



**Sueni de Souza Arouca**

**Método implícito para reconstrução de curvas  
a partir de pontos esparsos**

**Dissertação de Mestrado**

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-graduação em Matemática Aplicada do Departamento de Matemática da PUC-Rio

Orientador : Prof. Hélio Côrtes Vieira Lopes  
Co-Orientador: Prof. Luiz Carlos Pacheco R. Velho

Rio de Janeiro  
Fevereiro de 2006



**Sueni de Souza Arouca**

**Método implícito para reconstrução de curvas  
a partir de pontos esparsos**

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-graduação em Matemática Aplicada do Departamento de Matemática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

**Prof. Hélio Côrtes Vieira Lopes**

Orientador

Departamento de Matemática — PUC-Rio

**Prof. Luiz Carlos Pacheco R. Velho**

Co-Orientador

IMPA

**Prof. Marco Antônio Grivet Mattoso Maia**

CETUC — PUC-Rio

**Prof. Geovan Tavares dos Santos**

Departamento de Matemática — PUC-Rio

**Prof. Carlos Tomei**

Departamento de Matemática — PUC-Rio

**Prof. José Eugenio Leal**

Coordenador Setorial do Centro Técnico Científico — PUC-Rio

Rio de Janeiro, 13 de Fevereiro de 2006

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

### **Sueni de Souza Arouca**

Graduou-se em Licenciatura em Matemática na Universidade do Estado do Rio de Janeiro – UERJ (Rio de Janeiro, Brasil).

#### Ficha Catalográfica

Arouca, Sueni de Souza

Método implícito para reconstrução de curvas a partir de pontos esparsos / Sueni de Souza Arouca; orientador: Hélio Côrtes Vieira Lopes; co-orientador: Luiz Carlos Pacheco R. Velho. — Rio de Janeiro : PUC-Rio, Departamento de Matemática, 2006.

v., 64 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Matemática.

Inclui referências bibliográficas.

1. Matemática – Tese. 2. Aproximações de curvas implícitas. 3. Modelagem geométrica. 4. MPU. 5. Ridge regression. I. Lopes, Hélio Côrtes Vieira. II. Velho, Luiz Carlos Pacheco R.. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Matemática. IV. Título.

CDD: 510

## Agradecimentos

À minha família pelo apoio e incentivo.

Ao meu orientador Professor Hélio Lopes pelo apoio, carisma de sempre, e incentivo para a realização deste trabalho.

Ao meu namorado Alexandre Fernandes pelo carinho e apoio.

À Capes pelos auxílios concedidos.

Aos funcionários do departamento de Matemática, em especial a Creuza.

Aos meus colegas e amigos da PUC-Rio, em especial aos amigos Marcos Lage e Catiúscia Borges.

## Resumo

Arouca, Sueni de Souza; Lopes, Hélio Côrtes Vieira; Velho, Luiz Carlos Pacheco R.. **Método implícito para reconstrução de curvas a partir de pontos esparsos**. Rio de Janeiro, 2006. 64p. Dissertação de Mestrado — Departamento de Matemática, Pontifícia Universidade Católica do Rio de Janeiro.

Nas aplicações em computação gráfica e processamento de imagens, curvas e superfícies implícitas têm sido reconhecidas como a representação mais útil de objetos 2D ou 3D, principalmente porque elas permitem a descrição de formas complexas por uma fórmula. A maioria dos métodos implícitos usam curvas algébricas para aproximar globalmente a fronteira do objeto em uma imagem binária. Quando a forma do objeto é complexa, é comum elevar o grau da curva a fim de obter mais precisão na aproximação. Uma solução alternativa é decompor hierarquicamente o domínio em partes compactas e obter aproximações locais para o objeto em cada parte, e então juntar os pedaços com o objetivo de obter uma descrição global do objeto. O principal objetivo deste trabalho é apresentar um novo método de aproximação de curvas implícitas a partir de pontos esparsos que melhora o estado da arte.

## Palavras-chave

Aproximações de curvas implícitas. Modelagem geométrica. MPU. Ridge regression.

## Abstract

Arouca, Sueni de Souza; Lopes, Hélio Côrtes Vieira; Velho, Luiz Carlos Pacheco R.. **Implicit method for curve reconstruction from sparse points**. Rio de Janeiro, 2006. 64p. MsC Thesis — Department of Matemática, Pontifícia Universidade Católica do Rio de Janeiro.

In the field of computer vision and image analysis, implicit curves and surfaces have been recognized as the most useful representation for 2D or 3D objects, mainly because they allow description of shapes by a formula. Most of implicit methods uses algebraic curves to fit globally the frontier of the foreground in a binary image. When the foreground shape is complex, it is common to elevate the curve degree in order to obtain more precision on the approximation. An alternative solution is to decompose the domain hierarchically in compact parts and obtain local approximation for the object in each part, and then patch all together in order to obtain a global description of the object. The main objective of this work is to present a new method for implicit curve fitting from sparse point that improves the state of the art.

## Keywords

Implicit curve fitting. Geometric modeling. Multi-level partition of unity. Ridge regression.

## Sumário

1	Introdução	<b>11</b>
1.1	Descrição do problema	12
1.2	Contribuição	12
1.3	Organização da dissertação	13
2	Aproximação por curvas implícitas e partição da unidade	<b>14</b>
2.1	Curvas Algébricas	14
2.2	Aproximações algébricas por mínimos quadrados	15
2.3	Partição da Unidade	19
2.4	O método MPU	20
3	O novo método implícito	<b>25</b>
3.1	Descrição	25
3.2	Implementação	27
4	Resultados	<b>34</b>
4.1	Curvas Suaves	35
4.2	Curvas com singularidades	51
4.3	Completando buracos	59
5	Conclusão e trabalhos futuros	<b>62</b>

## Lista de figuras

1.1	Dados de entrada: pontos amostrados de uma curva e os vetores normais em cada um desses pontos.	12
1.2	Resultado do método: uma curva implícita que aproxima os dados de entrada da figura 1.1.	13
2.1	As figuras (a) e (d) ilustram aproximações obtidas usando o método <i>classical least squares</i> . Já nas figuras (b) e (e) foi utilizado o método <i>gradient one fitting</i> , e nas figuras (c) e (f) o método <i>ridge regression</i> . O grau do polinômio foi escolhido como sendo 6 para o alicate e 8 para o avião.	19
2.2	Exemplo da construção de uma <i>Quad-Tree</i> gerando uma subdivisão do domínio adaptada aos pontos.	21
2.3	Exemplo da região de suporte para cada nó da <i>Quad-Tree</i> .	22
2.4	Aumentando o raio do círculo do suporte para obter uma garantia de existir solução para o cálculo das quádricas.	23
2.5	Toro: Figura gerada pelo software MPU.	23
2.6	Bunny: Figura gerada pelo software MPU.	24
3.1	Refinamento local que permite uma melhor aproximação global.	28
3.2	Organização da classe <code>Quad_Tree</code> .	29
3.3	Classe <code>Curva_Implicita</code> .	30
4.1	Dois círculos: método 1 usando um polinômio de grau $d$ com $\mu = 0.125$ .	35
4.2	Dois círculos: método 2 usando um polinômio de grau $d$ com $\mu = 0.125$ e $\kappa = 0.001$ .	36
4.3	Dois círculos: método 3 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	37
4.4	Dois círculos: método 4 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ , $\kappa = 0.001$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	38
4.5	Taubin: método 1 usando um polinômio de grau $d$ com $\mu = 0.125$ .	39
4.6	Taubin: método 2 usando um polinômio de grau $d$ com $\mu = 0.125$ e $\kappa = 0.001$ .	40
4.7	Taubin: método 3 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	41
4.8	Taubin: método 4 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ , $\kappa = 0.001$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	42
4.9	Sorriso: método 1 usando um polinômio de grau $d$ com $\mu = 0.125$ .	43
4.10	Sorriso: método 2 usando um polinômio de grau $d$ com $\mu = 0.125$ e $\kappa = 0.001$ .	44
4.11	Sorriso: método 3 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	45

4.12	Sorriso: método 4 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ , $\kappa = 0.001$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	46
4.13	Hipociclóide: método 1 usando um polinômio de grau $d$ com $\mu = 0.125$ .	47
4.14	Hipociclóide: método 2 usando um polinômio de grau $d$ com $\mu = 0.125$ e $\kappa = 0.001$ .	48
4.15	Hipociclóide: método 3 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	49
4.16	Hipociclóide: método 4 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ , $\kappa = 0.001$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	50
4.17	Cúspide: método 1 usando um polinômio de grau $d$ com $\mu = 0.125$ .	51
4.18	Cúspide: método 2 usando um polinômio de grau $d$ com $\mu = 0.125$ e $\kappa = 0.001$ .	52
4.19	Cúspide: método 3 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	53
4.20	Cúspide: método 4 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ , $\kappa = 0.001$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	54
4.21	Nephroid: método 1 usando um polinômio de grau $d$ com $\mu = 0.125$ .	55
4.22	Nephroid: método 2 usando um polinômio de grau $d$ com $\mu = 0.125$ e $\kappa = 0.001$ .	56
4.23	Nephroid: método 3 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	57
4.24	Nephroid: método 4 usando localmente um polinômio de grau $d$ com $\mu = 0.125$ , $\kappa = 0.001$ e uma <i>Quad-Tree</i> com nível máximo igual a $l_{max}$ .	58
4.25	Círculo: (a) método 3 e (b) método 4 usando localmente um polinômio de grau $d = 2$ com $\mu = 0.17$ , $\kappa = 0.01$ e uma <i>Quad-Tree</i> com nível máximo igual a 5	59
4.26	Sorriso: (a) método 3 e (b) método 4 usando localmente um polinômio de grau $d = 2$ com $\mu = 0.17$ , $\kappa = 0.01$ e uma <i>Quad-Tree</i> com nível máximo igual a 5	60
4.27	Taubin: (a) método 3 e (b) método 4 usando localmente um polinômio de grau $d = 2$ com $\mu = 0.17$ , $\kappa = 0.01$ e uma <i>Quad-Tree</i> com nível máximo igual a 4	61

## Lista de tabelas

2.1	Tabela do número de coeficientes do polinômio	15
-----	-----------------------------------------------	----

# 1

## Introdução

Com o surgimento dos dispositivos óticos que capturam as formas geométricas de objetos através de pontos adquiridos em sua fronteira, veio o problema de reconstrução de curvas e superfícies. O problema de reconstrução a partir de pontos esparsos tem recebido muita atenção pela comunidade científica porque tais dispositivos de aquisição estão cada vez mais acessíveis. Além disso a sua solução gera muitas aplicações em computação gráfica [11], em visão computacional [17] e em modelagem geométrica [14]. É importante observar que ele pode se tornar um problema bastante complexo devido não só ao desconhecimento das relações de vizinhança e proximidade, mas também pela presença de ruído. Várias técnicas foram estudadas para resolvê-lo. Existem as que utilizam árvore geradoras mínimas [7], as que são baseadas em triangulação de Delaunay [3, 4, 5, 8], as que utilizam parametrizações locais [1, 2, 11, 12], e as que usam formulações implícitas [9, 13, 18], entre muitas outras. Essa dissertação estuda a técnica implícita de reconstrução de curvas considerando dados esparsos.

As curvas e superfícies implícitas são reconhecidamente as representações mais práticas de objetos 2D ou 3D, principalmente porque elas permitem a descrição de formas complexas por uma função computável [6, 9]. A maioria dos métodos implícitos usam curvas algébricas para aproximar globalmente a fronteira do objeto em uma imagem binária [17]. Quando a forma do objeto é complexa, é comum elevar o grau da curva a fim de obter mais precisão na aproximação. Uma solução alternativa é decompor hierarquicamente o domínio em partes compactas e obter aproximações locais para o objeto em cada parte, e então juntar esses pedaços com o objetivo de obter uma descrição global para ele. Othake et al. [13] propôs um método implícito em multi-resolução que utiliza essa solução. Ele foi originalmente proposto para reconstrução de superfícies em  $\mathbb{R}^3$ . Como esse método implícito usa partição da unidade, ele foi denominado MPU (Multilevel Partition of Unity Implicit).

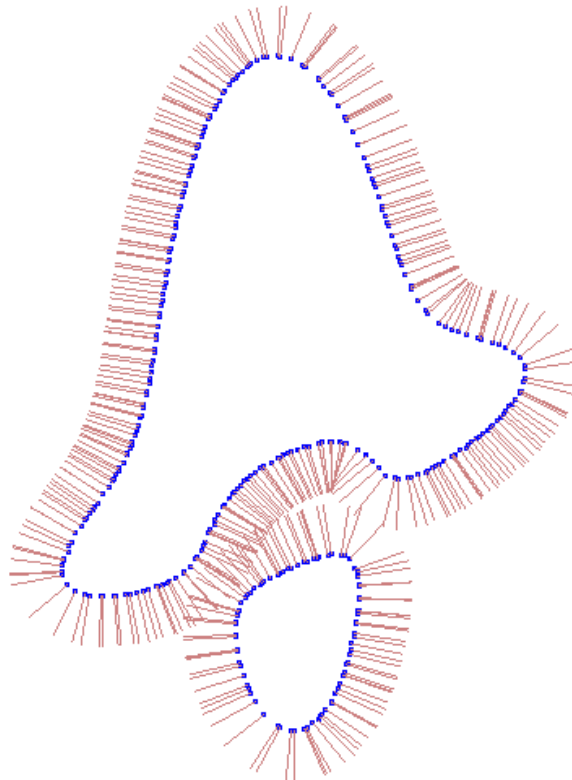


Figura 1.1: Dados de entrada: pontos amostrados de uma curva e os vetores normais em cada um desses pontos.

## 1.1

### Descrição do problema

Considere um conjunto de pontos  $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_q\} \in \mathbb{R}^2$  amostrados de uma curva planar  $\mathcal{C}$ . Suponha também que para cada ponto  $\mathbf{p}_i$  é dado o vetor normal unitário à curva  $\mathbf{n}_i$  no ponto. O conjunto desses vetores normais, denotado por  $\mathcal{N}$ , indica a orientação da curva.

A figura 1.1 ilustra um exemplo dos dados de entrada do problema.

O objetivo desse trabalho é obter uma função implícita  $F : \mathbb{R}^2 \rightarrow \mathbb{R}$  tal que a isocurva  $F^{-1}(0)$  aproxima adaptativamente  $\mathcal{C}$  usando um controle do erro local.

A figura 1.2 mostra o resultado do método para os dados referentes à figura 1.1.

## 1.2

### Contribuição

O método a ser proposto neste trabalho segue as principais idéias do MPU [13] restringindo-o ao plano. Entretanto, fornece diferentes estratégias para a aproximação local que permitem melhorar sua estabilidade numérica.

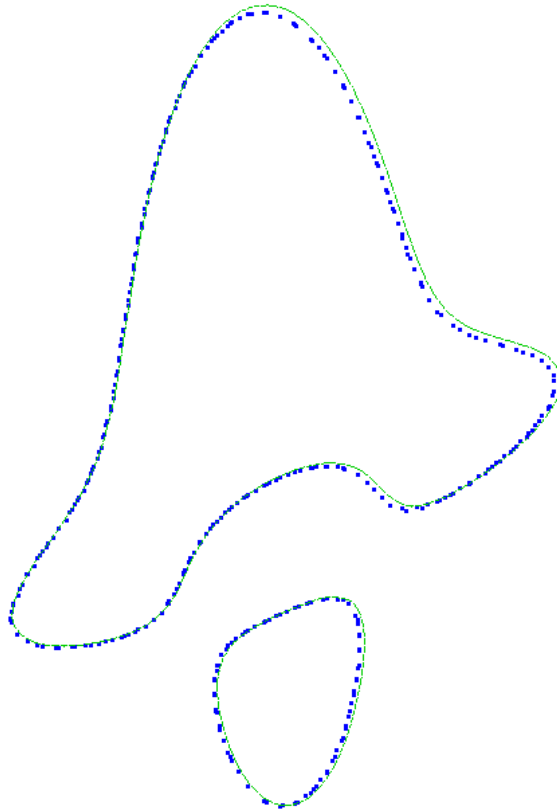


Figura 1.2: Resultado do método: uma curva implícita que aproxima os dados de entrada da figura 1.1.

### 1.3

#### Organização da dissertação

Essa dissertação foi organizada da seguinte maneira: o capítulo 2 introduz alguns conceitos básicos sobre aproximação por curvas algébricas, mínimos quadrados e partição da unidade. O capítulo 3 introduz o novo método e os seus algoritmos, assim como os detalhes de sua implementação. O capítulo 4 apresenta os resultados e as comparações. Finalmente, o capítulo 5 conclui e apresenta propostas de trabalhos futuros.

## 2

### Aproximação por curvas implícitas e partição da unidade

Este capítulo expõe alguns conceitos básicos necessários para o entendimento deste trabalho.

#### 2.1

##### Curvas Algébricas

Um subconjunto  $\mathcal{O} \subset \mathbb{R}^2$  é chamado de uma *curva implícita* se existe uma função  $F : U \rightarrow \mathbb{R}$ ,  $\mathcal{O} \subset U$ , e  $c \in \mathbb{R}$  tal que  $\mathcal{O} = F^{-1}(c)$ .

Uma curva implícita  $F^{-1}(c)$  é *regular* se  $F$  é diferenciável e satisfaz a condição que em cada ponto  $\mathbf{x} \in F^{-1}(c)$  o gradiente de  $F$  em  $\mathbf{x}$  não se anula.

Um *polinômio de grau  $d$*  definido no plano é uma função  $P_d : \mathbb{R}^2 \rightarrow \mathbb{R}$  especificada pela seguinte expressão:

$$P_d(x, y) = \sum_{0 \leq j+k \leq d} a_{j,k} x^j y^k. \quad (2-1)$$

Dá-se o nome de *curva algébrica de grau  $d$*  ao conjunto  $P_d^{-1}(0)$ .

Como a curva algébrica é o objeto matemático mais utilizado nessa dissertação, é conveniente adotar uma representação adequada para  $P_d$ , que é a mesma adotada por Tasdizen et al. em [17]:

$$P_d(x, y) = \mathbf{v}^t \mathbf{a}, \quad (2-2)$$

onde

$$\mathbf{a} = \left[ a_{0,0} \quad a_{1,0} \quad \dots \quad a_{d,0} \quad a_{d-1,1} \quad \dots \quad a_{0,d} \right]^t$$

e

$$\mathbf{v} = \left[ 1 \quad x \quad \dots \quad x^d \quad x^{d-1}y \quad \dots \quad y^d \right]^t.$$

O vetor  $\mathbf{a} \in \mathbb{R}^l$  representa os coeficientes  $(a_{j,k})_{0 \leq j; 0 \leq k; 0 \leq j+k \leq d}$  e o vetor  $\mathbf{v}$  representa os monômios do polinômio  $P_d$ . Como  $P_d$  possui grau  $d$  é fácil verificar que o número de termos que ele possui (que corresponde à dimensão de  $\mathbf{a}$  ou à dimensão de  $\mathbf{v}$ ) é exatamente  $l = \frac{(d+1)(d+2)}{2}$ .

Coeficientes do Polinômio $P_d$	
Grau $d$ da curva $\mathcal{C}$	Número de coeficientes $l$ do polinômio $P_d$
$d = 2$	$l = \frac{(2+1)(2+2)}{2} = 6$
$d = 3$	$l = \frac{(3+1)(3+2)}{2} = 10$
$d = 4$	$l = \frac{(4+1)(4+2)}{2} = 15$
$d = 5$	$l = \frac{(5+1)(5+2)}{2} = 21$
$d = 6$	$l = \frac{(6+1)(6+2)}{2} = 28$
$d = 7$	$l = \frac{(7+1)(7+2)}{2} = 36$
$d = 8$	$l = \frac{(8+1)(8+2)}{2} = 45$
$d = 9$	$l = \frac{(9+1)(9+2)}{2} = 55$
$d = 10$	$l = \frac{(10+1)(10+2)}{2} = 66$

Tabela 2.1: Tabela do número de coeficientes do polinômio

## 2.2

### Aproximações algébricas por mínimos quadrados

Considere os dados de entrada como sendo um conjunto de pontos  $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_q\} \in \mathbb{R}^2$  amostrados de uma curva planar  $\mathcal{C}$ , onde  $\mathbf{p}_i = (x_i, y_i)$ . Considere também o conjunto de vetores normais unitários correspondentes  $\mathcal{N} = \{\mathbf{n}_1, \mathbf{n}_2, \dots, \mathbf{n}_q\}$ .

Apresenta-se agora diversos métodos de como obter uma curva que aproxima  $\mathcal{C}$ .

#### 2.2.1

##### Minimizando a distância algébrica

Uma maneira simples de se obter uma curva algébrica  $P_d^{-1}(0)$  que aproxima a curva  $\mathcal{C}$  é o de minimizar a distância algébrica sobre o conjunto de pontos dados. Isto é, determinar os coeficientes de  $P_d^{-1}(0)$  tal que a soma da distância algébrica de cada um dos  $q$  pontos à curva, denotada por

$$e_{total} = \sum_{i=1}^q (P_d(x_i, y_i))^2 \quad (2-3)$$

seja a menor possível.

