Transformadas de Distância

Adelailson Peixoto
peixoto@inf.puc-rio.br

Luiz Velho
lvelho@visgraf.impa.br

PUC-Rio.Inf.MCC 35/00 Setembro, 2000

Resumo

O cálculo de transformadas de distância tem aplicações nas mais diversas áreas da Computação Gráfica. Uma das principais dificuldades no cálculo de funções distância está associada à passagem do universo matemático contínuo para o universo discreto. Isto porque um mesmo método, quando estudado no mundo discreto, pode apresentar propriedades completamente distintas daquelas apresentadas do ponto de vista contínuo. Este trabalho discute alguns dos principais métodos utilizados para o cálculo de transformadas de distância, dando ênfase principalmente às aplicações a dados matriciais (imagens e volumes). É feita também uma discussão sobre propagação de interfaces, como uma forma de calcular a transformada de distância.

Palavras Chave. funções implícitas, curvas de nível, propagação de interfaces, equações diferenciais, volumes, codificação de voxels.

Abstract

The computation of distance transforms has applications in several areas of Computer Graphics. One of the main difficulties in computing a distance function involves the transition between the continuous and discrete universes. When studied in the continuous universe, a method may present completely different properties than those exhibited in the discrete universe. This work presents a general framework about distance transforms, emphasizing mainly raster data (images and volumes). We will also discuss the interface propagation problem as an approach to distance transforms.

Keywords. implicit functions, level set, interface propagation, differential equations, volumes, voxel coding.

1-Introdução

Transformadas de Distância representam poderosas ferramentas utilizadas no processamento de *objetos gráficos* [6], nas mais diversas áreas da Computação Gráfica.

1.1-Objetivos

A transformada de distância *T*, aplicada a um objeto gráfico *O*, calcula um campo escalar (ou vetorial) que representa distâncias mínimas entre o objeto e os pontos do espaço no qual ele está envolvido. A transformada *T* pode ser definida da seguinte maneira:

$$T\ (O) = \min_{p_i \in O} dist(p, p_i),$$

onde *p* representa pontos arbitrários do espaço, e *dist* representa uma *função distância* ou *métrica* utilizada. Assim, para cada ponto *p* do espaço, a transformada calcula a distância de *p* ao ponto p_i (p_i pertence ao *suporte geométrico* [6] ou borda de *O*) que está mais próximo de *p*. É claro que com esta definição, para os pontos *p* situados na borda do objeto, tem-se *T(O)*=0. A figura 1 mostra a distância mínima entre um ponto *p* (interno ao objeto *O)* e a borda do objeto e a distância mínima entre um ponto *q* (externo) e o objeto.

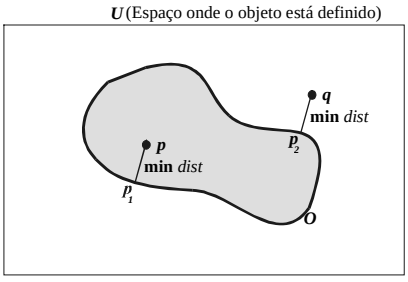


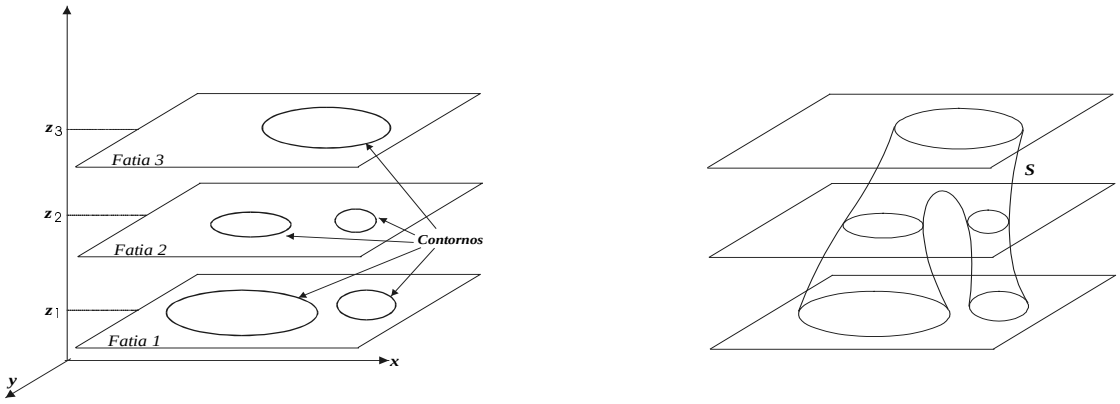
Figura 1: Distância mínima.

O resultado da transformada *T* aplicada a um objeto *O* depende da métrica ou função distância *dist*, como será visto mais adiante, no capítulo 2.

1.2-Aplicações

São inúmeras as aplicações que envolvem o uso de transformadas de distância, nas mais diversas áreas da Computação Gráfica. Em [4], o cálculo de campos distância é aplicado a metamorfose de objetos 3D. Dentre as diversas aplicações no processamento de dados volumétrcos, dois exemplos do uso de transformadas de distância são: em alguns métodos de reconstrução de superfícies a partir de um conjunto de seções bidimensionais e no cálculo de *esqueletos* de objetos volumétricos.

No primeiro exemplo funções distância são utilizadas como ferramentas para auxiliar na reconstrução de uma superfície *S*, a partir de um conjunto de contornos fechados situados em fatias paralelas (figura 2). Nos chamados *métodos implícitos* de reconstrução, a superfície *S* é definida como a isosuperfície *F=0*, onde *F* é uma função implícita que pode ser calculada a partir de funções distância definidas em cada fatia. Um *survey* dos métodos de reconstrução de superfícies pode ser encontrado em [12].



a) Contornos sobre as fatias b) Superfície reconstruída

Figura 2: Reconstrução de uma superfície a partir de contornos.

No segundo exemplo, funções distância são fundamentais para o cálculo de *esqueletos* (particularmente para o cálculo de *eixos mediais*) dos objetos. O esqueleto é formado pelos pontos internos que se encontram centralizados em relação à borda do objeto, ou seja, que possuem distância máxima da borda. Em [13] pode ser encontrado um *survey* com os principais métodos de extração de esqueletos de dados volumétricos. A figura 3, retirada de [19], mostra o esqueleto (pontos brancos centralizados) de um objeto volumétrico (intestino).

Figura 3: Esqueleto extraído de um objeto volumétrico.

2-Métricas

O resultado da transformada de distância aplicada a um objeto *O* depende de qual métrica ou função distância será utilizada. Dentre as várias métricas destacam-se: a métrica euclidiana, a *chessboard* e a *city block*.

2.1-Métrica Euclidiana

Na métrica euclidiana a função distância é definida como

$$dist(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2},$$

onde $p = (p_1, p_2, \ldots, p_n)$ e $q = (q_1, q_2, \ldots, q_n)$ são pontos do espaço *n*-dimensional.

Com esta definição a transformada *T* pode ser aplicada tanto a objetos do espaço contínuo quanto a objetos do espaço discreto (como é o caso dos objetos volumétricos). Apesar de ser a métrica ideal, pois trata das medidas reais dos espaços euclidianos, e consequentemente apresenta excelentes resultados nas aplicações, há algumas incoveniências computacionais com o uso da métrica euclidiana. O problema de utilizar a transformada de distância com tal métrica para objetos discretizados é que nem é algoritmicamente fácil implementá-la, nem seu cálculo computacional é eficiente, já que envolve o cálculo de quadrados e raízes. A figura 4 mostra o resultado da transformada de distância com a métrica euclidiana aplicada a um objeto 2D (uma imagem binária, onde pixels brancos, com valor 1, representam o objeto e pixels pretos, com valor 0, representam o *background*). O resultado (figura 4b) é uma outra imagem onde o valor de cada pixel representa sua distância euclidiana à borda do objeto.

a) Imagem binária original b) Tranformada distância com métrica euclidiana

Figura 4: Transformada de distância com métrica euclidiana.

Devidos às dificuldades de implementação e eficiência da métrica euclidiana, muitas aplicações a substituem por métrica regulares, como a *city block* e a *chessboard*, que possuem um cálculo computacional mais eficiente e são de fácil implementação [10]. A seguir estas métricas são definidas.

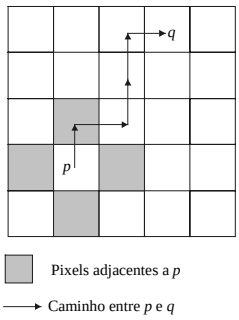
2.2-Métrica City Block

Nesta métrica, também conhecida como *métrica de Manhattan*, a função *dist* é definida como

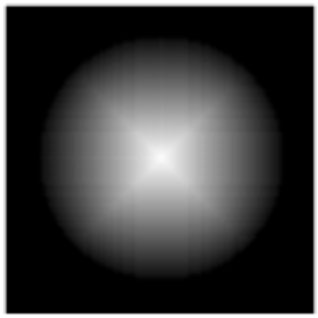
$$dist(p,q) = |p_1 - q_1| + |p_2 - q_2| + ... + |p_n - q_n|.$$

Assim a norma de um vetor é dada pela soma de suas componentes em cada eixo principal. A grande utilidade desta métrica surge quando aplicada a problemas em espaços discretados. Neste caso há uma interpretação interessante sobre a métrica *city block*: quando aplicada a objetos do espaço discreto, esta métrica assume que, para ir de um ponto *p* a um ponto *q*, só é possível andar nas direções dos eixos principais do sistema de coordenadas onde o objeto está definido (não é permitido andar nas direções diagonais). Esta observação fica clara quando o objeto considerado é uma imagem 2D ou um dado volumétrico. Conforme descrito abaixo, no caso de imagens esta métrica define a topologia *4-conectado* e no caso de volumes, define a topologia *6-conectado*.

No caso de imagens, onde um pixel *p* possui 8 pixels adjacentes (4 que compartilham uma aresta e 4 que compartilham um vértice com *p*) esta métrica determina que, saindo do pixel *p* só é possível andar nas direções dos pixels que compartilham uma aresta (figura 5a). Assim, se um pixel *q* compartilha uma aresta com *p*, então *dist*(*p*,*q*)=1 e *q* é um pixel vizinho a *p*. Se *q* compartilha um vértice com *p*, então *dist*(*p*,*q*)=2 (*q* não é vizinho de *p*). Por esta razão a métrica *city block* também é referenciada como métrica "1-2", no sentido de que, dado um ponto *p*, seus pixels adjacentes por aresta têm distância 1 (pixels vizinhos) e seus pixels adjacentes por vértices têm distância *2*. Como cada pixel possui 4 pixels vizinhos (por aresta), esta métrica define a topologia *4-conectado*. Para calcular a distância entre dois pixels quaisquer basta ir andando nas direções permitidas (horizontal e vertical) e contar a quantidade de pixels percorridos entre a origem e o destino. Na figura 5a a distância entre *p* e *q* é 5. A figura 5b mostra a transformada de distância usando a métrica *city block*, aplicada à imagem da figura 4a.



a) Pixels vizinhos: topologia 4-conectado



b) Campo distância gerado pela transformada

Figura 5: Transformada de distância usando a métrica *city block*.

