UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Renderização de Curvas Implícitas Discretizadas no Domínio da Imagem

Vera Alexandra Henriques Marcelino

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º Ciclo de Estudos)

Orientador: Professor Doutor Abel João Padrão Gomes

Covilhã, Outubro de 2011

Agradecimentos

Gostaria de expressar o meu agradecimento a todas as pessoas que, de diversos modos, contribuíram para a realização deste trabalho.

Ao meu orientador, Professor Doutor Abel Gomes por toda a sua disponibilidade e orientação prestadas, bem como pela partilha de conhecimentos sobre o tema.

Ao Gonçalo Amador pelo apoio e ideias tão úteis que me facultou ao longo deste trabalho.

À Catarina, à Etelvina, ao Gonçalo Dias e ao Rui Afonso pelos conselhos e momentos de lazer e descontração proporcionados, não só durante a realização deste trabalho mas também durante todo o percurso académico.

Aos meus amigos Alexandra, Ana Sofia, Edmundo, Filomena e Pedro, pela força, incentivo e ânimo que me transmitiram, assim como, pela amizade que desde sempre demonstraram.

Aos meus pais e restante família, principalmente à minha irmã por toda a sua paciência, ajuda e preocupação para comigo.

Resumo

A representação gráfica de curvas implícitas continua a ser um tópico de investigação importante em computação gráfica e geometria computacional, e tem aplicações em várias áreas de interesse como sejam, por exemplo, representação de símbolos em tipografia digital, delimitação de contornos em imagem médica gerada por tomografia axial, bem como na definição de trajetórias para a simulação de movimento de personagens em animação computacional e jogos de vídeo.

De forma sumária, pode dizer-se que esta dissertação propõe um algoritmo de pixelização de curvas implícitas que, ao que parece, não tem paralelo na literatura, a não ser nos algoritmos de rasterização de linhas retas e circunferências que incorporavam os sistemas gráficos primitivos, como por exemplo o algoritmo de Bresenham. De alguma maneira, o referido algoritmo de pixelização pode ser visto como uma generalização daqueles algoritmos primitivos no sentido de que se aplica a qualquer curva implícita, mesmo que ela apresente singularidades, pontos isolados, e outros pontos críticos.

Palavras-chave

Curva implícita, ponto crítico, ponto isolado, singularidade, pixelização.

Abstract

The graphical representation of implicit curves remains a major research topic in computer graphics and computational geometry, and has applications in several areas of interest such as, for example, representation of symbols in digital typography, delineation of contours in medical images generated by computerized axial tomography, as well as the definition of trajectories for the simulation of movement of characters in computer animation and video games.

Briefly speaking, it can be said that this dissertation proposes a pixelization algorithm for implicit curves that apparently has no parallel in literature, except in the rasterization algorithms of straight lines and circles incorporated in primitive graphics systems, such as Bresenham's algorithm. Somehow, this algorithm pixelization can be seen as a generalization of those primitive algorithms in that it applies to any curve implied, even if it presents singularities, isolated points, and other critical points.

Keywords

Implicit curve, critical point, singularity, isolated point, pixelization.

Conteúdo

Lista de Figuras

Lista de Símbolos e Siglas

f – Função de IR^n em IR

D – Domínio de *f*

Δ - Domínio da imagem

BSP - *Binary Space Partitioning*

EC – Estimação-Correção

TAC – Tomografia Axial Computorizada

Capítulo 1

Introdução

Este capítulo pretende enquadrar o tema da dissertação, as motivações que levaram à sua escolha, assim como os desafios colocados pela renderização de curvas implícitas em $\mathbb{R}^2$, as possíveis soluções, e ainda a organização da própria dissertação. No essencial, esta dissertação descreve um algoritmo de pixelização que, ao que parece, não tem paralelo na literatura, a não ser os algoritmos de rasterização de linhas retas e circunferências que incorporavam os sistemas gráficos primitivos, como por exemplo o algoritmo de Bresenham. De alguma forma, o algoritmo aqui proposto pode ser visto como uma generalização destes algoritmos primitivos no sentido de que se aplica a qualquer curva implícita, mesmo que ela apresente singularidades, pontos isolados, e outros pontos críticos.

1.1 Motivação

As curvas implícitas têm grande interesse e aplicabilidade, nomeadamente na reconstrução tridimensional de órgãos do corpo humano, a partir de imagens médicas como radiografias, imagens obtidas a partir de Tomografia Axial Computorizada (TAC), ressonância magnética ou Raio-X [3]. Os contornos dos órgãos humanos numa imagem médica são importantes porque permitem ao clínico identificar facilmente os referidos órgãos por inspeção visual. Tais contornos podem ser descritos por curvas implícitas. Isto significa que é possível reconstruir um dado órgão em 3D a partir da sequência dos seus contornos numa pilha de imagens TAC.

Outra possível aplicação de curvas implícitas é na representação de letras de vários tipos de fontes. Ou seja, é possível usar curvas implícitas para desenhar as letras de um alfabeto. Por exemplo, a fronteira do símbolo ♥ de um coração é dada por $\left(x^2+y^2-1\right)^3-x^2y^3=0$ (veja-se [13]).

As curvas implícitas também são utilizadas como percursos ou caminhos em cenas animadas. Normalmente este tipo de animações é necessário em jogos de computador, em particular na simulação de movimento de personagens realistas como, por exemplo, humanos. Em [27], Xu, Dan e Tang propõem um algoritmo de descrição do movimento de animações ao longo de curvas implícitas. Outra das suas aplicações é em controlo de robôs, ou seja, dada uma função implícita, o robô segue a trajetória definida pela curva. O percurso do robô também pode ser definido pela interseção de curvas de nível de várias funções [11].

1.2 Formulação do Problema

Como referido acima, o desafio de investigação colocado no início dos trabalhos conducentes a esta dissertação foi o de construir um algoritmo de pixelização de curvas implícitas que oferecesse garantias topológicas no domínio de imagem. Recorde-se que o problema das garantias topológicas é normalmente abordado no domínio da função que descreve a curva implícita, e não tanto no domínio da imagem.

Em termos mais gerais, este problema pode ser formulado da seguinte maneira. Dada uma superfície implícita definida por uma função real $f(p)$, em que $p \in \mathbb{R}^3$, e um plano de corte (em, por exemplo, $z=0$) que a atravessa, pode afirmar-se de forma genérica que a interseção da superfície com o plano é um contorno que pode ser representado por uma curva implícita. Portanto, um ponto da curva também pertence à superfície, ou seja, $f(p)=0$.

1.3 Organização da Dissertação

Esta dissertação está estruturada da seguinte forma:

- O Capítulo 1, que é o presente capítulo, faz o enquadramento do tema, incluindo a motivação científica que levou à escolha do mesmo, bem como a formulação do problema da pixelização de curvas implícitas.

- O Capítulo 2 apresenta os vários tipos de métodos existentes na literatura para representar curvas implícitas. Este capítulo consubstancia o levantamento sumário da pesquisa bibliográfica efetuada por forma a melhor contextualizar o leitor no assunto abordado.

- No Capítulo 3 são explicados todos os detalhes do algoritmo implementado, incluindo a resolução do problema de descontinuidade da curva no domínio da imagem.

- O Capítulo 4 descreve o percurso de investigação que conduziu ao desenvolvimento do algoritmo, as contribuições do trabalho de investigação, bem como as conclusões mais relevantes desta dissertação.

Capítulo 2

Revisão da Literatura

Neste capítulo pretende-se apresentar os principais métodos, bem como os respetivos algoritmos, utilizados na renderização de curvas implícitas em $\mathbb{R}^2$, para assim ter melhor enquadramento o algoritmo proposto no Capítulo 3, e que no fundo é a razão de ser desta dissertação.

2.1 Introdução

Em termos matemáticos, existem duas formas de definir uma curva: a forma paramétrica e a forma implícita. Recorde-se que a forma explícita é um caso particular da forma implícita. A forma paramétrica é essencialmente generativa no sentido de que cada ponto (x,y) da curva é obtido a partir do valor real dum parâmetro t pertencente a um dado intervalo I na linha real, ou seja, uma curva paramétrica é definida por uma função $f:I \subset \mathbb{R} \rightarrow \mathbb{R}^2$, tal que a um dado t corresponde um ponto $(x(t),y(t))$, em que $x(t)$ e $y(t)$ são as funções componentes de f.

Por outro lado, uma curva implícita é definida por uma função $f:I \subset \mathbb{R}^2 \rightarrow \mathbb{R}$, ou seja, é o conjunto de todos os pontos de $\mathbb{R}^2$ em que a função tem um mesmo valor, às vezes chamado o iso-valor. O iso-valor é normalmente zero, mas poderá ser qualquer outro valor real. Dito de outra maneira, uma curva implícita é o resultado da interseção do plano $z=0$ com uma superfície $z=f(x,y)$ definida em $\mathbb{R}^3$.

Na literatura [9] existem duas categorias principais de métodos que permitem renderizar curvas implícitas: os métodos numéricos e os métodos simbólicos. Os métodos simbólicos são métodos que utilizam processos de computação simbólica (por exemplo, fatorização simbólica) e até mesmo de conversão simbólica para calcular as componentes (representadas por sub-expressões) da curva [22], os pontos críticos (p.ex., auto-interseções, máximos, mínimos e pontos de viragem) da curva [12], bem como uma hipotética representação paramétrica da curva [2]. Como é sabido, em grande número de casos, e por manipulação simbólica, não é possível encontrar uma representação paramétrica global para uma curva que é definida implicitamente [10, pp. 23]. Os métodos simbólicos são exatos, mas têm ainda a desvantagem de serem muito menos eficientes que os métodos numéricos no que respeita a desempenho e velocidade de processamento.

Os métodos numéricos de renderização de curvas implícitas recorrem a métodos de análise numérica (p.ex., método de Newton) para calcular um conjunto discreto de pontos da curva. No entanto, a amostragem da curva deve ser feita de forma cuidadosa para assim garantir que a curva discretizada é topologicamente correta. Em particular, há que garantir que as auto-interseções e todas as componentes da curva seguiram um procedimento correto de amostragem. Na categoria dos métodos numéricos existem duas principais sub-categorias:

- *Métodos de continuação* (ou *métodos baseados no seguimento da curva*);
- *Métodos de partição do espaço*.

Pode desde já dizer-se que o algoritmo proposto nesta dissertação não pertence a nenhuma destas categorias. Ao invés, é um algoritmo que opera essencialmente no domínio da imagem, o qual pode ser visto como uma discretização ou, se se quiser, uma partição do domínio da função real que descreve a curva implícita.

2.2 Métodos de Continuação

A essência dos métodos de continuação reside no seguinte ideia: cada ponto da curva é calculado a partir do ponto anterior. Há duas categorias principais de métodos de continuação [10]:

- Métodos de estimação-correção;
- Métodos lineares segmentados.

Nos métodos de estimação-correção utiliza-se dois dispositivos matemáticos: um estimador para estimar o próximo ponto da curva a partir do ponto atual já calculado e, de seguida, um corretor que corrige ou faz convergir o ponto estimado em direção à curva. O próximo ponto da curva será obviamente o ponto corrigido.

Nos métodos lineares segmentados, utiliza-se uma técnica de pivotagem de triângulos sobre a curva, ou seja, um triângulo que interseta a curva é gerado a partir do triângulo anterior segundo uma regra de pivotagem. Noutras palavras, uma sequência conexa de triângulos é gerada ao longo da curva, sendo dois pontos consecutivos da curva resultantes da interseção entre um triângulo e a curva; o primeiro ponto é a porta de entrada no triângulo e o segundo ponto é a porta de saída do triângulo.

