	262.144
Coerência com quadtrees	3.799
Coerência maximal	2.828

Tabela 4: Comparação do número de testes para o caracter Na.

Antes de se passar aos próximos testes, reduziu-se os tamanhos dos caracteres Ni e Hon a patamares próximos de um texto de tamanho normal, cujo efeito é mostrado na figura abaixo.

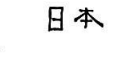
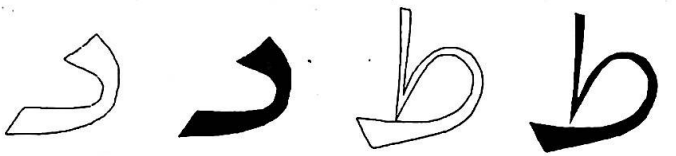


Figura 22: Caracteres japoneses preenchidos e reduzidos.

Exemplos adicionais foram construídos com caracteres de outras línguas, como os caracteres da língua árabe, mostrados a seguir.



(a) Daal (b) Preenchimento (c) Taa (d) Preenchimento

Figura 23: Preenchimento de caracteres árabes.

O caracter Daal é formado por 7 segmentos de curvas de Bézier e, o caracter Taa, por 10 segmentos. A Tabela 5 compara o número de testes:

Algoritmo	Número de Testes	
	Daal	Taa
Quadrático	262.144	262.144
Coerência com quadtrees	1.894	3.520
Cocrência maximal	1.369	2.600

Tabela 5: Comparação do número de testes para os caracteres árabes.

Se o interior ou exterior de uma curva discreta for muito coerente com respeito à divisão efetuada pelas *quadtrees*, o números de folhas nestas árvores fica próximo do perímetro da curva. Assim, o número de testes efetuados pelo algoritmo com coerência maximal pode superar o algoritmo com quadtrees, conforme mostra o exemplo a seguir:

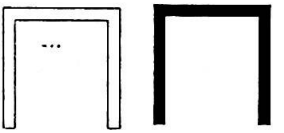


Figura 24: Preenchimento de poligonais com alto grau de coerência no interior e exterior.

A Tabela 6 compara o número de testes neste exemplo:

Algoritmo	Número de Testes
Quadrático	262.144
Coerência com quadtrees	2.500
Coerência maximal	2.947

Tabela 6: Comparação do número de testes para poligonais.

Foram também gerados vários testes mantendo-se uma determinada curva e variando-se o tamanho da imagem a ser gerada. O método com *quadtrees* não sofreu muita grandes variações nestes testes. O método com coerência maximal não sofreu variações na fase de testes, uma vez que tal fase independe do tamanho da imagem.

4.3 *Stroking*

A operação de *stroking* destina-se a modelar a forma produzida quando uma caneta ou, formalmente, uma *brush* é movida sobre um papel(ou *canvas*), ao longo de uma trajetória. Esta operação tem importantes aplicações, principalmente na produção de fontes.

4.3.1 Descrição da operação

Dadas duas regiões, chamadas *brush* e *caminho*, o resultado da operação de *stroking* utilizando estes elementos é definida como o conjunto de pontos formado pela união de imagens transladadas da *brush*, de tal modo que a origem da *brush* seja mapeada em algum ponto do caminho.

Esta operação pode ser formalizada em termos da soma de Minkowski:

Definição 4.17. *Sejam T e B duas regiões em $\mathbb{Z}^2$. Define-se a região de* stroking *T e brush B como o conjunto $T \oplus B$, com predicado de pertinência $\tau_{T\oplus B}(P)$.*

O predicado de pertinência $\tau_{T\oplus B}(P)$ decide se um ponto P pertence ou não à região $T \oplus B$.

Pol e Corthout [CorP92] descrevem uma implementação desta operação para curvas de Bézier discretas 8-conexas, descrito na próxima seção. Nota-se que a conectividade citada não representa uma restrição no espaço das curvas aceitas pelo algoritmo: curvas 4 e 6-conexas podem ser facilmente incorporadas ao algoritmo.

4.3.2 O algoritmo quadrático

Na implementação da operação de *stroking*, desenvolvida por Pol e Corthout [CorP92], primeiramente houve a especificação do predicado de pertinência $\tau_{T \oplus B}(P)$. Esta implementação requer dois itens associados com a brush:

- uma *bounding box* para a *brush*
- um predicado de pertinência para a *brush* $\tau_B : \mathbb{Z}^2 \times \mathbb{Z}^2 \rightarrow Boolean$.

A *bounding box* serve para acelerar os casos de rejeição na implementação do predicado de pertinência $\tau_{T \oplus B}$ e não necessita ser a menor *box* que contenha a *brush*. Porém, a performance do algoritmo será tanto maior quanto mais ajustada for a *bounding box*. O predicado de pertinência $\tau_B(Q, P)$ determina se um ponto P é coberto pela *brush* B transladada de Q. Por exemplo, uma brush circular de raio r pode delimitada por uma *bounding box* de lado $2r$ e especificado um predicado de pertinência pelo teste $\tau_B(Q, P)$: $(x_p - x_q)^2 + (y_p - y_q)^2 \leq r^2$.

O primeiro passo do algoritmo será detectar se um ponto P não está na região $T \oplus B$. Para um teste de rejeição rápida, basta verificar se P não está contido em $B_{box}(T) \oplus B_{Box}(B)$, onde $B_{box}(T)$ e $B_{Box}(B)$ representam *bounding boxes* do caminho e da *brush*, respectivamente. A corretude deste teste advém imediatamente da monotonicidade da soma de Minkowski. Caso o ponto P não satisfaça este teste de rejeição, o segundo passo do algoritmo será detectar se P é coberto pela *brush* colocada nos pontos extremos da curva: caso afirmativo, o algoritmo pára; caso negativo, a curva é dividida em duas partes e o processo continua recursivamente, num processo tipo divisão-e-conquista. A fase de divisão justifica-se pela distributividade da união em relação à soma de Minkowski.

O procedimento mostrado a seguir, contém a especificação do teste de pertinência para a origem.