Utilizando a representação vetorial de  $P_d$  em (2-2),  $e_{total}$  pode ser escrito como:

$$e_{total} = \mathbf{a}^t \left( \sum_{i=1}^q \mathbf{v}_i \mathbf{v}_i^t \right) \mathbf{a}, \quad (2-4)$$

onde

$$\mathbf{v}_i = \left[ 1 \quad x_i \quad \dots \quad x_i^d \quad x_i^{d-1} y_i \quad \dots \quad y_i^d \right]^t.$$

Para melhorar a notação, defina as matrizes  $\mathbf{M}$  de tamanho  $l \times q$  e  $\mathbf{S}$  de

tamanho  $l \times l$  da seguinte maneira:

$$\mathbf{M} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_q \end{bmatrix},$$

e

$$\mathbf{S} = \mathbf{M}\mathbf{M}^t = \sum_{i=1}^q \mathbf{v}_i \mathbf{v}_i^t.$$

Se fosse considerado simplesmente o problema de minimização como sendo

$$\min_{\mathbf{a}} \{\mathbf{a}^t \mathbf{S} \mathbf{a}\},$$

o vetor  $\mathbf{a} = \mathbf{0}$  seria a solução trivial. Para evitar isso, deve-se impôr a condição  $\|\mathbf{a}\|^2 = 1$ .

Com o uso do multiplicador de Lagrange  $\lambda$ , o problema de otimização condicionado torna-se:

$$\min_{\mathbf{a}} \{\mathbf{a}^t \mathbf{S} \mathbf{a} + \lambda(\mathbf{a}^t \mathbf{a} - 1)\}. \quad (2-5)$$

A solução desse problema é dada pelo autovetor unitário de  $S$  associado ao seu menor autovalor [18].

Em resumo, o método para a minimização da distância algébrica restrito à condição  $\|\mathbf{a}\|^2 = 1$  consiste simplesmente em encontrar o autovetor de  $\mathbf{S}$  associado ao seu menor autovalor em módulo usando, por exemplo, o método da potência em  $\mathbf{S}^{-1}$  [15]. Esse método será chamado de *classical least square*.

Embora este método seja invariante por transformações afins [18], ele apresenta alguns problemas. Seus resultados são extremamente sensíveis às pequenas perturbações nos dados de entrada. Além disso, a curva algébrica  $P_d^{-1}(0)$  obtida na aproximação não leva em consideração a continuidade do conjunto de dados de entrada, podendo gerar novas componentes conexas ou juntar componentes que eram originalmente separadas.

Uma técnica simples para tornar a aproximação por esse método mais estável é aplicar uma padronização nos conjuntos de dados, que consiste em colocar a origem do domínio no centróide do conjunto de dados e então escalonar os pontos dividindo-os pela média dos autovalores da matriz  $\mathbf{S}$ .

Para maiores detalhes sobre esse método consulte [16, 17, 18].

### 2.2.2

#### Aproximação considerando o gradiente

Para evitar os problemas de inconsistência de continuidade e de sensibilidade à pequenas perturbações do conjunto de dados de entrada, foi proposto um outro método que considera os vetores unitários normais à curva [17]. Esses

vetores são fornecidos como dados de entrada do sistema através do conjunto  $\mathcal{N}$ .

Define-se o conjunto  $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_q\}$  como sendo o conjunto dos vetores unitários tangentes à curva  $\mathcal{C}$ , onde  $\mathbf{t}_i$  é obtido aplicando uma rotação de  $\pi/2$  no sentido anti-horário ao vetor  $\mathbf{n}_i$ .

O método *Gradient One Fitting* a ser discutido foi proposto por Lei et al. em [10]. Ele faz uso do gradiente do polinômio  $P_d$  em  $(x_i, y_i)$ , que é denotado por

$$\nabla P_n(x_i, y_i) = \begin{bmatrix} \frac{\partial P_n}{\partial x}(x_i, y_i) \\ \frac{\partial P_n}{\partial y}(x_i, y_i) \end{bmatrix}, \quad (2-6)$$

Vale lembrar que se o vetor gradiente de  $P_d$  em  $(x_i, y_i)$  não é nulo, então ele é perpendicular ao vetor tangente da curva de nível de  $P_d$  que passa por  $(x_i, y_i)$ .

A proposta do método proposto por Lei et al. é novamente um problema de mínimos quadrados. Ele adiciona dois termos novos à função objetivo e retira a condição  $\|\mathbf{a}\|^2 = 1$ . Esses novos termos são funções do gradiente do polinômio  $P_d$ . O primeiro deles é  $\sum_{i=1}^q (\mathbf{n}_i^t \nabla P_d(x_i, y_i) - 1)^2$ , cujo objetivo é o de ajustar  $P_d$  de tal forma que o seu gradiente em  $(x_i, y_i)$  tenda a ficar alinhado com  $\mathbf{n}_i$ . O segundo termo é  $\sum_{i=1}^q (\mathbf{t}_i^t \nabla P_d(x_i, y_i))^2$ , que tem por objetivo forçar ainda mais o ajuste fazendo com que o vetor gradiente de  $P_d$  em  $(x_i, y_i)$  tenda a ficar perpendicular à  $\mathbf{t}_i$ . A função a ser minimizada fica, portanto, definida da seguinte forma:

$$e_{grad} = \sum_{i=1}^q \{(P_d(x_i, y_i))^2 + \mu(\mathbf{n}_i^t \nabla P_d(x_i, y_i) - 1)^2 + \mu(\mathbf{t}_i^t \nabla P_d(x_i, y_i))^2\} \quad (2-7)$$

onde  $\mu$  é o peso que se dá aos dois novos termos.

Com o objetivo de seguir a representação vetorial para o polinômio  $P_d$ , as seguintes matrizes e vetores são definidas:

– A matriz  $\mathbf{D}_i$  de tamanho  $l \times 2$ :

$$\mathbf{D}_i = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 2x_i & 0 \\ \vdots & \vdots \\ dx_i^{d-1} & 0 \\ (d-1)x_i^{d-2} \cdot y_i & x_i^{d-1} \\ \vdots & \vdots \\ y_i^{d-1} & (d-1)x_i \cdot y_i^{d-2} \\ 0 & d \cdot y_i^{d-1} \end{bmatrix}. \quad (2-8)$$

- O vetor gradiente  $\nabla P_d$ :

$$\nabla P_d = \nabla(\mathbf{v}_i^t \mathbf{a}) = (\mathbf{D}_i)^t \mathbf{a}. \quad (2-9)$$

- A matriz  $\mathbf{S}_N$  de tamanho  $l \times l$ :

$$\mathbf{S}_N = \sum_{i=1}^q \mathbf{D}_i \mathbf{n}_i \mathbf{n}_i^t \mathbf{D}_i^t.$$

- A matriz  $\mathbf{S}_T$  de tamanho  $l \times l$ :

$$\mathbf{S}_T = \sum_{i=1}^q \mathbf{D}_i \mathbf{t}_i \mathbf{t}_i^t \mathbf{D}_i^t.$$

- O vetor  $\mathbf{g}_N$  de tamanho  $l$ :

$$\mathbf{g}_N = \sum_{i=1}^q \mathbf{D}_i \mathbf{n}_i.$$

O problema de otimização para o método é:

$$\min_{\mathbf{a}} \{e_{grad}\} = \min_{\mathbf{a}} \{\mathbf{a}^t \mathbf{S} \mathbf{a} + \mu \mathbf{a}^t \mathbf{S}_N \mathbf{a} + \mu \mathbf{a}^t \mathbf{S}_T \mathbf{a} - 2\mu \mathbf{a}^t \mathbf{g}_N + \mu q\}$$

E a solução desse problema é obtido resolvendo o seguinte sistema de equações lineares:

$$\mathbf{a} = \mu (\mathbf{S} + \mu (\mathbf{S}_N + \mathbf{S}_T))^{-1} \mathbf{g}_N. \quad (2-10)$$

### 2.2.3

#### Técnica para reduzir a instabilidade numérica

Quando a matriz  $\mathbf{S} = (\mathbf{S} + \mu (\mathbf{S}_N + \mathbf{S}_T))$  não tiver posto máximo ou mostrar instabilidade numérica para resolver o sistema (2-10) pode-se usar a técnica denominada *ridge regression*, e será denotada por *RR*. Essa técnica é normalmente usada pelos estatísticos para remover a colinearidade dos dados de entrada. A primeira proposta para obter melhores curvas algébricas usando essa técnica foi feita por Tasdizen et al. em [17].

A técnica *RR* basicamente modifica o problema de otimização adicionando um novo termo:

$$\min_{\mathbf{a}} \{\mathbf{a}^t \mathbf{S} \mathbf{a} + \mu \mathbf{a}^t \mathbf{S}_N \mathbf{a} + \mu \mathbf{a}^t \mathbf{S}_T \mathbf{a} - 2\mu \mathbf{a}^t \mathbf{g}_N + \mu q + \kappa \mathbf{a}^t \mathbf{\Delta} \mathbf{a}\},$$

Onde  $\mathbf{\Delta}$  é uma matriz diagonal de tamanho  $l \times l$  e a constante real  $\kappa$  determina o peso que se dá ao novo termo na aproximação.

A solução para esse problema é obtido da seguinte forma:

$$\mathbf{a} = \mu(\mathbf{SS} + \kappa\mathbf{\Delta})^{-1}\mathbf{g}_N \quad (2-11)$$

Tasdizen et al. em [17] sugerem a seguinte escolha para matriz  $\mathbf{\Delta}$ .

$$\Delta_{vv} = \frac{i!j!}{(i+j)!} \sum_{k,l \geq 0; k+l=i+j} \frac{(k+l)!}{k!l!} \sum_{m=1}^q x_m^{2k} y_m^{2l}, \quad (2-12)$$

para  $i, j \geq 0$ ;  $i+j \leq d$ , onde  $v = j + \frac{(i+j+1)(i+j)}{2}$ .

Essa sugestão para  $\mathbf{\Delta}$  preserva a invariância por rotação do método descrito na seção 2.2.2 [17].

A seguir, a figura 2.1 [17] mostra uma comparação entre os três métodos apresentados.

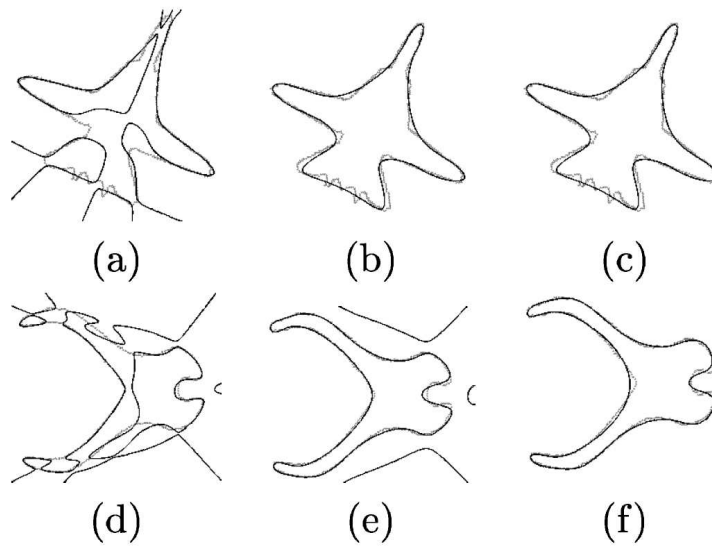


Figura 2.1: As figuras (a) e (d) ilustram aproximações obtidas usando o método *classical least squares*. Já nas figuras (b) e (e) foi utilizado o método *gradient one fitting*, e nas figuras (c) e (f) o método *ridge regression*. O grau do polinômio foi escolhido como sendo 6 para o alicate e 8 para o avião.

## 2.3

### Partição da Unidade

A *partição da unidade* é uma ferramenta matemática muito útil para combinar aproximações locais a fim de definir uma aproximação global. Importantes propriedades como o erro máximo global e a ordem de convergência podem ser herdadas do comportamento local.

A idéia básica da construção de uma aproximação global por partição da unidade é a seguinte:

1. dividir o domínio de interesse em diversos pedaços,

2. obter uma aproximação local para o subconjunto de dados pertencentes ao próprio pedaço,
3. obter uma aproximação global fazendo uma combinação ponderada das soluções locais através de funções suaves não negativas que correspondem aos pesos. Em cada ponto do domínio, a soma dessas funções peso deve ser um.

Mais precisamente, considere um domínio limitado  $\Omega \subset \mathbb{R}^2$  e denote por  $\{\varphi_i\}$  o conjunto de funções não negativas com suporte compacto tal que:

$$\sum_i \varphi_i(x, y) \equiv 1 \text{ para todo ponto } (x, y) \in \Omega.$$

Chame de  $\mathcal{F}_i$  o conjunto de funções com suporte compacto definidas em  $\text{supp}(\varphi_i)$ . Cada função pertencente a  $\mathcal{F}_i$  estaria representando uma aproximação local para os pontos do conjunto de entrada  $\mathcal{P}$  que pertencem à  $\text{supp}(\varphi_i)$ .

Uma aproximação global para a função  $f : \Omega \rightarrow \mathbb{R}^2$  poderia ser obtida da seguinte maneira:

$$f(x, y) \approx \sum \varphi_i(x, y) f_i(x, y). \quad (2-13)$$

onde  $f_i \in \mathcal{F}_i$ .

Considere  $\{w_i\}$  um conjunto de funções não-negativas com suporte compacto tal que:

$$\Omega \subset \bigcup_i \text{supp}(w_i).$$

As funções partições da unidade  $\varphi_i$  podem ser geradas da seguinte forma:

$$\varphi_i(x, y) = \frac{w_i(x, y)}{\sum_{j=1}^n w_j(x, y)} \quad (2-14)$$

A idéia por trás de uma aproximação construída através da partição da unidade pode ser resumida pelas equações (2-13) e (2-14). Essas equações formam a base do algoritmo MPU proposto por Ohtake et al. em [13].

## 2.4

### O método MPU

O método chamado de *Multilevel Partion of Unity* (MPU) foi proposto por Ohtake et al. em [13] originalmente para gerar uma superfície implícita em  $\mathbb{R}^3$ . Nessa dissertação restringimos o seu uso para curvas implícitas em  $\mathbb{R}^2$ .

Como seu próprio nome diz, o MPU usa a partição da unidade para obter aproximações globais a partir de aproximações locais usando uma estrutura

hierárquica para a subdivisão do domínio, que são feitas recursivamente através de uma *Quad-Tree*.

A *Quad-Tree* é um tipo especial de árvore onde todas as células ou são células folha ou têm quatro células filhas. Tendo como utilização principal a representação de uma decomposição espacial adaptativa do domínio através de um processo recursivo.

A figura 2.2 ilustra um exemplo de uma *Quad-Tree* adaptada aos pontos de entrada.

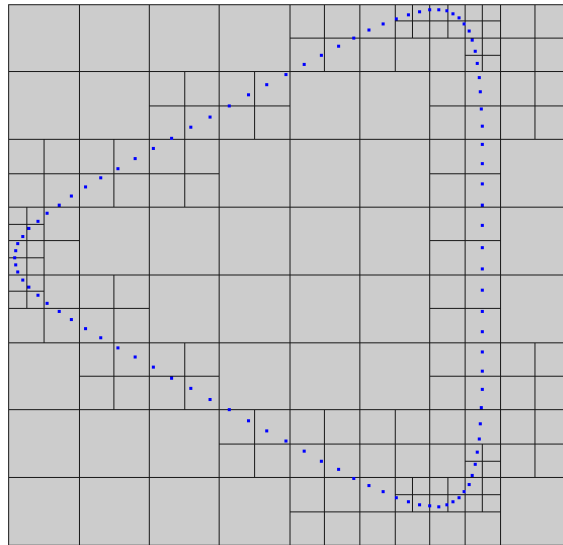


Figura 2.2: Exemplo da construção de uma *Quad-Tree* gerando uma subdivisão do domínio adaptada aos pontos.

Segue uma descrição concisa de como o método MPU constrói a função implícita que aproxima globalmente os pontos.

Inicialmente ele translada os pontos de  $\mathcal{P}$  tornando o seu centro de massa a origem do sistema de coordenadas. Depois ele escalona os pontos de tal forma que o quadrado  $\Xi = [-1, 1] \times [-1, 1]$  contenha todos eles. Por um abuso de notação, continua-se dando o nome de  $\mathcal{P}$  ao conjunto de pontos de entrada após essas duas transformações.

O método cria uma *Quad-Tree* usando um procedimento recursivo controlado pelo erro da aproximação local. O critério de refinamento consiste em identificar localmente o erro cometido pela aproximação e caso ele seja maior que uma determinada tolerância ele subdivide o quadrado em quatro e recursivamente repete o teste para cada um dos seus quadrantes.

Portanto, para cada nó  $i$  da *Quad-Tree* existe uma função peso usada na partição da unidade com suporte compacto definido como sendo um círculo de raio  $r_i$  centrado no centro geométrico do nó. Esse raio é escolhido como sendo

proporcional ao tamanho da diagonal do quadrado do nó  $i$ , denotado por  $d_i$ . Geralmente se faz  $r_i = 0.75d_i$ .

A figura 2.3 ilustra os suportes compactos associados a cada nó da estrutura *Quad-Tree*.

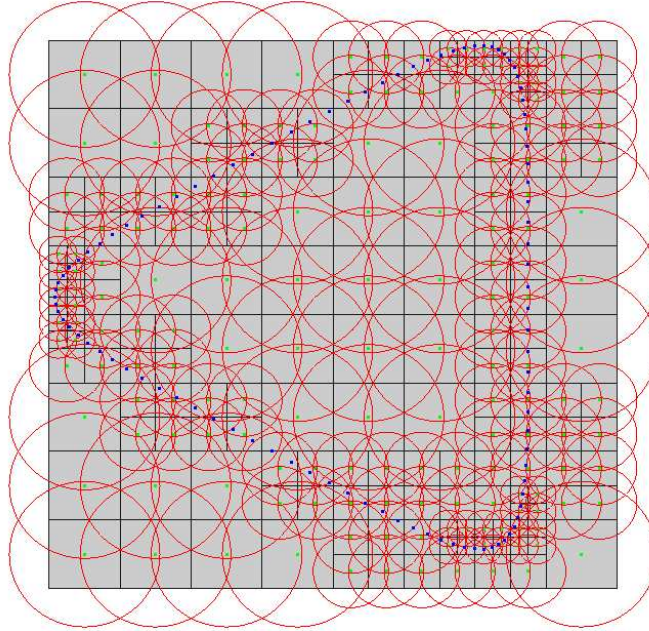


Figura 2.3: Exemplo da região de suporte para cada nó da *Quad-Tree*.

A aproximação local é feita usando uma quádrlica e ela só é calculada nos nós que possuem pontos de  $\mathcal{P}$ . A função de partição da unidade para cada nó da *Quad-Tree* é uma B-Spline quadrática  $b(t)$ :

$$\varphi_i(x, y) = b\left(\frac{3 |x - c_i|}{2r_i}\right) \quad (2-15)$$

onde  $c_i$  é centro do quadrado correspondente a um nó da *Quad-Tree*.

Para calcular as quádrlicas em cada nó, considera-se os pontos de  $\mathcal{P}$  que estão na sua região de suporte correspondente. Às vezes (especialmente se a densidade de  $\mathcal{P}$  não é uniforme) o círculo de raio  $r_i$  de um nó  $N_i$  não contém o número de pontos suficiente para uma estimação robusta da quádrlica que aproxima os pontos. Se o número de pontos for menor que o número mínimo de pontos suficiente para resolver o problema de mínimos quadrados para a determinação da quádrlica, deve-se aumentar o raio do círculo do suporte até que se obtenha a garantia de existir uma solução (a matriz a ser invertida possuir posto máximo), como ilustra a figura 2.4[13].

Ohtake et al. utilizaram uma função objetivo que é uma média ponderada da distância algébrica de cada ponto à curva, onde a ponderação é feita pela própria função peso  $\varphi_i$ . Mais informações podem ser encontradas em [13].

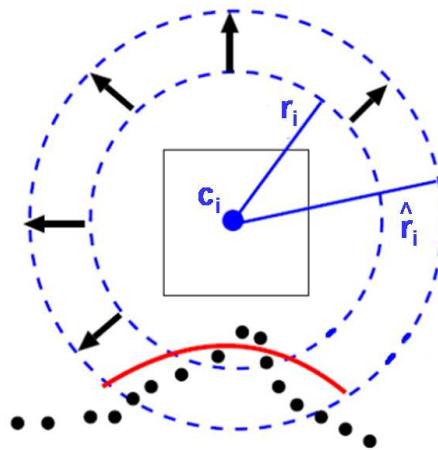


Figura 2.4: Aumentando o raio do círculo do suporte para obter uma garantia de existir solução para o cálculo das quádricas.

As figuras (2.5) e (2.6)[13] exemplificam superfícies implícitas no  $\mathbb{R}^3$  obtidas pelo método MPU.