2.2.1 Algoritmos de Estimação-Correção (EC)

Como já foi referido acima, esta categoria de algoritmos de continuação segue uma abordagem em duas etapas. Na primeira etapa, estima-se um novo ponto $\boldsymbol{x}_i$ da curva sobre a reta tangente (definida pelo vetor tangente t_i) à curva no ponto atual p_i, corrigindo-se então o ponto estimado em direção à curva através de um processo de convergência que

normalmente é um método numérico como, por exemplo, o método de Newton-Raphson (Figura 2.1). Deste processo de convergência que se inicia em x_i resultará o novo ponto p_{i+1}. No final, obtém-se uma sequência de pontos $p_0,p_1,...,p_N$ da curva, que são no fim de contas as amostras que servirão para linearizar a curva.

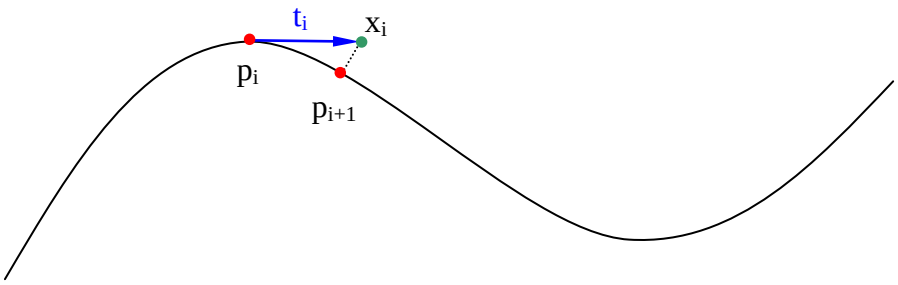


Figura 2.1: Ilustração do cálculo do ponto seguinte p_{i+1} a partir do ponto atual p_i.

Este método é rápido e eficiente, pois permite traçar curvas em poucas iterações. Contudo, padece de vários problemas ou desvantagens, nomeadamente:

1. ***Ciclicidade***. No caso de curvas fechadas, o algoritmo entra facilmente em ciclo, a não ser que haja um controlo sobre o retorno ao ponto inicial.
2. ***Desvio de curso***. Quando a curva passa perto de si própria, independentemente de ser no mesmo ramo ou não, poderá acontecer o novo ponto p_{i+1} não pertença ao mesmo ramo da curva de p_i, ou que simplesmente a passagem de p_i para p_{i+1} provoque um atalho na amostragem do mesmo ramo da curva. Em qualquer dos casos, diz-se que existe um desvio do curso ou da trajetória na amostragem da curva. Estes problemas podem ser em alguns casos colmatados se for reduzido o tamanho do passo x_i-p_i usado em cada iteração do método.
3. ***Componentes múltiplas***. Se a curva tiver mais do que uma componente, torna-se necessário ter alguma forma de detetar e determinar um ponto em cada componente de maneira a não deixar de fora alguma componente por amostrar e renderizar. Acontece que ainda está por encontrar um algoritmo completamente fiável que consiga detetar todas as componentes de uma curva implícita em $\mathbb{R}^2$.
4. ***Pontos críticos***. Em algoritmos de continuação, o atravessamento de pontos críticos da curva, ou seja, pontos de viragem, ápices e auto-interseções, é um problema de difícil resolução porque o sentido do processo de continuação pode inverter-se, ou simplesmente enveredar por um caminho ou ramo da curva já percorrido anteriormente. Além disso, se se utilizar um método numérico que utilize derivadas, como é o caso do corretor de Newton-Raphson, então o processo de amostragem crachará nos pontos críticos (ou seja, pontos de viragem, ápices e auto-interseções) da curva, pois aí pelo menos uma das derivadas parciais da função será nula.

Os pontos de viragem em que uma das derivadas deixa de existir e as singularidades (ápices e auto-interseções) nos quais ambas as derivadas desaparecem são pontos de difícil resolução quando a amostragem é realizada com métodos numéricos que fazem uso de derivadas, como é o caso do método de Newton-Raphson. Há duas maneiras possíveis de resolver este problema:

- ***Métodos numéricos sem derivação***. Se o problema reside na sua essência na utilização de derivadas, então utiliza-se um método que não use derivadas, como é o caso do método da falsa posição. No entanto, estes métodos numéricos alternativos baseiam-se na maior parte dos casos no Teorema do Valor Intermédio, ou seja, necessitam que haja variação de sinal da função em dois pontos diferentes para que se possa detetar um ponto intermédio em que a função se anula. Ora acontece que existem componentes da curva (em que a função se anula) em que de ambos os lados da componente a função tem o mesmo sinal para todos os pontos vizinhos de um lado e do outro da componente [18].
- **Resolução simbólica de singularidades**. Já que as derivadas parciais se anulam nas singularidades, pode usar-se o Teorema da Função Implícita para calculá-las através da resolução simbólica de um sistema de duas equações, ou seja, as equações resultantes do anulamento das derivadas parciais. Depois de identificadas, estas singularidades funcionam como pontos de partida para a amostragem de segmentos da curva através do método de estimação-correção.

2.2.2 Algoritmos Lineares Segmentados

Com este tipo de métodos, também conhecido por métodos de continuação linear segmentada, faz-se uma triangulação do domínio com o objetivo de efetuar a amostragem de uma dada curva implícita. A triangulação é normalmente feita com triângulos equiláteros ou então com triângulos isósceles. Repare-se que a triangulação é efetuada incrementalmente triângulo-a-triângulo, mas só com aqueles triângulos que intersetam a curva no domínio da função. Na prática, o conjunto dos triângulos que intersetam a curva acaba por representar uma aproximação à própria curva.

A geração incremental da triangulação segue o princípio de continuação aplicado a triângulos, ou seja, o próximo triângulo é gerado a partir do triângulo atual, o que resultará na determinação do próximo ponto a partir do ponto atual, que é o ponto de saída do triângulo atual. Este processo de geração de triângulos consecutivos é conhecido por pivotagem, e as triangulações daí resultantes são designadas por triangulações de Coxeter [10].

A técnica de pivotagem utilizada nas triangulações de Coxeter é essencialmente um método de geração de triângulos por reflexão ao torno de uma aresta [6] [7]. A aresta de reflexão do triângulo atual é a aresta que contém o ponto de saída da curva. O próximo triângulo é criado

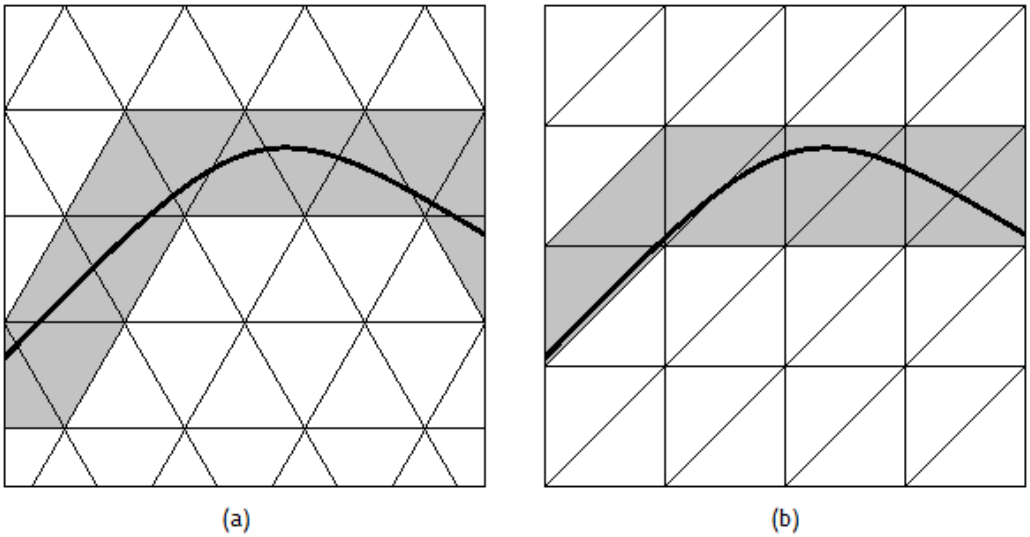


Figura 2.2: Triangulações obtidas por pivotagem com (a) triângulos equiláteros e (b) triângulos isósceles.

por reflexão do triângulo atual, que lhe é adjacente, através da aresta comum a ambos (Figura 2.2).

Como facilmente se observa pela Figura 2.2, a partir de regras de pivotagem diferentes obtêm-se triangulações diferentes, mas a mesma regra aplicada a diferentes tipos de triângulos também gera diferentes triangulações.

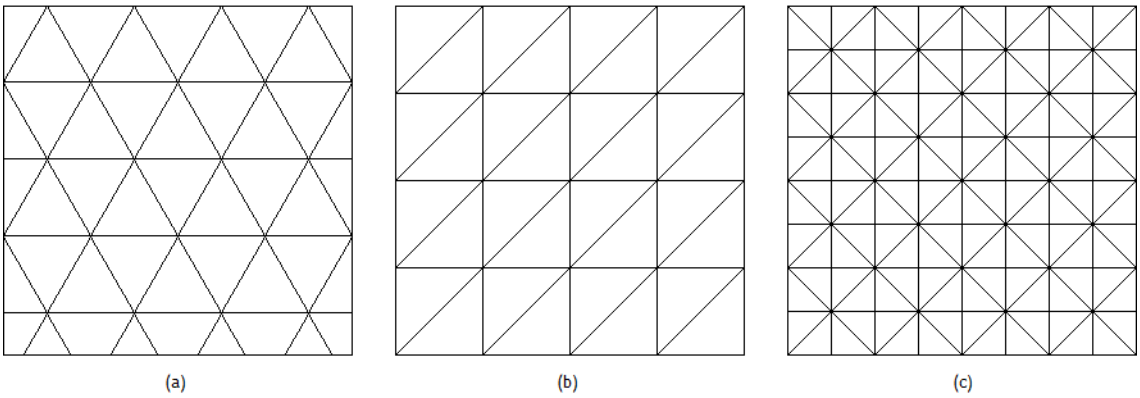


Figura 2.3: Três tipos de triangulação de Coxeter: (a) triangulação com triângulos equiláteros; (b) triangulação de Freudental; (c) triangulação de Todd.

Na Figura 2.3 apresentam-se os três tipos de triangulações mais comuns geradas por pivotagem. Por exemplo, a triangulação de Freudenthal (Figura 2.3(b)) utiliza triângulos isósceles e é semelhante à que utiliza triângulos equiláteros, pois também é assente em reflexões, mas neste caso não é propriamente um triângulo que é refletido, mas sim um vértice de um triângulo através do ponto médio da sua aresta oposta, o que originará um novo triângulo. Repare-se que à volta de cada vértice existem também seis triângulos, mas os seis ângulos que partilham o mesmo vértice têm diferentes valores. Para mais detalhes sobre

estas triangulações geradas por pivotagem veja-se os livros de Gomes et al. [10] e Allgower an Georg [1].

2.3 Partição espacial

A partição do espaço, seja do domínio completo da curva ou de apenas algumas regiões contidas nesse domínio, é outro tipo de métodos que permitem renderizar curvas implícitas. Esta partição pode ser feita uniformemente ou de forma não uniforme, isto é, em partes iguais ou alternativamente, em partes distintas. O tipo de partição uniforme, abordado a seguir, é o mais amplamente usado na literatura, dividindo-se em partição recursiva e não recursiva.

2.3.1 Partição Uniforme Recursiva

Relativamente à divisão recursiva, também denominada por subdivisão, pode dizer-se que consiste em particionar apenas as zonas do domínio que são atravessadas pela curva. O critério de paragem da subdivisão ocorre quando, por exemplo, uma das seguintes condições se verifica:

1. Quando a zona não é intersetada pela curva.
2. Quando a zona atinge o tamanho mínimo que é, por exemplo, o tamanho dum pixel.
3. Quando a curva dentro dessa zona é aproximadamente um segmento de reta.