Procedimento 1.2 Teste de pertinência $\tau_{T \oplus B}(Origem)$.

```
TesteStroking(Lista T,Brush B)
 Entrada:  T - Lista contendo os pontos de controle de uma curva de Bézier discreta.
           B - Brush.
 Saída:    TRUE - origem está contida em T ⊕ B.
           FALSE - caso contrário.
1: se #T = 0 então
2:    Devolva(FALSE)
3: fim se
4: se Origem ∉ B_box(T) ⊕ B_box(B) então
5:    Devolva(FALSE)
6: fim se
7: se Origem τ_B(T_0, Origem)=TRUE ou τ_B(T_#T, Origem)=TRUE então
8:    Devolva(TRUE)
9: fim se
10: se TesteStroking(Parte_Esquerda(T),B)=TRUE então
11:    Devolva(TRUE)
12: senão
13:    se TesteStroking(Parte_Direita(T), B)=TRUE então
14:       Devolva(TRUE)
15:    fim se
16: fim se
17: Devolva(FALSE)
```

A recursividade gerada por este procedimento tem caráter finito, pois o processo de subdivisão das curvas de Bézier discretas, Parte_Esquerda(T) e Parte_Direita(T), é finito, conforme explicitado na definição do operador de conversão curva–polígono.

O procedimento anterior pode ser utilizado para testar quaisquer outros pontos P: basta transladar a lista T de $-P$ antes de efetuar o teste. Observa-se que o procedimento utiliza, no máximo, $O(p)$ chamadas ao predicado de pertinência da *brush*, onde p representa o perímetro da curva representada pela lista T. Na maioria dos casos, a computação do predicado de pertinência para a *brush* pode ser computado com complexidade $O(1)$: tome-se como exemplo as *brushes* quadradas e circulares, usuais em aplicações práticas.

Utilizando-se este procedimento, existe um algoritmo natural para se calcular a região $T \oplus B$: para cada ponto P da imagem a ser gerada, verifica-se se $P \in T \oplus B$.

Isto pode ser especificado em termos do procedimento anterior:

Algoritmo 1.6 Stroking quadrático.

Stroking(Lista T, Brush B)

Entrada: T - Lista contendo os pontos de controle de uma curva de Bézier discreta.
B - Brush.

Saída: $T \oplus B$.

1: **para todo** $P \in$ *Imagem* **faça**
2: **se** Se TesteStroking(T-P,B)=TRUE **então**
3: Pintar P na imagem
4: **fim se**
5: **fim para**

Para se calcular a complexidade deste algoritmo, primeiramente observa-se que são efetuados r^2 testes de pertinência contra a região $T \oplus B$, onde r indica a resolução da imagem a ser gerada. Assim, este número de testes tem complexidade quadrática com respeito à resolução.

Apesar de polinomial na resolução, este algoritmo é considerado lendo na maioria das aplicações práticas, como *antialiasing* [Fab95] [FabF97]. A próxima seção aborda alguns métodos para redução desta complexidade, baseada em noções de *coerência*.

4.3.3 Redução da complexidade

A utilização do predicado de pertinência de Pol e Corthout [CorP92], no algoritmo trivial para calcular a região de *stroking* $T \oplus B$, tem comportamento quadrático com respeito à resolução. Para se reduzir esta complexidade, serão utilizados dois mecanismos de localização rápida de pontos pertencentes à região de *stroking*: *testes de coerência com quadtrees* e *testes baseados em* Π*-coerência*.

4.3.4 Coerência com *quadtrees*

O objetivo da coerência com quadtrees é localizar regiões, na imagem a ser gerada, que contenham exclusivamente pontos da região de *stroking* ou pontos fora dela. Para estabelecimento efetivo dos testes de coerência com quadtrees, é necessário a formalização do conceito de região coerente com respeito à região de stroking S.

Definição 4.18. *Seja S uma região produzida pela operação de stroking. Uma região R é chamada coerente, com respeito a S, se e somente se $R \cap S = R$ ou $R \cap S = \emptyset$.*

Esta definição equivale a dizer que uma região R é coerente, com respeito s S, se e somente se R está contida em S ou no seu complemento.

A seguinte proposição é usada para detectar se uma região R é coerente com respeito a uma região S:

Proposição 4.14. *Sejam R e S' regiões. Então valem as seguintes asserções:*

$$(P - R) \cap S' = \emptyset \Leftrightarrow P \notin S' \oplus R$$

$$(P - R) \subset S' \Leftrightarrow P \in S' \ominus R.$$

A figura 25 mostra duas regiões coerentes detectadas pelo teste da proposição anterior:

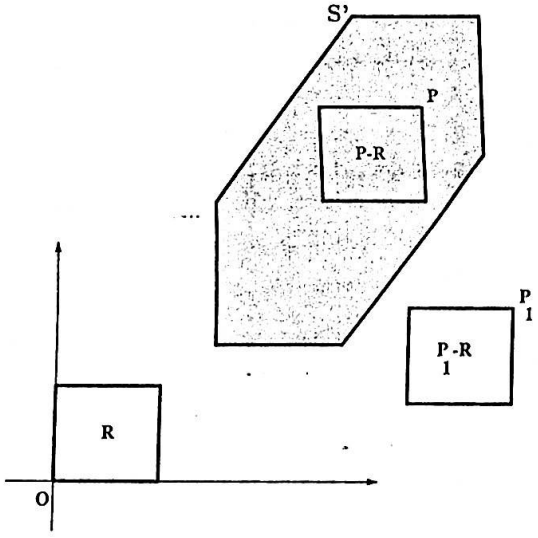


Figura 25: Regiões coerentes com respeito a uma região S'.

Como a região S' pode ser a soma de Minkowski de um caminho T e uma *brush* B, a Proposição 4.14 afirma que testar se $P - R$ é coerente com respeito a $T \oplus B$ é equivalente a checar se $P \in (T \oplus B) \oplus R$ ou $P \in (T \oplus B) \ominus R$. O teste $P \in (T \oplus B) \oplus R$ pode

se colocado na forma $P \in T \oplus (B \oplus R)$, usando a propriedade associativa da soma de Minkowski . O teste $P \in (T \oplus B) \ominus R$ pode ser refinado testando se $P \in T \oplus (B \ominus R)$, uma vez que $T \oplus (B \ominus R) \subset (T \oplus B) \ominus R$.