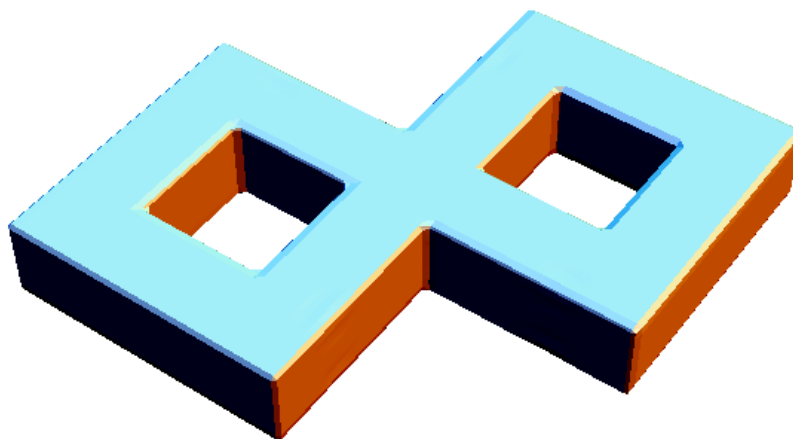


Figura 2.5: Toro: Figura gerada pelo software MPU.

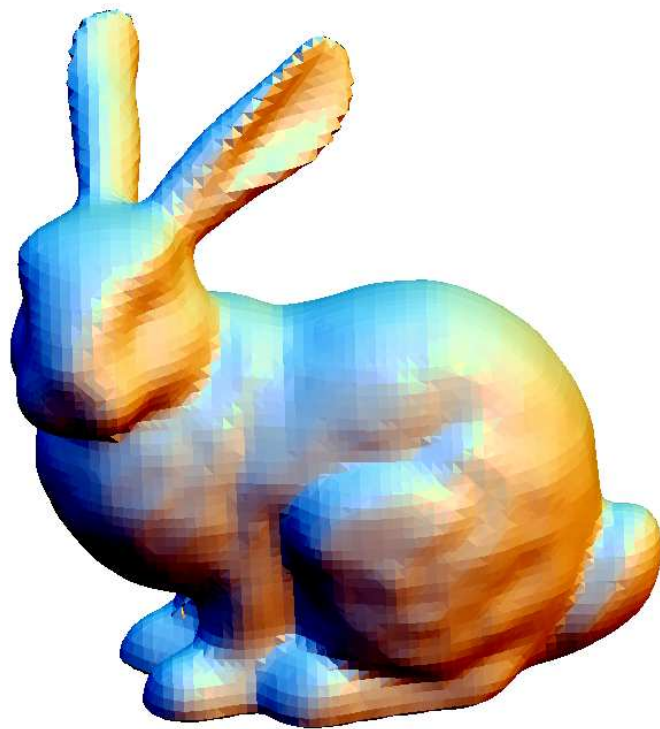


Figura 2.6: Bunny: Figura gerada pelo software MPU.

## 3

### O novo método implícito

Este capítulo introduz o novo método para determinar uma função implícita que utiliza partição da unidade. Ele inicia com uma breve introdução ao método, e depois descreve detalhes de sua implementação.

#### 3.1

##### Descrição

O novo método é de fato uma concatenação de dois importantes trabalhos: o do método MPU [13] e o proposto por Tasdizen et al. em [17], que foram explicados, respectivamente, nas seções 2.4 e 2.2.3.

O método considera como conjunto de dados de entrada as seguintes informações:

- o conjunto de pontos  $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_q\} \in \mathbb{R}^2$  amostrados de uma curva planar  $\mathcal{C}$ , onde  $\mathbf{p}_i = (x_i, y_i)$ .
- o correspondente conjunto dos vetores normais unitários  $\mathcal{N} = \{\mathbf{n}_1, \mathbf{n}_2, \dots, \mathbf{n}_q\}$ .

Já o conjunto  $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_q\}$  dos vetores tangentes à curva é obtido através do conjunto  $\mathcal{N}$ . O vetor  $\mathbf{t}_i$  é obtido aplicando à  $\mathbf{n}_i$  uma rotação de  $\pi/2$  no sentido anti-horário.

Assume-se aqui que os pontos de  $\mathcal{P}$  já foram transladados de tal forma que o seu centro de massa seja a origem do sistema de coordenadas. E que eles também foram escalonados de tal forma que todos estejam no quadrado  $\Xi = [-1, 1] \times [-1, 1]$ .

Esse quadrado  $\Xi$  de centro na origem e lado 2 serve como base para o processo de subdivisão guiado por uma estrutura hierárquica de subdivisão espacial do tipo *Quad-Tree*.

A *Quad-Tree* é criada utilizando um processo recursivo controlado pelo erro da aproximação local e pelo nível máximo denotado por  $l_{max}$  que é pré-determinado.

Então, cada nó  $i$  da *Quad-Tree* tem associado a ele:

- Uma função peso que é utilizada na partição da unidade  $\varphi_i$  com suporte compacto. Esse suporte compacto é estabelecido como sendo um círculo de raio  $r_i$  centrado no ponto central do nó. Onde o valor de  $r_i$  é definido como sendo  $\alpha$  vezes o tamanho da digonal do quadrado correspondente. Geralmente, usamos  $\alpha = 0.75$ .

Assim como no método MPU em [13] a função partição da unidade  $\varphi_i$  para cada quadrado  $\Xi_i$  é dada pela equação:

$$\varphi_i(x, y) = \frac{w_i(x, y)}{\sum_{j=1}^{n_f} w_j(x, y)}, \quad (3-1)$$

onde  $n_f$  é o número de nós folhas na *Quad-Tree* e  $w_i$  é uma B-Spline quadrática definida através da distância do ponto  $(x, y)$  ao centro do nó  $i$ . Se a distância de  $(x, y)$  ao centro do nó for maior do que  $r_i$ , então  $w_i(x, y) = 0$ .

- Um polinômio de grau fixo  $d$  que define a aproximação local  $F_i$ . Para determinar os coeficientes de tal polinômio se utiliza os pontos de  $\mathcal{P}$  que estão no suporte compacto da função  $\varphi_i$ . Esse subconjunto será denotado por  $\mathcal{P}_i$ . Considere que  $\mathcal{P}_i$  possui  $q_i$  pontos e que esses pontos estejam indexados de 1 à  $q_i$ . A aproximação local é feita utilizando os métodos apresentado na seção 2.2. Assim, a função objetivo a ser minimizada no nó  $i$  é:

$$\{\mathbf{a}^t \mathbf{S}_i \mathbf{a} + \mu \mathbf{a}^t \mathbf{S}_{\mathbf{N},i} \mathbf{a} + \mu \mathbf{a}^t \mathbf{S}_{\mathbf{T},i} \mathbf{a} - 2\mu \mathbf{a}^t \mathbf{g}_{\mathbf{N},i} + \mu q_i + \kappa \mathbf{a}^t \mathbf{\Delta}_i \mathbf{a}\}.$$

onde

- A matriz  $\mathbf{S}_i$  de tamanho  $l \times l$  é:

$$\mathbf{S}_i = \sum_{j=1}^{q_i} \mathbf{v}_j \mathbf{v}_j^t.$$

- A matriz  $\mathbf{S}_{\mathbf{N},i}$  de tamanho  $l \times l$  é:

$$\mathbf{S}_{\mathbf{N},i} = \sum_{j=1}^{q_i} \mathbf{D}_j \mathbf{n}_j \mathbf{n}_j^t \mathbf{D}_j^t.$$

- A matriz  $\mathbf{S}_{\mathbf{T},i}$  de tamanho  $l \times l$  é:

$$\mathbf{S}_{\mathbf{T},i} = \sum_{j=1}^{q_i} \mathbf{D}_j \mathbf{t}_j \mathbf{t}_j^t \mathbf{D}_j^t.$$

– O vetor  $\mathbf{g}_{N,i}$  de tamanho  $l$  é:

$$\mathbf{g}_{N,i} = \sum_{j=1}^q \mathbf{D}_j \mathbf{n}_j.$$

– A matriz diagonal  $\Delta_i$  é obtida preenchendo os elementos da sua diagonal da seguinte maneira:

$$(\Delta_i)_{vv} = \frac{h!j!}{(h+j)!} \sum_{k,l \geq 0; k+l=h+j} \frac{(k+l)!}{k!l!} \sum_{m=1}^{q_i} x_m^{2k} y_m^{2l},$$

para  $h, j \geq 0$ ;  $h+j \leq d$ , onde  $v = j + \frac{(h+j+1)(h+j)}{2}$ .

A solução desse problema é obtida resolvendo o seguinte sistema de equações lineares:

$$(\mathbf{S}_i + \mu \mathbf{S}_{N,i} + \mu \mathbf{S}_{T,i} + \kappa \Delta_i) \mathbf{a} = \mu \mathbf{g}_N. \quad (3-2)$$

O erro da aproximação local no nó  $i$  é considerado como sendo a média do quadrado das distâncias algébricas dos pontos de  $\mathcal{P}_i$ :

$$e_i = \frac{1}{q_i} \{\mathbf{a}^t \mathbf{S}_i \mathbf{a}\}.$$

A condição que determina se um nó  $i$  na *Quad-Tree* no nível  $l_i$  deve ou não ser refinado é um teste que verifica se o erro  $e_i$  calculado para os pontos utilizados na aproximação local daquele nó é maior que uma tolerância  $\varepsilon$  dada, e se a *Quad-Tree* não atingiu o seu nível máximo (i.e.,  $l_i < l_{max}$ ).

Finalmente, para avaliar a função implícita  $F$  que aproxima  $\mathcal{C}$  basta combinar as aproximações locais  $F_i$  juntamente utilizando as funções partição da unidade  $\varphi_i$ .

$$F(x, y) = \frac{\sum_{i=0}^{n_f} w_i(x, y) F_i(x, y)}{\sum_{j=1}^{n_f} w_j(x, y)}. \quad (3-3)$$

Segue uma descrição um pouco mais detalhada dos algoritmos que foram implementados.

## 3.2 Implementação

Esta seção apresenta os algoritmos utilizados no desenvolvimento computacional deste trabalho. Primeiramente, serão descritas as estruturas de dados utilizadas e depois os algoritmos de construção seguidos dos algoritmos de avaliação da função implícita.

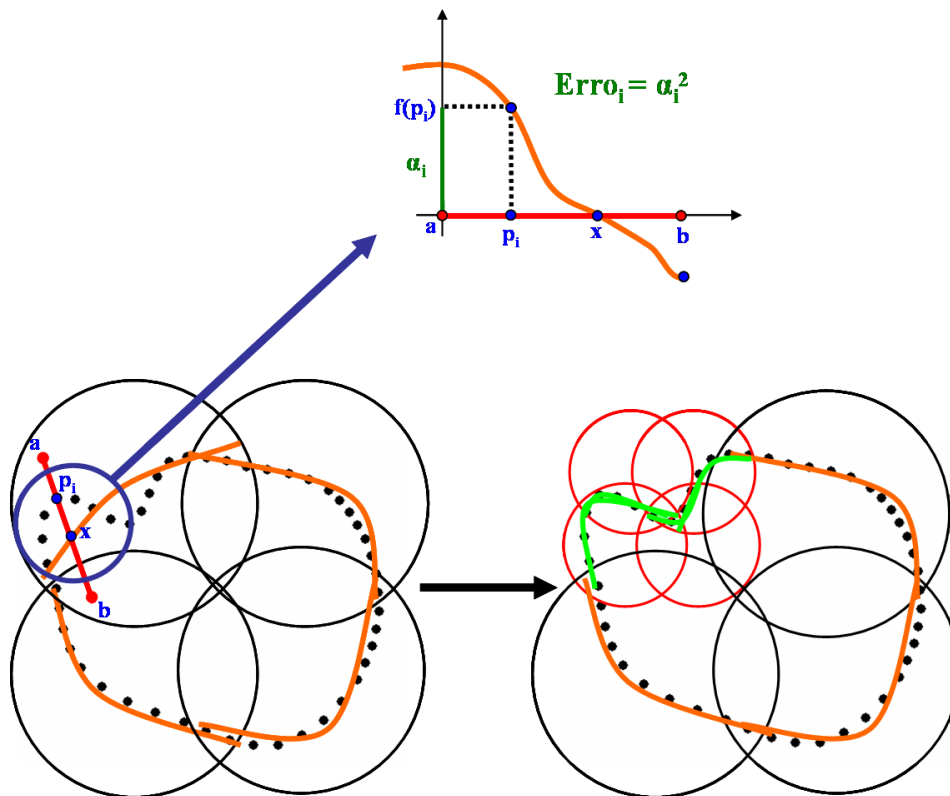


Figura 3.1: Refinamento local que permite uma melhor aproximação global.

### 3.2.1

#### A estrutura de dados

A estrutura de dados desse trabalho está organizada através das classes `Quad_Tree` e `Curva_Implicita`.

#### A classe `Quad_Tree`

Um objeto da classe `Quad_Tree` representa um nó na árvore *Quad-Tree*. Essa classe possui os seguintes membros:

- `double * a` : representa o vetor de coeficientes do polinômio de grau  $d$ .
- `double c[2]` : vetor contendo as duas coordenadas do centro do nó da árvore.
- `double r` : o raio que define o suporte compacto do nó da árvore.
- `int level` : o nível da árvore no qual se encontra o nó.
- `double size` : o tamanho do lado do quadrado correspondente ao nó.
- `int * I` : um vetor de inteiros que representa o conjunto dos índices  $i$  dos pontos que se encontram dentro do círculo de raio  $r$  centrado em  $c$ .
- `Quad_Tree * son[4]` : um vetor de ponteiros que apontam para os quatro filhos do nó.

- `Quad_Tree * father` : o ponteiro que aponta para o seu nó pai na árvore.

A figura 3.2 esquematiza a organização da classe `Quad_Tree`.

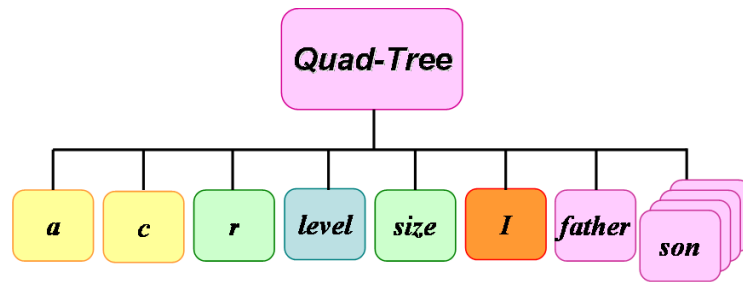


Figura 3.2: Organização da classe `Quad_Tree`.

Ela possui somente um construtor `default Quad_Tree::Quad_Tree()` que simplesmente inicia os membros da classe, atribuindo `NULL` aos ponteiros e zero às variáveis `double` e `int`. Seu destrutor `Quad_Tree::~~Quad_Tree()`, destrói a árvore percorrendo-a recursivamente, liberando o espaço de memória alocado para cada um de seus filhos e depois liberando o seu próprio espaço alocado.

### A classe `Curva_Implicita`

Os membros da classe `Curva_Implicita` são:

- `int d` : o grau do polinômio utilizado na aproximação local.
- `int l_max` : o nível máximo da `Quad_Tree`.
- `double alpha` : multiplicador para o raio da região de suporte.
- `double mu` : constante do método *gradient one fitting*.
- `double kappa` : constante do método *rigde regression*.
- `double epsilon` : tolerância para o erro local da aproximação.
- `Quad_Tree * root` : ponteiro para a raiz da árvore `Quad_Tree`.
- `double ** P` : corresponde ao conjunto de pontos amostrados de uma curva  $\mathcal{C}$ . Sua representação é feita através de uma matriz  $q \times 2$  de reais, onde  $\mathbf{p}_i = (\mathcal{P}[i][0], \mathcal{P}[i][1])$ .
- `double ** N` : corresponde ao conjunto de vetores normais à curva associados aos pontos de  $\mathcal{P}$ . Sua representação também é feita através de uma matriz  $q \times 2$  de reais, onde  $\mathbf{n}_i = (\mathcal{N}[i][0], \mathcal{N}[i][1])$ . Vale observar que o vetor tangente no ponto  $\mathbf{p}_i$  é  $\mathbf{t}_i = (-\mathcal{N}[i][1], \mathcal{N}[i][0])$ .
- `int q` : número de pontos em  $\mathcal{P}$ .

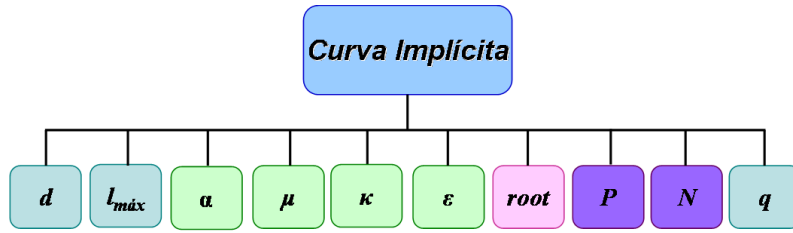


Figura 3.3: Classe Curva\_Implícita.

A figura 3.3 ilustra a organização dos membros dessa classe.

O construtor *default* da classe *Curva\_Implícita* possui os seguintes parâmetros para iniciação das variáveis:  $d$ ,  $l_{max}$ ,  $\mu$ ,  $\kappa$ ,  $\varepsilon$ ,  $\alpha$ ,  $q$ ,  $\mathcal{P}$  e  $\mathcal{N}$ . Os valores sugeridos para os parâmetros do método são:  $d = 2$ ,  $l_{max} = 5$ ,  $\mu = 0.125$ ,  $\kappa = 0.001$ ,  $\varepsilon = 0.1$ , e  $\alpha = 0.75$ . A árvore *Quad-Tree* será criada atribuindo ao termo *root* no construtor o valor retornado da função *Buildtree* que é explicada a seguir. O destrutor dessa classe libera o espaço alocado pela *Quad-Tree* e pelos vetores  $\mathcal{P}$  e  $\mathcal{N}$ .

### 3.2.2

#### A obtenção da função implícita

A função implícita global é determinada através da criação da árvore *Quad-Tree*. O procedimento recursivo *Buildtree* que a constrói é descrito no algoritmo 1. Os parâmetros desse procedimento são:

- `Quad_Tree * father`: ponteiro para o nó pai do nó a ser criado na estrutura;
- `int level`: nível no qual o nó será criado;
- `int l_max`: nível máximo para a árvore;
- `int d`: grau do polinômio;
- `double  $\alpha, \mu, \kappa, \varepsilon$` : constantes do método.
- `double **  $\mathcal{P}, \mathcal{N}$` : os conjuntos de dados de entrada;
- `int q`: número de pontos em  $\mathcal{P}$  e  $\mathcal{N}$ ;
- `double size`: lado do quadrado do nó a ser criado;
- `double * c`: centro geométrico do nó a ser criado;

Para construir a *Quad-Tree* um objeto instânciado da classe *Curva\_Implícita* chama a rotina *Buildtree* utilizando o seguinte comando:

```
root = Buildtree(NULL, 1, l_max, d,  $\alpha, \mu, \kappa, \varepsilon, \mathcal{P}, \mathcal{N}, q, 2, \mathbf{0}$ );
```

---

**Algorithm 1** `Quad_Tree *Quad_Tree::Buildtree(father, level, lmax, d, α, μ, κ, ε, P, N, q, size, c)`


---

```

1: if ( $l_i > l_{max}$ ) then
2:   return NULL;
3: end if
4: Cria nó  $N$ ;
5:  $N \rightarrow father = father$ ;
6:  $N \rightarrow center = \mathbf{c}$ ;
7:  $N \rightarrow size = size$ ;
8:  $N \rightarrow level = level$ ;
9:  $N \rightarrow radius = \alpha \times \sqrt{2} \times size$ ;
10:  $\mathcal{I} = \text{Subset}(\mathcal{P}, center, radius)$ ;
11: if ( $SizeOf(\mathcal{I}) \geq \frac{(d+1)(d+2)}{2}$ ) then
12:    $N \rightarrow \mathbf{a} = \text{Fit}(d, \mu, \kappa, \mathcal{I}, \mathcal{P}, \mathcal{N}, q)$ ;
13:    $erro \leftarrow \text{FittingError}(\mathbf{a}, \mathcal{I})$ ;
14:   if ( $erro > \varepsilon$ ) then
15:     for ( $i = 0; i < 4; i++$ ) do
16:       if ( $(i \& 1) == 0$ ) then
17:          $\mathbf{c}_i[0] = \mathbf{c} - size/2$ ;
18:       else
19:          $\mathbf{c}_i[0] = \mathbf{c} + size/2$ ;
20:       end if {para obter coordenada x do centro do filho}
21:       if ( $(i \& 2) == 0$ ) then
22:          $\mathbf{c}_i[1] = \mathbf{c} - size/2$ ;
23:       else
24:          $\mathbf{c}_i[1] = \mathbf{c} + size/2$ ;
25:       end if {para obter coordenada y do centro do filho}
26:        $N \rightarrow son[i] = \text{Buildtree}(N, (level + 1), l_{max}, d, \mu, \kappa, \varepsilon,$ 
          $\mathcal{P}, \mathcal{N}, q, \alpha, size/2, \mathbf{c}_i)$ ;
27:     end for
28:   end if
29: else
30:    $N \rightarrow \mathbf{a} = father \rightarrow \mathbf{a}$  ; {para copiar os coeficientes do polinômio do
     father para  $\mathbf{a}$ }
31: end if
32: return  $N$ ;

```

---

Onde:

- A função `Subset` retorna no vetor  $\mathcal{I}$  os índices dos pontos de  $\mathcal{P}$  que estão dentro do círculo de raio  $r$  centrado em  $\mathbf{c}$ .
- A função `Fit` calcula a aproximação local determinando os coeficientes do polinômio de grau  $d$  para serem armazenados em  $\mathbf{a}$ . Isso é feito resolvendo o sistema descrito na equação (3-2).
- A função `FittingError` calcula o erro da aproximação local no nó  $i$

usando a fórmula

$$e_i = \{\mathbf{a}^t \mathbf{S}_i \mathbf{a}\}.$$

### 3.2.3

#### A avaliação da função implícita

A classe `Curva_Implicita` possui um função que avalia para um ponto  $(x, y)$  dado qual é o valor da função implícita  $F$  cuja curva de nível 0 aproxima a curva  $\mathcal{C}$ . A declaração dessa função que é membro da classe `Curva_Implicita` é a seguinte: `double Evaluate(double x, double y);`

A implementação dessa função está no algoritmo 2.