No caso de um dado volumétrico, cada voxel pode conter até 26 voxels adjacentes (6 por faces, 12 por arestas e 8 por vértices). Partindo de um voxel *p* só é possível seguir nas

direções dos seus voxels adjacentes por face. Assim, se um voxel *q* compartilha uma face com *p*, então *dist*(*p*,*q*)=1 (*q* é vizinho de *p*). Se *q* compartilha uma aresta com *p*, então *dist*(*p*,*q*)=2 e se *q* compartilha vértice com *p*, então *dist*(*p*,*q*)=3. Como dois voxels só são vizinhos se compartilharem uma face, então diz-se que esta métrica define a topologia *6-conectado*. A figura 6 mostra um voxel com seus seis vizinhos. No caso 3D esta métrica é também conhecida como métrica "1-2-3", no sentido de que voxels que compartilham uma face têm distância 1 (voxels vizinhos), voxels que compartilham uma aresta têm distância 2 e voxels que compartilham um vértice têm distância 3.

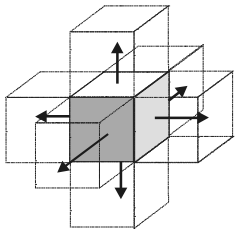


Figura 6: Métrica city block definindo topologia 6-conectado.

2.3-Métrica Chessboard

Na métrica *chessboard* a função *dist* é definida como

$$dist(p,q) = \max(|p_1 - q_1|, |p_2 - q_2|, \ldots, |p_n - q_n|).$$

Assim a norma de um vetor é definida como sendo a sua maior componente. Assim como a métrica *city block*, a maior motivação para o uso da métrica *chessboard* está voltada às aplicações discretas, uma vez que ambas procuram substituir a métrica euclidiana. A interpretação desta métrica, quando aplicada a objetos do espaço discreto, é que, para ir de um ponto *p* a um ponto *q*, é permitido se deslocar em todas as direções. Conforme descrito abaixo, no caso de imagens esta métrica define a topologia *8-conectado* e no caso de volumes, define a topologia *26-conectado*.

A principal motivação para o nome dessa métrica vem das aplicações discretas 2D, onde, saindo de um pixel *p* é possível fazer os movimentos que um rei faz em um tabuleiro de xadrez. Portanto é permitido andar tanto nas direções dos eixos principais (horizontal e vertical) quanto nas direções diagonais (figura 7a). Devido a isto, um pixel *p* possui 8 pixels vizinhos: os 4 vizinhos por aresta (direções horizontal e vertical) e os 4 vizinhos por vértices (direções diagonal). Diz, então que esta métrica define a topologia *8-conectado*. No caso 2D a métrica *chessblock* é também chamada de métrica "1-1", no sentido de que, dado um ponto *p*, tanto os pixels que compartilham arestas quanto os que compartilham vértices com *p* possuem distância 1. A figura 7b mostra a transformada de distância usando a métrica *chessboard*, aplicada à imagem da figura 4a.

No caso 3D discreto, partindo de um voxel *p* é permitido seguir em qualquer direção, ou seja, nas direções dos 6 voxel que compartilham uma face, na direção dos 12 voxels que

compartilham uma arestas e na direção dos 8 voxels que compartilham um vértice, totalizando 26 direções possíveis. Portanto, qualquer vizinho (por face, aresta ou vértice) é vizinho a *p* (ou seja possui distância 1). Diz-se então que, a métrica *chessblock* define a topologia *26-conectado*. No caso 3D esta métrica é também chamada de métrica "1-1-1", pois os vizinhos de um voxels *p* possuem distância 1 em relação aos vizinhos por faces, por arestas e por vértices.

a) Pixels vizinhos: topologia 8-conectado b) Campo distância gerado pela transformada.

Figura 7: Transformada de distância usando a métrica *city block*.

2.4-Métrica "n_f - n_a - n_v"

A métrica "1-2-3" (*city block* 3D discreta) e a métrica "1,1,1" (*chessblock* 3D discreta) podem ser vistas como casos particulares da métrica discreta "n_f-n_a-n_v", onde, dado um voxel *p*, n_f representa a distância entre *p* e seus voxels adjacentes por face, n_a representa a distância entre *p* e seus voxels adjacentes por aresta e n_v é a distância entre *p* e seus voxels adjacentes por vértices. É claro que no caso 2D a métrica "n_f-n_a-n_v" é referenciada simplesmente como métrica "n_a-n_v".

Dois exemplos de métricas "n_f-n_a-n_v" bastante utilizadas são a métrica "2-3-4" [5] e a métrica "3-4-5" [2].

2.5-Codificação de Voxels

Nas aplicações a imagens binárias, a transformada de distância resulta em uma nova imagem onde o valor de cada pixel indica sua distância a um determinado conjunto de pixels iniciais da imagem. Nas figuras 4b, 5b e 7b o conjunto de pixels iniciais é a borda da circunferência definida na figura 4a. Da mesma forma, quando a transformada é aplicada a um volume binário, o resultado também é um volume, onde o valor de cada voxel contém informação do campo distância gerado. A aplicação da transformada aos voxels do volume é feita através de um processo chamado *codificação de voxels* (no caso de uma imagem, chama-se codificação de pixels, porém o processo é o mesmo e será generalizado como codificação de voxels).

Codificação dos voxels de um objeto volumétrico *O* é uma operação, definida a partir de uma métrica, que se propaga recursivamente voxel a voxel do volume. Esta operação começa a ser aplicada em um conjunto inicial de voxels V_0 ($V_0 \subset O$) e se espalha pelos demais voxels de *O*, até que uma condição de parada seja atingida. Após a codificação dos voxels do volume ser gerada, tem-se então o campo distância definido.

No caso de uma imagem, a codificação se dá por um processo de propagação, semelhante à evolução de uma frente em chamas, que avança sobre uma região coberta de gramas. O conjunto V_0 pode ser comparado à borda da região, onde alguém pôs fogo (frente inicial) e a partir daí a chama vai avançando (frente evoluindo). Uma vez que um pixel é visitado, um valor (código do pixel) é associado a ele, indicando sua distância ao conjunto inicial V_0 (frente inicial). Assim, a codificação de voxels pode ser interpretada como a evolução de uma interface (por exemplo, o fogo) sobre um meio (por exemplo, a grama). Esta comparação, estudada mais adiante, será útil na tentativa de encontrar novas soluções para os problemas que envolvem transformada de distância.

A operação de propagação de voxels usando a métrica "n_f-n_a-n_v" pode ser descrita como se segue: primeiro, todos os voxels do objeto *O* são codificados com o valor infinito, em seguida todos os voxels do conjunto V_0 são codificados com o valor zero (início da propagação) (figuras 8a e 9a). A todos os vizinhos dos voxels de V_0 por faces é associado o valor n_f, a todos os vizinhos por arestas é associado o valor n_a e a todos os vizinhos por vértices é associado o valor n_v (figuras 8b e 9b). Durante a propagação, todos os voxels com um determinado código *n* são processados ao mesmo tempo. Assim, se voxels com valor *n* são processados, aos seus vizinhos por face, por aresta e por vértice são associados os valores $n+n_f$, $n+n_a$ e $n+n_v$, respectivamente, caso estes valores sejam menores do que os valores correntes dos voxels vizinhos. Este processo de codificação continua até que sejam atingidas as condições de parada.

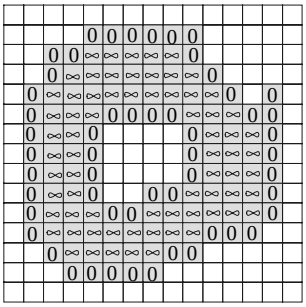
A escolha do conjunto de voxels iniciais V_0 depende das características que se deseja extrair da codificação. A seguir são escolhidos dois conjuntos distintos para V_0, que resultam em campos escalares com características diferentes: *Boundary Field* e *Single Field* [19].

Boundary Field. O conjunto V_0 é composto pelos voxels que formam a borda do objeto. A codificação gerada nos voxels forma um campo escalar distância tradicional, chamado *Bounday Field* (este campo foi gerado nas figuras 4b, 5b e 7b, com as métricas euclidiana, city block e chessboard, respectivamente). O código gerado para cada voxel interno indica sua distância à borda do objeto e será chamado *boundary-code*. Os voxels centralizados, em relação ao objeto, possuem código máximo. Estas informações são fundamentais para a extração de esqueletos de objetos volumétricos [13]. A figura 8c mostra um exemplo de boundary field, com a métrica "3-4".

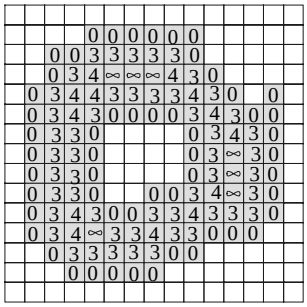
É importante observar que a métrica selecionada, ou seja a escolha dos valores de n_f, n_a e n_v, influencia no campo distância gerado. Alguns trabalhos que utilizam o boundary field,

para extração de esqueletos de dados volumétricos, podem ser encontrados em mostrados em [18] e [19].

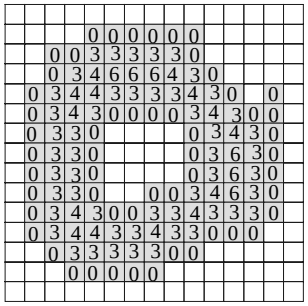
Single Field. O conjunto V_0 é formado por um único voxel inicial v_0. A codificação gerada nos voxels forma um campo escalar distância, chamado *Single Field*. O código gerado para cada voxel interno indica sua distância ao voxel inicial v_0 e será chamado *single-code*. A figura 9c mostra um exemplo de single field, com a métrica "1-2". Se o objeto é formado por partes desconexas, é necessária a escolha de um ponto inicial v_0 para cada parte desconexa.



a) Pontos do objeto (∞'s) e conjunto inicial V_0 (0's)



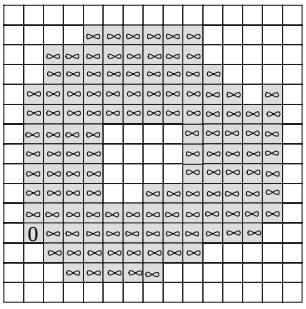
b) Resultado do processamento dos pixels de código 0.



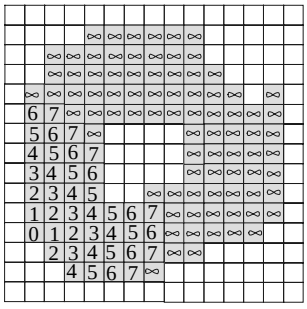
c) Campo distância final

Figura 8: Campo escalar *Boundary Field*, usando a métrica "3,4".

