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 [16] e o proposto por Tasdizen et al. em [20], 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 $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_q\} \in \mathbb{R}^2$ amostrados de uma curva planar $\mathcal{C}$, onde $\mathbf{p}_i = (x_i, y_i)$.
- o correspondente conjunto dos vetores normais unitários $\mathcal{N} = \{\mathbf{n}_1, \mathbf{n}_2, \ldots, \mathbf{n}_q\}$.

Já o conjunto $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_q\}$ 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 Ξ 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, a cada nó i da *Quad-Tree* tem-se associado:

– Uma função peso que é utilizada na partição da unidade φ_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 α vezes o tamanho da digonal do quadrado correspondente. Geralmente, usamos $\alpha = 0.75$.

  Assim como no método MPU em [16] a função partição da unidade φ_i para cada quadrado Ξ_i é dada pela equação:

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

  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 φ_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:

$$\{\mathbf{a}^t\mathbf{S_i}\mathbf{a} + \mu\mathbf{a}^t\mathbf{S_{N,i}}\mathbf{a} + \mu\mathbf{a}^t\mathbf{S_{T,i}}\mathbf{a} - 2\mu\mathbf{a}^t\mathbf{g_{N,i}} + \mu q_i + \kappa\mathbf{a}^t\mathbf{\Delta_i}\mathbf{a}\}.$$

  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 $\mathbf{S_{N,i}}$ de tamanho $l \times l$ é:

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

  – A matriz $\mathbf{S_{T,i}}$ de tamanho $l \times l$ é:

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

– 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 Δ_i é obtida preenchendo os elementos da sua diagonal da seguinte maneira:

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

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:

$$(\mathbf{S_i} + \mu \mathbf{S_{N,i}} + \mu \mathbf{S_{T,i}} + \kappa \mathbf{\Delta_i})\mathbf{a} = \mu \mathbf{g_{N,i}}. \tag{3-2}$$

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 ε 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 φ_i.

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

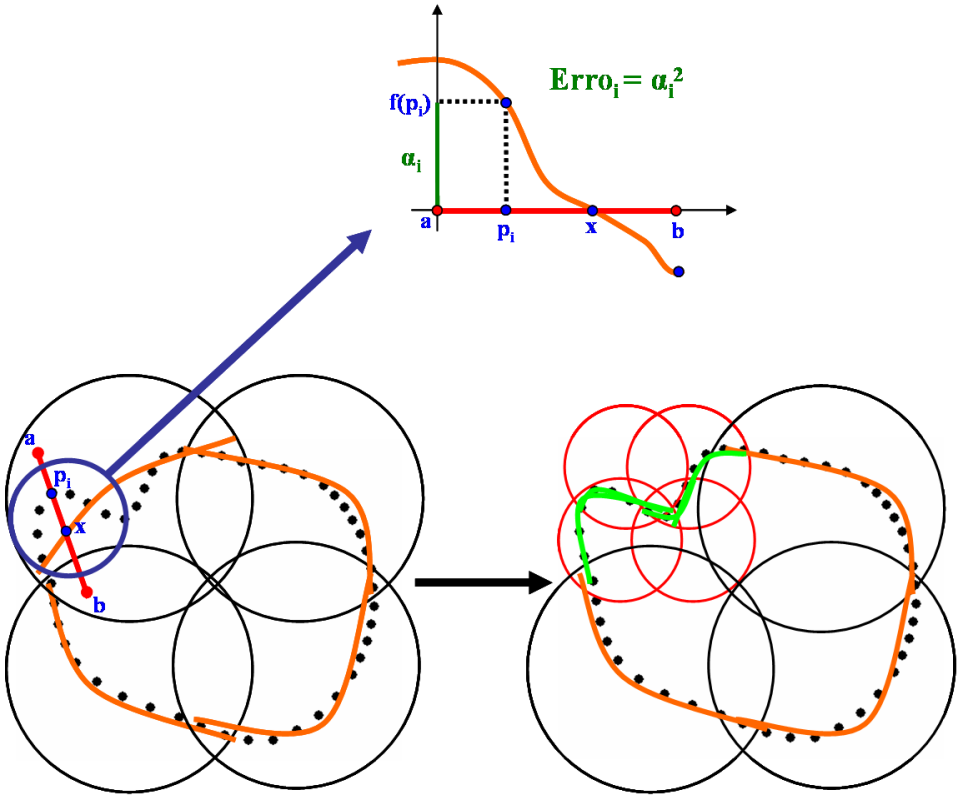

Figura 3.1: Refinamento local que permite uma melhor aproximação global.