Como a região de *stroking* cobre, usualmente, somente uma pequena porção da imagem, é mais eficiente começar a detecção pelo coerência no complemento da região. Para isto, utiliza-se o teste da primeira asserção da proposição 4.14, com a modificação do teste $P \in (T \oplus B) \oplus R$ mencionada anteriormente. Se o teste pela coerência no exterior falha, utiliza-se o teste no interior, utilizando-se a segunda asserção da Proposição 4.14, com o refinamento do teste $P \in (T \oplus B) \ominus R$ do parágrafo anterior. Seguindo-se esse método, pode ser derivado um algoritmo para calcular o *stroking* de curvas discretas com coerência. Quando uma imagem é para ser gerada, primeiro se detecta se a imagem inteira é coerente. Caso afirmativo, determina-se qual a cor de seus pontos. Caso negativo, a imagem é subdividida em quatro quadrantes, e o aplica-se recursivamente o algoritmo, criando uma estrutura de *quadtree*. Isto pode ser especificado pelo algoritmo a seguir:

Algoritmo 1.7 Stroking $T \oplus B$ com quadtrees.

StrokingComQuadtrees(Região R,Lista T,Brush B)

Entrada: R - região de interesse para preenchimento
T - Lista contendo os pontos de controle de uma curva de Bézier discreta.
B - Brush.

Saída: $T \oplus B$.

```
1: se R é coerente com respeito ao exterior de T ⊕ B então
2:    Retornar da chamada recursiva
3: senão
4:    se R é coerente com respeito ao interior de T ⊕ B então
5:       Pintar todos os pontos de R
6:    senão
7:       Subdividir R em quatro partes R1,R2,R3,R4
8:       StrokingComQuadtrees(R1,T,B)
9:       StrokingComQuadtrees(R2,T,B)
10:      StrokingComQuadtrees(R3,T,B)
11:      StrokingComQuadtrees(R4,T,B)
12:   fim se
13: fim se
```

A Figura 27 mostra um exemplo de aplicação deste algoritmo, onde considerou-se um

caminho como um segmento e curva de Bézier e a *brush*, um círculo de raio 10.

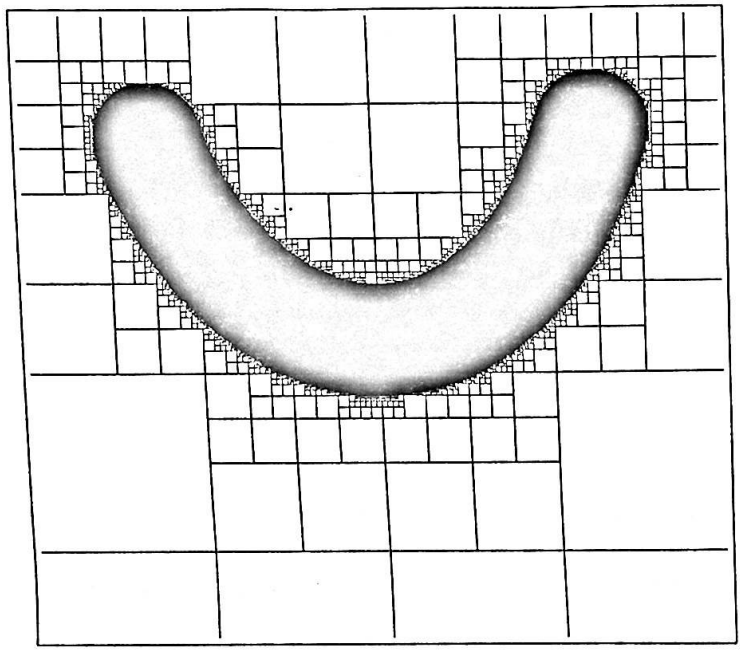


Figura 26: Exemplo de stroking com *quadtrees*.

Nota-se que este algoritmo sempre termina, pois em última instância de recursão tem-se um ponto: um ponto é sempre coerente com respeito a uma região de *stroking*.

A estrutura recursiva do Algoritmo 1.7 cria uma quadtree. Hunter [HunS79] e Samet [Sam90 mostram que o número de nós em uma *quadtree* é de ordem $O(r+p)$, onde p é o perímetro da curva discreta representada pela lista e r o nível máximo de subdivisão. Este nível pode atingir a resolução da imagem a ser gerada: assim, r pode ser entendido como a resolução da imagem. Para cada uma das regiões R candidatas à coerência, existe um ponto P associado. Para cada um destes pontos, precisa ser verificado se $P \in T \oplus (B \oplus R)$ e, em alguns casos, se $P \in T \oplus (B \ominus R)$. Para cada um destes pontos, são necessários $O(p)$ testes para efetiva computação do resultado. Assim, o algoritmo com *quadtrees* tem complexidade de testes $O((r+p)p)$.

4.3.5 Π-coerência

O método com *quadtrees* envolve muitos testes com as dilatações $B \oplus R$ e erosões $B \ominus R$, pois muitas regiões R podem ser geradas no processo recursivo para formar a *quadtree*. Para reduzir o número de tais dilatações, será proposto um novo método baseado num tipo diferente de coerência: a Π-coerência.

Definição 4.19. *Uma região R é Π-coerente com respeito a $T \oplus B$ se e somente se $T \oplus B \subseteq R$.*

Um exemplo simples de região Π-coerente é $T \oplus Q_{Box}(B)$, onde $Q_{Box}(B)$ denota uma *bounding box* quadrada da *brush* B. Isto pode ser provado facilmente utilizando-se o fato que $Q_{Box}(B) \subseteq R$ e aplicando-se a monotonicidade da soma de Minkowski. A escolha da *bounding box* quadrada não foi uma escolha aleatória: como o ambiente de trabalho é 8-conexo, esta brush permitirá uma computação eficiente de uma região Π-coerente.

O método para *stroking* utilizando o conceito de Π-coerência utiliza dois procedimentos auxiliares:

- *Cobertura*, que tem a finalidade de calcular uma região Π-coerente;
- *Refinamento*, cuja finalidade é extrair a região $T \oplus B$ da região Π-coerente calculada pelo procedimento Cobertura, utilizando a implementação do predicado de pertinência $\tau_{T \oplus B}$ desenvolvido por Pol e Corthout [CorP92].

O Procedimento Cobertura calcula a seguinte região Π-coerente:

$$\bigcup_{i=0}^{\#T} \{T_i \oplus Q_{Box}(B)\},$$

isto é, limita a região de stroking $T \oplus B$ pela cobertura $T \oplus Q_{Box}(B)$.

O cálculo de $T_i \oplus Q_{Box}(B)$ tem alto custo computacional para a lista toda. Usualmente, $\{T_i \oplus Q_{Box}(B)\} \cap \{T_{i+1} \oplus Q_{Box}(B)\} \neq \emptyset$: assim, evitar a computação dos pontos desta intersecção diminui o custo de cálculo da região Π-coerente descrita acima.