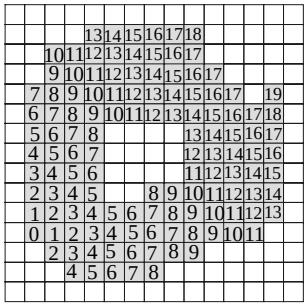
O campo single field pode ser utilizado, por exemplo, para extração do menor caminho entre dois voxels (ou pixels) v_1 e v_2, como mostrado em [18]. Este problema envolve duas etapas: a primeira etapa é a geração do single field, utilizando v_2 como ponto inicial (ou seja $v_2 = v_0$). A segunda etapa extrai o caminho mais curto: v_1 é escolhido como o primeiro voxel do caminho e o próximo voxel escolhido é o vizinho de v_1 que contém o menor código. Recursivamente, o próximo voxel é escolhido de maneira semelhante, como sendo o vizinho, com menor código, do voxel escolhido anteriormente. O último voxel escolhido será exatamente o v_2, que possui o menor single-code, ou seja, *0*.



a) Pontos do objeto (∞'s) e conjunto inicial V_0 (0)



b) Resultado após o processamento dos pixels de código 5.



c) Campo distância final

Figura 9: Campo escalar *Single Field*, usando a métrica "1,2".

3-Propagação de Interfaces

Como visto no capítulo anterior, a codificação de voxels, durante o cálculo da transformada de distância sobre um objeto matricial, pode ser comparada a uma frente em chamas se propagando sobre uma região coberta de gramas. Esta comparação pode ser bastante útil, no sentido de que o cálculo da transformada de distância pode recorrer à Teoria de Evolução de Interfaces (a frente em chamas se propagando é uma interface), numa tentativa de encontrar novas soluções para suas aplicações. Este capítulo relaciona propagação de interfaces a transformadas de distância, mostrando quais as vantagens de se usar esta abordagem para o cálculo de campos escalares distância.

3.1- Propagação de Interfaces e Campos Distância

Baseado na Teoria das Leis da Conservação Hiperbólica (Apêndice), Sethian [15][16] desenvolveu alguns métodos numéricos eficientes para a análise e cálculo de propagação de interfaces: o Método *Level Set* e o Método *Fast Marching*, que serão estudados adiante.

O método *Level Set* pode ser aplicado a propagação de interfaces de forma mais genérica, enquanto o método *Fast Marching* é aplicado a casos mais específicos de propagação, dentre eles no caso em que a interface em evolução é uma função distância. Uma das grandes vantagens da utilização destes métodos é que eles são baseados em formulações contínuas e, portanto, o método *Fast Marching* pode ser utilizado para propagar campos escalares distância em objetos contínuos.

Os campos distância *Boundary Field* e *Single Field* (seção 2.5) são decorrentes da métrica "n_f-n_a-n_v", ou seja, são aplicados a objetos já discretizados. Isto dificulta a utilização destes métodos, principalmente nos problemas que envolvem reconstrução dos objetos, uma vez que o ponto de partida já foi um objeto discreto. Como o método *Fast Marching* é definido a partir de formulações contínuas, ele é muito útil para o cálculo de campos distância onde é necessária a reconstrução do objeto. A seguir serão vistas as formulações de propagação de interfaces que originam o Método *Fast Marching*.

3.2- Formulações da Propagação de Interfaces

Uma interface pode ser geometricamente considerada uma curva ou uma superfície que separa dois meios que estão interagindo entre si. Ou seja, a interface diz respeito à borda ou fronteira que separa os dois meios. Suponha que a interface esteja se movendo em direção a sua normal, com uma dada velocidade *F*. A figura 10 mostra uma interface separando dois meios. A curva circular pode ser considerada, por exemplo, um ácido corroendo um material (região entre o retângulo e o círculo), onde o interior do círculo representa ausência de material. A velocidade da corrosão depende da resistência que o

ácido encontra, ou seja, nas partes mais resistentes a velocidade da corrosão é menor do que em outros locais.

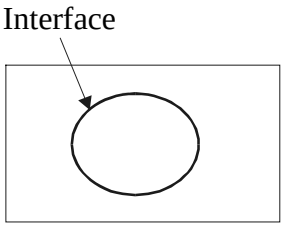


Figura 10: Interface separando dois meios.

Outro exemplo é que o círculo pode ser considerado uma interface que separa duas regiões: a região 1 (interior do círculo), formada pelos pontos cuja distância ao centro do círculo é menor ou igual ao seu raio e região 2 (entre o circulo e o retângulo), formada pelos pontos cuja distância ao centro do círculo é maior que seu raio. O aumento do raio do círculo significa que a região 1 está se propagando sobre a região 2.

De um modo geral, a velocidade de propagação *F* pode depender de vários fatores:

- Propriedades Locais – são aquelas que dependem da geometria local à curva, como curvatura, vetor normal, etc.
- Propriedades Globais – são aquelas que dependem da forma, posição e características específicas de uma determinada interface.
- Propriedades Independentes – são aquelas que dependem do posicionamento da interface, como por exemplo, um fluido no qual a interface está sendo conduzida.

Como este trabalho usa propagação de interfaces como uma forma de propagar distâncias, serão considerados apenas os casos em que a velocidade da propagação depende apenas de propriedades locais, como curvatura e normal (mais especificamente, os caso onde a velocidade é constante, conforme será visto posteriormente).

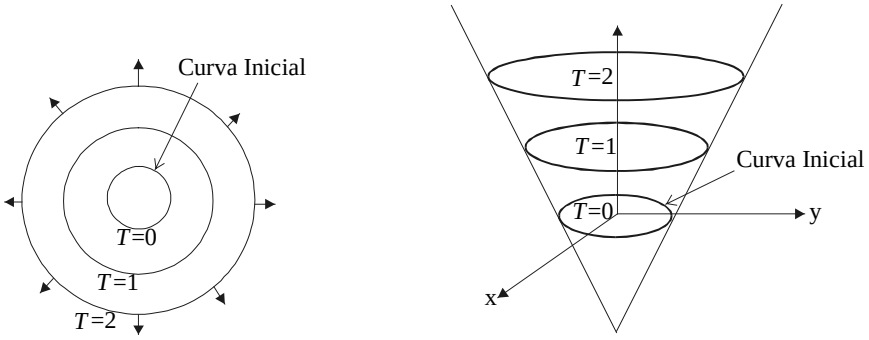
Existem duas maneiras básicas de formular o problema de evolução de interfaces: formulação do valor de borda e formulação do valor inicial, usadas para definir, respectivamente, o Método *Fast Marching* e o Método *Level Set*.

3.2.1 – Formulação do Valor de Borda

No caso de a interface da figura 10 ser considerada um ácido corrosivo, à medida que o tempo passa, a tendência é que o tamanho do círculo aumente, uma vez que a corrosão é feita apenas no sentido círculo-retângulo. No caso de o exemplo representar a propagação do campo distância, quanto mais o círculo avança sobre o retângulo, mais o campo distância se propaga. Nestes exemplos, o sentido de propagação da interface é sempre o mesmo, ou seja, a velocidade não muda de sinal, é sempre positiva. Com isto, a cada instante a interface ocupa uma nova posição, uma vez que está sempre avançando. Assim pode-se formular uma função *T* que associa a cada posição do espaço uma nova curva.

A formulação do valor de borda pode ser colocada da seguinte forma: Assim, partindo de uma curva ou interface inicial (instante inicial zero), a cada instante *T* a interface vai evoluindo, ocupando uma nova posição no espaço (figura 11a), ou seja, há um tempo *T* associado a cada nova interface resultante da evolução. Com isto cada curva pode ser vista como uma curva de nível de uma função tempo *T* (figura 11b).

A motivação do nome formulação do valor de borda surge do fato de que, a cada instante *T* que se deseja saber onde a interface se encontra, basta tomar a borda da superfície *T*(*x*,*y*) na altura *T*, conforme mostra a figura 11b.



a) Evolução da curva em cada instante b) Evolução vista como uma superfície T(x,y)

Figura 11: Propagação de interface vista com a formulação do valor de borda.

No caso unidimensional a função *T* é facilmente deduzida. Como em uma dimensão *distância = velocidade * tempo*, então 1= *F dT*/*dx*. Esta notação pode ser estendida para múltiplas-dimensões como

$$|\nabla T| F = 1.$$

Esta equação é chamada de equação “Eikonal”. Esta formulação define o método *Fast Marching* (seção 3.3), usado para resolver numericamente a propagação de interfaces, no caso de a velocidade ser positiva. No caso de a velocidade ser constante, este método pode ser aplicado para calcular a propagação do campo distância.

3.2.2 – Formulação do Valor Inicial

A formulação do valor inicial é aplicada quando a velocidade de propagação da interface pode alterar o sinal. Por exemplo, na figura 10, seja o interior do círculo considerado um bloco de gelo dentro de um recipiente com água (entre o círculo e o retângulo). A borda do gelo (interface de interação) pode diminuir se a temperatura da água aumentar e pode aumentar caso a temperatura diminua, ou seja, a velocidade de propagação da interface depende da diferença de temperatura entre o gelo e água. Neste caso a interface pode

avançar ou recuar, ou seja, a velocidade de propagação pode ser positiva ou negativa. Portanto esta formulação é mais genérica do que a do valor de borda.

Como a interface pode avançar ou recuar, é possível que ela passe pela mesma posição em instantes diferentes. Portanto não é possível definir uma função temporal *T* que associa a pontos do espaço uma interface.

Na formulação do valor inicial a interface original é considerada o conjunto de nível zero de uma função Φ. A evolução da interface então é associada a variações aplicadas à função Φ. Assim, como Φ também varia com o tempo, ela é definida em função dos parâmetros da interface e em função do tempo. Por exemplo, se a interface é uma curva evoluindo no plano, a função Φ é definida como Φ(*x*,*y*,*t*), onde *t* representa o eixo do tempo. Para aplicar uma propagação à curva no instante *t* basta calcular Φ, neste instante *t*, e tomar sua curva de nível zero, que sempre irá corresponder à curva evoluída (figura 12).