Repare-se que a curvatura da curva influencia a partição do domínio. Quanto maior for a curvatura, mais fina é a subdivisão do domínio junto à curva, como se pode observar pela Figura 2.4.

Existem duas formas de se fazer a partição recursiva: *bintree* e *quadtree*. A primeira consiste em subdividir o domínio em duas partes iguais e na segunda, como o próprio nome indica, o domínio é subdividido em quatro partes idênticas (Figura 2.4). A subdivisão *quadtree* tem vantagem sobre a partição uniforme não-recursiva (abordada a seguir), uma vez que o tamanho da partição se adapta à curvatura resolvendo algumas ambiguidades topológicas da curva que poderiam não ser detetadas através da partição uniforme.

Note-se, no entanto, que alguns pontos isolados ou outras pequenas componentes de uma curva podem não ser detetados recorrendo à partição quadtree, mesmo que a partição seja feita com uma resolução máxima. Uma das formas de resolver esta desvantagem das quatrees é o uso de aritmética afim ou aritmética intervalar [23] [24], funcionando esta última como detetor da curva no interior de cada partição. Se a função toma valor valor 0 sobre os lados do quadrado que forma a partição, então esta é intersetada pela curva. Para calcular estes pontos de interseção da curva com as arestas da partição são usados métodos numéricos como

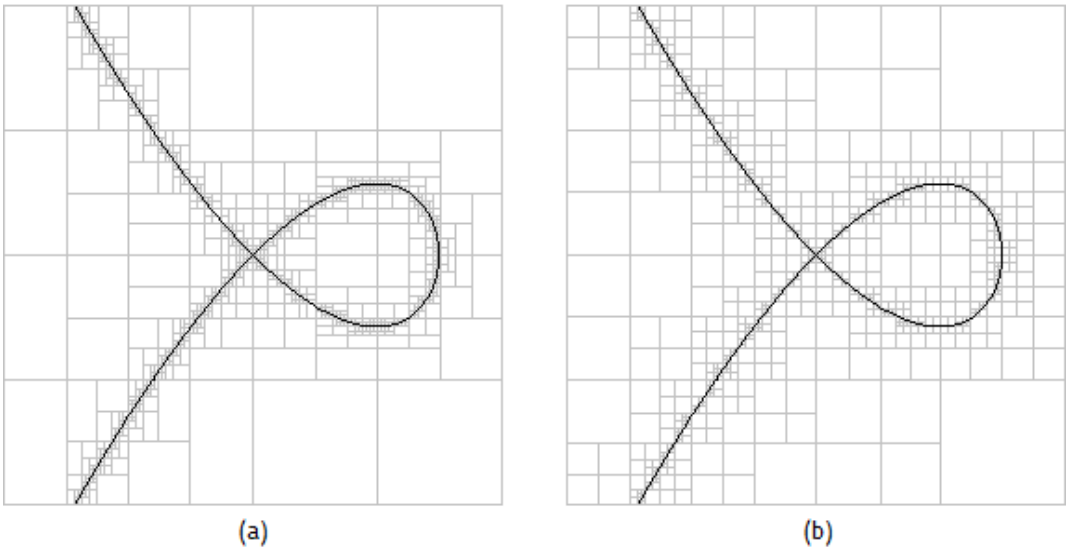


Figura 2.4: Dois tipos de partição do domínio: (a) partição *bintree*; (b) partição *quadtree*.

o método da bisseção, o método da falsa posição ou o método de Newton, mas também podem ser usados métodos simbólicos, como o método de Bézier [10].

2.3.2 Partição Uniforme Não-Recursiva

No que diz respeito à partição não recursiva, esta baseia-se na partição completa do domínio em regiões regulares. Essa partição pode ser feita através de quadrados ou triângulos, podendo estes últimos ser equiláteros ou isósceles, mantendo sempre o mesmo tamanho, em todo o domínio (Figura 2.5).

O algoritmo dos quadrados marchantes (*Marching Squares*) é um exemplo de algoritmo que permite representar curvas implícitas recorrendo à partição do domínio de forma uniforme, isto é, em quadrados alinhados com os eixos (Figura 2.6). O processamento de cada quadrado é feito individualmente e consiste no cálculo da função $f(x,y)$ nos seus quatro vértices, sendo estes valores guardados numa estrutura de dados, isto é, numa matiz em que cada elemento armazena os dados relativos a um quadrado. Os pontos de interseção da curva com as arestas do quadrado também são calculados através de métodos de cálculo de zeros [10].

Os passos mais importantes do algoritmo dos quadrados marchantes são:

1. a determinação da configuração topológica da curva dentro do quadrado, ou seja, a forma exata da curva;

2. o cálculo dos pontos resultantes da interseção curva-aresta.

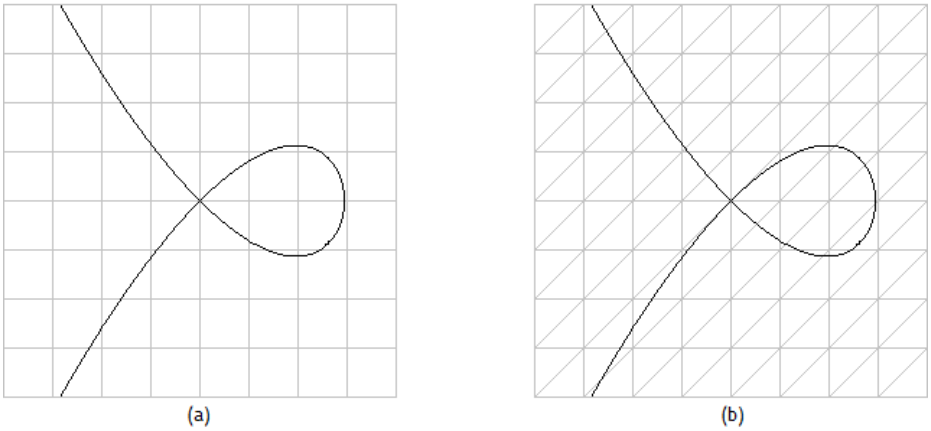


Figura 2.5: Partições uniformes não-recursivas do domínio da curva.

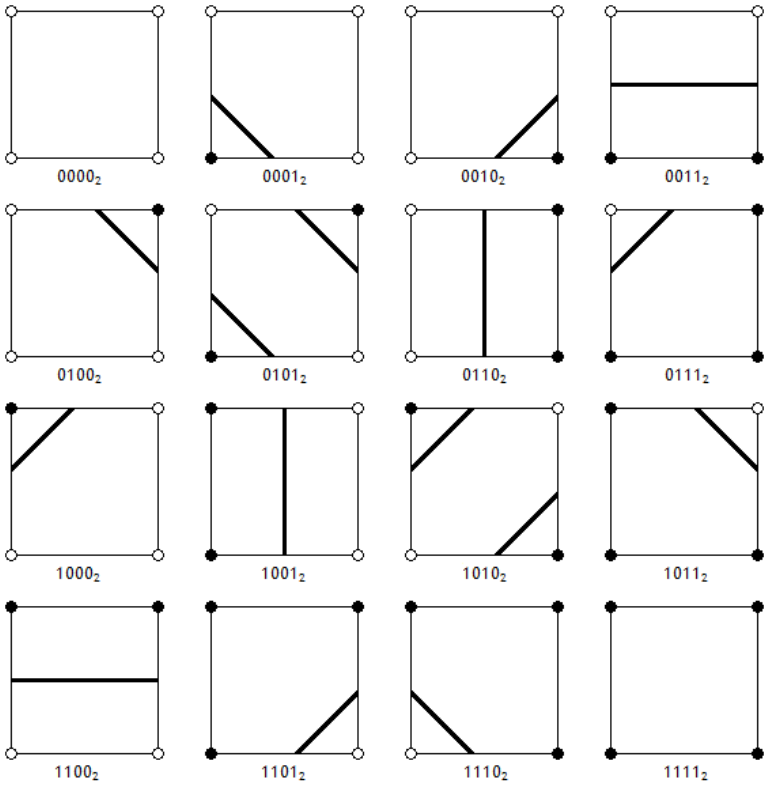


Figura 2.6: As 16 configurações topológicas da curva dentro de um quadrado marchante.

O cálculo da configuração topológica é feito recorrendo a um código de 4 bits, em que cada bit codifica o estado de um vértice, tomando valor 0 se a função é negativa nesse vértice e valor 1 se a função é positiva. Como se pode observar na Figura 2.6 há um código específico para cada configuração que serve de índice numa tabela de pesquisa onde são guardadas todas as configurações. Os dados armazenados nesta tabela são usados para representar a curva dentro de cada quadrado, com garantias topológicas razoáveis.

O algoritmo de quadrados marchantes abrange 16 configurações porque em cada um dos quatro vértices a função pode ser positiva ou negativa, logo existem 4^2 possibilidades de a

curva atravessar o quadrado. A codificação binária dos vértices é feita no sentido anti-horário, a partir do vértice inferior esquerdo.

Note-se que as configurações apresentadas não contemplam auto-interseções da curva dentro do quadrado, interseções da curva com os vértices, ou múltiplas interseções da curva com a mesma aresta. Para representar curvas com estas caraterísticas, através do algoritmo de quadrados marchantes, é feita a subdivisão recursiva dos quadrados até que cada um dos sub-quadrados coincida com alguma das configurações da Figura 2.6. Neste caso o tipo de partição deixa de ser uniforme não recursiva, passando a ser aplicado o algoritmo *quadtree* abordado anteriormente.

2.3.3 Partição Não-Uniforme

Tendo em consideração a revisão da literatura efetuada, não foi possível identificar qualquer método de partição não-uniforme que não fosse recursivo. Ou seja, os métodos de partição não-uniforme abordados na literatura são recursivos (subdivisão). Caraterizam-se essencialmente pelo tipo de partição (irregular) que utilizam e são aplicados tanto a curvas algébricas como a curvas definidas por funções reais mais gerais. Um exemplo deste tipo de métodos é apresentado em [9], denominado por *Binary Space Partitioning* (BSP) ou partição binária do espaço.

Ao contrário das bintrees vistas anteriormente, o algoritmo BSP descrito em [9] utiliza uma partição do espaço não alinhada com os eixos e dependente da curvatura da curva dentro de uma dada zona, isto é, faz mais subdivisões onde a curvatura da curva é maior. Após cada partição são encontrados os pontos de interseção da curva com as arestas da partição; é com base nesses pontos que se faz a representação final da curva. Este algoritmo é uma generalização do algoritmo *bintree* (árvore binária), abordado anteriormente, cuja principal diferença entre ambos reside no alinhamento da fronteira das partições com os eixos.

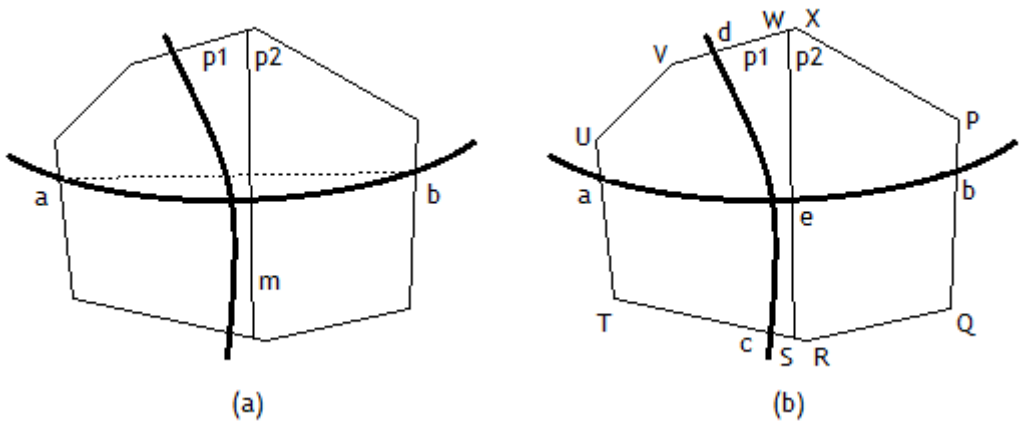


Figura 2.7: Exemplo de uma partição não-uniforme.