Tendo-se calculado a dilatação para uma parte da lista, isto é, $\bigcup_{i=0}^{k-1} \{T_i \oplus Q_{Box}(B)\}$, para $1 \leq k < \#T$, pode-se computar facilmente a contribuição da dilatação $T_k \oplus Q_{Box}(B)$ pelos casos mostrados na Figura 27:

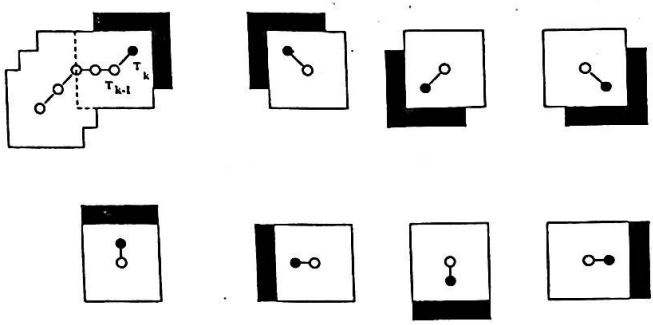


Figura 27: Contribuição da dilatação $T_k \oplus Q_{Box}(B)$ para $\bigcup_{i=0}^{k-1}\{T_i \oplus Q_{Box}(B)\}$.

Na implementação dos casos mostrados na figura, as regiões escuras representam linhas e colunas de pontos. Se $T_k = T_{k-1}$, então a computação da contribuição não é necessária. Observa-se que a computação de tal contribuição funciona porque está-se trabalhando num ambiente 8-conexo e pela escolha da *bounding box* quadrada. Utilizando-se esta forma de computar a contribuição, tem-se a seguinte especificação do procedimento Cobertura:

Procedimento 1.3 Cobertura da região de stroking.

Cobertura(Lista T,Brush B)
Entrada: T - Caminho (Curva Discreta)
B - Brush.

Saída: $\bigcup_{i=0}^{\#T}\{T_i \oplus Q_{Box}(B)\}$
1: $U \leftarrow T_0 \oplus B$
2: **para todo** T_i, $i = 1, ..., \#T$ **faça**
3: $U \leftarrow U \bigcup$ Contribuição de $T_i \oplus B$
4: **fim para**
5: Devolva (U)

A variável U do Procedimento 1.3 representa a estrutura de dados que irá armazenar a imagem final.

A figura 28 mostra os primeiros passos deste procedimento usando uma *brush* do tipo diamante:

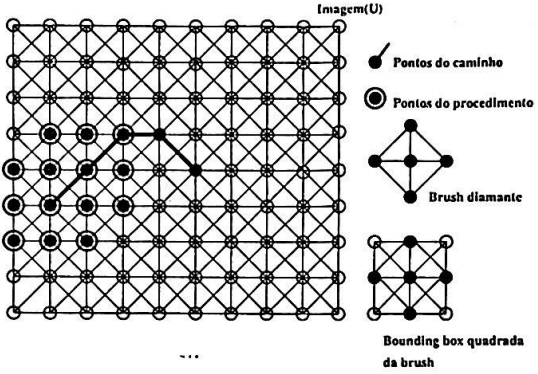


Figura 28: Cobertura com uma brush do tipo diamante.

Dada uma região Π-coerente U, com respeito à região de stroking $T \oplus B$, o procedimento Refinamento calcula a intersecção $U \cap (T \oplus B)$, usando o algoritmo de stroking básico.

Procedimento 1.4 Refinamento de uma região Π-coerente.

Refinamento(Lista T,Brush B,Region U)
Entrada: T - Pontos de controle da curva
B - Brush
U - Região Π-coerente
Saída: $T \oplus B$
1: **para todo** $P \in U$ **faça**
2: **se** Stroking(T,B,P)=FALSE **então**
3: Retire P de U
4: **fim se**
5: **fim para**
6: Devolva(U)

A Figura 29 mostra um exemplo de acão deste procedimento, usando uma *brush* do tipo diamante:

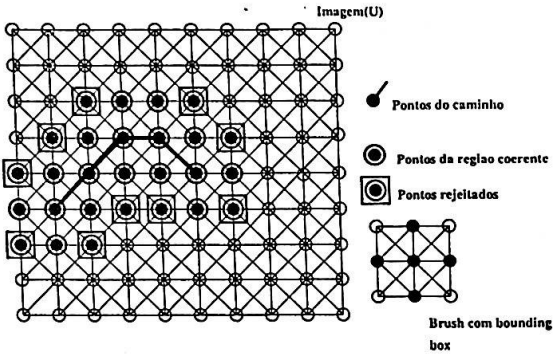


Figura 29: Refinamento de uma região Π-coerente.

Especificados os procedimentos anteriores, pode-se derivar o seguinte algoritmo para a operação de *stroking* usando Π-coerência:

Algoritmo 1.8 Stroking com Π-coerência.

StrokingComPiCoerencia(Lista T,Brush B)

Entrada: T - Lista contendo os pontos de controle de uma curva de Bézier discreta.
B - Brush.

Saída: U - Região de stroking $T \oplus B$

1: L←Curva de Bézier Discreta com pontos de controle de T
2: U←Cobertura(L,B)
3: U←Refinamento(T,B,U)
4: Devolva (U)

Para assegurar o correto funcionamento do Algoritmo 1.8, precisa-se provar que os procedimentos Cobertura(T,B) e Refinamento(T,B,U) cumprem suas funções. O primeiro resultado estabelece que a o procedimento Cobertura(T,B) consegue computar uma região Π-coerente com respeito a $T \oplus B$.

Teorema 4.8. *O procedimento Cobertura(T,B) calcula uma região coerente U com respeito à região de stroking* $T \oplus B$.

...

Calculada uma região U Π-coerente com respeito a $T \oplus B$, o próximo resultado prova que o procedimento Refinamento(T,B,U) consegue extrair a região $T \oplus B$ de U.