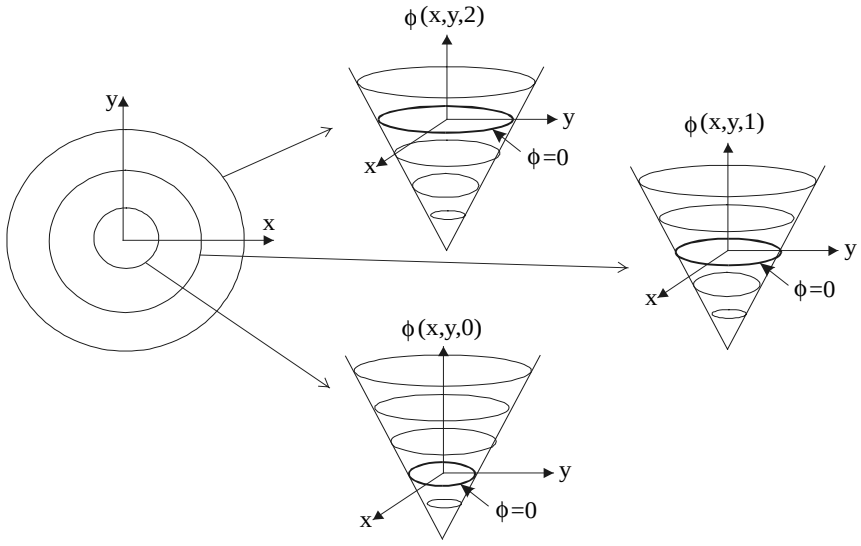


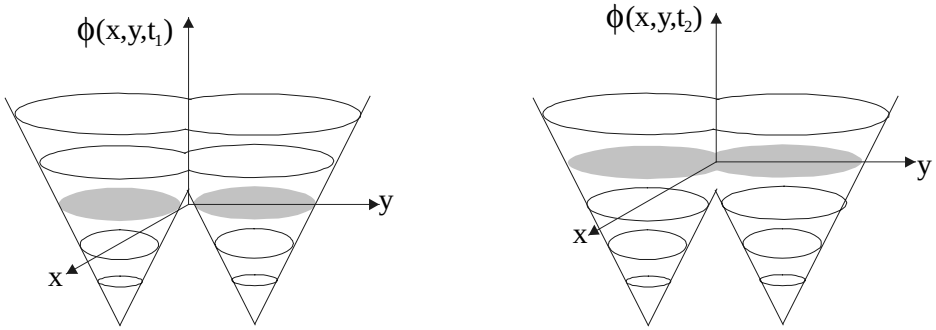
Figura 12: Propagação de interface vista com a formulação do valor inicial.

À primeira vista, pode parecer incoerência transformar um problema de propagação de curva em um problema de propagação de superfície (Φ). A questão é que a função Φ será sempre bem comportada, mesmo nos casos onde a interface, ao evoluir, altera completamente sua topologia, como se dividir em duas novas curvas ou se houver fusão de duas curvas em uma, como mostra a figura 13. Para acessar a curva evoluída em qualquer instante *T*, basta tomar a curva de nível zero de Φ(*x*,*y*,*T*).

Matematicamente a formulação do valor inicial pode ser definida a partir da equação Φ(*x*(*t*), *t*)=0 (onde *x*(*t*) representa o espaço de propagação da interface em qualquer dimensão), já que a interface evoluída corresponde aos valores onde Φ se anula. Derivando esta equação, pela regra da cadeia, então $\Phi_t + \Phi(x(t),t).x'(t) = 0$. Como $F=x'(t).n$ e $n=\nabla\Phi/|\nabla\Phi|$, então

$$\Phi_t + F|\nabla\Phi| = 0,$$

Esta formulação é utilizada para definir o Método *Level Set*. Como este método não é aplicado a propagação de campos distâncias, este trabalho não fará uma abordagem sobre o mesmo. O leitor interessado pode consultar a referência [16].



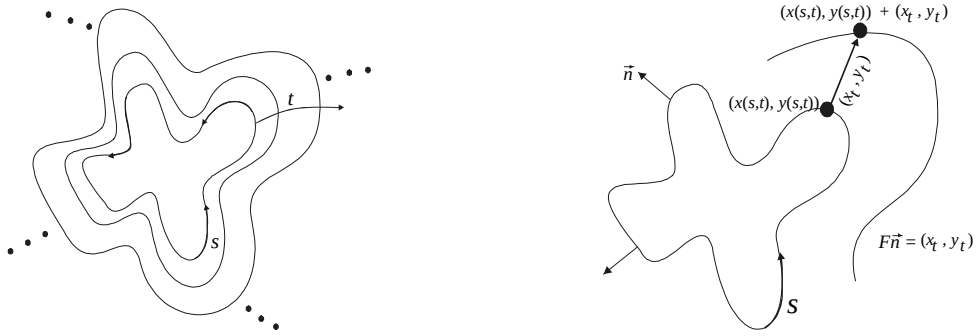
a) Instante t_1: interface contém duas partes b) Instante t_2: a interface se propagou e contém uma parte

Figura 13: Mudança topológica de uma interface durante sua propagação.

3.3- Evolução de Curvas

Nesta seção serão discutidos alguns aspectos da velocidade de propagação de interfaces e como esta velocidade deve ser formulada, de modo que a propagação possa ser utilizada para o cálculo de distâncias.

Seja uma curva paramétrica $\varphi(s) = (x(s), y(s))$, simples e suave. Considere-se que a curva esteja movendo em direção a sua normal, com uma velocidade F. O objetivo é descrever o movimento da curva durante a sua evolução. A curva $\varphi(s)$ pode ser considerada como uma interface que separa dois meios. A evolução da curva pode ser definida a partir da parametrização dada por $\varphi(s,t)$, onde s é o parâmetro da curva ($0<s<S$) e t é o parâmetro da evolução (figura 14a). Sem perda de generalidade, a teoria de evolução de curvas pode ser estendida para evolução de superfícies.



a) Parametrização da evolução da curva b) Cálculo da evolução

Figura 14: Evolução da curva.

Durante a evolução, para cada valor de t, há uma nova curva $\varphi(s,t)$ gerada pela parametrização da evolução. No instante t da evolução, cada ponto $\varphi(s,t) = (x(s,t), y(s,t))$ sofrerá um deslocamento através do vetor $(x_t(s,t) , y_t(s,t)) = F(k(s,t))\, n(s,t)$, ou seja:

onde $k = (y_{ss}x_s - x_{ss}y_s) / (x_s^2 + y_s^2)^{3/2}$ é a expressão paramétrica da curvatura, dentro da função velocidade $F(k)$ e o vetor normal é dado por $n = (y_s , -x_s) / (x_s^2+y_s^2)^{3/2}$. Assim, a nova posição do ponto $(x(s,t), y(s,t))$ será $(x(s,t), y(s,t)) + (x_t,y_t)$ (figura 14b). Esta formulação é uma representação Lagrangeana do movimento da curva, pois os valores de $(x(s,t), y(s,t))$ descrevem tal movimento.

$$y_t = -F\left[\frac{y_{ss}x_s - x_{ss}y_s}{(x_s^2 + y_s^2)^{3/2}}\right]\left(\frac{x_s}{(x_s^2 + y_s^2)^{1/2}}\right),$$

$$x_t = F\left[\frac{y_{ss}x_s - x_{ss}y_s}{(x_s^2 + y_s^2)^{3/2}}\right]\left(\frac{y_s}{(x_s^2 + y_s^2)^{1/2}}\right),$$

Quando a velocidade de propagação F é constante, todos os pontos da curva se deslocam de uma mesma distância. Assim, todos os pontos da nova curva evoluída situam-se a uma mesma distância da curva anterior. É justamente essa situação que interessa para o cálculo de propagação de distâncias. Portanto, nos métodos para propagação de interfaces, a velocidade F será constante.

Exemplo1. Durante a evolução, uma curva pode, em tempo finito, perder sua suavidade. Por exemplo [16], seja a curva definida por $\varphi(s,0) = (1\text{-}s, (1+\cos 2\pi s)/2)$ se propagando com velocidade constante F=1. Em cada instante t, a solução pode ser obtida avançando-se cada ponto da curva a uma distância t, na direção da sua normal, ou seja:

$$x(s,t) = \frac{y_s(s,t=0)}{(x_s^2(s,t=0) + y_s^2(s,t=0))^{1/2}} t + x(s,t=0),$$

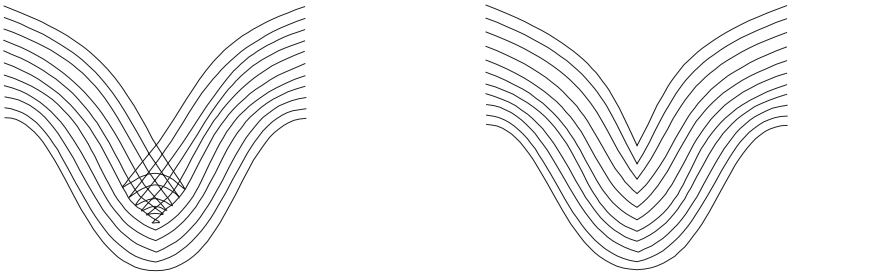
$$y(s,t) = \frac{-x_s(s,t=0)}{(x_s^2(s,t=0) + y_s^2(s,t=0))^{1/2}} t + y(s,t=0).$$

Como mostrado na figura 15a, ao evoluir, a frente apresenta pontos não diferenciáveis (“bicos”), onde a normal não está definida e, portanto não está claro como a evolução deve prosseguir. Nestes pontos a frente desenvolve uma “cauda” em tempo finito. Dependendo da aplicação de evolução de curvas, o resultado da figura 15a pode ser considerado correto, porém, como o objetivo é usar propagação de interfaces para cálculo de distâncias, em um determinado instante t, a frente deve ser formada apenas pelo conjunto de pontos localizados a uma distância t da curva inicial e, portanto a solução correta seria considerar a evolução da figura 15b. Portanto, de um modo geral, a solução correta depende da natureza da interface em discussão. Como o objetivo deste trabalho é usar evolução como propagação de distância, situações ambíguas como da figura 15a devem ser evitadas.

Como encontrar então uma solução sem ambigüidade, como a da figura 15b, uma vez que as fórmulas de evolução, descritas acima, levam naturalmente à solução ambígua (da figura 15a)? Sethian [14] definiu e estabeleceu uma *condição de entropia* para resolver tal problema. “Entropia” se refere à organização das informações da interface, à medida que ela evolui. Em termos gerais, uma condição de entropia é utilizada para que nenhuma nova informação (como a cauda da figura 15a) possa ser criada, durante a evolução.

Direcionando o problema de evolução de curvas para o cálculo de distâncias, a questão agora é saber sob que condições (condições de entropia) uma curva, se propagando com velocidade constante, fornece a solução correta para a propagação de distância (já que os cálculos acima levam à solução errada da figura 15a). A próxima seção e o apêndice descrevem tais condições.

Para aplicar a condição de entropia, Sethian recorreu às leis da conservação hiperbólica [7][8], como mostra o apêndice.



a) Evolução com criação de novas informações b) Evolução com condição de entropia

Figura 15: Propagação da curva com velocidade unitária.

3.3.1 – Efeitos da Curvatura na evolução

Como dito antes, serão considerados apenas os casos de evolução onde F é constante. Porém, para tentar resolver o problema dos “bicos”, surgidos no Exemplo 1, é importante analisar um caso mais geral, onde a velocidade depende da curvatura e é dada por $F(k)=1-\varepsilon k$ (ε é uma constante real e $\varepsilon \geq 0$).