---

**Algorithm 2** `double Curva_Implicita::Evaluate(x, y)`

---

```

1: if (root ≠ NULL) then
2:   return root → Evaluate(x, y, d, tw) / tw;
3: else
4:   return -1.0;
5: end if

```

---

É fácil observar que essa função chama a partir da raiz da árvore uma outra função da classe `Quad_Tree` com o mesmo nome, porém com dois parâmetros a mais que são: o grau do polinômio e a soma total das funções peso. Esse último é depois utilizado para dividir o resultado retornado.

A função membro `Evaluate` da classe `Quad_Tree` é declarada da seguinte forma:

```
double Evaluate(double x, double y, int d, double &tw);
```

Ela retorna o numerador da equação (3-3) que corresponde a:

$$\sum_{i=0}^{n_f} w_i(x, y) F_i(x, y),$$

e em  $tw$  é retornado o valor do denominador da mesma equação:

$$tw = \sum_{j=1}^{n_f} w_j(x, y).$$

Vale lembrar que  $n_f$  é o número de nós da *Quad-Tree* que são folhas.

Consequentemente, o valor retornado pela função `Evaluate` da classe `Curva_Implicita` é

$$\text{Evaluate}(x, y) = \frac{\sum_{i=0}^{n_f} w_i(x, y) F_i(x, y)}{\sum_{j=1}^{n_f} w_j(x, y)}.$$

E o algoritmo 3 é a implementação da função `Evaluate` da classe `Quad_Tree`.

---

**Algorithm 3** `double Quad_Tree::Evaluate( $x, y, d, tw$ )`

---

```

1: if ( $son[0] \neq NULL$ ) then
2:   for ( $sum = 0.0, i = 0; i < 4; i++$ ) do
3:      $sum+ = son[i] \rightarrow Evaluate(x, y, d, tw)$ ;
4:   end for
5:   return  $sum$ ;
6: else
7:    $w \leftarrow EvalWeight(x, y)$ ;
8:   if ( $w \neq 0.0$ ) then
9:      $t \leftarrow Function(x, y, \mathbf{a}, d)$ ;
10:     $tw+ = w$ ;
11:    return  $w \times t$ ;
12:   else
13:     return  $0.0$ ;
14:   end if
15: end if

```

---

Onde:

- A função `EvalWeight` retorna o valor da função peso  $w_i(x, y)$  no ponto  $(x, y)$ .
- A função `Function` calcula o valor no ponto  $(x, y)$  do polinômio de grau  $d$  com o vetor de coeficientes  $\mathbf{a}$ .

## 4

### Resultados

Nesse capítulo é feita uma comparação entre os seguintes métodos:

1. *gradient one fitting*.
2. *gradient one fitting* com *ridge regression*.
3. O novo método usando *gradient one fitting*.
4. O novo método usando *gradient one fitting* com *ridge regression*.

Essa numeração para os métodos será utilizada nas legendas das figuras a seguir.

Primeiramente serão mostrados os resultados para curvas suaves. Depois foram escolhidas curvas com singularidades, para exemplificar o desempenho do método mesmo para esses casos, onde a aproximação não deveria funcionar. Finalmente, são exemplificados resultados do método considerando curvas incompletas (faltando alguns pontos).

## 4.1 Curvas Suaves

O exemplo a ser mostrado considera pontos em duas componentes conexas.

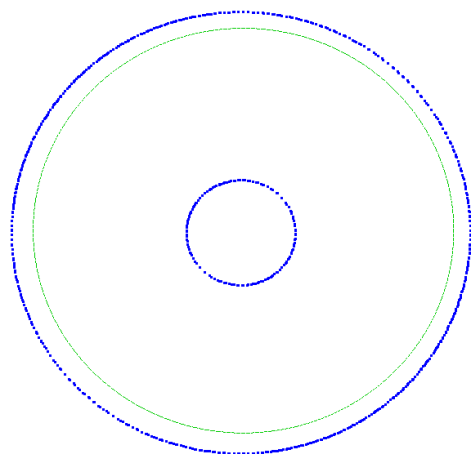
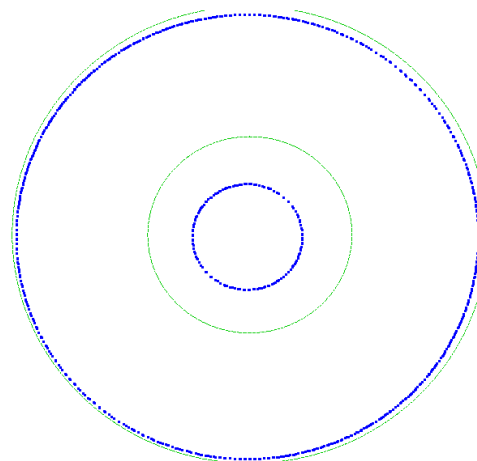
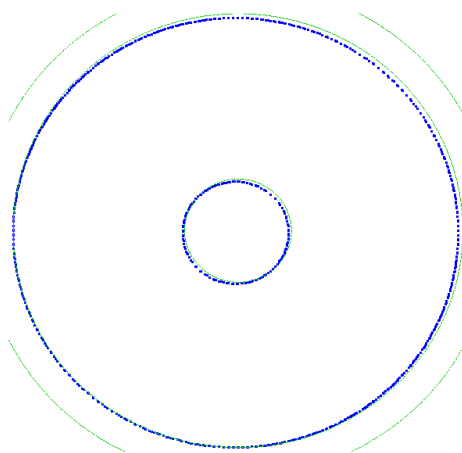
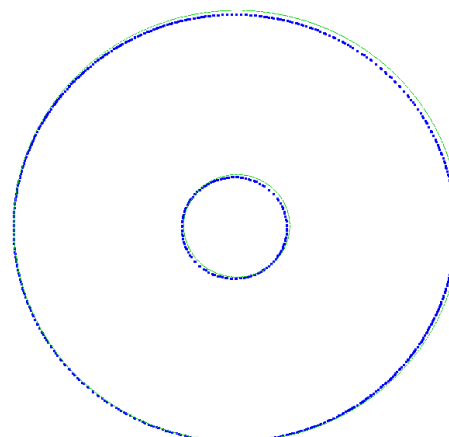
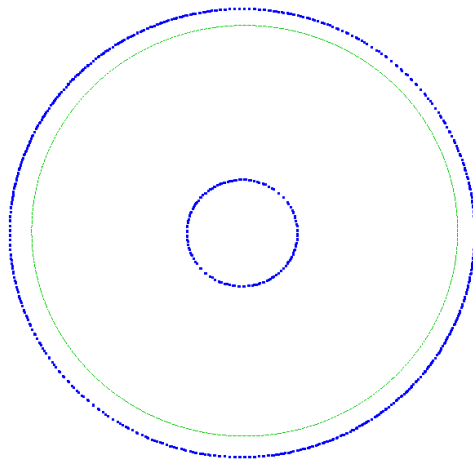
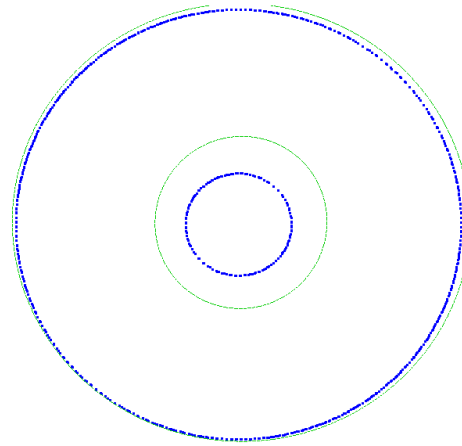
4.1(a):  $d = 2$ 4.1(b):  $d = 4$ 4.1(c):  $d = 6$ 4.1(d):  $d = 8$ 

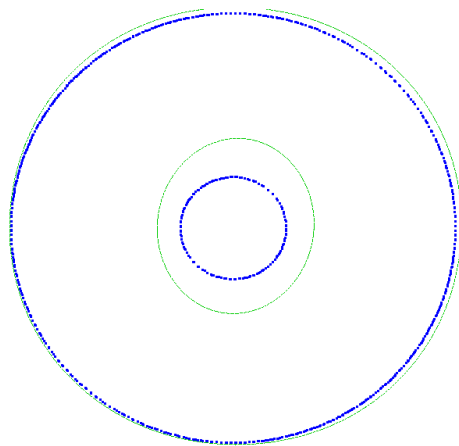
Figura 4.1: Dois círculos: método 1 usando um polinômio de grau  $d$  com  $\mu = 0.125$ .



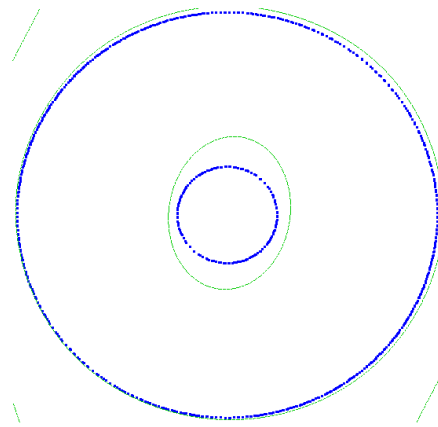
4.2(a):  $d = 2$



4.2(b):  $d = 4$



4.2(c):  $d = 6$



4.2(d):  $d = 8$

Figura 4.2: Dois círculos: método 2 usando um polinômio de grau  $d$  com  $\mu = 0.125$  e  $\kappa = 0.001$ .

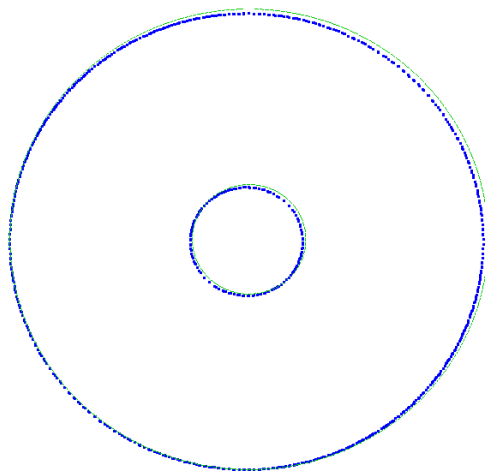
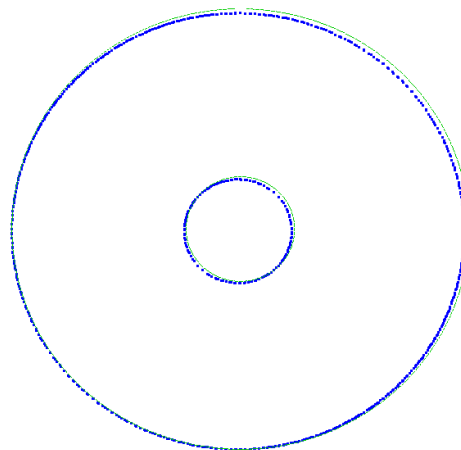
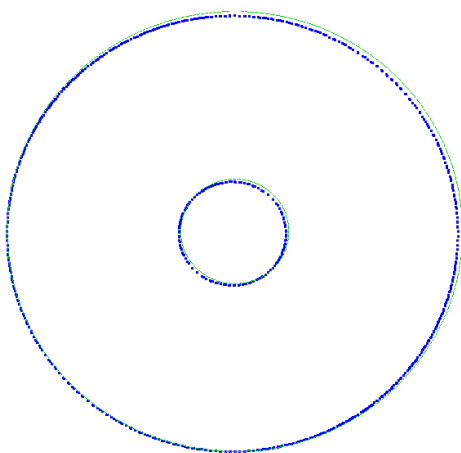
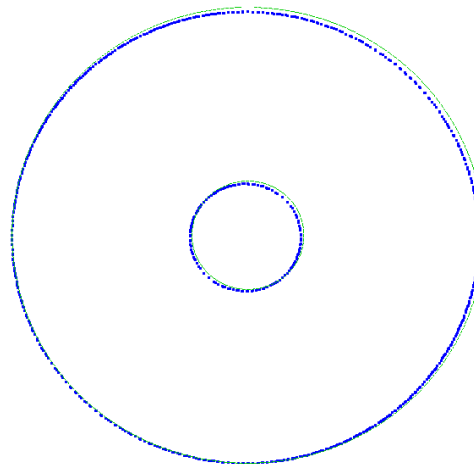
4.3(a):  $d = 2$  ;  $l_{max} = 2$ 4.3(b):  $d = 2$  ;  $l_{max} = 3$ 4.3(c):  $d = 2$  ;  $l_{max} = 4$ 4.3(d):  $d = 2$  ;  $l_{max} = 5$ 

Figura 4.3: Dois círculos: método 3 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .

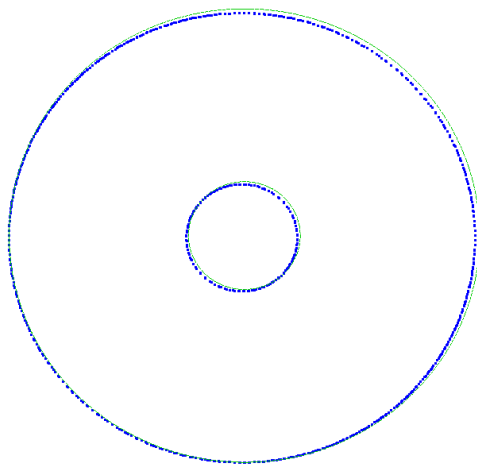
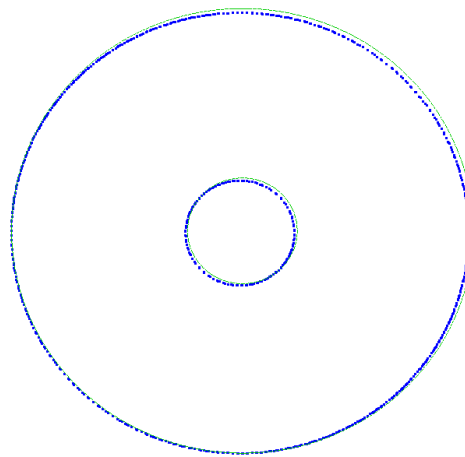
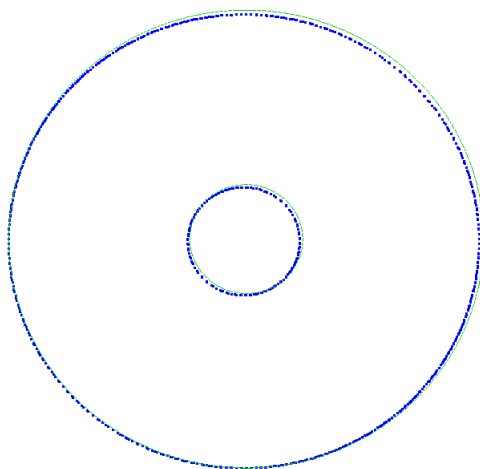
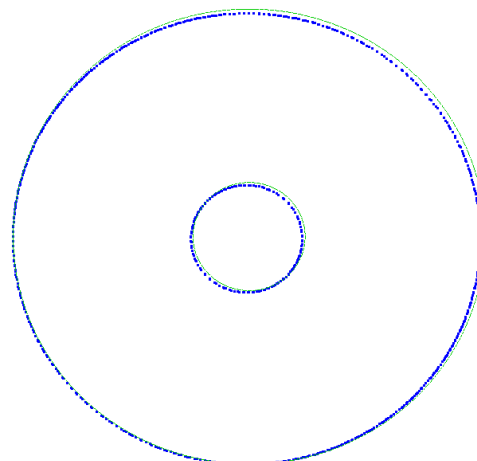
4.4(a):  $d = 2$  ;  $l_{max} = 2$ 4.4(b):  $d = 2$  ;  $l_{max} = 3$ 4.4(c):  $d = 2$  ;  $l_{max} = 4$ 4.4(d):  $d = 2$  ;  $l_{max} = 5$ 

Figura 4.4: Dois círculos: método 4 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$ ,  $\kappa = 0.001$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .

Este outro exemplo a ser mostrado ilustra os pontos e suas respectivas normais, além da aproximação feita pelos métodos.

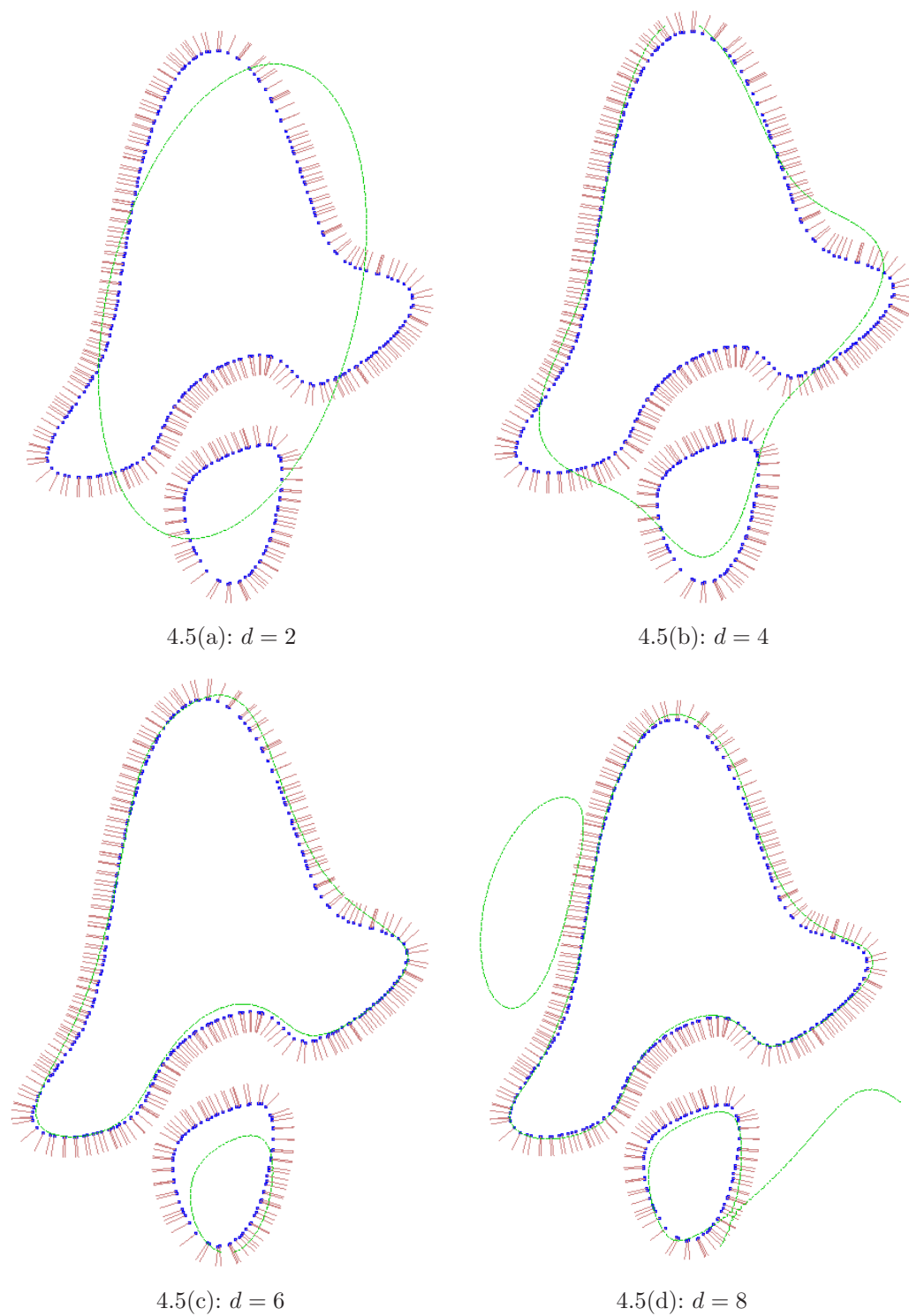
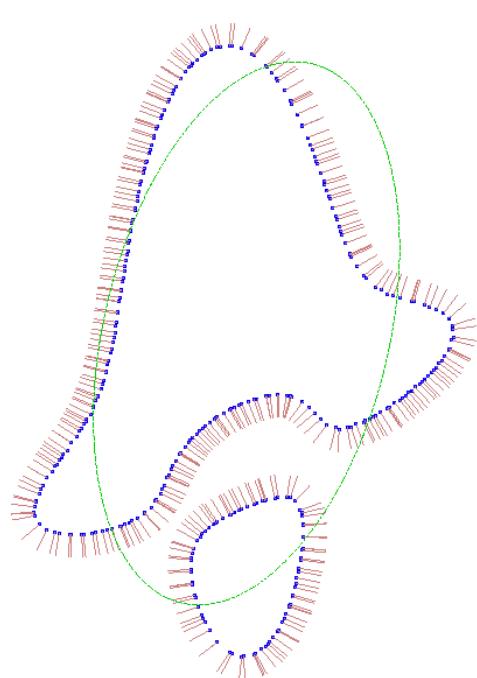
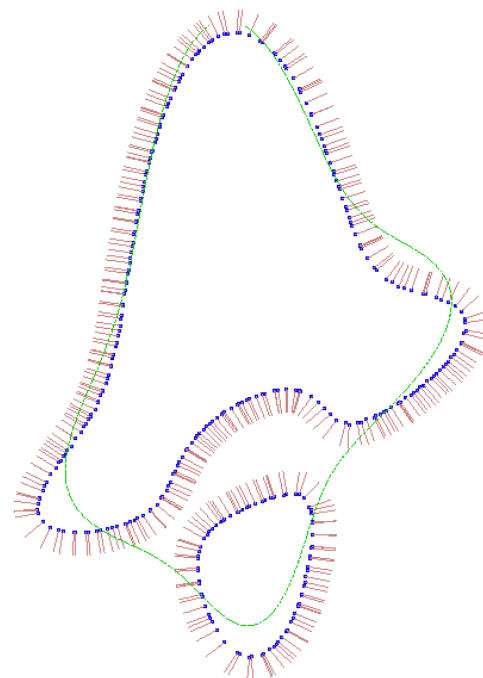