Teorema 4.9. *O procedimento Refinamento(T,B,U) calcula a região $T \oplus B$ de uma região Π-coerente U.*

Para se computar o número de testes efetuados pelo Algoritmo 1.8 na região $T \oplus B$, é necessário estimar a cardinalidade da região $\bigcup_{i=0}^{\#T}\{T_i \oplus Q_{Box}(B)\}$, retornada pelo procdimento Cobertura. Para isto, considere-se c o número de pontos numa linha (ou coluna) da região $Q_{Box}(B)$. No pior caso, quando o caminho é uma diagonal, tem-se no máximo $c^2 + 2c(p-1)$ pontos na região de teste para refinamento, onde p representa o perímetro do caminho: c^2 provém da cardinalidade de $T_0 \oplus Q_{Box}(B)$ e $2c(p-1)$ da quantidade das contribuições dos pontos T_i , $i = 1, \ldots, \#T$. Assim, tem-se $O((c+p)c)$ testes na região $T \oplus B$. Para se comparar esta complexidade com a obtida no algoritmo que usa *quadtrees*, $O((r+p)p)$ são necessárias algumas considerações. Usualmente, o tamanho da *brush* B é muito menor que a imagem a ser gerada na operação de *stroking*. Deste fato, tem-se $c < r$. Além disto, também é usual o tamanho da *brush* ser menor que o tamanho do caminho T: nestes casos, tem-se $c < p$. Estes afirmações permitem concluir que, se o tamanho da *brush* for bem menor que os tamanhos da imagem a ser gerada e do caminho, o algoritmo com Π-coerência reduz a complexidade do número de testes necessários para se produzir o resultado da operação de *stroking*. Esta dependência do tamanho da *brush* será melhor reafirmada nos testes resultantes da implementação, mostrados nas próximas seções.

A próxima seção compara de maneira mais efetiva os três algoritmos apresentados.

4.3.6 Comparação das complexidades teóricas

Os três algoritmos apresentados anteriormente praticamente são formados de duas operações básicas:

- **teste**, que decide se um ponto está dentro ou fora da região $T \oplus B$;
- **atribuição**, que atribui cores (preto ou branco)[8] aos pontos testados.

Assim, como na operação de preenchimento, a comparação téorica das complexidades destes algoritmos será feita com base nestas operações.

[8] Ou, alternativamente, cor do interior e exterior.

A complexidade da fase de teste já foi estabelecida para os três algoritmos ao longo deste capítulo, sumarizada na Tabela 7. Os algoritmos quadrático e com *quadtrees* fazem somente uma atribuição a cada ponto da imagem a ser gerada: assim, fazem $O(r^2)$ atribuições, onde r representa a resolução da imagem. O algoritmo de Π-coerência faz, no máximo, três atribuições de cor a um mesmo ponto da imagem: a primeira, que coloca todos os pontos da imagem com a cor do exterior da região $T \oplus B$; a segunda, resultante do processo de cobertura; a terceira, resultante do processo de refinamento. Logo, este algoritmo também faz $O(r^2)$ atribuições.

A tabela abaixo compara a complexidade destas duas operações básicas para os três algoritmos:

Algoritmo	Operação de teste	Operação de atribuição
Quadrático	$O(r^2)$	$O(r^2)$
Coerência com quadtrees	$O((r+p)p)$	$O(r^2)$
Π-Coerência	$O((c+p)c)$	$O(r^2)$

Tabela 7: Comparação das complexidades teóricas para contorno de caminhos.

O passo mais custoso num algoritmo de contorno de caminhos é justamente computar quem está dentro ou fora da região de interesse (fase de teste). A simples observação da Tabela 7 não evidencia diretamente uma redução da complexidade do algoritmo de Π-coerência em relação ao algoritmo com *quadtrees*, em virtude da dependência do tamanho da *brush*. Para se verificar situações práticas em que há redução do número de testes, foram implementados os três algoritmos curvas de Bézier. Alguns detalhes de implementação e vários resultados práticos dos algoritmos são analisados nas duas próximas seções.

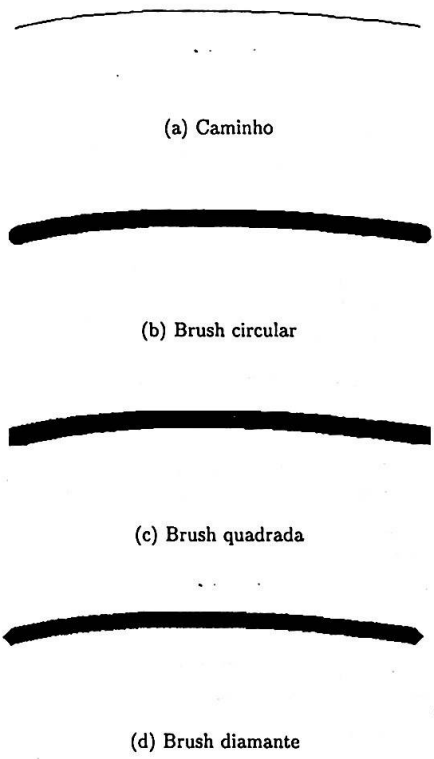
4.3.7 Alguns resultados numéricos

A partir dos algoritmos anteriores, foram efetuados vários testes para averiguar o comportamento da implementação e comparar resultados em situações práticas. Mediu-se apenas a quantidade de testes interior/exterior feitos pelos algoritmos, uma vez que as atribuição de cor aos pontos é, essencialmente, a mesma.

O ambiente dos testes foi um computador IBM Pentium 200 MHz. A resolução das imagens é sempre 512×512, a menos que explicitado o contrário. As imagens geradas foram armazenadas no formato *bitmap*, escaladas a 50%, convertidas para PostScript e incluídas nesta dissertação através do pacote **graphicx**, disponível para LaTeX. A impressão foi

efetuada numa impressora HPLaserJet 5Si a 600 dpi.

Os primeiros resultados apenas mostram o efeito das três *brushes* implementadas num segmento de curva de Bézier como caminho. Nestes exemplos, usou-se um tamanho de *brush* igual a 10.



(a) Caminho

(b) Brush circular

(c) Brush quadrada

(d) Brush diamante

Figura 30: Efeitos da variação do tipo de *brush*.

As *brushes* do tipo circular e quadrada são largamente utilizadas na maioria dos sistemas de pintura.

O próximo resultado utilizou o caracter japonês Ni, com uma brush de tamanho baixo (4):

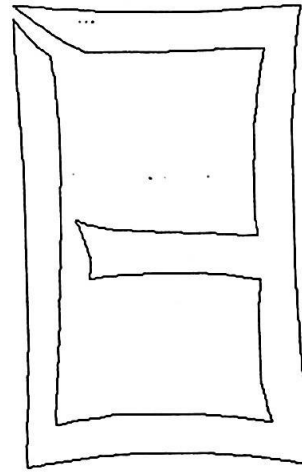


Figura 31: Curvas do caracter Ni.