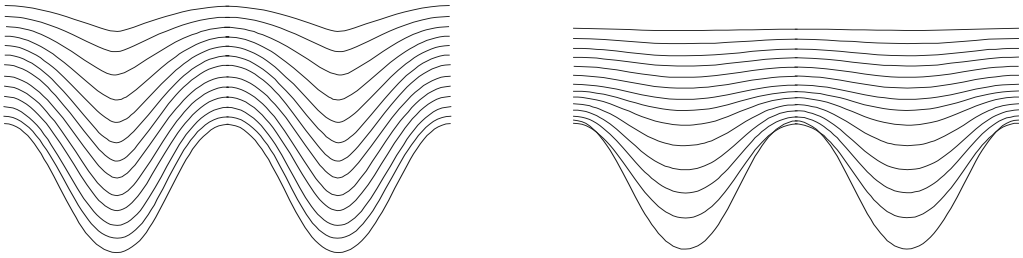
Se $\varepsilon=0$, a velocidade é constante (como no Exemplo 1, onde $F(k)=1$) e a curva não se mantém suave durante a evolução, ou seja, em algum momento perde a diferenciabilidade, desenvolvendo “bicos”. Como nestes bicos o vetor normal não está definido, não está claro como a evolução deve prosseguir e podem aparecer novas informações (como a cauda), e seria necessária a condição de entropia para que a curva evolua corretamente.

Seja agora $\varepsilon>0$, ou seja, $F(k) = 1-\varepsilon k$ e não é constante. A inclusão do termo εk altera profundamente a forma como a curva evolui e é este termo que dirá como esquemas numéricos eficientes serão construídos, para auxiliar na correta condição de entropia.

Para uma melhor compreensão dos efeitos da curvatura, analisemos a equação diferencial (evolução da curvatura) $k_t = \varepsilon k_{\theta\theta} + \varepsilon k^3 - k^2$ [14], que descreve como a curvatura se comporta durante a evolução da curva. A segunda derivada de k é tomada em relação ao comprimento de arco θ. Segundo Sethian [14], o termo $(\varepsilon k^3 - k^2)$ é responsável pelas singularidades (bicos) geradas na curva, porém é balanceado pelo termo $(\varepsilon k_{\theta\theta})$, que é responsável por suavizar a curva.

Exemplo 2. Seja a evolução da função cosseno, definida no Exemplo 1. Suponha agora que a função velocidade seja $F(k) = 1- \varepsilon k$. Considerando $\varepsilon=0$, a curva vai evoluir de maneira análoga à figura 15a, ou seja, vai desenvolver "bicos", pois $F(k)=1$. Isto ocorre por que, como $\varepsilon=0$, a equação de evolução da curvatura contém apenas o termo que cria singularidades (a equação de evolução será $k_t = - k^2$).

Considerando agora $\varepsilon>0$, a equação de evolução da curvatura conterá também o termo $(\varepsilon k_{\theta\theta})$ que suaviza a curva durante a evolução. A figura 16a mostra a evolução da curva cosseno com a velocidade $F(k) = 1 - 0.025k$ e a figura 16b mostra a mesma curva com velocidade $F(k) = 1 - 0.25k$.



a) $F(k) = 1 - 0.025k$ b) a) $F(k) = 1 - 0.25k$

Figura 16: Propagação onde a curva se mantém suave.

Assim, pode-se concluir que quanto menor o valor de ε menor será a contribuição do termo $(\varepsilon k_{\theta\theta})$ e, portanto, a evolução tende a ser menos suave (figura 16a). Quanto maior o valor de ε, maior será a contribuição do termo $(\varepsilon k_{\theta\theta})$ e mais suave será a evolução (figura 16b). Em [14] Sethian mostra que quando $\varepsilon>0$, a curva se mantém infinitamente diferenciável, ou seja, C^{∞}.

Nenhuma das duas condições ($\varepsilon=0$ e $\varepsilon>0$) pode ser utilizada para calcular distâncias a partir de evolução de curvas, pois se $\varepsilon=0$, a curva desenvolve "bicos"(figura 15a) e se $\varepsilon>0$, a velocidade não é constante (figura 16). A correta solução pode, então, ser postulada pela condição abaixo:

Solução de Entropia: o limite de uma propagação com curvatura (onde $F(k) = 1- \varepsilon k$), quando $\varepsilon \rightarrow 0$, é igual a uma propagação com velocidade constante ($F(k) = 1$). Assim, para se obter a evolução da figura 15b, através da condição de entropia, deve-se usar a função velocidade $F(k) = 1 - \varepsilon k$, fazendo ε tender a 0.

4-Métodos Numéricos: *Fast Marching*

Este capítulo discute métodos numéricos para propagar interfaces. Dentre estes métodos, destaca-se o *Fast Marching* que, além de ser bastante eficiente, pode ser aplicado em propagação de distâncias, quando a propagação é feita com velocidade constante. Por fim é mostrado um exemplo [3], que utiliza o Método *Fast Marching* com este fim.

4.1-Alguns Métodos Numéricos

Os métodos numéricos para evolução surgem a partir de suas formulações no universo contínuo. Assim, as formulações mostradas nas seções 3.2.1, 3.2.2 e 3.3 são utilizadas para definir alguns destes métodos. É importante destacar que a presente seção procura discutir a importância destes métodos para o cálculo de propagação de distâncias.

4.1.1 – Formulação Lagrangeana

Na formulação Lagrangeana (seção 3.3), considerando a propagação com velocidade constante (F=1), a evolução é calculada através das equações

$$x_t = \frac{y_s}{(x_s^2 + y_s^2)^{1/2}}, \qquad y_t = -\frac{x_s}{(x_s^2 + y_s^2)^{1/2}},$$

Como a velocidade é constante, esta evolução produz um campo distância. Para definir um método numérico de evolução basta discretizar estas equações. O intervalo de parametrização da curva $[0,S]$ é dividido em M intervalos de tamanho Δs, produzindo M+1 amostras $s_i = i.\Delta s$ (i=0, …, M). Também é feita uma discretização do tempo em intervalos de tamanho Δt. O valor da curva em movimento, em cada ponto $i.\Delta s$ e no instante $n.\Delta t$, é dado por $\varphi_i^n=(x_i^n,y_i^n)$. O objetivo é numericamente calcular os novos valores (distâncias) de $\varphi_i^{n+1} = (x_i^{n+1}, y_i^{n+1})$, produzidos pela evolução.

As derivadas parciais em relação a s podem ser numericamente resolvidas como (central difference):

$$x_s \approx (x_{i+1}{}^n - x_{i-1}{}^n)/2\Delta s, \qquad y_s \approx (y_{i+1}{}^n - y_{i-1}{}^n)/2\Delta s,$$

As derivadas parciais em relação ao tempo podem ser aproximadas por (forward difference): $x_t \approx (x_i^{n+1} - x_i^n)/\Delta t$ e $y_t \approx (y_i^{n+1} - y_i^n)/\Delta t$.

Substituindo estas derivadas nas fórmulas de evolução (x_t,y_t) acima, e rearrumando os termos, obtém-se a expressão numérica para o cálculo da evolução:

$$(x_i^{n+1}, y_i^{n+1}) = (x_i^n, y_i^n) + \Delta t \frac{(y_{i+1}^n - y_{i-1}^n, x_{i-1}^n + x_{i+1}^n)}{((x_{i+1}^n - x_{i-1}^n)^2 + (y_{i+1}^n - y_{i-1}^n)^2)^{1/2}},$$

A expressão numérica que calcula a evolução pode agora ser aplicada para calcular a nova distância (x_i^{n+1},y_i^{n+1}) a partir de pontos (x_i^n, y_i^n). O problema maior na aplicação deste método está na instabilidade dos resultados. Se a distância Δs é muito pequena as amostras da curva ficam muito próximas e o quociente do segundo membro da equação da evolução pode ficar muito próximo a zero. Isto causa uma grande instabilidade numérica na propagação da distância. A próxima etapa da propagação, então, já seria calculada sobre estes erros, causando uma drática propagação do erro. É importante lembrar também que, como não está sendo aplicada nenhuma condição de entropia, este método ainda pode gerar resultados semelhantes ao da figura 15a, o que tornaria errada a solução da propagação de distância.

4.1.2 – Método *Level Set*

O método *Level Set* é aplicado aos casos onde podem ocorrer mudanças na sinal da velocidade. Por esta razão ele não é aplicado ao cálculo de propagação de distâncias. Apesar disto, alguns trabalhos, como [3], utilizam o *Level Set* como uma etapa de pré-processamento para o cálculo de distância. Por isto serão feitos alguns comentários sobre este método.

O método numérico *Level Set* foi desenvolvido a partir da formulação do valor inicial, apresentada na seção 3.2.2, onde a interface em evolução é sempre o conjunto de nível zero de uma função Φ, que também evolui em função do tempo e a equação de evolução é dada por $\Phi_t + F|\nabla\Phi|=0$.

Esta equação é um caso particular da equação de Hamilton-Jacobi $\alpha U_t + H(U_x,U_y,U_z) = 0$, onde $\alpha=1$ e $H(\nabla\Phi)=F|\nabla\Phi|$. Conforme mostrado no apêndice, a equação de Hamilton-Jacobi pode ser resolvida numericamente a partir de métodos de leis de conservação hiperbólica [16]. A partir da solução da equação de Hamilton-Jacobi, que já inclui as condições de entropia, o Método *Level Set* é definido através da equação:

$$\Phi_{ijk}^{n+1}=\Phi_{ijk}^{n} - \Delta t.[\max(F_{ijk},0)\nabla^{+} + \min(F_{ijk},0)\nabla^{-}], \text{ onde}$$

$$\nabla^{+} =[\max(D_{ijk}^{-x},0)^2 +\min(D_{ijk}^{+x},0)^2 +\max(D_{ijk}^{-y},0)^2 +\min(D_{ijk}^{+y},0)^2 +\max(D_{ijk}^{-z},0)^2 +\min(D_{ijk}^{+z},0)^2]^{1/2},$$

$$\nabla^{-} =[\max(D_{ijk}^{+x},0)^2 +\min(D_{ijk}^{-x},0)^2 +\max(D_{ijk}^{+y},0)^2 +\min(D_{ijk}^{-y},0)^2 +\max(D_{ijk}^{+z},0)^2 +\min(D_{ijk}^{-z},0)^2]^{1/2}$$

e $D_{ijk}^{-x} = (\Phi_{ijk} - \Phi_{i-1,j,k})/\Delta x$ e $D_{ijk}^{+x} = (\Phi_{i+1,jk} - \Phi_{ijk})/\Delta x$.

D_{ijk}^{-y}, D_{ijk}^{+y}, D_{ijk}^{-z} e D_{ijk}^{+z} são definidos de maneira análoga.

Neste método é definida uma grade, onde o valor associado a cada elemento da grade representa o valor da função Φ, em um determinado instante da evolução. Em um instante posterior, para aplicar a evolução, todos os valores da grade são atualizados. A interface, alvo da propagação propriamente dita, é sempre representada pelos pontos da grade, onde $\Phi=0$. A figura 17 mostra a evolução de uma curva φ_{ij}^n no plano. Os valores associados aos pontos (i,j) da grade representam curvas de nível da função Φ_{ij}^n (daí o

nome deste método ser *Level Set*), de modo que a curva φ_{ij}^{n} é sempre representada pela curva de nível $\Phi_{ij}^{n}=0$.