A Figura 2.7 ajuda a perceber como são feitas as sucessivas partições recorrendo ao algoritmo BSP. Dada uma zona inicial, é calculado o conjunto de pontos de interseção da curva com a fronteira da zona (Figura 2.7(a)). $\overline{ab}$ é o segmento que une os dois pontos da curva mais distantes entre si, pertencentes à zona, e *m* é a sua mediatriz que subdivide a zona em duas sub-zonas *p1* e *p2*. O segmento $\overline{WS}$ da linha de bisseção m que interseta a zona inicial passa a ser um segmento de fronteira pertencente tanto a *p1* como a *p2*. É importante que tanto os segmentos de fronteira da zona inicial como os pontos da curva pertencentes a essa mesma fronteira sejam redistribuídos pelas duas novas zonas *p1* e *p2*. Como se pode observar na Figura 2.7(b), a zona inicial é delimitada pelos segmentos $\overline{XP}$, $\overline{PQ}$, $\overline{QR}$, $\overline{RT}$, $\overline{TU}$, $\overline{UV}$ e $\overline{VX}$; os seus pontos de interseção com a curva são *a*, *b*, *c* e *d*. Após a divisão da zona inicial em duas sub-zonas, a zona *p1* passará a ser delimitada pelos segmentos $\overline{WS}$, $\overline{ST}$, $\overline{TU}$, $\overline{UV}$ e $\overline{VW}$, ao passo que a zona *p2* será delimitada por $\overline{WS}$, $\overline{SR}$, $\overline{RQ}$, $\overline{QP}$, $\overline{PX}$ e $\overline{XW}$. Note-se que os pontos da curva, *a*, *c* e *d* passarão a pertencer a *p1*, enquanto *b* pertencerá a *p2*. O novo ponto *e* que resulta da interseção da mediatriz com a curva pertencerá tanto a *p1* como a *p2*. Após serem encontrados os pontos de interseção da mediatriz com a curva, cada uma das sub-zonas é também recursivamente particionada, aplicando o mesmo algoritmo.

2.4 Notas finais

Como se verá no próximo capítulo, o algoritmo desenvolvido nesta dissertação não se enquadra em nenhuma das categorias de algoritmos descritos nas seções anteriores do presente capítulo. Podem apenas evidenciar-se algumas caraterísticas em comum com o algoritmo de Bresenham para segmentos de reta no domínio da imagem [4].

Recorde-se que o algoritmo de Bresenham para segmentos de reta é um dos primeiros algoritmos desenvolvidos em computação gráfica e que consiste em colorir os pixéis que são atravessados por um segmento de reta definido entre dois pixéis que correspondem aos pontos terminais do segmento de reta. O cálculo dos pixéis a colorir é feito recorrendo a operações matemáticas elementares de incrementação e decrementação de coordenadas discretas, as quais dependem do declive do segmento de reta.

O algoritmo descrito no próximo capítulo é bem mais próximo do algoritmo de rasterização de Chandler para curvas implícitas mais genéricas [5]. Note-se, no entanto, que o algoritmo de Chandler é intrinsecamente um algoritmo de continuação de pixéis, ao passo que o algoritmo proposto no próximo capítulo não o é. Além disso, o algoritmo descrito no próximo capítulo é capaz de detetar e representar corretamente singularidades, incluindo pontos isolados, bem como componentes ou partes da curva na vizinhança das quais não existe variação de sinal da função.

Capítulo 3

Algoritmo de Pixelização de Curvas Implícitas

O algoritmo que se descreve neste capítulo é um algoritmo híbrido que combina informação do domínio $D \subset \mathbb{R}^2$ e do correspondente domínio Δ da imagem no ecrã, por forma a efetuar a discretização da curva.

3.1 Descrição breve do algoritmo

Na sua essência, o algoritmo proposto nesta dissertação baseia-se, em primeira instância, na criação de um domínio Δ de pixéis associado ao domínio $D \subset \mathbb{R}^2$. Este domínio objetiva fazer a amostragem da curva numa base de pixel-a-pixel. Este processo de amostragem baseia-se no Corolário de Bolzano para identificar se a curva passa entre dois pixéis consecutivos, quer vertical quer horizontalmente. Esta abordagem permite representar a curva de forma correta, incluindo pontos críticos (por exemplo, singularidades e extremos).

O algoritmo inicia-se com a construção de uma máscara de pixéis (Figura 3.1) com tamanho igual ao da janela de visualização no ecrã (Algoritmo 1). Depois verifica-se se cada um dos pixéis da máscara é ou não atravessado pela curva e em caso afirmativo é colorido. A seguir são explicados os detalhes do algoritmo.

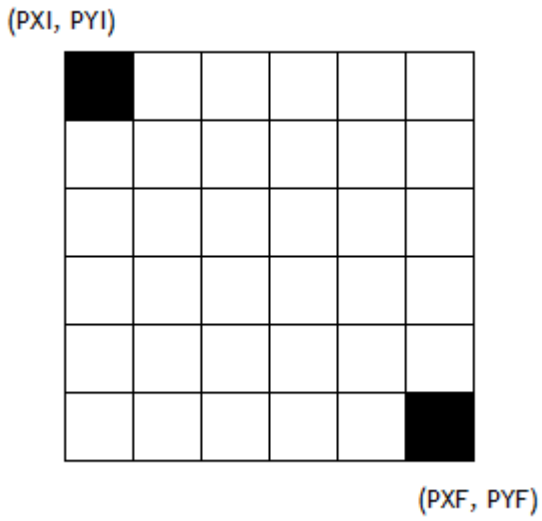


Figura 3.1: Exemplo de uma máscara de pixéis.

Algoritmo 1 Algoritmo de pixelização de curvas implícitas

Input:
PXI: pixel *X* mínimo da janela de visualização
PXF: pixel *X* máximo da janela de visualização
PYI: pixel *Y* mínimo da janela de visualização
PYF: pixel Y máximo da janela de visualização
C: curva
Output: máscara com os pixéis da curva coloridos

```
begin
    Construir máscara de pixéis de tamanho [PXF-PXI]×[PYF-PYI]
    for py ← PYI to PYF do
        for px ← PXI to PXF do
            if pixel[px][py] ∈ C then
                Colorir pixel[px][py] da curva
            endif
        endfor
    endfor
end
```

Como se pode depreender do Algoritmo 1, a máscara é representada por uma matriz em que cada posição representa um pixel. O seu tamanho (número de linhas e colunas) é dado pela resolução da janela de visualização considerada.

Um problema que surgiu no decorrer do desenho e implementação do algoritmo, e que vai ser detalhado posteriormente, foi o da resolução de pontos críticos, ou seja, singularidades e pontos extremos. Para se proceder à sua resolução recorreu-se à informação de vizinhança guardada na estrutura de dados de cada pixel. Esta informação fica guardada em duas variáveis: a variável *p* que é uma *flag* que indica se o pixel pertence ou não à curva e a variável *nv* que guarda o número de pixéis vizinhos que também pertencem à curva. Consideram-se como vizinhos de um pixel os seus oito pixéis adjacentes.

```
typedef struct pixel{
      int p;
      int nv;
};
```

3.2 Mapeamento entre pixéis e pontos

O algoritmo de pixelização de curvas implícitas requer uma forma de mapear pontos do domínio da função em pixéis no domínio da imagem, e vice-versa (Figura 3.2). O número de pixéis (comprimento e largura) do domínio da imagem é dado pelo número de pixéis da janela de visualização. Cada pixel é identificado por um par de coordenadas, tendo o pixel do canto superior esquerdo na máscara as coordenadas (0,0), em que a primeira coordenada indica a coluna e a segunda faz referência à linha da máscara em que o pixel se localiza. O incremento da primeira coordenada é feito da esquerda para a direita (pixéis em diferentes colunas) enquanto o da segunda é feito de cima para baixo (pixéis em diferentes linhas). Na Figura 3.2, as coordenadas (px_i, py_i) referem-se ao pixel inicial, enquanto (px_f, py_f) indica a localização do pixel final da máscara. Como já foi referido anteriormente, cada um dos pixéis é inicializado com duas informações guardadas nas variáveis *p* e *nv* da estrutura de dados *pixel*.

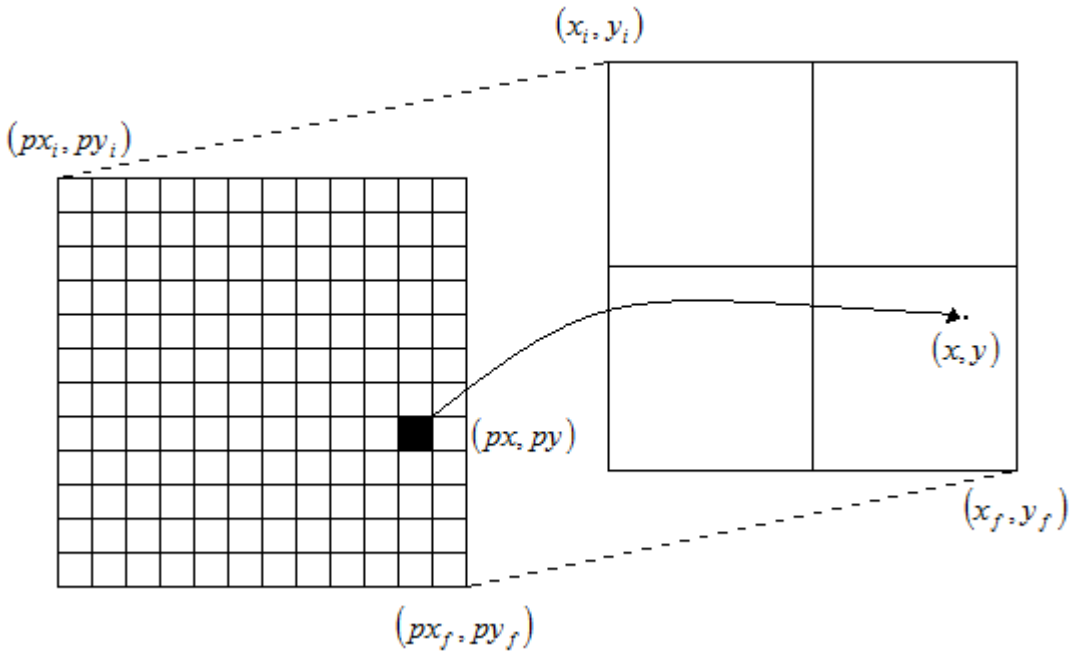


Figura 3.2: Correspondência entre pixéis e pontos.