Algoritmo	Número de Testes		
	Brush circular	Brush quadrada	Brush diamante
Quadrático	262.144	262.144	262.144
Quadtrees	146.167	147.101	157.104
Π-coerência	18.104	18.104	18.104

Tabela 8: Comparação do número de testes de contorno para o caracter Ni.

Os resultados da Tabela 8 reforçam a tese que, quanto menor o tamanho da *brush*, mais eficiente será o método de Π-coerência. Além disto, nota-se uma variação do número de testes quando ocorre uma mudança de *brush* no algoritmo com *quadtree*. Isto não ocorre com o mecanismo de *Pi*-coerência: mantendo-se o tamanho constante, não ocorre mudança na *bounding box* quadrada da *brush*.

O próximo exemplo mostra uma situação em que a *brush* tem tamanho grande (30).

Neste exemplo, foi utilizado o caracter japonês Hon, distribuído por grande parte da imagem.

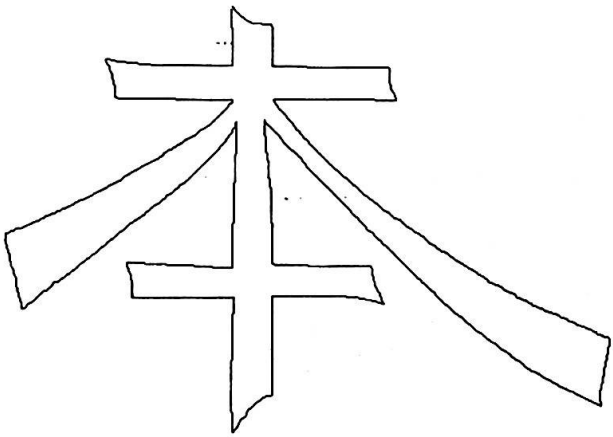


Figura 32:. Curvas do caracter Hon.

Algoritmo	Número de Testes		
	Brush circular	Brush quadrada	Brush diamante
Quadrático	262.144	262.144	262.144
Quadtrees	94.728	88.645	98.215
Π-coerência	109.755	109.755	109.755

Tabela 9: Comparação do número de testes de contorno para o caracter Hon.

Neste exemplo, além de uma grande distribuição da curvas pela imagem, o tamanho da *brush*, relativamente grande, contribuiu para a queda da performance do algoritmo de Π-coerência, conforme explicitado na Tabela 9. Isto já era esperado na comparação teórica das complexidades.

Para reforçar a dependência do tamanho da *brush* no desempenho do algoritmo de Π-coerência, foi gerado um exemplo de pior caso para entrada do algoritmo (segmento diagonal).

Figura 33: Segmento diagonal na imagem (redução de 30%).

Variando-se o tamanho da *brush* circular, obteve-se a seguinte tabela:

Tamanho	Quadrático	Quadtrees	Π-coerência
4	262.144	69.206	5.262
6	262.144	72.755	8.176
8	262.144	74.266	10.869
10	262.144	76.617	13.379
20	262.144	80.063	26.478
30	262.144	83.174	38.925
40	262.144	86.187	50.975
60	262.144	86.054	73.812
80	262.144	82.138	95.391
100	262.144	77.984	114.976

Tabela 10: Efeitos da variação do tamanho da *brush*.

A dependência do tamanho da *brush* é notória pelos resultados da Tabela 10. Além disto, ocorre um efeito interessante com o algoritmo de *quadtrees*: o número de testes cresce até um certo ponto, quando passo a cair. O mesmo não acontece com o algoritmo de Π-coerência, que tem comportamento crescente com o aumento do tamanho da *brush*.

4.4 *Antialiasing*

Os algoritmos de prenchimento e *stroking* podem ser utilizados em várias aplicações importantes. Este capítulo mostra uma aplicação destes algoritmos para uma técnica de *antialiasing*.

4.4.1 A necessidade de *antialiasing*

Sistemas gráficos de exibição matricial são dispositivos discretos: em tais sistemas, uma imagem é gerada atribuindo-se uma intensidade para cada pixel pertencente a uma matriz bidimensional finita de *pixels*. Essa natureza discreta acarreta um dos principais problemas dos sistemas de exibição matricial: a inabilidade de se obter contornos contínuos e de se exibir os detalhes finos de uma cena. Um exemplo familiar é o aspecto serrilhado das imagens de retas e contornos num dispositivo de saída do tipo CRT (*Cathode Ray Tube*). Outro exemplo é a cintilação devida a pequenos objetos em movimento, que aparecem e desaparecem numa exibição dinâmica. Tais defeitos ocorrem quando uma função de variáveis contínuas que contém abrupta alteração de intensidade é aproximada por amostras discretas. O erro desse processo de discretização foi originariamente chamado *aliasing* em Teoria do Processamento de Sinais e esta terminologia tem sido usada em Computação Gráfica.

O conjunto de técnicas para atenuar os efeitos de *aliasing* é conhecido comumente como *antialiasing*.

Para exemplificar estes mecanismos, a Figura 35 mostra duas versões de uma mesma curva discreta:

(a) Aliasing (b) Antialiasing

Figura 34: Efeitos de *aliasing* e *antialiasing*.

Na versão mostrada em (a), não foi efetuado nenhum tratamento para *aliasing*, onde são notórios os efeitos do tipo serrilhado. Em (b) foi aplicada uma técnica de *antialiasing*, cujo aspecto é visivelmente mais confortável.

4.4.2 A técnica de Fabris e Forrest para *antialiasing*

Pré-filtragem é geralmente considerada a abordagem ideal para antialiasing, mas tem sido difícil de ser implementada com exatidão para geometrias complexas, tais como curvas ou para uma escolha arbitrária de filtros.

Em [Fab95, FabF97, FabF97b], Fabris e Forrest desenvolveram uma técnica para antialiasing de curvas bidimensionais contínuas baseada em pré-filtragem que, não só evita os problemas geométricos e numéricos encontrados em técnicas anteriores, mas também permite o uso de uma classe genérica de filtros, o que a torna aplicável a diferentes dispositivos de saída.

O processo de rasterização utilizado em [Fab95, FabF97, FabF97b] consiste no teste interior/exterior, descrito nos capítulos anteriores. Neste processo, não houve uma preocupação especial no que diz respeito à eficiência. O uso de um teste interior/exterior com sua complexidade original - quadrática com relação à resolução - torna a implementação consideravelmente lenta. Utilizando-se o mecanismo de Π-coerência, a implementação da técnica tornou-se bastante rápida, como mostram os resultados da próxima seção.