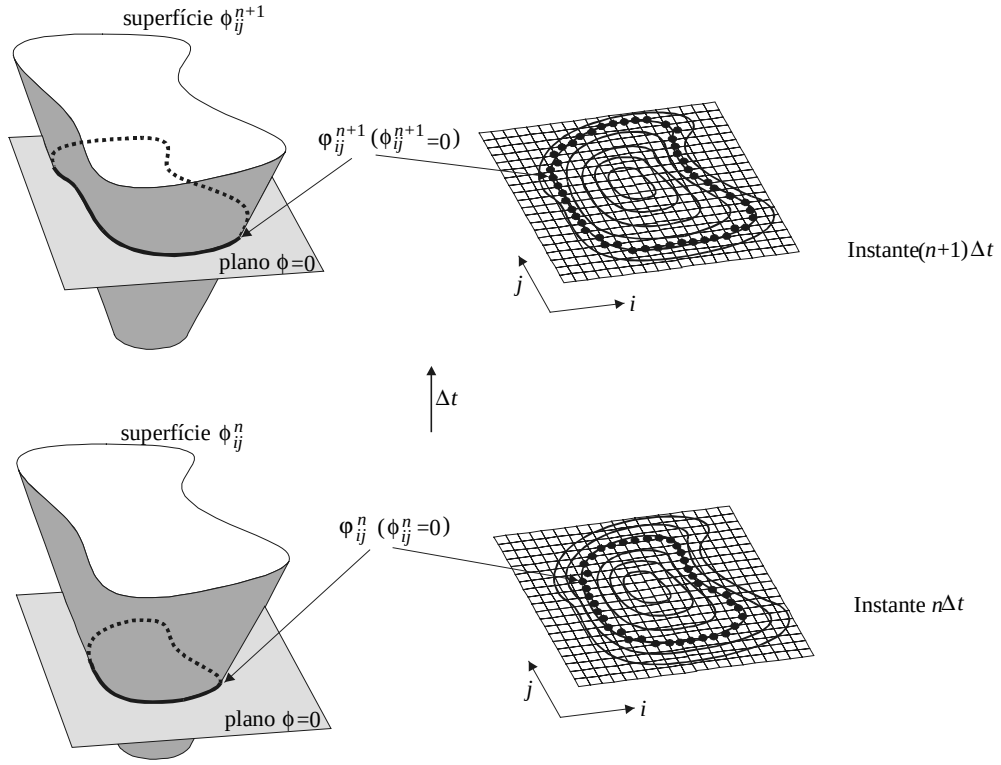


Figura 17: Método Level Set: a curva φ_{ij}^{n} evoluída é tomada sempre como $\Phi_{ij}^{n}=0$.

Como mostra a figura 17, a cada instante, os valores de todos os pontos da grade são atualizados, representando as curvas de nível da função Φ, naquele instante. Desta forma, todas as curvas de nível são atualizadas, não apenas a curva de nível zero (na figura 17, são destacados os pontos da grade correspondentes à curva de nível zero). Em muitas situações, esta atualização de todas as curvas de nível da grade é necessária. Porém há outras situações em que apenas uma curva é de interesse (curva de nível zero), sendo desnecessária a atualização de todos os pontos da grade. Nestes casos apenas os ponto da grade vizinhos à curva de interesse devem ser atualizados. Esta forma de atualização origina um método level set chamado de *narrow band*.

No método narrow band, apenas os pontos vizinhos à curva de nível zero são atualizados em cada instante. Uma vez que uma aplicação tenha interesse apenas na evolução da curva de nível zero, é desnecessário atualizar todas as curvas de nível. A narrow band é um conjunto de pontos da grade situados ao redor dos pontos correpondentes à curva de nível zero (figura 18). A largura da narrow band é definida pelo usuário.

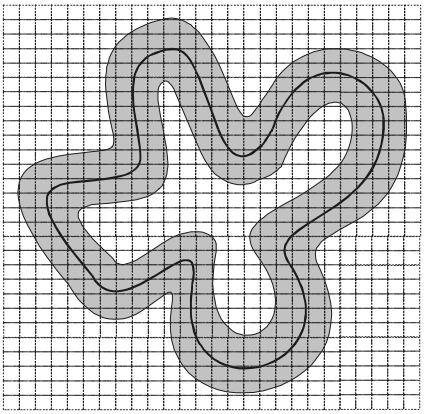
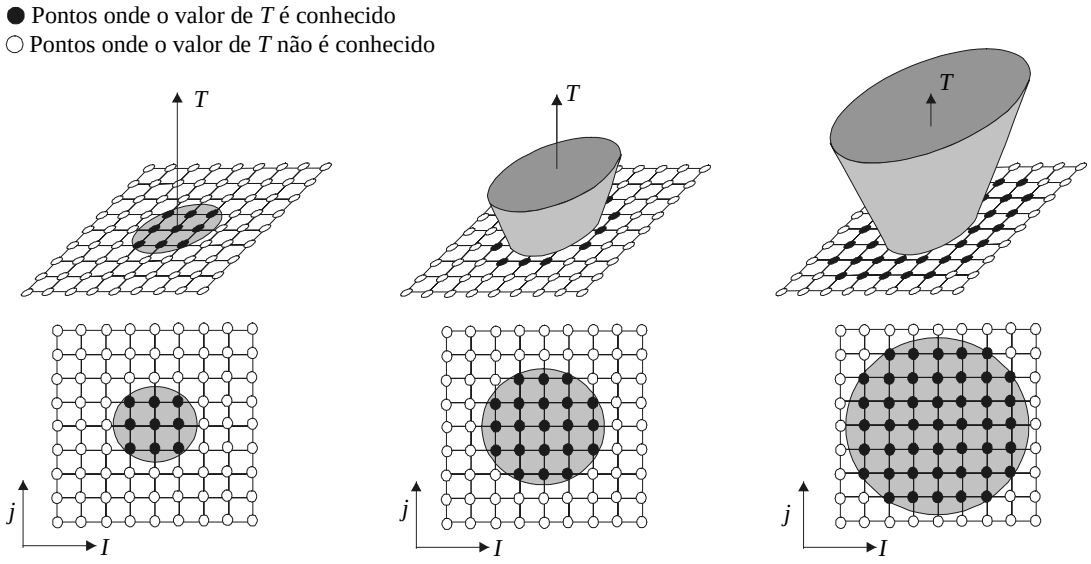


Figura 18: Os pontos da grade situados na região cinza fazem parte da narrow band.

4.3-Método Fast Marching

O Método Fast Marching propaga a interface baseando-se na formulação do valor de borda, (seção 3.2.1), onde uma função tempo *T* associa cada ponto do espaço ao instante em que a interface atinge este ponto (x,y). Esta formulação é utilizada apenas no caso onde a velocidade é sempre positiva (ou sempre negativa). Quando a interface se propaga com velocidade constante esta formulação pode ser utilizada para calcular campos distância .



a) Curva inicial (*T=0*) b) Atualização de *T* na propagação c) Atualização de *T*

Figura 19: Construção da função *T* no Método *Fast Marching*.

Ao contrário do método *Level Set*, o método *Fast Marching* é um problema estacionário, no sentido de que a funcão *T* é fixa, não se altera com a propagação da interface (ao contrário da função Φ).

A idéia do Método *Fast Marching* é que, partindo de uma interface inicial, discretizada sobre uma grade, a função *T* vai sendo construída sobre os pontos da grade, à medida que a interface se propaga. A figura 19a mostra uma curva sobre uma grade 2D, onde nos pontos (*i*,*j*) que correspondem à interface inicial, T_{ij}=0 (início da propagação). Nos demais pontos, externos à interface, o valor de *T* não é conhecido. O objetivo do método é justamente calcular estes valores de *T*, de maneira eficiente, à medida que a curva evolui (figura 19). Em cada iteração da propagação é construída uma nova camada da superfície *T*.

O Método *Fast Marching* é formulado a partir da equação Eikonal $|\nabla T|\, F = 1$, que é um caso particular da equação de Hamilton-Jacobi (apêndice). A solução numérica desta equação é baseada nas Leis da Conservação Hiperbólica. Baseada na solução discreta da equação de Hamilton-Jacobi, a equação Eikonal pode ser resolvida através do esquema:

onde $D_{ijk}^{-x} = (T_{ijk} - T_{i-1,j,k})/\Delta x$ e $D_{ijk}^{+x} = (T_{i+1,jk} - T_{ijk})/\Delta x$. D_{ijk}^{-y}, D_{ijk}^{+y}, D_{ijk}^{-z} e D_{ijk}^{+z} são definidos de maneira análoga.

$$\begin{bmatrix} \max(D_{ijk}^{-x},0)^2 + \min(D_{ijk}^{+x},0)^2 \\ + \max(D_{ijk}^{-y},0)^2 + \min(D_{ijk}^{+y},0)^2 \\ + \max(D_{ijk}^{-z},0)^2 + \min(D_{ijk}^{+z},0)^2 \end{bmatrix}^{1/2} = \frac{1}{F_{ijk}},$$

A forma padrão de se resolver estas equações requer iterações. A cada iteração *iter* os pontos da grade T_{ijk}^{iter} vão sendo calculados, a partir dos vizinhos de T_{ijk} da iteração anterior, conforme o algoritmo:

```
Para iter=1,n
        Para i,j,k=1,dim
                Resolver a equação para T_ijk^(iter+1), a partir de
                T_i-1,j,k^iter, T_i+1,j,k^iter, T_i,j-1,k^iter, T_i,j+1,k^iter, T_i,j,k-1^iter, T_i,j,k+1^iter.
        FimPara
FimPara
```

A figura 20 mostra a vizinhança do ponto T_{ijk}. em uma grade 3D.

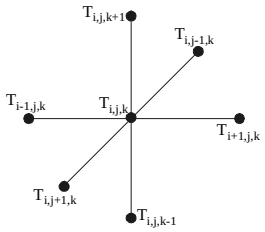


Figura 20: Vizinhança de um ponto.