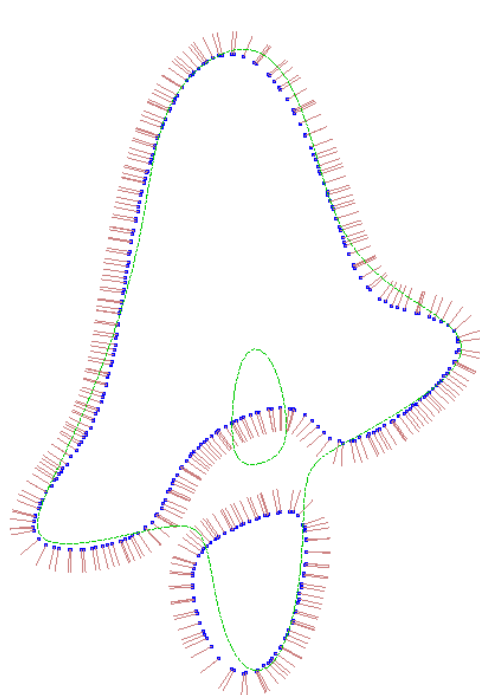
Figura 4.5: Taubin: método 1 usando um polinômio de grau  $d$  com  $\mu = 0.125$ .



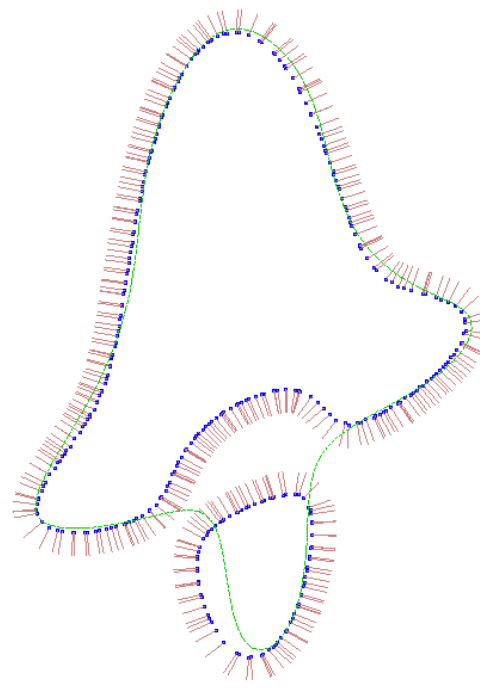
4.6(a):  $d = 2$



4.6(b):  $d = 4$

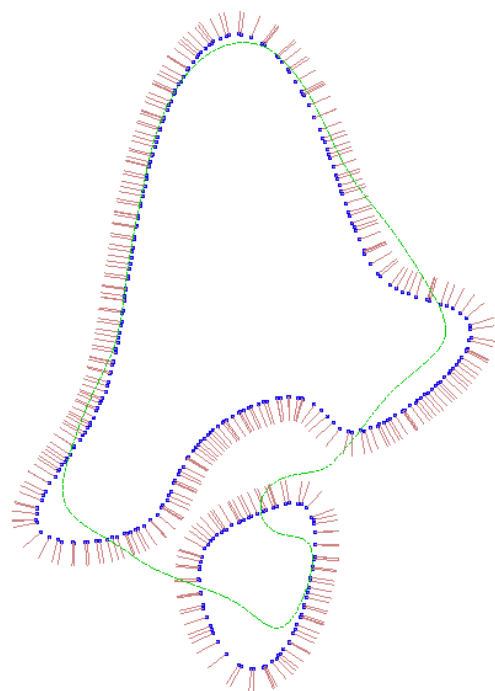


4.6(c):  $d = 6$

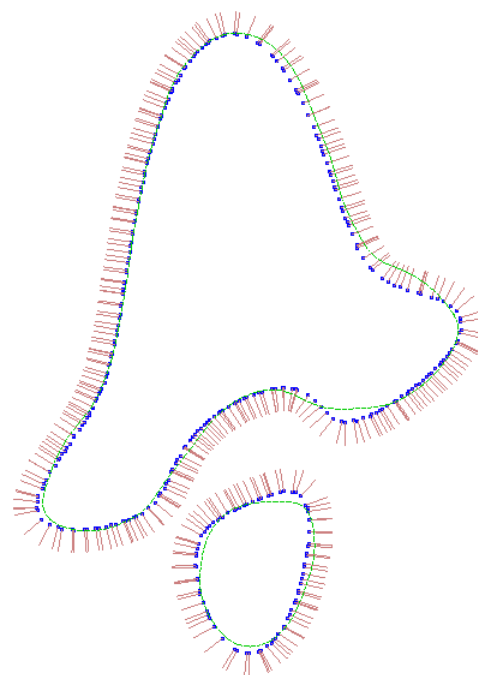


4.6(d):  $d = 8$

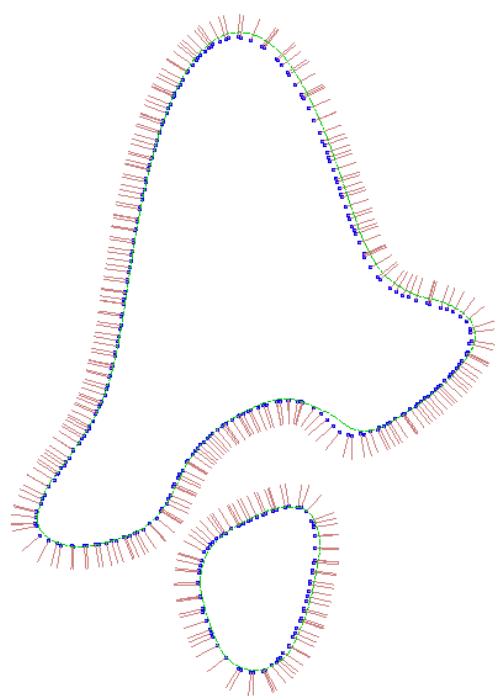
Figura 4.6: Taubin: método 2 usando um polinômio de grau  $d$  com  $\mu = 0.125$  e  $\kappa = 0.001$ .



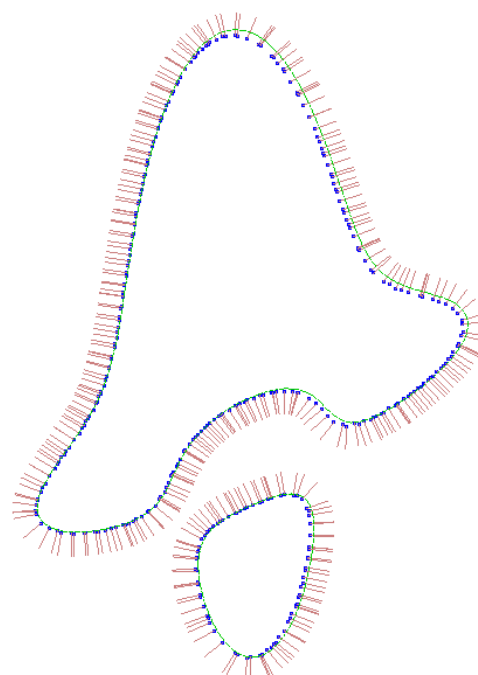
4.7(a):  $d = 2 ; l_{max} = 2$



4.7(b):  $d = 2 ; l_{max} = 3$

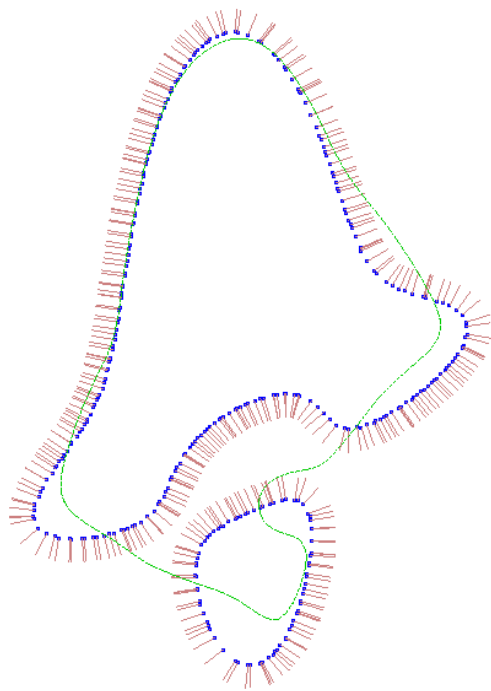


4.7(c):  $d = 2 ; l_{max} = 4$

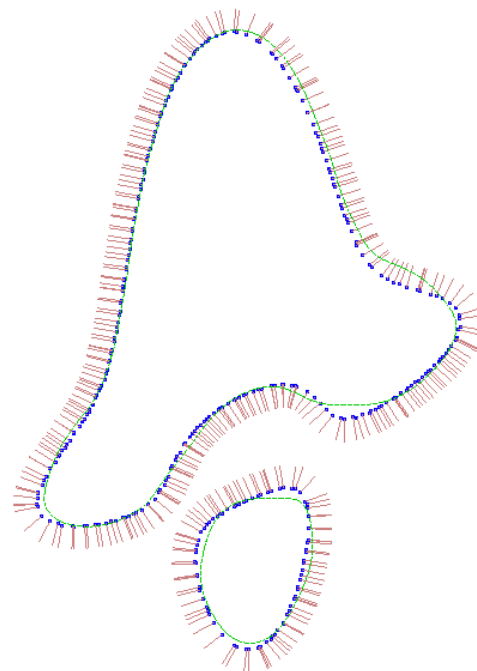


4.7(d):  $d = 2 ; l_{max} = 5$

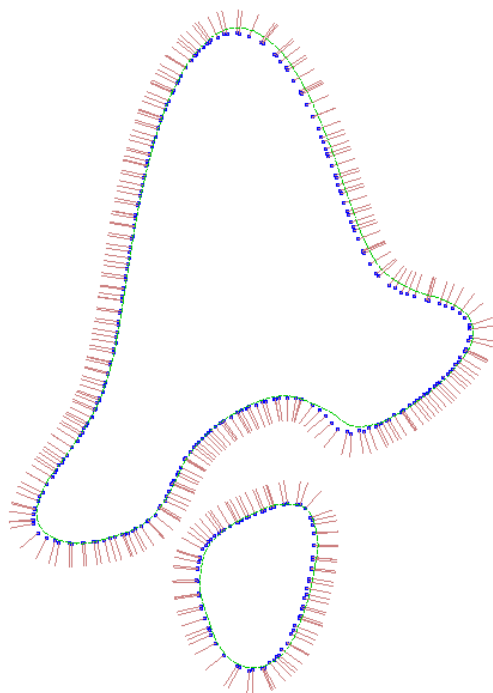
Figura 4.7: Taubin: método 3 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .



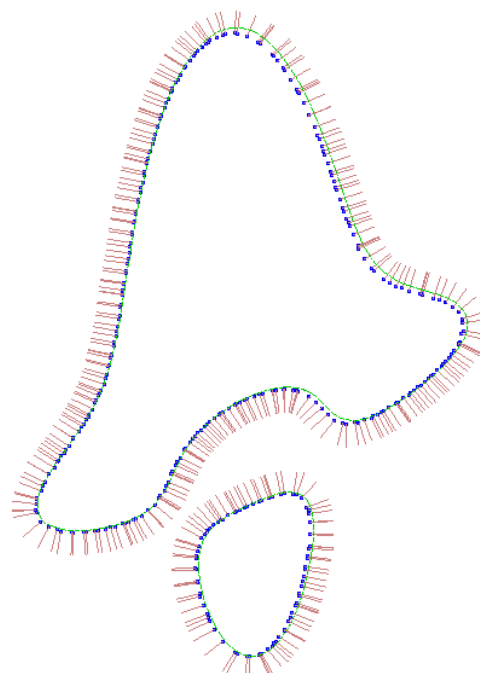
4.8(a):  $d = 2 ; l_{max} = 2$



4.8(b):  $d = 2 ; l_{max} = 3$



4.8(c):  $d = 2 ; l_{max} = 4$



4.8(d):  $d = 2 ; l_{max} = 5$

Figura 4.8: Taubin: método 4 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$ ,  $\kappa = 0.001$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .

Os dois próximos exemplos são de curvas suaves com apenas uma componente conexa.

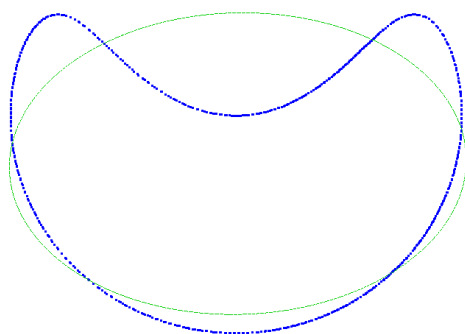
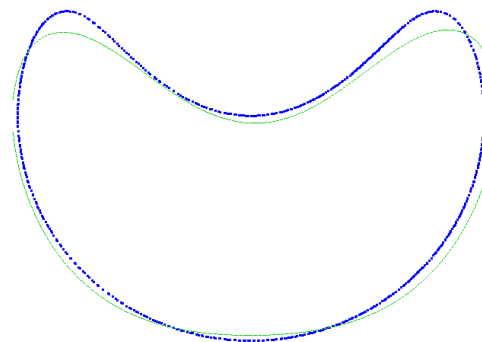
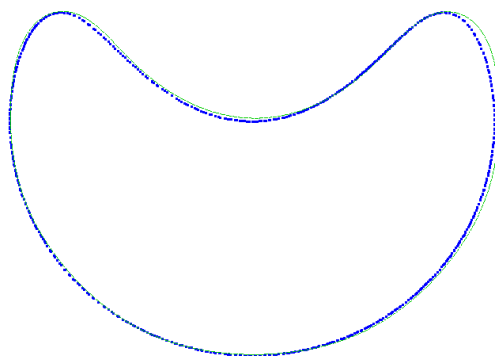
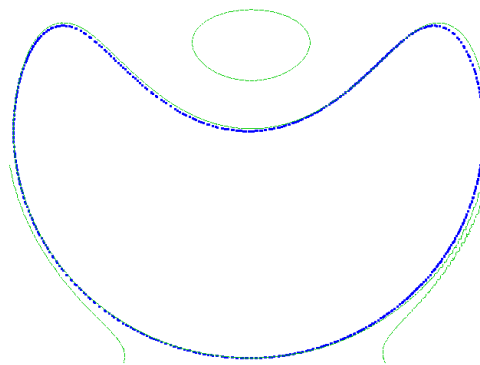
4.9(a):  $d = 2$ 4.9(b):  $d = 4$ 4.9(c):  $d = 6$ 4.9(d):  $d = 8$ 

Figura 4.9: Sorriso: método 1 usando um polinômio de grau  $d$  com  $\mu = 0.125$ .

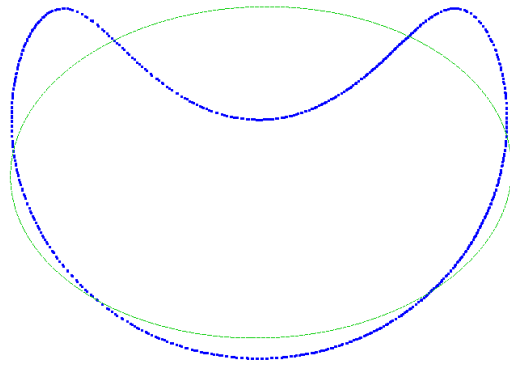
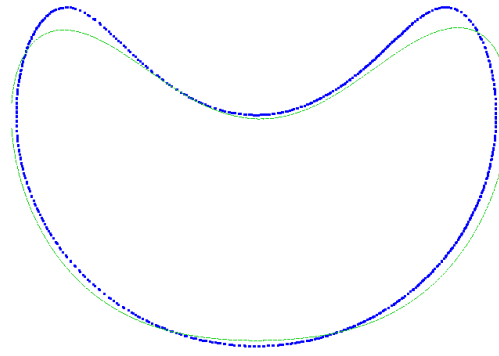
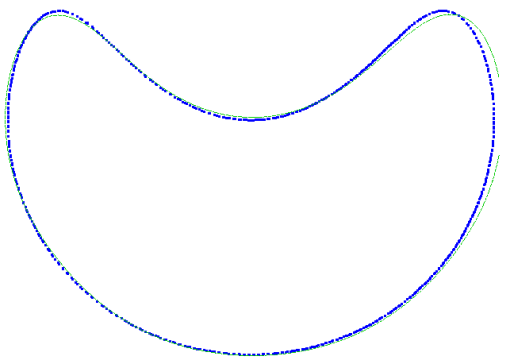
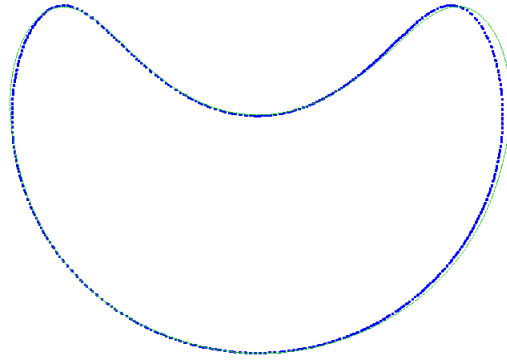
4.10(a):  $d = 2$ 4.10(b):  $d = 4$ 4.10(c):  $d = 6$ 4.10(d):  $d = 8$ 

Figura 4.10: Sorriso: método 2 usando um polinômio de grau  $d$  com  $\mu = 0.125$  e  $\kappa = 0.001$ .

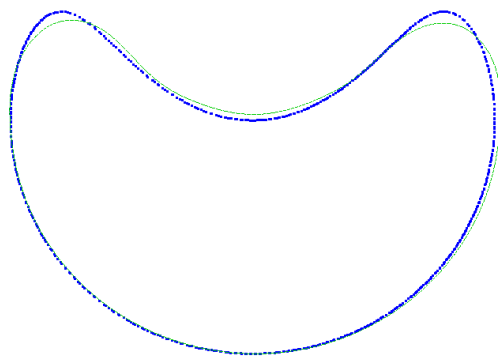
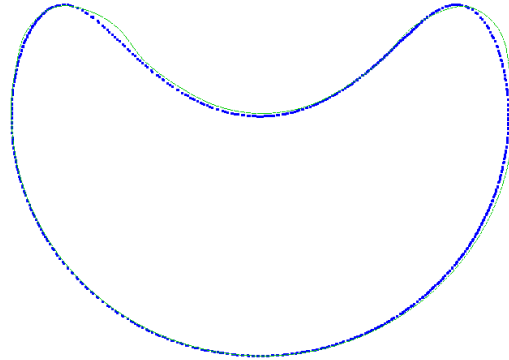
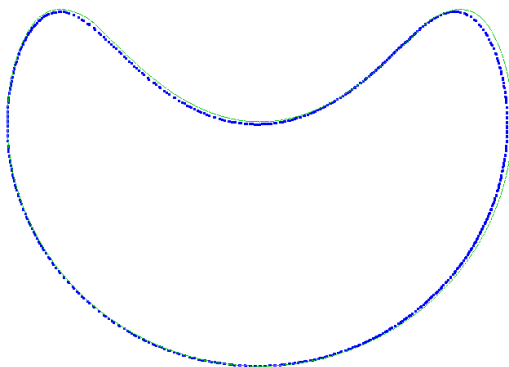
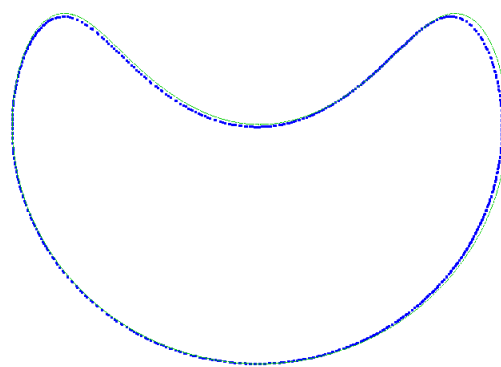
4.11(a):  $d = 2$  ;  $l_{max} = 2$ 4.11(b):  $d = 2$  ;  $l_{max} = 3$ 4.11(c):  $d = 2$  ;  $l_{max} = 4$ 4.11(d):  $d = 2$  ;  $l_{max} = 5$ 

Figura 4.11: Sorriso: método 3 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .

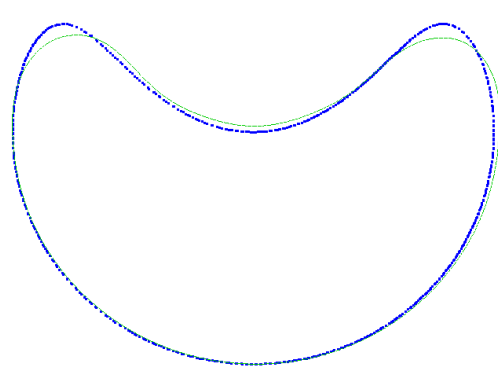
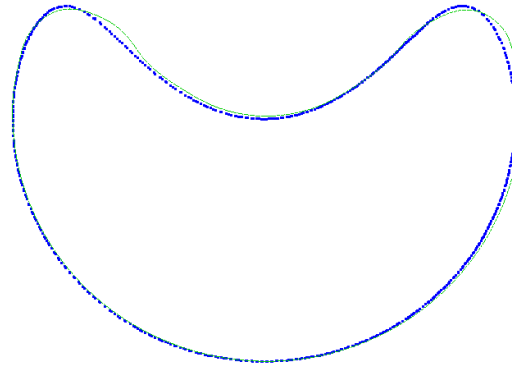
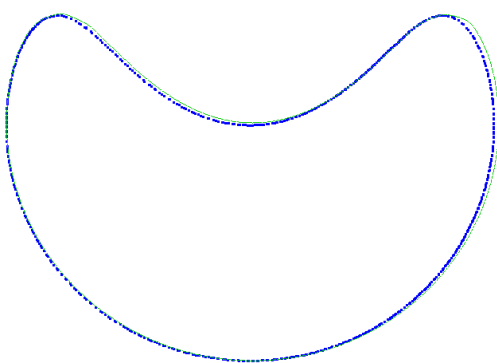
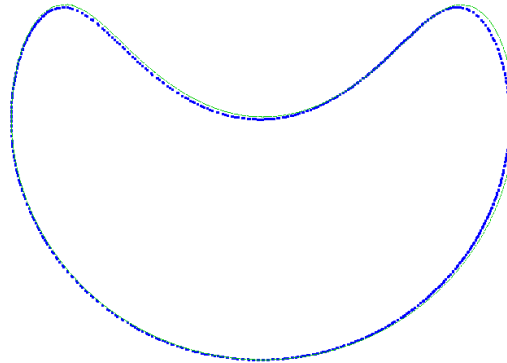
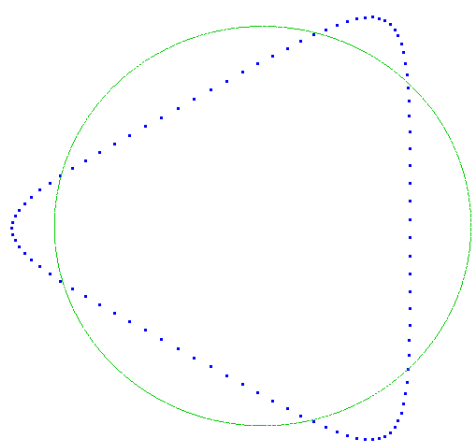
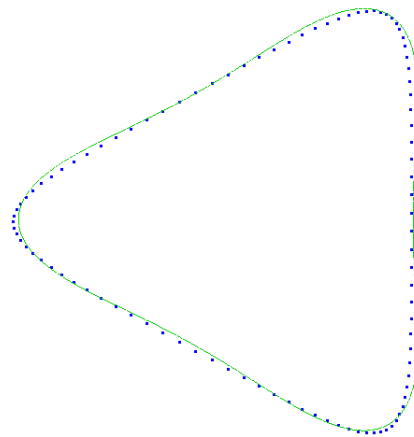
4.12(a):  $d = 2 ; l_{max} = 2$ 4.12(b):  $d = 2 ; l_{max} = 3$ 4.12(c):  $d = 2 ; l_{max} = 4$ 4.12(d):  $d = 2 ; l_{max} = 5$ 

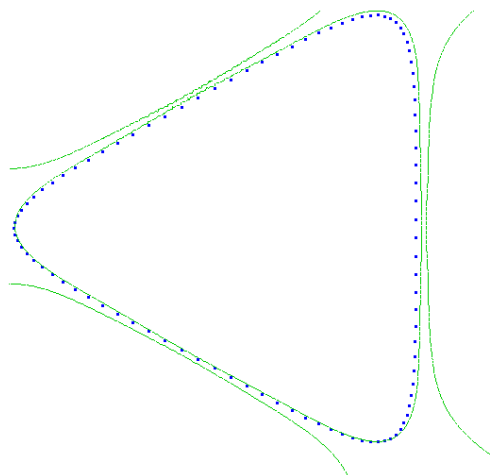
Figura 4.12: Sorriso: método 4 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$ ,  $\kappa = 0.001$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .



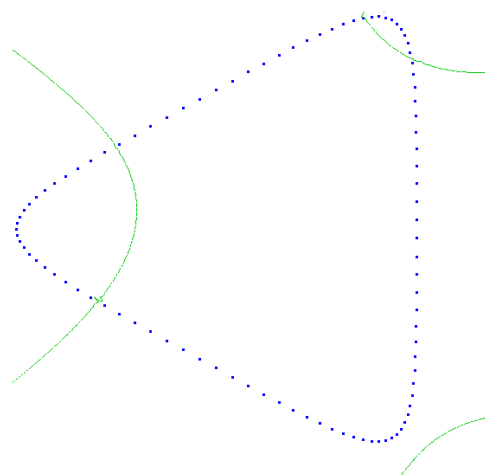
4.13(a):  $d = 2$



4.13(b):  $d = 4$

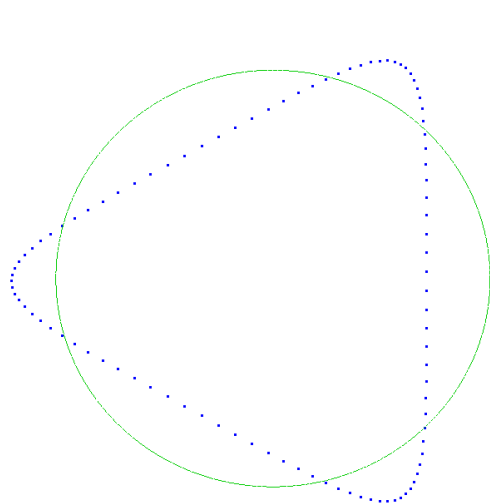


4.13(c):  $d = 6$

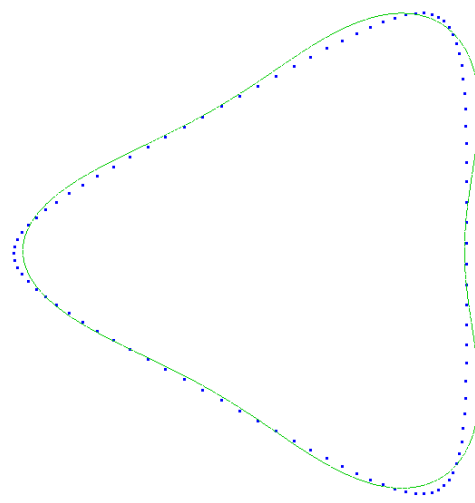


4.13(d):  $d = 10$

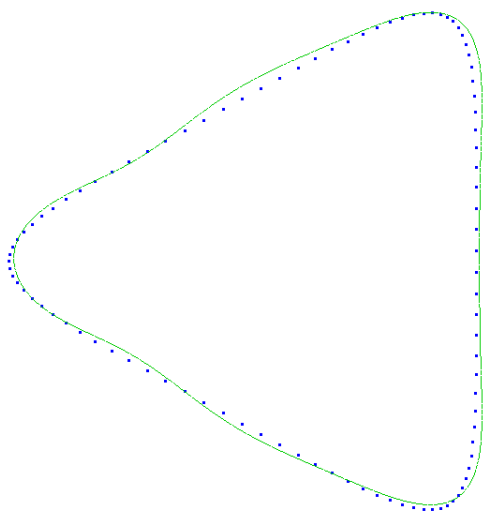
Figura 4.13: Hipociclóide: método 1 usando um polinômio de grau  $d$  com  $\mu = 0.125$ .



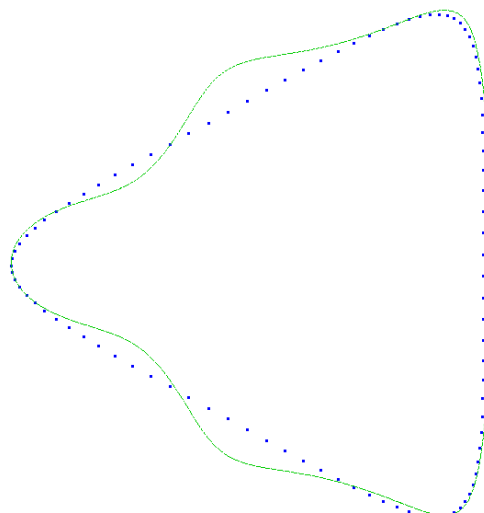
4.14(a):  $d = 2$



4.14(b):  $d = 4$



4.14(c):  $d = 6$



4.14(d):  $d = 10$

Figura 4.14: Hipociclóide: método 2 usando um polinômio de grau  $d$  com  $\mu = 0.125$  e  $\kappa = 0.001$ .

Neste exemplo as aproximações dos métodos estará acompanhada da construção da árvore *Quad-Tree*.

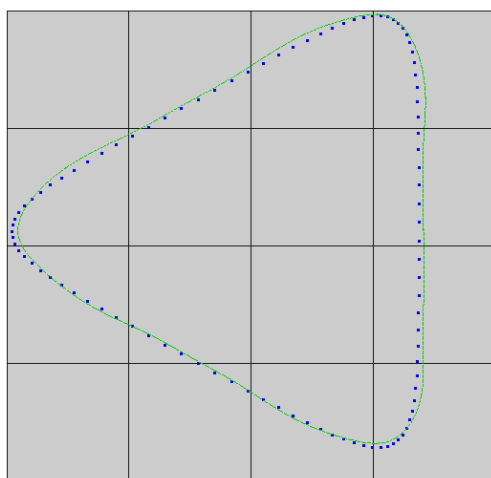
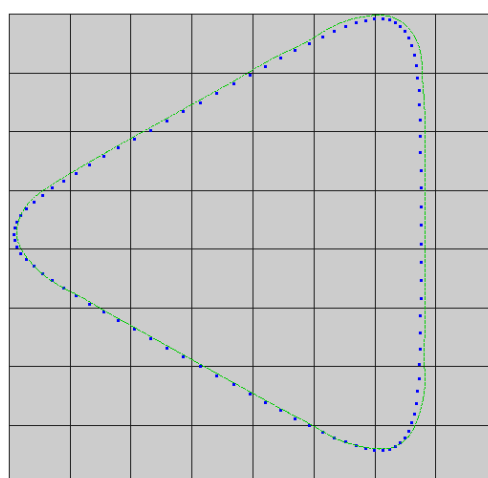
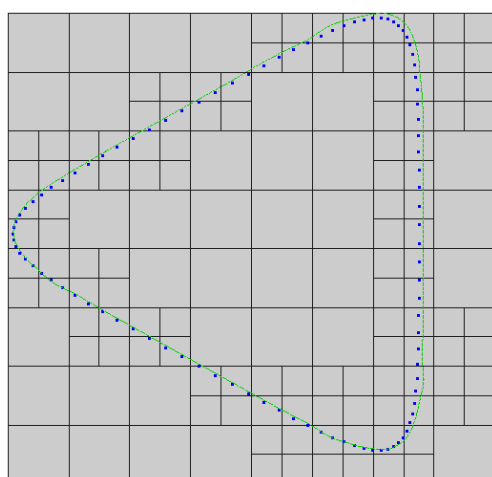
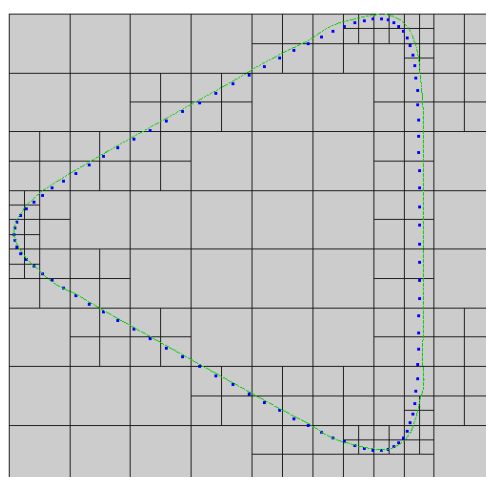
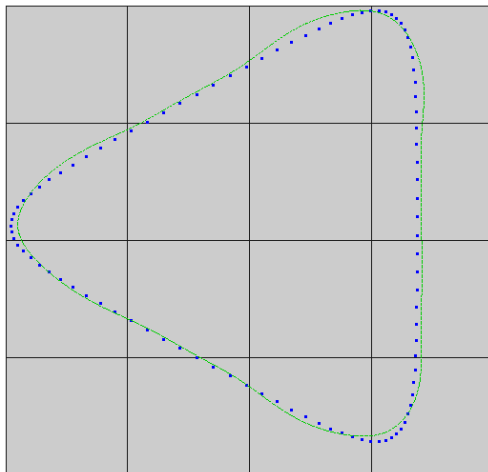
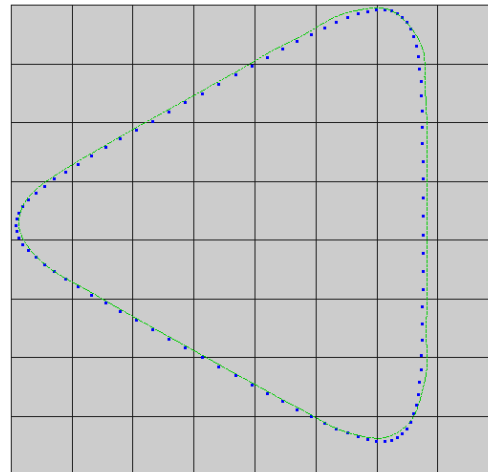
4.15(a):  $d = 2$  ;  $l_{max} = 2$ 4.15(b):  $d = 2$  ;  $l_{max} = 3$ 4.15(c):  $d = 2$  ;  $l_{max} = 4$ 4.15(d):  $d = 2$  ;  $l_{max} = 5$ 

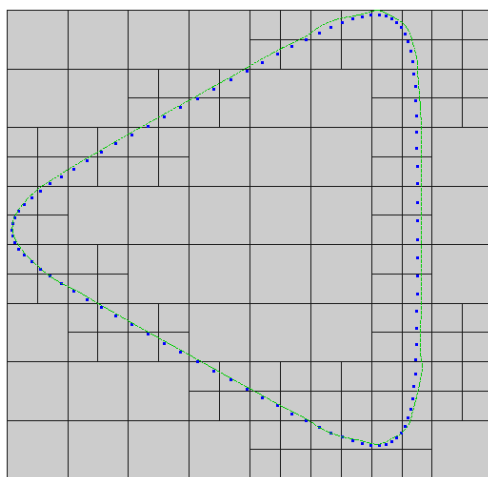
Figura 4.15: Hipociclóide: método 3 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .



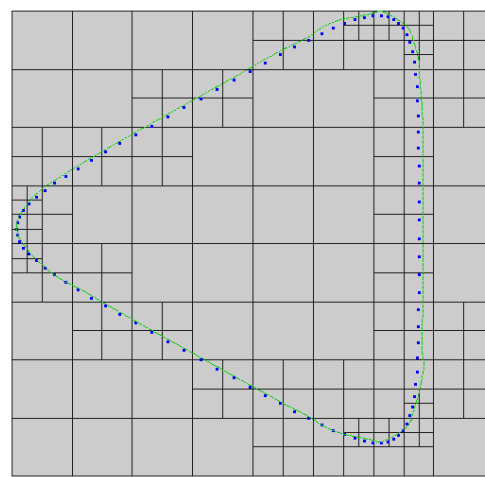
4.16(a):  $d = 2 ; l_{max} = 2$



4.16(b):  $d = 2 ; l_{max} = 3$



4.16(c):  $d = 2 ; l_{max} = 4$



4.16(d):  $d = 2 ; l_{max} = 5$

Figura 4.16: Hipociclóide: método 4 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$ ,  $\kappa = 0.001$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .

## 4.2

### Curvas com singularidades

Agora serão ilustradas curvas com singularidades. Sob ponto de vista teórico, todos esses métodos não deveriam funcionar. Mas mesmo assim, o novo método proposto obteve boas aproximações.

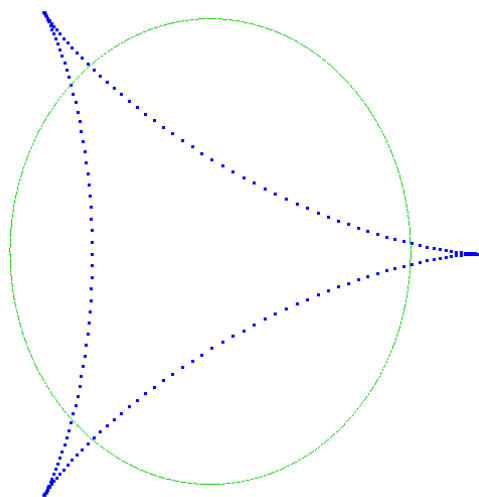
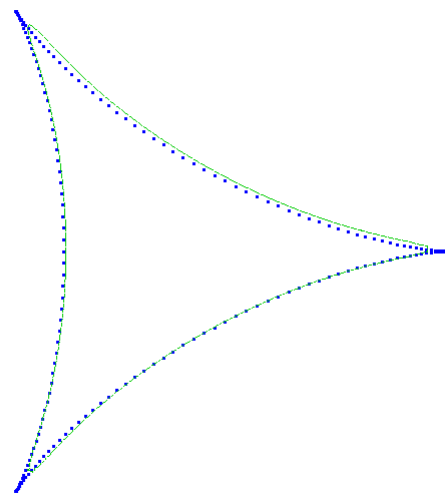
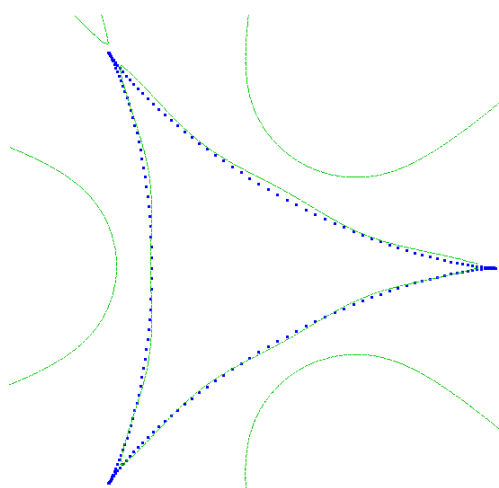
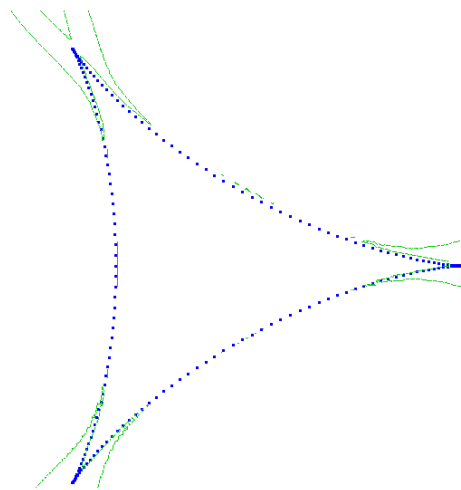
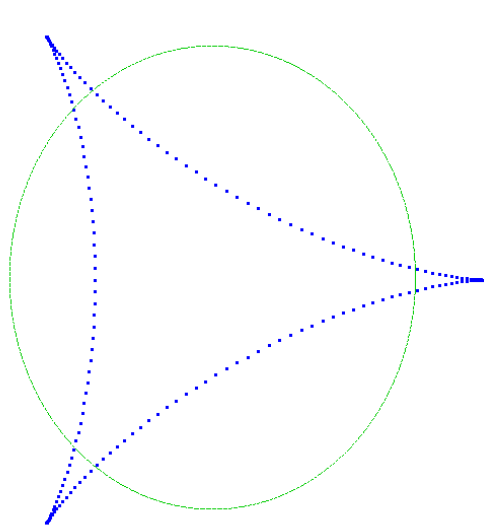
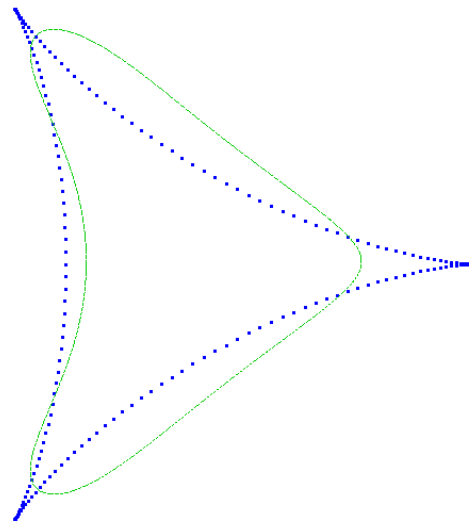
4.17(a):  $d = 2$ 4.17(b):  $d = 4$ 4.17(c):  $d = 6$ 4.17(d):  $d = 8$ 

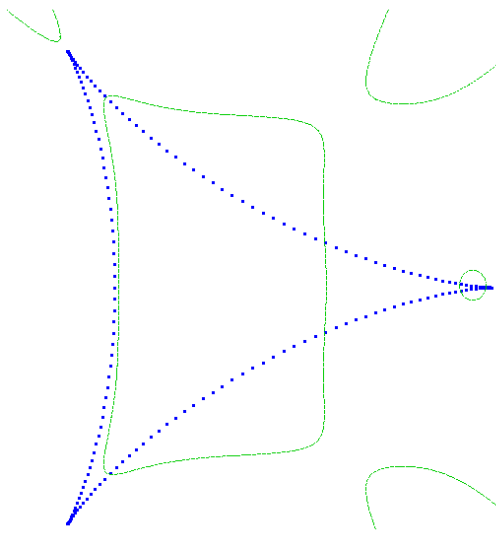
Figura 4.17: Cúspide: método 1 usando um polinômio de grau  $d$  com  $\mu = 0.125$ .