4.4.3 Alguns resultados

Utilizando-se a implementação da técnica de Fabris e Forrest, com o mecanismo de Π-coerência, foram gerados vários exemplos para se verificar o comportamento da técnica frente ao novo mecanismo de teste e medidas de tempo. A medida de tempo foi tomada em minutos e segundos, ao invés da contagem do número de operações, com o intuito de se ter uma noção de tempo de execução numa aplicação prática. Os testes foram efetuados num equipamento IBM Pentium 200 MHz. A resolução das imagens é sempre 512×512. Para melhor definição dos resultados, utilizou-se uma resolução de sub-pixel $2^6 \times 2^6$. As imagens geradas foram armazenadas no formato *bitmap*, escaladas, convertidas para PostScript e incluídas através do pacote **graphicx**, disponível para LaTeX. A impressão foi efetuada numa impressora HPLaserJet 5Si a 600 dpi. Como o objetivo primordial era introduzir velocidade na técnica de *antialiasing*, não foi efetuado nenhum tratamento de *gamma correction*[9], fato que pode, eventualmente, refletir negativamente na imagem final. Entretanto, isto não produz nenhuma alteração significante de desempenho na técnica de *stroking* incorporada, uma vez que se trata de uma etapa de pós-processamento.

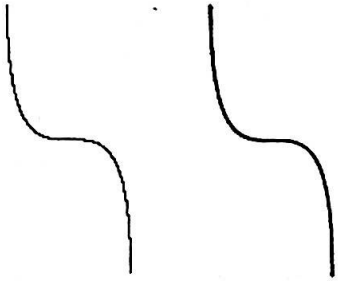
Os primeiros resultados, mostrados nos casos (1) a (6) nas próximas páginas, ressaltam o comportamento da técnica frente a várias particularidades geométricas das curvas. As curvas geradas têm largura 1 e foi utilizado um filtro do tipo Bartlett. As imagens fi-

[9]Tipicamente, deseja-se que a resposta do monitor de vídeo seja linear com um sinal de entrada. Porém, na prática, observa-se um comportamento não-linear entre a intensidade emitida pelo monitor e a voltagem do sinal de entrada. A compensação desta não-linearidade é conhecida como *gamma correction*, podendo ser efetuada diretamente por *hardware* por alguns sistemas ou por tabelas de compensação, via *software*.

nais estão em tamanho natural. A Tabela 11 sumariza os tempos necessários para a computação de cada uma das imagens com *antialiasing*:

Caso	Tempo (segundos)
1	12
2	12
3	19
4	10
5	15
6	16

Tabela 11: Tempos para várias particularidades geométricas de curvas.



(a) Aliasing (b) Antialiasing

Figura 35: Caso (1) - Curva com porções quase verticais e horizontais.

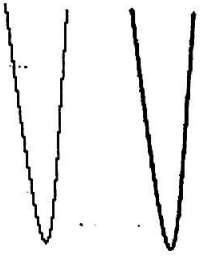


Figura 36: Caso (2) - Curva com pequeno raio de curvatura.

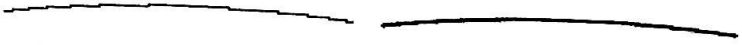


Figura 37: Caso (3) - Curva com grande raio de curvatura.

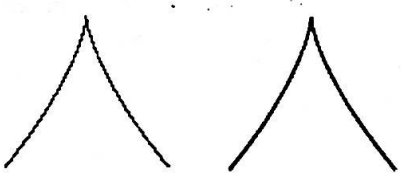


Figura 38: Caso (4) - Curva com um cúspide.

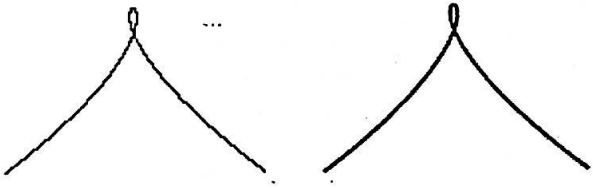


Figura 39: Caso (5) - Curva com um pequeno *loop*.

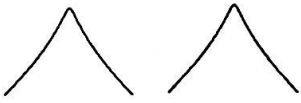


Figura 40: Caso (6) - Curva com dois pontos de inflexão.

Os próximos resultados, mostrados nas páginas seguintes, evidenciam o efeito da variação da largura da curva no desempenho do implementação. Ressalta-se que, quanto maior a largura da curva, maior a região de teste interior/exterior. As imagens estão em tamanho natural e as medidas de largura estão em unidade de *pixel*.

A Tabela 12 compara os tempos necessários para a produção de cada uma destas imagens.

Largura	Tempo (segundos)
0.5	7
1.0	15
1.5	17
2.0	26
2.5	27
3.0	37

Tabela 12: Tempos com variação da largura das curvas.

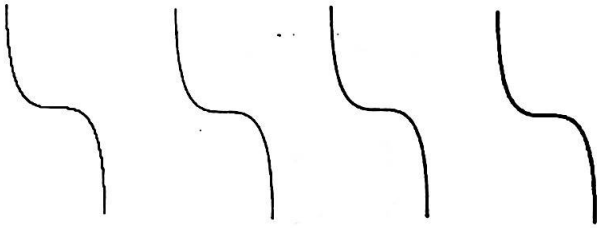
Figura 41: Curva utilizada para teste de variação de largura.

(a) 0.5 (b) 1.0 (c) 1.5 (d) 2.5 (e) 3.0

Figura 42: Variação de largura das curvas.

A variação do filtro, utilizado na técnica, interfere no tamanho da região de teste interior/exterior, evidenciada pela alteração de suporte do filtro. Quanto maior o suporte do filtro, maior será a área de testes e, conseqüentemente, maior o tempo necessário para

antialiasing. Para reforçar esta afirmação, foram gerados testes com tamanho crescente do suporte dos filtros. As imagens correspondentes aos testes encontram-se na próxima página.



(a) Curva Teste (b) Box (c) Bartlett (d) Bell

Figura 43: Efeitos da variação do filtro.

A Tabela 13 mostra o efeito temporal desta variação:

Filtro	Suporte	Tempo (minutos:segundos)
Box	$[-0.5, 0.5]$	11
Bartlett	$[-1.0, 1.0]$	12
Bell	$[-1.5, 1.5]$	20

Tabela 13: Tempos com variação dos filtros.