Segue uma descrição um 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.

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` $\mathbf{c}[2]$: 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 *` $\mathcal{I}$: um vetor de inteiros que representa o conjunto dos índices i dos pontos que se encontram dentro do círculo de raio r centrado em $\mathbf{c}$.
- `Quad_Tree *` $son[4]$: um vetor de ponteiros que apontam para os quatros 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`.

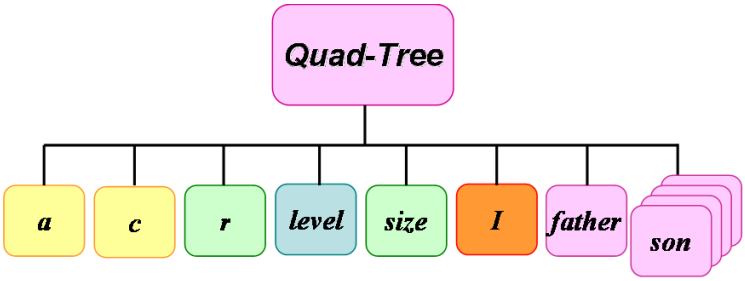

Figura 3.2: Organização da classe `Quad_Tree`.

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` α : multiplicador para o raio da região de suporte.
- `double` μ : constante do método *gradient one fitting*.
- `double` κ : constante do método *rigde regression*.
- `double` ε : tolerância para o erro local da aproximação.
- `Quad_Tree *` $root$: ponteiro para a raiz da árvore *Quad_Tree*.

- `double **` $\mathcal{P}$: 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 **` $\mathcal{N}$: 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 $\mathcal{P}$.

A figura 3.3 ilustra a organização dos membros dessa classe.

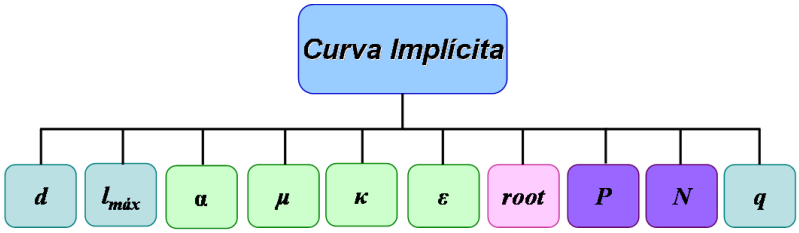

Figura 3.3: Classe `Curva_Implícita`.

O construtor *default* da classe `Curva_Implícita` possui os seguintes parâmetros para iniciação das variáveis: d, l_{max}, μ, κ, ε, α, 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 Implicita* 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`$(father, level, l_{max}, d, \alpha, \mu, \kappa, \varepsilon, \mathcal{P}, \mathcal{N}, q, size, \mathbf{c})$

1: **if** $(l_i > l_{max})$ **then**
2: return NULL;
3: **end if**
4: Cria nó N;
5: $N \rightarrow father = father$;
6: $N \rightarrow \mathbf{c} = \mathbf{c}$;
7: $N \rightarrow size = size$;
8: $N \rightarrow level = level$;
9: $N \rightarrow r = \alpha \times \sqrt{2} \times size$;
10: $\mathcal{I} =$ `Subset`$(\mathcal{P}, \mathbf{c}, r)$;
11: **if** $(SizeOf(\mathcal{I}) \geq \frac{(d+1)(d+2)}{2})$ **then**
12: $N \rightarrow \mathbf{a} =$ `Fit`$(d, \mu, \kappa, \mathcal{I}, \mathcal{P}, \mathcal{N}, q)$;
13: $erro \leftarrow$ `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: $N \rightarrow son[i] =$ `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_Implícita` 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_Implícita` é a seguinte: `double Evaluate(double x, double y);`

A implementação dessa função está no algoritmo 2.

Algorithm 2 `double Curva_Implícita::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 raíz 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_Implícita` é

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

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] ≠ NULL) then
2:    for (sum = 0.0, i = 0; i < 4; i++) do
3:       sum+ = son[i] → Evaluate(x, y, d, tw);
4:    end for
5:    return sum;
6: else
7:    w ← EvalWeight(x, y);
8:    if (w ≠ 0.0) then
9:       t ← Function(x, y, a, d);
10:      tw+ = w;
11:      return w × 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 $\mathbf{a}$.