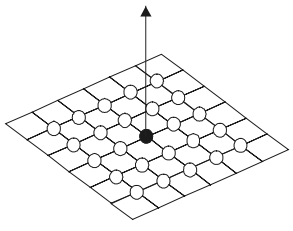
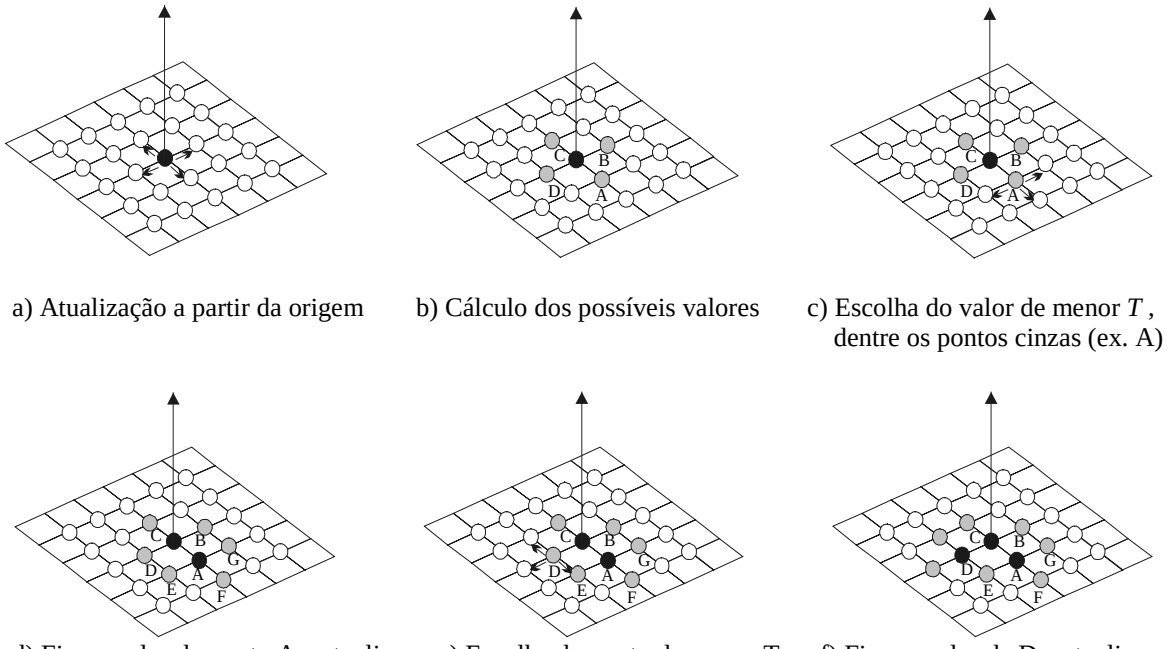


Figura 21: Início do método Fast Marching.

Durante o cálculo dos valores de *T*, a informação vai sempre se propagando a partir dos pontos com menores valores de *T*. A figura 22 explica como o processo de propagação de uma interface 2D se dá a cada iteração. Os pontos pretos representam as posições onde a

função *T* é conhecida e os pontos brancos, posições onde *T* é desconhecida. Partindo de uma interface inicial (figura 21), representada por um ponto preto, onde *T*=0 (figura 22a), são calculados os valores de seus quatro vizinhos (representados pelos pontos cinzas *A*, *B*, *C* e *D*, na figura 22b), através da equação Eikonal discreta. Dentre estes quatro pontos, a propagação deve seguir a partir daquele que tiver o menor valor de *T*. Ou seja o algoritmo para propagação requer que haja uma ordenação dos pontos cinzas. Supondo que o ponto *A* contém o menor *T*, a propagação deve prosseguir a partir dele (figura 22c), ou seja, o ponto *A* é setado para preto, indicando que a propagação já é conhecida em *A*, e seus vizinhos são calculados (representados pelos pontos cinzas *E*, *F* e *G*, na figura 22d). É importante observar que o ponto preto inicial, apesar de ser vizinho de *A*, não entrou neste processo, pois a propagação tem sentido único, não é retroativa. A propagação deve continuar a partir do ponto cinza (*B*, *C*, *D*, *E*, *F* ou *G*) que tiver o menor valor de *T*. Mais uma vez será necessária a rotina de ordenação para selecionar o ponto cinza de menor *T*. Supondo que o ponto *D* contém o menor valor, a propagação deve seguir a partir dele (figura 22e). O valor de *T* é calculado nos vizinhos de *D*, que são setados para cinza. É importante observar que um dos vizinhos de *D* (o ponto *E*) já era cinza, pois também é vizinho de *A*, mas seu valor deve ser recalculado (apenas os vizinhos pretos são poupados). O processo se repete até que a função *T* seja determinada.



a) Atualização a partir da origem

b) Cálculo dos possíveis valores

c) Escolha do valor de menor *T* , dentre os pontos cinzas (ex. A)

d) Fixa o valor do ponto A e atualiza seus vizinhos (tornam-se cinzas)

e) Escolha do ponto de menor T, dentre os pontos cinzas (ex. D)

f) Fixa o valor de D e atualiza seus vizinhos.

Figura 22:Procedimento de Atualização do Método Fast Marching.

Utilizando a equação Eikonal, o cálculo de *T* nos vizinhos nunca fornece valores menores do que os pontos já conhecidos (pontos pretos), e portanto, a “marcha” segue sempre em um único sentido, se afastando da origem.

Para a propagação nos pornots da grade, há três categorias de pontos: pontos com valores conhecidos (pretos), pontos candidatos a prosseguirem com a propagação (cinzas) e pontos desconhecidos (brancos), que seriam os pontos “distantes” da interface. Na busca da solução, a marcha se dá sempre transformando pontos brancos em pontos cinzas e cinzas em pretos, conforme mostra a figura 23.

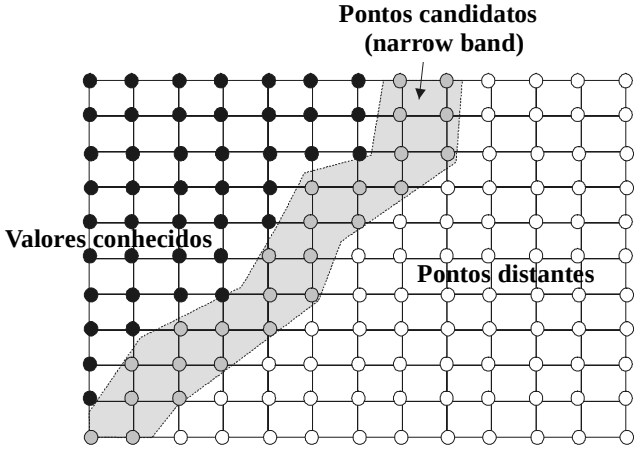


Figura 23: Progressão do Método Fast Marching

Uma das grandes vantagens deste método, além da eficiência, é que ele pode ser aplicado para cálculo de distâncias de objetos contínuos. A discretização do objeto pode ser feita em qualquer resolução, o que permite que possa ser controlada a margem de erro, nos casos que requerem reconstrução do objeto.

Em [3], Breen utiliza o método fast marching para calcular campos distâncias de objetos. Em seguida estes campos são utilizados para converter objetos representados por uma árvore CSG para representação volumétrica.

Um grande desafio é o desenvolvimento de Métodos *Fast Marching* adaptativos. No método descrito acima, os pontos onde os valores de *T* são calculados são amostrados regularmente. Isto apresenta algumas desvantagens, principalmente quando se deseja concentrar informações sobre determinadas regiões da interface em evolução. Uma opção é aumentar a resolução da grade, o que implica em desperdício de recursos, pois nem toda a interface precisaria desta alta resolução. Com grades adaptativas seria possível fazer propagação em multi resolução, o que acarretaria em inúmeras vantagens para as mais diversas aplicações.

5-Conclusões

Este trabalho apresentou uma conceituação geral sobre transformadas de distância, mostrando alguns métodos numéricos para se calcular esta transformada

O resultado de uma transformada de distância aplicada a um objeto depende da métrica utilizada. A métrica está relacionada à distância espacial dos pontos do objetos. A métrica natural de se aplicar a objetos do espaço euclidiano é a métrica euclidiana. Apesar de apresentar excelentes resultados, esta métrica apresenta vários problemas de eficiência computacional, além de não ser algoritmicamente trivial.

Várias métricas regulares são utilizadas para substituir a métrica euclidiana, numa tentaviva de aumentar a eficiência do cálculo computacional, sem perder os ótimos resultados fornecidos. Dentre estas destacam-se as métrica city block e a métrica chessboard. Estas duas são casos particulares da métrica "n_f-n_e-n_v", utilizada para gerar campos distâcnias de objetos matriciais, como imagens e volumes. No caso de volumes, valor n_f representa a distância às faces dos voxels, n_e representa a distância às arestas e n_v representa a distância aos vértices.

O cálculo do campo distância nos pontos de um objeto é feito através da propagação das distâncias em relação a um dado conjunto de pontos iniciais. Assim, o cálculo de distância pode ser visto como uma frente ou interface se propagando a partir de uma frente inicial. Baseado neste paradigma, a Teoria de Propagação de Interfaces pode ser utilizada para auxiliar no desenvolvimento de métodos numéricos eficientes, como o *Fast Marching* que pode ser aplicado a objetos formulados no universo contínuo. Isto é uma grande vantagem em relação aos métodos que utilizam a métrica "n_f-n_e-n_v", por exemplo, pois estes últimos só podem ser aplicados a objetos discretos e não seriam muito úteis nos casos onde é necessária a reconstrução do objeto.

A formulação de propagação de interfaces é feita da seguinte forma: dada uma curva paramétrica inicial, sua evolução é feita através de uma parametrização temporal. Sua propagação se dá na direção normal, e a velocidade de propagação é em função da curvatura. Quando a velocidade de propagação é constante, a propagação de interfaces pode ser utilizada para o cálculo de campos distância.

Baseado na Teoria das Leis de Conservação Hiperbólica Sethian [16] desenvolveu métodos numéricos eficientes, Método *Fast Marching* e Método *Level Set*, para propagar uma curva a partir de uma posição inicial. O primeiro método, por ser aplicado nos casos onde a velocidade de propagação é positiva, pode ser utilizado para calcular campos distância de forma bastante eficiente, a partir da equação Eikonal discreta. Além disso apresenta a vantagem de poder ser aplicado a objetos formulados em espaços contínuos.

Apêndice - Leis de Conservação Hiperbólica

Este apêndice introduz as Leis de Conservação Hiperbólica [7][8], mostrando qual sua relação com propagação de interfaces. Em [16] Sethian utilizou métodos numéricos de leis de conservação hiperbólica, para desenvolver os métodos Fast Marching e Level Set, que resolvem numericamente problemas de propagação de curvas.

Dada $u(x,t)$, onde t representa o parâmetro temporal de $u(x)=u(x,0)$, uma equação diferencial da forma

$$u_t + [G(u)]_x = 0$$

é conhecida como uma "lei de conservação hiperbólica". $G(u)$ é conhecida como a *função fluxo*. Um exemplo simples é a equação de Burger, onde $G(u) = u^2/2$, e portanto

$$u_t + uu_x = 0.$$

Esta equação de onda, que descreve o movimento de um fluido compressível em uma dimensão, mostra a relação entre a variação espacial (u_x) e a variação temporal (u_t) do fluido representado pela função u. A solução para esta equação apresenta descontinuidades, ao longo do tempo, ou seja, $u(x,t)$ pode ser descontínua, mesmo que o dado inicial $u(x,0)$ seja suave. Para tornar as soluções suaves, acrescenta-se o termo εu_{xx} (onde $\varepsilon \geq 0$) à direita da equação acima, ou seja,

$$u_t + uu_x = \varepsilon u_{xx.}$$

O termo u_{xx}, chamado de *viscosidade do fluido*, age como um termo suavizador, impedindo a formação de descontinuidades de u, ao longo do tempo. Pode-se demonstrar que se $\varepsilon>0$ a solução se mantém suave ao longo do tempo (se $\varepsilon=0$, então $u_t + uu_x = 0$, e a solução não se manterá suave). Uma vez visto (ainda que rapidamente) o que é uma lei de conservação hiperbólica, o que esta teoria tem a ver com evolução de curvas?