Finalmente, se a curva for fechada, pode-se preenchê-la e aplicar a técnica de *antialiasing* na bordas, como mostra a seqüência do exemplo a seguir, onde ambos tratamento de *aliasing* e preenchimento foram feitos em alta resolução, utilizando-se um filtro simples (*box filtering*).

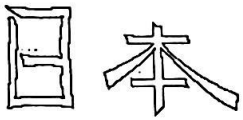


Figura 44: Curvas fechadas.

Figura 45: Curvas fechadas preenchidas e com *antialiasing*.

5 Considerações finais

Algoritmos baseados na imagem desepenham papel fundamental em computação gráfica. O desenvolvimento de formalismos que suportem tais algoritmos é de grande interesse computacional. Em especial, o problema do teste interior/exterior em dimensões mais altas, ou mesmo em dimensões fracionárias, constitui um desafio devido a alta complexidade dos objetos. Mecanismos que tornem a computação mais simples, eficiente e robusta são cada vez mais necessários. Assim, fica evidente que a proximidade entre computação gráfica e áreas como análise, topologia e geometria torna-se bastante estreita.

Referências

[AdoB86] Adobe Systems Incorporated, *PostScript Language Reference Manual*. Addison–Wesley Publishing Company, Reading, 1986.

[BleSM88] Bleser, T.W., Silbert, J.L., McGee, J.P. Charcoal sketching: returning control to the artist. ACM Transactions on Graphics, 7(1), 76–81, Janeiro 1988.

[BurZ85] Burde, G., Zieschang, H. *Knots*. Walter de Gruyter, Berlin, 1985.

[CorP92] Corthout, M.E.A., Pol, E.J.D. *Point Containment and the Pharos Chip.* Tese de Doutorado, Leiden University, the Netherlands, 1992.

[Ede87] Edelsbruner, H. *Algorithms in Combinatorial Geometry.* Springer–Verlag, Heidelberg, 1987.

[Fiu89] Fiume, E.L. *The Mathematical Structure of Raster Graphics.* Academic Press, 1989.

[Fab95] Fabris, A.E. *Robust Antialiasing of Curves.* PhD thesis, University of East Anglia, Norwich, Novembro 1995.

[FabF97] Fabris, A.E., Forrest, A.R. Anti–aliasing of curves by discrete pre–filtering.*Computer Graphics (SIGGRAPH'97 Proceedings)*, **31**, 317–326, Agosto 1997.

[FabF97b] Fabris, A.E., Forrest, A.R. High quality rendering of two–dimensional continuous curves. *SIBGRAPI'97 Conference Proceedings*, IEEE Computer Society Press, 10–17, Outubro 1997.

[FabSF97] Fabris, A.E., Silva, L., Forrest, A.R. An efficient filling algorithm for non–simple closed curves using the Point Containment paradigm. *SIBGRAPI'97 Conference Proceedings*, IEEE Computer Society Press, 2–9, Outubro 1997.

[FabSF98] Fabris, A.E., Silva, L., Forrest, A.R. Stroking Discrete Polynomial Bézier Curves via Point Containment Paradigm. *SIBGRAPI'98 Conference Proceedings*, IEEE Computer Society Press, Outubro 1998.

[Fol96] Foley, J.D. et al. *Computer Graphics: Principles and Practice (second edition in C).* Addison–Wesley Publishing Company, Reading, 1996.

[For85] Forrest, A.R. Computational geometry in practice. *Fundamental Algorithms for Computer Graphics*, Ed. R.A.Earnshaw, Springer–Verlag, 707–724, 1985.

[For88] Forrest, A.R. Geometric computing environments: some tentative thoughts. *Theoretical Foundations of Computer Graphics and CAD*, Springer–Verlag, 185–197, 1988.

[Fra86] Franklin, W.R. Problems with raster graphics algorithms. *Data Structures for Raster Graphics*, Eds. L.R.A. Kessener, F.J. Peters, M.L.P. van Lierop, Springer–Verlag, 1–7, 1986.

[Gla95] Glassner, A.S. *Principles of Digital Image Synthesis (First volume).* Morgan Kaufmann Publishers Inc., San Francisco, California, 1995.

[GhoM84] Ghosh, P.K., Mudur, S.P. The brush-trajectory approach to figure specification: some algebraic solutions. *ACM Transactions on Graphics*, 3(2), 1–24, Abril 1984.

[GuiRS83] Guibas, L.J., Ramshaw, L.H., Stolfi, J. A kinetic framework for computational geometry. *Proceedings of* 24^{th} *IEEE Symposium on the Foundations of Computer Science*, 100–111, 1983.

[GuiP74] Guillemin, V., Pollack, A. *Differential Topology.* Prentice–Hall, Inc., New Jersey, 1974.

[HunS79] Hunter, G.M., Steiglitz, K. Operations on images using quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 145-153, 1979.

[Kau87] Kauffam, L.H. *On Knots.* Princeton University Press, Princeton, 1987.

[KlaS91] Klassen, R.V. Drawing antialiased cubic spline curves. *ACM Transactions on Graphics*, 10(1), 92–108, Janeiro 1991.

[Knu86] Knuth, D.E. *The METAFONT book.* Addison–Wesley, Reading, MA, 1986.

[Mas91] Massey, W.S. *A Basic Course in Algebraic Topology.* Springer–Verlag, New York, 1991.

[New79] Newman, W., Sproull, R. *Principles of Intercative Computer Graphics*, 2^a edição, MacGraw–Hill, New York, 1979.

[Pal91] Palka, B.P. *An Introduction to Complex Function Theory.* Springer–Verlag, New York, 1991.

[Per88] Perky, T.S. PostScript prints anything: a case history. *IEEE Spectrum*, 42–46, Maio 1988.

[PosF89] Posch, K.C., Fellner, W.D. The circle–brush algorithm. *ACM Transactions on Graphics*, 8(1), 1–24, Janeiro 1989.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison–Wesley, 1990.

[Ser82] Serra, J. *Image Analysis and Mathematical Morphology.* Academic Press, 1982.

[Str86] Strassmann, S. Hairy brushes. *Computer Graphics (SIGGRAPH'86 Proceedings)*, 20, 225–232, Agosto 1986.

[Tan88] Tang, G.Y. Region filling with the use ot the discrete Green Theorem. *Computer Vision, Graphics and Image Processing*, 42, 297–305, 1988.

[Whi83] Whitted, T. Anti-aliased line drawing using brush extrusion. *Computer Graphics (SIGGRAPH'83 Proceedings)*,17,151–156, Julho 1983.