A correspondência entre pixéis da imagem e pontos do domínio da função faz-se à custa das seguintes equações:

$$\frac{px - px_i}{px_f - px_i} = \frac{x - x_i}{x_f - x_i} \tag{1}$$

$$\frac{py - py_i}{py_f - py_i} = \frac{y - y_i}{y_f - y_i} \tag{2}$$

O cálculo de qualquer ponto (x, y) do domínio da função correspondente a um pixel do domínio da imagem é feito através do isolamento das variáveis x e y das equações (3) e (4):

$$x = \frac{px - px_i}{px_f - px_i}(x_f - x_i) + x_i \tag{3}$$

$$y = \frac{py - py_i}{py_f - py_i}(y_f - y_i) + y_i \tag{4}$$

Nas equações anteriores, as variáveis px e py indicam a coordenada do pixel correspondente ao ponto (x, y) do domínio, px_i e py_i são as coordenadas do pixel superior esquerdo da janela de visualização, px_f e py_f constituem as coordenadas do pixel inferior direito da janela de visualização (Figura 3.1). Por exemplo, para calcular as coordenadas de um ponto correspondente a um pixel de uma janela de tamanho 500×500 pixéis, as coordenadas px_i e py_i teriam ambas o valor 0 (posição inicial da janela) e px_f e py_f teriam ambas valor 500 (posição final da janela). Nas mesmas equações, x_i e y_i são as coordenadas do ponto do domínio correspondente ao pixel de coordenadas (px_i, py_i) e x_f e y_f são as coordenadas do ponto do correspondente ao pixel (px_f, py_f). Por exemplo, para um domínio de representação da curva nos intervalos x=[-5, 5] e y=[-5, 5], x_i e x_f teriam respetivamente os valores -5 e 5, enquanto y_i e y_f possuiriam respetivamente os valores 5 e -5.

No cálculo dos pontos reais correspondentes aos pixéis da máscara, é estabelecida uma correspondência direta entre o domínio discreto da imagem (500×500 pixéis) e o domínio contínuo da curva (x=[-5, 5] e y=[-5, 5]), dada pelas equações (3) e (4) e que qualifica o algoritmo como híbrido.

3.3 Coloração dos pixéis da curva

Antes de se iniciar a coloração dos pixéis do domínio da imagem (veja-se Algoritmo 1), ambas as variáveis da estrutura correspondente a cada pixel, são inicializadas com o valor zero, o que significa que nenhum pixel pertence ainda à curva, nem tem vizinho algum que pertença.

A coloração de um pixel é feita com base no Corolário de Bolzano, que é um corolário do Teorema do Valor Intermédio. Este corolário estabelece que se uma função real tiver valores com sinais distintos em dois pontos do domínio, então haverá pelo menos um ponto intermédio em que a função tomará o valor zero. Ora, estes pontos em que a função se anula são precisamente os pontos da curva.

A verificação da existência de variação de sinal entre pixéis consecutivos é feita quer horizontal (da esquerda para a direita) quer verticalmente (de cima para baixo) para cada pixel. Isto é, a comparação do valor da função entre pixéis contíguos faz-se percorrendo o domínio (da imagem) por linhas no sentido descendente, sendo cada linha percorrida da esquerda para a direita. Na realidade o valor da função não é calculado nos pixéis, mas sim em dois pontos do domínio da função correspondentes a dois pixéis consecutivos da imagem.

No processo de coloração de pixéis adstritos a uma dada curva implícita foram considerados os seguintes casos:

1. ***Função com valor zero***. Se em algum ponto, correspondente a um pixel, a função tiver valor zero, então esse ponto pertence à curva e, nesse caso, o respetivo pixel também é colorido e a sua variável *p* é alterada para o valor 1.
2. ***Função <u>com</u> variação de sinal entre pixéis consecutivos***. Se após a comparação do valor de *f* em pixéis contíguos se verificar a existência de inversão de sinal, determina-se em qual dos respetivos pontos a função tem menor valor em módulo, colorindo-se então o pixel correspondente. Para além de ser colorido, o pixel é também assinalado como pertencente à curva, alterando-se o valor da variável *p* do pixel para 1.
3. ***Função <u>sem</u> variação de sinal entre pixéis consecutivos***. Pode também acontecer que o sinal do valor da função seja o mesmo nos dois pixéis analisados. Ora, isso não significa que a curva não passe entre eles. Sucede que pode haver um número par de ramos da curva que passam entre os dois pixéis (Figura 3.3), e desse modo os dois têm valor da função com o mesmo sinal. Pode ainda acontecer que em dois pixéis consecutivos a função tenha o mesmo sinal sem que entre eles passe um número par de ramos da curva. Essa última situação acontece quando existe um mínimo (em valor absoluto) da curva próximo do segmento que une os dois pixéis (*a* e *b*), mas que efetivamente não pertence à curva, não havendo assim nenhuma interseção da curva com tal segmento entre *a* e *b*.

O primeiro caso não é tão frequente quanto se possa pensar, pois a natureza discreta e finita da representação de números reais em computador pode impossibilitar que um ponto infinitesimalmente próximo da curva seja considerado um ponto da curva. No entanto, quando um pixel adstrito a um ponto do domínio da função é considerado como parte integrante da curva, tal pixel é simplesmente colorido.

Algoritmo 2 Algoritmo de coloração quando existe um número ímpar de segmentos da curva entre dois pixéis consecutivos

```
Input:
Δ: domínio de pixéis de tamanho [PXF-PXI] × [PYF-PYI]
Output:
Δ: domínio de pixéis colorido
begin
    for py ← PYI to PYF do
        Calcular coordenada y do ponto correspondente a py
        Calcular coordenada y1 do ponto correspondente a py+1
        for px ← PXI to PXF do
            Calcular coordenada x do ponto correspondente a px
            Calcular coordenada x1 do ponto correspondente a px+1
            if f(x,y) × f(x1,y) ≤ 0 then
                if |f(x,y)| ≤ |f(x1,y)| then
                    Assinalar pixel (px,py) como pixel da curva
                    Colorir pixel correspondente ao ponto (x,y)
                else
                    Assinalar pixel (px+1,py) como pixel da curva
                    Colorir pixel correspondente ao ponto (x+1,y)
                endif
            endif
            if f(x,y) × f(x,y1) ≤ 0 then
                if |f(x,y)| ≤ |f(x,y1)| then
                    Assinalar pixel (px,py) como pixel da curva
                    Colorir pixel correspondente ao ponto (x,y)
                else
                    Assinalar pixel (px,py+1) como pixel da curva
                    Colorir pixel correspondente ao ponto (x,y+1)
                endif
            endif
        endfor
    endfor
end
```

O segundo caso refere-se à situação em que um número ímpar de segmentos da função passa entre dois pixéis consecutivos. Ora, a passagem de um só segmento é o caso mais usual, mas há sempre a possibilidade, ainda que dificilmente, de o mesmo acontecer com qualquer outro

número ímpar de segmentos. No entanto, em termos de visualização da curva, não faz qualquer diferença o número ímpar de segmentos que passam entre dois pixéis consecutivos, colorindo-se para tanto um dos pixéis (Algoritmo 2).

O terceiro caso acontece quando, em dois pixéis contíguos, a função tem o mesmo sinal. Neste caso, é feito um estudo mais aprofundado do valor da função em pontos compreendidos entre os dois pixéis e que distam entre si 0.000001 vezes a distância entre os dois pixéis. A Figura 3.3 esquematiza esses pontos, em que:

- *a* e *b* são pixéis contíguos horizontalmente;
- *a* e *d* são pixéis contíguos verticalmente;
- p_1, p_2, p_3, …, p_n são os pontos em que se divide o segmento $\overline{ab}$ e que distam entre si 0.000001* $\overline{ab}$;
- *(x,y)* é o ponto correspondente ao pixel *a*;
- *(x_prox,y)* é o ponto correspondente ao pixel *b*;
- *(x,y_prox)* é o ponto correspondente ao pixel *d*;
- PSx = (x_prox-x)×0.000001;
- xS = x+PSx;
- PSy = (y-y_prox)×0.000001;
- yS = y-PSy;
- os segmentos mais carregados representam a curva.

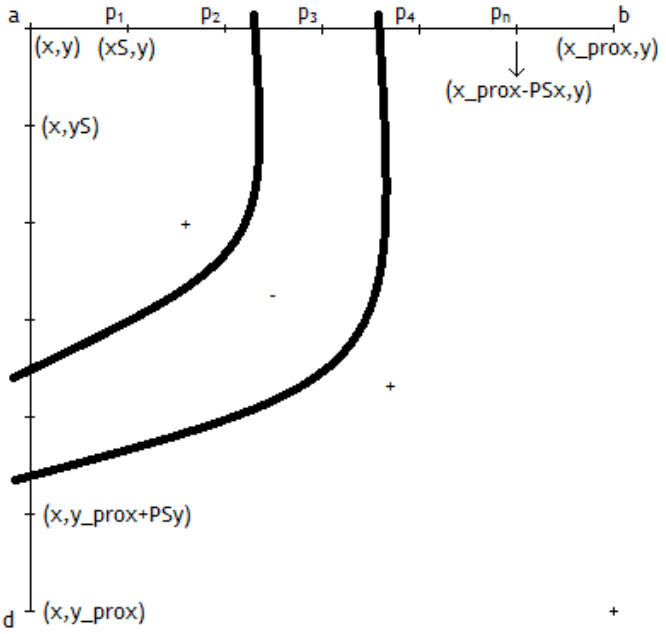


Figura 3.3: Exemplo de dois ramos da curva entre pixéis contíguos.

Algoritmo 3 Algoritmo de coloração quando existe um número par de segmentos da curva ou de um ponto correspondente a um mínimo (em valor absoluto) da função, entre 2 pixéis

---

Input: **Δ**: domínio de pixéis de tamanho [**PXF-PXI**] × [**PYF-PYI**]
Output: **Δ**: domínio de pixéis colorido
begin
for *py* ← *PYI* **to** *PYF* **do**
    Calcular coordenada *y* do ponto correspondente a *py*
    Calcular coordenada *y1* do ponto correspondente a *py+1*
    for *px* ← *PXI* **to** *PXF* **do**
        Calcular coordenada *x* do ponto correspondente a *px*
        Calcular coordenada *x1* do ponto correspondente a *px+1*
        if $f(x,y) \times f(x1,y) > 0$ **then**
            if $|f(xS,y)| < |f(x,y)|$ **&&** $|f(x\_prox-PSx,y)| < |f(x\_prox,y)|$ **then**
                for *xS* ← *xS* **to** *(x_prox-PSx)* **step** *PSx*
                    if $|f(xS+PSx,y)| > |f(xS,y)|$ **then**
                        if $|f(xS,y)| < 0.00000000009$ **then**
                            Colorir pixel correspondente ao ponto *(xS,y)*
                            Assinalar pixel *(px,py)* como pixelda curva
                        endif
                        break
                    endif
                endfor
            endif
        endif
        if $f(x,y) \times f(x,y1) > 0$ **then**
            if $|f(x,yS)| < |f(x,y)|$ **&&** $|f(x,y\_prox+PSy)| < |f(x,y\_prox)|$ **then**
                for *yS* ← *yS* **to** *(y_prox+PSy)* **step** *-PSy*
                    if $|f(x,yS-PSy)| > |f(x,yS)|$ **then**
                        if $|f(x,yS)| < 0.00000000009$ **then**
                            Colorir pixel correspondente ao ponto *(x,yS)*
                            Assinalar pixel *(px,py)* como pixel da curva
                        endif
                        break
                    endif
                endfor
            endif
        endif
    endfor
endfor
end

As situações consideradas no terceiro caso e descritas no Algoritmo 3 acontecem quando o valor da função diminui no sentido das extremidades para o interior do segmento $\overline{ab}$, ou seja, $f(a)>f(p_1)$ e $f(p_n)<f(b)$. Para determinar com mais exatidão os pontos de $\overline{ab}$ pertencentes à curva, no caso de existirem, percorre-se o segmento com passo de tamanho de $0.000001\times\overline{ab}$, a partir de p_1 em direção a p_n, calculando o valor da função em todos os pontos p_i.