Relação entre Evolução de Curvas e Leis de Conservação Hiperbólica.

Seja uma curva inicial $\varphi(x,0)$, evoluindo com velocidade $F(k)=1-\varepsilon k$, cuja evolução $\varphi(x,t)$ se mantém um gráfico ao longo do tempo. Em [16] e [17] Sethian demonstra que tal evolução pode ser vista como uma lei de conservação hiperbólica (com viscosidade), dada por

$$u_t + [-(1+u^2)^{1/2}]_x = \varepsilon\left[\frac{u_x}{1+u^2}\right]_x,$$

onde $u=d\varphi/dx$. Nesta equação, a função fluxo é $G(u) = -(1+u^2)^{1/2}$.

É interessante analisar esta relação sob os dois pontos de vista: de evolução de curvas e de lei de conservação hiperbólica. Se $\varepsilon=0$, então F é constante ($F(k)=1$) e, conforme visto nas seções 3.3, a evolução de φ não é suave; já a equação hiperbólica não tem termo de

viscosidade e a variação de *u*, ao longo do tempo, também não é suave. Se $\varepsilon>0$, então *F* depende da curvatura ($F(k)=1-\varepsilon k$) e a evolução de α é suave; no caso da equação hiperbólica não há termo de viscosidade e a variação de *u*, ao longo do tempo, é suave. Conclusão: *O papel da curvatura em evolução de curvas é análogo ao papel da viscosidade em leis de conservação hiperbólica.*

Em [16], capítulo 5, Sethian descreve alguns métodos numéricos para resolver algumas equações de leis de conservação hiperbólica

Equações de Hamilton-Jacobi

Na seção 3.1 foram vistas duas formulações para o problema de propagação de curvas: formulação do valor de borda, definida a través da equação $|\nabla T| F = 1$, e a formulação do valor inicial, definida através da equação $\Phi_t + F|\nabla\Phi| = 0$. Estas duas equações são casos particulares da equação genérica, chamada equação de *Hamilton-Jacobi*, definida como

$$\alpha U_t + H(U_x,U_y,U_z) = 0,$$

(considerando o problema em 3 dimensões). A função *H* é chamada de *Hamiltoniana*. Na formulação do valor inicial, $\alpha=1$, $U=\Phi$ e a Hamiltoniana vale

$$H(U_x,U_y,U_z) = F\sqrt{U_x^2+U_y^2+U_z^2}.$$

Na formulação do valor de borda $\alpha=0$, $U=T$ e a Hamiltoniana vale

$$H(U_x,U_y,U_z) = F\sqrt{U_x^2+U_y^2+U_z^2}-1.$$

Seja a equação de Hamilton-Jacobi unidimensional $\alpha U_t+H(U_x)=0$. Fazendo $U_x=u$ e diferenciando ambos os lados, é obtida a equação de lei de conservação hiperbólica $u_t+[H(u)]_x=0$ equivalente. Utilizando métodos numéricos de lei de conservação hiperbólica, Sethian [16] deduziu a seguinte expressão para calcular numericamente esta equação 1D de Hamilton-Jacobi:

$$\alpha\left(\frac{U_i^{n+1}-U_i^n}{\Delta t}\right) = -g\left(\frac{U_i^n-U_{i-1}^n}{\Delta x},\frac{U_{i+1}^n-U_i^n}{\Delta x}\right), \quad (1)$$

onde U_i^n é o valor da *i*-ésima amostra da função *U* no instante *n*, Δt é o intervalo de tempo e Δx é o intervalo entre as amostra ao longo do eixo *x*, conforme mostra a figura 16. A figura mostra uma grade cujos pontos representam uma discretização do tempo e espaço, onde a evolução da função *U* está sendo calculada. Assim a cada ponto (*i*,*n*) da grade há um valor U_i^n associado. Em cada instante *n+1* as amostras U_i^{n+1} são calculadas, representando assim a nova curva gerada pela evolução a partir da amostras U_i^n.

Para melhor compreender a expressão numérica (1), é importante analisá-la cuidadosamente. Na verdade, trata-se da versão discreta da equação de Hamilton-Jacobi unidimensional $\alpha U_t+H(U_x)=0$, reescrita como $\alpha U_t = -H(U_x)$. A versão discreta da

derivada U_t é dada por $(U_i^{n+1} - U_i^n)/\Delta t$ (forward difference) e a Hamiltoniana está sendo calculada como $g[(U_i^n - U_{i-1}^n)/\Delta x , (U_{i+1}^n - U_i^n)/\Delta x]$. A função $g(u_1,u_2)$, chamada de *função fluxo*, é calculada a partir dos métodos numéricos de leis de conservação hiperbólica (em [16] Sethian mostra como calculá-la) e depende da função velocidade F usada para evoluir a interface. Na expressão (1) a função fluxo recebe como parâmetros as derivadas (backward difference e forward difference, respectivamente) discretas em relação a x.

A equação (1) pode ser reescrita como

$$\alpha U_i^{n+1} = \alpha U_i^n - \Delta t.g\left(\frac{U_i^n - U_{i-1}^n}{\Delta x}, \frac{U_{i+1}^n - U_i^n}{\Delta x}\right),$$

Esta última equação mostra como a curva evolui no instante $n+1$, a partir das informações da curva no instante n. O cálculo de equações de Hamilton-Jacobi em dimensões maiores, onde a Hamiltoniana é simétrica, pode ser feito através da equação

$$U_{ijk}^{n+1} = U_{ijk}^n - \Delta t.g,$$

onde g é calculada com os seguintes parâmetros:

$$g\left(\frac{U_{ijk}^n - U_{i-1,j,k}^n}{\Delta x}, \frac{U_{i+1,j,k}^n - U_{ijk}^n}{\Delta x}, \frac{U_{ijk}^n - U_{i,j-1,k}^n}{\Delta y}, \frac{U_{i,j+1,k}^n - U_{ijk}^n}{\Delta y}, \frac{U_{ijk}^n - U_{i,j,k-1}^n}{\Delta z}, \frac{U_{i,j,k+1}^n - U_{ijk}^n}{\Delta z}\right)$$

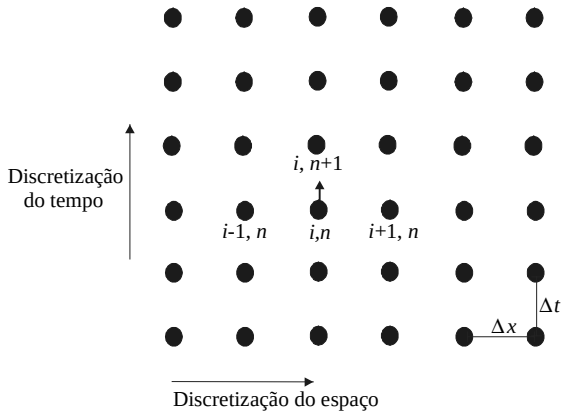


Figura 24: Cálculo numérico da evolução.

Bibliografia

[1] Berger, M. & Colella, P. 1989. Local Adaptive Mesh Refinement for Shock Hydrodynamics. *J. Comp. Phys*, **1**, 82, 62-84.

[2] Borgefors, G. 1986. Distance Transformations on Digital Images. *Computer Vision and Image Processing*, **34**, 344-371.

[3] Breen, D. E.; Mauch, S. & Whitaker, R. T. 1998. 3D Scan Convertion of CSG Models into Distance Volumes. *Proceedings of Symposium on Volume Visualization, ACM SIGGRAPH*, 7-14.

[4] Cohen-Or, D.; Levin, D. & Solomivici, A. 1998. Three-Dimensional Distance Field Metamorphosis. *ACM Transactions on Graphics.* **17**, 2, 116-141.

[5] Dorst, L. 1986. Pseudo-Euclidian Skeletons. *The 8th International Conference on Pattern Recognition, Paris France, Washington, D.C. IEEE Computer Society Press, Los Angeles, CA.* 286-289.

[6] Gomes, J. M.; Costa, B.; Darsa, L. & Velho, L. 1996. Graphical Objects. *The Visual Computer*, **12**, 269-282.

[7] Lax, P. D. 1970. Hyperbolic Systems for Conservation Laws and the Mathematical Theory of Shock Waves. *SIAM Reg. Conf. Series, Lectures in Applied Math*, **11**, 1-47.

[8] Le Veque, R. J. 1992. Numerical Methods for Conservation Laws. Birkhauser, Basel.

[9] Milne, B. 1995. Adaptive Level Set Methods Interface. PhD. Thesis. Dept. of Mathematics, University of California, Berkeley, CA.

[10] Niblack, C. W.; Gibbons, P. B. & Capson, D. W. 1992. Generating Skeletons and Centerlines from the Distance Transform. *Graphical Models and Image Processing.* **54**, 5, 420-437.

[11] Paglieroni, D. W. 1992. Distance Transforms: Properties and Machine Vision Applications. *CVGIP: Graphical Models and Image Processing*, **54**, 1, 56-74.

[12] Peixoto, A. & Gattass, M. 2000. Reconstrução de Superfícies a partir de Seções Bidimensionais. Qualificação MCC28/00, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

[13] Peixoto, A. & Carvalho, P. C. P. 2000. Esqueletos de Objetos Volumétricos. Qualificação MCC34/00, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

[14] Sethian, J. A. 1985. Curvature and the Evolution of Fronts. *Comm. in Math. Phys*, **101**, 4, 487-499.

[15] Sethian, J. A. 1996. A Fast Marching Level Set Methods for Monotonically Advancing Fronts. *Proceedings of the National Academy of Sciences*, **93**, 4, 1591-1595.

[16] Sethian, J. A. 1999. Level Set Methods and Fast Marching Methods. Cambridge University Press, Cambridge UK.

[17] Sethian, J. A. 1987. Numerical Methods for Propagating Fronts. *Variational Methods for Free Surface Interfaces*. Eds. P. Concus and R. Finn, Springer-Verlag, NY.

[18] Zhou, Y.; Kaufman, A. & Toga, A. W. 1998. Three-Dimensional Skeleton and Centerline Generation Based on an Approximate Minimum Distance Field. *Visual Computer*, **14**, 303-314.

[19] Zhou, Y. & Toga, A. W. 1999. Efficient Skeletonization of Volumetric Objects. *IEEE Transactions on Visualization and Computer Graphics* , **5**, 3, 196-209.