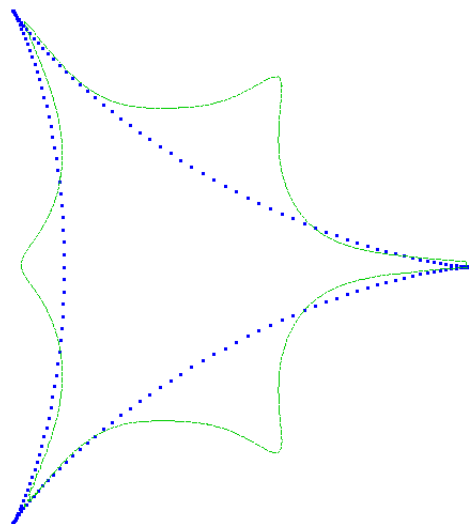
4.18(a):  $d = 2$



4.18(b):  $d = 4$



4.18(c):  $d = 6$



4.18(d):  $d = 8$

Figura 4.18: Cúspide: método 2 usando um polinômio de grau  $d$  com  $\mu = 0.125$  e  $\kappa = 0.001$ .

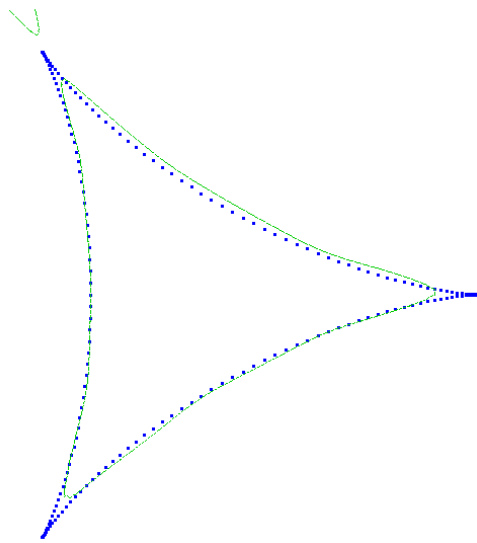
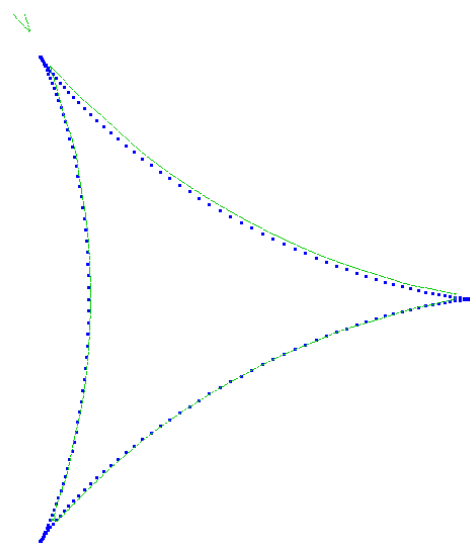
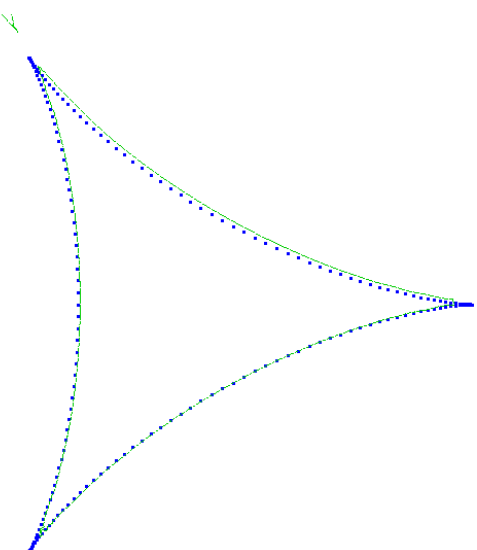
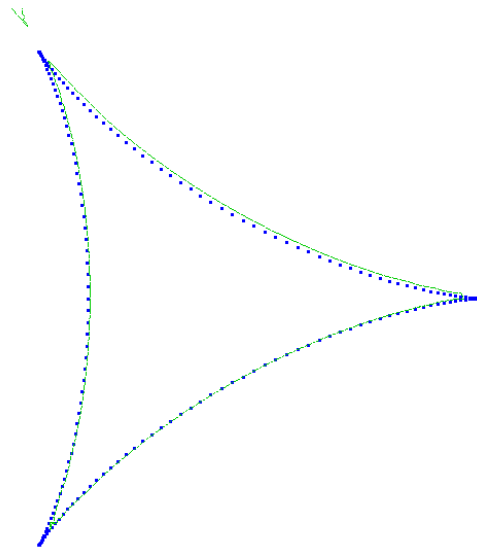
4.19(a):  $d = 2 ; l_{max} = 2$ 4.19(b):  $d = 2 ; l_{max} = 3$ 4.19(c):  $d = 2 ; l_{max} = 4$ 4.19(d):  $d = 2 ; l_{max} = 5$ 

Figura 4.19: Cúspide: método 3 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .

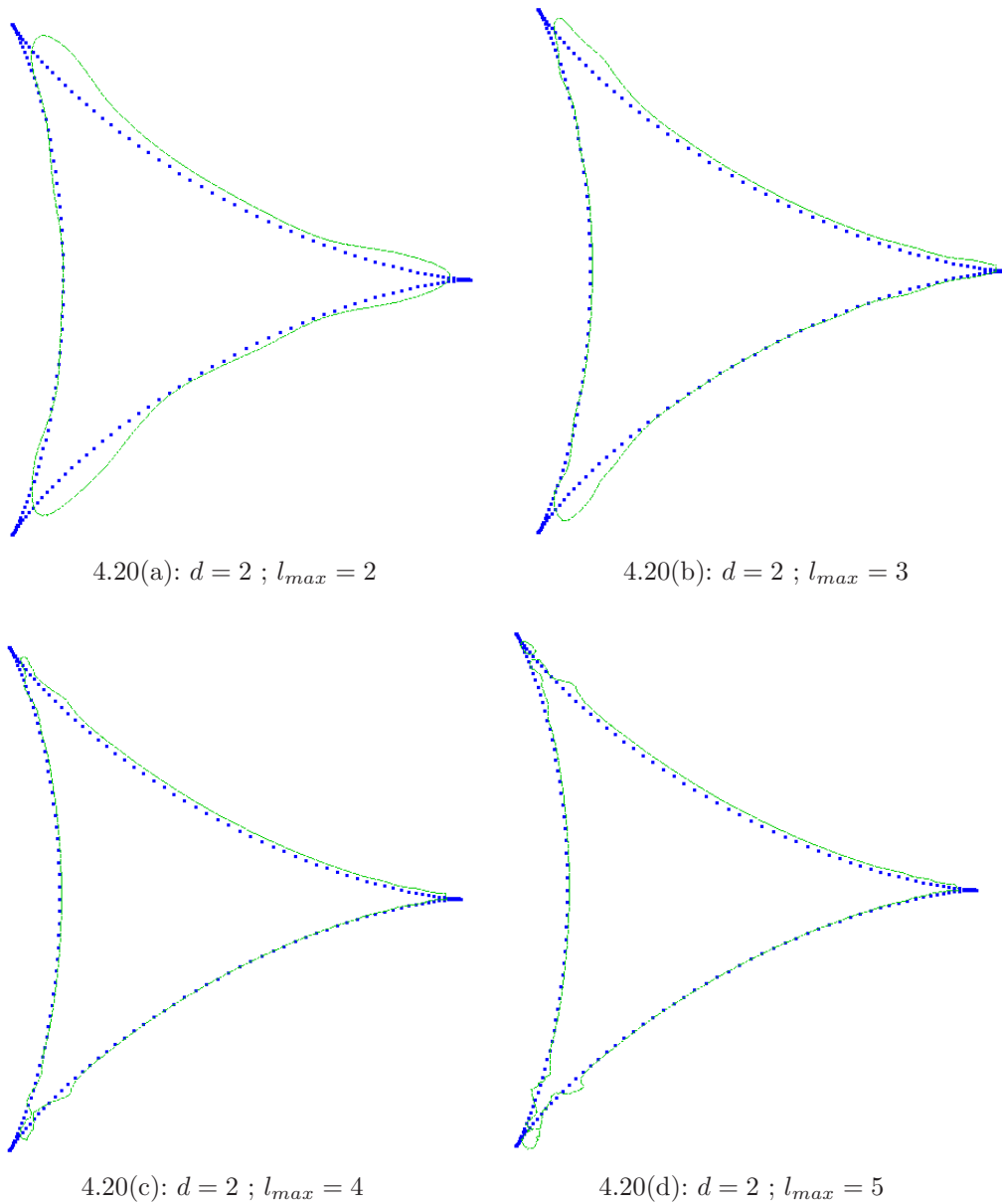
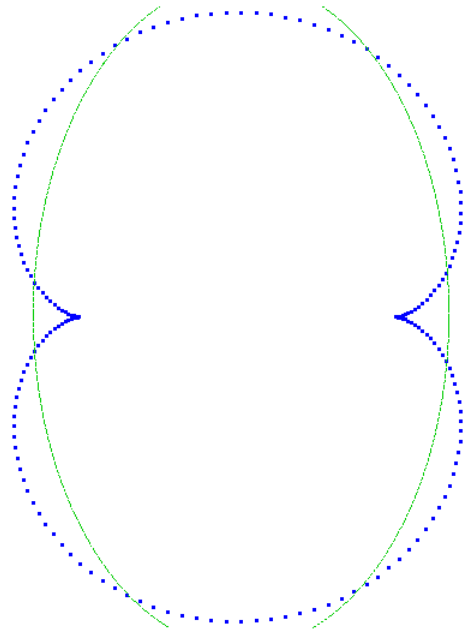
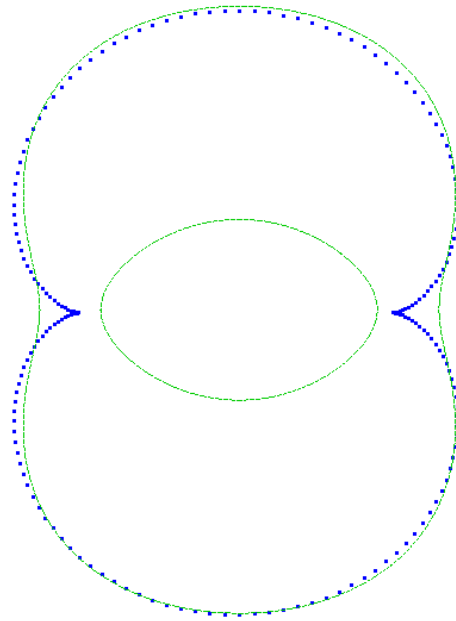


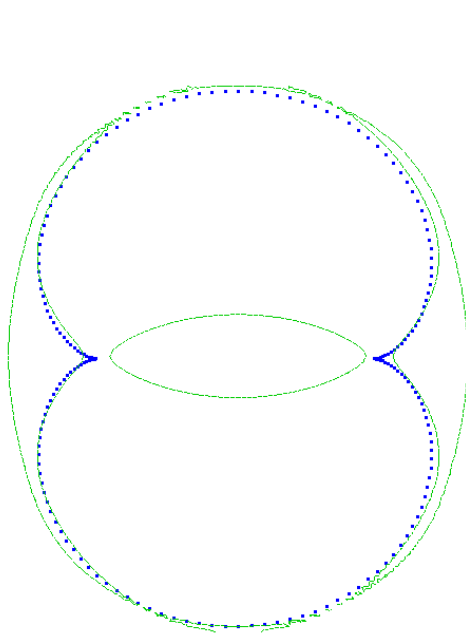
Figura 4.20: Cúspide: método 4 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$ ,  $\kappa = 0.001$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .



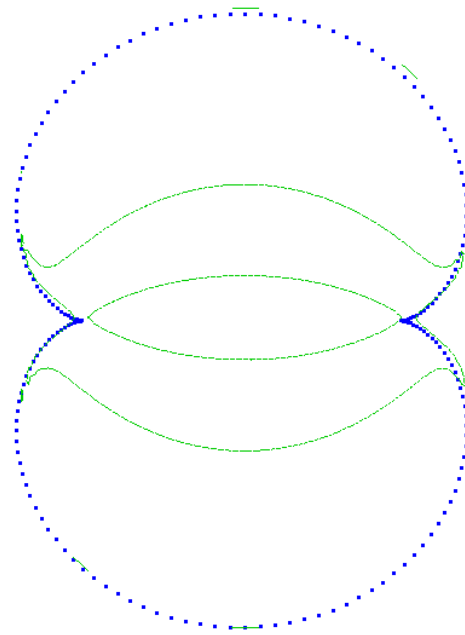
4.21(a):  $d = 2$



4.21(b):  $d = 4$

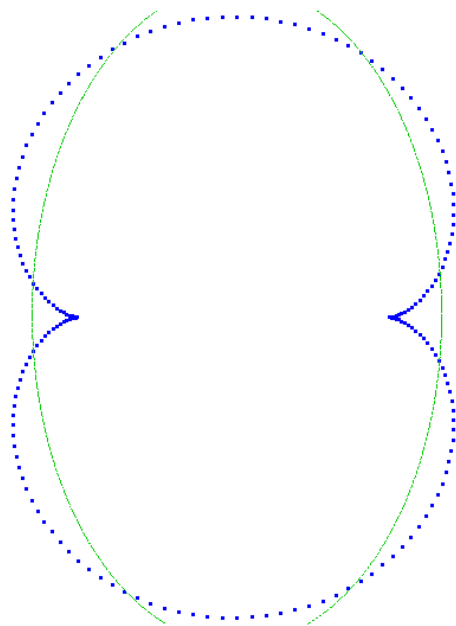


4.21(c):  $d = 6$

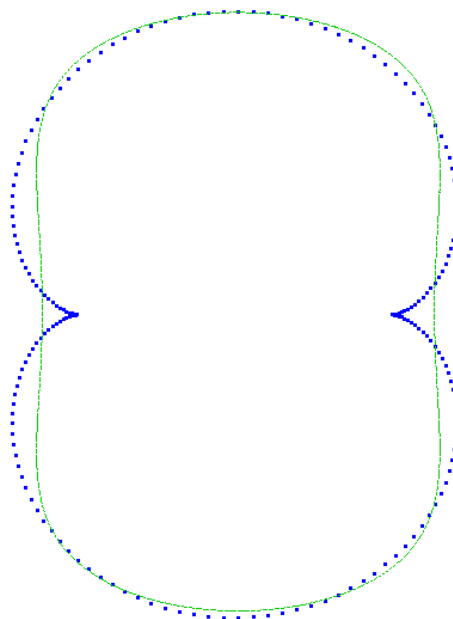


4.21(d):  $d = 8$

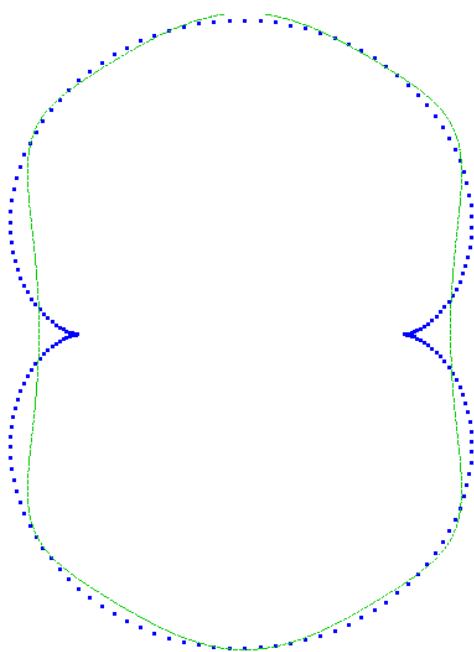
Figura 4.21: Nephroid: método 1 usando um polinômio de grau  $d$  com  $\mu = 0.125$ .



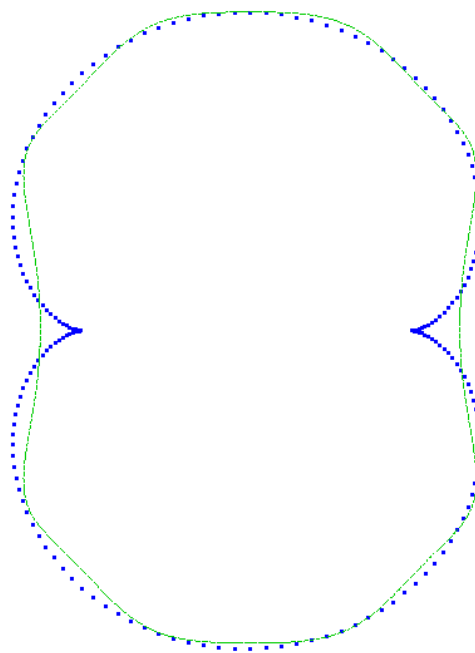
4.22(a):  $d = 2$



4.22(b):  $d = 4$

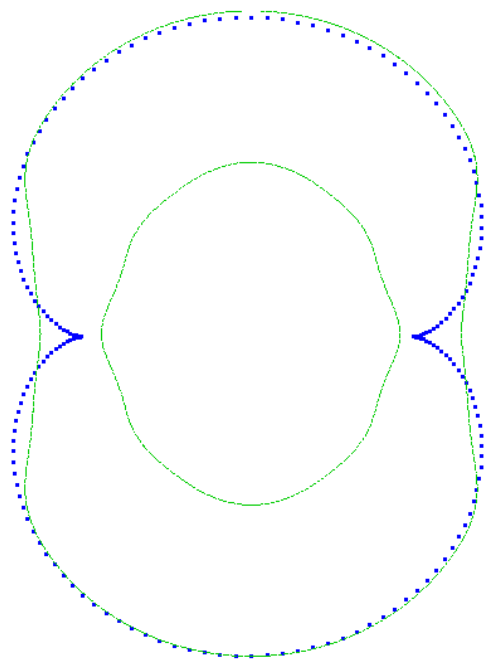


4.22(c):  $d = 6$

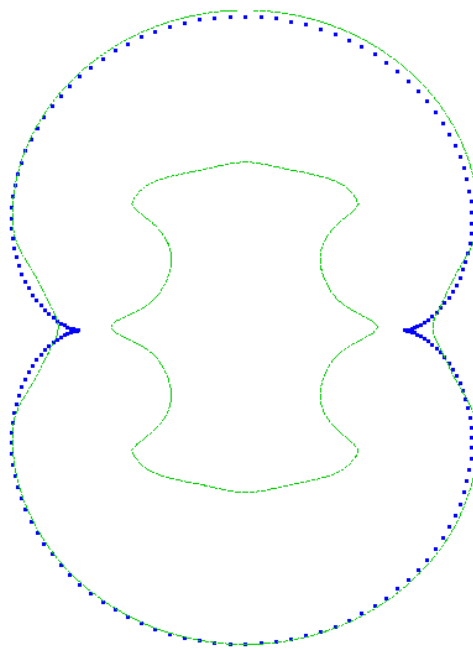


4.22(d):  $d = 8$

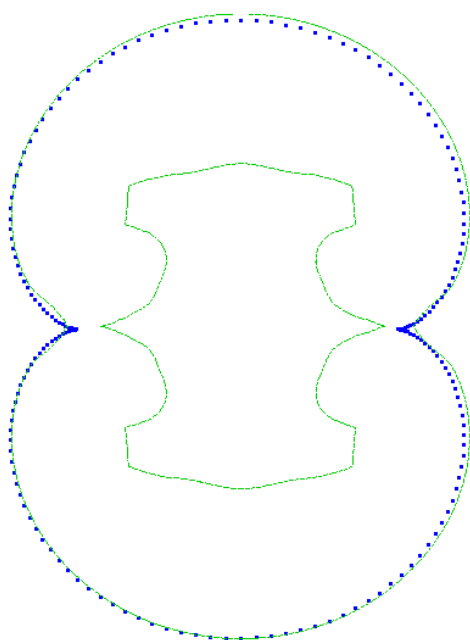
Figura 4.22: Nephroid: método 2 usando um polinômio de grau  $d$  com  $\mu = 0.125$  e  $\kappa = 0.001$ .