À medida que o segmento é percorrido, e caso o valor da função diminua em valor absoluto, então isso significa que se caminha no sentido da aproximação progressiva da curva. Quando em algum p_i o valor de f em valor absoluto aumentar relativamente a $f(p_{i-1})$, significa que a interseção da curva com o segmento $\overline{ab}$, ou o ponto correspondente a um mínimo (em valor absoluto) da curva, se localiza entre p_{i-1} e p_i. Para identificar com exatidão qual destas situações se verifica é testado o valor de f em p_{i-1}.

Se $f(p_{i-1})<9\times10^{-11}$, considera-se que p_{i-1} pertence à curva, ou seja, existe uma interseção da curva com o segmento $\overline{ab}$ no ponto p_{i-1}, sendo este último colorido e o pixel a assinalado como pertencente à curva, na matriz de pixéis. Em termos de visualização da curva é indiferente assinalar o pixel a ou o b como pertencente à curva, uma vez que p_{i-1} se localiza algures entre os dois pixéis, portanto, por convenção é o pixel da esquerda do segmento que é assinalado como pertencente à curva. No caso de um segmento entre dois pixéis contíguos verticalmente (a e d da Figura 3.3), o pixel assinalado como pertencente à curva é o superior.

Se $f(p_{i-1})>9\times10^{-11}$, podem-se verificar duas situações:

- Existe um ponto do segmento $\overline{ab}$ que corresponde a um mínimo da curva, e nesse caso nenhum dos pixéis é assinalado como pertencente à curva, nem nenhum ponto do segmento é colorido.
- p_{i-1} pode pertencer à curva, mesmo se $f(p_{i-1})$ for, em módulo, maior que 9×10^{-11}. Isto mostra que este valor, abaixo do qual um ponto é considerado ponto da curva, pode não ser um valor de referência, que permita classificar inequivocamente um ponto como pertencente à curva, ou não. Neste caso, o procedimento do algoritmo é o mesmo da situação anterior, ou seja, nenhum dos pixéis a e b é assinalado como pertencente à curva, o que provoca uma descontinuidade na curva, no caso de p_{i-1} ser mesmo um ponto da curva. A resolução desta descontinuidade será explicada na seção seguinte.

Após ser determinada uma interseção da curva com o segmento $\overline{ab}$, ou um ponto correspondente a um mínimo da curva, algures entre os dois pixéis, a análise do segmento é interrompida mesmo sem p_n ser atingido, porque se considera que, mesmo no caso de existirem duas ou mais interseções da curva com $\overline{ab}$, todas elas podem ser representadas

apenas por um pixel. Isto porque em termos visuais, é impercetível se se trata de dois ou mais ramos da curva, tendo em conta que a distância que os separa é inferior à distância entre dois pixéis.

Note-se que todo este procedimento é aplicado tanto a pixéis contíguos horizontalmente, como verticalmente, sempre que nos dois pixéis a função tenha o mesmo sinal, como se observa no Algoritmo 3.

Finda a análise de todos os pixéis do domínio, para cada um deles é contado o número de pixéis vizinhos que pertencem à curva; esse número é guardado na variável *nv* da estrutura de dados *pixel* e que será útil na resolução de eventuais problemas de descontinuidade de pixelização da curva.

3.4 Resolução das descontinuidades da pixelização da curva

Como foi anteriormente explicado, é possível que entre dois pixéis com igual sinal da função, passem dois ou mais ramos da curva. No cálculo da interseção destes ramos com o segmento $\overline{ab}$ apenas é encontrado um possível e único ponto de interseção da curva com o segmento, o ponto p_{i-1}, que é considerado como pertencente ao primeiro ramo da curva, ou seja o ramo mais próximo do pixel *a* (Figura 3.3). A interseção do segundo ramo e dos seguintes, se existirem, com o segmento $\overline{ab}$ resulta nos pontos localizados entre p_i e p_n, mas estes não são calculados, porque em termos de visualização da curva, os vários ramos, entre os dois pixéis, podem ser representados apenas pelo ponto p_{i-1}.

Pode acontecer que o ponto p_{i-1} que foi encontrado pertença à curva mas $f(p_{i-1}) > 9\times10^{-11}$, logo p_{i-1} não é considerado ponto da curva e de acordo com o algoritmo 3 nenhum pixel é colorido, provocando assim uma descontinuidade na curva, como se observa na Figura 3.4.

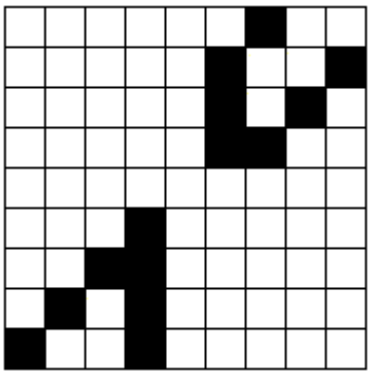


Figura 3.4: Exemplo de uma curva com uma descontinuidade de pixéis.

A Figura 3.5 representa um domínio cujos pixéis representados a preto definem a curva. Os pixéis de sinal '+' são aqueles que têm um valor positivo da função, ao passo que os pixéis de

sinal '-' têm um valor negativo da função. Segundo o Algoritmo 2 não é assumida a existência de ramos da curva entre os pixéis *a* e *b*, porque de acordo com a mesma figura a função tem o mesmo sinal nos dois pontos correspondentes aos dois pixéis, mantendo-se assim a descontinuidade.

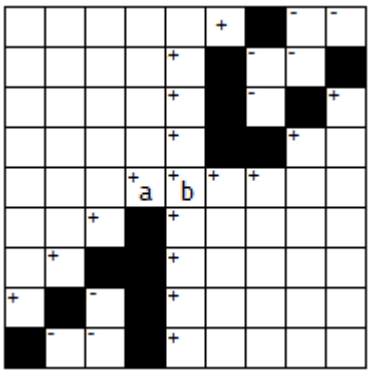


Figura 3.5: Exemplo de uma curva com indicação do sinal de *f* nos pixéis próximos da curva.

Quando um pixel é considerado como pertencente à curva, para além de ser colorido, é também guardada essa informação na variável *p* da estrutura *pixel*. Após todos os pixéis do domínio terem sido analisados, são contados, para cada um deles, quantos dos seus vizinhos pertencem à curva. Este número de pixéis vizinhos é guardado na variável *nv* da estrutura *pixel*. Estas duas informações relativas a cada pixel permitem resolver as descontinuidades de pixéis existentes na curva, da seguinte forma:

- Se algum pixel pertencente à curva tiver dois vizinhos contíguos entre si, que também pertençam à curva, é colorido o pixel oposto para garantir continuidade dos segmentos da curva, como mostra a Figura 3.6, em que *c* é o pixel central, *v1* e *v2* são os pixéis vizinhos, contíguos, que pertencem à curva e *b* é o novo pixel a ser colorido.
- No caso de existir um pixel que pertença à curva e tenha apenas um vizinho não contíguo, é colorido o pixel oposto a esse vizinho, como representado da Figura 3.7, em que *v1* pertence à curva e é o pixel vizinho do pixel central *c*. O novo pixel *b* será então colorido.

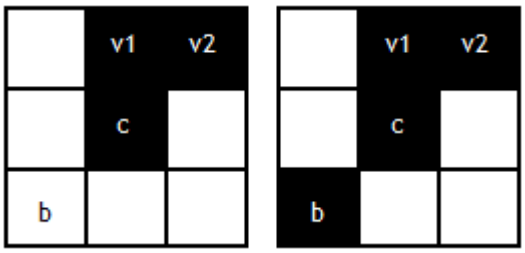


Figura 3.6: Dois pixéis contíguos (*v1* e *v2*) vizinhos do pixel central *c*.

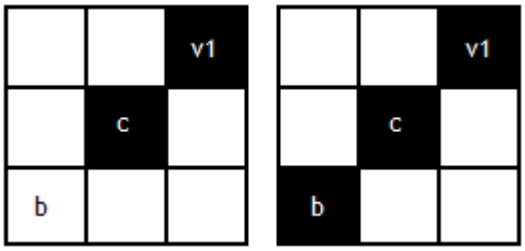


Figura 3.7: Um pixel vizinho (*v1*) do pixel central *c*.

A Figura 3.8 exemplifica uma singularidade, antes e depois de ser resolvido o problema de descontinuidade. Neste caso foi colorido o pixel *b* porque o pixel central *c* possui dois vizinhos (v1 e v2) contíguos entre si.

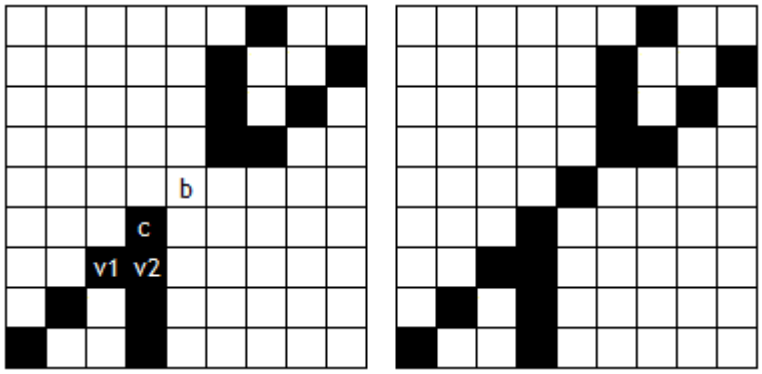


Figura 3.8: Antes e depois do tratamento do problema de descontinuidade numa singularidade.

3.5 Resolução de pontos isolados da curva

Aquando da análise da máscara para a deteção e correção de eventuais problemas de descontinuidade da curva são também identificados e representados os pontos isolados da curva, no caso de existirem. Um ponto isolado é um ponto da curva que não tem qualquer vizinho pertencente à curva; é óbvio que nesse ponto a função tem valor 0. O algoritmo desenvolvido colora um pixel correspondente a um ponto isolado se algum dos seguintes casos ocorrer:

- Nesse pixel a função tem exatamente valor 0 (veja-se a condição de igualdade do primeiro **if** do Algoritmo 2).

- O valor da função nesse pixel é, em módulo, inferior ao valor de *f* em qualquer pixel vizinho, com a condição de que todos os pixéis têm o mesmo sinal da função.

Note-se que para além de se efetuar a coloração do ponto ou pixel isolado, e ativar a variável *p* da estrutura de dados *pixel* com o valor 2, para assim se conseguir distinguir de outros

pontos da curva (que têm o valor *p*=1), a variável *nv* da estrutura de dados *pixel* continuará com o valor 0 já que o pixel continua sem vizinhos pertencentes à curva.

3.6 Desvantagens do algoritmo

No estado atual de desenvolvimento do algoritmo de pixelização de curvas implícitas, pode dizer-se que ainda tem as seguintes insuficiências:

- ***Componentes sem variação de sinal***. As curvas que contêm componentes sem variação de sinal não são ainda representadas de forma correta, pois numa primeira abordagem o algoritmo baseava-se tão só na variação de sinal entre dois pontos para representação da curva (Corolário de Bolzano). No entanto, o algoritmo evoluiu de maneira a se conseguir resolver este problema a breve trecho.

- ***Regiões extensas de pixéis sem segmentos da curva***. Este problema tem mais que ver com o desempenho global do algoritmo do que propriamente com as garantias topológicas das curvas. Isto acontece porque o algoritmo tem de analisar de igual modo todos os pares de pixéis do domínio, mesmo quando nessa região não passa a curva. Esta análise exaustiva da existência de algum segmento da curva entre cada par de pixéis consecutivos horizontal e verticalmente deteriora o desempenho do algoritmo, ainda que seja de extrema importância para representar corretamente os pontos críticos da curva, como singularidades e extremos.

No entanto, espera-se que num futuro próximo se possa resolver estes problemas por forma a garantir a pixelização topologicamente correta de qualquer curva implícita, bem como melhorar o desempenho global do algoritmo através do descartar prévio de regiões de pixéis onde não existe qualquer segmento da curva.

3.7 Codificação do algoritmo

No desenvolvimento do algoritmo foram implementadas as seguintes funções em linguagem C:

`float f(float x, float y)` - Calcula e devolve o valor da função num ponto de coordenadas `(x,y)`.

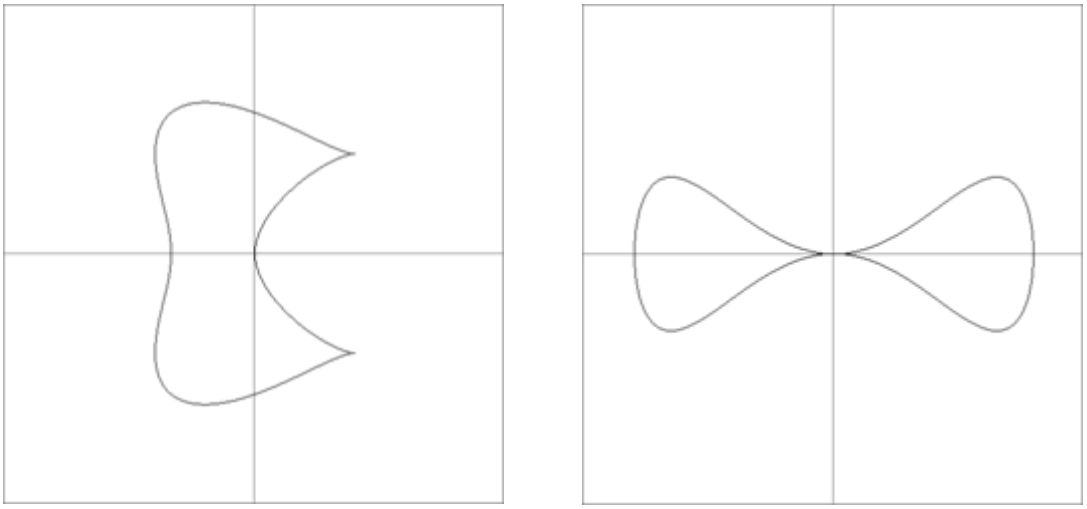
`void desenha_curva()` - Para o domínio da curva é construida a correspondente máscara de pixéis recorrendo à função `criarMascara()`. A essência do algoritmo encontra-se nesta função.

`Struct pixel **criarMascara (int px_x, int px_y)` - Permite criar a máscara ou matriz de pixéis recorrendo a funções de alocação dinâmica de memória. Os parâmetros `px_x` e `px_y` indicam o tamanho da máscara, em pixéis.

`Struct pixel **freeMascara(struct pixel **masc, int px_x, int px_y)` - Esta função serve para libertar a memória utilizada para processar a máscara de pixéis após todos os pixéis da máscara terem sido analisados, isto é, após a curva estar totalmente renderizada. Os parâmetros `px_x` e `px_y` indicam o tamanho da máscara, em pixéis e `masc` é a variável que contém a máscara.

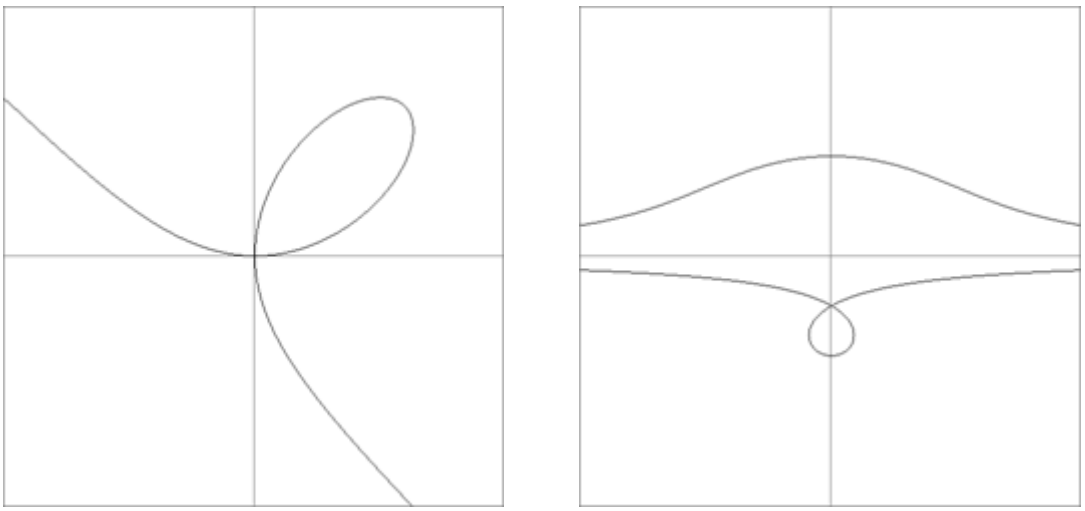
3.8 Exemplos de renderização de curvas implícitas

A seguir são apresentados alguns exemplos de curvas implícitas obtidos através do algoritmo desenvolvido nesta dissertação, mais concretamente no domínio $x = [-5,5]$ e $y = [-5,5]$, bem como os respetivos tempos de execução.



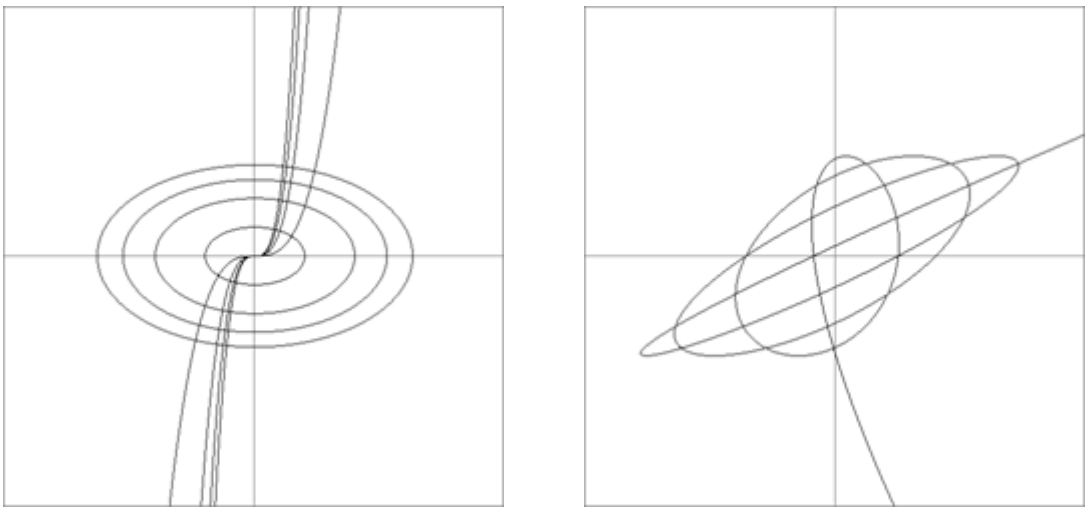
(a) t = 1 segundo (b) t < 1 segundo

Figura 3.9: Curvas algébricas (a) $(x-2)^2\left(x^2-4\right)+\left(y^2-4^2\right)^2=0$; (b) $256y^2-x^4\left(16-x^2\right)=0$.



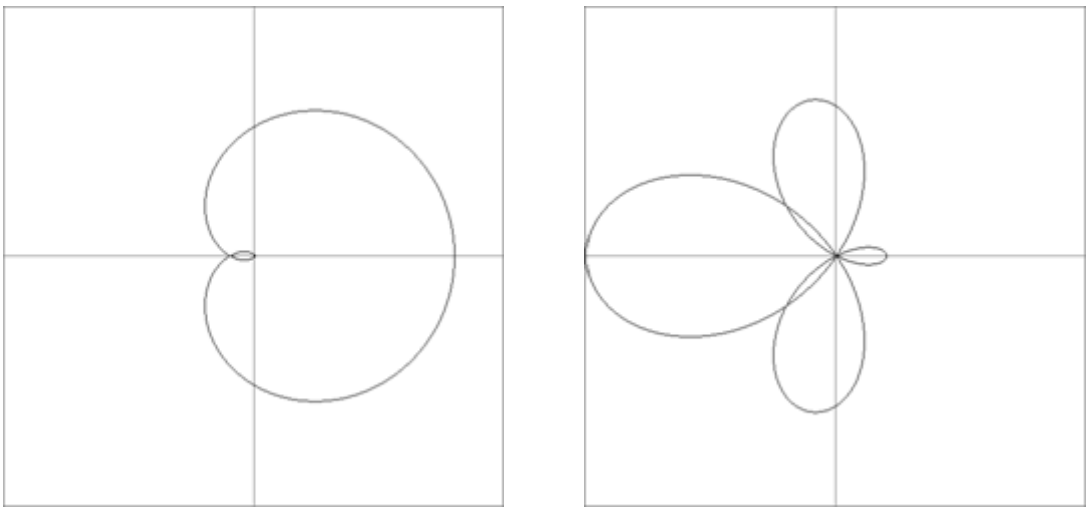
(a) t < 1 segundo (b) t = 1 segundo

Figura 3.10: Curvas algébricas: (a) $x^3 + y^3 - 6xy = 0$; (b) $x^2y^2 - (y+1)^2(4 - y^2) = 0$.



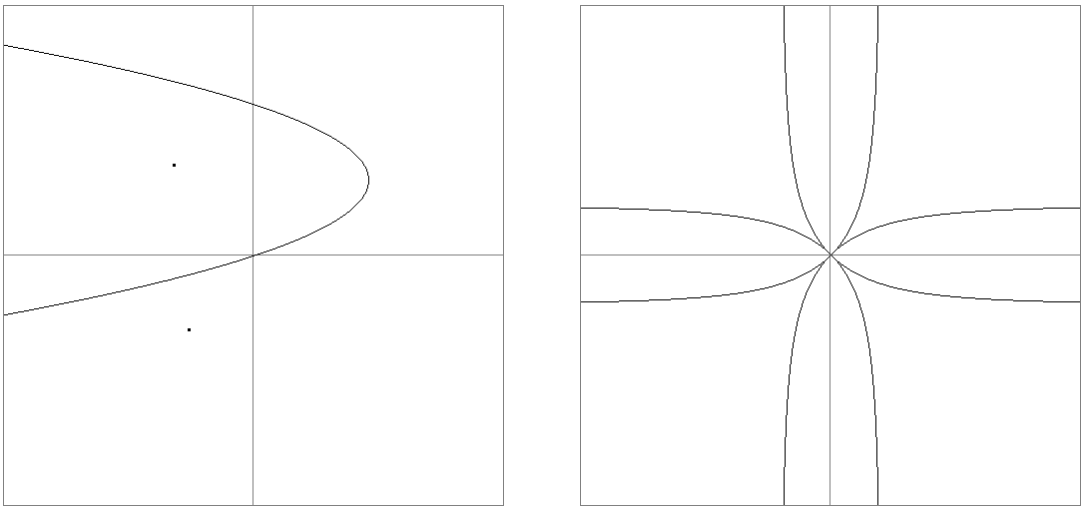
(a) t = 2 segundos (b) t = 27 segundos

Figura 3.11: Curvas algébricas: (a) $(y - x^3)(x^2 + 3y^2 - 1) * (y - 4x^3)(x^2 + 3y^2 - 4) * (y - 7x^3)(x^2 + 3y^2 - 7) * * (y - 10x^3)(x^2 + 3y^2 - 10) = 0$; (b) $-y^8 + x^7 - 7x^6y + 21x^5y^2 - 35x^4y^3 + 35x^3y^4 - 21x^2y^5 + 7xy^6 - y^7 + 8y^6 - -7x^5 + 35x^4y - 70x^3y^2 + 70x^2y^3 - 35xy^4 + 7y^5 - 20y^4 + 14x^3 - 42x^2y + 42xy^2 - 14y^3 + 16y^2 - 7x + 7y - 2 = 0$.