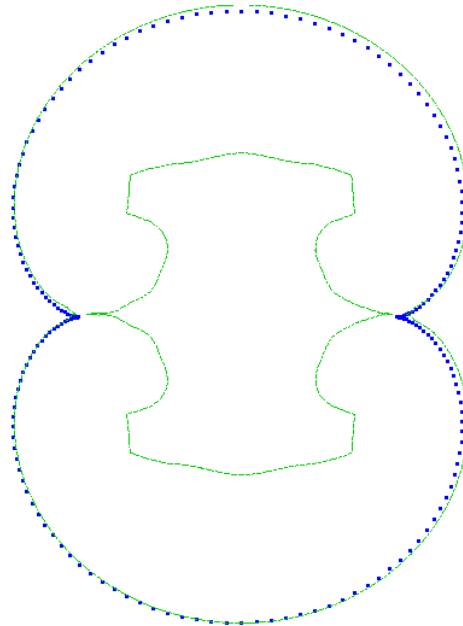
4.23(a):  $d = 2 ; l_{max} = 2$



4.23(b):  $d = 2 ; l_{max} = 3$

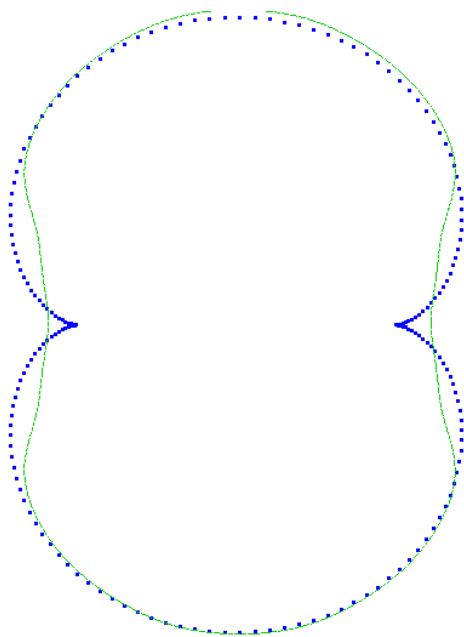


4.23(c):  $d = 2 ; l_{max} = 4$

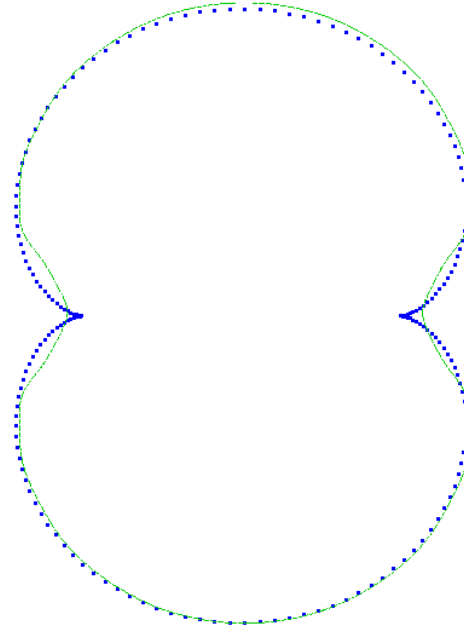


4.23(d):  $d = 2 ; l_{max} = 5$

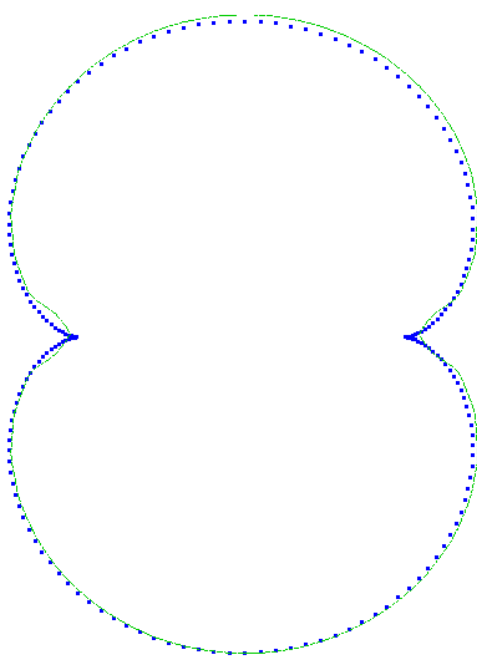
Figura 4.23: Nephroid: método 3 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .



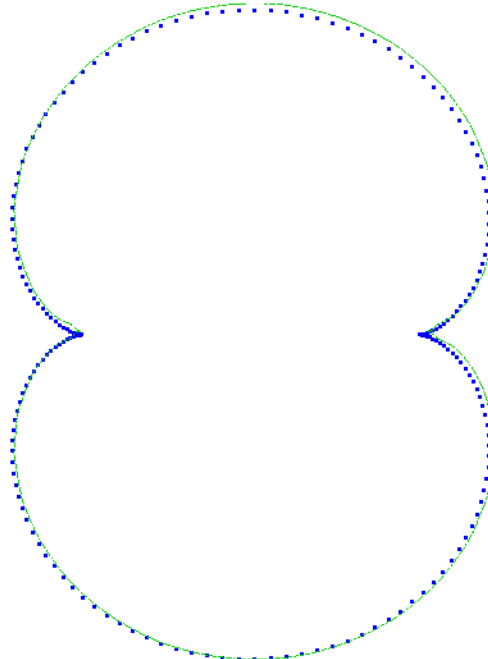
4.24(a):  $d = 2 ; l_{max} = 2$



4.24(b):  $d = 2 ; l_{max} = 3$



4.24(c):  $d = 2 ; l_{max} = 4$



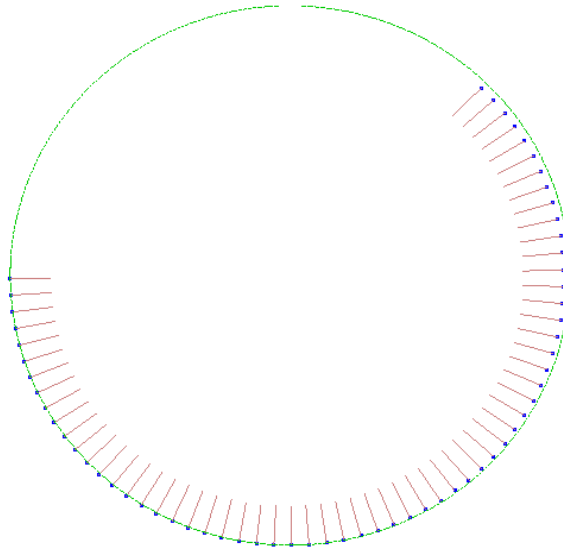
4.24(d):  $d = 2 ; l_{max} = 5$

Figura 4.24: Nephroid: método 4 usando localmente um polinômio de grau  $d$  com  $\mu = 0.125$ ,  $\kappa = 0.001$  e uma *Quad-Tree* com nível máximo igual a  $l_{max}$ .

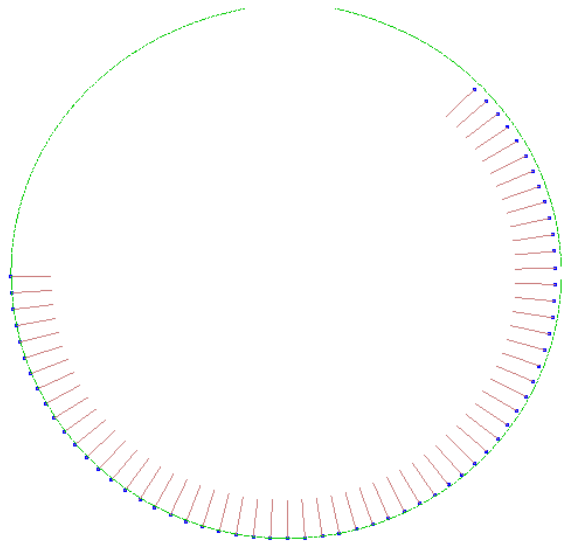
### 4.3

#### Completando buracos

Muitas vezes os dados vindos dos dispositivos vêm com ruído ou com buracos. Esses buracos podem ser muitas vezes completados diretamente por esses métodos implícitos. Seguem alguns exemplos para ilustrar a performance do novo método na solução desse problema.

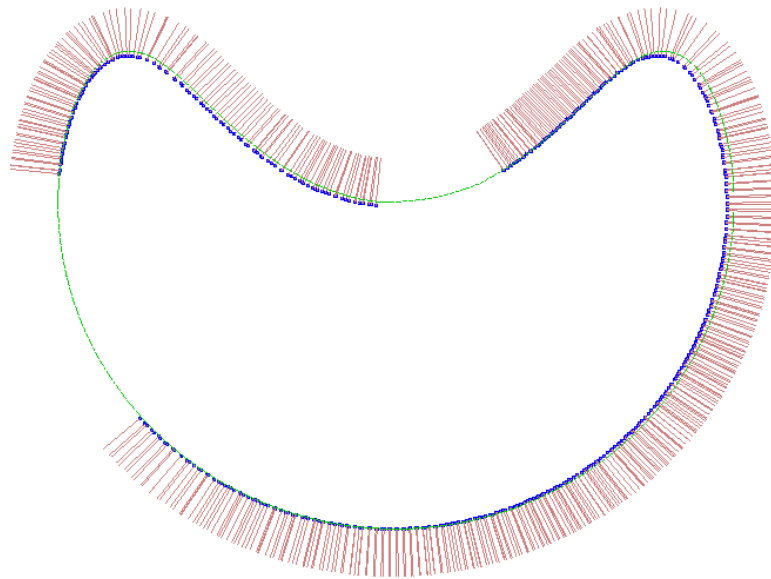


4.25(a): Utilizando o método 3

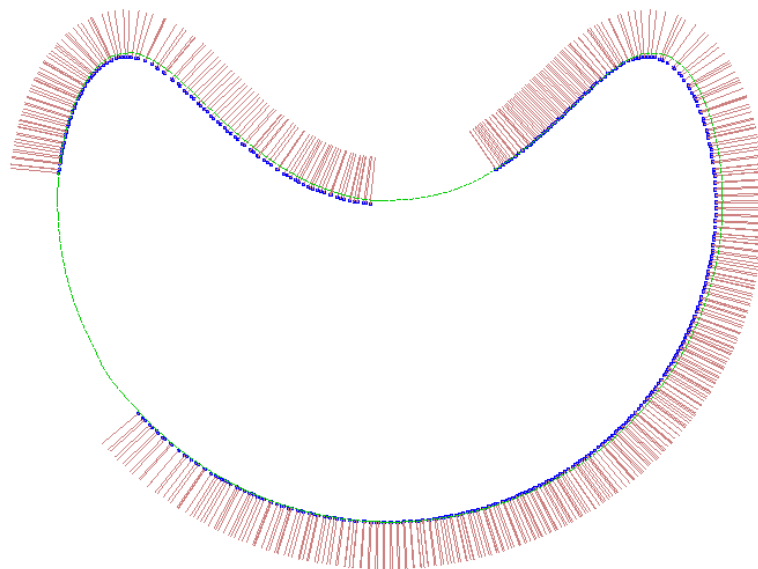


4.25(b): Utilizando o método 4

Figura 4.25: Círculo: (a) método 3 e (b) método 4 usando localmente um polinômio de grau  $d = 2$  com  $\mu = 0.17$ ,  $\kappa = 0.01$  e uma *Quad-Tree* com nível máximo igual a 5

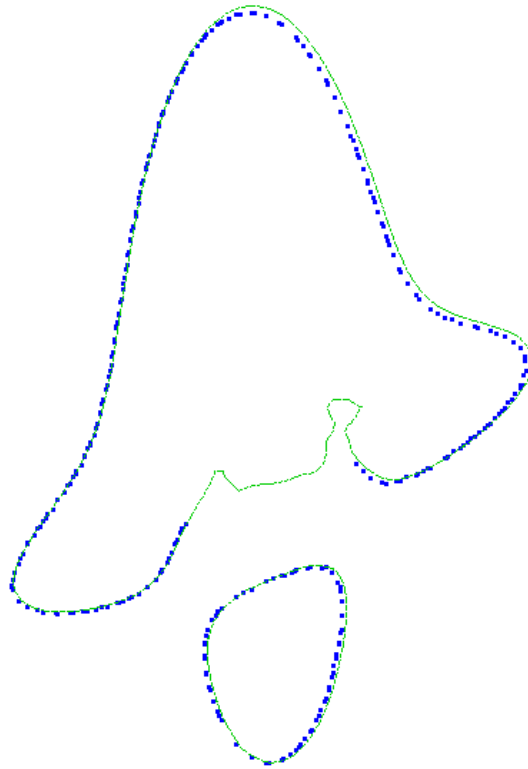


4.26(a): Utilizando o método 3

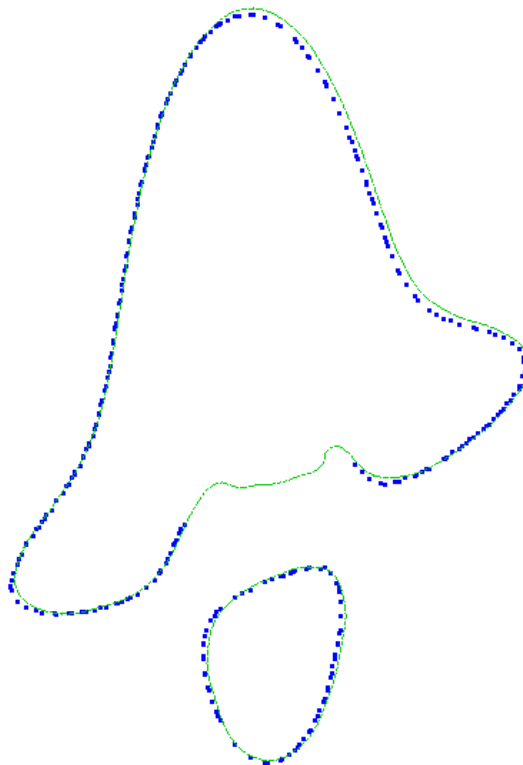


4.26(b): Utilizando o método 4

Figura 4.26: Sorriso: (a) método 3 e (b) método 4 usando localmente um polinômio de grau  $d = 2$  com  $\mu = 0.17$ ,  $\kappa = 0.01$  e uma *Quad-Tree* com nível máximo igual a 5



4.27(a): Utilizando o método 3



4.27(b): Utilizando o método 4

Figura 4.27: Taubin: (a) método 3 e (b) método 4 usando localmente um polinômio de grau  $d = 2$  com  $\mu = 0.17$ ,  $\kappa = 0.01$  e uma *Quad-Tree* com nível máximo igual a 4

## 5

### Conclusão e trabalhos futuros

Essa dissertação apresentou um novo método baseado na partição da unidade para construção de uma curva implícita que aproxima um conjunto de pontos esparsos no plano. Ele é uma combinação do método MPU [13] com o de *Ridge Regression* [17].

Por um lado, as suas principais diferenças em relação ao MPU [13] estão presentes principalmente no processo de obtenção da aproximação local da curva e no algoritmo de avaliação da função em um ponto.

Por outro lado, a sua diferença em relação ao trabalho de Tasdizen et al. [17] está na introdução da partição da unidade para a obtenção de uma aproximação global à partir de aproximações locais. Esse fato faz com que não seja mais necessário usar polinômios de grau muito alto para poder representar objetos complexos através de uma função implícita. Pois com o uso da partição da unidade é possível obter representações implícitas globais desses objetos combinando, através da partição da unidade, aproximações locais de grau baixo.

O método apresentou resultados visualmente bastante eficientes. Além disso ele ficou bem flexível devido aos seus parâmetros de entrada que controlam os pesos dados às normais, às tangentes e aos coeficientes para o *ridge regression*.

Algumas sugestões para trabalhos futuros são:

- Generalizar o método para gerar superfícies implícitas no  $\mathbb{R}^3$ .
- Implementar um método que ao invés de usar na aproximação local um polinômio bivariado em cada nó (i.e.,  $F_i(x, y) = P_d(x, y)$ ) utilize funções univariadas  $F_i(x, y) = y - P_d(x)$  ou  $F_i(x, y) = x - P_d(y)$ .
- Implementar um método que ao invés de usar a *Quad-Tree* utilize uma estrutura de divisão hierárquica binária, como a *Binary Space Partition* (BSP).
- Implementar novas formas de se combinar as aproximações locais, como por exemplo usar as normas do máximo, ou normas  $p$ .

## Referências Bibliográficas

- [1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *IEEE Visualization 2001*, pages 21–28, October 2001. ISBN 0-7803-7200-x. 1
- [2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003. 1
- [3] N. Amenta, M. Bern, and D. Eppstein. The crust and the  $\beta$ -skeleton. *Graphical Models and Image Processing*, 60:125–135, 1998. 1
- [4] N. Amenta, S. Choi, and R. K. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry*, 19(2-3):127–153, 2001. 1
- [5] F. Bernardini and C. L. Bajaj. Sampling and reconstructing manifolds using alpha-shapes. In *Proc. 9th Canadian Conf. Computational Geometry*, pages 193–198, 1997. 1
- [6] M. M. Blane, Z. Lei, H. Çivi, and D. B. Cooper. The 3l algorithm for fitting implicit polynomial curves and surfaces to data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(3):298–313, 2000. 1
- [7] L. H. de Figueiredo and J. Gomes. Computational morphology of curves. *The Visual Computer*, 11(2):105–112, 1995. 1
- [8] T. K. Dey, K. Mehlhorn, and E. A. Ramos. Curve reconstruction: Connecting dots with good reason. In *ACM Symposium on Computational Geometry*, pages 197–206, 1999. 1
- [9] D. Keren. Topologically faithful fitting of simple closed curves. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(1):118–123, 2004. 1
- [10] Z. Lei, M. M. Blane, and D. B. Cooper. 3l fitting of higher degree implicit polynomials. In *WACV '96: Proceedings of the 3rd IEEE Workshop on Applications of Computer Vision (WACV '96)*, page 148, Washington, DC, USA, 1996. IEEE Computer Society. 2.2.2

- [11] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3D scanning of large statues. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 131–144. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. 1
- [12] B. Mederos, L. Velho, and L. H. de Figueiredo. Moving least squares multiresolution surface approximation. In *Proceedings of the XVI Brazilian Symposium on Computer Graphics and Image Processing*, 2003. 1
- [13] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003. 1, 1.2, 2.3, 2.4, 2.4, 2.4, 3.1, 5
- [14] H. Pottmann, I. Lee, and T. Randrup. Reconstruction of kinematic surfaces from scattered data. Technical report, Technical Report No. 48, Institut fur Geometrie, 1998. 1
- [15] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes in C*. Cambridge Univ. Press, 1988. 2.2.1
- [16] T. Tasdizen, J.-P. Tarel, and D. Cooper. Improving the stability of algebraic curves for applications. *IEEE Transactions on Image Processing*, 9(3):405–416, March 2000. <http://www-rocq.inria.fr/tarel/ip00.html>. 2.2.1
- [17] T. Tasdizen, J. P. Tarel, and D. B. Cooper. Algebraic curves that work better. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'99)*, volume II, pages 35–41, Fort Collins, Colorado, USA, 1999. <http://www-rocq.inria.fr/tarel/cvpr99.html>. 1, 2.1, 2.2.1, 2.2.2, 2.2.3, 2.2.3, 2.2.3, 3.1, 5
- [18] 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.*, 13(11):1115–1138, 1991. 1, 2.2.1