(a) t = 1 segundo (b) t < 1 segundo

Figura 3.12: Curvas algébricas: (a) $27\left(x^2+y^2\right)^2-4\left(x^2+y^2-x\right)^3=0$; (b) $\left(x^2+y^2\right)\left(x^2+y^2+2x\right)^2-9\left(x^2-y^2\right)^2=0$.



(a) t < 1 segundo (a) t < 1 segundo

Figura 3.13: Curvas algébricas: (a) $\left(x-2.3+\left(y-1.5\right)^2\right)*\left(\left(x+1.3\right)^2+\left(y+1.5\right)^2\right)*\left(\left(x+1.6\right)^2+\left(y-1.8\right)^2\right)=0$; (b) $x^2y^2\left(x^2+y^2\right)-\left(x^2-y^2\right)^2=0$.

Como seria de esperar as curvas com mais auto-interseções são as que demoram mais tempo a ser renderizadas. Chama-se a atenção também para os dois pontos isolados da Figura 3.13(a). A máquina utilizada para renderizar estas curvas foi um *laptop* equipado com um processador

Intel Core i5, com 4GB de memória principal, placa gráfica *onboard*, e sistema operativo Windows 7.

3.9 Notas finais

Conforme ficou demonstrado pelo algoritmo proposto, é possível renderizar curvas implícitas genéricas sem utilizar quaisquer técnicas de continuação ou de partição espacial do domínio da função, mesmo na presença de pontos críticos e de componentes da curva sem variação de sinal.

No entanto, as componentes sem variação de sinal é um problema que ainda não está completamente resolvido, mas o estudo e as incursões algorítmicas já realizadas a nível da próxima versão do algoritmo permitem afirmar que este problema será também solucionado.

Na sua essência, o algoritmo opera no domínio da imagem, à semelhança do que acontecia nos primórdios das técnicas de rasterização utilizadas em computação gráfica, das quais o algoritmo de Bresenham é um dos seus principais representantes.

Capítulo 4

Conclusões

O algoritmo apresentado cumpre o objetivo inicial desta dissertação, isto é, a pixelização de curvas implícitas. Porém, ao longo do seu desenvolvimento houve a necessidade de alterar o método inicialmente proposto devido ao fato de não ser eficaz, bem como não oferecer garantias topológicas na renderização de algumas curvas, em especial nas singularidades.

O trajeto do trabalho de investigação inicialmente delineado para o desenvolvimento do algoritmo previa a partição do domínio da função em quadrados com tamanho de aresta 0.05. Para cada quadrado eram determinadas as interseções da curva com as arestas, através da variação de sinal da função entre os dois vértices de cada aresta (Corolário de Bolzano). Está claro que se a referida aresta fosse atravessada por um par de ramos da curva, a condição de Bolzano não se verificava porque em ambos os vértices da aresta a função tinha o mesmo sinal. Assim, para determinar o número exato de ramos que atravessavam a aresta, fazia-se a análise de variação de sinal, nas extremidades de pequenos segmentos em que a aresta tinha sido previamente dividida. Sucede que uma aresta pode ser atravessada por inúmeros ramos da curva, muito próximos uns dos outros, o que dificultava o apuramento do número total de ramos que intersetavam a aresta, mesmo se os segmentos em que a aresta fosse dividida tivessem um tamanho muito reduzido.

Dependendo do número total de interseções da curva com as arestas de cada quadrado, inferíamos o comportamento da curva dentro do quadrado, processo que se tornava mais complexo quanto maior o número de interseções da curva com o quadrado. Com base nestes problemas tornou-se emergente a implementação de um método que permitisse o estudo da curva a partir do domínio da imagem. Esta abordagem surgiu pela simples razão de a unidade mínima de visualização ser o pixel. Assim, mesmo que por entre dois pixéis passem vários ramos da curva, segundo o algoritmo apresentado, apenas um dos pixéis é colorido, representando um ou vários pontos da curva; isto porque em termos visuais é impercetível se entre dois pixéis passam um ou mais ramos da curva.

O algoritmo descrito nesta dissertação faz a renderização de uma dada curva implícita no plano através da sua pixelização no domínio da imagem. Consideramos que esta abordagem é uma forma eficaz de representar curvas implícitas no plano 2D, uma vez que pontos críticos como singularidades e pontos isolados são corretamente renderizados. Quando comparado com algoritmos de continuação, por exemplo, este algoritmo apresenta-se mais fiável na representação de singularidades (incluindo as que são formadas por três ou mais ramos da curva) e de outros pontos críticos, uma vez que a renderização da curva é feita de igual modo

em todo o domínio, através da análise de pixéis vizinhos, não havendo um tratamento particular dos pontos críticos. Também não existe o problema de a curva renderizada fazer atalhos à curva real, ou fazer derivação de trajetória, ou mesmo a possibilidade de formação de ciclos durante a traçagem da curva, ao contrário do que acontece nos métodos de continuação, devido ao fato de a análise da mesma ser feita pixel a pixel e não baseada no seguimento da curva.

Como foi referido anteriormente, uma desvantagem que observámos ao algoritmo, que é a incapacidade de representar de forma adequada, ramos da curva que separem zonas do domínio com igual sinal da função, isto é, regiões do domínio em que f tem o mesmo sinal em ambos os lados de um ramo da curva. Ora, como o algoritmo atual apresentado se baseia na mudança de sinal da função entre pixéis vizinhos, consideramo-lo por enquanto inadequado para representar este tipo de curvas.

Em suma, a maior contribuição desta dissertação foi ser capaz de renderizar pontos críticos de curvas implícitas no domínio da imagem, independentemente da complexidade local da curva; por exemplo, quando há vários ramos da curva a confluírem num ponto de auto-interseção ou quando existem um ou mais pontos isolados. Ao contrário de outros algoritmos, o algoritmo aqui descrito utiliza o domínio da imagem em vez do domínio da função, ou seja não se enquadra nem na categoria dos algoritmos de continuação nem na categoria dos algoritmos de partição espacial.

Bibliografia

[1] E. Allgower e K. Georg, Introduction to Numerical Continuation Methods, *Classics in Applied Mathematics vol. 45, SIAM Press*, 1990.

[2] J. Bloomenthal, Poligonization of implicit surfaces, *Computer Aided Geometric Design,* vol. 5, n. 4, pp. 341–355, 1988.

[3] I. Braude, J. Marker, K. Museth, J. Nissanov, e D. Breen, Contour-based surface reconstruction using MPU implicit models, *Graphical Models*, vol. 69, n. 2, pp. 139-157, 2007.

[4] J. Bresenham, Algorithm for computer control of a digital plotter, *IBM Systems Journal*, vol. 4, n. 1, pp. 25–30, 1965.

[5] R. Chandler, A Tracking Algorithm For Implicitly Defined Curves, *IEEE Computer Graphics & Applications*, vol. 8, n. 2, pp. 83–89, 1988.

[6] H. Coxeter, Discrete groups generated by reflections, *The Annals of Mathematics*, vol. 35, n. 3, pp. 588-621, July 1934.

[7] D. Dobkin, S. Levy, W. Thurston e A. Wilks, Contour tracing by piecewise linear approximations. *ACM Transactions on Graphics*, vol. 9, n. 4, pp. 389-423, 1990.

[8] P. Emeliyanenko, E. Berberich e M. Sagraloff, Visualizing Arcs of Implicit Algebraic Curves, Exactly and Fast, *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part I*, pp. 608-619, 2009.

[9] A. Gomes, J. Morgado e E. Pereira, A BSP-Based Algorithm for Dimensionally Nonhomogeneous Planar Implicit Curves with Topological Guarantees, *ACM Transactions on Graphics*, vol. 28, n. 2, artigo 27, 2009.

[10] A. Gomes, I. Voiculescu, J. Jorge, B. Wyvill e C. Galbraith, *Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms*, Springer-Verlag, 2009.

[11] V. Gonçalves, C. Maia, L. Pimenta e G. Pereira, *Navegação de Robôs Utilizando Curvas Implícitas*, Controle & Automação, vol. 21, n. 1, pp. 43-57, 2010.

[12] L. Gonzalez-Vega e I. Necula, Efficient topology determination of implicitely defined algebraic plane curves. *Computer Aided Geometric Design*, vol. 19, pp. 719-743, 2002.

[13] F. Hussain e M. Pitteway, *Rasterizing the outlines of fonts*, Electronic Publishing, vol. 6, n. 3, pp. 171-181, 1993.

[14] S. Krishnan e D. Manocha, Numeric-Symbolic Algorithms for Evaluating One-Dimensional Algebraic Sets, *Proc. ISSAC*, pp. 59-67, 1995.

[15] K. Mochizuki, *Robust and adaptive polygonization of implicit curves and surfaces*, University of Aizu, Fevereiro, 2004.

[16] T. Moller e R. Yagel, *Efficient rasterization of implicit functions*, Ohio State University, 1995.

[17] J. Morgado e A. Gomes, A derivative-free tracking algorithm for implicit curves with singularities, Proceedings of the 4th International Conference on Computational Conference (ICCS'04), vol. 3039 de *Lecture Notes in Computer Science*. Springer-Verlag, New York, 2004.

[18] J. Morgado e A. Gomes, A generalized false position numerical method for finding zeros and extrema of a real function, Proceedings of the International Conference in Computational Methods in Sciences and Engineering (ICCMSE'05), vol. 4 de *Lecture Series on Computer and Computational Sciences*, Brill Academic Publishers, pp. 425-428, 2004.

[19] J. Morgado e A. Gomes, Fast representation of implicit curves through space subdivision, Proc Ibero-American Symp., *Computer Graphics*, Guimarães, Portugal, pp. 27-30, Julho, 2002.

[20] J. Morgado, Poligonização de Curvas e Superfícies Implícitas Não-Homogéneas com Preservação Topológica, Universidade da Beira Interior, Novembro, 2005.

[21] J. Morgado, A. Gomes, Rendering Implicit Curves with Isolated Points through Binary Space Partitioning, *The Virtual - Portuguese Journal of Computer Graphics*, Dezembro, 2004.

[22] A. Raposo e A. Gomes, Polygonization of multi-component non-manifold implicit surfaces through a symbolic-numerical continuation algorithm, Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and South-East Asia (GRAPHITE'06), *ACM Press*, pp. 399-406, 2006.

[23] J. Snyder, Generative Modelling for Computer Graphics and CAD, *Academic Press*, 1992.

[24] J. Snyder, Interval analysis for computer graphics. *Computer Graphics*, vol. 26, n. 2, (SIGGRAPH'92), pp. 121-130, Julho 1992.

[25] G. Taubin, Distance Approximations for Rasterizing Implicit Curves, *ACM Transactions on Graphics*, vol. 13, pp. 3-42, 1994.

[26] G. Taubin, Estimation of Planar Curves, Surfaces and Nonplanar Space Curves Defined by Implicit Equations with Applications to Edge and Range Image Segmentation, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, n. 11, pp. 1115-1138, 1991.

[27] H. Xu, D. Li e S. Tang, *Kinematic Inbetweening along Implicit Curves for Motion Animations*. ICAT Workshops, 16th International Conference on Artificial Reality and Telexistence, pp. 1-4, 2006.

[28] Y. Zheng-sheng, C. Yao-zhi, O. Min-jae, K. Tae-wan e P. Qun-sheng, An efficient method for tracing planar implicit curves, *Journal of Zhejiang University SCIENCE A*, pp. 1115-1123, 2006.